

Yade 文档(3rd 版)

第 *3* 版 基于 *Yade* 版本 *2021-11-19.git-639e121*，*2022*年*10*月*23*日

作者

- **Václav Šmilauer** 自由顾问 (http://woodem.eu)
- **Vasileios Angelidakis** 纽卡斯尔大学，英国
- **Emanuele Catalano** 格勒诺布尔-阿尔卑斯大学，3SR实验室。
- **Robert Caulk** 格勒诺布尔-阿尔卑斯大学，3SR实验室。
- **Bruno Chareyre** 格勒诺布尔-阿尔卑斯大学，3SR实验室。
- **William Chèvremont** 格勒诺布尔-阿尔卑斯大学，LRP
- **Sergei Dorofeenko** 俄罗斯科学院化学物理问题研究所，切尔诺戈洛夫卡
- **Jérôme Duriez** 法国国家农业、食品与环境研究院，艾克斯-马赛大学，RECOVER，普罗旺斯地区艾克斯，法国
- **Nolan Dyck** 西安大略大学
- **Jan Eliáš** 布尔诺理工大学
- **Burak Er** 布尔萨技术大学
- **Alexander Eulitz** 柏林工业大学 / 机床与工厂管理研究所
- **Anton Gladky** 弗赖贝格工业大学
- **Ning Guo** 香港科技大学
- **Christian Jakob** 弗赖贝格工业大学
- **François Kneib** 格勒诺布尔-阿尔卑斯大学，3SR实验室 / 格勒诺布尔国立环境与农业科技研究院
- **Janek Kozicki** 格但斯克工业大学
- **Donia Marzougui** 格勒诺布尔-阿尔卑斯大学，3SR实验室。
- **Raphaël Maurin** 格勒诺布尔国立环境与农业科技研究院
- **Chiara Modenese** 牛津大学
- **Gerald Pekmezi** 阿拉巴马大学伯明翰分校
- **Luc Scholtès** 格勒诺布尔-阿尔卑斯大学，3SR实验室。
- **Luc Sibille** 南特大学，GeM实验室
- **Jan Stránský** 布拉格捷克理工大学
- **Thomas Sweijen** 乌得勒支大学
- **Klaus Thoeni** 纽卡斯尔大学（澳大利亚）
- **Chao Yuan** 格勒诺布尔-阿尔卑斯大学，3SR实验室。

引用本文档

在科学出版物中提及 Yade-DEM软件时，请通过“数字对象标识符”引用，格式如下：Šmilauer V. 等（2021年）Yade 文档 第3版。Yade 项目。数字对象标识符：10.5281/zenodo.5705394。http://yade-dem.org

另请参阅 http://yade-dem.org/doc/citing.html。

目录

1 导览 1

1.1 简介 1
1.1.1 入门 .1 1.1.2 架构概述 .6

1.2 T教程 15
1.2.1 简介.15 1.2.2 实践操作 .15 1.2.3 数据挖掘 .25 1.2.4 设置模拟.30 1.2.5 进阶及更多内容 .34 1.2.6 教程示例 .35 1.2.7 更多示例 .45

2 Yade 面向用户 5 **1**

2.1 离散元法公式 .51
2.1.1 碰撞检测 .51 2.1.2 创建颗粒间相互作用 .55 2.1.3 运动学变量 .57 2.1.4 接触模型（示例） .60 2.1.5 运动积分 .60 2.1.6 周期性边界条件 .68 2.1.7 计算方面 .72

2.2 User's m手册 73
2.2.1 场景构建 .73 2.2.2 控制模拟 .93 2.2.3 后处理 .109 2.2.4 Python 特性与技巧 .114 2.2.5 扩展 Yade .114 2.2.6 故障排除.115

2.3 Yade 包装类参考 .117
2.3.1 刚体 .117 2.3.2 相互作用 .149 2.3.3 全局引擎.188 2.3.4 部分引擎 .266 2.3.5 分派器 .290 2.3.6 函子 .294 2.3.7 包围体创建 .295 2.3.8 相互作用几何创建 .300 2.3.9 相互作用物理创建 .316 2.3.10 本构定律 .330 2.3.11 内力 .351 2.3.12 回调 .351 2.3.13 预处理器 .352 2.3.14 渲染 .357 2.3.15 仿真数据 .369 2.3.16 其他类 .379

2.4 Yade 模块参考 .386 2.4.1 yade.bodiesHandling 模块 .386 2.4.2 yade.export 模块 .387 2.4.3 yade.geom 模块 .392 2.4.4 yade.gridpfacet 模块 .395 2.4.5 yade.libVersions 模块 .399 2.4.6 yade.linterpolation 模块 .402 2.4.7 yade.log 模块 .403 2.4.8 yade.math 模块 .404 2.4.9 yade.minieigenHP 模块 .427 2.4.10 yade.mpy 模块 .474 2.4.11 yade.pack 模块 .479 2.4.12 yade.plot 模块 .490 2.4.13 yade.polyhedra_utils 模块 .494 2.4.14 yade.post2d 模块 .494 2.4.15 yade.qt 模块 .497 2.4.16 yade.timing 模块 .499 2.4.17 yade.utils 模块 .500 2.4.18 yade.ymport 模块 .516

2.5 安装 .520
2.5.1 软件包 .521 2.5.2 Docker .522 2.5.3 源代码 .523 2.5.4 加速编译 .529 2.5.5 云计算 .530 2.5.6 GPU 加速 .530 2.5.7 特殊构建版本 .530 2.5.8 Yubuntu .531 2.6 致谢 Yade .531

3 Yade 程序员 533
3.1 程序员 .533 3.1.1 构建系统 .533 3.1.2 开发工具 .534 3.1.3 调试 .535 3.1.4 回归测试 .541 3.1.5 约定 .543 3.1.6 支持框架 .548 3.1.7 仿真框架 .572 3.1.8 运行时结构 .578 3.1.9 Python 框架 .579 3.1.10 添加新的 Python/C++ 模块 .581 3.1.11 保持兼容性 .583 3.2 GitLab 上的 Yade .584 3.2.1 快速检出（只读） .584 3.2.2 GitLab 上的分支 .584 3.2.3 合并请求 .587 3.2.4 推送指南 .588

4 理论背景与扩展 **589**
4.1 离散元法公式 .589 4.2 使用 Yade 和 OpenFOAM 进行 CFD-DEM 耦合模拟 .589 4.2.1 背景 .589 4.2.2 设置算例 .592 4.2.3 后处理 .594 4.3 使用 Yade 和 Escript 进行有限元-离散元分层多尺度建模 .4.3.1 简介 .594 4.3.2 有限元公式 .595

4.3.3 多尺度求解过程.595 4.3.4 YADE 端的工作.596 4.3.5 Escript 端的工作.596 4.3.6 示例测试.597 4.3.7 免责声明.597 4.4 在 Yade 中模拟声发射.598 4.4.1 摘要.598 4.4.2 模型描述.598 4.4.3 在 Yade 中启用该算法.599 4.4.4 声发射的可视化与后处理.600 4.4.5 考虑岩石非均质性.601 4.5 使用 YADE 一维垂向 VANS 流体求解.601 4.5.1 YADE 中的离散元-流体耦合与流体求解.603 4.5.2 阻力与浮力的应用（HydroForceEngine::action）.603 4.5.3 固相平均（HydroForceEngine::averageProfile）.604 4.5.4 流体求解 \HydroForceEngine::fluidResolution.605 4.6 势粒子与势块.606 4.6.1 简介.606 4.6.2 势粒子代码（PP）.606 4.6.3 势块代码（PB）.607 4.6.4 引擎.609 4.6.5 接触定律.610 4.6.6 PP 和 PB 的形状定义.611 4.6.7 PP 和 PB 的体定义.612 4.6.8 边界粒子.613 4.6.9 可视化.614 4.6.10 轴对齐包围盒.615 4.6.11 块生成算法.616 4.6.12 示例.616 4.6.13 免责声明.616 4.6.14 引用.616

5 性能增强 617

5.1 使用图形处理器加速 Yade 的 FlowEngine .61

7 5.1.1 摘要.617 5.1.2 硬件、软件和模型要求.617 5.1.3 安装 CUDA.618 5.1.4 安装 OpenBLAS 和 LAPACK.618 5.1.5 安装 SuiteSparse.618 5.1.6 编译 Yade.618 5.1.7 控制图形处理器.619 5.1.8 性能提升.619 5.2 MPI 并行化.620 5.2.1 概念.620 5.2.2 操作说明.621 5.2.3 MPI 初始化与通信.623 5.2.4 划分.629 5.2.5 合并.632 5.2.6 预期提示与问题.632 5.2.7 控制变量.633 5.2.8 基准测试.633 5.3 在亚马逊 EC2 上通过云计算使用 YADE.634 5.3.1 摘要.634 5.3.2 启动 EC2 实例.634 5.3.3 安装 YADE 和管理文件.636 5.3.4 在终端中绘制输出.637 5.3.5 注释.638 5.4 高精度计算.638

5.4.1 安装 .6395.4.2 支持的模块 .6395.4.3 双精度、四精度及更高精度 .6405.4.4 兼容性 .6415.4.5 调试.645

6 文献 第 6 47

6.1 Yade 技术档案 .6476.1.1 关于 .6476.1.2 贡献.6476.1.3 联系方式 .6476.1.4 档案 .6476.2 Yade相关出版物 .6486.2.1 引用 Yade .6486.2.2 期刊文章 .6486.2.3 会议资料和书籍章节 .6486.2.4 硕士和博士学位论文.6486.2.5 Yade 技术档案.6486.3 引用 .648

7 索引和表格 649

参考文献 651

Python 模块索引 679

第1章

导览

1.1 简介

1.1.1 入门

开始在 Yade 中操作之前，你应该具备一些相关知识。

- 在 Linux 系统中，运行 yade 需要掌握命令行基础知识。请在网上查找相关教程。

- Python 语言；我们推荐官方的 Python 教程。进一步阅读该主题的其他资料，例如 DiveintoPython，当然也不会有坏处。

建议您亲自尝试所有介绍的命令。不要害怕进行实验。

提示： 有时，以 .pdf 格式阅读本文档可能会更舒适。例如，在 Okular PDF 查看器中，点击链接比在网页浏览器中刷新页面更快；要返回，请按快捷键 `Alt Shift ←`。您可以查看 部分引擎 的继承图来试试，然后返回。

启动 Yade

Yade 主要通过终端运行；其命令名称是 `yade`。[1]（如果您不是通过软件包安装的，可能需要为命令指定具体路径[2]）：

```
$ yadeWelcome to YadeTCP python prompt on localhost:9001,
 auth cookie `sdksuy'TCP info provider on localhost:21000
```

（续见下页）

[1]可执行文件名可能带有后缀，例如版本号（`yade-0.20`），具体取决于编译选项。Debian 系统上的打包版本默认总会提供不带后缀的 `yade` 别名，通常指向最新稳定版（如果没有安装稳定版，则指向最新快照）。您可以使用 `update-alternatives` 进行更改。[2] 通常，Unix *Shell*（命令行）定义了环境变量 `PATH`，用于确定在未提供路径、仅输入文件名时搜索可执行文件的目录。通常，$PATH 包含 `/usr/bin/`、`/usr/local/bin`、`/bin`等；您可以在 Shell 中输入 `echo $PATH` 来查看 `PATH`（目录之间以 :分隔）。如果 Yade 可执行文件不在 `PATH`所包含的目录中，您必须手动指定其路径，也就是在文件名前输入路径，例如 `/home/user/bin/yade` 及类似路径。您也可以切换到该目录本身（`cd ~/bin/yade`，其中 `~` 会自动替换为您的主目录），然后输入 `./yade`（其中的 . 表示当前目录，因此 `./`表示要在当前目录中查找该文件）。为减少输入，您可以将 Yade 的安装目录添加到 `PATH`中，通常通过编辑 `~/.profile` 文件（该文件通常会在 Shell 启动时自动执行），添加类似于 `export PATH=/home/user/bin:$PATH`的行来实现。您也可以在该文件中通过 `alias yade="/home/users/bin/yade"` 定义一个别名。具体细节取决于您使用的 Shell（bash、zsh、tcsh 等），您可以在 Linux/Unix 入门资料中找到更多信息。

（接上页）

```
[[ ^L clears screen, ^U kills line. F12 controller, F11 3d view, F10 both, F9 generator, F8␣
→plot. ]]
Yade [1]:
```

以下开头几行将为您提供一些关于

- 一些关于远程控制的信息，您现在可能暂时用不到；
- 刚刚出现的命令行的基本帮助（`Yade [1]:`）。

输入 `quit()`、 `exit()`，或者直接按下 `^D` （`^` 是按下 `Ctrl` 键的常用书面简写，因此这里的 `^D` 表示 `Ctrl D`）即可退出 Yade。

命令行是 IPython，一种具备增强交互功能的 Python shell；它具有持久历史记录（会记住您上次会话中的命令）、搜索等功能。有关更多详细信息，请参阅 IPython 的文档。

通常，你不会手动输入 Yade 命令，而是使用 脚本（用于描述和运行模拟的 Python 程序）。我们来看一个最简单的脚本，它只会打印“Hello world!”：

```
print("Hello world!")
```

将这样的脚本保存为 `hello.py` 后，就可以将其作为参数传给 Yade：

```
$ yade hello.pyWelcome to YadeTCP python prompt on localhost:9001, auth cookie `askcsu'
TCP info provider on localhost:21000
Running script hello.py                                          ## the script is being run
Hello world!                                                     ## output from the script
[[ ^L clears screen, ^U kills line. F12 controller, F11 3d view, F10 both, F9 generator, F8␣
→plot. ]]Yade [1]:
```

Yade 会运行脚本，然后返回命令行。[3] 如果希望 Yade 在运行脚本后立即退出，请使用 `-x`开关：

```
$ yade -x script.py
```

命令行选项不止 `-x` 这一项，运行 `yade -h` 即可查看全部选项。

选项：

-v, --version 显示程序的版本号并退出

-h, --help 显示此帮助消息并退出

-j THREADS, --threads=THREADS OpenMP 线程数量用于运行；默认为 1。 等同于设置 OMP_NUM_THREADS 环境变量。

--cores=CORES 设置 OpenMP 线程数量（与 –threads 相同），并且另外，将线程的亲和性设置为指定核心。

--update 使用以下方式更新给定脚本中的已弃用的类名文本搜索与替换。已更改的文件将进行备份并添加 ~ 后缀。完成后退出，不运行任何模拟。

--nice=NICE 提高 nice级别（即 按给定数值降低优先级。

[3] 标准Python解释器在运行完脚本后就会退出。Yade 之所以采取相反的做法，是因为脚本大多只负责设置模拟并启动运行；由于计算通常在后台线程中运行，脚本在技术上已经执行完毕，但计算仍在运行。

-x	脚本运行结束时退出
-f	设置日志详细级别，默认为 -f3 (yade.log.WARN) 适用于全部类
-n	在不使用图形界面的情况下运行（等同于取消设置-设置 DISPLAY 环境变量)
--test	运行回归测试套件后退出；如果所有测试通过，退出状态为 0 如果全部测试通过则为 1，测试失败则为 2，未指定异常。
--check	运行一系列用户自定义检查测试，如下文所述。位于scripts/checks-and-tests/checks/README和回归测试
--性能	启动性能测试以衡量性能。
--stdperformance	启动标准化测试以衡量性能性，它会不断重试运行基准测试直到性能的标准差低于1%。一种常见的模拟类型是：球体落入盒子中，并给予足够的时间使其稳定下来在那里。 注意：最好将其与参数 *-j* 一起使用线程 （如上所述）。
--quickperformance	启动快速测试以测量性能。与上文相同，但只进行两次短暂运行，不尝试以小误差查找计算机的性能。
--no-gdb	Yade 崩溃时不显示回溯（仅在使用 –debug 时有效)[4]。

快速内联帮助

所有可从 IPython Shell 调用的函数，都可以通过在函数名后附加 `?`，或对它们调用 `help(…)`command，快速查看帮助：

```
Yade [1]: O.run?Docstring:run( (Omega)arg1 [, (int)nSteps=-1 [, (bool)wait=False]]) -> None :
Run the simulation. *nSteps* how many steps to run, then stop (if positive); *wait* will␣
→cause not returning to python until simulation will have stopped.Type:      method
Yade [2]: help(O.pause)Help on method pause:pause(...) method of yade.wrapper.Omega instance
pause( (Omega)arg1) -> None :
Stop simulation execution. (May be called from within the loop, and it will stop after␣
→the current step).
```

使用制表符补全机制是快速了解可用函数的方法，例如输入 `O.`，然后按 Tab 键。

创建模拟

要创建模拟，可以使用专门的文件生成器类来创建完整场景，并可接收一些参数。生成器使用 C 编写++，其职责仅限于良好地

[4] 在某 Linux 系统堆栈跟踪将产生 `Operation not permitted` 错误。请参阅调试章节以获取求解方法 .

定义的场景。例如 , 创建三轴试验场景：

```
Yade [3]: TriaxialTest(numberOfGrains=200).load()

Yade [4]: len(O.bodies)
Out[4]: 206
```

生成器是支持属性访问的常规 Yade 对象。

也可以通过 Python 脚本构建场景；这为开发提供了更大的灵活性和更高的速度，也是创建模拟的推荐方式。Yade 提供了用于简化刚体构建、从文件导入几何体以及复用常用代码的模块。由于这一主题涉及的内容较多，因此在用户手册中进行了说明。

运行模拟

如下文所述，循环由按顺序运行一系列引擎构成。可以通过以下方式查询步数： `O.iter` ，而通过以下方式前进一步： `O.step()`。每一步都会使虚拟时间按当前时间步长向前推进， `O.dt` 该时间步长可以直接赋值，也可以（通常更好的做法是）由*GlobalStiffnessTimeStepper*自动确定（如果存在）：

```
Yade [5]: O.iter
Out[5]: 0

Yade [6]: O.time
Out[6]: 0.0

Yade [7]: O.dt=1e-4

Yade [8]: O.dynDt=False #else it would be adjusted automaticaly during first iteration

Yade [9]: O.step()

Yade [10]: O.iter
Out[10]: 1

Yade [11]: O.time
Out[11]: 0.0001
```

不过，模拟通常会连续运行。可通过 `O.run()` 和`O.pause()`启动/停止循环；注意， `O.run()` 会将控制权返回给 Python，而模拟会在后台运行；如果要等待模拟完成，请使用 `O.wait()`。可以通过 `O.run(1000)`运行固定步数；`O.run(1000,True)` 则会运行并等待。若要在达到绝对步数时停止，可以设置 `O.stopAtIter` ，然后照常调用`O.run()`。

```
Yade [12]: O.run()

Yade [13]: O.pause()

Yade [14]: O.iter
Out[14]: 1715

Yade [15]: O.run(100000,True)

Yade [16]: O.iter
Out[16]: 101715

Yade [17]: O.stopAtIter=500000

Yade [18]: O.run()
```

（下页续）

（接上一页）

```
Yade [19]: O.wait()

Yade [20]: O.iter
Out[20]: 500000
```

保存与加载

模拟可以在任何时刻保存到二进制文件中（如果文件名带有“.gz”或“.bz2”等扩展名，还可以选择压缩）。也可以保存为 XML 文件，不过生成的文件会更大，保存/加载速度也更慢；当文件名包含“xml”时会使用此格式。虽然存在一些限制，但通常可以在之后加载场景并继续模拟，就像模拟从未中断过一样。请注意，由于保存的场景是 Yade 内部对象的转储，因此可能无法（很可能无法）用不同版本的 Yade 打开。有时可以通过使用“.xml”格式迁移保存的文件来解决此问题。

```
Yade [21]: O.save('/tmp/a.yade.bz2')

Yade [22]: O.reload()

Yade [23]: O.load('/tmp/another.yade.bz2')
```

将模拟保存为 XML 的主要用途，是将其用作模拟检查点的临时内存存储，例如重新加载初始状态并使用不同参数再次运行（例如拉伸/压缩试验，每次试验都从相同的初始状态开始）。函数 `O.saveTmp()` 和 `O.loadTmp()` 可以选择指定一个槽名称，以便在内存中通过该名称找到它们：

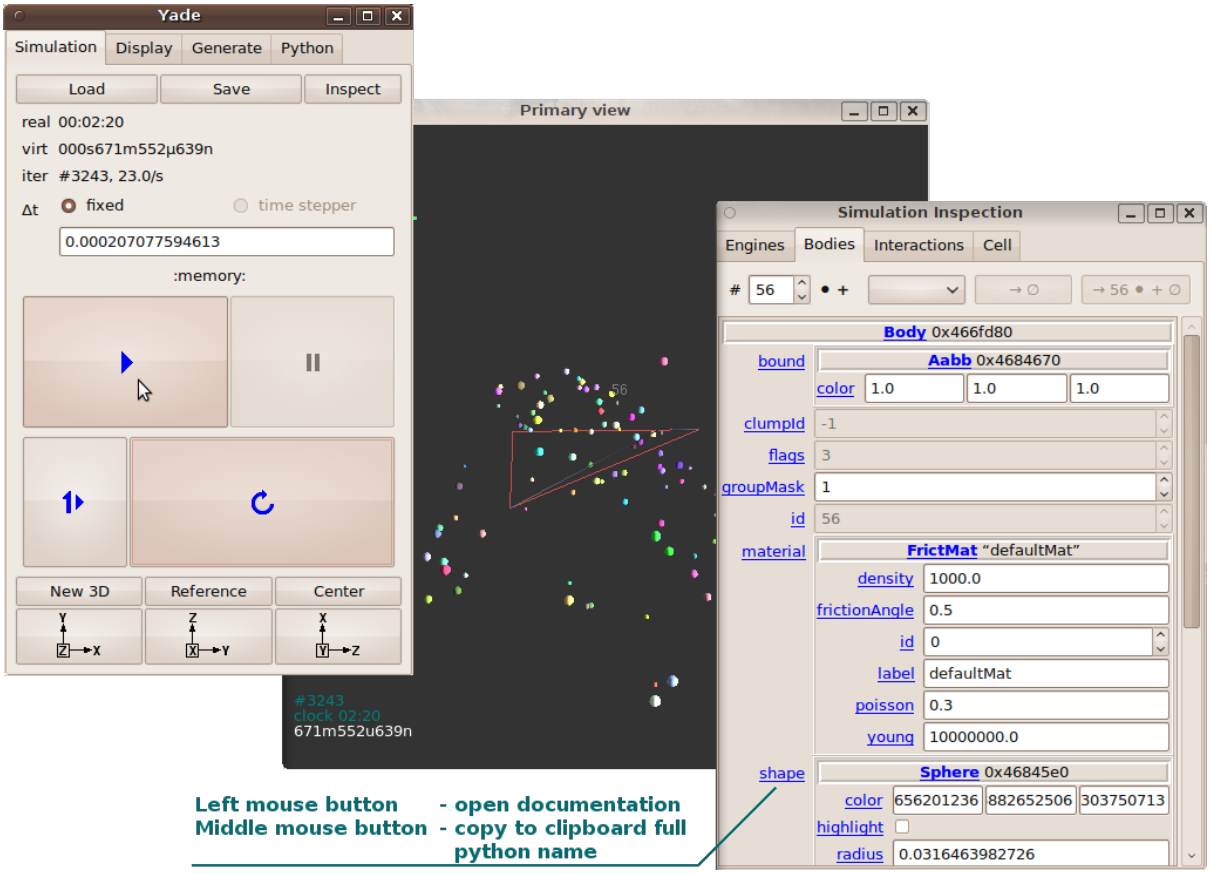
```
Yade [24]: O.saveTmp()

Yade [25]: O.loadTmp()

Yade [26]: O.saveTmp('init') ## named memory slot

Yade [27]: O.loadTmp('init')
```

可以通过 `O.reset()` 将模拟重置为空状态。

有时，在原始模拟暂时挂起时运行另一项模拟会很有用，例如动态创建堆积体时。 `O.switchWorld()` 可在主模拟和次级模拟之间切换。

图形界面

Yade 可以选择编译时加入 QT 图形界面（已支持 Qt 4 和 Qt 5）。可在命令行中按下 `F12` 启动；运行脚本时也会自动启动。



左侧的控制窗口（图 *imgQtGui*）称为 `Controller` （可通过 `yade.qt.Controller()` 从 Python 调用，或在终端中按下 `F12` 键调用）：

1. 模拟选项卡的功能大多不言自明，可用于基本的模拟控制。2. 显示选项卡提供了各种渲染相关选项，这些选项适用于所有已打开的视图（视图数量可以为零个或多个；点击新建三维按钮即可打开新视图）。3. *Python* 选项卡只有一个简单的文本输入区；例如，当脚本运行导致命令行受阻时，可以在此输入 Python 命令。

在检查窗口中（位于图 *imgQtGui*右侧），可以实时查看和修改所有模拟数据。

1. 点击任意蓝色超链接的鼠标左键即可打开文档。
2. 点击鼠标中键会将完整限定的 Python 名称复制到剪贴板，然后可在终端中点击鼠标中键（或按下 `Ctrl-V`）将其粘贴。

可以使用鼠标和键盘快捷键控制三维视图；在三维视图中按下 `h` 键时会显示帮助。请注意，打开三维视图会显著减慢模拟运行速度；它仅用于快速检查模拟是否运行平稳。高级后处理内容详见专门的数据挖掘章节。

1.1.2 架构概述

下面将概述 Yade 架构。由于许多功能都直接体现在以 Python 编写的模拟脚本中，熟悉这门语言将有助于理解示例。至于其他内容，这些知识并非必不可少，你可以忽略代码示例。

数据与函数

为确保软件设计的灵活性，yade 明确区分了两类：数据组件和功能组件。前者只存储数据，不提供功能；后者则定义对数据进行操作的函数。在编程中，这称为访问者模式（功能组件会“访问”数据，但不会与数据显式绑定）。

整个模拟，即数据和函数，都存储在一个 Scene 对象中。可通过 Python 中的*Omega* 类访问（它是一个单例），默认存储在 O 全局变量中：

```
Yade [28]: O.bodies       # some data components
Out[28]: <yade.wrapper.BodyContainer at 0x7f0cfbeadeb0>

Yade [29]: len(O.bodies)  # there are no bodies as of yet
Out[29]: 0

Yade [30]: O.engines      # functional components, empty at the moment
Out[30]: []
```

数据组件

刚体

YADE 仿真（类 场景，但在 *Omega* 中隐藏地表示为 Python 对象）由刚体、它们的相互作用以及由此产生的广义力（均存储在特殊容器中）构成。

每个刚体包含以下内容：

形状表示颗粒的几何形状（与其空间方向无关），例如球体、面片或无限墙；通常在模拟过程中不会改变。

材料存储与力学行为相关的特性，例如杨氏模量或密度。这些特性与颗粒的形状和尺寸无关；通常保持恒定，也可能由多个刚体共享。

状态包含状态变量，尤其是空间位置和方向、线和角速度；它由积分器在每一步进行更新。派生类还会包含与该刚体当前状态相关的其他信息，例如温度、平均损伤或断裂连接。

Bound 用于近似（“第1遍”）接触检测；会根据刚体的运动按需更新。目前，*Aabb*最常用作*Bound*。有些刚体可能没有*Bound*，这种情况下它们不参与接触检测。

（除了这 4 个组件外，刚体还关联着一些其他次要数据，例如 *Body::id* 或*Body::mask*。）

这四个属性都可以是不同的类型，并且派生自各自对应的基类型。Yade 会经常根据这些类型决定如何进行计算： *Sphere* + *Sphere* 碰撞的处理方式必须不同于 *Facet* + *Sphere* 碰撞。作出这些决策的对象称为 分派器，理解它们对于理解 Yade 的运行机制至关重要；下文将对此进行讨论。

手动为每个颗粒明确指定全部 4 个属性并不实际；系统中定义了一些实用函数，可以使用所有必要的组成部分创建这些属性。例如，我们可以使用 *utils.sphere* 创建球形颗粒：

```
Yade [31]: s=utils.sphere(center=[0,0,0],radius=1)

Yade [32]: s.shape, s.state, s.mat, s.bound
Out[32]:(<Sphere instance at 0x38d9450>,
```

（下页继续）

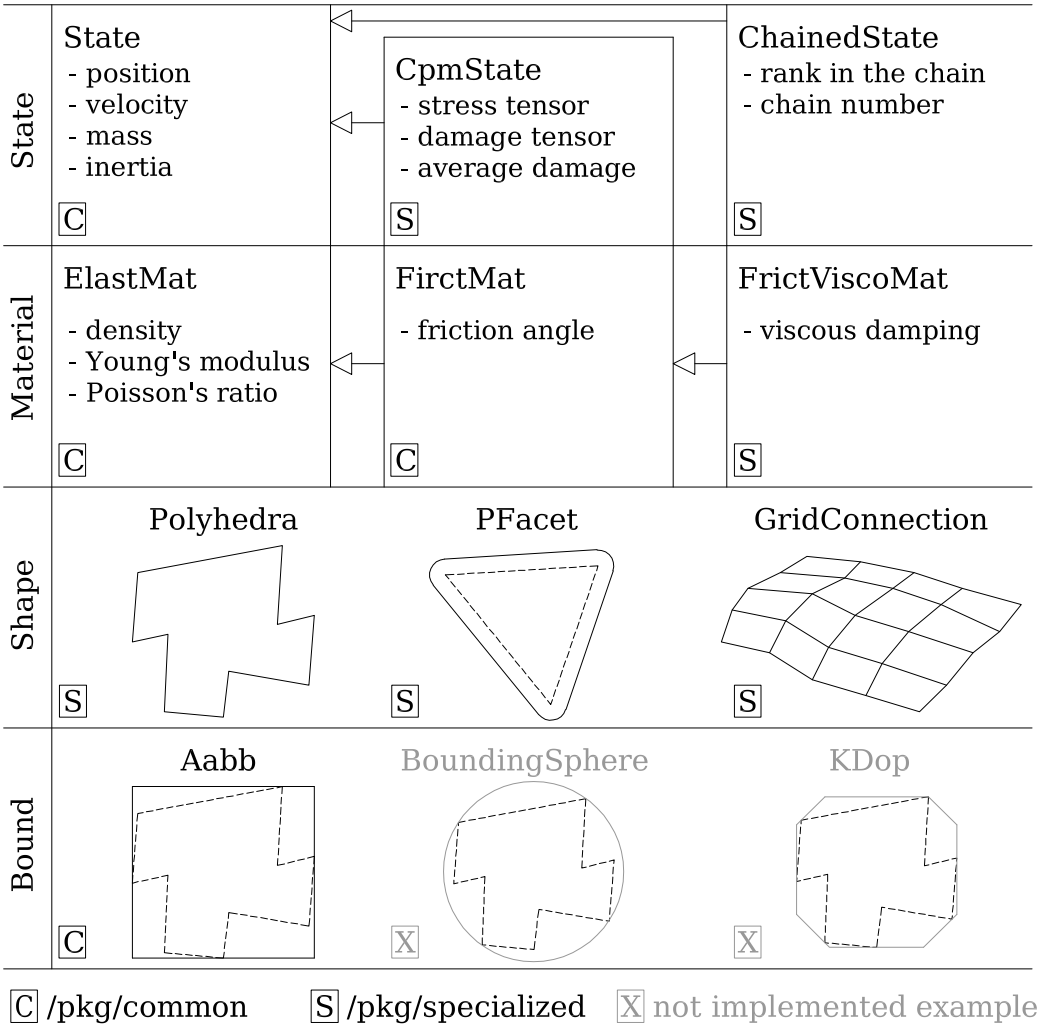


图 1：用于描述刚体的状态、*CpmState*、链式状态，材料、*ElastMat*、*FrictMat*、*FrictViscoMat*、形状、多面体、*PFacet*、网格连接，*Bound*，*Aabb*。

（接上一页）

```
 <State instance at 0x3b2d060>,
 <FrictMat instance at 0x3d09ef0>,
 None)

Yade [33]: s.state.pos
Out[33]: Vector3(0,0,0)

Yade [34]: s.shape.radius
Out[34]: 1.0
```

我们可以看到，创建了一个材料类型为 *FrictMat* （默认类型，除非你另行指定其他 材料）且包围体类型为 *Aabb* （轴对齐包围盒）的球体。它的位置在原点，半径为 1.0。最后，可以将此对象插入模拟中；我们还可以再插入一个球体。

```
Yade [35]: O.bodies.append(s)
Out[35]: 0

Yade [36]: O.bodies.append(utils.sphere([0,0,2],.5))
Out[36]: 1
```

在每种情况下，返回值都是所插入刚体的 *Body.id* 。

由于此前模拟中没有任何内容，第一个球体的标识符为 0，第二个为 1。除非你之后想访问这个特定刚体，否则无需保存标识符值；它会保存在 刚体 自身内部。你可以通过标识符访问刚体：

```
Yade [37]: O.bodies[1].state.pos
Out[37]: Vector3(0,0,2)

Yade [38]: O.bodies[100]    # error because there are only two bodies
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
----> 1 O.bodies[100]    # error because there are only two bodies

IndexError: Body id out of range.
```

出于标识符唯一性的考虑，不允许重复添加同一个刚体：

```
Yade [39]: O.bodies.append(s)  # error because this sphere was already added
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
----> 1 O.bodies.append(s)  # error because this sphere was already added

IndexError: Body already has id 0 set; appending such body (for the second time) is not␣
 →allowed.
```

可以使用标准的 Python 迭代语法遍历刚体：

```
Yade [40]: for b in O.bodies:
....:     print(b.id,b.shape.radius)
....:
0 1.0
1 0.5
```

相互作用

相互作用 *ns* 始终存在于一对刚体之间；通常由碰撞器根据空间

al

邻近关系创建；不过，也可以显式创建，并独立于距离而存在。每个相互作用包含两个组成部分：

IGeom 保存发生碰撞的两个颗粒的几何构型；随着相关颗粒移动，它会自动更新，并可用于查询各种几何特征，例如穿透距离或剪切应变。

> 根据颗粒形状类型的组合，存储需求可能有所不同；因此存在若干派生类，例如用于表示球体+球体、圆柱体+球体等形状之间接触几何的类。不过请注意，许多类型的接触都可以用基本的球体-球体几何来表示（例如在*Ig2_墙_球体_Sc-Geom*中）。

IPhys 表示相互作用的非几何特征；其中一些特征通过某种平均算法（例如根据颗粒的杨氏模量计算接触刚度）由接触颗粒的材料计算得出，另一些则可能是损伤等内部变量。

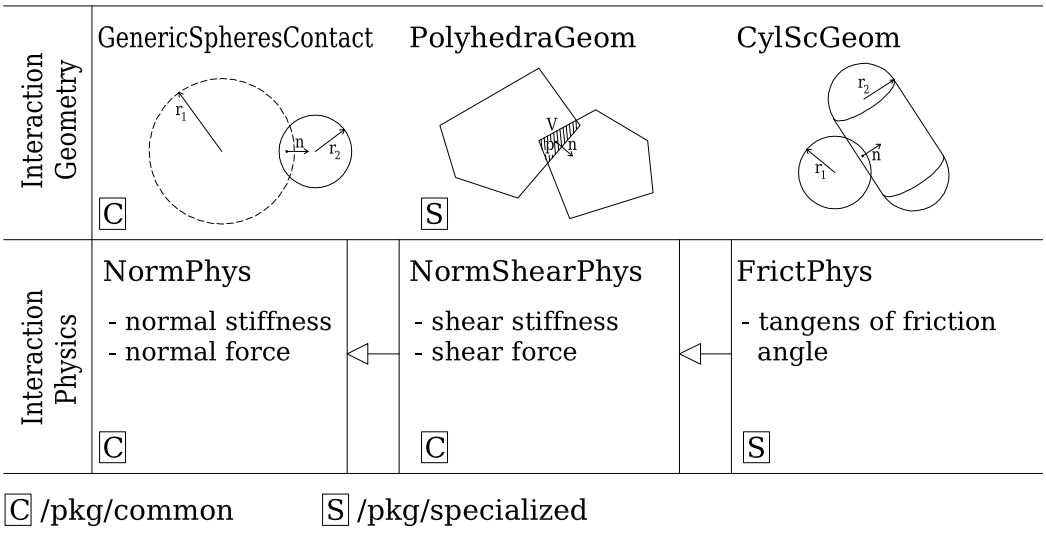


图 2：用于描述相互作用的具体类示例：*IGeom*、*Generic-SpheresContact*、*PolyhedraGeom*、*CylScGeom*、*IPhys*、*NormPhys*、*NormShearPhys*、*FrictPhys*。

假设此时相互作用已经创建。我们可以通过标识符对访问它们：

```
Yade [41]: O.interactions[0,1]
Out[41]: <Interaction instance at 0x384c250>

Yade [42]: O.interactions[1,0]      # order of ids is not important
Out[42]: <Interaction instance at 0x384c250>

Yade [43]: i=O.interactions[0,1]

Yade [44]: i.id1,i.id2
Out[44]: (0, 1)

Yade [45]: i.geom
Out[45]: <ScGeom instance at 0x315d280>

Yade [46]: i.phys
Out[46]: <FrictPhys instance at 0x3d07c20>

Yade [47]: O.interactions[100,10111]      # asking for non existing interaction throws exception
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
```

（续见下页）

（接上一页）

```
----> 1 O.interactions[100,10111]     # asking for non existing interaction throws exception

IndexError: No such interaction
```

广义力

广义力包括力、力矩、强制位移和旋转；它们只会临时存储，在一个计算步内有效，之后会重置为零。出于并行计算的考虑，它们以累加器的形式工作，也就是说，只能对其进行累加、读取和重置。

```
Yade [48]: O.forces.f(0)
Out[48]: Vector3(0,0,0)

Yade [49]: O.forces.addF(0,Vector3(1,2,3))

Yade [50]: O.forces.f(0)
Out[50]: Vector3(1,2,3)
```

您很少会从 Python 修改力；通常是在 c++ 代码 中完成，相关文档可参阅《程序员手册》。

函数组件

在典型的离散元模拟中，以下序列会被反复运行：

- 重置刚体上来自前一步的力
- 近似碰撞检测（第1遍）
- 检测刚体之间的精确碰撞，并根据需要更新相互作用
- 求解相互作用，将力施加到刚体上
- 应用其他外部条件（例如重力）。
- 通过积分运动方程，根据力改变刚体的位置。

每项操作都由一个引擎表示，它是模拟中的功能元素。引擎的序列称为模拟循环。

引擎

图 *img-yade-iter-loop*所示的模拟循环可以用 Python 描述如下（细节将在稍后说明）；每个 `O.engines` 项目都是一个派生自引擎类型的实例：

```
O.engines=[# reset forcesForceResetter(),
# approximate collision detection,  create interactions
InsertionSortCollider([Bo1_Sphere_Aabb(),Bo1_Facet_Aabb()]),
# handle interactionsInteractionLoop(
[Ig2_Sphere_Sphere_ScGeom(),Ig2_Facet_Sphere_ScGeom()],
[Ip2_FrictMat_FrictMat_FrictPhys()],
[Law2_ScGeom_FrictPhys_CundallStrack()], ), # apply other conditions
GravityEngine(gravity=(0,0,-9.81)),
```

（续见下页）

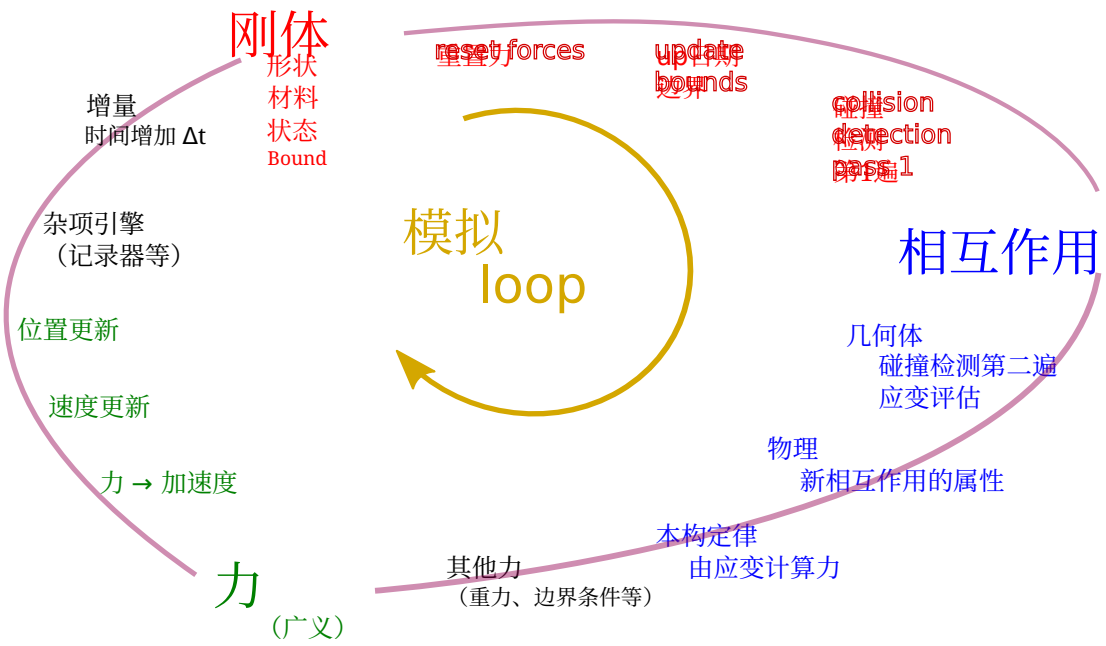


图 3：典型的模拟循环；每一步都从 11 点钟方向的以刚体为中心的步骤开始，接着是相互作用步骤、力施加步骤和其他事项，最后以时间更新结束。

（续前页）

```
        # update positions using Newton's equations
        NewtonIntegrator()
]
```

引擎有 3 种基本类型：

全局引擎 作用于整个模拟（例如 *ForceResetter* 会将作用于刚体的力归零，或 *GravityEngine* 遍历所有刚体，并根据其质量施加力）

部分引擎 仅作用于某些预选刚体（例如 力引擎 对某些 选定的 刚体施加恒力）

分派器 本身不执行任何计算；它们只是调用其他函数，这些函数由函数对象表示，即 函子。每个函子都是专门化的，能够处理特定的对象类型；当分派器处理此类对象时，就会分派相应的函子。

分派器与函子

对于近似碰撞检测（第1遍），我们要计算模拟中全部刚体的边界；假设我们要计算轴对齐包围盒类型的边界。由于具体形状不同，所用的精确算法也不同，因此我们需要提供函子来处理所有具体情况。在上文 `O.engines=[…]`声明的内容中，有这样一行：

```
InsertionSortCollider([Bo1_Sphere_Aabb(),Bo1_Facet_Aabb()])
```

创建了*InsertionSortCollider*（它内部使用*BoundDispatcher*，不过这是实现细节）。它会遍历所有刚体，并根据每个刚体的形状类型，分派一个函子来创建或更新该刚体的边界。在此例中，它有两个函子，一个处理球体，另一个处理面片。

这个名称由几个部分组成： `Bo` （用于创建*Bound*的函子），它接受1 种类型，即*Sphere*，并创建一个*Aabb*（轴对齐包围盒；它派生自*Bound*）。*Aabb* 对象由*InsertionSortCollider* 自身使用。所有 `Bo1` 函子都派生自*BoundFunctor*。

下一部分，读数

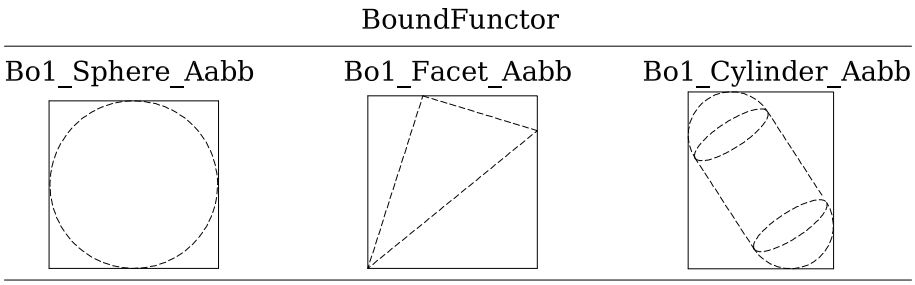


图 4：示例生成 Aabb的绑定函子，可接受多种不同类型，例如*Sphere*、*Facet*或*Cylinder*。在所示算例中， `Bo1` 函子从单个特定*Shape*实例生成*Aabb*，因此函子名称中的数量为`1` 。每个函子都使用*Shape*的特定几何信息，即*Facet* 中节点的位置或球体的半径来计算*Aabb*。

```
InteractionLoop(
[Ig2_Sphere_Sphere_ScGeom(),Ig2_Facet_Sphere_ScGeom()],
[Ip2_FrictMat_FrictMat_FrictPhys()],
[Law2_ScGeom_FrictPhys_CundallStrack()],),
```

隐藏 3 个整数 部的分派器位于*InteractionLoop* 引擎中；它们全部作用于相互作用，并且 e, 为了提高性 能，将它们组合在一起：

IGeomDispatcher* 使用 *IGeomFunctor 采用第一组函子（`Ig2`），这些函子根据 2 种形状 对象的组合进行分派。分派后的函子会解析精确碰撞构型，并在发生碰撞时创建与该相互作用关联的相互作用几何 *IGeom* （名称中的 `Ig` 由此而来）。即使近似碰撞检测中发生了重叠（例如，*Aabb* 发生重叠，但形状并未重叠），函子也可能判定实际上并未发生碰撞。在这种情况下，该属性会被设为 false，并安排移除该相互作用。

1. 第一个函子 *Ig2_Sphere_Sphere_ScGeom*，在两个球体 相互作用时调用，并在适当情况下创建 *ScGeom* 实例。2. 第二个函子 *Ig2_Facet_Sphere_ScGeom*，在面片 与 球体相互作用时调用，并可能再次创建一个 *ScGeom* 实例。

所有 `Ig2` 函子都派生自 *IGeomFunctor* （相关文档可在同一处查阅）。

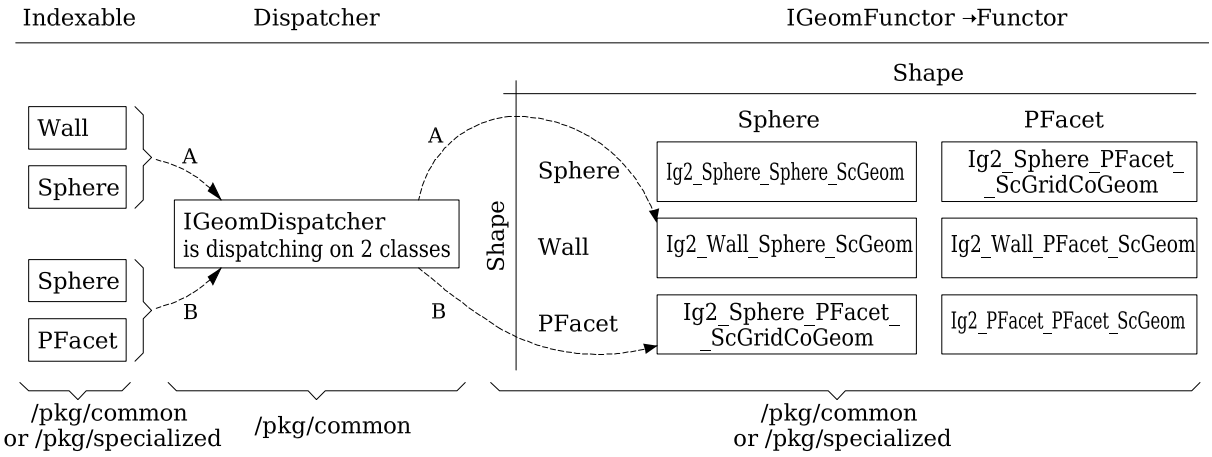


图 5：生成 *ScGeom* 或 *ScGridCoGeom* 的相互作用几何函子 接受两种不同类型（因此名称中含有 2，例如 `Ig2`）的对象，如 球体、墙 或 *PFacet*。这些函子会使用 形状 的特定几何信息（即 *PFacet* 中节点的位置或球体半径）来计算相互作用几何。

IPhysDispatcher* 使用 *IPhysFunctor 根据两个 材料的组合，将调用分派到第二组函子；这些函子返回一个 *IPhys* 实例（其 `Ip` 前缀）。在本例中，只使用一个函子，*Ip2_FrictMat_FrictMat_FrictPhys*，它根据两个 *FrictPhys* 创建*FrictMat*。

`Ip2` 函子派生自*IPhysFunctor*。

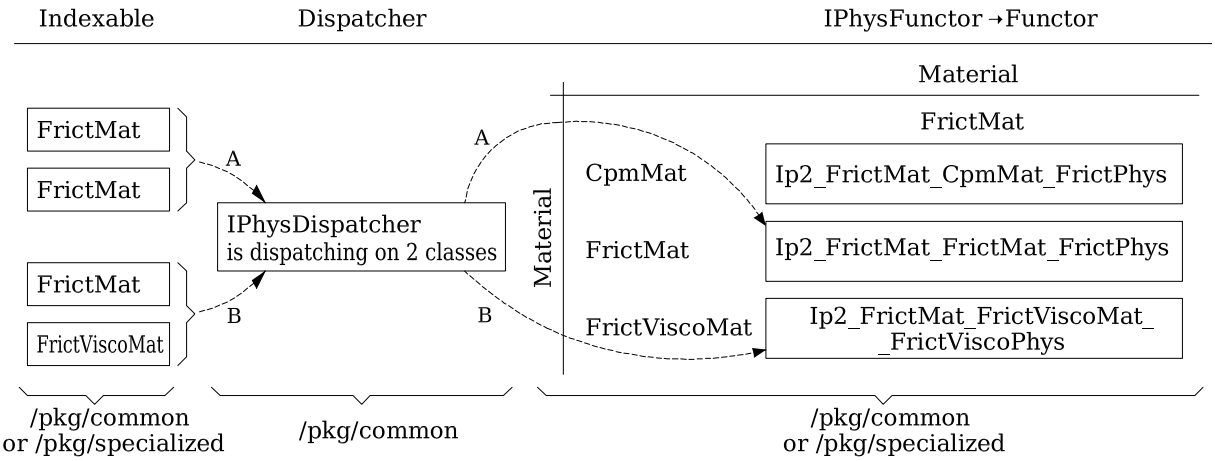


图 6：示例相互作用物理函子（*Ip2_FrictMat_CpmMat_FrictPhys*、*Ip2_FrictMat_Frict- Mat_ FrictPhys* 和 *Ip2_FrictMat_FrictViscoMat_FrictViscoPhys*）生成 *FrictPhys* 或 *FrictVisco- Phys*，并接受两种不同类型的材料（因此为 `Ip2`），例如 *CpmMat*、*FrictMat* 或 *FrictViscoMat*。

LawDispatcher* 使用 *LawFunctor 将分派到第三组函子，其依据是每个特定相互作用中由前置函子创建的 *IGeom* 和 *IPhys*（因此其名称中再次包含 `2`）。`Law2` 函子代表本构定律；它们通过计算作用于相互作用体的力（排斥力、吸引力、剪切力.....）来解析相互作用，或以其他方式更新相互作用状态变量。

`Law2` 所有函子都继承自*LawFunctor*。

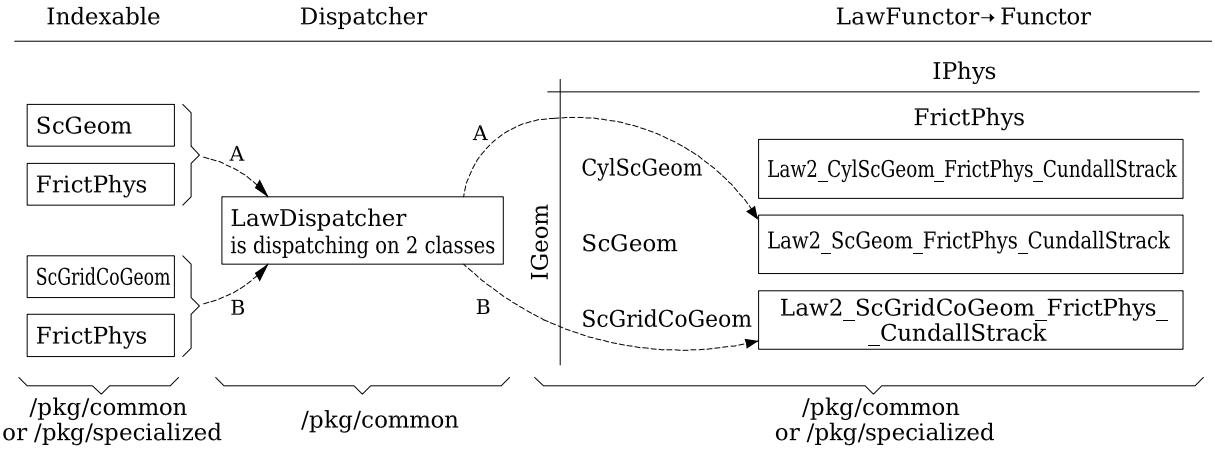


图 7：示例 *Law* 函子（*定律2_CylScGeom_FrictPhys_CundallStrack*、*定律2_ScGeom_Frict-Phys_ CundallStrack* 和*定律2_ScGridCoGeom_FrictPhys_CundallStrack*）它们各自根据选定的本构定律计算力。

由前面的函子生成并由后续函子接收的类型构成一条链；用户有责任满足类型要求（参见图 *img-dispatch-loop*）。否则会引发异常（并附带说明）。

注意： Yade 启动时，O.engines 会被填充为一个合理的默认列表，因此尝试简单操作时，无需严格 ly

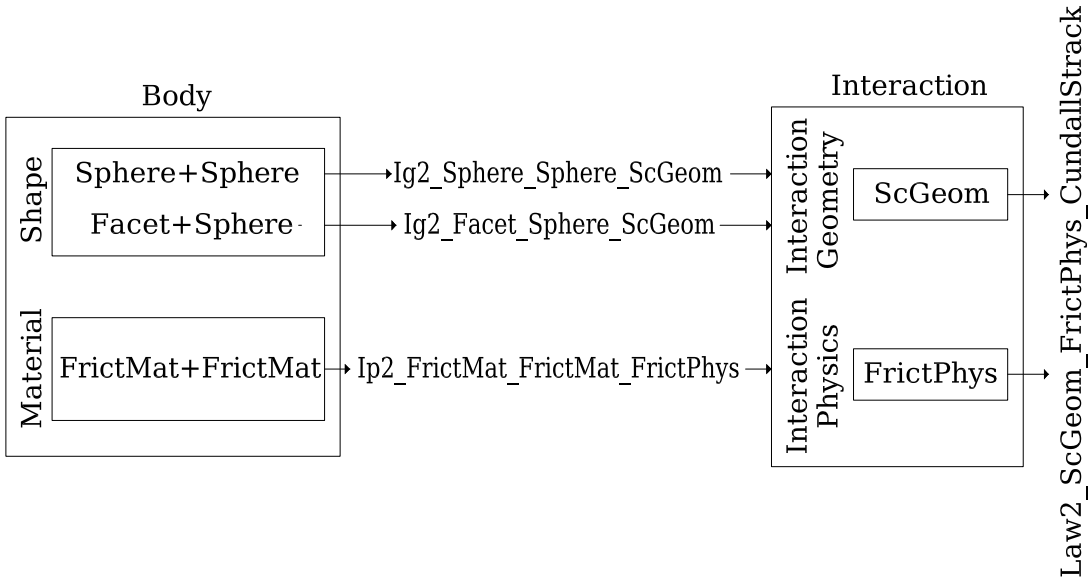


图 8：生成并接受特定类型的函子链。在所示算例中， `Ig2` 函子根据所有已处理的形状组合生成 *ScGeom* 实例； `Ig2` 函子生成 *FrictMat*。本构律函子 `Law2` 接受所生成的类型组合。请注意，这些类型在函子的类名中有所说明。

重新定义它。默认场景能够正确处理具有摩擦属性的球体、盒体和面片，并动态调整时间步长。示例请参见examples/simple-scene/simple-scene-default-engines.py。

1.2 教程

本教程最初是为德累斯顿工业大学 / 土木工程学院 / 岩土工程研究所 / 举办的一门课程编写的讲义，时间为2011年1月。本教程旨在为没有建模经验但具备力学知识的读者提供快速且相当实用的简介。读者应具备一定的计算机素养，不过基础知识将在实践操作部分中进行复习。

实际上，这门课程并未完全按照本文档进行，而是基于交互式编写和注释简单的示例，这些示例大多由参与者提出；感谢他们提供的想法和建议。

1.2.1 简介

本章简介总结于以下演示文稿中：Yade：过去、现在与未来，另附一些不同的示例。该演示文稿来自2011年，未包含最新内容。截至2019年，其内容在事实上仍然正确。

1.2.2 实践操作

Shell 基础

目录树

目录树是在操作系统中组织文件的层级方式。Linux 中一个典型的（简化）目录树如下：

```
/            Root
  boot        System startup
  bin         Low-level programs
  lib         Low-level libraries
  dev         Hardware access
  sbin        Administration programs
  proc        System information
  var         Files modified by system services
  root        Root (administrator) home directory
  etc         Configuration files
  media       External drives
  tmp         Temporary files
  usr         Everything for normal operation (usr = UNIX system resources)
       bin       User programs
       sbin      Administration programs
       include   Header files for c/c++
       lib       Libraries
       local     Locally installed software
       doc       Documentation
  home        Contains the user's home directories
       user      Home directory for user
       user1     Home directory for user1
```

请注意，只有一个 root `/`；其他所有圆盘（例如 USB 闪存盘）都会挂载到目录树中的某个点（例如 in `/media`）。

Shell 导航

Shell 是 UNIX 命令行，是与计算机交互的界面。别害怕。

四处移动

Shell 始终由某个 user在某个具体的 machine中操作；这两者都是固定的。我们可以在目录结构中移动，而我们当前所在的位置就是当前目录。默认情况下，它是主目录，其中包含属于相应用户的全部文件：

```
user@machine:~$                          # user operating at machine, in the directory ~ (= user
 →'s home directory)
user@machine:~$ ls .                     # list contents of the current directory
user@machine:~$ ls foo                   # list contents of directory foo,  relative to the␣
 →dcurrent directory ~ (= ls ~/foo = ls /home/user/foo)
user@machine:~$ ls /tmp                  # list contents of /tmp
user@machine:~$ cd foo                   # change directory to foo
user@machine:~/foo$ ls ~                 # list home directory (= ls /home/user)
user@machine:~/foo$ cd bar               # change to bar (= cd ~/foo/bar)
user@machine:~/foo/bar$ cd ../../foo2    # go to the parent directory twice,  then to foo2 (cd ~/
 →foo/bar/../../foo2 = cd ~/foo2 = cd /home/user/foo2)
user@machine:~/foo2$ cd                  # go to the home directory (= ls ~ = ls /home/user)
user@machine:~$
```

用户通常只有在主目录（简称~，通常为 `/home/user`）和 `/tmp` 中才有写入权限（即修改文件的权限），而在系统的大多数其他部分则只有读取文件的权限：

```
user@machine:~$ ls /root    # see what files the administrator has
ls: cannot open directory /root: Permission denied
```

Keys

命令行上有用的键包括：

<制表符>	显示当前输入内容的可能补全项（尽情使用！）
^C（=Ctrl+C）	删除当前直线
^D	退出 Shell
↑↓	在命令历史中上下移动
^C	中断当前正在运行的程序
^\	终止当前正在运行的程序
Shift-PgUp	向上滚动屏幕（显示之前的输出）
Shift-PgDown	向下滚动屏幕（显示未来的输出；仅适用于量子计算机）

运行程序

当一个程序正在运行时（未提供其完整路径），系统会在多个目录中搜索同名程序；这些目录由 `$PATH` 指定：

```
user@machine:~$ echo $PATH     # show the value of $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
user@machine:~$ which ls       # say what is the real path of ls
```

命令行的第一部分是要运行的程序（`which`），其余部分是参数（本例中为 `ls` ）。程序能识别哪些参数由程序本身决定。许多程序可以接受称为选项 的特殊参数，它们以 `-` （后接一个字母）或 `--` （后接若干单词）开头；常见选项之一是 `-h` 或 `--help`，用于显示程序的使用方法（试试 `ls --help`）。

每个程序通常都有完整的手册页 （或*man* 手册页）文档，例如可以使用 `man ls` 查看（使用 `q` 退出）

启动 Yade

如果计算机上安装了 Yade，那么（大致来说）它可以像其他程序一样运行；如果不带任何参数运行，就会进入“对话模式”，并显示一个命令行：

```
user@machine:~$ yadeWelcome to Yade 2019.01aTCP python prompt on localhost:9002,
 auth cookie `adcusk'XMLRPC info provider on http://localhost:21002
[[ ^L clears screen, ^U kills line. F12 controller, F11 3d view, F10 both, F9 generator, F8␣
→plot. ]]Yade [1]:                                          #### hit ^D to exit
Do you really want to exit ([y]/n)?Yade: normal exit.
```

命令行实际上是 `python`，并加入了一些 Yade 特有功能。（使用 `python` 或 `ipython` 命令可以运行纯 Python 解释器。）
无需在命令行中逐条输入命令，可以将它们写入文件（扩展名为 .py），然后将其作为参数传递给 Yade：

```
user@machine:~$ yade simulation.py
```

如需查看完整帮助，请参阅 `man yade`

练习

1. 打开终端，进入主目录 2. 创建一个新的空文件，并将其保存到 `~/first.py`3. 将目录切换到 `/tmp`；删除文件 `~/first.py`4. 运行程序 `xeyes`5. 查看 Yade 的帮助。6. 查看 Yade 的手册页 7. 运行 Yade，退出后再次运行。

Python 基础

我们假设读者熟悉 Python 教程，这里只简要回顾一些基本功能。以下内容可在纯 Python 解释器中运行（`python` 或 `ipython`），也可在 Yade 中运行；Yade 是 Python 的超集。

数值运算和模块：

```
Yade [1]: (1+3*4)**2          # usual rules for operator precedence, ** is exponentiation
Out[1]: 169

Yade [2]: import math         # gain access to "module" of functions

Yade [3]: math.sqrt(2)        # use a function from that module
Out[3]: 1.4142135623730951

Yade [4]: import math as m    # use the module under a different name

Yade [5]: m.cos(m.pi)
Out[5]: -1.0

Yade [6]: from math import *  # import everything so that it can be used without module name

Yade [7]: cos(pi)
Out[7]: -1.0
```

变量：

```
Yade [8]: a=1; b,c=2,3        # multiple commands separated with ;, multiple assignment

Yade [9]: a+b+c
Out[9]: 6
```

序列

列表

列表是长度可变、可以修改的序列，用花括号 `[...]`表示，其元素通过数值索引访问：

```
Yade [10]: a=[1,2,3]          # list of numbers

Yade [11]: a[0]               # first element has index 0
Out[11]: 1
```

（续见下页）

（接上一页）

```
Yade [12]: a[-1]              # negative counts from the end
Out[12]: 3

Yade [13]: a[3]               # error
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
----> 1 a[3]               # error

IndexError: list index out of range

Yade [14]: len(a)             # number of elements
Out[14]: 3

Yade [15]: a[1:]              # from second element to the end
Out[15]: [2, 3]

Yade [16]: a+=[4,5]           # extend the list

Yade [17]: a+=[6]; a.append(7) # extend with single value, both have the same effect

Yade [18]: 9 in a             # test presence of an element
Out[18]: False
```

可以通过多种方式创建列表：

```
Yade [19]: range(10)
Out[19]: range(0, 10)

Yade [20]: range(10)[-1]
Out[20]: 9
```

列表：小于 20 的偶数的平方，即 $\left\{a^2 \ \forall a \in \{0,\cdots,19\} \mid 2 \| a\right\}$（请注意相似之处）：

```
Yade [21]: [a**2 for a in range(20) if a%2==0]
Out[21]: [0, 4, 16, 36, 64, 100, 144, 196, 256, 324]
```

元组

元组是不可变序列：

```
Yade [22]: b=(1,2,3)

Yade [23]: b[0]
Out[23]: 1

Yade [24]: b[0]=4              # error
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
----> 1 b[0]=4              # error

TypeError: 'tuple' object does not support item assignment
```

字典

从键到值的映射：

```
Yade [25]: ende={'one':'ein' , 'two':'zwei' , 'three':'drei'}
Yade [26]: de={1:'ein' , 2:'zwei' , 3:'drei'}; en={1:'one' , 2:'two' , 3:'three'}
Yade [27]: ende['one']          ## access valuesOut[27]: 'ein'Yade [28]: de[1],
 en[2]Out[28]: ('ein', 'two')
```

函数、条件语句

```
Yade [29]: 4==5
Out[29]: False

Yade [30]: a=3.1

Yade [31]: if a<10:
   ....:     b=-2          # conditional statement
   ....: else:
   ....:     b=3
   ....:

Yade [32]: c=0 if a<1 else 1      # trenary conditional expression

Yade [33]: b,c
Out[33]: (-2, 1)

Yade [34]: def square(x): return x**2    # define a new function
   ....:

Yade [35]: square(2)                     # and call that function
Out[35]: 4
```

练习

1. 阅读以下代码，并说明 `a` 和 `b` 的值：

```
a=range(5)
b=[(aa**2 if aa%2==0 else -aa**2) for aa in a]
```

Yade 基础

Yade 对象的构造方式如下（这一过程也称为“实例化”，因为我们创建了抽象类的具体实例：单个球体是抽象类 球体的一个实例，就像苏格拉底是“人”的一个实例一样）：

```
Yade [36]: Sphere           # try also Sphere?
Out[36]: yade.wrapper.Sphere

Yade [37]: s=Sphere()       # create a Sphere,  without specifying any attributes

Yade [38]: s.radius         # 'nan' is a special value meaning "not a number" (i.e. not␣
 →defined)
Out[38]: nan
```

（续见下页）

（续前页）

```
Yade [39]: s.radius=2        # set radius of an existing object

Yade [40]: s.radius
Out[40]: 2.0

Yade [41]: ss=Sphere(radius=3)    # create Sphere, giving radius directly

Yade [42]: s.radius, ss.radius      # also try typing s.<tab> to see defined attributes
Out[42]: (2.0, 3.0)
```

颗粒

颗粒是模拟中的“数据”部分；它们是将经历某些过程的对象，但目前还未定义这些过程。

单个粒子

有许多预定义函数可用于创建特定类型的颗粒；要创建球体，需要（例如，参见 *utils.sphere* 的源代码）：

1. 创建刚体2. 将 *Body.shape* 设置为 *Sphere* 的一个实例，并指定半径

3. 设置 *Body.material* （使用最后定义的材料，否则创建默认材料）　　d)
4. 在 *Body.state*中设置位置和方向，并根据 材料和 形状计算质量和转动惯量

为避免这些工作，*utils* 模块中定义了简写函数；举几个例子： *utils.sphere*、*utils.facet*、*utils.wall*。

```
Yade [43]: s=utils.sphere((0,0,0),radius=1)    # create sphere particle centered at (0,0,0)␣
→with radius=1
Yade [44]: s.shape                      # s.shape describes the geometry of the particle
Out[44]: <Sphere instance at 0x3f1cd20>
Yade [45]: s.shape.radius               # we already know the Sphere classOut[45]: 1.0
Yade [46]: s.state.mass, s.state.inertia # inertia is computed from density and geometry
Out[46]:(4188.790204786391,
Vector3(1675.516081914556253,1675.516081914556253,1675.516081914556253))
Yade [47]: s.state.pos                  # position is the one we prescribed
Out[47]: Vector3(0,0,0)
Yade [48]: s2=utils.sphere((-2,0,0),radius=1,fixed=True)     # explanation below
```

在上一个示例中，粒子通过 `fixed=True` 参数固定在空间中，该参数传递给 *utils.sphere*；这样的粒子不会移动，从而形成基本边界条件。

一个颗粒 对象尚未成为模拟的一部分；要将其纳入模拟，需要调用一个特殊函数 *O.bodies.append*（另请参见 *O.bodies* 和 *Scene*）：

```
Yade [49]: O.bodies.append(s)           # adds particle s to the simulation; returns id of␣
→the particle(s) added
Out[49]: 24
```

包

有一些函数可用于生成*pack*模块中的特定颗粒排列；例如，可以使用*pack.SpherePack*类生成球体的球体云（随机疏松堆积）：

```
Yade [50]: from yade import pack

Yade [51]: sp=pack.SpherePack()                    # create an empty cloud; SpherePack contains␣
 →only geometrical information

Yade [52]: sp.makeCloud((1,1,1),(2,2,2),rMean=.2) # put spheres with defined radius inside box␣
 →given by corners (1,1,1) and (2,2,2)
Out[52]: 6

Yade [53]: for c,r in sp: print(c,r)                # print center and radius of all particles␣
 →(SherePack is a sequence which can be iterated over)
   ....:
Vector3(1.274443445943540087,1.367880531586413095,1.275812677797829142) 0.2
Vector3(1.395144492365041344,1.682654071894964964,1.504498259316015663) 0.2
Vector3(1.691901039056097789,1.646021131505697621,1.775464815231047933) 0.2
Vector3(1.698276069049370784,1.239026483132536161,1.307143271925819583) 0.2
Vector3(1.546530002061732745,1.248592120112199888,1.780182798004842359) 0.2
Vector3(1.778823825441760 32,1.662755981140189299,1.256899030310328236) 0.2

Yade [54]: sp.toSimulation()                       # create particles and add them to the␣
 →simulation
Out[54]: [25, 26, 27, 28, 29, 30]
```

边界

utils.facet（三角形 面片）和 *utils.wall*（无限坐标轴对齐平面 墙）几何体通常用于定义边界。例如，可以这样为模拟创建一个“地板”：

```
Yade [55]: O.bodies.append(utils.wall(-1,axis=2))
Out[55]: 31
```

还有其他便捷函数（例如*utils.facetBox*用于创建封闭或开放的矩形盒，或一系列*ymport*函数）

查看内部

The 可以通过多种方式检查模拟。所有数据都可以直接从 Python 访问 ：

```
Yade [56]: len(O.bodies)
Out[56]: 32

Yade [57]: O.bodies[10].shape.radius   # radius of body #10 (will give error if not sphere,␣
 →since only spheres have radius defined)
Out[57]: 0.16

Yade [58]: O.bodies[12].state.pos      # position of body #12
Out[58]: Vector3(1.224408970802669305,1.614102010700998235,1.392374433964128411)
```

除此之外，Yade 在启动时会显示以下内容（位于命令行之前的那一行）：

```
[[ ^L clears screen, ^U kills line. F12 controller, F11 3d view, F10 both, F9 generator, F8␣
 →plot. ]]
```

控制器 按下 F12 会打开一个用于控制模拟的窗口。虽然大型模拟通常无需人工干预（以“无头”模式运行，不进行任何图形交互），但在小型示例中，这项功能很方便。窗口中还会显示模拟的基本信息（稍后会用到）。

三维视图 可以通过 F11 打开三维视图（也可以点击控制器 中的按钮——见下文）。可以使用许多键盘快捷键来操作它（按下 h 可查看基本帮助），也可以用鼠标移动、旋转和缩放。控制器的“显示”选项卡中可以设置与显示相关的选项（例如是否绘制颗粒）。

点击控制器中相应的按钮即可打开检查器 检查器 。它会显示并更新当前模拟的内部数据。具体来说，可以查看引擎、颗粒（刚体）和相互作用（相互作用）。点击各个属性名称即可跳转到文档中相应的章节。

练习

1. 这段代码会做什么？

```
Yade [59]: O.bodies.append([utils.sphere((2*i,0,0),1) for i in range(1,20)])
Out[59]: [32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]
```

2. 创建一个简单模拟：在盒子中放入球体云 (0,0,0) 并且 (1,1,1) 平均半径为 .1。（提示：*pack.SpherePack.makeCloud*）3. 将上面创建的球体云用角点为 (0,0,0) 和 (1,1,1) 的盒子围起来；保持盒子顶部敞开。（提示：*utils.facetBox*；在命令行中输入 utils.facetBox? 或 utils.facetBox?? 可获取帮助）4. 打开三维视图，尝试放大和缩小；调整坐标轴，使 z 朝上，y 朝右，且 x 朝向你。

引擎

引擎定义了颗粒所执行的进程。正如我们从理论简介中了解到的，引擎的序列称为模拟循环。下面定义一个简单的相互作用循环：

```
Yade [60]: O.engines=[                    # newlines and indentations are not important until␣
→the brace is closed
   ....:     ForceResetter(),
   ....:     InsertionSortCollider([Bo1_Sphere_Aabb(),Bo1_Wall_Aabb()]),
   ....:     InteractionLoop(            # dtto for the parenthesis here
   ....:         [Ig2_Sphere_Sphere_ScGeom(),Ig2_Wall_Sphere_ScGeom()],
   ....:         [Ip2_FrictMat_FrictMat_FrictPhys()],
   ....:         [Law2_ScGeom_FrictPhys_CundallStrack()]
   ....:     ),
   ....:     NewtonIntegrator(damping=.2,label='newtonCustomLabel')      # define a label␣
→newtonCustomLabel under which we can access this engine easily
   ....: ]
   ....:

Yade [61]: O.engines
Out[61]:
[<ForceResetter instance at 0x3cc9610>,
 <InsertionSortCollider instance at 0x2ac5490>,
 <InteractionLoop instance at 0x3c3c3d0>,
 <NewtonIntegrator instance at 0x37bfa50>]

Yade [62]: O.engines[-1]==newtonCustomLabel    # is it the same object?
Out[62]: True
```

（续见下页）

（接上一页）

```
Yade [63]: newtonCustomLabel.damping
Out[63]: 0.2
```

不必将所有内容都输入命令行，也可以将模拟描述在一个文件（脚本）中，然后将该文件作为参数运行yade。因此，除非确有必要，我们不再显示命令行，而只显示脚本部分。如下：

```
O.engines=[                    # newlines and indentations are not important until the brace is␣
→closed
        ForceResetter(),
        InsertionSortCollider([Bo1_Sphere_Aabb(),Bo1_Wall_Aabb()]),
        InteractionLoop(           # dtto for the parenthesis here
                [Ig2_Sphere_Sphere_ScGeom(),Ig2_Wall_Sphere_ScGeom()],
                [Ip2_FrictMat_FrictMat_FrictPhys()],
                [Law2_ScGeom_FrictPhys_CundallStrack()]
        ),
        GravityEngine(gravity=(0,0,-9.81)),                    # 9.81 is the gravity␣
→acceleration,  and we say that
        NewtonIntegrator(damping=.2,label='newtonCustomLabel') # define a label under which we␣
→can access this engine easily
]
```

除了运行引擎之外，定义它们的运行频率也同样重要。有些引擎只会偶尔运行（稍后我们会看到），而大多数引擎则会一直运行；引擎连续两次运行之间的时间称为时间步长（Δt）。时间步长的取值存在一个数学极限，称为临界时间步长，它根据颗粒的属性计算得出。由于已有相应的函数，我们只需使用*utils.PWaveTimeStep*来设置时间步长：

```
O.dt=utils.PWaveTimeStep()
```

E模拟循环每次结束时，时间 `O.time` 都会按时间步长 `O.dt`推进：

```
Yade [64]: O.dt=0.01

Yade [65]: O.time
Out[65]: 0.0

Yade [66]: O.step()

Yade [67]: O.time
Out[67]: 0.01
```

进行单次模拟实验时，将其保存到内存中会很方便；在完成所有定义后，可以通过以下方式实现：

```
O.saveTmp()
```

练习

1. 像上面的示例一样定义引擎，运行检查器并依次点击各个引擎，查看它们的序列。

2. 编写一个简单的脚本，使其能够

   1. 按照上一个练习定义颗粒（盒子内的球体云，顶部敞开）；2. 定义一个如上所示的简单模拟循环；3. 将 Δt 设为临界 P波 Δt

4. 将模拟初始状态保存到内存

3. 多次运行先前定义的模拟，同时更改时间步长的值（使用 按钮重新加载初始构型）。

   1. 随着你增大 Δt 至高于 P波值时，观察会发生什么。2. 尝试在运行模拟前更改重力参数。3. 尝试更改阻尼

4. 重新加载模拟，打开三维视图，再打开检查器，在三维视图中选择一个颗粒（按住 Shift 键并单击）。然后运行模拟，观察作用于该颗粒的力如何变化；在模拟进行到一半时暂停，查看该颗粒的相互作用。

5. 到什么程度时，我们可以认为沉积已完成，从而停止模拟?

另请参阅：

弹跳球示例展示了一个基本模拟。

1.2.3 数据挖掘

Read

本地数据

模拟中的所有数据都可以通过 Python 访问；打开检查器后，可以点击各种数据的蓝色标签：单击鼠标左键可查看文档，中键单击可复制对象名称（使用 `Ctrl-V` 或中键单击，可将其粘贴到其他位置）。其他有趣的对象还包括（完整列表请参见*Omega* ）：

1. *O.engines*

   引擎通过其在模拟循环中的索引（位置）访问：

```
O.engines[0]      # first engine
O.engines[-1]     # last engine
```

   注意：如果修改 *O.engines* ，索引可能会发生变化。 标记在下文的章节中介绍，是可靠访问特定引擎的更好解决方案。

2. *O.bodies*

   刚体通过其 标识符进行识别，并且在整个模拟过程中保证不会改变：

```
O.bodies[0]                                                    # first body
[b.shape.radius for b in O.bodies if isinstance(b.shape,Sphere)]    # list of radii of␣
→all spherical bodies
sum([b.state.mass for b in O.bodies])                          # sum of masses of␣
→all bodies
numpy.average([b.state.vel[0] for b in O.bodies])              # average velocity in␣
→x direction
```

   注意：无法保证 *Body.id* 的唯一性，因为新创建的刚体可能会重新使用已删除的刚体的标识符。

3. *O.forces*

作用于每个颗粒的广义力（力、力矩）。这些力通常会在每个步开始时由 *ForceResetter* 重置，随后单个相互作用产生的力会在 *InteractionLoop* 中累积。要访问这些数据，请使用：

```
O.forces.f(0)      # force on #0
O.forces.t(1)      # torque on #1
```

4. *O.*相互作用

   相互作用通过 标识符 来识别，这些标识符对应于相互作用的颗粒（它们会在模拟过程中自动创建和删除）：

```
O.interactions[0,1]    # interactions of #0 with #1
O.interactions[1,0]    # the same object
O.bodies[0].intrs()    # all interactions of body #0
for i in O.bodies[12].intrs(): print (i.isReal,i.id1,i.id2)     # get some info about␣
 →interactions of body #12
[(i.isReal,i.id1,i.id2) for i in O.bodies[12].intrs()]          # same thing, but make a␣
 →list
```

标签

引擎 和 函子 可以 被赋予标签，这意味着会自动创建同名的 Python 变量。

```
Yade [1]: O.engines=[
   ...:    NewtonIntegrator(damping=.2,label='newtonCustomLabel')
   ...: ]
   ...:

Yade [2]: newtonCustomLabel.damping=.4

Yade [3]: O.engines[0].damping              # O.engines[0] and newtonCustomLabel are the same␣
 →objects
Out[3]: 0.4

Yade [4]: newtonCustomLabel==O.engines[0]   # O.engines[0] and newtonCustomLabel are the same␣
 →objects
Out[4]: True
```

练习

1. 找出此表达式的含义：

```
max([b.state.vel.norm() for b in O.bodies])
```

2. 运行 重力沉积脚本，在模拟运行几秒后暂停。编写表达式，计算

   1. 动能 $\sum \frac{1}{2} m_i |v_i|^2$2. 平均质量（提示：
   使用 numpy.average）3. 所有颗粒的最大 z
   坐标 4. 刚体 #1 的相互作用数量

全局数据

模拟中全局发生情况的实用度量：

不平衡力最大接触力与最大单刚体力之比；静态性的度量，使用 *unbalancedForce*计算。

孔隙率 孔隙体积与总体积之比；使用 孔隙率 计算。
配位数 每个颗粒的平均相互作用数量， *avgNumInteractions*
应力张量（周期性边界条件） 相互作用中的平均力，使用 *nor-malShearStressTensors* **计算**

织构张量 接触在空间中的分布（尚未实现）；可使用 *plotDi-rections* 进行可视化

能量

可以通过以下方式启用对模拟中所有组件的能量数据（例如重力功、动能、塑性耗散、阻尼耗散）的评估：

```
O.trackEnergy=True
```

随后，可以通过 *O.energy*访问能量值；它是一个字典，可以通过以下方式获取其中的条目： `keys()` ，并通过以下方式获取它们的值： `O.energy[key]`。

Save

PyRunner

要保存刚刚学会访问的数据，我们需要在模拟循环中调用 Python。*PyRunner* 正是为此而创建的；它继承了 周期引擎 的周期控制功能，并将要运行的代码作为文本接收（必须加引号，即放在 `'...'`内），该属性名为 *command*。例如，将以下内容添加到*O.engines* 中，每经过一秒墙上时钟时间就会打印当前步数：

```
O.engines=O.engines+[ PyRunner(command='print(O.iter)',realPeriod=1) ]
```

在*command* 中编写复杂代码很不方便；遇到这种情况，我们可以定义一个将被调用的函数：

```
def myFunction():'''Print step number,
 and pause the simulation is unbalanced force is smaller than 0.05.'''print(O.iter)
if utils.unbalancedForce()<0.05:print('Unbalanced force is smaller than 0.05, pausing.')
O.pause()
```

现在可以将此函数添加到 *O.engines*：

```
O.engines+=[PyRunner(command='myFunction()',iterPeriod=100)]
```

或者，一般来说，可以这样写：

```
O.engines=[
        # ...
        PyRunner(command='myFunction()',iterPeriod=100) # call myFunction every 100 steps
]
```

警告： 如果函数是在 运行中的 Yade 会话（IPython）中声明的，并且 PyRunner 属性*updateGlobals* 设置为 *False*，那么将会发生错误 `NameError: name 'myFunction' is not defined` 除非使用 command 更新 pythonglobals(),

```
globals().update(locals())
```

练习

1. 运行重力沉积模拟，但进行以下修改：

    1. *utils.unbalancedForce* 每 2 秒打印一次。2. 每 1000 步检查一次不平衡力的值

        - 如果小于 0.2，则将阻尼设为 0.8（提示：使用标签）
        - 如果小于 0.1，则暂停模拟

保持历史记录

Yade 提供了用于存储和绘制变量的 绘图 模块（稍后会讨论绘图本身）。我们先导入此模块，并声明要绘制的变量名：

```
from yade import plot
plot.plots={'t':('coordNum','unForce',None,'Ek')}                    # kinetic energy will have␣
↪legend on the right as indicated by None separator.
```

使用 *PyRunner* 和 *plot.addData* 函数即可定期存储数据。此外，让我们启用能量跟踪：

```
O.trackEnergy=True
def addPlotData():
    # this function adds current values to the history of data, under the names specified
plot.addData(t=O.time,Ek=utils.kineticEnergy(),coordNum=utils.avgNumInteractions(),
↪unForce=utils.unbalancedForce())
```

现在可以将此函数添加到 *O.engines*中：

```
O.engines+=[PyRunner(command='addPlotData()',iterPeriod=20)]
```

或者，一般来说，可以这样写：

```
O.engines=[  # ...,
        PyRunner(command='addPlotData()',iterPeriod=20)          # call the addPlotData␣
↪function every 20 iterations
]
```

历史记录存储在 *plot.data*中，可通过变量名访问，例如 `plot.data['Ek']`，还可通过 *plot.saveDataTxt*保存到文本文件（用于在 Yade 外部进行后处理）。

Plot

plot 提供了将使用 *plot.addData* 保存的历史记录绘制成二维图的功能。要绘制的数据通过字典 *plot.plots* 指定

```
plot.plots={'t':('coordNum','unForce',None,'Ek')}
```

所有值的历史记录以用于*plot.addData*的名称给出；字典的键为 x轴的值，而值则是 y 轴上的数据序列；`None` 用于区分左右轴上的数据（它们会分别缩放）。绘图本身通过以下方式创建：

```
plot.plot()          # on the command line, F8 can be used as shorthand
```

绘图窗口打开时，它会定期更新，以便实时查看模拟演化。

能量图

绘制全部能量贡献会比较困难，因为可能无法预先知道所有能量的名称。幸运的是，Yade 提供了一种处理方法。该方法分为两部分：

1. *plot.addData* 会接收当前已定义的所有能量：

   ```
   plot.addData(i=O.iter,total=O.energy.total(),**O.energy)
   ```

   该 *O.energy.total* 函数会将所有能量相加。是用于将字典（请记住，*O.energy* 是一个字典）转换为命名函数参数的特殊 Python 语法，因此以下两个命令是等价的：

   ```
   function(a=3,b=34)              # give arguments as arguments
   function(**{'a':3,'b':34})      # create arguments from dictionary
   ```

2. 要绘制的数据通过一个 函数 指定，该函数给出要绘制的数据名称，而不是直接提供数据名称：

   ```
   plot.plots={'i':['total']+O.energy.keys()}
   ```

   其中， `total` 是我们在上文赋予 `O.energy.total()` 的名称，而 `O.energy.keys()` 将始终返回当前已定义能量的列表。

能量图示例

例如，在 实时 Yade 会话中绘制能量图，比如运行examples/test/triax-basic-without-plots.py ，效果如下：

```
from yade import plotO.trackEnergy=True
O.step()                          # performing a single simulation step is necessary to␣
→populate O.energy.keys()plot.plots={'t':O.energy.keys()+['total']}def addPlotData():
# this function adds current values to the history of data, under the names specified
plot.addData( t=O.time , total=O.energy.total() , **O.energy )
O.engines+=[PyRunner(command='addPlotData()',iterPeriod=20)]
globals().update(locals())        # do this only because this is an example of a live yade␣
→session
```

按 F8 显示绘图窗口，按 F11 显示三维视图，然后按　开始模拟。

使用多个图

也可以绘制多个独立图表，例如：

```
plot.plots={ 't':('total','kinetic') , 't ':['elastPotential','gravWork'] , 't  ':('nonviscDamp
→') }
```

警告： 不同图表中不能声明同名变量。因此，上面使用空格来表示同一变量 `t`。

有一点需要注意：在 运行中的 Yade 会话中启动 examples/test/triax-basic-without-plots.py 后，示例如下：

```
from yade import plot
O.trackEnergy=True
plot.plots={ 't':('total','kinetic') , 't ':['elastPotential','gravWork'] , 't  ':('nonviscDamp
→') }

def addPlotData():
        # assign value to all three: 't', 't ' and 't  ' with single t=... assignment
        plot.addData( t=O.time , total=O.energy.total()   **O.energy )

O.engines+=[PyRunner(command='addPlotData()',iterPeriod=20)]

globals().update(locals())        # do this only because this is an example of a live yade␣
→session

plot.plot(subPlots=False)         # show plots in separate windows

plot.plot(subPlots=True)          # same as pressing F8: close current plot windows and reopen␣
→a single new one
```

按 F8 键 打开绘图窗口，按 F11 显示三维视图，然后按 开始模拟，参见下方的 视频 ：

练习

1. 计算 y 方向的平均动量。2. 运行 重力沉积 脚本，绘制不平衡力和动能。3. 脚本运行时，尝试更改 *NewtonIntegrator.damping* 参数（分别通过 检查器 和命令行进行更改）。这会如何影响不平衡力和动能的演变？4. 思考并写下所有能量源（输入）；也写下所有能量汇（耗散）。5. 模拟 重力沉积 并绘制模拟过程中所有能量的变化。

另请参阅：

大多数 示例 教程中的示例使用 *Yade* 的绘图功能，部分示例还会跟踪 模拟的能量 e
模拟 ·

1.2.4 设置模拟

另请参阅：

示例 重力沉积，固结试验，周期性简单剪切，周期性三轴试验 讨论此处讨论的主题。

参数化研究

模拟的输入参数（例如粒径分布、阻尼、各种接触参数等）会影响结果，但通常并不知道它们之间的解析关系。为了研究这种影响，可以运行一系列仅在少数参数上有所不同的类似模拟，并比较结果。Yade 可以在批处理模式下运行，其中将模拟脚本与参数表合取使用，参数表用于指定

脚本每次运行所用的参数值。批处理模拟以非交互方式运行，即无需用户干预；因此，用户必须明确启动和停止模拟。
假设我们想研究阻尼对动能演变的影响。脚本需要在几个地方进行调整：

1. 我们必须确保脚本从参数表中读取相关参数。这通过*utils.readParamsFromTable*实现；读取的参数会作为变量创建在`yade.params.table` 模块中：

```
readParamsFromTable(damping=.2)       # yade.params.table.damping variable will be created
from yade.params import table              # typing table.damping is easier than yade.
→params.table.damping
```

   请注意，*utils.readParamsFromTable* 会为其参数指定默认值；如果脚本未以非批处理模式运行，则会使用这些默认值。
2. 在适当的位置使用表格中的参数：

```
NewtonIntegrator(damping=table.damping),
```

3. 模拟以非交互方式运行；因此，我们必须指定它应在何处停止：

```
O.engines+=[PyRunner(iterPeriod=1000,command='checkUnbalancedForce()')]     # call our␣
→function defined below periodically

def checkUnbalancedForce():
   if unbalancedForce<0.05:                                 # exit Yade if unbalanced force␣
→drops below 0.05
      plot.saveDataTxt(O.tags['d.id']+'.data.bz2')      # save all data into a unique file␣
→before exiting
      import sys
      sys.exit(0)                                          # exit the program
```

4. 最后，我们必 须在脚本的最后启动模拟：

本：

```
O.run()              # run forever, until stopped by checkUnbalancedForce()
waitIfBatch()        # do not finish the script until the simulation ends; does nothing␣
→in non-batch mode
```

参数表 是一个简单的文本文件（例如 `params.txt` ），其中每一行都指定一个要运行的模拟：

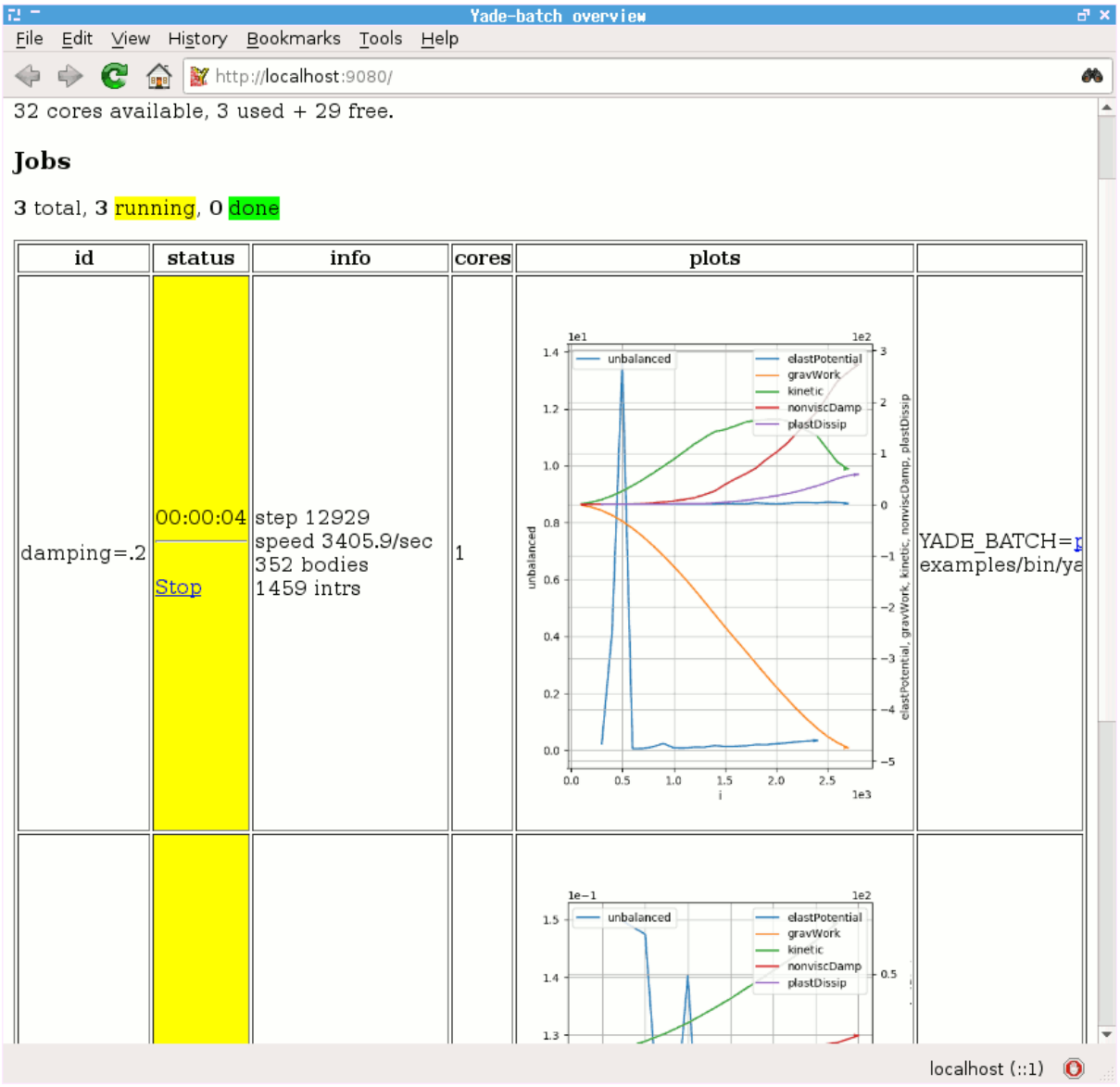
```
# comments start with # as in python
damping      # first non-comment line is variable name
.2
.4
.6
```

最后，使用特殊的批处理命令运行模拟：

```
user@machine:~$ yade-batch params.txt simulation.py
```

练习

1. 运行重力沉积脚本的批处理模式，并将阻尼设为`.2`、`.4`、`.6`。2. 在批处理运行期间查看 http://localhost:9080 概览页面（图*imgBatchExample*）。



边界

在无穷空间中运动的颗粒通常需要一些约束，才能使模拟具有意义。

支撑体

到目前为止，支撑体（不可移动的颗粒）一直提供着必要的边界：在重力沉积脚本中，*geom.facetBox* 由以下元素在内部构成：面片（三角剖分元素），它们 `fixed` 分布于空间中；面片也用于任意三角剖分曲面（参见用户手册中的相关章节）。另一种常用的边界是*utils.wall*（无限轴对齐平面）。

周期性

周期性边界是通过使用周期性（而非无限）空间创建的“边界”。该边界由 *O.periodic=True* 激活，空间配置由 *O.cell* 描述。它非常适合研究块体材料行为，因为它避免了边界效应，从而减少了所需的颗粒数量。另一方面，它可能不适合研究局部化，因为任何单元级效应（例如剪切带）也必须满足周期性。

周期单元由其 参考尺寸 （与全局坐标轴对齐的盒子的尺寸）以及 当前变换来描述；该变换可以表示拉伸、剪切和旋转。通过 速度梯度指定变形，该梯度会在下一步之前更新 变换。 相似变形可以使 速度梯度在整个单元内扩散，从而使边界融入整个单元。

可以使用 *PeriTriaxController*控制应力和应变；还可以通过 目标值状态，使用 *PeriTriaxController.stressMask*。
以下代码会创建周期性球体云并进行压缩，以达到 σ_x=-10 千帕， σ_y=-10 千帕和ε_z=-0.1。由于应力是针对 y 和 z 指定的，应力掩码是二进制数 0b011 （x→1、y→2、z→4，以十进制表示1+2=3）。

```
Yade [1]: sp=pack.SpherePack()Yade [2]: sp.makeCloud((1,1,1),(2,2,2),rMean=.16,periodic=True)
Out[2]: 20Yade [3]: sp.toSimulation()              # implicitly sets O.periodic=True,
 and O.cell.refSize→to the packing period size
Out[3]: [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]
Yade [4]: O.engines+=[PeriTriaxController(goal=(-1e4,-1e4,-.1),stressMask=0b011,maxUnbalanced=.
→2, doneHook='functionToRunWhenFinished()')]
```

当模拟运行时， *PeriTriaxController* 会接管控制，并在完成钩子于目标值达到时调用。使用 PeriTriaxController 的完整模拟可能如下所示：

```
from __future__ import print_functionfrom yade import pack, plotsp = pack.SpherePack()
rMean = .05sp.makeCloud((0, 0, 0), (1, 1, 1), rMean=rMean, periodic=True)sp.toSimulation()
O.engines = [ForceResetter(),
InsertionSortCollider([Bo1_Sphere_Aabb()], verletDist=.05 * rMean),
InteractionLoop([Ig2_Sphere_Sphere_ScGeom()], [Ip2_FrictMat_FrictMat_FrictPhys()],␣
→[Law2_ScGeom_FrictPhys_CundallStrack()]),NewtonIntegrator(damping=.6),PeriTriaxController(
goal=(-1e6, -1e6, -.1), stressMask=0b011, maxUnbalanced=.2, doneHook=→'goalReached()',
 label='triax', maxStrainRate=(.1, .1, .1), dynCell=True),
PyRunner(iterPeriod=100, command='addPlotData()')]O.dt = .5 * utils.PWaveTimeStep()
O.trackEnergy = Truedef goalReached():
print('Goal reached, strain', triax.strain, ' stress', triax.stress)O.pause()
def addPlotData():plot.addData(sx=triax.stress[0],sy=triax.stress[1],sz=triax.stress[2],
ex=triax.strain[0],ey=triax.strain[1],ez=triax.strain[2],i=O.iter,
```

（下页继续）

（接上一页）

```
                unbalanced=utils.unbalancedForce(),
                totalEnergy=O.energy.total(),
                **O.energy  # plot all energies
        )


plot.plots = {
        'i': (('unbalanced', 'go'), None, 'kinetic'),
        ' i': ('ex', 'ey', 'ez', None, 'sx', 'sy', 'sz'),
        'i ': (O.energy.keys, None, ('totalEnergy', 'bo'))
}
plot.plot()
O.saveTmp()
O.run()
```

1.2.5 进阶与更多

粒径分布

参见 周期性三轴试验 和examples/test/psd.py

团簇

团簇；参见 周期性三轴试验

测试定律

LawTester, scripts/checks-and-tests/law-test.py

新定律

可视化

参见示例 三维后处理和视频录制

- *VTKRecorder* &ParaView
- *makeVideo*
- *SnapshotEngine*
- doc/sphinx/tutorial/05-3d-postprocessing.py
- examples/test/force-network-video.py
- doc/sphinx/tutorial/make-simulation-video.py

将 Python 2 脚本转换为 Python 3

以下是将脚本转换为 Python 3 时常见操作的不完全列表 .

必须：

- `print ...` 变为 `print(...)`,
- `myDict.iterkeys()`, `myDict.itervalues()`, `myDict.iteritems()` 变为 `myDict.keys()`, `myDict.values()`, `myDict.items()`,
- `import cPickle` 变为 `import pickle`,
- “ 和 `<>` 运算符不再被识别,
- 禁止在缩进中混用制表符和空格，因此 Yade 中的所有脚本都使用制表符缩进。

应进行检查，但并非总是强制要求：

- （欧几里得）两个整数相除： `i1/i2` 变为 `i1//i2`,
- `myDict.keys()`、 `myDict.values()`、 `myDict.items()` 有时会变为 `list(myDict.keys())`、`list(myDict.values())`、 `list(myDict.items())` （取决于你的用法）,
- `map()`、 `filter()`、 `zip()` 有时会变为 `list(map())`、 `list(filter())`、 `list(zip())` （取决于你的用法）,
- 字符串编码现在到处都采用 UTF-8，这可能会导致包含特殊字符的用户输入/输出（键盘、文件.....）出现问题。

可选：

- `# encoding: utf-8` 不再需要

1.2.6 教程示例

此教程的在线版本包含嵌入式视频。

弹跳球

以下示例位于文件 doc/sphinx/tutorial/01-bouncing-sphere.py。

```
# basic simulation showing sphere falling ball gravity,
# bouncing against another sphere representing the support

# DATA COMPONENTS

# add 2 particles to the simulation
# they the default material (utils.defaultMat)
O.bodies.append(
        [
                # fixed: particle's position in space will not change (support)
                sphere(center=(0, 0, 0),  radius=.5,  fixed=True),
                # this particles is free, subject to dynamics
                sphere((0, 0, 2), .5)
        ]
)

# FUNCTIONAL COMPONENTS

# simulation loop -- see presentation for the explanation
O.engines = [
```

（下页继续）

（接上页）

```
        ForceResetter(),
        InsertionSortCollider([Bo1_Sphere_Aabb()]),
        InteractionLoop(
                [Ig2_Sphere_Sphere_ScGeom()],    # collision geometry
                [Ip2_FrictMat_FrictMat_FrictPhys()],    # collision "physics"
                [Law2_ScGeom_FrictPhys_CundallStrack()]  # contact law -- apply forces
        ),
        # Apply gravity force to particles. damping: numerical dissipation of energy.
        NewtonIntegrator(gravity=(0, 0, -9.81), damping=0.1)
]

# set timestep to a fraction of the critical timestep
# the fraction is very small,  so that the simulation is not too fast
# and the motion can be observed
O.dt = .5e-4 * PWaveTimeStep()

# save the simulation, so that it can be reloaded later, for experimentation
O.saveTmp()
```

重力沉积

以下示例为 i 文件 doc/sphinx/tutorial/02-gravity-depositio n.py。

```
# gravity deposition in box, showing how to plot and save history of data,
# and how to control the simulation while it is running by calling
# python functions from within the simulation loop

# import yade modules that we will use below
from yade import pack, plot

# create rectangular box from facets
O.bodies.append(geom.facetBox((.5, .5, .5), (.5, .5, .5), wallMask=31))

# create empty sphere packing
# sphere packing is not equivalent to particles in simulation, it contains only the pure␣
 →geometry
sp = pack.SpherePack()
# generate randomly spheres with uniform radius distribution
sp.makeCloud((0, 0, 0), (1, 1, 1), rMean=.05, rRelFuzz=.5)
# add the sphere pack to the simulation
sp.toSimulation()

O.engines = [
        ForceResetter(),
        InsertionSortCollider([Bo1_Sphere_Aabb(), Bo1_Facet_Aabb()]),
        InteractionLoop(
                # handle sphere+sphere and facet+sphere collisions
                [Ig2_Sphere_Sphere_ScGeom(), Ig2_Facet_Sphere_ScGeom()],
                [Ip2_FrictMat_FrictMat_FrictPhys()],
                [Law2_ScGeom_FrictPhys_CundallStrack()]
        ),
        NewtonIntegrator(gravity=(0, 0, -9.81), damping=0.4),
        # call the checkUnbalanced function (defined below) every 2 seconds
        PyRunner(command='checkUnbalanced()', realPeriod=2),
        # call the addPlotData function every 200 steps
        PyRunner(command='addPlotData()', iterPeriod=100)
]
O.dt = .5 * PWaveTimeStep()

# enable energy tracking; any simulation parts supporting it
```

（下页继续）

（接上一页）

```
# can create and update arbitrary energy types,  which can be
# accessed as O.energy['energyName'] subsequently
O.trackEnergy = True


# if the unbalanced forces goes below .05, the packing
# is considered stabilized, therefore we stop collected
# data history and stop
def checkUnbalanced():
        if unbalancedForce() < .05:
                O.pause()
                plot.saveDataTxt('bbb.txt.bz2')
                # plot.saveGnuplot('bbb') is also possible


# collect history of data which will be plotted
def addPlotData():
        # each item is given a names, by which it can be the unsed in plot.plots
        # the **O.energy converts dictionary-like O.energy to plot.addData arguments
        plot.addData(i=O.iter, unbalanced=unbalancedForce(), **O.energy)


# define how to plot data: 'i' (step number) on the x-axis, unbalanced force
# on the left y-axis, all energies on the right y-axis
# (O.energy.keys is function which will be called to get all defined energies)
# None separates left and right y-axis
plot.plots = {'i': ('unbalanced', None, O.energy.keys)}

# show the plot on the screen, and update while the simulation runs
plot.plot()

O.saveTmp()
```

固结试验

以下示例位于文件doc/sphinx/tutorial/03-oedometric-test.py。

```
# gravity deposition,  continuing with oedometric test after stabilization
# shows also how to run parametric studies with yade-batch
# The components of the batch are:# 1. table with parameters,
 one set of parameters per line (ccc.table)
# 2. readParamsFromTable which reads respective line from the parameter file
# 3. the simulation muse be run using yade-batch,  not yade#
# $ yade-batch --job-threads=1 03-oedometric-test.table 03-oedometric-test.py
## load parameters from file if run in batch
# default values are used if not run from batch
readParamsFromTable(rMean=.05, rRelFuzz=.3, maxLoad=1e6, minLoad=1e4)
# make rMean,  rRelFuzz,  maxLoad accessible directly as variables later
from yade.params.table import *# create box with free top,
 and ceate loose packing inside the boxfrom yade import pack,  plot
O.bodies.append(geom.facetBox((.5, .5, .5), (.5, .5, .5), wallMask=31))
sp = pack.SpherePack()
sp.makeCloud((0, 0, 0), (1, 1, 1), rMean=rMean, rRelFuzz=rRelFuzz)
sp.toSimulation()
```

（下页续）

（接上一页）

```
O.engines = [
        ForceResetter(),
        # sphere, facet, wall
        InsertionSortCollider([Bo1_Sphere_Aabb(), Bo1_Facet_Aabb(), Bo1_Wall_Aabb()]),
        InteractionLoop(
                # the loading plate is a wall, we need to handle sphere+sphere, sphere+facet,␣
→sphere+wall
                [Ig2_Sphere_Sphere_ScGeom(), Ig2_Facet_Sphere_ScGeom(), Ig2_Wall_Sphere_
→ScGeom()],
                [Ip2_FrictMat_FrictMat_FrictPhys()],
                [Law2_ScGeom_FrictPhys_CundallStrack()]
        ),
        NewtonIntegrator(gravity=(0, 0, -9.81), damping=0.5),
        # the label creates an automatic variable referring to this engine
        # we use it below to change its attributes from the functions called
        PyRunner(command='checkUnbalanced()', realPeriod=2, label='checker'),
]
O.dt = .5 * PWaveTimeStep()

# the following checkUnbalanced, unloadPlate and stopUnloading functions are all called by the
→'checker'
# (the last engine) one after another; this sequence defines progression of different stages␣
→of the
# simulation, as each of the functions, when the condition is satisfied, updates  'checker' to␣
→call
# the next function when it is run from within the simulation next time


# check whether the gravity deposition has already finished
# if so, add wall on the top of the packing and start the oedometric test
def checkUnbalanced():
        # at the very start, unbalanced force can be low as there is only few contacts,  but it␣
→does not mean the packing is stable
        if O.iter < 5000:
                return
        # the rest will be run only if unbalanced is < .1 (stabilized packing)
        if unbalancedForce() > .1:
                return
        # add plate at the position on the top of the packing
        # the maximum finds the z-coordinate of the top of the topmost particle
        O.bodies.append(wall(max([b.state.pos[2] + b.shape.radius for b in O.bodies if␣
→isinstance(b.shape, Sphere)])<style 标识符='2'>, axis=2<style 标识符='5'>, sense=-1))
        global plate  # without this line, the plate variable would only exist inside this␣
→function
        plate = O.bodies[-1]  # the last particles is the plate
        # Wall objects are "fixed" by default, i.e. not subject to forces
        # prescribing a velocity will therefore make it move at constant velocity (downwards)
        plate.state.vel = (0, 0, -.1)
        # start plotting the data now, it was not interesting before
        O.engines = O.engines + [PyRunner(command='addPlotData()', iterPeriod=200)]
        # next time, do not call this function anymore, but the next one (unloadPlate) instead
        checker.command = 'unloadPlate()'


def unloadPlate():
        # if the force on plate exceeds maximum load,  start unloading
        if abs(O.forces.f(plate.id)[2]) > maxLoad:
                plate.state.vel *= -1
                # next time, do not call this function anymore, but the next one␣
→(stopUnloading) instead
```

（续见下页）

（续前页）

```
                checker.command = 'stopUnloading()'


def stopUnloading():
        if abs(O.forces.f(plate.id)[2]) < minLoad:
                # O.tags can be used to retrieve unique identifiers of the simulation
                # if running in batch, subsequent simulation would overwrite each other's␣
 →output files otherwise
                # d (or description) is simulation description (composed of parameter values)
                # while the id is composed of time and process number
                plot.saveDataTxt(O.tags['d.id'] + '.txt')
                O.pause()


def addPlotData():
        if not isinstance(O.bodies[-1].shape, Wall):
                plot.addData()
                return
        Fz = O.forces.f(plate.id)[2]
        plot.addData(Fz=Fz, w=plate.state.pos[2] - plate.state.refPos[2],␣
 →unbalanced=unbalancedForce(), i=O.iter)


# besides unbalanced force evolution, also plot the displacement-force diagram
plot.plots = {'i': ('unbalanced',),  'w': ('Fz',)}
plot.plot()

O.run()
# when running with yade-batch, the script must not finish until the simulation is done fully
# this command will wait for that (has no influence in the non-batch mode)
waitIfBatch()
```

批处理表

要以批处理模式运行同一个脚本 doc/sphinx/tutorial/03-oedometric-test.py 以测试不同参数，请执行 command `yade-batch 03-oedometric-test.table 03-oedometric-test.py`，还可以访问页面 http://localhost:9080 查看批处理仿真进度。

```
rMean rRelFuzz maxLoad
.05 .1 1e6
.05 .2 1e6
.05 .3 1e6
```

周期性简单剪切

以下示例位于文件 doc/sphinx/tutorial/04-periodic-simple-shear.py中。

```
# encoding: utf-8

# script for periodic simple shear test, with periodic boundary
# first compresses to attain some isotropic stress (checkStress),
# then loads in shear (checkDistorsion)
#
# the initial packing is either regular (hexagonal), with empty bands along the boundary,
# or periodic random cloud of spheres
#
# material friction angle is initially set to zero, so that the resulting packing is dense
```

（续见下页）

（接上一页）

```
# (sphere rearrangement is easier if there is no friction)
#

# setup the periodic boundary
from __future__ import print_function
O.periodic = True
O.cell.hSize = Matrix3(2, 0, 0, 0, 2, 0, 0, 0, 2)

from yade import pack, plot

# the "if 0:" block will be never executed, therefore the "else:" block will be
# to use cloud instead of regular packing, change to "if 1:" or something similar
if 0:
        # create cloud of spheres and insert them into the simulation
        # we give corners, mean radius, radius variation
        sp = pack.SpherePack()
        sp.makeCloud((0, 0, 0), (2, 2, 2), rMean=.1, rRelFuzz=.6, periodic=True)
        # insert the packing into the simulation
        sp.toSimulation(color=(0, 0, 1))  # pure blue
else:
        # in this case, add dense packing
        O.bodies.append(pack.regularHexa(pack.inAlignedBox((0, 0, 0), (2, 2, 2)), radius=.1,␣
→gap=0,  color=(0, 0, 1)))

# create "dense" packing by setting friction to zero initially
O.materials[0].frictionAngle = 0

# simulation loop (will be run at every step)
O.engines = [
        ForceResetter(),
        InsertionSortCollider([Bo1_Sphere_Aabb()]),
        InteractionLoop(
                # interaction loop
                [Ig2_Sphere_Sphere_ScGeom()],
                [Ip2_FrictMat_FrictMat_FrictPhys()],
                [Law2_ScGeom_FrictPhys_CundallStrack()]
        ),
        NewtonIntegrator(damping=.4),
        # run checkStress function (defined below) every second
        # the label is arbitrary,  and is used later to refer to this engine
        PyRunner(command='checkStress()', realPeriod=1, label='checker'),
        # record data for plotting every 100 steps; addData function is defined below
        PyRunner(command='addData()', iterPeriod=100)
]

# set the integration timestep to be 1/2 of the "critical" timestep
O.dt = .5 * PWaveTimeStep()

# prescribe isotropic normal deformation (constant strain rate)
# of the periodic cell
O.cell.velGrad = Matrix3(-.1, 0, 0, 0, -.1, 0, 0, 0, -.1)

# when to stop the isotropic compression (used inside checkStress)
limitMeanStress = -5e5


# called every second by the PyRunner engine
def checkStress():
        # stress tensor as the sum of normal and shear contributions
        # Matrix3.Zero is the intial value for sum(...)
        stress = getStress().trace() / 3.
```

（续见下页）

（接上一页）

```
        print('mean stress', stress)
        # if mean stress is below (bigger in absolute value) limitMeanStress, start shearing
        if stress < limitMeanStress:
                # apply constant-rate distorsion on the periodic cell
                O.cell.velGrad = Matrix3(0, 0, .1, 0, 0, 0, 0, 0, 0)
                # change the function called by the checker engine
                # (checkStress will not be called anymore)
                checker.command = 'checkDistorsion()'
                # block rotations of particles to increase tanPhi, if desired
                # disabled by default
                if 0:
                        for b in O.bodies:
                                # block X,Y,Z rotations, translations are free
                                b.state.blockedDOFs = 'XYZ'
                                # stop rotations if any, as blockedDOFs block accelerations␣
→really
                                b.state.angVel = (0, 0, 0)
                # set friction angle back to non-zero value
                # tangensOfFrictionAngle is computed by the Ip2_* functor from material
                # for future contacts change material (there is only one material for all␣
→particles)
                O.materials[0].frictionAngle = .5  # radians
                # for existing contacts, set contact friction directly
                for i in O.interactions:
                        i.phys.tangensOfFrictionAngle = tan(.5)


# called from the 'checker' engine periodically, during the shear phase
def checkDistorsion():
        # if the distorsion value is >.3, exit; otherwise do nothing
        if abs(O.cell.trsf[0, 2]) > .5:
                # save data from addData(...) before exiting into file
                # use O.tags['id'] to distinguish individual runs of the same simulation
                plot.saveDataTxt(O.tags['id'] + '.txt')
                # exit the program
                #import sys
                #sys.exit(0) # no error (0)
                O.pause()


# called periodically to store data history
def addData():
        # get the stress tensor (as 3x3 matrix)
        stress = sum(normalShearStressTensors(), Matrix3.Zero)
        # give names to values we are interested in and save them
        plot.addData(exz=O.cell.trsf[0, 2], szz=stress[2, 2], sxz=stress[0, 2],␣
→tanPhi=(stress[0, 2] / stress[2, 2]) if stress[2, 2] != 0 else 0, i=O.iter)
        # color particles based on rotation amount
        for b in O.bodies:
                # rot() gives rotation vector between reference and current position
                b.shape.color = scalarOnColorScale(b.state.rot().norm(), 0, pi / 2.)


# define what to plot (3 plots in total)
## exz(i), [left y axis, separate by None:] szz(i), sxz(i)
## szz(exz), sxz(exz)
## tanPhi(i)
# note the space in 'i ' so that it does not overwrite the 'i' entry
plot.plots = {'i': ('exz', None, 'szz', 'sxz'), 'exz': ('szz', 'sxz'), 'i ': ('tanPhi',)}

# better show rotation of particles
```

（续见下页）

（接上页）

```
Gl1_Sphere.stripes = True

# open the plot on the screen
plot.plot()

O.saveTmp()
```

三维后处理

以下示例位于文件 doc/sphinx/tutorial/05-3d-postprocessing.py 中。此示例将运行 20000年 次迭代，并保存 `*.png` 快照，然后根据这些快照制作视频 `3d.mpeg` 。

```
# demonstrate 3d postprocessing with yade
#
# 1. qt.SnapshotEngine saves images of the 3d view as it appears on the screen periodically
#    makeVideo is then used to make real movie from those images
# 2. VTKRecorder saves data in files which can be opened with Paraview
#    see the User's manual for an intro to Paraview

# generate loose packing
from yade import pack, qt
sp = pack.SpherePack()
sp.makeCloud((0, 0, 0), (2, 2, 2), rMean=.1, rRelFuzz=.6, periodic=True)
# add to scene, make it periodic
sp.toSimulation()

O.engines = [
        ForceResetter(),
        InsertionSortCollider([Bo1_Sphere_Aabb()]),
        InteractionLoop(
                # interaction loop
                [Ig2_Sphere_Sphere_ScGeom()],
                [Ip2_FrictMat_FrictMat_FrictPhys()],
                [Law2_ScGeom_FrictPhys_CundallStrack()]
        ),
        NewtonIntegrator(damping=.4),
        # save data for Paraview
        VTKRecorder(fileName='3d-vtk-', recorders=['all'], iterPeriod=1000),
        # save data from Yade's own 3d view
        qt.SnapshotEngine(fileBase='3d-', iterPeriod=200, label='snapshot'),
        # this engine will be called after 20000 steps,  only once
        PyRunner(command='finish()', iterPeriod=20000)
]
O.dt = .5 * PWaveTimeStep()

# prescribe constant-strain deformation of the cell
O.cell.velGrad = Matrix3(-.1, 0, 0, 0, -.1, 0, 0, 0, -.1)

# we must open the view explicitly (limitation of the qt.SnapshotEngine)
qt.View()


# this function is called when the simulation is finished
def finish():
        # snapshot is label of qt.SnapshotEngine
        # the 'snapshots' attribute contains list of all saved files
        makeVideo(snapshot.snapshots, '3d.mpeg',  fps=10,  bps=10000)
        O.pause()
```

（续见下页）

（接上页）

```
# set parameters of the renderer, to show network chains rather than particles
# these settings are accessible from the Controller window, on the second tab ("Display") as␣
→well
rr = yade.qt.Renderer()
rr.shape = False
rr.intrPhys = True
```

周期性三轴试验

以下示例位于文件 doc/sphinx/tutorial/06-periodic-triaxial-test.py。

```
# encoding: utf-8

# periodic triaxial test simulation
#
# The initial packing is either
#
# 1. random cloud with uniform distribution, or
# 2. cloud with specified granulometry (radii and percentages), or
# 3. cloud of clumps, i.e. rigid aggregates of several particles
#
# The triaxial consists of 2 stages:
#
# 1. isotropic compaction, until sigmaIso is reached in all directions;
#    this stage is ended by calling compactionFinished()
# 2. constant-strain deformation along the z-axis, while maintaining
#    constant stress (sigmaIso) laterally; this stage is ended by calling
#    triaxFinished()
#
# Controlling of strain and stresses is performed via PeriTriaxController,
# of which parameters determine type of control and also stability
# condition (maxUnbalanced) so that the packing is considered stabilized
# and the stage is done.
#

from __future__ import print_function
sigmaIso = -1e5

#import matplotlib
#matplotlib.use('Agg')

# generate loose packing
from yade import pack, qt, plot

O.periodic = True
sp = pack.SpherePack()
if 0:
        ## uniform distribution
        sp.makeCloud((0, 0, 0), (2, 2, 2), rMean=.1, rRelFuzz=.3, periodic=True)
else:
        ## create packing from clumps
        # configuration of one clump
        c1 = pack.SpherePack([((0, 0, 0), .03333), ((.03, 0, 0), .017), ((0, .03, 0), .017)])
        # make cloud using the configuration c1 (there could c2, c3, ...; selection between␣
→them would be random)
        sp.makeClumpCloud((0, 0, 0), (2, 2, 2), [c1]<style 标识符='2'>, periodic=True<style 标识
符='5'>, num=500)

# setup periodic boundary, insert the packing
sp.toSimulation()
```

（下页续）

（接上一页）

```
O.engines = [
        ForceResetter(),
        InsertionSortCollider([Bo1_Sphere_Aabb()]),
        InteractionLoop([Ig2_Sphere_Sphere_ScGeom()], [Ip2_FrictMat_FrictMat_FrictPhys()],␣
→[Law2_ScGeom_FrictPhys_CundallStrack()]),
        PeriTriaxController(
                label='triax',
                # specify target values and whether they are strains or stresses
                goal=(sigmaIso, sigmaIso, sigmaIso),
                stressMask=7,
                # type of servo-control
                dynCell=True,
                maxStrainRate=(10, 10, 10),
                # wait until the unbalanced force goes below this value
                maxUnbalanced=.1,
                relStressTol=1e-3,
                # call this function when goal is reached and the packing is stable
                doneHook='compactionFinished()'
        ),
        NewtonIntegrator(damping=.2),
        PyRunner(command='addPlotData()', iterPeriod=100),
]
O.dt = .5 * PWaveTimeStep()


def addPlotData():
        plot.addData(
                unbalanced=unbalancedForce(),
                i=O.iter,
                sxx=triax.stress[0],
                syy=triax.stress[1],
                szz=triax.stress[2],
                exx=triax.strain[0],
                eyy=triax.strain[1],
                ezz=triax.strain[2],
                # save all available energy data
                Etot=O.energy.total(),
                **O.energy
        )


# enable energy tracking in the code
O.trackEnergy = True

# define what to plot
plot.plots = {
        'i': ('unbalanced',),
        'i ': ('sxx', 'syy', 'szz'),
        ' i': ('exx', 'eyy', 'ezz'),
        # energy plot
        ' i ': (O.energy.keys, None, 'Etot'),
}
# show the plot
plot.plot()


def compactionFinished():
        # set the current cell configuration to be the reference one
        O.cell.trsf = Matrix3.Identity
        # change control type: keep constant confinement in x,y, 20% compression in z
```

（下页续）

（接上一页）

```
        triax.goal = (sigmaIso, sigmaIso, -.2)
        triax.stressMask = 3
        # allow faster deformation along x,y to better maintain stresses
        triax.maxStrainRate = (1., 1., .1)
        # next time, call triaxFinished instead of compactionFinished
        triax.doneHook = 'triaxFinished()'
        # do not wait for stabilization before calling triaxFinished
        triax.maxUnbalanced = 10


def triaxFinished():
        print('Finished')
        O.pause()
```

1.2.7 更多示例

同一列表还提供嵌入式视频版本，可在线查看，但网速较慢的互联网连接不建议观看。

完整的示例列表见文件 examples/list_中的_examples.txt。以下列出了其中一些示例的视频。

FluidCouplingLBM

- *refFastBuoyancy*，源文件，视频。

FluidCouplingPFV

- *refFastOedometer*，源文件，视频。

HydroForceEngine

- *refFastBuoyantParticles*，源文件，视频。
- *refFastFluidizedBed*，源文件，视频。
- *refFastSedimentTransportExample*，源文件，视频。
- *refFastLaminarShearFlow*，源文件，视频。
- *refFastPostProcessValidMaurin2015*，源文件，视频。
- *refFastValidMaurin2015*，源文件，视频。

周期性边界

- *refFastCellFlipping*，源文件，视频。
- *refFastPeri3dController-example1*，源文件，视频。
- *refFastPeri3dController-shear*，源文件，视频。
- *refFastPeri3dController-triaxialCompression*，源文件，视频。
- *refFastPeriodic-compress*，源文件，视频。
- *refFastPeriodic-shear*，源文件，视频。
- *refFastPeriodic-simple-shear*，源文件，视频。

- *refFastPeriodic-simple*，源文件，视频。
- *refFastPeriodic-triax-settingHsize*，源文件，视频。
- *refFastPeriodic-triax*，源文件，视频。
- *refFastPeriodicSandPile*，源文件，视频。

PotentialBlocks

- *refFastWedgeYADE*，源文件，视频。
- *refFastCubePBscaled*，源文件，视频。

PotentialParticles

- *refFastCubePPscaled*，源文件，视频。

WireMatPM

- *refFastWirecontacttest*，源文件，视频。
- *refFastWirepackings*，源文件，视频。
- *refFastWiretensiltest*，源文件，视频。

自适应积分器

- *refFastSimple-scene-plot-NewtonIntegrator*，源文件，视频。
- *refFastSimple-scene-plot-RungeKuttaCashKarp54*，源文件，视频。

团聚体

- *refFastCompress*，源文件，视频。
- *refFastSimulation*，源文件，视频。

Baraban

- *refFastBicyclePedalEngine*，源文件，视频。
- *refFastBaraban*，源文件，视频。
- *refFastRotating-cylinder*，源文件，视频。

推土机

- *refFastBulldozer*，源文件，视频。

毛细管

- *refFastCapillar*，源文件，视频。

毛细拉普拉斯-杨

- *refFastCapillaryPhys* 示例，源文件，视频。
- *refFastCapillaryBridge*，源文件，视频。

链式圆柱体

- *refFastCohesiveCylinderSphere*，源文件，视频。
- *refFastChained-cylinder-roots*，源文件，视频。
- *refFastChained-cylinder-spring*，源文件，视频。

团簇

- *refFastAddToClump* 示例，源文件，视频。
- *refFastApply-buoyancy-clumps*，源文件，视频。
- *refFastClump-hopper-test*，源文件，视频。
- *refFastClump-hopper-viscoelastic*，源文件，视频。
- *refFastClump-inbox-viscoelastic*，源文件，视频。
- *refFastClump-viscoelastic*，源文件，视频。
- *refFastReleaseFromClump-example*，源文件，视频。
- *refFastReplaceByClumps-example*，源文件，视频。
- *refFastTriax-basic-with-clumps*，源文件，视频。

混凝土

- *refFastBrazilian*，源文件，视频。
- *refFastInteraction-histogram*，源文件，视频。
- *refFastPeriodic*，源文件，视频。
- *refFastTriax*，源文件，视频。
- *refFastUniax-post*，源文件，视频。
- *refFastUniax*，源文件，视频。

输送机

- *refFastConveyor*，源文件，视频。

圆柱体

- *refFastBendingbeams*，源文件，视频。
- *refFastCylinder-cylinder*，源文件，视频。
- *refFastCylinderconnection-roots*，源文件，视频。
- *refFastMikado*，源文件，视频。

可变形元件

- *refFastMinimalTensileTest*，源文件，视频。
- *refFastTestDeformableBodies*，源文件，视频。
- *refFastTestDeformableBodies-pressure*，源文件，视频。

网格

- *refFastCohesiveGridConnectionSphere*，源文件，视频。
- *refFastGridConnection-Spring*，源文件，视频。
- *refFastSimple-GridConnection-Falling*，源文件，视频。
- *refFastSimple-Grid-Falling*，源文件，视频。

Gts-horse

- *refFastGts-horse*，源文件，视频。
- *refFastGts-operators*，源文件，视频。
- *refFastGts-random-pack-obb*，源文件，视频。
- *refFastGts-random-pack*，源文件，视频。

沙漏

- *refFastHourglass*，源文件，视频。

包

- *refFastPacks*，源文件，视频。

PFacet

- *refFastGts-pfacet*，源文件，视频。
- *refFastMesh-pfacet*，源文件，视频。
- *refFastPFacets-grids-spheres-interacting*，源文件，视频。
- *refFastPfacetcreators*，源文件，视频。

多面体

- *refFastBall*，源文件，视频。
- *refFastHorse*，源文件，视频。
- *refFastIrregular*，源文件，视频。
- *refFastSphere-interaction*，源文件，视频。
- *refFastSplitter*，源文件，视频。
- *refFastInteractinDetectionFactor*，源文件，视频。
- *refFastScGeom*，源文件，视频。
- *refFastTextExport*，源文件，视频。

多面体断裂

- 单轴压缩，源文件，视频。

二维环

- *Cundall* 阻尼环,源文件,视频.
- 简单黏弹性环,源文件, 视频.

杆穿透

- 模型，源文件，视频。

简单场景

- 双球法向黏性，源文件，视频。
- 保存后重新加载，源文件，视频。
- 简单场景默认引擎，源文件，视频。
- 简单场景能量跟踪，源文件，视频。
- *refFastSimple-scene-plot*，源文件，视频。
- *refFastSimple-scene*，源文件，视频。

Stl-gts

- *refFastGts-stl*，源文件，视频。

Tesselation封装器

- *refFastTesselationWrapper*，源文件，视频。

Test

- *refFastNet-2part-displ-unloading*，源文件，视频。
- *refFastNet-2part-displ*，源文件，视频。
- *refFastBeam-l6geom*，源文件，视频。
- *refFastClump-facet*，源文件，视频。
- *refFastClumpPack*，源文件，视频。
- *refFastCollider-stride-triax*，源文件，视频。
- *refFastCollider-stride*，源文件，视频。
- *refFastCombined-kinematic-engine*，源文件，视频。
- *refFastEnergy*，源文件，视频。
- *refFastFacet-box*，源文件，视频。
- *refFastFacet-sphere-ViscElBasic-peri*，源文件，视频。
- *refFastFacet-sphere-ViscElBasic*，源文件，视频。
- *refFastFacet-sphere*，源文件，视频。

- *refFastHelix*，源文件，视频。
- *refFastInterpolating-force*，源文件，视频。
- *refFastKinematic*，源文件，视频。
- *refFastMindlin*，源文件，视频。
- *refFastMulti*，源文件，视频。
- *refFastPack-cloud*，源文件，视频。
- *refFastPack-inConvexPolyhedron*，源文件，视频。
- *refFastPv-section*，源文件，视频。
- *refFastPeriodic-geom-compare*，源文件，视频。
- *refFastPsd*，源文件，视频。
- *refFastSphere-sphere-ViscElBasic-peri*，源文件，视频。
- *refFastSubdomain-balancer*，源文件，视频。
- *refFastTest-sphere-facet-corner*，源文件，视频。
- *refFastTest-sphere-facet*，源文件，视频。
- *refFastTriax-basic*，源文件，视频。
- *refFastTriax-basic-without-plots*，源文件，视频。
- *refFastUnvRead*，源文件，视频。

四面体

- *refFastOneTetra*，源文件，视频。
- *refFastOneTetraPoly*，源文件，视频。
- *refFastTwoTetras*，源文件，视频。
- *refFastTwoTetrasPoly*，源文件，视频。

第2章

面向用户的 Yade

2.1 离散元法公式

本章将从数学角度描述显式离散元模拟的一般特征，并适当参照 Yade 对这些算法的实现。下面大致按照这些特征在模拟中出现的顺序进行介绍；首先，两个颗粒可能建立新的相互作用，其中包括

1. 检测颗粒之间的碰撞；
2. 创建新的相互作用并确定其属性（例如刚度）；这些属性可以预先计算，也可以根据两个颗粒的属性推导得出；

然后，对已经存在的相互作用执行以下操作：

1. 应变评估；
2. 基于应变的应力计算；
3. 对相互作用中的颗粒施加力。

这个简化的描述仅用于说明本章各截面的排列顺序。稍后将给出对这一模拟循环更详细的描述。

在本章中，我们将接触的运动学变量称为“应变”，尽管在这一尺度上，通常也会称之为“位移”。哪种说法更合适，取决于所采用的概念模型，因此无法脱离具体问题单独判定。熟悉位移概念的读者可以在心中将法向应变和剪切应变分别替换为法向位移和剪切位移，而不会改变下文的含义。

2.1.1 碰撞检测

概述

两个颗粒之间碰撞构型的精确计算可能相当耗时（例如 球体 与 面片之间）。取一对一般刚体 i 和 j，以及它们由点集表示的“精确”（即数值实现所允许的精度意义上的）空间谓词（在 Yade 中称为 形状 ）P_i，P_j，检测通常分两步进行：

1. 使用近似谓词 $\tilde{P}_i$ 和 $\tilde{P}_j$进行快速碰撞检测；它们经过预先构造，能够抽象出 P_i 和 P_j 的个别特征，并满足以下条件

$$\forall \mathbf{x} \in \mathbb{R}^3 : x \in P_i \Rightarrow x \in \tilde{P}_i \tag{2.1}$$

($\tilde{P}_j$也同理)。这个近似谓词称为“包围体”（*Bound*，在 Yade 中），因为它从外部包围了任意颗粒的体积（这是蕴含关系的结果）。因此 $(P_i \cap P_j) \neq$

$\tilde{\emptyset} \Rightarrow (P_i \cap P_j) \neq \emptyset$ 并且，应用否定后件律，

$$\left(\tilde{P}_i \cap \tilde{P}_j\right) = \emptyset \Rightarrow \left(P_i \cap P_j\right) = \emptyset \tag{2.2}$$

这才是严格意义上的候选排除规则。

2. 通过在（2.2）中筛除不可能发生的碰撞，可以在可能存在的相互作用上运行开销更大的精确碰撞检测算法，并筛除剩余的伪配对 $\tilde{\emptyset} \wedge \left(P_i \cap P_j\right) = \emptyset$ $\left(\tilde{P}_i \cap \tilde{P}_j\right) \neq$ P_i P_j 。

这些算法作用于形状类型的所有可能组合，并且必须能够处理它们。

这里我们只关注第一步。

算法

碰撞评估算法一直是机器人学、计算机图形学和模拟等领域广泛研究的主题。大致可分为两组：

层次算法 通过递归划分空间，并在第一遍检查中限制近似检查的数量；因为低层级包围体只有在属于同一个高层级包围体时才可能相交。层次结构中的元素是不同类型的包围体：八叉树 [Jung1997], 包围球 [Hubbard1996], k-DOP [Klosowski1998]。

扁平算法 **直接使用包围体**，而不先将其分组到层次结构中；让我们只提及颗粒模拟中常用的两种：

扫描与裁剪算法 使用轴对齐包围盒，只有当它们在所有坐标轴上都重叠时才会发生重叠。这些算法的复杂度大致为 $\mathcal{O}(n \log n)$，其中 n 表示颗粒数量，前提是它们利用了模拟的时间相干性。

网格算法 用一组间距规则的有限点表示连续 R^3 空间，从而实现极快的邻居搜索；其复杂度可达到 $\mathcal{O}(n)$ 复杂度[Munjiza1998] 以及近期研究提出了克服该方法一项主要缺点的途径：必须根据模拟中最大的颗粒调整网格单元大小（[Munjiza2006], “多步”扩展）。

时间相干性 表示模拟中的颗粒运动并非任意发生，而是受物理定律支配。利用这一认识可以优化性能。

运动方程积分的数值稳定性决定了 Δt （第 稳定性考量节）的上限，也因此限制了颗粒在一步内的位移。[Munjiza2006]，考虑到了这一点，这意味着任一颗粒在一步内移动的距离不得超过一个相邻网格单元，从而使 $\mathcal{O}(n)$ 复杂度成为可能；下文所述扫描与修剪算法的周期变体也利用了这一点。

在更精细的层面上，通常会扩大 P_i 谓词，使其在2.1条件下，连续满足多个时间步；随后，第一次碰撞检测遍历便可按步长运行，从而大幅加快模拟。Verlet 最初发表的这项优化 [Verlet1967] 采用了扩大的邻居列表，因此这项技术得名为*Verlet* 列表。不过在一般情况下，未必会使用邻居列表，此时通常采用术语*Verlet* 距离。

扫描与剔除

下面详细介绍 Yade 中用于碰撞检测的扫描与修剪算法（类 InsertionSortCollider）。轴对齐包围盒（Aabb）用作 P_i；每个 Aabb 由下角点和上角点 $\in R^3$ 确定（下文中，$\tilde{P}_i^{x0}$、$\tilde{P}_i^{x1}$分别是 $\tilde{P}_i$ 沿x轴等方向的最小/最大坐标）。由各种颗粒形状（例如 Sphere、Facet、Wall）构造 Aabb 很简单，由相应的 BoundFunctor 派生类处理（Bo*1*_Sphere_Aabb、Bo*1*_- Facet_Aabb，.....）。

两个 *Aabb*'s 是否重叠，可通过判断各轴上区间重叠的合取来确定（*fig-sweep-and-prune*）：

$$\left(\tilde{P}_i \cap \tilde{P}_j\right) \neq \emptyset \Leftrightarrow \bigwedge_{w\in\{x,y,z\}} \left[\left(\left(\tilde{P}_i^{w0}, \tilde{P}_i^{w1}\right) \cap \left(\tilde{P}_j^{w0}, \tilde{P}_j^{w1}\right)\right) \neq \emptyset\right]$$

其中，(a, b) 表示 R中的区间。

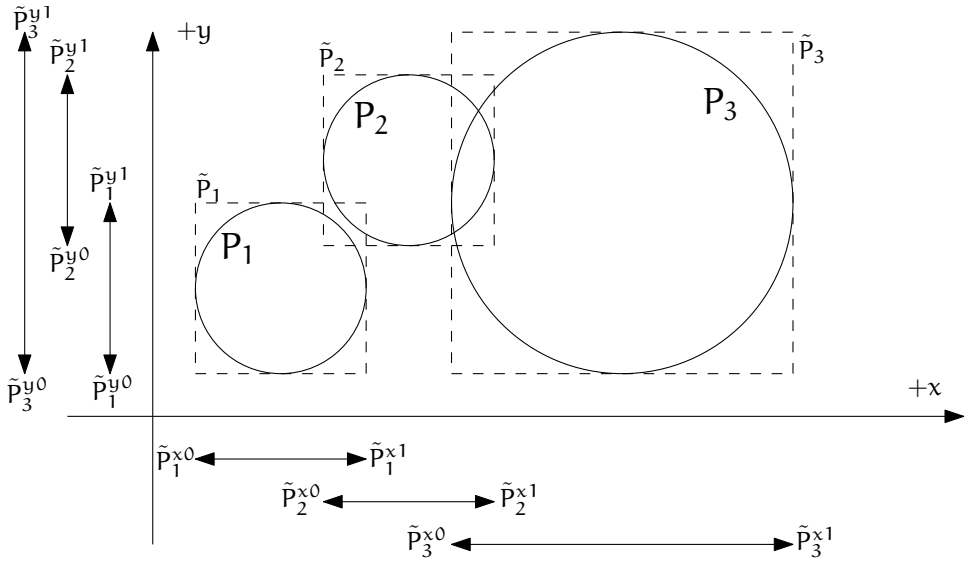


图 1：扫描与修剪算法（以二维展示），其中 *Aabb* 用每个球体沿各轴的最小值和最大值表示。只有当 *Aabb* 在所有轴上都重叠时，它们才存在空间重叠。~ ~ ~ ~在这种情况下，$P_1 \cap P_2 \neq \emptyset$（但请注意，$P_1 \cap P_2 = \emptyset$）以及 $P_2 \cap P_3 \neq \emptyset$。}

碰撞器为每个轴 $w \in \{x, y, z\}$分别维护 3 个列表（数组） L_w

$$L_w = \bigcup_i \left\{\tilde{P}_i^{w0}, \tilde{P}_i^{w1}\right\}$$

其中 i 遍历所有颗粒。 L_w 数组（已排序的集合）包含每个 *Aabb* 的最小和最大角点的相应坐标（我们称这些坐标为*Bound*，下文中如此称呼）；除 Bound 外，每个列表元素还包含 `id`，用于指明其所属的颗粒，以及表示它是下界还是上界的标志。

在初始步中，所有列表都会排序（使用快速排序，平均 $\mathcal{O}(n$对数 $n)$），并使用一个轴来创建初始相互作用：遍历每个刚体的下边界与上边界之间的范围，其中间的边界表示可能存在 *Aabb* 重叠，因此还必须检查其余各轴。

在后续的每一步中，列表都已预先排序。当一个颗粒的坐标刚刚越过另一个颗粒的坐标时，就会出现逆序；逆序的数量受模拟的数值稳定性及其物理意义的限制（这使算法具有时空连贯性）。如果相邻元素的顺序颠倒，插入排序算法就会交换它们；对于已排序和未排序的列表，其复杂度分别介于 $\mathcal{O}(n)$ 和 $\mathcal{O}(n^2)$之间。就我们的用途而言，只需处理逆序；根据排序算法的特性，逆序会在排序循环内部被检测到。逆序可能表示：

- 沿当前轴发生重叠：如果上边界与下边界发生逆序（交换），也就是说，坐标较高的上边界原本排在坐标较低的下边界之前，顺序不正确。接着检查其他两个轴上的重叠；如果所有轴上都存在重叠，就会创建新的潜在相互作用。

- 沿当前轴的重叠结束；若下界与上界反转（交换），则如果这两个颗粒之间只有势相互作用，该相互作用就会被删除。
- 如果两个边界都为上界或都为下界，则无需执行任何操作。

非周期插入排序

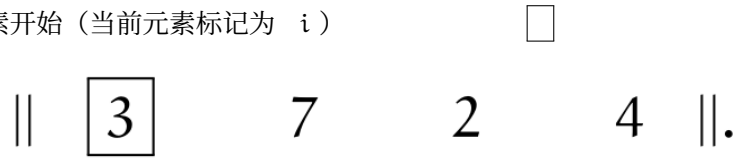
下面通过一个数列试样来演示排序算法：



从左向右遍历各元素；每个元素都会不断与左邻元素交换，并向左移动，直到满足以下任一条件：

(≤)	与左邻元素的排序顺序正确，或
(||)	该元素位于序列的开头。

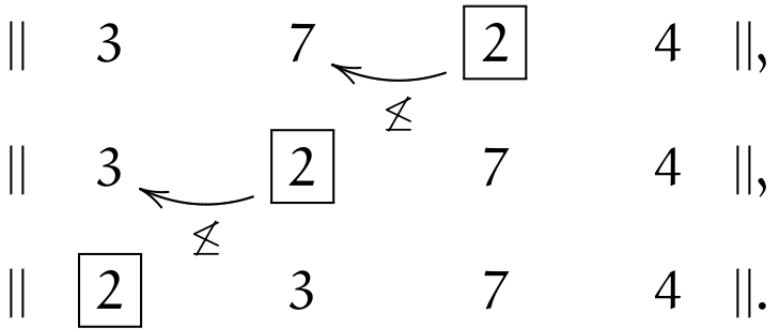
我们从最左侧的元素开始（当前元素标记为 i）



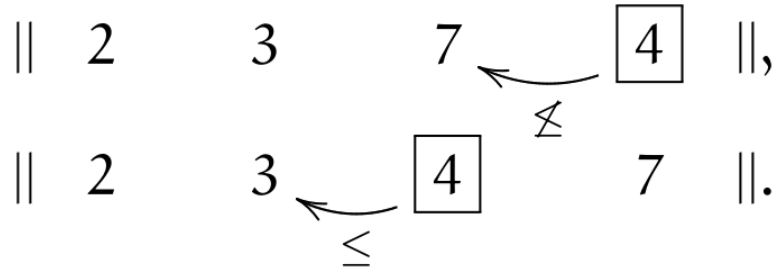
显然，它立即满足（||），于是我们转到下一个元素：



条件（≤）成立，因此我们向右移动。2 的顺序不正确（违反了 (≤)），发生了两次逆序；之后，（||）成立：



最后一个元素 4 起初违反了（≤），但经过一次逆序后便满足了该条件



遍历完全部元素后，序列现在已排序。

显然，如果初始序列已排序，只需遍历元素，无需处理任何逆序（这会在 $\mathcal{O}(n)$ 时间内完成）。

在模拟排序过程中，每遇到一次逆序，都会调用用于检查变化的函数*Aabb*重叠情况，并创建或删除相互作用。

排序算法的周期变体详见周期性插入排序算法，以及其他与周期性边界相关的主题。

使用 Verlet 距离进行优化

如上所述， [Verlet1967]曾探讨过通过˜扩大谓词 P_i，来实现稀疏地运行碰撞检测的可能性。

在 Yade 中，这是通过在所有维度上将颗粒的*Aabb*扩大一个固定的相对长度（或 Verlet 距离）来实现的 ΔL （*InsertionSortCollider.sweepLength*）。假设碰撞器上次在第 m 步运行，而当前步为第 n。*NewtonIntegrator* 通过比较当前m 与 n 参考位置，追踪每个颗粒在这两步之间移动的累计距离，参考位置的时间为 n （*Bound::refPos*），

$$L_{mn} = |X^n - X^m| \tag{2.3}$$

一旦某个颗粒满足以下条件，就会触发碰撞器重新运行：

$$L_{mn} > \Delta L. \tag{2.4}$$

ΔL 主要由参数 *InsertionSortCollider.verletDist* 定义。可以通过赋予正值直接设置，也可以通过赋予负值间接设置（该负值定义了 ΔL 与最小颗粒半径的比例）。此外，还可以使用 *InsertionSortCollider.targetInterv* 为每个颗粒独立调整 ΔL。较大的 ΔL 将分配给速度最快的颗粒，使所有颗粒理想情况下都能在经过这个“目标”迭代次数后到达其边界的边缘。使用 Verlet 距离所得到的结果高度取决于模拟的性质以及对 *InsertionSortCollider.targetInterv* 的选择。如果问题的某些部分包含高速颗粒，而其他部分不移动，则为每个颗粒独立调整尺寸尤其有效。否则，与 targetInterv=0（为所有颗粒赋予相同的 ΔL ）相比，通常不会有显著收益。

选择合适的 Verlet 距离时，还需要考虑颗粒数量和可用线程数。较大的距离会减少碰撞器（以单线程运行）所耗费的时间，并增加计算相互作用（多线程）所耗费的时间。通常，对于使用多核计算机且包含超过 ΔL 颗粒的大型模拟，将使用较大的 10^5 。另一方面，对于使用单处理器且颗粒数少于 10^4 的模拟，较小的 ΔL 可能更有利。用户的基准测试可在 Yade 维基上找到（例如参见 https://yade-dem.org/wiki/Colliders_性能）。

2.1.2 创建颗粒间相互作用

上文所述的碰撞检测只是近似的。精确碰撞检测取决于各个颗粒的几何形状，并单独处理。在 Yade 术语中，碰撞器只会创建潜在相互作用；潜在相互作用会通过专用算法进行精确评估，这些算法用于处理两个球体或其他组合之间的碰撞。精确碰撞检测必须在每个时间步长运行，因为颗粒可能在每一步改变彼此的位置（如果使用 Verlet 距离优化，碰撞器则只会偶尔运行）。运动学变量一文介绍了一些精确碰撞检测算法；在 Yade 中，这些算法由继承自*IGeomFunctor* 的类实现（前缀为 `Ig2`）。

除了检测几何重叠（在 Yade 中对应于*IGeom* ）之外，还需要确定相互作用的非几何属性（*IPhys*）。在 Yade 中，每当产生新相互作用时，都会调用一个继承自*IPhysFunctor* 的函子（前缀为 `Ip2`），根据两个颗粒各自的材料类型计算这些属性。

刚度

基本 DEM 相互作用定义了两种刚度：法向刚度 K_N 和剪切（正切）刚度 K_T。理想情况下， K_N 应与颗粒材料的虚拟杨氏模量相关，而 K_T 则

通常取计算所得 K_N的一定比例。 K_T/K_N 比值决定了颗粒排列的宏观泊松比，这可通过量纲分析说明：弹性连续体有两个参数（E 和 ν），基本 DEM 模型也有两个参数，且尺寸相同，分别为 K_N 和K_T/K_N；因此，宏观泊松比仅由 K_T/K_N 决定，而宏观杨氏模量则与 K_N 成比例，并受 K_T/K_N影响。

当然，这种分析做了大量简化，未考虑粒子半径分布、堆积构型以及其他可能的参数，例如后文引入的相互作用半径。

法向刚度

Yade 中常用的算法将法向相互作用刚度计算为两个弹簧串联时的刚度，其中弹簧长度等于球体半径（*fig-spheres-contact-stiffness*）。

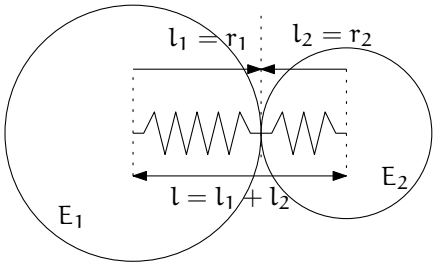


图 2：用两个弹簧串联表示两个球体之间接触的法向刚度。

定义距离 $l = l_1 + l_2$，其中 l_i 是接触点与球心之间的距离，初始时（粗略地说）等于球体半径。球心之间的距离变化 Δl 按照两球体的柔度成比例地分配为各自的变形 $\Delta l = \Delta l_1 + \Delta l_2$ 。位移变化 Δl_i 产生力 $F_i = K_i \Delta l_i$，其中 K_i 确保成比例关系，并具有物理意义 ~，其量纲为刚度； K_i 与球体材料模量 E_i 以及某个长度 l_i 有关，该长度与 r_i 成比例。

$$
\begin{aligned}
\Delta l &= \Delta l_1 + \Delta l_2 \\
K_i &= E_i \tilde{l}_i \\
K_N \Delta l &= F = F_1 = F_2 \\
K_N \left(\Delta l_1 + \Delta l_2\right) &= F \\
K_N \left(\frac{F}{K_1} + \frac{F}{K_2}\right) &= F \\
K_1^{-1} + K_2^{-1} &= K_N^{-1} \\
K_N &= \frac{K_1 K_2}{K_1 + K_2} \\
K_N &= \frac{E_1 \tilde{l}_1 E_2 \tilde{l}_2}{E_1 \tilde{l}_1 + E_2 \tilde{l}_2}
\end{aligned}
$$

最常用的 ~用于计算相互作用属性的类 *Ip2_FrictMat_FrictMat_FrictPhys* 使用 $l_i = 2$ r_i.

~一些公式定义了一个等效截面 A_{eq}，在这种情况下，它会出现在 l_i 项中，形式为$K_i = E_i \tilde{l}_i = E_i \frac{A_{eq}}{l_i}$。混凝土模型就是一个例子（*Ip2_CpmMat_CpmMat_CpmPhys*），其中$A_{eq} =$ 最小值(r_1, r_2)。

基于上述原因，不应假定颗粒尺度的 E_i 与宏观模量相等 E 。某些公式，例如 [Hentz2003], ，会引入参数以使二者在数值上匹配。我们认为这并不合适，因为这样会将这些值与用于标定的特定球体排列特征绑定在一起。

其他参数

对于不同的材料模型，非弹性参数各不相同。不过，通常情况下，如果有意义，这些参数会根据颗粒的材料属性取平均值。例如，*Ip2_CpmMat_CpmMat_CpmPhys*会对大多数物理量取平均值，而*Ip2_FrictMat_FrictMat_FrictPhys* 将内摩擦角计算为$\varphi = \text{最小值}(\varphi_1, \varphi_2)$，以避免与无摩擦刚体发生摩擦。

2.1.3 运动学变量

一般情况下，两个颗粒的相互构型有 6 个自由度（DoFs），就像三维空间中的一根梁：两个颗粒各自都有 6 个自由度，但相互作用本身可以在空间中平移和旋转（两个球体均参与其中），因此本身也有 6 个自由度；于是 $12-6=6$。它们如 *fig-spheres-dofs*所示。

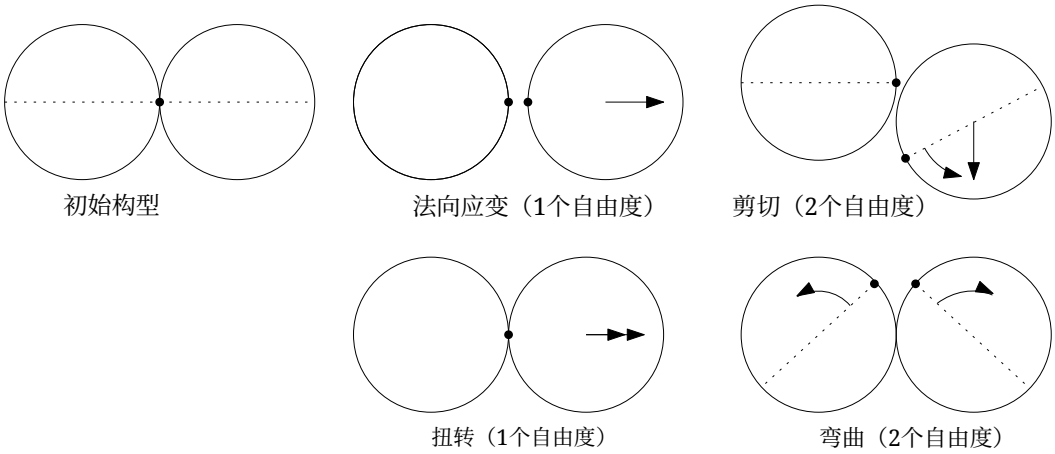


图 3：两个球体构型的自由度。若沿相互作用轴（$\mathbf{n}$）的线速度存在差异，就会产生法向运动；垂直于 $\mathbf{n}$ 的线速度差异，以及 $\boldsymbol{\omega}_1+\boldsymbol{\omega}_2$ 垂直于 $\mathbf{n}$ 的部分，会产生剪切；$\boldsymbol{\omega}_1-\boldsymbol{\omega}_2$ 平行于 $\mathbf{n}$ 的部分会引起扭转；$\boldsymbol{\omega}_1-\boldsymbol{\omega}_2$ 垂直于 $\mathbf{n}$ 的部分会引起弯曲。

下文只描述相对运动的法向分量和剪切分量，不讨论扭转和弯曲。原因是大多数接触本构定律都不涉及后两者。

法向变形

常数

考虑两个球体，其初始中心分别为 $\mathbf{C}_1$、$\mathbf{C}_2$，半径分别为 r_1、r_2，并且它们发生了接触。接触中球体的顺序是任意的，不会影响其行为。接下来我们定义长度

$$d_0 = |\bar{\mathbf{C}}_2 - \bar{\mathbf{C}}_1|$$
$$d_1 = r_1 + \frac{d_0 - r_1 - r_2}{2}, \qquad d_2 = d_0 - d_1.$$

这些物理量在相互作用的整个生命周期内都是恒定的，并且仅在建立相互作用时计算一次。距离 d_0 是参考距离，例如，可用于将绝对位移转换为无量纲应变。此外，在该距离处（对于常见接触定律），球体之间既没有排斥力，也没有吸引力，因此称为平衡距离。

距离 d_1 和 d_2 定义球体的缩减（或扩展）半径；几何半径 r_1 和 r_2 仅用于碰撞检测，可能与 d_1 和 d_2不同，如图 *fig-sphere-sphere*所示。

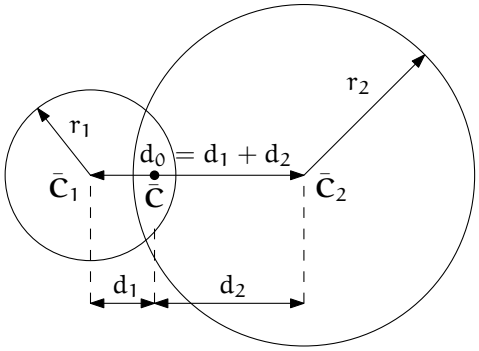


图 4：两个球体初始接触的几何形态；此算例展示了接触创建时已发生重叠的球体（这种情况可能出现在模拟开始时），以¯确保普遍性。初始接触点 C 位于重叠区域的中间。

在球体之间的平均接触数量应增加的算例中，会利用这种差异，例如影响压缩时的响应或稳定堆积体。在这种情况下，还会基于相互作用半径 R_I为几何上并未重叠的球体创建相互作用；该无量纲参数用于确定接触检测的“非局部性”。对于 $R_I = 1$，只有彼此接触的球体才被视为接触；一般条件为

$$d_0 \leq R_I(r_1 + r_2). \tag{2.5}$$

该值 R_I 会直接影响每个球体的平均相互作用数量（渗流），而˜对于某些模型而言，这是获得真实结果所必需的。在这种情况下，*Aabb*（或 P_i 一般的谓词）必须相应地扩大（*Bo1_Sphere_Aabb.aabbEnlargeFactor*）。

接触截面

某些本构定律以应变和应力表示（定律*2_ScGeom_CpmPhys_Cpm*，例如后文所述的混凝土模型）；在这种情况下，必须引入接触的等效截面，以满足量纲要求。其精确定义相当任意；CPM 模型（*Ip2_CpmMat_CpmMat_CpmPhys*）采用如下关系

$$A_{eq} = \pi \min(r_1, r_2)^2 \tag{2.6}$$

如果所用的本构定律以应力和应变的形式定义，该值将用于将应力转换为力。请注意，也可以使用不同于 π 的值；它只会缩放宏观堆积刚度。我们选择 A_{eq} 表示圆的面积，仅仅是为了便于直观地将其视为颗粒中心之间的桁架状单元。此外，也可以使用不同于 $\min(r_1, r_2)$ 的函数，不过结果应与 r_1 和 r_2 呈线性关系，这样颗粒尺寸缩放后，方程仍能给出一致的结果。

变量

在模拟过程中，球体运动时（随着 $\mathbf{C}_1^\circ$ 和$\mathbf{C}_2^\circ$ 变化），以下状态变量会更新：

$$\mathbf{n}^\circ = \frac{\mathbf{C}_2^\circ - \mathbf{C}_1^\circ}{|\mathbf{C}_2^\circ - \mathbf{C}_1^\circ|} \equiv \widehat{\mathbf{C}_2^\circ - \mathbf{C}_1^\circ} \tag{2.7}$$

and

$$\mathbf{C}^\circ = \mathbf{C}_1^\circ + \left(d_1 - \frac{d_0 - |\mathbf{C}_2^\circ - \mathbf{C}_1^\circ|}{2}\right)\mathbf{n}. \tag{2.8}$$

接触点 $\mathbf{C}^\circ$ 始终位于球体重叠区域的中间（即使重叠量为负值，此时它位于球体之间空隙的中间）。接触平面始终垂直于接触平面法线 $\mathbf{n}^\circ$，并经过 $\mathbf{C}^\circ$。

法向位移和应变可以定义为

$$u_N = |\mathbf{C}_2^\circ - \mathbf{C}_1^\circ| - d_0,$$
$$\varepsilon_N = \frac{u_N}{d_0} = \frac{|\mathbf{C}_2^\circ - \mathbf{C}_1^\circ|}{d_0} - 1.$$

由于 u_N 始终与 $\mathbf{n}$对齐，因此必要时可以将其存储为乘以 $\mathbf{n}$ 的标量值。

对于压缩程度极大的模拟，使用对数应变可能更有益；这样，当两个球体的中心相互靠近时，应变会趋于 $-\infty$ （而不是 -1）。否则，排斥力将保持有限，球体可能会彼此穿透。因此，我们可以如下调整法向应变的定义：

$$\varepsilon_N = \begin{cases} \log\left(\frac{|\mathbf{C}_2^\circ - \mathbf{C}_1^\circ|}{d_0}\right) & \text{if } |\mathbf{C}_2^\circ - \mathbf{C}_1^\circ| < d_0 \\ \frac{|\mathbf{C}_2^\circ - \mathbf{C}_1^\circ|}{d_0} - 1 & \text{otherwise.} \end{cases}$$

然而，这种定义的缺点是会使接触刚度实际上增大（直至无穷），因此需要调整 Δt，以免模拟变得不稳定。使用基于刚度的时间步进器可以进行这种动态调整（*GlobalStiffnessTimeStepper* （在 Yade 中）。

剪切变形

为了保持 $\mathbf{u}_T$ 的一致性（例如，如果两个球体保持彼此间的配置不变，但在空间中任意移动，则 $\mathbf{u}_T$ 必须保持恒定），那么 $\mathbf{u}_T$ 要么必须跟踪球体的空间运动，要么必须（以某种方式）完全依赖球体局部数据。

剪切应变的几何意义如*fig-shear-2d*所示。

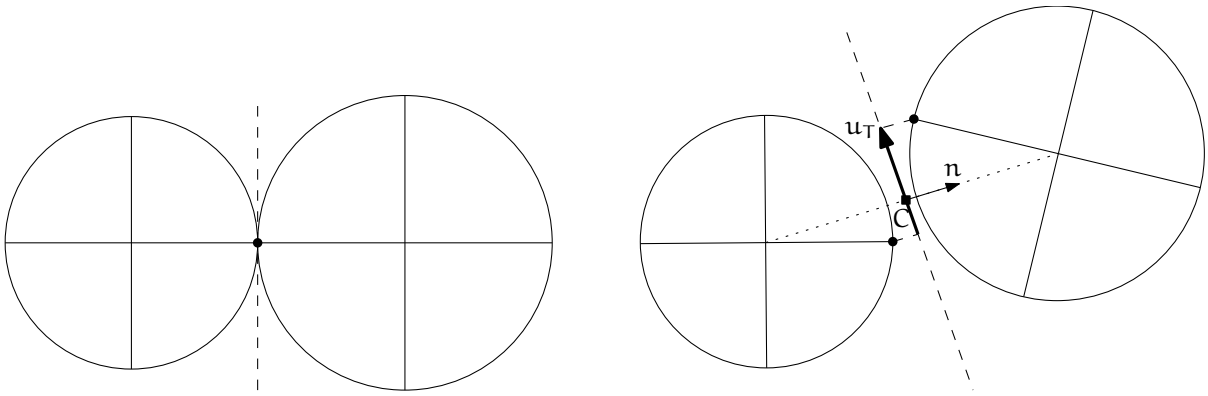


图 5：球体相对运动（包括线性运动和旋转）引起的剪切位移 $\mathbf{u}_T$ 的演变。左侧为初始接触状态，右侧为一个颗粒发生位移和旋转后的状态。

经典增量算法广泛应用于离散元代码中，并有大量文献对其进行描述（[Luding2008], [Alonso2004]）。Yade 在 *ScGeom* 类中实现了这一算法。每一步都会更新剪切位移 $\mathbf{u}_T$；更新增量可分解为两部分：全局空间中的相互作用运动（即 $\mathbf{C}$ 和 $\mathbf{n}$）以及球体的相对运动。

1. 球体位置的变化会导致接触移动 $\mathbf{C}_1$ 和 $\mathbf{C}_2$，从而按照（2.8）和（2.7）更新当前的 $\mathbf{C}^\circ$和 $\mathbf{n}^\circ$。$\mathbf{u}_T^-$ 在前一步 $\mathbf{n}^-$处与接触平面垂直，必须对其进行更新，使得 $\mathbf{u}_T^- + (\Delta\mathbf{u}_T) = \mathbf{u}_T^\circ \perp \mathbf{n}^\circ$；具体做法是先将其正交投影到平面上（这可能会减小 $|\mathbf{u}_T|$），再添加与空间旋转相对应的内容：

相互作用，而不是：

$$(\Delta \mathbf{u}_T)_1 = -\mathbf{u}_T^- \times (\mathbf{n}^- \times \mathbf{n}^\circ)$$
$$(\Delta \mathbf{u}_T)_2 = -\mathbf{u}_T^- \times \left(\frac{\Delta t}{2}\mathbf{n}^\circ \cdot (\boldsymbol{\omega}_1^\ominus + \boldsymbol{\omega}_2^\ominus)\right)\mathbf{n}^\circ$$

2. 球体的相互运动，仅取其垂直于 $\mathbf{n}^\circ$ 的分量； $\boldsymbol{v}_{12}$ 表示球体在接触点处的相对速度：

$$\boldsymbol{v}_{12} = \left(\boldsymbol{v}_2^\ominus + \boldsymbol{\omega}_2^\ominus \times (-d_2\mathbf{n}^\circ)\right) - \left(\boldsymbol{v}_1^\ominus + \boldsymbol{\omega}_1^\ominus \times (d_1\mathbf{n}^\circ)\right)$$
$$\boldsymbol{v}_{12}^\perp = \boldsymbol{v}_{12} - (\mathbf{n}^\circ \cdot \boldsymbol{v}_{12})\mathbf{n}^\circ$$
$$(\Delta \mathbf{u}_T)_3 = -\Delta t \boldsymbol{v}_{12}^\perp$$

最后，我们计算

$$\mathbf{u}_T^\circ = \mathbf{u}_T^- + (\Delta \mathbf{u}_T)_1 + (\Delta \mathbf{u}_T)_2 + (\Delta \mathbf{u}_T)_3.$$

2.1.4 接触模型（示例）

相互作用的运动学变量通过本构定律，用于确定作用在两个球体上的力。在离散元法（DEM）中，某些本构定律通常用应变和应力表示，而另一些则采用位移/力表述。此处描述的定律属于后一类。

此处介绍的本构定律是离散元法（DEM）中最常见的一种，最初由 Cundall 提出。上一节描述了与接触模型无关的运动学变量，而力的计算则取决于所建模材料的性质。此处介绍的本构定律是最简单的非黏聚弹性摩擦接触模型，Yade 在 *Law2_ScGeom_-FrictPhys_CundallStrack*（所有本构定律都派生自基类 *LawFunctor*）。

建立新接触时（详见 引擎），其属性（*IPhys*）根据与两个颗粒相关联的材料 计算得出。对于简单的摩擦材料 *FrictMat*，*Ip2_-FrictMat_FrictMat_FrictPhys* 会创建一个新的 *FrictPhys* 实例，用于定义法向刚度 K_N、剪切刚度 K_T 和摩擦角 φ。

每一步中，给定法向位移和剪切位移 $u_N, \mathbf{u}_T$，即可计算法向力和剪切力（如果$u_N > 0$，则删除接触且不生成任何力）：

$$\mathbf{F}_N = K_N u_N \mathbf{n},$$
$$\mathbf{F}_T^t = K_T \mathbf{u}_T$$

其中， $\mathbf{F}_N$ 是法向力， $\mathbf{F}_T^t$ 是试探切向力。采用简单的非关联应力返回算法来计算最终剪切力

$$\mathbf{F}_T = \begin{cases} \mathbf{F}_T^t \frac{|\mathbf{F}_N| \tan\varphi}{|\mathbf{F}_T^t|} & \text{if } |\mathbf{F}_T^t| > |\mathbf{F}_N| \tan\varphi, \\ \mathbf{F}_T^t & \text{otherwise.} \end{cases}$$

随后将总力 $\mathbf{F} = \mathbf{F}_N + \mathbf{F}_T$ 施加到两个颗粒上——每个颗粒都会在每一步中累积作用于其上的力和力矩。由于计算出的力作用于接触点 $\mathbf{C}$，而该点与球体中心不同，因此还必须考虑由 $\mathbf{F}$ 产生的力矩。

$$\mathbf{F}_1 += \mathbf{F} \qquad \mathbf{F}_2 += -\mathbf{F}$$
$$\mathbf{T}_1 += d_1(-\mathbf{n}) \times \mathbf{F} \qquad \mathbf{T}_2 += d_2\mathbf{n} \times \mathbf{F}.$$

2.1.5 运动积分

每个颗粒都会累积其参与的接触所产生的广义力（力和力矩）。随后，这些广义力用于分别对每个颗粒的运动方程进行积分；因此，本节省略用于表示第 i 个颗粒的 i 索引。

这里采用常见的蛙跳格式（也称为 Verlet 格式），并对非球形颗粒的旋转作了一些调整，具体如下文所述。“蛙跳”这一名称源于：位置/方向的偶数阶导数在整步点已知，而奇数阶导数在半步点已知。回顾一下，我们分别用 a^-、 a°、 a^+ 表示在 $t-\Delta t$、 t 和 $t+\Delta t$ 处的 a 整步值；用 $a^\ominus$、 $a^\oplus$ 表示在 $t-\Delta t/2$、 $t+\Delta t/2$ 处的 a 半步值。

所述积分算法在 Yade 的*NewtonIntegrator* 类中实现。

位置

积分运动包括使用颗粒当前的加速度 $\ddot{u}^\circ$，将其位置从当前值 u° 更新为下一个时间步长处的值 u^+。已知作用于该颗粒的当前作用力 F 及其质量 m，加速度的计算很简单：

$$\ddot{u}^\circ = F/m.$$

使用步长为 Δt 的二阶有限差分，可得

$$\ddot{u}^\circ \cong \frac{u^- - 2u^\circ + u^+}{\Delta t^2}$$

由此可得

$$\begin{aligned} u^+ &= 2u^\circ - u^- + \ddot{u}^\circ \Delta t^2 = \\ &= u^\circ + \Delta t \underbrace{\left(\frac{u^\circ - u^-}{\Delta t} + \ddot{u}^\circ \Delta t \right)}_{(\dagger)}. \end{aligned}$$

通常， u^- 尚未知（只有 u° 已知）；不过，我们注意到

$$\dot{u}^\ominus \simeq \frac{u^\circ - u^-}{\Delta t},$$

即前一步的平均速度，该值是已知的。将这个近似值代入 (†)项后，我们还注意到当前步的平均速度可以近似为

$$\dot{u}^\oplus \simeq \dot{u}^\ominus + \ddot{u}^\circ \Delta t,$$

其中 (†)；最终得到

$$u^+ = u^\circ + \Delta t \left(\dot{u}^\ominus + \ddot{u}^\circ \Delta t \right).$$

因此，该算法可以表述为：首先计算当前步的平均速度 $\dot{u}^\oplus$，我们需要将其存储起来以供下一步使用（正如现在使用其旧值 $\dot{u}^\ominus$ 一样），然后计算下一时间步的位置 u^+：

$$\begin{aligned} \dot{u}^\oplus &= \dot{u}^\ominus + \ddot{u}^\circ \Delta t \\ u^+ &= u^\circ + \dot{u}^\oplus \Delta t. \end{aligned}$$

位置在 $i\Delta t$ 时刻已知（如果 Δt 保持不变），而速度在 $i\Delta t + \frac{\Delta t}{2}$ 时刻已知。由于它们以这种方式交错（彼此跳过），因此产生了“蛙跳”算法这一俗称。

方向（球形）

颗粒取向的更新 q° 与位置更新的方式类似。首先，我们根据已知的当前力矩 T计算当前角加速度 $\dot{\omega}^\circ$。对于球形颗粒，其惯性张量在任意方向下都是对角的（因此在当前全局取向下也如此），满足 $I_{11} = I_{22} = I_{33}$，我们可以写成

$$\dot{\omega}_i^\circ = T_i / I_{11},$$

我们使用相同的近似方案，得到一个类似于（2.1.5）

$$\boldsymbol{\omega}^{\oplus} = \boldsymbol{\omega}^{\ominus} + \Delta t \dot{\boldsymbol{\omega}}^{\circ}.$$

构造表示旋转向量 $\boldsymbol{\omega}^{\oplus}\Delta t$ 的四元数 Δq，使其满足

$$(\Delta q)_{\vartheta} = |\boldsymbol{\omega}^{\oplus}|,$$
$$(\Delta q)_{u} = \widehat{\boldsymbol{\omega}^{\oplus}}$$

最后，通过旋转复合计算下一个方向 q^{+}

$$q^{+} = \Delta q q^{\circ}.$$

方向（非球形）

非球形颗粒的旋转积分比其位置的积分复杂得多，因为它们的局部参考系不是惯性系。在惯性矩阵 $\mathbf{I}$ 为对角矩阵的局部参考系中，刚体的旋转可用欧拉方程的连续形式描述（$i \in \{1,2,3\}$、 i、 j、 k为连续索引）：

$$\mathbf{T}_i = \mathbf{I}_{ii}\dot{\boldsymbol{\omega}}_i + (\mathbf{I}_{kk} - \mathbf{I}_{jj})\boldsymbol{\omega}_j\boldsymbol{\omega}_k.$$

由于当前 $\boldsymbol{\omega}$ 和 $\dot{\boldsymbol{\omega}}$的值同时存在，无法使用标准的蛙跳算法求解（平动以及球形物体的旋转也是如此；在这些情况下，该方程简化为 $\mathbf{T} = \mathbf{I}\dot{\boldsymbol{\omega}}$）。

此处介绍的算法由 [Allen1989] （第 84–89 页）所述，由 Fincham 为分子动力学问题设计；该算法通过中步/整步估计器扩展蛙跳算法，用于估算基本公式中在整步/中步位置已知的物理量。尽管它受到过批评，而且已有更精确的算法（[Omelyan1999], [Neto2006], [Johnson2008]），但由于相对简单，目前 Yade 仍采用此算法。

每个刚体都有自己的局部坐标系，该坐标系基于刚体的惯性主轴。我们用 $\tilde{\bullet}$ 表示局部坐标中的向量。局部坐标系的方向由当前颗粒的方向 q° （以四元数表示）给出；这个四元数可以表示为（当前的）旋转矩阵 $\mathbf{A}$。因此，每个向量 $\mathbf{a}$ 都按如下方式变换： $\tilde{\mathbf{a}} = q\mathbf{a}q^{*} = \mathbf{A}\mathbf{a}$。由于 $\mathbf{A}$ 是一个旋转（正交）矩阵，逆旋转为 $\mathbf{A}^{-1} = \mathbf{A}^{\mathsf{T}}$。

对于所讨论的给定颗粒，我们已知

- $\tilde{\mathbf{I}}^{\circ}$ （常量）惯性矩阵；由于是在局部主坐标系中，因此为对角矩阵，
- $\mathbf{T}^{\circ}$ 外力矩，
- q° 当前方向（及其等效旋转矩阵 $\mathbf{A}$），
- $\boldsymbol{\omega}^{\ominus}$ 中间步角速度，
- $\mathbf{L}^{\ominus}$ 中间步角动量；这是一个辅助变量，除.....之外还必须跟踪 r
  在此算法中使用。它在初始步中将为零向量。

我们的目标是计算后三个量的新值，即 $\mathbf{L}^{\oplus}$、 q^{+}、 $\boldsymbol{\omega}^{\oplus}$。首先估算当前角动量，并计算当前局部角速度：

$$\mathbf{L}^{\circ} = \mathbf{L}^{\ominus} + \mathbf{T}^{\circ}\frac{\Delta t}{2}, \qquad \tilde{\mathbf{L}}^{\circ} = \mathbf{A}\mathbf{L}^{\circ},$$
$$\mathbf{L}^{\oplus} = \mathbf{L}^{\ominus} + \mathbf{T}^{\circ}\Delta t, \qquad \tilde{\mathbf{L}}^{\oplus} = \mathbf{A}\mathbf{L}^{\oplus},$$
$$\tilde{\boldsymbol{\omega}}^{\circ} = \tilde{\mathbf{I}}^{\circ\,-1}\tilde{\mathbf{L}}^{\circ},$$
$$\tilde{\boldsymbol{\omega}}^{\oplus} = \tilde{\mathbf{I}}^{\circ\,-1}\tilde{\mathbf{L}}^{\oplus}.$$

然后，我们使用 q° 和 $\widetilde{\omega}^\circ$ 计算 $\dot{q}^\circ$:

$$\begin{pmatrix} \dot{q}^\circ_w \\ \dot{q}^\circ_x \\ \dot{q}^\circ_y \\ \dot{q}^\circ_z \end{pmatrix} = \frac{1}{2} \begin{pmatrix} q^\circ_w & -q^\circ_x & -q^\circ_y & -q^\circ_z \\ q^\circ_x & q^\circ_w & -q^\circ_z & q^\circ_y \\ q^\circ_y & q^\circ_z & q^\circ_w & -q^\circ_x \\ q^\circ_z & -q^\circ_y & q^\circ_x & q^\circ_w \end{pmatrix} \begin{pmatrix} 0 \\ \widetilde{\omega}^\circ_x \\ \widetilde{\omega}^\circ_y \\ \widetilde{\omega}^\circ_z \end{pmatrix},$$
$$q^\oplus = q^\circ + \dot{q}^\circ \frac{\Delta t}{2}.$$

我们以与（ $\dot{q}^\oplus$ 相同的方式计算 $q^\oplus$ 和 $\widetilde{\omega}^\oplus$ 中的值，但向前平移 2.1.5个单位 $\Delta t/2$ 。然后，最终可以计算出所需的值

$$q^+ = q^\circ + \dot{q}^\oplus \Delta t,$$
$$\omega^\oplus = \mathbf{A}^{-1} \widetilde{\omega}^\oplus$$

团簇（刚性聚集体）

离散元模拟经常使用颗粒的刚性聚集体来模拟复杂形状 [Price2007]称之为 团簇，通常由许多球体组成。团簇的动力学属性根据其成员的属性计算：

- 对于互不重叠的团簇成员，团簇的质量 m_c 为所有成员质量之和；惯性张量 I_c 使用平行轴定理计算：$I_c = \sum_i (m_i * d_i^2 + I_i)$，其中 m_i 是团簇成员 i的质量， d_i 是团簇成员中心 i 到团簇质心的距离， I_i是团簇成员 i的惯性张量。
- 对于相互重叠的团簇成员，团簇的质量 m_c 在团簇的轴对齐包围盒（*Aabb*）内按规则网格间距划分的单元格上求和，惯性张量使用平行轴定理计算：$\mathbf{I}_c = \sum_j (m_j * d_j^2 + I_j)$，其中 m_j 是单元格 j的质量， d_j 是单元格中心到团簇质心的距离， I_j 是单元格 j的惯性张量。

局部坐标轴的方向经过调整，使其成为主轴，且惯性张量为对角形式；同时改变团簇的方向，以补偿局部系统的旋转，从而使团簇成员在全局空间中的位置保持不变。所有团簇成员在局部坐标系中的初始位置和取向都会被存储。

在 Yade（类 团簇）中，在模拟期间，团簇成员在碰撞检测和接触求解方面的行为与独立颗粒相同，但同一团簇内的成员之间不会创建接触。只有在运动积分阶段，它们才会被特殊处理。系统不会分别对每个成员进行积分，而是将作用于这些颗粒的力/力矩 $\mathbf{F}_i$， $\mathbf{T}_i$ 转换为作用于团簇本身的力/力矩。设 r_i 每个颗粒相对于团簇质心的位置（以全局取向表示）。那么，团簇所受的合力和合力矩为

$$\mathbf{F}_c = \sum F_i,$$
$$\mathbf{T}_c = \sum r_i \times F_i + T_i.$$

随后，使用非球形旋转积分对团簇的运动进行积分。之后，在全局空间中移动团簇成员，使其在团簇局部坐标系中的初始位置和取向保持不变。这样一来，团簇成员的相对位置始终相同，从而呈现刚性团聚体的行为。

数值阻尼

在准静态现象的模拟中，消散颗粒的动能是可取的。由于大多数本构定律（包括 定律__ScGeom__FrictPhys__Basic 如上所示，接触模型（示例）） 不包含基于速度的阻尼（例如 [Addetta2001]中的阻尼），因此可以使用人工数值阻尼。其公式见 [Pfc3dManual30], ，不过我们对其版本稍作了调整。其

基本思路是通过 $(\Delta F)_d$ 比较当前加速度方向与颗粒速度方向，减小使颗粒速度增大的力，反之亦然。该操作按分量进行，因此这种阻尼方案显然不符合物理规律，因为它不具备坐标系旋转不变性；另一方面，它的计算非常简单。Cundall 提出了如下形式（由于该形式分别适用于所有颗粒，因此省略颗粒索引 i）：

$$\frac{(\Delta \mathbf{F})_{dw}}{\mathbf{F}_w} = -\lambda_d \operatorname{sgn}(\mathbf{F}_w \dot{\mathbf{u}}_w^{\ominus}), \quad w \in \{x, y, z\}$$

其中 λ_d 是阻尼系数。这种形式有几个优点 [Hentz2003]：

- 它作用于力（加速度），而不会限制匀速运动；
- 它与颗粒的特征频率无关，因此所有颗粒都受到同等阻尼；
- 它只需要无量纲参数 λ_d，该参数无需缩放。

在 Yade 中，我们采用经过调整的形式

$$\frac{(\Delta \mathbf{F})_{dw}}{\mathbf{F}_w} = -\lambda_d \operatorname{sgn} \mathbf{F}_w \underbrace{\left(\dot{u}_w^{\ominus} + \frac{\ddot{u}_w^{\circ} \Delta t}{2} \right)}_{\simeq \dot{u}_w^{\circ}}, \tag{2.9}$$

这里，我们用括号中的步内估计值替换了之前的步间速度 $\dot{u}^{\ominus}$。这样做是为了避免锁定力：当每一步施加力导致速度改变符号时，就会出现这种锁定力；也就是说，相关颗粒以 $2\Delta t$ 周期围绕平衡位置振荡。

在 Yade 中，阻尼（2.9）在 *NewtonIntegrator* 引擎中实现；阻尼系数 λ_d 为*NewtonIntegrator.damping*。

稳定性考量

临界时间步长

为了确保显式积分方案的稳定性，对 Δt 施加了上限：

$$\Delta t_{cr} = \frac{2}{\omega_{max}} \tag{2.10}$$

其中，ω_{max} 是系统中的最高特征频率。

单质量弹簧系统

质量为 m、刚度为 K 的单个一维质量弹簧系统由以下方程描述

$$m\ddot{x} = -Kx$$

其中，x 是相对于平均（平衡）位置的位移。简谐振动的求解为$x(t) = A\cos(\omega t + \varphi)$，其中相位 φ 和振幅 A 由初始条件决定。角频率

$$\omega^{(1)} = \sqrt{\frac{K}{m}} \tag{2.11}$$

不依赖于初始条件。由于只有一个质量，$\omega_{max}^{(1)} = \omega^{(1)}$。将 (2.11) 代入 (2.10)，得到

$$\Delta t_{cr}^{(1)} = 2/\omega_{max}^{(1)} = 2\sqrt{m/K}$$

对于单个振子。

广义质点-弹簧系统

对于一般的质点-弹簧系统，当两个相连的质量 m_i, m_j 反向运动时，频率最高；假设它们的速度相等（这是一个保守的假设），且由刚度为 K_i的弹簧连接： m_i 的位移 Δx_i 会伴随 m_j的 $\Delta x_j = -\Delta x_i$ ，得到$\Delta F_i = -K_i(\Delta x_i - (-\Delta x_i)) = -2K_i\Delta x_i$。由此得到表观刚度 $K_i^{(2)} = 2K_i$，进而得到整个系统的最大特征频率

$$\omega_{max} = \max_i \sqrt{K_i^{(2)}/m_i}.$$

因此，总体临界时间步长为

$$\Delta t_{cr} = \frac{2}{\omega_{max}} = \min_i 2\sqrt{\frac{m_i}{K_i^{(2)}}} = \min_i 2\sqrt{\frac{m_i}{2K_i}} = \min_i \sqrt{2}\sqrt{\frac{m_i}{K_i}}. \tag{2.12}$$

这个方程可用于平移和旋转中的全部 6 个自由度（DOF），方法是考虑广义质量矩阵和刚度矩阵 M 和 K，并将分数 $\frac{m_i}{K_i}$ 替换为 $M.K^{-1}$的特征值。此时，临界时间步长与频率最高的特征模态相关：

$$\Delta t_{cr} = \min \Delta t_{crk}, \quad k \in \{1, ..., 6\}. \tag{2.13}$$

离散元模拟

在离散元模拟中，单颗粒刚度 $\mathbf{K}_{ij}$ 由其参与接触的刚度决定。假设每个接触的法向刚度为 K_{Nk}、剪切刚度为 $K_{Tk} = \xi K_{Nk}$ ，且其方向由法向量 $\mathbf{n}_k$确定。平移刚度矩阵 $\mathbf{K}_{ij}$ 可定义为该颗粒参与的所有接触贡献之和（索引 k），即 [Chareyre2005]。

$$\mathbf{K}_{ij} = \sum_k (K_{Nk} - K_{Tk})\mathbf{n}_i\mathbf{n}_j + K_{Tk} = \sum_j K_{Nk}\left((1-\xi)\mathbf{n}_i\mathbf{n}_j + \xi\right) \tag{2.14}$$

与 i和 $j \in \{x, y, z\}$。方程（2.13）和（2.14）确定 Δt_{cr} 在模拟中的值。Yade 中的 *GlobalStiffnessTimeStepper* 引擎实现了一种推广到全部 6 个自由度的类似方法。包含旋转项的广义刚度推导非常相似，可参见[AboulHosn2017]。

出于计算效率方面的考虑，不计算刚度矩阵的特征值。这里仅假设各自由度彼此解耦，并使用 $K.M^{-1}$ 的对角线项来进行近似。在典型的机械系统中，这些近似值效果良好。

需要注意的是， $\omega_{max} > 0$ 有一个重要条件：如果颗粒之间没有接触，且 $\omega_{max} = 0$，我们将得到值 $\Delta t_{cr} = \infty$。虽然形式上正确，但这个值在数值上是错误的：我们默认刚度在每个时间步长内保持恒定，而当颗粒碰撞并产生接触时，情况并非如此。因此，在没有接触的情况下，必须根据未来的相互作用预先估算刚度，如下一节所示。

通过 Δt_{cr} 波传播速度进行估算

在没有相互作用的情况下估算时间步长，是基于相互作用刚度与颗粒属性之间的联系。请注意，在本截面中，符号 E 和 ρ 特指颗粒的杨氏模量和密度，而非宏观结构的杨氏模量和密度。

在 Yade 中，颗粒关联有 材料 ，它定义了密度 ρ (*Material.density*)，还可能定义（在 *ElastMat* 及其派生类中）颗粒的“杨氏模量” E (*ElastMat.young*) 。 ρ 用于根据颗粒的质量 m 初始计算其 ρ，而在颗粒间创建新的相互作用时会考虑 E ，并影响刚度 K_N。已知 m 和 K_N，我们可以估算每个颗粒的 2.14（显然我们忽略了

- 每个颗粒的相互作用数量 N_i；不过，对于“合理”的半径分布，几何条件会施加一个上限（例如，等半径球体的堆积体中为 12）；
- 颗粒刚度 E_i、 E_j之间的精确关系，假设只有 K_N 以某种方式与它们成比例。

通过定义 E 和 ρ，颗粒具有类似连续体的物理量。连续体方程的显式积分方案会根据声速 $\sqrt{E/\rho}$设定临界时间步长；在一个步长内，弹性波传播的距离不得超过积分点之间的最小距离 l_{min}。由于 E、 ρ 是弹性连续体的参数，且 l_{min} 预先固定，因此可得

$$\Delta t_{cr}^{(c)} = l_{min}\sqrt{\frac{\rho}{E}}.$$

就我们的目的而言，我们分别为每个颗粒定义 E 和 ρ； l_{min} 可替换为球体半径 R_i；严格来说，也可以使用 $l_{min} = 2R_i$，但考虑到球体与面片（其厚度为零）可能发生相互作用，我们改用 $l_{min} = R_i$。于是

$$\Delta t_{cr}^{(p)} = \min_i R_i\sqrt{\frac{\rho_i}{E_i}}.$$

此算法在 *utils.PWaveTimeStep* 函数中实现。

让我们将此结果与（2.12）进行比较；这需要作出若干简化假设：

- 所有颗粒均为球形，且半径相同，为 R;
- 球体的材料具有相同的 E 和 ρ;
- 每个球体的平均接触数为 N;
- 每个颗粒周围的接触具有足够均匀的空间分布;
- 所有相互作用的 $\xi = K_N/K_T$ 比值均为常数;
- 接触刚度 K_N 根据 E，使用以下形式的公式计算

$$K_N = E\pi' R', \tag{2.15}$$

其中 π' 是取决于所使用算法的某个常数footnote{例如，在 Yade 中， $\pi' = \pi/2$ 在混凝土颗粒模型（*Ip2_CpmMat_CpmMat_CpmPhys*）中，而 $\pi' = 2$ 在经典离散元模型（*Ip2_FrictMat_FrictMat_FrictPhys*）中。}以及 R' 是接触球体之间距离的一半；当相互作用半径为 R 时，其值等于 $R_I = 1$。如果 $R_I = 1$ （从而$R' \equiv R$），则所有相互作用都具有相同的刚度 K_N。在其他情况下，我们将 K_N 视为由平均 R' 计算得到的平均刚度（见下文）。

由于所有颗粒具有相同的参数，在下面的公式中我们省略 i 索引。

我们尝试根据（2.14）表示平均单颗粒刚度。它是对所有相互作用求和，其中 K_N和 ξ 是不会随相互作用旋转的标量，而 n_w 是 w单位相互作用法向量的第 n个分量 n_w^2。由于我们假设空间均匀分布，因此可以用其平均值 n_w^2替换 n。将 n_w^2 的分量识别为方向余弦后， $1/3$的平均值为 n_w^2。我们通过对所有可能的取向进行积分来求得平均值，这些取向在空间中均匀分布：

此外，由于所有方向都相同，我们可以将单体刚度写为 $K = \mathbf{K}_w$，适用于所有 $w \in \{x, y, z\}$。由此可得

$$K = \sum K_N\left((1-\xi)\frac{1}{3} + \xi\right) = \sum K_N\frac{1+2\xi}{3}$$

并可将常数项（所有项）置于总和之前。 $\sum 1$ 等于每个球体的接触数，即 N。由此得到

$$K = NK_N\frac{1-2\xi}{3},$$

我们将 K 代入（2.12），并使用（2.15）：

$$\Delta t_{cr} = \sqrt{2}\sqrt{\frac{m}{K}} = \sqrt{2}\sqrt{\frac{\frac{4}{3}\pi R^3 \rho}{NE\pi' R\frac{1-2\xi}{3}}} = \underbrace{R\sqrt{\frac{\rho}{E}}}_{\Delta t_{cr}^{(p)}} 2\sqrt{\frac{\pi/\pi'}{N(1-2\xi)}}.$$

由 P波速度预测的时间步长 $\Delta t_{cr}^{(p)}$ 与数值稳定的时间步长 Δt_{cr} 之比，是最后一个（无量纲）项的倒数：

$$\frac{\Delta t_{cr}^{(p)}}{\Delta t_{cr}} = 2\sqrt{\frac{N(1+\xi)}{\pi/\pi'}}.$$

该比值的实际值取决于堆积体 N、 $K_N/K_T = \xi$ 比值的特性，以及根据颗粒刚度计算接触刚度的方式。下面以 Yade 中的两个模型为例说明：

C混凝土颗粒模型 根据等效面积计算接触刚度 A_{eq} 第一项（2.6），

$$A_{eq} = \pi R^2 K_N = \frac{A_{eq} E}{d_0}.$$

d_0 是初始接触长度；当相互作用半径为（2.5）时 $R_I > 1$，平均而言，它大于 $2R$。对于 $R_I = 1.5$，我们可以粗略估算 $d_0 = 1.25 \cdot 2R = \frac{5}{2}R$，得到

$$K_N = E\left(\frac{2}{5}\pi\right) R$$

其中 $\frac{2}{5}\pi = \pi'$ 与（2.15）进行比较。

相互作用半径 $R_I = 1.5$ 会使半径相同的球体密堆积中每个球体的平均相互作用数达到 $N \approx 12$，球体半径为 R。通过校准 $\xi = 0.2$，使其与所需的宏观泊松比 $\nu = 0.2$相匹配。

最后，我们得到比值

$$\frac{\Delta t_{cr}^{(p)}}{\Delta t_{cr}} = 2\sqrt{\frac{12(1-2\cdot 0.2)}{\frac{\pi}{(2/5)\pi}}} = 3.39,$$

这表明 P 波算法会显著高估该值。

N非黏聚干摩擦模型是 Cundall 提出的基本模型，详见接触模型（示例）。假设球体半径近似恒定 R 且堆积体相当密实，则每个球体平均会有 $N = 6$ 个相互作用（这对应于恒定半径球体的最大密堆积）。如果我们使用*Ip2_FrictMat_FrictMat_FrictPhys* 类，则有 $\pi' = 2$，因为$K_N = E2R$；我们再次使用 $\xi = 0.2$ （因为没有更有意义的取值）。在这种情况下，我们得到结果

$$\frac{\Delta t_{cr}^{(p)}}{\Delta t_{cr}} = 2\sqrt{\frac{6(1-2\cdot 0.2)}{\pi/2}} = 3.02$$

这再次高估了数值临界时间步长。

总之，P 波时间步长给出的估算值与实际的 Δt_{cr}成比例，但对于所示算例，为了保证模拟稳定，应采用约为 $\Delta t = 0.3\Delta t_{cr}^{(p)}$的值。

非弹性 Δt 约束

需要指出的是，确保运动积分的数值稳定性并非唯一的约束。在颗粒以相对较高的速度运动的系统中，一个时间步长内的位置变化可能导致损伤等非弹性不可逆效应。要获得合理结果，所需的 Δt 可能更小 Δt_{cr}。对于这类情况，我们尚无严格推导出的规则。

2.1.6 周期性边界条件

虽然大多数离散元模拟发生在 $\mathbb{R}^3$ 空间中，但通常可以通过改用周期性空间来避免边界效应。为满足周期性条件，周期性空间通过重复平行六面体单元构造而成。在 Yade 中，周期性空间由 *Cell* 类实现。单元在参考坐标系中的几何形状由平行六面体的三条边定义。对应的基向量存储在矩阵 $\mathbf{H}$ (*Cell.hSize*).

初始 $\mathbf{H}$ 可在模拟开始时显式定义为一个 3x3 矩阵。对可用的形状没有任何限制：任何平行六面体都可作为初始单元。如果基向量轴对齐，那么只定义其尺寸可能比定义完整的 $\mathbf{H}$ 矩阵更方便；在这种情况下，只需定义 $\mathbf{H}$ 中的列范数（参见*Cell.size*）。

定义初始单元的几何体后，通常不应通过直接赋值来修改 $\mathbf{H}$。相反，应通过下文介绍的速度梯度 *Cell.velGrad* 来定义其变形率。只有该变量才能在本构定律和 Newton 积分器（*NewtonIntegrator*）中正确考虑周期性变形。

变形处理

单元随时间的变形通过一个张量定义，该张量表示均匀速度场 $\nabla\boldsymbol{v}$ 的梯度（*Cell.velGrad*）。此梯度表示旋转和拉伸的任意组合。它可以由外部施加，也可以由边界控制器更新（参见*PeriTriaxController* 或 *Peri3dController*），以达到目标应变值或维持指定的应力。

速度梯度会随时间自动积分，累积的变换体现在变换矩阵 $\mathbf{F}$ (*Cell.trsf*) 和单元当前的形状中 $\mathbf{H}$。每步的变换更新如下（对于 $\mathbf{H}$也类似），其中 I 为单位矩阵：

$$\mathbf{F}^{+} = (\mathrm{I} + \nabla\boldsymbol{v}\Delta\mathrm{t})\mathbf{F}^{\circ}.$$

$\mathbf{F}$ 可设为 在模拟的任意时刻恢复为单位矩阵，以便将当前状态定义为应变的参考 e 应变 边界控制器中的定义。这样做不会对 $\mathbf{H}$产生影响。

除了自动积分单元变换之外，还可以选择对全部颗粒进行相似位移，使 $\nabla\boldsymbol{v}$ 在整个模拟过程中生效（通过*Cell.homoDeform*启用）。这样可以避免速度梯度变化引起的所有边界效应。

周期单元中的碰撞检测

通常的实现会在颗粒位置越过边界时，通过 wrap 将其位置绕回，使其出现在单元的另一侧，从而强制颗粒位置保持在单元内。由于我们希望避免位置突然变化（这会导致颗粒速度与每步位移变化不一致），因此选择了另一种方法。

近似碰撞检测

第1遍碰撞检测（基于扫描与修剪算法，见第 扫描与修剪 节）针对颗粒的轴对齐包围盒（*Aabb*）进行。在碰撞检测阶段，所有*Aabb* 的边界在第一步中都会被 wrap 到单元内部。在之后的每次运行中，每个边界都会记录其初始位置相对于 *Aabb* 给定坐标的单元平移量，并在颗粒运动期间根据 *Aabb* 更新边界时反复使用此偏移量。边界通过周期性插入排序算法（见第 周期性插入排序算法 节）进行排序，该算法会追踪周期单元边界 ‖。

将两个 *Aabb* 逆序后，会检查它们沿三个坐标轴的碰撞，并为此将实际坐标 wrap 到单元内。

该算法会检测碰撞，效果如同所有颗粒都位于单元内，但无需构造“幽灵粒子”（用于表示从另一侧进入单元的颗粒的周期映像），也无需更改颗粒的位置。

实现（部分也由算法本身）要求颗粒沿任一轴的跨度不超过当前单元格尺寸的一半；原因是否则两个颗粒之间可能在两侧各出现两个（或更多）接触。由于 Yade 通过两个刚体的 *Body.id* 来识别接触，因此这些接触将无法区分。

存在剪切时，扫描与剔除碰撞检测器无法沿三个轴独立地对边界进行排序：沿 x 轴的碰撞取决于颗粒在 y 轴上的相对位置。因此，包围盒 以变换坐标表示，这些坐标在碰撞检测意义上彼此垂直。这需要一些额外计算：球体在变换坐标中的 *Aabb* 将不再是立方体，而是长方体，因为球体经过变换后会呈现为椭球体。反过来，仿真空间中的球体将具有一个平行六面体形状的包围“盒”，它是在变换后坐标轴下包围椭球体的长方体（该 *Aabb* 的坐标轴与变换后的单元基底对齐）。如图 *fig-cell-shear-aabb*所示。

随着剪切导致 *Aabb* 增大，单个颗粒不跨越变换后轴一半以上范围的限制会变得严格。考虑到半径为 r 、所在单元中 $x' \equiv x$，$z' \equiv z$，但 $\angle(y, y') = \varphi$， x-span of the *Aabb* 将乘以 $1/$ 余弦 φ。对于可能适合用于模拟的无限剪切$\varphi \to \pi/2$，有 $1/$余弦 $\varphi \to \infty$。幸运的是，通过认识到所有周期单元的准单位矩阵性质，可以轻松规避这一限制：这些周期单元若在空间中重复，会通过其角点生成相同的网格。周期单元可以翻转，同时保持所有粒子相互作用不变，如图 *fig-cell-flip*所示。只需调整相互作用的 *Interaction.cellDist* 并重新初始化碰撞器（`Collider::invalidatePersistentData`）。单元翻转已在 *utils.flipCell* 函数中实现。

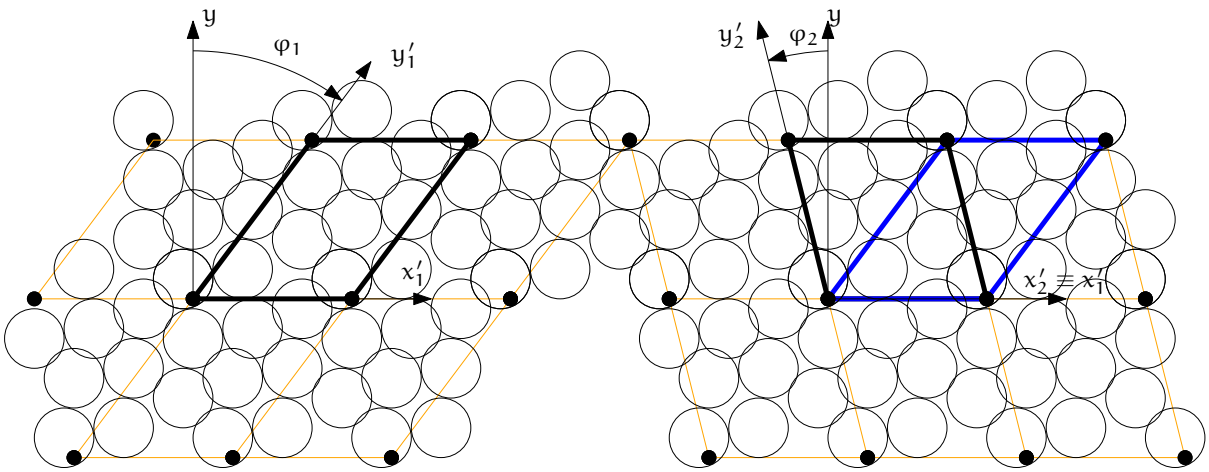


图 6：翻转单元（*utils.flipCell*），以避免随着 φ 增大而导致包围盒跨度无限拉伸。从模拟的角度来看，单元翻转不会影响相互作用。左侧的周期排列与右侧相同，只是单元在相同的重复网格点之间的位置不同；与此同时，$|\varphi_2| < |\varphi_1|$和球体包围盒的 x 跨度经 $1/\cos\ \varphi$ 拉伸后会变小。翻转可以重复进行，从而实现有效的无限剪切。

此算法在 *InsertionSortCollider* 中实现，并在模拟具有周期性时使用（*Omega.isPeriodic*）；各个 边界函子 负责计算剪切后的 ***Aabb***；目前已针对球体和面片实现（分别对应 *Bo1_Sphere_Aabb* 和 *Bo1_Facet_Aabb* ）。

精确碰撞检测

当碰撞器检测到近似接触（在*Aabb* 层级上），且接触尚不存在时，它会创建势接触，随后由精确碰撞算法进行检查（具体取决于形状的组合）。由于颗粒可以跨多个周期单元相互作用（回想一下，我们从不改变

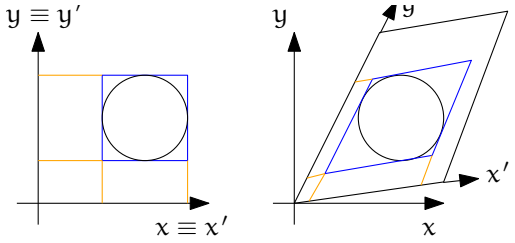


图 7：在模拟空间坐标中构建球体的轴对齐包围盒（*Aabb*）（无周期单元，左侧）以及在变换后的单元坐标中构建球体的轴对齐包围盒（右侧）；其中，碰撞检测轴 x'、y' 与模拟空间坐标轴 x、y并不相同。边界在各轴上的投影以橙色线条表示。

它们在仿真空间中的位置），碰撞器会将颗粒的相对单元坐标嵌入相互作用本身（*Interaction.cellDist*）中，作为一个整数向量 c。将当前单元格尺寸 $\mathbf{T}\mathbf{s}$ 与 c逐分量相乘，即可得到非周期性 $\Delta\mathbf{x}$ 中的粒子偏移量 R^3；该值会从*InteractionLoop*传递给计算精确碰撞的函子（*IGeomFunctor*），后者会将其加到颗粒*Interaction.id2*的位置上。

通过存储整数偏移量 c，随着单元参数变化，$\Delta\mathbf{x}$ 会自动更新。

周期性插入排序算法

将扫描与修剪算法（见 扫描与修剪）扩展到周期性边界条件并非易事。其核心是插入排序算法的一种周期变体，该算法需要跟踪每个边界的“周期”；例如，取周期 $\langle 0、10)$，则 $8_1 \equiv -2_2 < 2_2$（下标表示周期）。要高效地实现这一点（避免边界从一个周期 wrap 到另一个周期时在内存中来回移动数据），需要移动周期边界而非边界本身，并确保在容器边缘进行比较时过程透明无碍。

该算法还经过扩展，可处理非正交周期性 单元 边界，方法是在变换后的坐标（而非笛卡尔坐标）中进行运算；这会改变对刚体所在的笛卡尔坐标系中的 *Aabb*的计算（详见 近似碰撞检测）。

排序算法会跟踪 *Aabb* 沿所有坐标轴的极值点。在碰撞器初始化时，每个值都会被赋予一个整数周期，即它与单元内部的距离，以该值沿相应坐标轴的单元尺寸表示，并将其 wrap 到单元内部的值。周期数以下标表示。

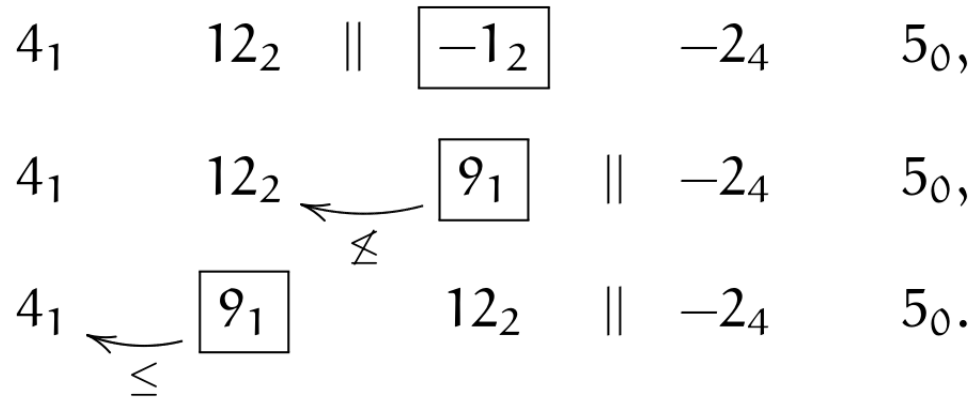
我们来看一个沿 x 轴的坐标序列示例（在实际算例中，元素数量应为偶数，因为每个颗粒都有一对最大值和最小值；不过，本例仅展示排序算法。）

$$4_1 \qquad 12_2 \quad \| \quad -1_2 \qquad -2_4 \qquad 5_0$$

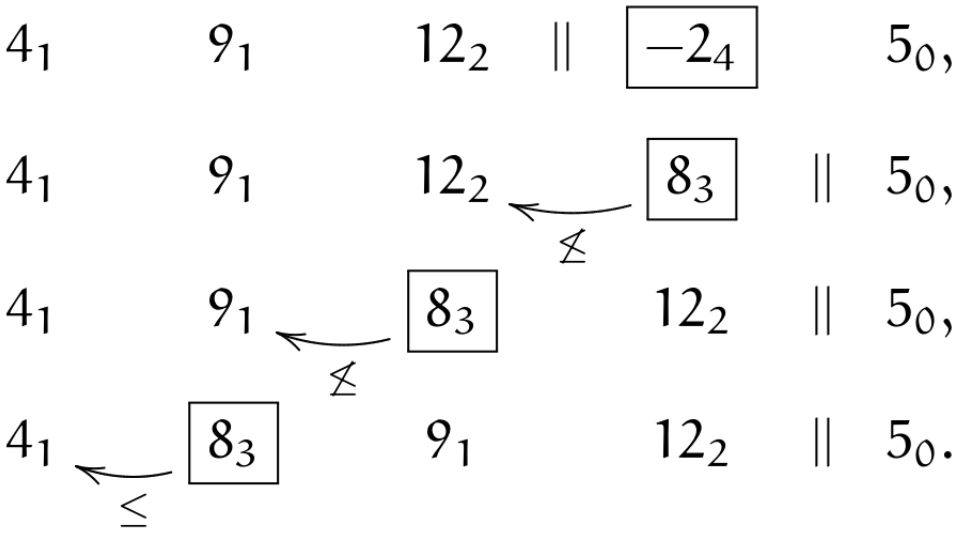
单元尺寸为 $s_x = 10$。因此，4_1 值表示该极值点的实际坐标 x_i 为$x_i + 1 \cdot 10 = 4$，即 $x_i = -4$。 $\|$ 符号表示周期单元边界。
排序从单元中的第一个元素开始，即 $\|$ 的右侧，并像非周期性变体中那样将元素逆序。不过，由于存在边界 $\|$，规则会更加复杂：

(≤)	如果相邻元素已按顺序排列，则停止反转；
(‖•)	当前元素位于 ‖ 左侧，且小于 0（下周期边界）；在这种情况下，将元素的周期减 1，将其坐标减小 s_x 并向右移动 ‖；
(•‖)	当前元素位于 ‖ 右侧，且高于 s_x （上周期边界）；将元素的周期递增，将其坐标增加 s_x，并将其向左移动 ‖；
(≰)	比较时，跨越 ‖ 的逆序必须从左侧坐标中减去 s_x。如果这些元素顺序不正确，就交换它们，但在移动过程中必须更改它们的周期 ‖. 如有必要，应用 (‖○)；
(‖○)	如果经过（≰）后，现在位于 ‖ 右侧的元素具有 $x_i < s_x$，则将其坐标减去 s_x，并将其周期减一。不要移动 ‖。

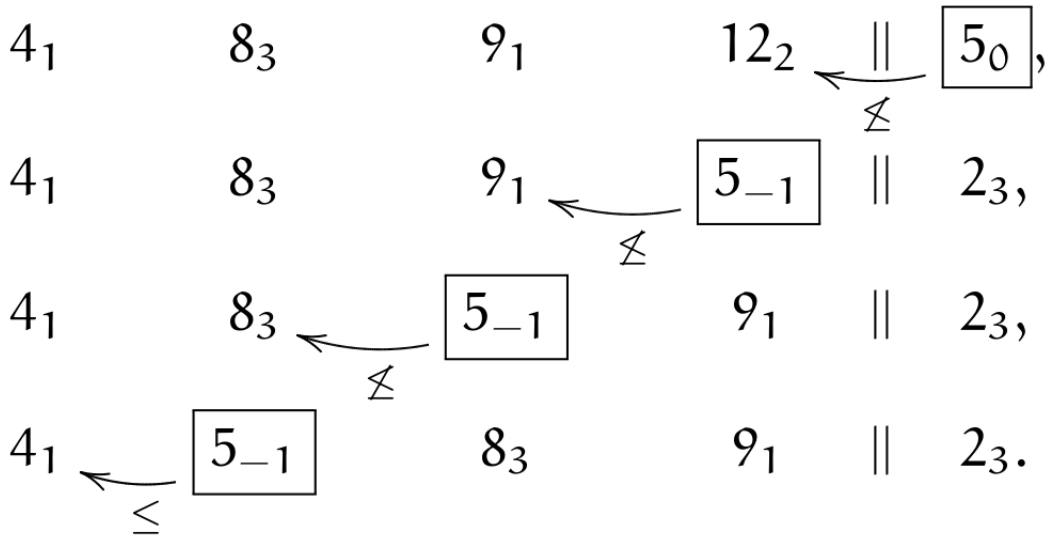
在第一步中，应用 (‖•)，并与 12_2 发生逆序；然后由于 (≤)，我们停止：



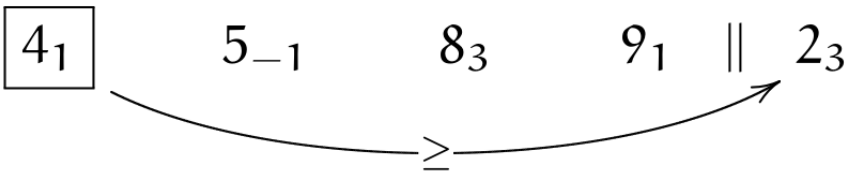
我们转到下一个元素 -2_4；首先应用（‖•），然后反转，直到（≤）：



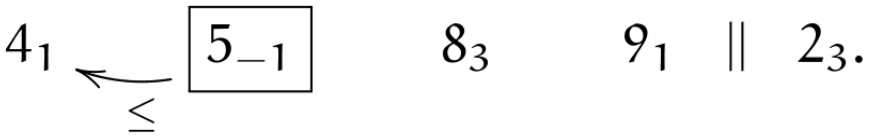
下一个元素是 5_0；我们满足 (≰)，因此不能比较 $12_2 > 5_0$，而必须执行 $(12_2 - s_x) = 2_3 \leq 5$；交换 ‖ 时，我们调整周期并应用 (‖○)，将 12_2 变为 2_3；然后继续求逆，直到(≤)：



我们（绕回）移动到 4_1 , , 其顺序如下：



最后一个元素也是如此



2.1.7 计算方面

Cost

与隐式积分方案相比，采用显式积分方案的离散元计算在模拟过程中需要相对较多的步。模拟时长为 Z of simulation spanningT 秒（模拟时间），包含 N 个颗粒、体积为 V 的模拟的总计算时间取决于：

- 与步数成线性关系 $i = T/(s_t \Delta t_{cr})$，其中 s_t 是时间步安全系数； Δt_{cr} 可利用 P 波速度通过以下方式估算 E 和 ρ （第 根据波传播速度估算 *Dtcr*）作为 $\Delta t^{(p)}_{cr} = r\sqrt{\frac{\rho}{E}}$。因此

$$i = \frac{T}{s_t r}\sqrt{\frac{E}{\rho}}.$$

- 颗粒数量 N；在模拟域体积 V 和颗粒半径 r固定的情况下

$$N = p\frac{V}{\frac{4}{3}\pi r^3},$$

其中 p 是堆积孔隙率，对于半径相近的球体密集不规则堆积体，该值约为 $1\,2$ 。

这种依赖关系并非严格线性（线性是最理想的情况），因为某些算法的缩放并非线性；一个例子是第 扫描与剪枝节中介绍的碰撞检测算法，其缩放大致为 $\mathcal{O}(\text{N对数 N})$。

只要堆积特性相同，相互作用的数量就会随 N 缩放。

- 计算核心的数量 $n_{\text{中央处理器}}$；理想情况下，如果所有算法都实现了并行化（在 Yade 中，碰撞检测并未并行化），其依赖关系将呈逆线性。

假设呈线性缩放。此外，假设要模拟的材料（E、 ρ）和仿真设置（V、 T）已预先给定。最后，无量纲常数 s_t、 p 和 n_{cpu} 均为固定值。这样只剩下最后一个自由度： r。我们可以写成

$$Z \propto iN\frac{1}{n_{cpu}} = \frac{T}{s_t r}\sqrt{\frac{E}{\rho}}p\frac{V}{\frac{4}{3}\pi r^3}\frac{1}{n_{cpu}} \propto \frac{1}{r}\frac{1}{r^3} = \frac{1}{r^4}.$$

这个（相当显而易见的）结果对于理解离散元法缩放至关重要；如果我们想得到更精细的结果，将“网格”细化一倍，即把 r 减半，计算时间将增长 $2^4 = 16$ 倍。

对于非常粗略的估算，可以利用已知的模拟结果来获得计算机的“常数”

$$\mu = \frac{Z}{Ni}$$

表示每个颗粒、每个时间步长所需的时间（对于当前的计算机，数量级为 10^{-6}s）。μ 只有在模拟特征相似且缩放中的非线性效应没有重大影响时才有用；也就是说， N应与参考算例处于相同的数量级。

结果非确定性

自然会认为，多次运行同一个模拟会得到完全相同的结果：尽管计算采用有限精度，舍入误差在每次运行中也应相同且确定。对于单线程计算而言，情况确实如此，因为所有操作的确切顺序由模拟本身决定；但对于多线程计算，情况就不再如此，后文将对此进行详细说明。

显式离散元中最直接的并行处理方式，是任意顺序处理相互作用。每个相互作用的应变和应力都是独立计算的，但必须将各相互作用产生的力相加。如果求和顺序也是任意的（在 Yade 中，每个线程按相互作用的处理顺序累积力，然后再将各线程的力相加），那么结果可能会略有不同。例如

```
(1/10.)+(1/13.)+(1/17.)=0.23574660633484162
(1/17.)+(1/13.)+(1/10.)=0.23574660633484165
```

由于相互作用产生的力以近似随机的顺序分配给刚体，因此，单线程计算与多线程计算所得的刚体合力 F_i 可能不同；即便多线程计算的参数完全相同，不同次运行的结果也可能不同。内核的确切线程调度无法预测，因为它取决于异步事件（硬件中断）以及系统上运行的其他无关任务；而线程调度最终决定了相互作用产生的力的求和顺序。

2.2 用户手册

2.2.1 场景构建

添加颗粒

该*BodyContainer* 保存刚体对象，用于模拟；可通过以下方式访问： `O.bodies`。

创建 Body 对象

刚体对象很少由其组成部分（形状、*Bound*、状态、材料）手动构造；通常会使用便捷函数 球体、面片和墙来创建它们。使用这些函数还能确保更好的向后兼容性，以防刚体的内部结构发生变化。这些函数接收颗粒的几何形状以及其他若干特征。详情请参阅其文档。如果多个（或许多）刚体使用相同的材料，可以按照下文所述，将其添加到 `O.materials`中以供共享。

定义材料

该 `O.materials` 对象（*Omega.materials*的实例）存放为刚体定义的共享材料。它仅支持添加操作，通常只包含少量实例（但数量不受限制）。

`label` 赋予每种材料的标签是可选的，但可以传递给球体和其他用于构造刚体的函数。由 `O.materials.append` 返回的是该材料的 `id`，也可以传递给球体——这种方式比使用标签稍快一些，不过对于颗粒数量较少的情况，速度差异并不明显，而且可能不那么方便。

如果调用球体时未指定材料，则使用最后一个定义的材料；这是一种方便的默认设置。如果尚未定义材料（因此不存在最后一个材料），则会创建默认材料：FrictMat(density=1e3,young=1e7,poisson=.3,frictionAngle=.5)。严肃的模拟中不应出现这种情况，但在简单脚本中很方便，因为此时精确的材料属性通常并不重要。

```
Yade [1]: len(O.materials)
Out[1]: 0

Yade [2]: idConcrete=O.materials.append(FrictMat(young=30e9,poisson=.2,frictionAngle=.6,label=
 →"concrete"))

Yade [3]: O.materials[idConcrete]
Out[3]: <FrictMat instance at 0x3f01be0>

# uses the last defined material
Yade [4]: O.bodies.append(sphere(center=(0,0,0),radius=1))
Out[4]: 0

# material given by id
Yade [5]: O.bodies.append(sphere((0,0,2),1,material=idConcrete))
Out[5]: 1

# material given by label
Yade [6]: O.bodies.append(sphere((0,2,0),1,material="concrete"))
Out[6]: 2

Yade [7]: idSteel=O.materials.append(FrictMat(young=210e9,poisson=.25,frictionAngle=.8,label=
 →"steel"))

Yade [8]: len(O.materials)Out[8]: 2
# implicitly uses "steel" material, as it is the last one now
Yade [9]: O.bodies.append(facet([(1,0,0),(0,1,0),(-1,-1,0)]))
Out[9]: 3
```

添加多个颗粒

如上所示，使用 `append` 方法逐个或一次添加多个刚体：

```
Yade [10]: O.bodies.append(sphere((0,10,0),1))
Out[10]: 0

Yade [11]: O.bodies.append(sphere((0,0,2),1))
Out[11]: 1

# this is the same, but in one function call
Yade [12]: O.bodies.append([
   ....:    sphere((0,0,0),1),
   ....:    sphere((1,1,3),1)
   ....: ])
   ....:
Out[12]: [2, 3]
```

下一节介绍的许多函数会返回刚体列表，可直接将其添加到模拟中，包括

- 堆积生成器，例如 *pack.randomDensePack*、 *pack.regularHexa*
- 表面函数 *pack.gtsSurface2Facets*
- 导入函数 *ymport.gmsh*、 *ymport.stl*、.....

由于这些函数在内部使用 球体 和 面片 ，因此它们接受传递给这些函数的其他参数。具体而言，每个刚体的材料将按照上述规则进行选择（未指定时使用最后一个材料、按标签选择、按索引选择等）。

颗粒团簇

在某些情况下，你可能需要将单个颗粒组成刚性团聚体（即在模拟过程中，颗粒之间的相对位置保持不变）。我们称之为团簇。团簇在内部由一种特殊的刚体表示，其成员通过团簇 *ID* 引用（另请参见 *isClump*、*isClumpMember* 和*isStandalone*）。与所有刚体一样，团簇也有一个位置，即所有成员的（质量）平衡点。团簇刚体本身与其他刚体之间没有相互作用。团簇之间的相互作用由团簇成员之间的相互作用表示。同一团簇的成员之间不存在相互作用。

YADE 支持多种创建团簇的方式：

- 使用一条命令直接创建团簇和球体（团簇成员）：

函数*appendClumped()* 专为此任务设计。例如，我们可以添加两个绑在一起的球体：

```
Yade [13]: O.bodies.appendClumped([
   ....:     sphere([0,0,0],1),
   ....:     sphere([0,0,2],1)
   ....: ])
   ....:
Out[13]: (2, [0, 1])

Yade [14]: len(O.bodies)
Out[14]: 3

Yade [15]: O.bodies[1].isClumpMember, O.bodies[2].clumpId
Out[15]: (True, 2)

Yade [16]: O.bodies[2].isClump, O.bodies[2].clumpId
Out[16]: (True, 2)
```

-> *appendClumped()* 返回标识符元组 (clumpId,[memberId1,memberId2,...])

- 使用现有球体并将它们团簇在一起：

对于此算例，可以将函数 *clump()* 应用于现有刚体的列表：

```
Yade [17]: bodyList = []

Yade [18]: for ii in range(0,5):
   ....:    bodyList.append(O.bodies.append(sphere([ii,0,1],.5)))#create a "chain" of 5 spheres
   ....:

Yade [19]: print(bodyList)
[0, 1, 2, 3, 4]

Yade [20]: idClump=O.bodies.clump(bodyList)
```

-> *clump()* 返回 clumpId

- 另一种方法是使用团簇模板，将给定堆积体中的独立球体（参见*SpherePack* 和*make-Cloud*）替换为团簇。

这可以通过一个名为*replaceByClumps()*的函数来实现。该函数接收一个*clumpTemplates()* 列表和一个数量列表，并将球体替换为团簇。新团簇的体积将与被替换球体的体积相同（计算团簇的体积、质量和惯性时会考虑重叠，并假定只有成对重叠）。

-> *replaceByClumps()*返回一个元组列表： `[(clumpId1,[memberId1,memberId2,...]),(clumpId2,[memberId1,memberId2,...]), ...]`

还可以向团簇中添加刚体，并从团簇中释放刚体。也可以删除团簇（其成员将变为独立刚体）。

此外，YADE 还允许实现团簇的圆度或堆积体的圆度系数。可以通过排除列表将堆积体的部分区域排除在圆度测量之外。

```
Yade [21]: bodyList = []Yade [22]: for ii in range(1,5):
....:    bodyList.append(O.bodies.append(sphere([ii,ii,ii],.5)))
....:Yade [23]: O.bodies.clump(bodyList)Out[23]: 4
Yade [24]: RC=O.bodies.getRoundness()Yade [25]: print(RC)
0.25619141423166986
```

-> *getRoundness()* 返回堆积体或其部分区域的圆度系数 RC

注意： 请查看examples/clumps/ 文件夹。里面有一些示例，展示了团簇相关的不同函数的用法。

球体堆积

用球体排列来表示任意形状的实体，会产生球体堆积问题，即如何在空间中排列球体，使其近似填充给定实体。对于离散元模拟，可能有若干要求。

1. 球体半径分布。只要不限制球体半径，任意体积都可以被球体完全填满；在这种情况下，球体数量可以是无穷的，而球体半径趋近于零。由于颗粒数量和最小球体半径（通过临界时间步长）都会决定计算成本，因此必须指定半径分布。最

典型的分布是均匀分布：平均值±离散度；如果离散度为零，所有球体的半径都相同。2. 光滑边界。有些算法以一种让球体沿边界排列的方式处理边界，从而使边界表面更加光滑。3. 堆积密度，即球体体积与实体尺寸之比。它与半径分布密切相关。4. 配位数，即每个球体的平均接触数。5. 各向同性（与规则性/不规则性相关）；通常不希望堆积体存在优先方向，除非所建模的实体也具有这种方向偏好。6. 允许的球体重叠；有些算法可能生成球体略微重叠的堆积体。由于重叠通常会在离散元法中产生力，因此无重叠的堆积体有时称为“无应力 。

体积表示

在 Yade 中，有两种方法可表示相关实体的精确体积：边界表示法和构造实体几何。尽管二者存在根本差异，但在 Yade 中都被抽象为 谓词 类。Predicate 提供以下功能：

1. 为关联的实体定义轴对齐包围盒（也可选择定义定向包围盒）；2. 判断给定点位于实体内部还是外部；大多数谓词还可以（精确或近似地）判断该点是否在实体内部，并且 与所表示实体的边界保持指定的填充距离（使该体积的球体不会突出实体之外）。

构造实体几何（CSG）

CSG 方法通过几何基本实体或基本几何体（球体、圆柱体、盒子、圆锥等）以及它们之间的布尔运算来描述体积。Yade 中定义的基本实体包括 *inCylinder*、*inSphere*、*inEllipsoid*、*inHyperboloid*、*notInNotch*。

例如，双曲面体（狗骨形）拉压试验试样可以通过这种方式构建（如图 *img-hyperboloid*所示）：

```
from yade import pack

## construct the predicate first
pred=pack.inHyperboloid(centerBottom=(0,0,-.1),centerTop=(0,0,.1),radius=.05,skirt=.03)
## alternatively: pack.inHyperboloid((0,0,-.1),(0,0,.1),.05,.03)

## pack the predicate with spheres (will be explained later)
spheres=pack.randomDensePack(pred,spheresInCell=2000,radius=3.5e-3)

## add spheres to simulation
O.bodies.append(spheres)
```

边界表示（BREP）

用边界表示实体比使用 CSG 体积灵活得多，但通常只能近似表示。Yade 提供了与 GNU 三角剖分曲面库 (GTS) 的接口，既可导入 GTS 可读取的曲面，也可直接在模拟脚本中显式构造曲面。这样便能参数化构造相当复杂的形状；其中提供了一些函数，可将二维折线生成一组三维折线（*pack.revolutionSurfaceMeridians*），并对这组三维折线之间的曲面进行三角剖分（*pack.sweptPolylines2gtsSurface*）。

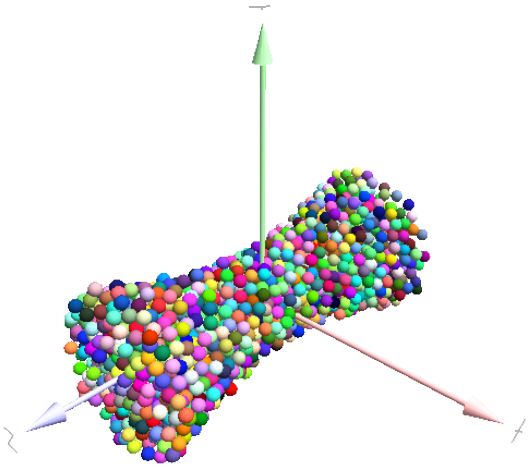


图 8：使用 *pack.inHyperboloid* 谓词构造试样，并使用*pack.randomDensePack*打包。

例如，我们可以构造一个简单的漏斗（examples/funnel.py，见 *img-funnel* ）：

```
from numpy import linspacefrom yade import pack# angles for points on circles
thetas=linspace(0,2*pi,num=16,endpoint=True)
# creates list of polylines in 3d from list of 2d projections# turned from 0 to π
meridians=pack.revolutionSurfaceMeridians(
[[(3+rad*sin(th),10*rad+rad*cos(th)) for th in thetas] for rad in linspace(1,2,
→num=10)],linspace(0,pi,num=10))# create surfacesurf=pack.sweptPolylines2gtsSurface(
meridians+[[Vector3(5*sin(-th),-10+5*cos(-th),30) for th in thetas]]  # add funnel top)
# add to simulation0.bodies.append(pack.gtsSurface2Facets(surf))
```

GTS 曲面对象可用于两种用途：

1. *pack.gtsSurface2Facets* 函数可在模拟中创建三角剖分曲面（由 面片 颗粒生成），如漏斗示例所示。（也可以使用 *ymport.stl*直接从 STL 文件导入三角剖分曲面。）2. *pack.inGtsSurface* 谓词可使用曲面作为封闭体积的边界表示来创建。

examples/gts-horse/gts-horse.py（图 *img-horse*）展示了这两种用法；首先，导入一个 GTS 曲面：

```
import gts
surf=gts.read(open('horse.coarse.gts'))
```

该表面对象用作堆积体的谓词：

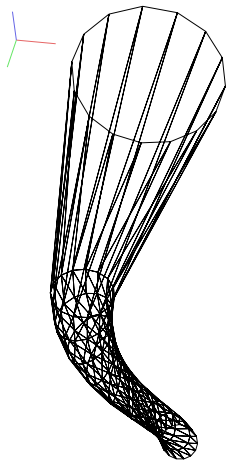


图 9：使用 examples/funnel.py 脚本构建的三角剖分漏斗。

```
pred=pack.inGtsSurface(surf)
aabb=pred.aabb()
radius=(aabb[1][0]-aabb[0][0])/40
O.bodies.append(pack.regularHexa(pred,radius=radius,gap=radius/4.))
```

随后，将其平移，并作为模拟本身中三角剖分曲面的底数：

```
surf.translate(0,0,-(aabb[1][2]-aabb[0][2]))
O.bodies.append(pack.gtsSurface2Facets(surf,wire=True))
```



图 10：导入的 GTS 曲面（马）用作填充谓词（顶部），由面片 构建的表面（底部）。请参阅 http://www.youtube.com/watch?v=PZVruIlUX1A 观看此模拟的影片。

谓词上的布尔运算

一对谓词（记为 `A` 和 `B`）上的布尔运算定义如下：

- 交集 `A & B`（合取）：点必须同时属于两个谓词。
- 并集 `A | B`（析取）：点必须属于第一个谓词或第二个谓词。
- 差集 `A - B`（与第二个谓词取反后的合取）：点必须属于第一个谓词且不属于第二个谓词。
- 对称差 `A ^ B`（异或）：点必须恰好属于两个谓词中的一个。

组合谓词也能正确地定义其包围盒。例如，我们可以取一个盒子，并从其内部减去圆柱体，使用 `A - B` 操作（图*img-predicate-difference*）：

```
pred=pack.inAlignedBox((-2,-2,-2),(2,2,2))-pack.inCylinder((0,-2,0),(0,2,0),1)
spheres=pack.randomDensePack(pred,spheresInCell=2000,radius=.1,rRelFuzz=.4,
 →returnSpherePack=True)
spheres.toSimulation()
```

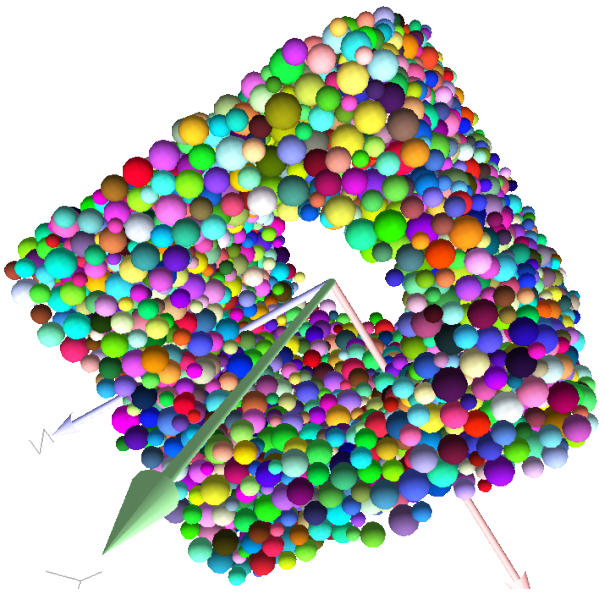


图 11：使用这两个谓词的差集从盒子内部移除圆柱体 .

堆积算法

下面介绍的算法作用于由中心和半径定义的几何球体。除下文记录的少数例外情况外，过程如下：

1. 计算球体的位置和半径（有些函数使用体积谓词来计算，有些则不使用） 2. 对每个计算出的位置和半径调用球体函数；它会接收堆积函数的额外关键字参数（即堆积函数定义中未指定的参数；这些参数会在 `**kw`中注明）。每次球体函数调用都会创建实际的刚体对象，其形状为球体形状。返回刚体对象列表。3. 堆积函数返回的列表可使用*toSimulation()*添加到模拟中。旧代码使用*O.bodies.append*调用。

以穿孔盒为例：

```
pred=pack.inAlignedBox((-2,-2,-2),(2,2,2))-pack.inCylinder((0,-2,0),(0,2,0),1)
spheres=pack.randomDensePack(pred,spheresInCell=2000,radius=.1,rRelFuzz=.4,wire=True,color=(0,0,1),material=1,
returnSpherePack=True)
```

关键字参数 `wire`， `color` 和 `material` 未在 *pack.randomDensePack*中声明，因此会传递给 球体，相关说明也见该处。 `spheres` 现在是一个 *SpherePack* 对象。

```
spheres.toSimulation()
```

下文介绍的堆积算法会生成致密堆积。如果需要疏松堆积，*SpherePack*类提供了用于生成疏松堆积的函数，可通过其 *makeCloud()* 方法调用。该方法在内部用于动力学算法中初始构型的生成。例如：

```
from yade import pack
sp=pack.SpherePack()
sp.makeCloud(minCorner=(0,0,0),maxCorner=(3,3,3),rMean=.2,rRelFuzz=.5)
```

会用球体填满给定的盒子，直到无法再放入更多球体。该对象可用于向模拟中添加球体：

```
sp.toSimulation()
```

几何

几何算法无需进行动力学模拟即可计算堆积体，其优点包括

- 速度；
- 球体恰好相切，不存在重叠（有些人称之为“无应力”堆积体）；

它们的主要缺点是，除特定情况（规则堆积）外，无法精确指定半径分布；球体半径由算法给出，这也就意味着系统已被确定。如果你的问题对精确的半径分布有要求，可以考虑改用动力学算法。

规则

Yade 定义了用于生成恒定半径球体堆积体的生成器，可与上述体积谓词配合使用。它们是致密正交堆积（*pack.regularOrtho*）和致密六方堆积（*pack.regularHexa*）。后者会生成所谓的“六方最密堆积”，可达到最大密度（http://en.wikipedia.org/wiki/Close-packing_的_球体）。

规则堆积的一个明显缺点是，它们具有很强的方向偏好，在某些情况下这可能并不构成问题。

不规则

随机几何算法与上述体积谓词完全不集成；相反，它们采用自身的边界/体积定义，并在球体定位过程中使用。另一方面，这使它们能够遵循边界，也就是让球体在适当的位置与边界接触，而不是在两者之间留下空隙。

GenGeo 是用于堆积生成的库（Python 模块），由 ESyS-Particle 开发。它通过随机插入具有给定半径范围的球体来创建堆积体。插入的球体彼此恰好接触；更重要的是，如果球体位于边界附近，它们也会与边界接触。边界由 GenGeo 库中的特殊对象表示（球体、圆柱体、盒子、凸多面体等）。因此，GenGeo 不能用于由上文所述的 yade 谓词表示的体积。

通过此模块生成的堆积体可以直接通过 *ymport.gengeo* 导入，也可以通过*ymport.gengeoFile* 从已保存的文件导入。示例脚本位于 examples/test/genCylLSM.py。GenGeo 的完整文档可在 ESyS 文档网站 上找到。

有 Debian 软件包 esys-particle 和 python-demgengeo。

动态

用于随机致密堆积的最通用算法由 *pack.randomDensePack*提供。首先，通过在长方体体积中随机放置球体，生成由请求的半径分布（目前仅支持均匀分布）指定半径的无重叠球体疏松堆积。当无法再插入更多球体时，通过运行离散元模拟对堆积体进行压缩，然后再解除压缩（这些“应力”的确切值见 py/pack/pack.py ）；使用 *Omega.switchScene* 可避免影响现有模拟。最后，使用提供的谓词裁剪所得堆积体，如上所述。

从其本　质上说，这种方法可能耗时较长；有两项设置可使计算　　n
时间较短 er:

- 如果指定了球体数量 `spheresInCell` 参数，则只会创建一个较小的试样，并采用周期性 边界，然后重复该试样以填充谓词。根据 `spheresInCell` 参数（建议取几千），这样可以获得规则性较低、质量较高的堆积体。

- 提供 `memoizeDb` 参数后，*pack.randomDensePack* 会先在指定文件（SQLite 数据库）中查找参数相似的堆积体。如果找到，则直接从数据库读取并返回该堆积体。如果数据库中不存在相似的堆积体，则按常规流程运行，并在返回结果前将其保存到数据库中，因此后续使用相同参数调用时会很快返回结果。

如果需要获得完整的周期性堆积体（而不是由谓词裁剪的堆积体），可以使用*pack.randomPeriPack*。

如有特殊需求，也可以亲自“手动”创建堆积体。例如，可以用面片构造填充边界，让随机放置在空间中的球体在重力作用下落下。

三角剖分曲面

Yade 集成了 GNU 三角剖分曲面库，并通过 GTS 模块在 Python 中提供接口。GTS 提供了多种曲面操作函数（粗化、剖分、简化、导入），详情请参阅其文档。

GTS 曲面是几何对象，可作为一组颗粒插入模拟中，这些颗粒的*Body.shape* 类型为 面片 ——单个三角剖分元素。*pack.gtsSurface2Facets* 可用于将 GTS 曲面的三角剖分转换为刚体列表，以便通过 `O.bodies.append` 插入模拟中。

默认情况下，Facet 粒子创建为非*Body.dynamic* （其惯性质量为零）。这意味着它们固定在空间中，受到力作用时也不会移动。不过，您可以

- 使用 部分引擎 为面片指定任意运动（例如 平移引擎 或 旋转引擎）；

- 显式指定该颗粒的质量和惯性；
- 将该颗粒归入一个团簇，并为团簇本身指定质量和惯性（如下所述）。

注意： 面片目前只能与球体相互作用，不能与其他面片相互作用，即使它们是动态的。两个面片发生碰撞不会产生相互作用，因此面片上不会受到任何力。

导入

Yade 目前提供 3 种格式，可从外部文件导入三角剖分曲面，相关功能位于*ymport*模块中：

ymport.gts GTS 原生格式的文本文件。

ymport.stl 立体光刻格式，可以是文本或二进制形式；可从 Blender 导出，也可从许多 CAD 系统导出。

ymport.gmsh。 GMSH 原生格式的文本文件，适用于 GMSH，这是一款广受欢迎的开源网格划分程序。

如果需要在创建面片列表之前处理曲面，可以查看 py/ymport.py 文件，其中定义了导入函数。大多数情况下，这些函数都相当简单。

参数化构造

GTS 模块提供了一种便捷的方法，可通过顶点、边和三角形创建表面。

不过，通常可以方便地将表面描述为空间中两条折线之间的表面。例如，圆柱体是两个多边形（闭合折线）之间的表面。 *pack.sweptPolylines2gtsSurface*提供了通过三角剖分连接多条折线的功能。

注意： pack.sweptPolylines2gtsSurface 的实现相当简单：所有折线必须具有相同的长度，并且连接时会按照各折线中点的索引（而非距离）在点之间生成三角形。另一方面，如果 `threshold` 参数为正数，则点可以重合：距离小于 `threshold` 的顶点所构成的退化三角形会自动移除。

若要高效（从代码长度角度而言）地操作列表，需要熟悉 Python 中的列表推导式。

另一个 e 示例可在examples/mill.py （完全参数化）或examples/funnel.py（采用 h 硬编码 数字）。

创建相互作用

在典型算例中，颗粒在模拟过程中发生碰撞时会创建相互作用。具体来说，碰撞器会检测颗粒之间的近似接触，然后由*IGeomFunctor* 检测精确碰撞。
某些材料模型（例如混凝土模型）依赖于比球体几何接触更密集的初始相互作用网络：将球体的半径乘以一个称为**相互作用半径**（或**相互作用比**）的无量纲因子，以创建此初始网络。通常按以下方式进行（示例见examples/concrete/uniax.py）：

1. 调整近似碰撞检测，使其也能检测相互作用半径内颗粒之间的近似接触。具体做法是将 *Bo1_Sphere_-Aabb.aabbEnlargeFactor* 的值设为相互作用半径。

2. 如果输入未接触的两个球体，几何函子（`Ig2`）通常会返回“没有接触”。因此，必须为其指定与 *Bo1_Sphere_Aabb.aabbEnlargeFactor* 相同的值（*Ig2_Sphere_Sphere_ScGeom.interactionDetectionFactor* ）。

   请注意，仅球体 + 球体 相互作用已支持；在 *distFactor* 中没有类似的参数 *Ig2_Facet_Sphere_ScGeom*。这是有意为之，因为相互作用半径对于以球体堆积表示的块体材料有意义，而面片通常表示边界条件，不应纳入这种密集的相互作用网络。

3. 运行一个模拟步，以创建初始网络。
4. 将 `Bo1` 和 `Ig2` 函子的相互作用半径重新重置为其默认值。

5. 继续模拟；已经建立的相互作用不会被删除（前提是正在使用的 Law2函子允许这样做）。

In 代码中，这种情形可能看起来与此类似（标记的说明见标记内容 ）：

```
intRadius=1.5
damping=0.05

O.engines=[
   ForceResetter(),
   InsertionSortCollider([
      # enlarge here
      Bo1_Sphere_Aabb(aabbEnlargeFactor=intRadius,label='bo1s'),
      Bo1_Facet_Aabb(),
        ]),
   InteractionLoop(
      [
         # enlarge here
         Ig2_Sphere_Sphere_ScGeom(interactionDetectionFactor=intRadius,label='ig2ss'),
         Ig2_Facet_Sphere_ScGeom(),
      ],
      [Ip2_CpmMat_CpmMat_CpmPhys()],
      [Law2_ScGeom_CpmPhys_Cpm(epsSoft=0)],   # deactivated
   ),
   NewtonIntegrator(damping=damping,label='damper'),
]

# run one single step
O.step()

# reset interaction radius to the default value
bo1s.aabbEnlargeFactor=1.0
ig2ss.interactionDetectionFactor=1.0

# now continue simulation
O.run()
```

按需创建单个相互作用

可以使用*createInteraction*独立于碰撞检测，在一对颗粒之间创建相互作用。此函数会查找并使用匹配的 Ig2 和 Ip2 函子。无论给定颗粒之间的距离如何，都会创建相互作用（通过向 Ig2 函子传递一个特殊参数，即使没有几何接触也强制创建相互作用）。应使用适当的本构定律，以避免在下一个模拟步中删除该相互作用。

```
Yade [26]: O.materials.append(FrictMat(young=3e10,poisson=.2,density=1000))
Out[26]: 0Yade [27]: O.bodies.append([
....:     sphere([0,0,0],1),
....:     sphere([0,0,1000],1)
....: ])
....:Out[27]: [0, 1]

# only add InteractionLoop,  no other engines are needed now
Yade [28]: O.engines=[
....:     InteractionLoop(
....:         [Ig2_Sphere_Sphere_ScGeom(),],
....:         [Ip2_FrictMat_FrictMat_FrictPhys()],
....:         [] # not needed now (
```

conti下页继续)

（接上页）

```
   ....:     )
   ....: ]
   ....:

Yade [29]: i=createInteraction(0,1)

# created by functors in InteractionLoopYade [30]: i.geom, i.phys
Out[30]: (<ScGeom instance at 0x37acbe0>, <FrictPhys instance at 0x3ed45b0>)
```

如果要创建许多相互作用，这个方法会相当慢（每次创建相互作用时都会重复查找函子）。这种情况下，请在 yade-dev@lists.launchpad.net上提出建议，让 *createInteraction*函数也接受 id 对列表。

基础引擎

Yade 中典型的离散元模拟在每一步至少会执行以下操作（详情请参阅 函数组件 ）：

1. 重置前一步的力 2. 检测新的碰撞 3. 处理相互
作用 4. 对颗粒施加力并更新其位置

这些点分别对应一个或多个引擎：

```
O.engines=[
   ForceResetter(),          # reset forcesInsertionSortCollider([...]),
     # approximate collision detection
   InteractionLoop([...],[...],[...]) # handle interactions
   NewtonIntegrator()         # apply forces and update positions
]
```

引擎的顺序很重要。在大多数情况下，任何额外的引擎都应放在 交互循环之后：

- 如果它会施加力，就应放在 *NewtonIntegrator*之前，否则力将永远不会生效。
- 如果它会使用刚体的位置，也应放在 *NewtonIntegrator*之前，否则将使用下一步的位置（在许多情况下这可能并不重要，例如使用 *VTKRecorder*进行可视化输出）。

必须一次性指定 *O.engines* 序列（原因在于，尽管引擎本身是按引用传递的，但该序列会从 C 复制 到 Python，或从 Python 复制到 C++ ++）。这包括修改现有的 `O.engines`；因此

```
O.engines.append(SomeEngine()) # wrong
```

不会生效；

```
O.engines=O.engines+[SomeEngine()] # ok
```

必须改用 。例如，要在位置 #2 之后插入一个引擎，请使用 Python 切片语法 n:

```
O.engines=O.engines[:2]+[SomeEngine()]+O.engines[2:]
```

注意： Yade 启动时，O.engines 会填入一份合理的默认列表，因此在尝试简单操作时，并非一定要重新定义它。默认场景会正确处理球体、盒体和

面片的摩擦属性，并动态调整时间步长。你可以在examples/simple-scene/simple-scene-default-engines.py中找到一个示例。

函子的选择

在上面的例子中，我们省略了函子，只写了椭圆... 而已。如分派器和函子中所述，函子及其关联的分派器共有 4 种。用户可以选择使用哪些函子，但选择必须保持一致。

Bo1 函子

`Bo1` 必须根据所用的碰撞器来选择函子；函子直接传递给碰撞器（碰撞器内部使用 *BoundDispatcher*）。

目前（2019年1月），最常见的选择是 *InsertionSortCollider*，它使用 *Aabb*；在这种情况下，必须使用创建 *Aabb* 的函子。根据模拟中的颗粒形状，选择合适的函子：

```
O.engines=[...,
InsertionSortCollider([Bo1_Sphere_Aabb(),Bo1_Facet_Aabb()]), ...
...]
```

使用超过必要数量的函子（例如，当模拟中没有*Bo1_面片__Aabb* 时）不会影响性能。另一方面，如果现有形状缺少对应的函子，这些刚体将无法与其他刚体发生碰撞（它们会自由地相互穿透）。

此外还有其他碰撞器，不过它们目前仅用于实验：

- *SpatialQuickSortCollider* 是一种用于验证正确性的碰撞器，基于*Aabb*运行；其速度明显慢于*InsertionSortCollider*。
- *PersistentTriangulationCollider* 仅适用于球体；它不使用*BoundDispatcher*，因为它直接对球体进行操作。
- *FlatGridCollider* 是一种概念验证性质的基于网格的碰撞器，会在内部计算网格位置（也不使用 *BoundDispatcher* ）

Ig2 函子

`Ig2` 函子选择（它们全部都派生自*IGeomFunctor*）取决于

1. 应当发生碰撞的形状组合；例如：

   ```
   InteractionLoop([Ig2_Sphere_Sphere_ScGeom()],[],[])
   ```

   将处理球体 + 球体之间的碰撞，但不会处理面片 + 球体之间的碰撞——如果需要处理这种情况，则必须使用一个额外的函子：

   ```
   InteractionLoop([
      Ig2_Sphere_Sphere_ScGeom(),
      Ig2_Facet_Sphere_ScGeom()
   ],[],[])
   ```

   同样，缺少组合会导致给定的形状组合自由地相互穿透。对于每一对形状，函子有多种可能的选择，因此不能默认将它们放入*InsertionSortCollider* 中。刚体彼此穿过的一个常见错误是没有添加必要的函子。

2. *IGeom* 类型，由 Law2 函子（下文）接受；它是函子名称中 Law2（例如，*定律2_ScGeom_CpmPhys_Cpm* 接受*ScGeom*）。

Ip2 函子

Ip2 函子（派生自 *IPhysFunctor*）必须根据以下因素进行选择：

1. 模拟中的材料 组合。在大多数情况下， Ip2 函子处理两个相同 材料 类的实例（例如 *Ip2_FrictMat_FrictMat_FrictPhys* 用于两个采用 *FrictMat* 的刚体）
2. *IPhys* 被本构定律（Law2 函子）接受，这是 Law2 函子名称的第二部分（例如 *Law2_ScGeom_FrictPhys_CundallStrack* 接受 *FrictPhys*）

注意： 与 Bo1 和 Ig2 函子不同，未处理的 材料 组合会触发异常，属于错误情况。

Law2 函子

Law2 函子是选择 Ig2 和 Ip2 函子的最终依据；函子本身的选择不受限制，因为它只施加力，不会创建新对象。
在大多数模拟中，只会使用一个 Law2 函子；不过，也可以使用多个函子，并根据 *IGeom* 和 *IPhys* 的组合进行分派。这些对象分别由 Ig2 和 Ip2 函子生成（而这又分别基于 形状 和 材料的组合）。

注意： 与 Ip2 函子的情况一样，若某个 *IGeom* 和 *IPhys* 的组合未被任何 Law2 函子处理，则会报错。

警告： Yade 中有许多 Law2，而且随时可能出现新的。在某些情况下，不同的函子只是同一接触定律的不同实现（例如 *Law2_ScGeom_FrictPhys_-CundallStrack* 和 *Law2_L3Geom_FrictPhys_ElPerfPl*）。此外，有时某个函子的独特功能也可以作为更通用函子的特殊情况来实现。因此，对于给定的本构行为，用户可以在不同函子之间进行选择。遇到这种情况时，强烈建议优先选择使用最广泛、经过验证最多的实现。可用函子列表按成熟度从高到低排列，并会持续更新，见 此处，以供选择参考。

示例

让我们给出几个已创建并接受的类 型链的示例。 型。

基本 DEM 模型

假 我们想要使用定律2_ScGeom_FrictPhys_CundallStrack 本构定律。我们看到 at

1. Ig2 函子必须创建*ScGeom*。例如，如果模拟中有球体和盒体，我们就需要能接受以下组合的函子：球体 + 球体和盒体 + 球体。我们不希望盒体之间发生相互作用（事实上，也没有这样的函子）。因此我们需要*Ig2_球体_球体_ScGeom* 和*Ig2_盒体_球体_ScGeom*。

2. Ip2 函子应创建*FrictPhys*。查看*InteractionPhysicsFunctors*，这里只有*Ip2_FrictMat_FrictMat_FrictPhys*。这就要求我们对颗粒使用 *FrictMat*。

因此，结果为：

```
InteractionLoop(
    [Ig2_Sphere_Sphere_ScGeom(),Ig2_Box_Sphere_ScGeom()],
    [Ip2_FrictMat_FrictMat_FrictPhys()],
    [Law2_ScGeom_FrictPhys_CundallStrack()]
)
```

混凝土模型

在这个算例中，我们的目标是使用 *定律2_ScGeom_CpmPhys_Cpm* 本构定律。

- 我们在模拟中使用 球体 和 面片，因此会选择 `Ig2` 接受这些类型并生成 *ScGeom*的函子： *Ig2_Sphere_Sphere_ScGeom* 和*Ig2_Facet_Sphere_ScGeom*。
- 我们必须使用 材料 来创建 *CpmPhys*。我们发现 *CpmPhys* 仅由 *Ip2_CpmMat_CpmMat_CpmPhys*创建，因此所有颗粒都应选择 *CpmMat* 作为材料。

因此，我们将使用：

```
InteractionLoop(
    [Ig2_Sphere_Sphere_ScGeom(),Ig2_Facet_Sphere_ScGeom()],
    [Ip2_CpmMat_CpmMat_CpmPhys()],
    [Law2_ScGeom_CpmPhys_Cpm()]
)
```

施加条件

在大多数模拟中，我们并不希望所有颗粒都在空间中自由漂浮。施加边界条件有多种方式，可以阻止所有或部分颗粒相对于全局空间运动。

运动约束

- *Body.dynamic* 决定刚体是否会被 *NewtonIntegrator* 加速；对于零质量的刚体，必须将其设为 false，因为施加非零力会导致无限位移。

  面片 是此处的算例： 面片 默认情况下会被设为非动态，因为它们的体积和质量均为零（可以通过将 `dynamic=True` 传递给 面片，或设置 *Body.dynamic* 来更改；还必须将 *State.mass* 设为非零值）。墙也是如此。

  只需以下操作即可将球体设为非动态：

  ```
  b = sphere([x,y,z],radius,dynamic=False)
  b.dynamic=True #revert the previous
  ```

- *State.blockedDOFs* 允许有选择地阻塞全局空间中的 6 个自由度中的任意一个。例如，将球体限制为仅在 xy平面内运动，可以这样写：

  ```
  Yade [31]: O.bodies.append(sphere((0,0,0),1))
  Out[31]: 0

  Yade [32]: O.bodies[0].state.blockedDOFs='zXY'
  ```

  与 *Body.dynamic* 相比，受阻自由度 只会阻止选定方向上的力（和加速度）。实际上，`b.dynamic=False` 几乎只是 `b.state.blockedDOFs=='xyzXYZ'` 的简写形式。一个细微的差异是，前者会自动重置速度分量，而后者不会。如果指定了线速度或角速度，它们将不受 受阻自由度 的影响。这也意味着，如果在自由度

通过 blockedDOFs 赋值来阻止时，刚体仍会以阻止时的速度继续运动。差异如下所示：

```
Yade [33]: b1 = sphere([0,0,0],1,dynamic=True)Yade [34]: b1.state.blockedDOFs
Out[34]: ''Yade [35]: b1.state.vel = Vector3(1,0,0) #we want it to move...
Yade [36]: b1.dynamic = False #... at a constant velocity
Yade [37]: print(b1.state.blockedDOFs, b1.state.vel)xyzXYZ Vector3(0,0,0)
Yade [38]: # oops, velocity has been reset when setting dynamic=False
Yade [39]: b1.state.vel = (1,0,0) # we can still assign it now
Yade [40]: print(b1.state.blockedDOFs, b1.state.vel)xyzXYZ Vector3(1,0,0)
Yade [41]: b2 = sphere([0,0,0],1,dynamic=True) #another try
Yade [42]: b2.state.vel = (1,0,0)
Yade [43]: b2.state.blockedDOFs = "xyzXYZ" #this time we assign blockedDOFs directly,␣
→velocity is unchangedYade [44]: print(b2.state.blockedDOFs, b2.state.vel)
xyzXYZ Vector3(1,0,0)
```

有时可能需要根据某些条件约束由生成的球体堆积构成的部分颗粒的运动，例如位于试样底部的颗粒；可以通过带条件地循环遍历全部刚体来实现：

```
for b in O.bodies:
   # block all particles with z coord below .5:
   if b.state.pos[2]<.5: b.dynamic=False
```

上文介绍的任意空间谓词也可以在这里使用：

```
from yade import pack
pred=pack.inAlignedBox(lowerCorner,upperCorner)
for b in O.bodies:
   if not isinstance(b.shape,Sphere): continue # skip non-spheres
   # ask the predicate if we are inside
   if pred(b.state.pos,b.shape.radius): b.dynamic=False
```

施加运动和力

施加速度

如果某个自由度被约束，并沿该方向赋予速度（平移速度或转动速度），那么刚体将以恒定速度运动。这是施加刚体运动更简单且推荐的方法。例如，这将使刚体沿 x 以恒定速度运动（它仍可沿 y和 z自由加速）：

```
O.bodies.append(sphere((0,0,0),1))
O.bodies[0].state.blockedDOFs='x'
O.bodies[0].state.vel=(10,0,0)
```

相反，直接修改位置可能会破坏 Yade 的算法，尤其是与碰撞检测和接触定律相关的算法，因为这些算法都基于刚体的速度。因此，除非你确实清楚自己在做什么，否则不要通过这种方式施加运动：

```
O.bodies.append(sphere((0,0,0),1))
O.bodies[0].state.blockedDOFs='x'
O.bodies[0].state.pos=10*O.dt #REALLY BAD! Don't assign position
```

施加力

对刚体施加力或力矩，需要调用 *ForceContainer* 的函数。操作很简单：

```
O.forces.addF(0,(1,0,0)) #applies for one step
```

这样，力只会作用一个时间步，并会在每一步开始时重置。因此，在某一步开始时施加力不会产生任何效果，因为它会立即被重置。唯一的办法是在模拟循环中放置一个 *PyRunner* 。

也可以通过另一个函数永久施加力（在这种情况下，command 是否在时间步开始时执行并不重要）：

```
O.forces.setPermF(0,(1,0,0)) #applies permanently
```

该力会在多次迭代中持续生效，直到被对 `O.forces.setPermF(id,f)` 的另一次调用覆盖，或被 `O.forces.reset(resetAll=True)`清除。可以通过`O.forces.permF(id)`检查作用于刚体的永久力。

边界控制器

由 边界控制器 派生的引擎会在模拟过程中施加边界条件，可以直接施加，也可以通过影响多个刚体来实现。详情请参阅各自的文档；特别值得关注的是

- *UniaxialStrainer*，用于以恒定速率沿一个轴施加应变；适用于绘制单轴加载算例的应力-应变图。示例见 examples/concrete/uniax.py 。
- 三轴应力控制器 使用 6 个壁面（*Box* 对象），沿 3 个相互垂直的轴对长方体形堆积体施加指定的应力/应变。
- *PeriTriaxController* 用于在采用周期性边界条件的模拟中，沿 3 个轴分别施加应力/应变（单元）

场施加器

派生自 场施加器 的引擎会作用于所有颗粒。最常用的是 *GravityEngine*施加均匀加速度场（*GravityEngine* 已弃用，请改用 *NewtonIntegrator.gravity* ）。

部分引擎

派生自 部分引擎 的引擎定义了用于指定受影响刚体的 标识符 属性。其中有几个值得特别提及：

- 平移引擎 和 旋转引擎 用于对订阅者施加恒定速度的线性运动和转动。
- 力引擎和力矩引擎每个时间步都将给定的力/力矩施加到已订阅刚体上。
- 步进位移器用于在每个时间步施加广义位移增量；专为在两个颗粒上测试本构定律时精确控制运动而设计。

只有在需要指定复杂类型的力场或位移场时，部分引擎才真正发挥作用。若要让刚体以恒定速度运动，或施加单个力，施加运动和力中介绍的方法要简单得多。有几种插值引擎（*InterpolatingDi-rectedForceEngine* 用于施加模不断变化的力，插值螺旋引擎用于施加角速度不断变化的螺旋位移；参见examples/test/helix.py，此外可能还有其他引擎）；参考现有示例，编写新的插值引擎相当简单。

便捷功能

标记事物

引擎和函子可以定义一个 `label` 属性。每当 `O.engines` 序列被修改时，都会创建或更新具有相应名称的 Python 变量；由于这一操作发生在 `__builtins__` 命名空间中，因此可以立即从任何位置访问这些名称。这一功能曾用于创建相互作用，以便同时更改多个函子中的相互作用半径。

警告：请确保不要使用会覆盖（或遮蔽）你已经通过该变量名使用的对象的标签。注意不要使用语法错误的名称，例如“er*452”或“my engine”；只能使用 Python 中允许的变量名。

模拟标签

Omega.tags 是一个字典（它的行为类似字典，但其 C 语言实现++ 有所不同），用于将键映射到标签。与可以创建的常规 Python 字典不同，

- `O.tags`会随模拟一起保存和加载；
- `O.tags` 有一些预先初始化的值。

Yade 启动后， `O.tags` 包含以下内容：

```
Yade [45]: dict(O.tags) # convert to real dictionary
Out[45]:{'author': 'bchareyre~(bchareyre@HP-ZBook-15-G3)',
'isoTime': '20220726T141512','id': '20220726T141512p61547',
'd.id': '20220726T141512p61547',
'id.d': '20220726T141512p61547'}
```

作者 创建模拟时从系统中获取的真实姓名、用户名和计算机名称

标识符 此 Yade 实例（或创建已加载模拟的实例）的唯一标识符。由日期、时间和进程编号组成。如果并行运行模拟，并希望避免相互覆盖输出，则此标识符很有用；将 `O.tags['id']` 嵌入输出文件名中（作为目录名或文件名的一部分）即可避免这种情况。详细说明请参见 分离作业输出文件 。

ISO时间 模拟创建的时间（精确到秒）。

d.id, id.d 仿真描述和标识符以句点连接（反之亦然）。描述用于批处理作业；在非批处理作业中，这些标签与标识符相同。

只需赋值即可添加自定义标签，但左侧对象必须是字符串，且不得包含 =。

```
Yade [46]: O.tags['anythingThat I lik3']='whatever'

Yade [47]: O.tags['anythingThat I lik3']
Out[47]: 'whatever'
```

保存 Python 变量

Python 变量的生命周期有限；特别是，如果保存模拟，重新加载后变量就会丢失。因此，Yade 对数据持久化提供了有限的支持（内部使用 O.tags 的特殊值）。相关函数是 *saveVars* 和*loadVars*。

saveVars 接受一个字典（变量名及其对应的值）和一个 标记 （用于标识变量集的字符串）；它会将该字典保存在模拟中。加载模拟（例如，从 XML 文件加载）后，可以通过调用 yade.params.标记，在命名空间中重新创建这些变量：*loadVars* 并传入相同的标识标记：

```
Yade [48]: a=45; b=pi/3
Yade [49]: saveVars('ab',a=a,b=b)
# save simulation (we could save to disk just as well)
Yade [49]: O.saveTmp()Yade [51]: O.loadTmp()
Yade [52]: loadVars('ab')Yade [53]: yade.params.ab.a
Out[53]: 45# import like this
Yade [54]: from yade.params import abYade [55]: ab.a,
 ab.bOut[55]: (45, 1.0471975511965976)# also possible
Yade [56]: from yade.params import *Yade [57]: ab.a,
 ab.bOut[57]: (45, 1.0471975511965976)
```

如果变量很多，逐一枚举可能会很繁琐；创建局部作用域（例如在 Python 中定义一个函数）会有所帮助：

```
def setGeomVars():
        radius=4
        thickness=22
        p_t=4/3*pi
        dim=Vector3(1.23,2.2,3)
        #
        # define as much as you want here
        # it all appears in locals() (and nothing else does)
        #
        saveVars('geom',loadNow=True,**locals())

setGeomVars()
```

（下页继续）

（续前页）

```
from yade.params.geom import *
# use the variables now
```

注意：只有可以进行 pickle 序列化的类型才能传递给 *saveVars*。

2.2.2 控制模拟

跟踪变量

运行 Python 代码

特殊引擎 *PyRunner* 可用于定期调用 Python 代码，调用内容通过 `command`参数指定。可通过指定计算时间（`realPeriod`）、虚拟时间（`virtPeriod`）或迭代次数（`iterPeriod`）来控制周期。

例如，若要每 5 秒打印一次动能（使用 动能），可将以下引擎添加到 `O.engines`中：

```
PyRunner(command="print('kinetic energy',kineticEnergy())",realPeriod=5)
```

运行更复杂的命令时，定义一个外部函数并仅在引擎中调用它会更方便。由于 `command` 在脚本命名空间中运行，因此可以调用脚本中定义的函数。下面定期打印刚体 0 和 1 之间相互作用的信息：

```
def intrInfo(id1,id2):
        try:
                i=O.interactions[id1,id2]
                # assuming it is a CpmPhys instance
                print (d1,id2,i.phys.sigmaN)
        except:
                # in case the interaction doesn't exist (yet?)
                print("No interaction between",id1,id2)
O.engines=[...,
        PyRunner(command="intrInfo(0,1)",realPeriod=5)
]
```

警告： 如果函数是在 实时 yadesession（IPython）中声明的，那么除非使用 command 更新 Python 全局变量，否则 `NameError:name 'intrInfo' is not defined` 将会出错

```
globals().update(locals())
```

下面将给出更多有用的示例。

绘图 模块提供了用于跟踪各种数据的简单接口和存储功能。尽管该模块最初仅用于绘图，但现在也广泛用于跟踪一般变量。

数据位于*plot.data* 字典中，该字典将变量名映射到其值的列表；使用*plot.addData*函数可添加这些数据。

```
Yade [58]: from yade import plot

Yade [59]: plot.data
Out[59]: {}

Yade [60]: plot.addData(sigma=12,eps=1e-4)
```

（续见下页）

（续前页）

```
# not adding sigma will add a NaN automatically
# this assures all variables have the same number of records
Yade [61]: plot.addData(eps=1e-3)

# adds NaNs to already existing sigma and eps columns
Yade [62]: plot.addData(force=1e3)

Yade [63]: plot.data
Out[63]:
{'sigma': [12, nan, nan],
 'eps': [0.0001, 0.001,  nan],
 'force': [nan, nan, 1000.0]}

# retrieve only one column
Yade [64]: plot.data['eps']
Out[64]: [0.0001, 0.001, nan]

# get maximum eps
Yade [65]: max(plot.data['eps'])
Out[65]: 0.001
```

每次调用 *plot.addData* 时，都会向全部列添加一条新记录；这确保不同列中的线始终匹配。特殊值 `nan` 或 `NaN` (非数值) 会被插入以标记该记录无效。

注意：它 不能有两个同名的列，因为数据是以字典形式存储的 y.

若要定期记录数据，请使用 *PyRunner*。它将记录 z 坐标、刚体 #1 的速度、迭代次数和模拟时间（每 20 次迭代记录一次）：

```
O.engines=O.engines+[PyRunner(command='myAddData()', iterPeriod=20)]

from yade import plot
def myAddData():
        b=O.bodies[1]
        plot.addData(z1=b.state.pos[2], v1=b.state.vel.norm(), i=O.iter, t=O.time)
```

注意：对于 plot.data，可以使用任意字符串作为列标签。不过，如果名称中包含空格（例如 `my funny column`），或是 Python 保留关键字（例如 `for`），则传递给*plot.addData* 的唯一方法是使用字典：

```
plot.addData(**{'my funny column':1e3, 'for':0.3})
```

前导或尾随空白字符的列是个例外。它们在 *plot.plots*中会被特殊处理，不应使用（见下文）。

在 `myAddData` 函数中，可以方便地通过标签访问引擎：

```
O.engines=[...,
        UniaxialStrainer(...,label='strainer')
]
def myAddData():
        plot.addData(sigma=strainer.avgStress,eps=strainer.strain)
```

在这种情况下，显然，带标签的对象必须定义所用的属性（*UniaxialStrainer.strain*和 *UniaxialStrainer.avgStress* 在本例中）。

变量绘图

上文介绍了如何通过使用 *plot.addData*存储变量来跟踪它们。这些数据可以直接用于绘图。Yade 在 *plot* 模块中提供了简单快捷的绘图功能。当然，由于可以通过 *plot.data*直接访问底层数据，因此也可以用其他方式处理这些数据。

plot.plots 字典是对图形的简单定义。键为横轴变量，值为纵轴变量元组，以用于 *plot.addData* 的字符串形式给出；字典中的每个条目分别代表一个图形：

```
plot.plots={
        'i':('t' ),     # plot t(i)
        't':('z1','v1') # z1(t) and v1(t)
}
```

使用 *plot.data* 中的数据以及 *plot.plots* 中的绘图定义进行实际绘图，可以通过调用*plot.plot* 函数来触发。

绘图实时更新

Yade 支持在计算过程中实时更新图形。其由以下设置控制：

- *plot.live* - 设置 `yade.plot.live=True` 后，可以在计算运行时观察绘图的更新。否则，设置为 `False` 。
- *plot.liveInterval* - 这是绘图更新之间的时间间隔，单位为秒。
- *plot.autozoom* - 设置为 `True` 后，绘图将自动重新缩放。

控制线条属性

在本小节中，我们使用一个基本的完整脚本，例如examples/simple-scene/simple-scene-plot.py，稍后我们将修改它，让绘图更加美观。该文件中感兴趣的代码行如下，并会生成下方所示的图像：

```
plot.plots={'i':('t'),'t':('z_sph',None,('v_sph','go-'),'z_sph_half')}
```

折线图可以带有一个可选的第二个字符串参数，由线条颜色（例如`'r'`、 `'g'` 或`'b'`）、线型（例如`'-'`、 `'--'` 或 `':'`）以及线条标记（`'o'`、 `'s'` 或 `'d'`）组成。使用‘ro:’参数可以创建带圆形标记的红色点线。有关全部选项的列表，请参阅http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot

例如，使用以下 plot.plots() command 将生成如下图形：

```
plot.plots={'i':(('t','xr:'),),'t':(('z_sph','r:'),None,('v_sph','g--'),('z_sph_half','b-.'))}
```

而以下内容将生成如下图形：

```
plot.plots={'i':(('t','xr:'),),'t':(('z_sph','Hr:'),None,('v_sph','+g--'),('z_sph_half','*b-.
→'))}
```

注意： 你可以在 matplotlib 教程中了解更多内容 http://matplotlib.sourceforge.net/users/pyplot_tutorial.html 和文档http://matplotlib.sourceforge.net/users/pyplot_tutorial.html#controlling-line-properties

---

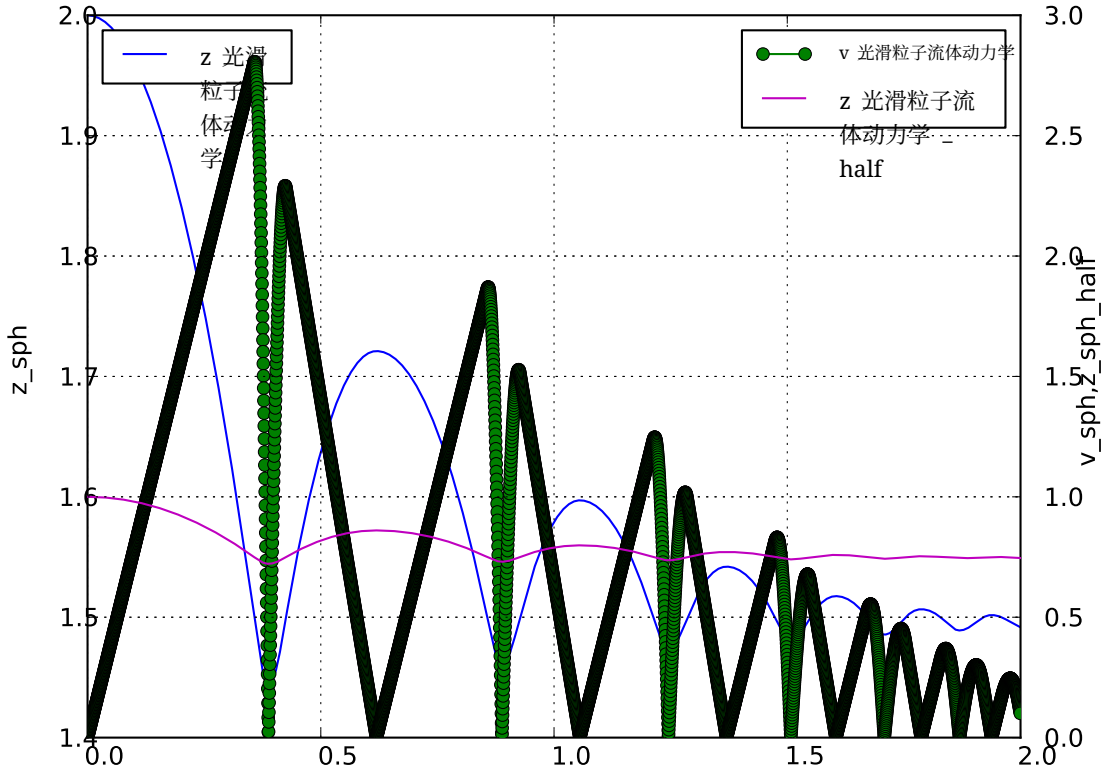


图12：由以下脚本生成的图形：examples/simple-scene/simple-scene-plot.py。

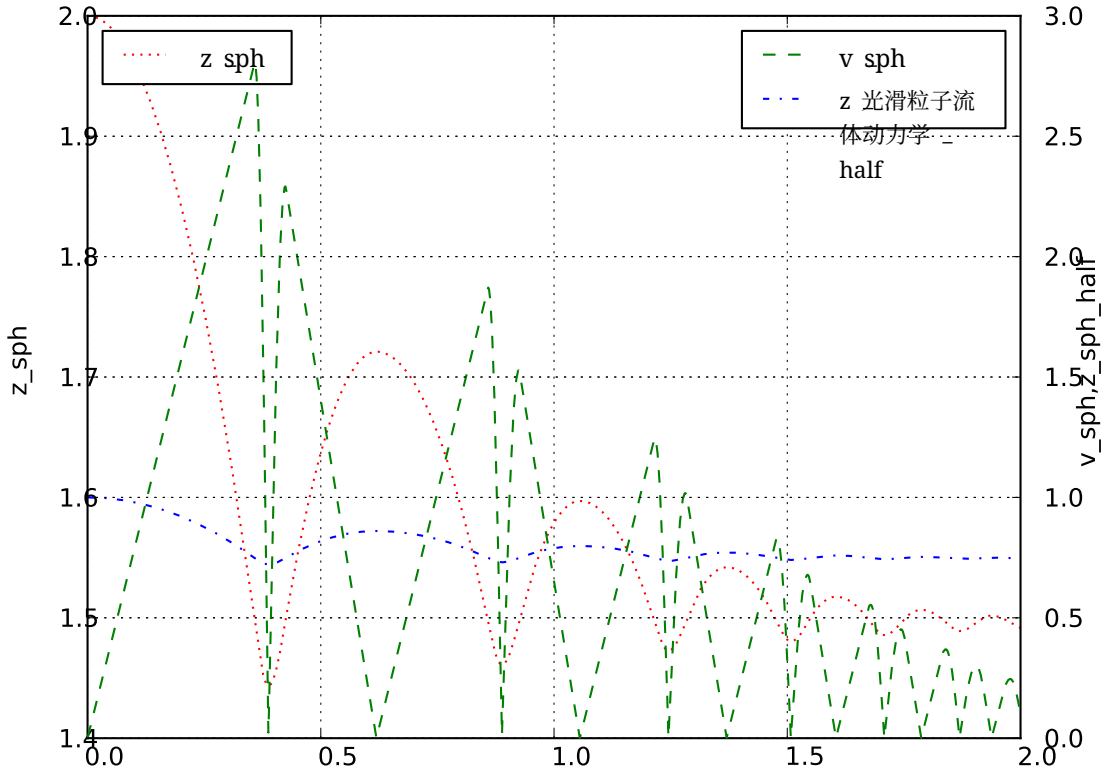


图 13：通过按上述方式更改参数以调用 plot.plots 生成的图形。

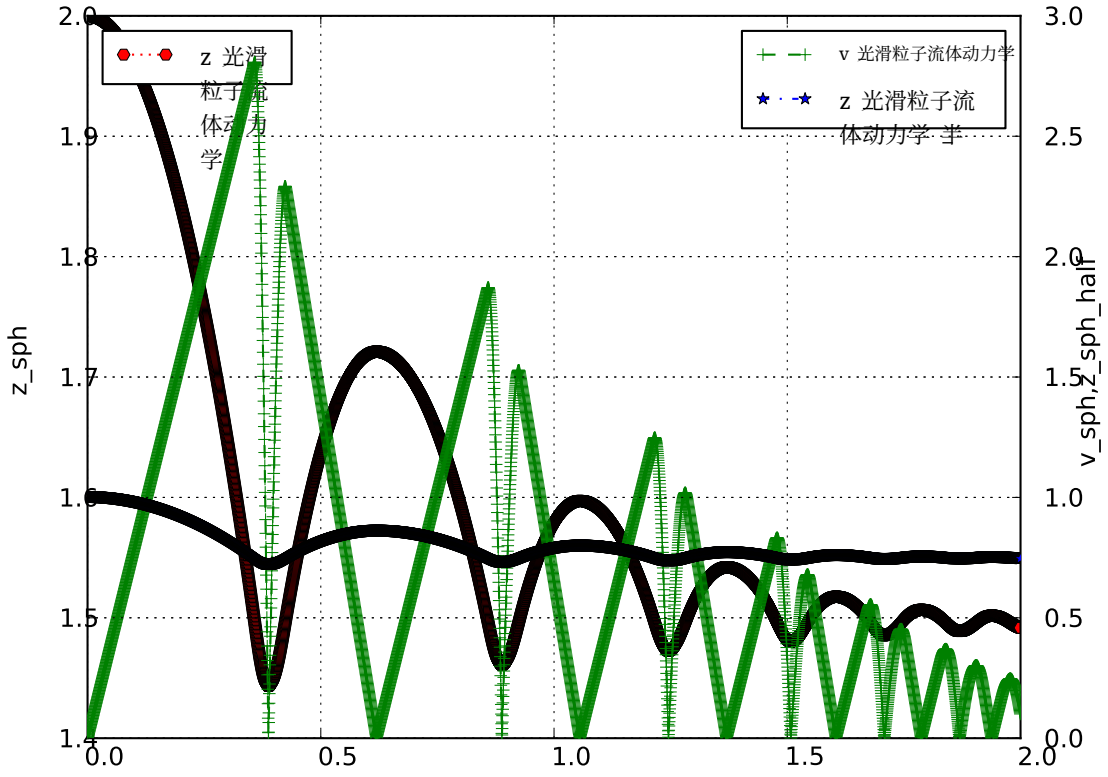


图 14：通过更改上述 plot.plots 的参数生成的图形。

注意： 请注意， `'i':(('t','xr:'),)`中多了一个逗号，否则 `'xr:'` 不会被识别为线型参数，而会被视为要绘制的额外数据。

控制文本标签

可以在绘图标签中使用 TeX 语法。例如，在 examples/simple-scene/simple-scene-plot.py 中使用以下两行代码，将生成如下图形：

```
plot.plots={'i':(('t','xr:'),),'t':(('z_sph','r:'),None,('v_sph','g--'),('z_sph_half','b-.'))}
plot.labels={'z_sph':'$z_{sph}$' , 'v_sph':'$v_{sph}$' , 'z_sph_half':'$z_{sph}/2$'}
```

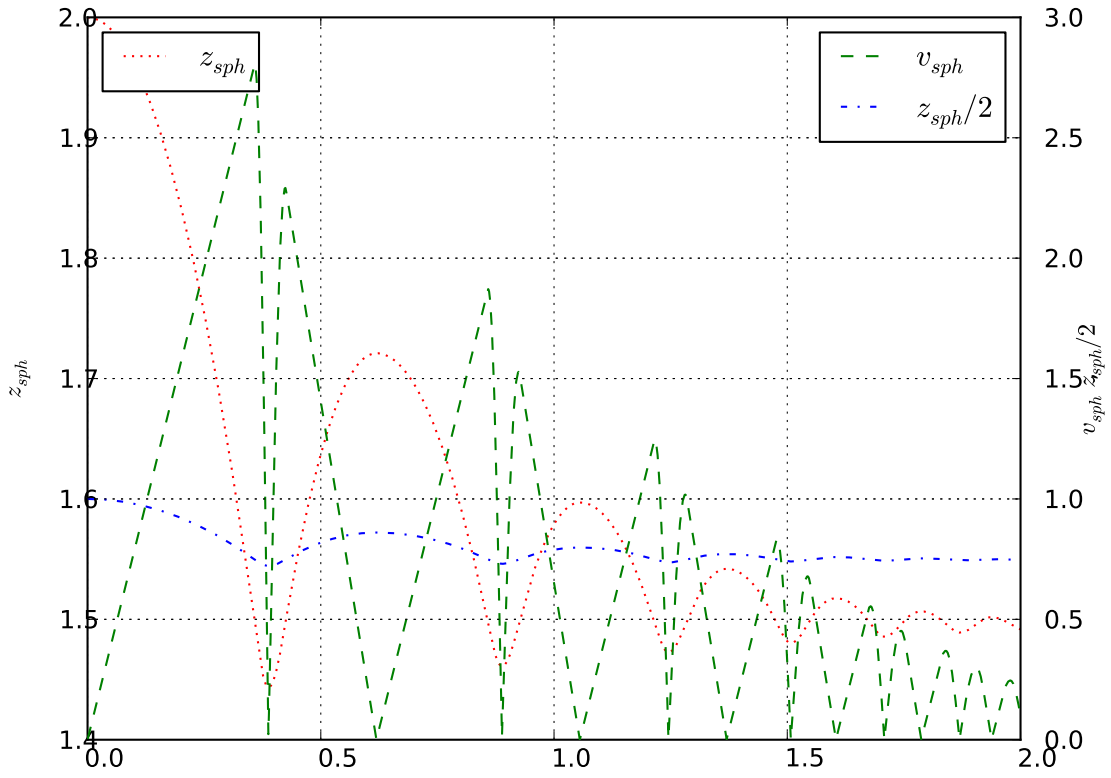


图 15：由 examples/simple-scene/simple-scene-plot.py 生成的带有 TeX 标签的图形。

希腊字母只需在这些标签中使用 `'$\alpha$'`、 `'$\beta$'` 等即可。要更改字体样式，可以使用以下 command：

```
yade.plot.matplotlib.rc('mathtext', fontset='stixsans')
```

不过，这并不是 Yade 的一部分，而是 matplotlib 的一部分。如果你想要更复杂的功能，确实应该查阅 matplotlib 用户手册http://matplotlib.sourceforge.net/users/index.html

多幅图形

由于 *plot.plots* 是一个字典，因此不可能有多个条目使用同一个键（横轴变量），因为它们会相互覆盖：

```
Yade [66]: plot.plots={
   ....:     'i':('t',),
   ....:     'i':('z1','v1')
   ....: }....:


Yade [67]: plot.plots
Out[67]: {'i': ('z1', 'v1')}
```

不过，您可以在横轴变量前后添加空格来区分它们；查找 *plot.data* 中的变量时，空格会被自动移除——两个 x轴都会使用 `i` 列：

```
Yade [68]: plot.plots={
   ....:     'i':('t',),
   ....:     'i ':('z1','v1') # note the space in 'i '
   ....: }....:


Yade [69]: plot.plots
Out[69]: {'i': ('t',), 'i ': ('z1', 'v1')}
```

拆分 y1、y2 轴

为了避免 `y` 轴上的数值范围差异过大，可以将左侧和右侧的 `y` 轴分开（如`axes x1y2` 在 gnuplot 中）。只需在绘图说明符中插入 `None`；此前的变量将按常规方式绘制（在左侧 *y* 轴上），此后的变量则显示在右侧：

```
plot.plots={'i':('z1',None,'v1')}
```

导出

图形和数据可以导出到外部文件，以便稍后使用 *Gnuplot<http://www.gnuplot.info/>* 进行后处理。方法是使用 *plot.saveGnuplot* 函数。请注意，使用 plot.addData 添加的全部数据都会被保存——包括模拟期间未实时绘制的数据。之后，编辑生成的 .gnuplot 文件，即可绘制已添加的任意数据。

- 数据文件会与 gnuplot 文件分开保存（使用 bzip2 压缩），因此可以使用其他程序处理这些数据。特别是，`numpy.genfromtxt` (此处有文档说明) 对于将这些数据重新导入 Python 很有用；解压缩会自动进行。

- 可以通过 gnuplot 运行 gnuplot 文件来生成图形；详情请参阅 *plot.saveGnuplot* 文档。

若要使用 gnuplot 以外的工具进行后处理，也可以使用 *plot.saveDataTxt* 将数据导出为另一种文本文件。

停止条件

对于预先确定步数的模拟，可以进行如下设置：

```
# absolute iteration number
O.stopAtIter=35466
O.run()
O.wait()
```

or

```
# number of iterations to run from now
O.run(35466,True) # wait=True
```

使模拟运行 35466 次迭代后停止。

通常，需要根据模拟本身的演变情况做出决策，而这种情况事先尚不可知。在这种情况下，会定期调用一个检查特定条件的函数；如果满足该条件，`O.pause` 或者可以调用其他函数来停止模拟。详情请参阅文档中的 *Omega.run*、*Omega.pause*、*Omega.step*、*Omega.stopAtIter* 。

对于旨在达到静态平衡的模拟，*unbalancedForce* 可以提供有用的指标（详情请参阅其文档）；例如，若目标值为 `1e-2` 或更低，我们可以这样做：

```
def checkUnbalanced():
        if unbalancedForce<1e-2: O.pause()

O.engines=O.engines+[PyRunner(command="checkUnbalanced()",iterPeriod=100)]

# this would work as well, without the function defined apart:
#   PyRunner(command="if unablancedForce<1e-2: O.pause()",iterPeriod=100)

O.run(); O.wait()
# will continue after O.pause() will have been called
```

可以定期检查任意函数，这些函数也可以使用通过*plot.addData*跟踪的变量历史记录。例如，以下是examples/concrete/uniax.py中损伤控制的简化版本；当当前应力低于峰值应力的一半时停止：

```
O.engines=[...,
        UniaxialStrainer=(...,label='strainer'),
        PyRunner(command='myAddData()',iterPeriod=100),
        PyRunner(command='stopIfDamaged()',iterPeriod=100)
]

def myAddData():
        plot.addData(t=O.time,eps=strainer.strain,sigma=strainer.stress)

def stopIfDamaged():
        currSig=plot.data['sigma'][-1] # last sigma value
        maxSig=max(plot.data['sigma']) # maximum sigma value
        # print something in any case, so that we know what is happening
        print(plot.data['eps'][-1],currSig)
        if currSig<.5*maxSig:
                print("Damaged, stopping")
                print('gnuplot',plot.saveGnuplot(O.tags['id']))
                import sys
                sys.exit(0)

O.run(); O.wait()
# this place is never reached, since we call sys.exit(0) directly
```

检查点

有时，回退到过去某个时间点的模拟状态，并从那里使用不同参数继续模拟，会很有用。例如，拉伸/压缩试验会使用相同的初始状态，但沿两个不同方向加载。为此提供了两个函数，*Omega.saveTmp* 和 *Omega.loadTmp* ；内存用作存储介质，这意味着保存速度更快，而且 Yade 结束后模拟数据也会消失。

```
O.saveTmp()
# do something
O.saveTmp('foo')
O.loadTmp()      # loads the first state
O.loadTmp('foo') # loads the second state
```

警告： `O.loadTmp` 不能从引擎内部调用，因为在加载 模拟之前，旧模拟必须完成当前迭代；否则会导致死锁，因为 `O.loadTmp` 会等待当前迭代完成，而当前迭代会阻塞于 `O.loadTmp`。

必须采用一种特殊技巧：定义一个在当前迭代之后运行的独立函数，并从一个仅为此目的启动的独立线程中调用该函数：

```
O.engines=[...,PyRunner('myFunc()',iterPeriod=345)]

def myFunc():
        if someCondition:
                import thread
                # the () are arguments passed to the function
                thread.start_new_thread(afterIterFunc,())
def afterIterFunc():
        O.pause(); O.wait() # wait till the iteration really finishes
        O.loadTmp()

O.saveTmp()
O.run()
```

远程控制

Yade 可以通过网络进行远程控制。在 Yade 启动时，除其他消息外，还会显示以下线：

```
TCP python prompt on localhost:9000,  auth cookie `dcekyu'
TCP info provider on localhost:21000
```

它们说明了两个端口，分别接受两种不同类型的连接。

Python 提示符

`TCP python prompt` 这是一个经过身份验证的 Telnet 服务器，提供完整的 Python 命令行。它监听端口 9000；如果该端口已被占用（例如被另一个 Yade 实例占用），则使用更高的端口。
使用身份验证 cookie，可以通过 telnet 建立连接：

```
$ telnet localhost 9000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Enter auth cookie: dcekyu
__   __    ____                  __  _____ ____ ____
\ \ / /_ _|  _ \  ___    ___   / / |_   _/ ___|  _ \
 \ V / _` | | | |/ _ \  / _ \ / /    | || |   | |_) |
  | | (_| | |_| |  __/ | (_) / /     | || |___|  __/
  |_|\__,_|____/ \___|  \___/_/      |_| \____|_|

(connected from 127.0.0.1:40372)
>>>
```

Python 伪提示符 `>>>` 让您能够像往常一样，以各种方式编写命令来操纵模拟。需要注意两点：

1. 新的 Python 解释器（>>>）位于与 `Yade [1]:` 命令行分离的命名空间中。为方便起见，`from yade import *` 会先在新的 Python 实例中运行，但无法访问局部变量和全局变量（只有内置变量可用）。
2. （伪） `>>>` 解释器不具备 IPython 丰富的交互功能，而 IPython 可以处理通常的 `Yade [1]:` 命令行；因此，你将无法使用命令历史、 `?` 帮助等功能。

注意： 允许访问 Python 解释器，就可能完全控制系统（包括读取用户文件）。因此，**连接仅允许来自本地主机**，不允许通过网络远程连接。当然，你可以通过网络使用 SSH 登录系统，以获得远程访问权限。

警告： 身份验证 cookie 很容易遭到暴力破解攻击。尽管监听器在每次登录失败后会暂停 5 秒（并断开连接），但在共享计算机上进行时间很长的模拟时，仍可能通过反复试探猜出 cookie。

信息提供程序

`TCP Info provider` 监听端口 21000（或更高端口），并在连接时返回有关当前仿真的一些基本信息；随后立即终止连接。这些信息是用标准 pickle模块序列化的 Python 字典，以字符串形式表示。

批处理系统（如下所述）使用此功能来获取各个模拟的进度和预计时间信息。如果你想自行访问这些信息，可以查看core/main/yade-batch.in 了解详情。

批处理排队与执行（yade-batch）

Yade 提供了一个轻量级系统，可使用不同参数运行同一模拟；它负责将参数值赋给模拟脚本中的 Python 变量、根据可用核心数和所需核心数安排作业等。整个批处理由 2 个文件组成：

模拟脚本 常规 Yade 脚本，调用 *readParamsFromTable* 从参数表获取参数。为了让脚本也能在批处理之外运行，*readParamsFromTable*会为参数提供默认值，这些值可能会被参数表中的值覆盖。

> *readParamsFromTable* 通过检查由批处理系统设置的 `PARAM_-TABLE` 环境变量，获知要读取哪个参数文件以及其中的哪一行。

参数表 简单的文本文件，每一行代表一个参数集。该文件由*readParamsFromTable* （使用 *TableParamReader* 类）读取，并由模拟脚本调用，如上所述。为了便于阅读文本文件，可以使用制表符；*readParamsFromTable*会忽略这些制表符。参数不局限于数值，也可以通过 `"quoting"` 将其括起来（`' '` 也可以代替 `" "`）。这对名义参数很有用。

批处理可以按如下方式运行

```
yade-batch parameters.table simulation.py
```

它会智能地针对参数表中的每一行运行一次模拟。一个最小示例见examples/test/batch/params.table和 examples/test/batch/sim.py，下面是另一个示例。

示例

假设我们想研究参数密度 和初始速度 对球体落在固定盒子上的位置的影响。我们创建如下参数表：

```
description density initialVelocity # first non-empty line are column headings
reference   2400    10
hi_v           =    20              # = to use value from previous line
lo_v           =     5
# comments are allowed
hi_rho      5000    10
# blank lines as well:

hi_rho_v       =    20
hi_rh0_lo_v    =     5
```

每行给出这 2 个参数的一种组合，并为此模拟指定一个（可选的）描述。

在模拟文件中，我们在脚本开头从表格读取参数；每个参数都有默认值，如果参数文件中未指定，则使用该默认值：

```
readParamsFromTable(gravity=-9.81,density=2400,initialVelocity=20,
noTableOk=True      # use default values if not run in batch)
from yade.params.table import *
print(gravity, density, initialVelocity)
```

调用 *readParamsFromTable*后，会在 yade.params.table模块中创建相应的 Python 变量，可直接在脚本中使用，例如：

```
GravityEngine(gravity=(0,0,gravity))
```

让我们看看运行批处理时会发生什么：

```
$ yade-batch batch.table batch.py
Will run `/usr/local/bin/yade-trunk' on `batch.py' with nice value 10,  output redirected to␣
→`batch.@.log',  4 jobs at a time.Will use table `batch.table',  with available lines 2, 3、
4、5、6、7。Will use lines  2 (reference),  3 (hi_v),  4 (lo_v),  5 (hi_rho),  6 (hi_rho_v),
 7 (hi_rh0_lo_v).Master process pid 7030
```

这些线提供了批处理的一般信息：较好的级别、日志文件名、将使用多少个核心（4）；表格名称，以及包含参数的行号；最后，哪些线将被使用；主进程进程 ID可用于通过以下方式终止（停止）整个批处理： kill command。

```
Job summary:
   #0 (reference/4): PARAM_TABLE=batch.table:2 DISPLAY=  /usr/local/bin/yade-trunk --threads=4␣
→--nice=10 -x batch.py > batch.reference.log 2>&1
   #1 (hi_v/4): PARAM_TABLE=batch.table:3 DISPLAY=  /usr/local/bin/yade-trunk --threads=4 --
→nice=10 -x batch.py > batch.hi_v.log 2>&1
   #2 (lo_v/4): PARAM_TABLE=batch.table:4 DISPLAY=  /usr/local/bin/yade-trunk --threads=4 --
→nice=10 -x batch.py > batch.lo_v.log 2>&1
   #3 (hi_rho/4): PARAM_TABLE=batch.table:5 DISPLAY=  /usr/local/bin/yade-trunk --threads=4 --
→nice=10 -x batch.py > batch.hi_rho.log 2>&1
   #4 (hi_rho_v/4): PARAM_TABLE=batch.table:6 DISPLAY=  /usr/local/bin/yade-trunk --threads=4 -
→-nice=10 -x batch.py > batch.hi_rho_v.log 2>&1
   #5 (hi_rh0_lo_v/4): PARAM_TABLE=batch.table:7 DISPLAY=  /usr/local/bin/yade-trunk --
→threads=4 --nice=10 -x batch.py > batch.hi_rh0_lo_v.log 2>&1
```

显示所有作业及其各自将运行的命令行。此时，批处理开始运行。

```
#0 (reference/4) started on Tue Apr 13 13:59:32 2010#0 (reference/4) done    (exit status 0),
 duration 00:00:01,  log batch.reference.log#1 (hi_v/4) started on Tue Apr 13 13:59:34 2010
#1 (hi_v/4) done    (exit status 0),  duration 00:00:01,  log batch.hi_v.log
#2 (lo_v/4) started on Tue Apr 13 13:59:35 2010#2 (lo_v/4) done    (exit status 0),
 duration 00:00:01,  log batch.lo_v.log#3 (hi_rho/4) started on Tue Apr 13 13:59:37 2010
#3 (hi_rho/4) done    (exit status 0),  duration 00:00:01,  log batch.hi_rho.log
#4 (hi_rho_v/4) started on Tue Apr 13 13:59:38 2010#4 (hi_rho_v/4) done    (exit status 0),
 duration 00:00:01,  log batch.hi_rho_v.log
#5 (hi_rh0_lo_v/4) started on Tue Apr 13 13:59:40 2010
#5 (hi_rh0_lo_v/4) done    (exit status 0),  duration 00:00:01,  log batch.hi_rh0_lo_v.log
```

作业状态变更信息将持续打印，直到：

```
All jobs finished, total time  00:00:08Log files:
batch.reference.log batch.hi_v.log batch.lo_v.log batch.hi_rho.log batch.hi_rho_v.log batch.hi_
→rh0_lo_v.logBye.
```

分离作业输出文件

在模拟过程中，可能会将数据输出到外部文件（使用 *VTKRecorder* 等类），因此，文件名应采用不会被同一批处理中后续作业（或并发作业）覆盖的命名方式。为此提供了一个特殊标签 `O.tags['id']`，它由日期、时间和进程 ID 组成，因此始终唯一（例如 `20100413T144723p7625`）；此外，按字母顺序排列 `id` 标签也就是按时间顺序排列。若要将所用参数集或作业描述（如果已设置）添加到文件名中，可以添加 O.tags[‘params’] 。

对于规模较小的模拟，在所有输出文件名前加上 `O.tags['id']` 就足够了：

```
saveGnuplot(O.tags['id'])
```

对于规模较大的模拟，建议先创建一个同名的单独目录，然后再将所有文件放入其中：

```
os.mkdir(O.tags['id'])O.engines=[# …
VTKRecorder(fileName=O.tags['id']+'/'+'vtk'),# …]# …
O.saveGnuplot(O.tags['id']+'/'+'graph1')
```

控制并行计算

可用核心的默认总数由 `/proc/cpuinfo` （由 Linux 内核提供）决定；此外，如果设置了 `OMP_NUM_THREADS` 环境变量，则取这两个值中的最小值。`-j`/`--jobs` 选项可用于覆盖此数量。

默认情况下，每个作业都会使用所有可用核心，因此各作业实际上会并行运行。可通过 `--job-threads` 选项全局更改每个作业使用的核心数。

名为 `!OMP_NUM_THREADS` 的表格列（列名前加上 `!` 通常表示设置环境变量，而非 Python 变量）用于分别控制每个作业的线程数（如果该列存在）。

如果某个作业的核心数超过核心总数，系统会发出警告，并改用核心总数。

合并各个作业生成的 gnuplot

为了便于比较，我们通常希望获得一张包含批处理中所有作业的图形。批处理系统提供了一种启发式方法来实现这一功能。 `yade-batch` 必须使用 `--gnuplot` 选项运行，并指定一个文件名，用于保存合并后的图形：

```
yade-trunk --gnuplot merged.gnuplot batch.table batch.py
```

在模拟过程中，照常收集数据（使用 *plot.addData*），并通过*plot.saveGnuplot* 保存到 gnuplot 文件（它会创建两个文件：gnuplot 命令文件和压缩数据文件）。作业完成后，批处理系统会扫描日志文件，查找以下形式的行 `gnuplot [something]`。因此，为了打印这条特殊行，我们添加：

```
print('gnuplot',plot.saveGnuplot(O.tags['id']))
```

并在脚本末尾（甚至在 waitIfBatch() 之后）添加，这会打印：

```
gnuplot 20100413T144723p7625.gnuplot
```

到输出中（重定向到日志文件）。

此文件本身包含一张图：

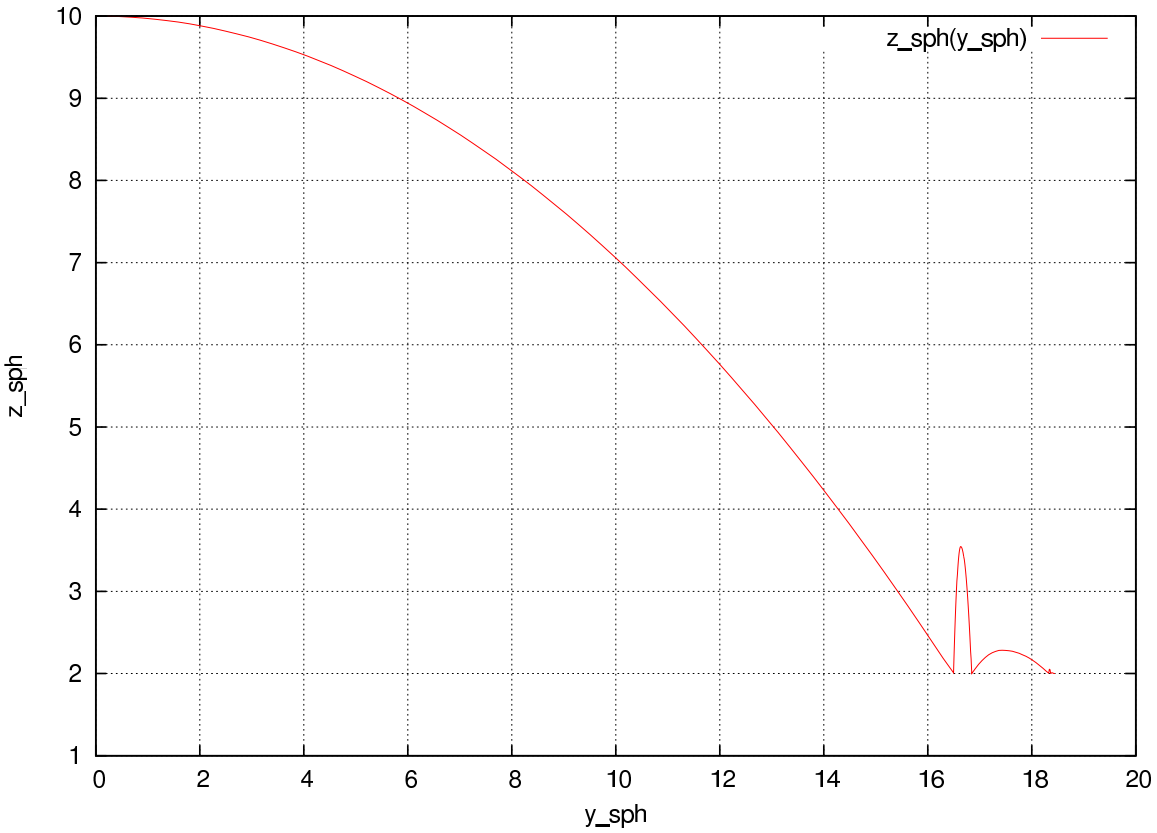


图 16：批处理中的单个作业生成的图形。

最后，批处理系统会识别全部 gnuplot 文件，并尝试通过汇总 `merged.gnuplot`文件将它们合并。

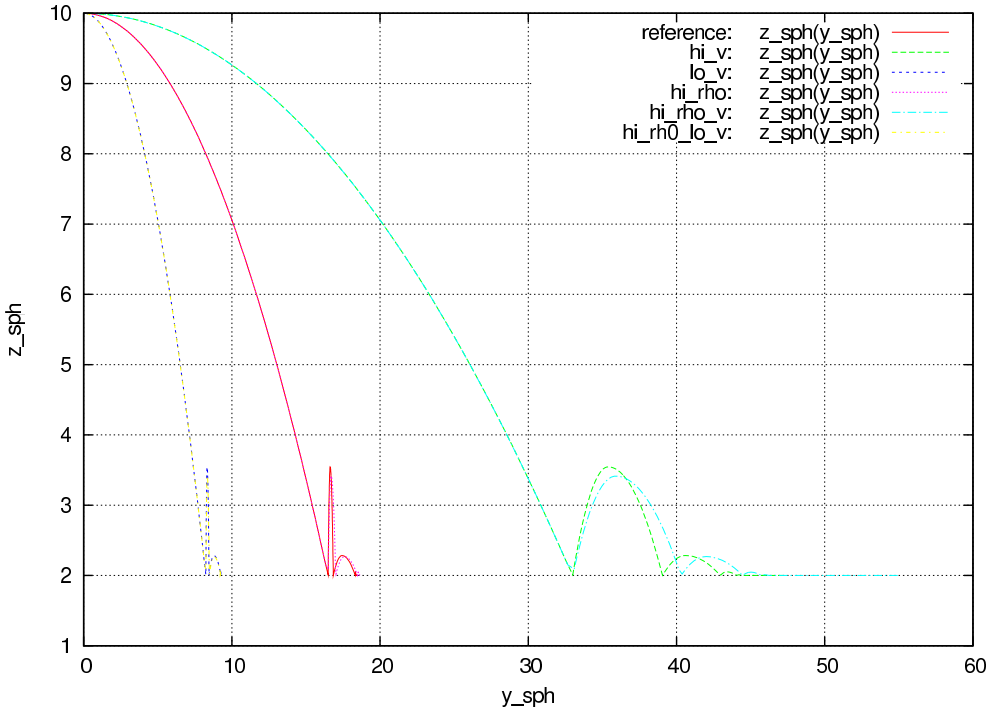


图17：批处理中所有作业合并后的图形。请注意，标签前会加上作业描述，以便区分各条直线。

HTTP 概述

作业运行期间，批处理系统会通过运行在 9080 端口上的简易 HTTP 服务器提供进度信息。普通网页浏览器可以通过请求`http://localhost:9080` URL 访问此页面（例如，使用终端时可用 `lynx` ）。此页面也可通过网络远程访问。

Running for 00:10:19, since Tue Apr 13 16:17:11 2010.

Pid 9873

4 slots available, 4 used, 0 free.

Jobs

4 total, **2** running, **1** done

id	status	info	slots	command
_geomType=B	00:10:19	96.33% done step 9180/9530 avg 14.9596/sec 10267 bodies 65506 intrs	2	PARAM_TABLE=iParams.table:2 DISPLAY= /usr/local/bin/yade-trunk --threads=2 --nice=10 -x indent.py > indent._geomType=B.log 2>&1
_geomType=smallA	00:09:53	(no info)	2	PARAM_TABLE=iParams.table:3 DISPLAY= /usr/local/bin/yade-trunk --threads=2 --nice=10 -x indent.py > indent._geomType=smallA.log 2>&1
_geomType=smallB	00:00:24	6.95% done step 694/9985 avg 35.8212/sec 9021 bodies 58352 intrs	2	PARAM_TABLE=iParams.table:4 DISPLAY= /usr/local/bin/yade-trunk --threads=2 --nice=10 -x indent.py > indent._geomType=smallB.log 2>&1
_geomType=smallC	(pending)	(no info)	2	PARAM_TABLE=iParams.table:5 DISPLAY= /usr/local/bin/yade-trunk --threads=2 --nice=10 -x indent.py > indent._geomType=smallC.log 2>&1

图 18：批处理过程中可在 9080 端口访问的摘要页面（每 5 秒自动更新一次）。可能的作业状态包括等待中、运行中、已完成和失败。

基于作业的集群 (OAR) 上的批处理执行

对于配有调度系统的高性能计算集群，以下脚本可能会很有用。与 yade-batch 完全一样，它会处理从参数表中将参数值分配给模拟脚本中的 Python 变量，以及作业提交。此脚本是为 基于 OAR 的系统编写的，也可以扩展到其他系统。在这些系统上，作业通常不能无限期运行，并且有特定的时长分配。整个作业提交由 3 个文件组成：

模拟脚本： 常规 Yade 脚本，通过调用 *readParamsFromTable* 从参数表获取参数。为了使脚本在批处理系统之外也能运行，*readParamsFromTable*会为参数采用默认值，这些值可能会被参数表中的值覆盖。

> *readParamsFromTable* 通过检查由批处理系统设置的 `PARAM_-TABLE` 环境变量，确定要读取哪个参数文件以及哪一行。

参数表： 简单的文本文件，每行表示一个参数集。此文件由*readParamsFromTable* (通过 *TableParamReader* 类）读取，并由模拟脚本调用，如上文所述。为了便于阅读文本文件，可以使用制表符；*readParamsFromTable*会忽略这些制表符。参数不限于数值。也可以通过 `"quoting"` 它们（ `' '` 也可以替代 `" "`）使用字符串。这对于名义参数很有用。

作业脚本： Bash 脚本，用于在计算节点上调用 yade。该脚本最终会创建临时文件夹、将数据保存到存储服务器等。脚本必须采用模板格式，其中某些标签会在执行时被特定值替换：

- `__YADE_COMMAND__` 将替换为实际的 Yade 运行命令
- `__YADE_LOGFILE__` 将替换为日志文件路径（输出到标准输出）
- `__YADE_ERRFILE__` 将替换为错误文件路径（输出到标准错误）
- `__YADE_JOBNO__` 将替换为由（启动脚本进程 ID）-（作业序号）组成的标识符
- `__YADE_JOBID__` 将替换为由全部参数值组成的标识符

批处理可以按如下方式运行

```
yade-oar --oar-project=<your project name> --oar-script=job.sh --oar-walltime=hh:mm:ss␣
↪parameters.table simulation.py
```

并且会为参数表中的每一行生成一个启动脚本并提交一个作业。一个最简示例见 examples/oar/params.tableexamples/oar/job.sh 和 examples/oar/sim.py。

注意： 必须指定 –oar-walltime 或在 params.table 中添加 !WALLTIME 列。!WALLTIME 会覆盖 –oar-walltime

---

警告： 默认情况下不会编译 yade-oar。使用 CMake 的 -DENABLE_OAR=1选项启用它。

2.2.3 后处理

三维渲染与视频

有多种方法可以生成模拟视频：

1. 在模拟过程中捕获屏幕输出（三维渲染窗口） – 有一些工具可用于此目的（例如 Istanbul 或 RecordMyDesktop，它们也为大多数 Linux 发行版打包提供）。输出效果是“所见即所得”，既有其优点，也有其缺点。
2. 使用 *SnapshotEngine* 定期生成帧快照（参见examples/test/force-network-video.py、exam-ples/bulldozer/bulldozer.py或 examples/test/beam-l6geom.py ，其中包含完整示例）：

   ```
   O.engines=[
     #...
     SnapshotEngine(iterPeriod=100,fileBase='/tmp/bulldozer-',viewNo=0,label='snapshooter')
   ]
   ```

   这会保存编号文件，例如 `/tmp/bulldozer-0000.png`。这些文件可以使用外部工具（如 mencoder 及类似工具）处理，也可以直接使用 *makeVideo*：

   ```
   makeVideo(frameSpec,out,renameNotOverwrite=True,fps=24,kbps=6000,bps=None)
   ```

   视频使用默认的 mencoder 编解码器（MPEG-4）进行编码。
3. 专用后处理工具，尤其是 ParaView。更多详情将在下一节中介绍。

ParaView

在仿真过程中保存数据

ParaView 基于 可视化工具包，它定义了用于保存各种类型数据的格式。其中一种格式（带有 `.vtu` 扩展名）可由一个专用引擎 *VTKRecorder*写入。它被添加到模拟循环中：

```
O.engines=[
        # ...
        VTKRecorder(iterPeriod=100,recorders=['spheres','facets','colors'],fileName='/tmp/p1-')
]
```

- *iterPeriod* 决定保存仿真数据的频率（除了 *iterPeriod*，也可以使用*virtPeriod* 或 *realPeriod*）。如果周期太长（数据只保存了几次），视频中的帧数就会很少。

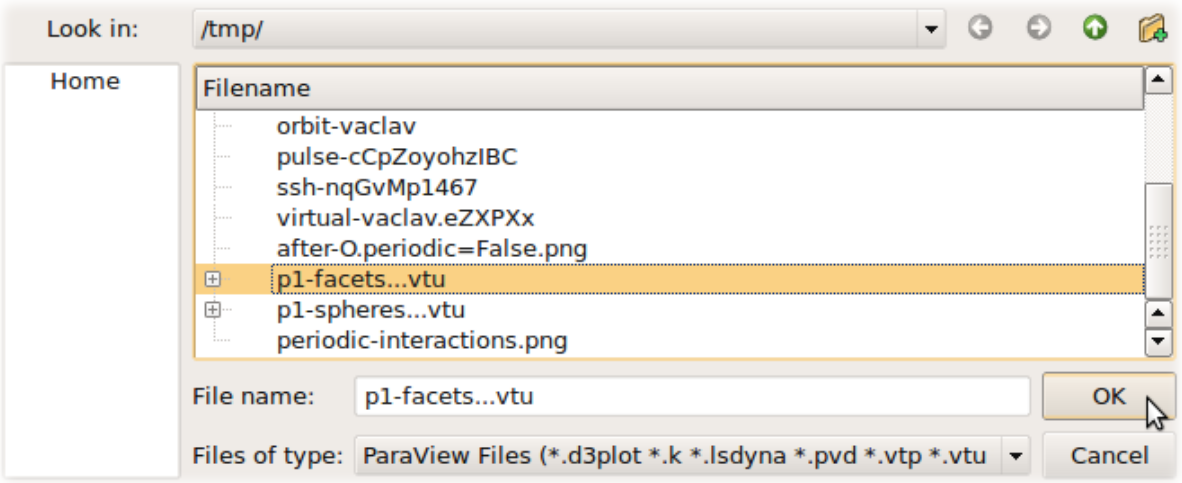
- *fileName* 是保存文件的前缀。在本例中，输出文件将命名为 `/tmp/p1-spheres.0.vtu` 和 `/tmp/p1-facets.0.vtu`，其中数字表示迭代次数；由于会生成许多文件，建议将它们放在单独的目录中。

- 记录器 决定要保存哪些数据

export.VTKExporter 作用类似，但差别在于它更加灵活。它会保存与刚体关联的任何用户定义变量。

将数据加载到 ParaView

所有文件集（`spheres`、 `facets`.....）都必须在 ParaView 中逐一打开。打开对话框会自动将编号文件合并为一组，方便选择全部文件：

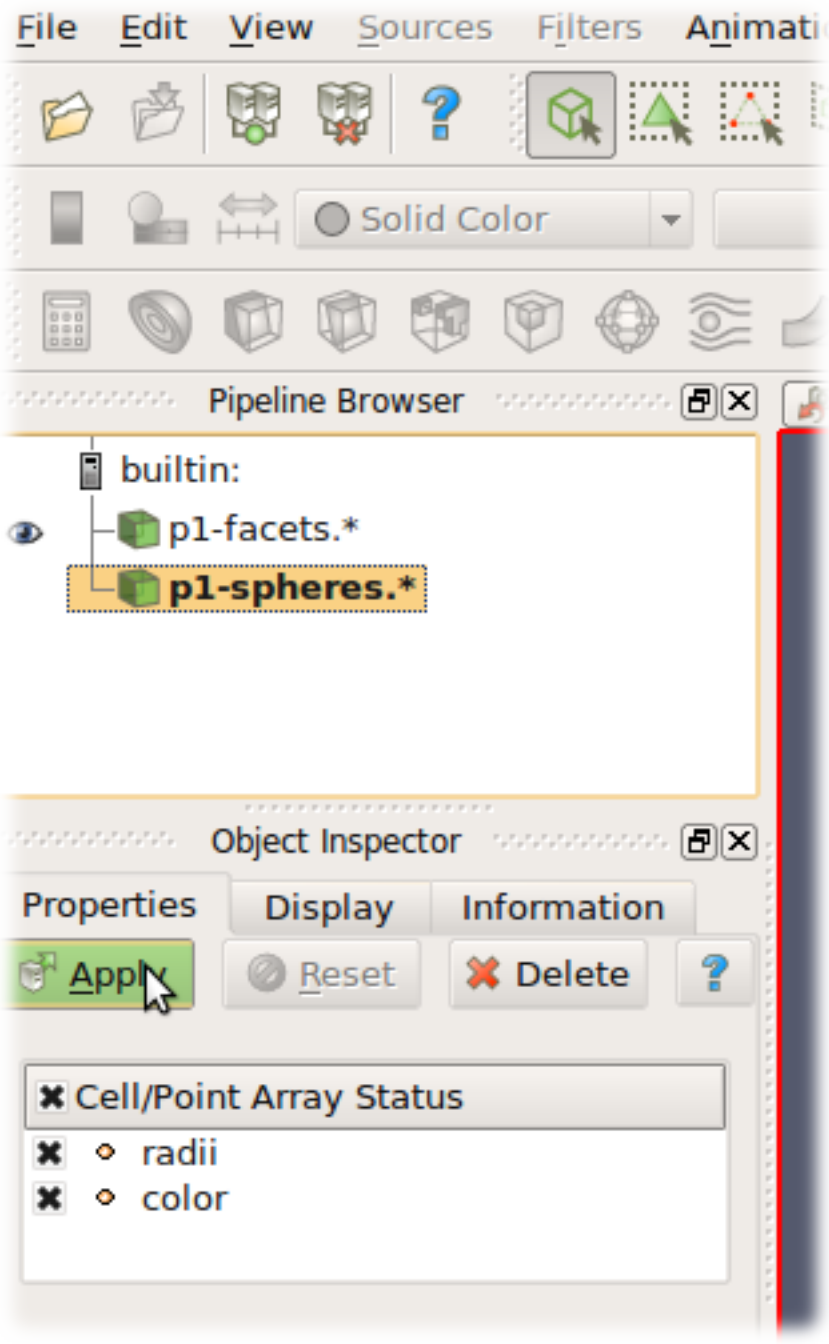


点击“对象检查器”子窗口中的“应用”按钮，使加载的对象可见。你可以在“管线浏览器”中查看已显示对象的树状结构：

渲染球形颗粒。字形

球体只会显示为点。若要使它们看起来像球体，必须向其中添加“图元”

`p1-spheres.*` 使用流水线中的项目  图标。然后（在对象检查器中）设置


File
Edit
View
Sources
Filters
Animati
Solid Color
Pipeline Browser
builtin:
p1-facets.*
p1-spheres.*
Object Inspector
Properties
Display
Information
Apply
Reset
Delete
Cell/Point Array Status
radii
color

- 将“字形类型”设为球体
- 将“半径”设为*1*
- 将“缩放模式”设为标量 （标量 已在上方设为文件中保存的半径 值，因此半径为*1* 的球体将按其实际半径进行缩放）
- 将“设置缩放因子”设为*1*
- 可选择取消勾选“屏蔽点”和“随机模式”（出于性能考虑，它们会使部分颗粒不被渲染；具体数量由“最大点数”控制）

点击“应用”后，球体便会显示出来。它们会渲染在原始白色点的上方；你可以点击管线浏览器中 `p1-spheres.*` 旁边的眼睛图标来隐藏这些白色点。

渲染球形颗粒。PointSprite

显示球体的另一种方式是使用 *PointSprite* 插件。与字形相比，这种技术所需的内存要少得多。

- “工具 ->管理插件”
- “PointSprite_插件 -> 加载所选项 -> 关闭”
- 加载 VTU 文件
- “表示方式 -> 点精灵”
- “点精灵 -> 按比例缩放 -> 半径”
- “编辑半径传递函数 -> 成比例 -> 乘数 = 1.0 -> 关闭”

将相互作用渲染为力链

由 `VTKRecorder` （以下步骤生成的是圆锥体而非管状体）或 `export.VTKExporter(...).exportInteractions(what=dict(forceN='i.phys.normalForce.norm()'))` （以下步骤生成的是每个相互作用的恒定半径管状体）保存的数据：

- 加载相互作用 VTP 或 VTK 文件
- 筛选器 -> 单元数据转点数据
- 筛选器 -> 管状体
- 按“forceN”设置颜色
- 将“变化半径”设为“按标量”
- 设置“半径”和“半径系数”，使结果看起来合适（在三维后处理教程脚本中，Radius=0.0005 和 Radius Factor=100 看起来比较合适）

面片透明度

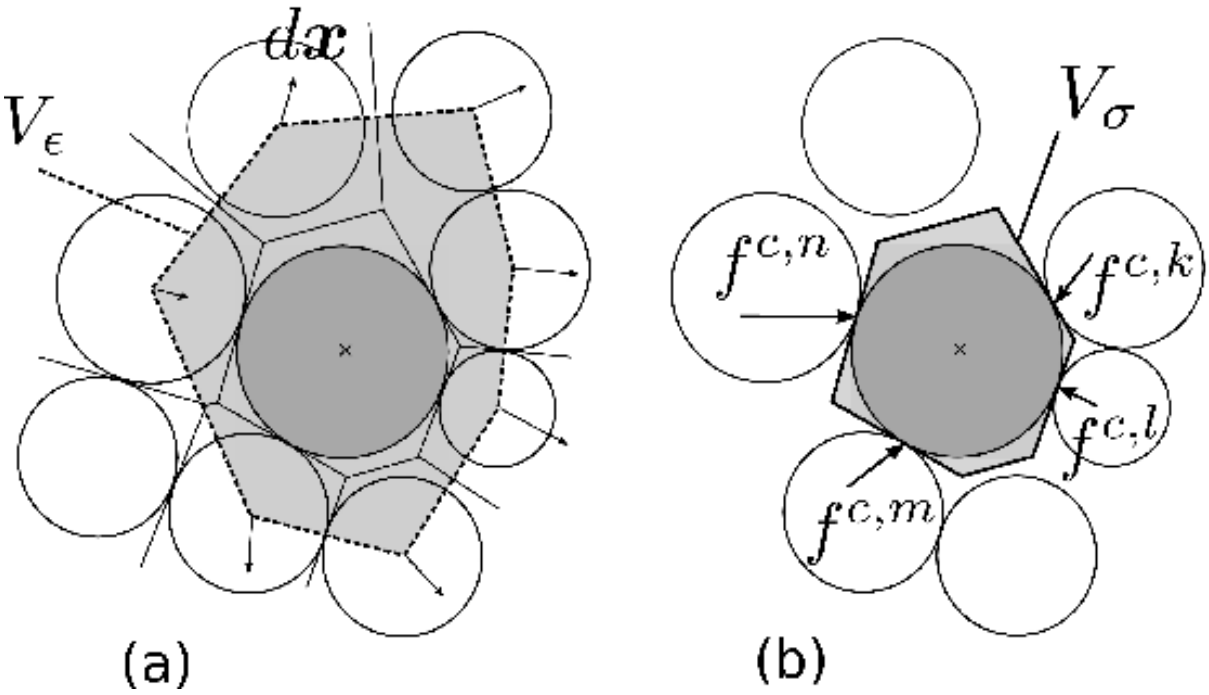
如果想让面片对象变得透明，请在管线浏览器中选择 `p1-facets.*` ，然后转到显示选项卡中的对象检查器。在“样式”下，可以将“不透明度”值设为小于 1 的数值。

动画

您可以在帧之间移动视角 动（已保存的快照），方法是使用“动画”菜单。设置视 e ，缩放等操作达到满意效果后，可通过文件/保存动画保存动画。

微观应力与微观应变

有时，通过等效应变场或应力场来可视化离散元模拟会很有用。使用 *TesselationWrapper*即可实现。此类负责对场景中的球体进行三角剖分，按需构建剖分，并提供计算所得物理量的访问途径：每个球体的体积、孔隙率和局部变形。微观应变和微观应力定义在下方所示的以颗粒为中心的子域尺度上，详见 [Catalano2014a] 。



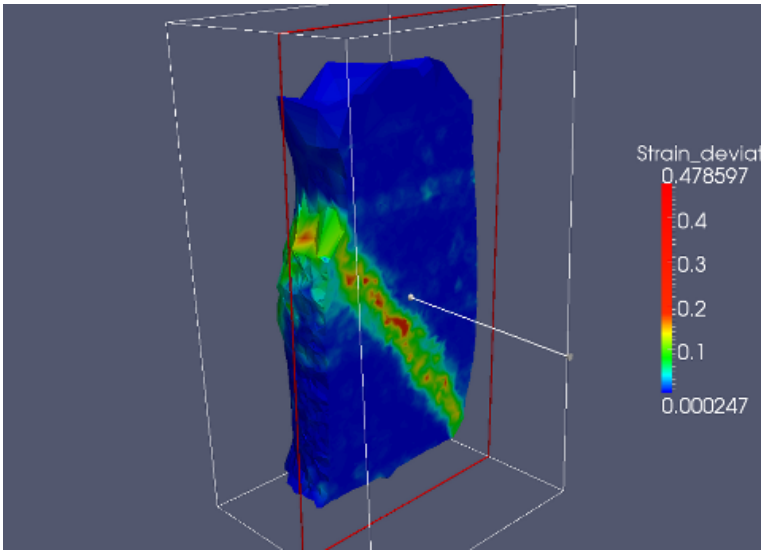
微观应变

下面是使用 paraview 可视化的 *defToVtk* 函数输出（本例中使用 Yade 的 Tessela- tionWrapper 处理了 Edward Ando 在格勒诺布尔大学 3SR实验室获取的砂样实验数据）。如上一节所述，该输出使用 paraview 进行可视化。模拟也可以生成类似的结果：

```
tt=TriaxialTest()
tt.generate("test.yade")
O.load("test.yade")
O.run(100,True)
TW=TesselationWrapper()
TW.triangulate()        #compute regular Delaunay triangulation, don't construct tesselation
TW.computeVolumes()     #will silently tesselate the packing, then compute volume of each␣
 →Voronoi cell
TW.volume(10)           #get volume associated to sphere of id 10
TW.setState(0)          #store current positions internaly for later use as the "0" state
O.run(100,True)         #make particles move a little (let's hope they will!)
TW.setState(1)          #store current positions internaly in the "1" (deformed) state
#Now we can define strain by comparing states 0 and 1, and average them at the particles scale
TW.defToVtk("strain.vtk")
```

微观应力

应力场可以通过将 TesselationWrapper 返回的体积与 *bodyStressTensors*给出的单颗粒应力相结合来生成。由于应力 σ 由 bodyStressTensor 得出时隐含着除以



固体颗粒的体积 V_b，因此必须对其重新归一化，以获得 [Catalano2014a]（其中的方程 39）所定义的微观应力，即 $\sigma^k = \sigma^k \times V_b^k/V_\sigma^k$ 其中 V_σ^k 是剖分中分配给颗粒 k 的体积。例如：

```
#"b" being a body
TW=TesselationWrapper()
TW.setState()
TW.computeVolumes()
s=bodyStressTensors()
stress = s[b.id]*4.*pi/3.*b.shape.radius**3/TW.volume(b.id)
```

与其他值一样，应力也可以使用 *ex-port.VTKExporter* 导出为 VTK 文件，以便在 ParaView 中显示。

2.2.4 Python 特性与技巧

在其他 Python 应用程序中导入 Yade

Yade 可以导入到其他 Python 应用程序中。为此，需要以某种方式将 yade 可执行文件扩展为 .py。最简单的方法是创建一个符号链接，即（假设你的 Yade 可执行文件名为“yade-trunk”，并且你想将其命名为“yadei mport.py”）：

```
$ cd /path/where/you/want/yadeimport
$ ln -s /path/to/yade/executable/yade-trunk yadeimport.py
```

然后，你需要让 Python 能够找到 yadeimport.py。你可以导出 PYTHONPATH 环境变量，也可以直接在 Python 脚本中使用 sys.path：

```
import sys
sys.path.append('/path/where/you/want/yadeimport')
from yadeimport import *

print(Matrix3(1,2,3, 4,5,6, 7,8,9))
print(O.bodies)
# any other Yade code
```

2.2.5 扩展 Yade

- 新颗粒形状

- 新的本构定律

2.2.6 故障排除

崩溃

有　可能会遇到 Yade 崩溃的情况，即 Yade 终止运行并显示如下错误信息：

```
Segmentation fault (core dumped)
```

但没有进一步的说明。这类情况的常见原因包括

- Yade 本身的程序错误；
- 特定模拟中的致命状况（例如无法进行分派）；
- 显卡驱动程序问题。

尝试使用启用调试的 Yade 版本重现该错误（运行相同的脚本）。发生崩溃时，调试器会自动启动，并显示代码的回溯（本例中，我们手动触发了崩溃）：

```
Yade [1]: import os,signalYade [2]: os.kill(os.getpid(),signal.SIGSEGV)
SIGSEGV/SIGABRT handler called; gdb batch file is `/tmp/yade-YwtfRY/tmp-0'
GNU gdb (GDB) 7.1-ubuntuCopyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".For bug reporting instructions,
 please see:<http://www.gnu.org/software/gdb/bugs/>.
[Thread debugging using libthread_db enabled]
[New Thread 0x7f0fb1268710 (LWP 16471)][New Thread 0x7f0fb29f2710 (LWP 16470)]
[New Thread 0x7f0fb31f3710 (LWP 16469)]...
```

看似晦涩难懂的信息，对开发者定位缺陷根源很有帮助。具体来说，通常会显示第 `<signal handler called>` 行；其下方的代码行就是缺陷的来源（至少很可能是）：

```
Thread 1 (Thread 0x7f0fcee53700 (LWP 16465)):
#0  0x00007f0fcd8f4f7d in __libc_waitpid (pid=16497, stat_loc=<value optimized out>,␣
→options=0) at ../sysdeps/unix/sysv/linux/waitpid.c:41
#1  0x00007f0fcd88c7e9 in do_system (line=<value optimized out>) at ../sysdeps/posix/system→c:149
#2  0x00007f0fcd88cb20 in __libc_system (line=<value optimized out>) at ../sysdeps/posix/
→system.c:190#3  0x00007f0fcd0b4b23 in crashHandler (sig=11) at core/main/pyboot.cpp:45
#4  <signal handler called>
#5  0x00007f0fcd87ed57 in kill () at ../sysdeps/unix/syscall-template.S:82
#6  0x000000000051336d in posix_kill (self=<value optimized out>, args=<value optimized out>)␣
→at ../Modules/posixmodule.c:4046
#7  0x00000000004a7c5e in call_function (f=Frame 0x1c54620, for file <ipython console>, line 1,
→ in <module> (), throwflag=<value optimized out>) at ../Python/ceval.c:3750
#8  PyEval_EvalFrameEx (f=Frame 0x1c54620, for file <ipython console>, line 1, in <module> (),␣
→throwflag=<value optimized out>) at ../Python/ceval.c:2412
```

如果您认为这可能是 Yade 中的错误，请按照下面的说明提交错误报告。请务必附上从终端获取的完整 Yade 输出，包括启动消息和调试器输出——使用鼠标右键选择，用鼠标中键将错误报告粘贴到文件中，然后附加该文件。同时也请附上您的模拟脚本。

报告缺陷

缺陷是对问题（功能缺陷、文档错误、崩溃）或功能请求的统称。相关问题会在 https://gitlab.com/yade-dev/trunk/issues上跟踪。
报告新缺陷时，请尽可能具体；请说明所用的 Yade 版本、系统版本，以及*printAllVersions()*的输出，详情请参阅上文关于崩溃的章节。

获取帮助

问答

提示： P请使用 Launchpad 问答界面 https://answers.launchpad.net/yade/ 来提出问题 s
关于 Yade e.

如果您不熟悉计算机相关讨论邮件列表，请先阅读此 Wiki 页面（这是面向 Yade 的精简版 《如何聪明地提问》），再发布问题，以提高获得帮助的机会。请务必说明您使用的 Yade 版本 （启动 Yade 时会显示；更好的方式是使用函数 *libVersions.printAllVersions*打印版本信息），以及您是从源代码还是软件包安装的、所用的操作系统（例如 Ubuntu 18.04），如果使用编译版本，还要说明是否对源代码做过本地修改。

邮件列表

除了 Launchpad 问答界面外，Yade 还有两个邮件列表。它们都托管在 http://www.launchpad.net 上。发布消息前，您必须先注册 Launchpad，并通过将自己加入负责运行该列表的同名“团队”来订阅该列表。

yade-users@lists.launchpad.net 是面向所有 Yade 用户的一般讨论列表。请加入 yade-usersteam ，以便发布消息。邮件列表存档:

- https://lists.launchpad.net/yade-users/
- http://www.mail-archive.com/yade-users@lists.launchpad.net/

yade-dev@lists.launchpad.net 用于讨论 Yade 开发；您必须是yade-dev team 的成员才能发布消息。此列表存档于两个位置:

- https://lists.launchpad.net/yade-dev/
- http://www.mail-archive.com/yade-dev@lists.launchpad.net/

Wiki

http://www.yade-dem.org/wiki/

私人和／或付费支持

如果您不想在邮件列表中透露详情，可以通过私人邮件联系开发者，而不是在邮件列表中联系他们。这也是提议为

实现 Yade 尚未具备的重要功能提供经济报酬的合适方式。不过通常情况下，我们会要求在完成后将该功能纳入公开代码库，以便社区中的其他人也能从中受益。

2.3 Yade 包装类参考

2.3.1 刚体

Body

class yade.wrapper.Body(继承可序列化对象)模拟的基本元素颗粒；与其他刚体相互作用。

aspherical(=否)此刚体沿主轴方向的惯性是否不同；*NewtonIntegrator* 使用此标志调用非球形体的旋转积分例程，这种操作开销更大。

bound(=未初始化)*Bound*，用于碰撞检测时近似体积。

bounded(=是)创建此刚体时是否应创建*Body.bound* 。请注意，没有 *bound* 的刚体不会参与碰撞检测。（在 c++中，使用 Body::isBounded/Body::setBounded）

chain返回该刚体所属链的标识符。

clumpId
该刚体所属团簇的标识符；若不属于任何团簇，则为无效数值；参见*Body::isStandalone*、*Body::isClump*、*Body::isClumpMember* 属性。

不应直接从 Python 修改，请改用*O.bodies.appendClumped* 。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

dynamic(=是)此刚体是否会受力移动。（在 c++ 中，使用Body::isDynamic/Body::setDynamic）

flags(=标志_有界)各种刚体相关标志的位。请勿直接访问。在 C++中，使用 isDynamic/setDynamic、isBounded/setBounded、isAspherical/setAspherical。在 Python 中，使用*Body.dynamic*、*Body.bounded*、*Body.aspherical*。

groupMask(=*1*)
用于相互作用检测的位掩码：从碰撞器的角度来看，两个刚体的 groupMask 必须至少有一位相同，才能发生相互作用。

id(=*Body::ID_无*)此刚体的唯一标识符。

intrs(*(Body)arg1*) → 列表：返回此刚体参与的所有实际相互作用的列表。

isClump
如果此刚体本身是团簇，则为 True；否则为 False。

isClumpMember
如果该刚体是团簇成员，则为真；否则为假。

isStandalone
如果该刚体既不是团簇，也不是团簇成员，则为真；否则为假。

iterBorn(=*-1*)该刚体加入模拟时的步数。**mask**简写形式：*Body::groupMask*

mat简写形式：*Body::material*

material(=未初始化)与此刚体关联的材料实例。**shape**(=未初始化)几何形状。**state**(=新的状态)物理状态。**timeBorn**(=*-1*)该刚体加入模拟时的时间。**updateAttrs**(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → 无：根据给定的字典更新对象属性

形状

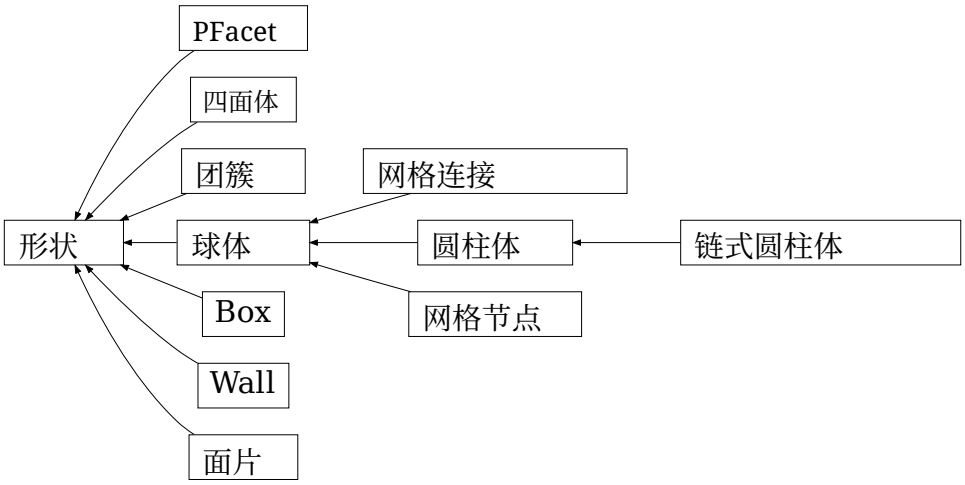


图 19：形状的继承图。另请参阅：盒子、链式圆柱体、团簇、圆柱体、面片、网格连接、网格节点、*PFacet*、球体、四面体、墙。

class yade.wrapper.Shape(继承可序列化对象)刚体的几何体**color**(=*Vector3r(1，1，1)*)渲染颜色（归一化 RGB）。**dict**(*(*可序列化对象*)*参数*1*) →字典：返回属性字典。**dispHierarchy**(*(*形状*)*参数*1*[，*(*布尔值*)*名称=真]) → 列表：返回分派类列表（从下至上），以类实例本身开头，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex返回此实例的类索引。**highlight**(=假)渲染此形状时是否高亮显示。**updateAttrs**(*(*可序列化对象*)*参数*1*，*(*字典*)*参数*2*) → 无：根据给定字典更新对象属性**wire**(=假)此形状是否使用颜色曲面渲染，或仅以线框渲染（仍可由渲染器的全局配置覆盖）。

class yade.wrapper.**Box**(继承 形状 →可序列化对象)**color**(=*Vector3r(1，1，1)*)渲染颜色（归一化 RGB）。**dict**(*(*可序列化对象*)*参数*1*) →字典：返回属性字典。**dispHierarchy**(*(*形状*)*参数*1*[, *(bool)*名称=*True*]) → 列表：返回分派类列表（从下至上），首先是类实例本身，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。**extents**(=取消初始化)长方体的半尺寸**highlight**(=*false*)渲染时是否高亮显示此形状。**updateAttrs**(*(*可序列化对象*)*参数*1*，*(*字典*)*参数*2*) → None：根据给定字典更新对象属性**wire**(=*false*)此形状是否使用颜色曲面进行渲染，或者仅以线框渲染（仍可由渲染器的全局配置覆盖）。

class yade.wrapper.**ChainedCylinder**(继承 圆柱体 → 球体 → 形状 → 可序列化对象)可变形链式圆柱体的几何体，使用几何体 圆柱体。**chainedOrientation**(=*Quaternionr::Identity()*)节点1方向相对于节点间向量的偏差**color**(=*Vector3r(1，1，1)*)渲染颜色（归一化 RGB）。**dict**(*(*可序列化对象*)*参数*1*) →字典：返回属性字典。**dispHierarchy**(*(*形状*)*参数*1*[, *(bool)*名称=*True*]) → 列表：返回分派类列表（从下至上），首先是类实例本身，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。**highlight**(=否)渲染时是否高亮显示此形状。**initLength**(=*0*)无拉伸长度，用作拉伸应变的参考

length(=非数)长度 [米]radius(=非数)半径 [米]segment(=Vector3r::Zero())长度向量 updateAttrs((可序列化对象)参数1，(字典)参数*2*) → 无：根据给定字典更新对象属性wire(=否)此形状是否使用颜色曲面渲染，还是仅以线框渲染（仍可由渲染器的全局配置覆盖）。 class yade.wrapper.Clump(继承自形状 →可序列化对象)刚性颗粒聚集体color(=Vector3r(1, 1, 1))渲染颜色（归一化 RGB）。dict((可序列化对象)参数*1*) →字典：返回属性字典。dispHierarchy((形状)参数*1*[, (布尔值)名称=真]) → 列表：返回分派类列表（从下至上），从类实例本身开始，最后为顶层可索引对象。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。highlight(=否)渲染时是否高亮显示此形状。ids(=未初始化)构成颗粒的标识符（仅供参考；直接修改不会产生任何效果）。

members
以 {'id1':(relPos,relOri),...} 的形式返回团簇成员updateAttrs((可序列化对象)参数*1*，(字典)辐角*2*) → 无：根据给定字典更新对象属性wire(=否)此形状是否使用颜色曲面渲染，或仅使用线框渲染（仍可由渲染器的全局配置覆盖）。class yade.wrapper.Cylinder(继承 球体 → 形状 → 可序列化对象)圆柱体几何结构，是直线与球体的闵可夫斯基和。color(=三维向量(*1*，*1*，*1*))用于渲染的颜色（归一化 RGB）。dict((可序列化对象)辐角*1*) →字典：返回属性字典。dispHierarchy((形状)辐角*1*[, (布尔值)名称=真]) → 列表：返回分派类列表（从低到高），最后以顶层可索引对象开头，即类实例本身。如果 names 为真（默认值），则返回类名而不是数值索引。

dispIndex
返回此实例的类索引。highlight(=否)渲染时是否高亮显示此形状。

`length`(=非数)长度 [米]`radius`(=非数)半径 [米]`segment`(=Vector3r::Zero())长度向量 `updateAttrs`((可序列化对象)参数1, (字典)辐角*2*) → 无：根据给定字典更新对象属性`wire`(=否)此形状是否以颜色曲面渲染，还是仅以线框渲染（仍可由渲染器的全局配置覆盖）。

`class yade.wrapper.Facet`(继承自 Shape →可序列化对象)面片（三角形颗粒）几何体。`area`(=非数)面片的面积`color`(=Vector3r(1, 1, 1))渲染颜色（归一化 RGB）。`dict`((可序列化对象)辐角*1*) → 字典：返回属性字典。`dispHierarchy`((形状)辐角*1*[, (布尔值)名称=真]) → 列表：返回分派类列表（从下往上），从类实例本身开始，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名，而非数值索引。

`dispIndex`

返回此实例的类索引。`highlight`(=否)渲染时是否高亮显示此形状。`normal`(=Vector3r(NaN, NaN, NaN))面片的法向量（在局部坐标系中）`setVertices`((Facet)arg1, (Vector3)arg2, (Vector3)arg3, (Vector3)arg*4*) → 无：待办`updateAttrs`((Serializable)arg1, (dict)arg*2*) → 无：根据给定字典更新对象属性`vertices`(=向量<Vector3r>(3, Vector3r(NaN, NaN, NaN)))局部坐标中的顶点位置。`wire`(=否)此形状是否以彩色曲面渲染，还是仅以线框渲染（仍可由渲染器的全局配置覆盖）。

`class yade.wrapper.GridConnection`(继承 Sphere → 形状 → 可序列化对象)网格连接形状（参见 [Effeindzourou2016], [Bourrier2013]）。网格连接是网格中用于连接两个网格节点的组件。强烈建议使用 gridpfacet.gridConnection 来生成正确的网格连接。`addPFacet`((GridConnection)arg1, (Body)刚体) → 无：向网格连接添加一个 PFacet。`cellDist`(=Vector3i(0, 0, 0))如果使用周期性边界条件，刚体间距以单元格尺寸为单位。请注意，网格连接的周期性边界条件尚未完全实现。

color(=*Vector3r(1，1，1)*)
渲染颜色（归一化 RGB）。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

dispHierarchy(*(*形状*)*辐角 *1*[, *(*布尔值*)*名称=*True*]) → 列表：返回分派类列表（从下至上），从类实例本身开始，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

getPFacets(*(*网格连接*)*辐角 *1*) →对象：获取链接的 PFacet 列表。

highlight(=否)渲染时是否高亮显示此形状。

node1(=未初始化)第一 刚体 网格连接所连接的对象。

node2(=未初始化)第二 刚体 网格连接所连接的对象。

periodic(=否)如果连接了来自不同周期的两个节点，则为 true。

radius(=非数)
半径 [米]

updateAttrs(*(*可序列化对象*)arg1*，*(*字典*)*辐角 *2*) → 无:
根据给定的字典更新对象属性

wire(=*false*)此形状是否使用彩色曲面渲染，或者仅以线框渲染（仍可由渲染器的全局配置覆盖）。

class yade.wrapper.GridNode(继承 球体 → 形状 → 可序列化对象)
网格节点形状，是网格的组成部分。要创建网格，请先放置节点，它们将定义网格的空间离散化。强烈建议使用 *gridpfacet.gridNode* 来生成正确的网格节点。请注意，网格节点只应与其他网格节点发生相互作用。球体-网格接触仅由 网格连接处理。

addConnection(*(*网格节点*)*参数*1*, *(*刚体*)*刚体）→ None：向网格节点添加一个网格连接。

addPFacet(*(*网格节点*)*参数*1*,*(*刚体*)*刚体）→ None：向网格节点添加一个 PFacet。

color(=*Vector3r(1, 1, 1)*)用于渲染的颜色（归一化 RGB）。dict(*(*可序列化对象*)*辐角 *1*) →字典：返回属性字典。

dispHierarchy(（形状）辐角 *1*[,（布尔值）名称=真]) → 列表：返回分派类列表（从下到上），以类实例本身开头，顶层可索引对象结尾。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

getConnections(*(*网格节点*)*辐角 *1*) →对象：获取链接到网格连接的列表。

getPFacets(*(网格节点)*辐角 *1*) →对象：获取链接到*PFacet*的列表。**highlight**(=否)渲染时此形状是否会高亮显示。**radius**(=非数)半径 [米]**updateAttrs**(*(可序列化对象)*参数*1*，*(字典)*辐角 *2*) → 无：根据给定字典更新对象属性**wire**(=否)渲染此形状时是否使用颜色曲面，或仅使用线框（仍可由渲染器的全局配置覆盖）。

class yade.wrapper.PFacet(继承形状 → 可序列化对象)

PFacet（粒子面）几何体（参见 [Effeindzourou2016], [Effeindzourou2015a]）。强烈建议使用 *gridpfacet* 中的辅助函数（例如 gridpfacet.pfacetCreator1-4），以生成正确的*PFacet* 元素。

area(=非数)
PFacet 的面积

cellDist(=*Vector3i(0, 0, 0)*)使用周期性边界条件时，刚体之间的距离（以单元格尺寸为单位）。请注意，PFacet 的周期性边界条件尚未完全实现。

color(=*Vector3r(1, 1, 1)*)渲染颜色（归一化 RGB）。

conn1(=已取消初始化)第一 刚体 所连接的 PFacet。

conn2(=已取消初始化)第二 刚体 所连接的 PFacet。

conn3(=已取消初始化)第三 刚体 所连接的 PFacet。

dict(*(可序列化对象)*辐角 *1*) →字典：
返回属性字典。

dispHierarchy(*(形状)*辐角 *1*[, *(布尔值)*名称=是]) → 列表：返回分派类列表（从下至上），以类实例本身开头，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

highlight(=*false*)
渲染时是否高亮显示此形状。

node1(=已取消初始化)
第一刚体所连接的 PFacet。

node2(=已取消初始化)
第二刚体所连接的 PFacet。

node3(=已取消初始化)
第三个刚体所连接的 PFacet。

normal(=*Vector3r(*非数，非数，非数*)*)
PFacet 的法向量（在局部坐标系中）

radius(=*-1*)
PFacet 的半径

updateAttrs((可序列化对象)参数1, (字典)辐角2) → 无：根据给定字典更新对象属性wire(=false)此形状是否使用颜色曲面进行渲染，还是仅以线框方式渲染（仍可由渲染器的全局配置覆盖）。class yade.wrapper.Sphere(继承自 形状 → 可序列化对象)球形颗粒的几何体。color(=Vector3r(1, 1, 1))渲染颜色（归一化 RGB）。dict((可序列化对象)辐角1) →字典：返回属性字典。dispHierarchy((形状)辐角1[, (布尔值)名称=是]) → 列表：返回分派类列表（从下至上），以类实例本身开头，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex

返回此实例的类索引。highlight(=false)渲染时是否高亮显示此形状。radius(=非数)半径 [米]updateAttrs((可序列化对象)arg1、(字典)arg2) → None：根据给定的字典更新对象属性wire(=false)此形状是否使用彩色曲面渲染，还是仅以线框渲染（仍可由渲染器的全局配置覆盖）。class yade.wrapper.Tetra(继承形状 →可序列化对象)四面体几何。color(=Vector3r(1, 1, 1))渲染颜色（归一化 RGB）。dict((可序列化对象)arg1) →字典：返回属性字典。dispHierarchy((形状)arg1[, (布尔值)名称=True]) → 列表：返回分派类列表（从下至上），从类实例本身开始，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex

返回此实例的类索引。highlight(=false)渲染时是否高亮显示此形状。updateAttrs((可序列化对象)arg1，(字典)arg2) → 无：根据给定字典更新对象属性v(=std::vector<Vector3r>(4))局部坐标系中的四面体顶点。wire(=false)此形状是否以颜色曲面渲染，还是仅以线框渲染（仍可被渲染器的全局配置覆盖）。

class yade.wrapper.Wall(继承自形状 → 可序列化对象)表示与坐标系对齐的无穷平面（轴对齐墙）的对象。axis(=*0*)法向量的轴；可为 0、1、2，分别对应 +x、 +y、 +z（墙的方向不考虑刚体的方向）color(=Vector3r(1, 1, 1))渲染颜色（归一化 RGB）。dict((可序列化对象)arg*1*）→字典：返回属性字典。dispHierarchy((形状)arg*1*[, (布尔值)names=True])) → 列表：返回分派类列表（从下到上），从类实例本身开始，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。dispIndex返回此实例的类索引。highlight(=false)渲染时是否高亮显示此形状。sense(=*0*)与墙相互作用的一侧：-1 表示仅负侧，0 表示两侧， +1 表示仅正侧updateAttrs((可序列化对象)arg1，(字典)arg*2*) → 无：根据给定字典更新对象属性wire(=false)此形状是否以颜色曲面渲染，还是仅以线框渲染（仍可被渲染器的全局配置覆盖）。

状态

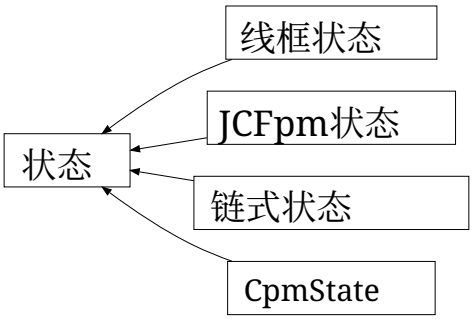


图 20：State 的继承图。另请参阅：链式状态、*CpmState*、*JCFpm*状态、线框状态。

class yade.wrapper.State(继承自可序列化对象)刚体的状态（空间构型、内部变量）。angMom(=*Vector3r::Zero()*)当前角动量angVel(=*Vector3r::Zero()*)当前角速度

blockedDOFs
: 线速度/角速度始终保持恒定（等于零向量或用户定义的值），不受施加的力/力矩影响的自由度。可包含 'xyzXYZ'（平移和旋转）的字符串。

densityScaling(=*1*)
(自动更新) 参见 *GlobalStiffnessTimeStepper::targetDt*。

dict(*(可序列化对象)*辐角 *1*) →字典:
返回属性字典。

dispHierarchy(（状态）辐角 *1*[, （布尔值）名称=真]) → 列表：返回分派类的列表（从下至上），从类实例本身开始，顶层可索引对象位于最后。如果 names 为真（默认值），则返回类名而不是数值索引。

dispIndex
返回此实例的类索引。

displ(（状态）辐角 *1*) → 三维向量：相对于参考位置（位置- *refPos*）

inertia(=*Vector3r::Zero()*)关联刚体在局部坐标系中的惯性。

isDamped(=*true*)
NewtonIntegrator 中的阻尼*NewtonIntegrator* 可通过将此变量设为 FALSE 来对单个颗粒停用。例如，阻尼不适用于在重力作用下自由飞行的颗粒，但仍可能适用于同一模拟中的其他颗粒。

mass(=*0*)此刚体的质量

ori当前方向。

pos当前位置。

refOri(=*Quaternionr::Identity()*)参考方向

refPos(=*Vector3r::Zero()*)参考位置

rot(*(State)*辐角 *1*) → 三维向量:
旋转自参考方向（以旋转向量表示）

se3(=*Se3r(Vector3r::Zero(), Quaternionr::Identity())*)
位置和方向作为一个对象。

updateAttrs(*(可序列化对象)arg1*, *(字典)*辐角 *2*) → 无:
从给定字典更新对象属性

vel(=*Vector3r::Zero()*)当前线速度。

class yade.wrapper.ChainedState(继承状态 →可序列化对象)
链式刚体的状态，其中包含连接关系信息，以便跟踪跨越连续单元的接触。链是一个一维列表，可通过秩和链编号获取链式刚体的标识符。

addToChain(*(链式状态)*参数*1*, *(int)*刚体标识符) → 无：将刚体添加到当前活动链angMom(=*Vector3r::Zero()*)当前角动量angVel(=*Vector3r::Zero()*)当前角速度bId(=*-1*)包含 - 的刚体标识符，仅用于加载后操作。

blockedDOFs
线速度/角速度始终保持恒定（等于零或用户定义的值）、不受施加的力/力矩影响的自由度。可包含“xyzXYZ”的字符串（平移和旋转）。

chainNumber(=*0*)链ID。

currentChain = 0

densityScaling(=*-1*)(自动更新)参见*GlobalStiffnessTimeStepper::targetDt*。

dict((可序列化对象)辐角*1*) → 字典：返回属性字典。

dispHierarchy((状态)辐角*1*[, (布尔值)名称=真]) → 列表：返回分派类列表（从下到上），从类实例本身开始，顶层可索引对象位于最后。如果 names 为 true（默认值），则返回类名而不是数值索引。

dispIndex
返回此实例的类索引。

displ((状态)辐角*1*) → 三维向量：相对于参考位置的(位置- 参考位置)

inertia(=*Vector3r::Zero()*)关联刚体在局部坐标系中的惯性。

isDamped(=*true*)
NewtonIntegrator 中的阻尼 可以通过将此变量设为 FALSE 来对单个颗粒停用。例如，重力作用下自由飞行的颗粒不适合使用阻尼，但同一模拟中的其他颗粒仍可能适用阻尼。

mass(=*0*)此刚体的质量

ori当前方向。

pos当前位置。**rank**(=*0*)链中的秩。**refOri**(=*Quaternionr::Identity()*)参考方向**refPos**(=*Vector3r::Zero()*)参考位置**rot**(（*State*）参数*1*) → Vector3：从参考方向（作为旋转向量）进行旋转**se3**(=*Se3r(Vector3r::Zero(), Quaternionr::Identity())*)作为一个对象的位置和方向。**updateAttrs**(（*Serializable*）参数*1*,（字典）参数*2*) → 无：根据给定字典更新对象属性**vel**(=*Vector3r::Zero()*)当前线速度。

class yade.wrapper.**CpmState**(继承 状态 → 可序列化对象)供*CPM* 模型 使用的刚体状态信息。

这些内容都不用于计算（至少目前不用），仅用于后处理。

angMom(=*Vector3r::Zero()*)当前
角动量

angVel(=*Vector3r::Zero()*)
当前角速度

blockedDOFs
线速度/角速度始终保持恒定（等于零或用户定义的值），不受施加的力/力矩影响的自由度。可包含 'xyzXYZ' 的字符串（平移和旋转）。

damageTensor(=*Matrix3r::Zero()*)通过微平面理论平均计算得到的损伤张量。
state.damageTensor.trace() =state.normDmg

densityScaling(=*-1*)*(自动更新)*参见
GlobalStiffnessTimeStepper::targetDt。

dict(*(Serializable)*辐角*1*) →字典:
返回属性字典。

dispHierarchy(*(State)*辐角 *1*[, *(bool)names=True*]) → 列表：返回分派类列表（从下到上），从类实例本身开始，顶层可索引对象位于最后。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

displ(*(State)*辐角 *1*) → 三维向量：相对于参考位置(位置-参考位置)

epsVolumetric(=*0*)此刚体周围的体积应变（目前未使用）

inertia(=*Vector3r::Zero()*)关联刚体在局部坐标系中的惯性。

isDamped(=是)NewtonIntegrator 中的阻尼可以通过将此变量设为 FALSE，为单个颗粒停用。例如，在重力作用下自由飞行的颗粒不适合施加阻尼，但在同一模拟中的其他颗粒仍可能适用阻尼。

mass(=*0*)此刚体的质量

normDmg(=*0*)平均损伤，包括已删除的接触（严格来说，这并非损伤，而是 1-relResidualStrength）

numBrokenCohesive(=*0*)已完全损伤的（黏聚）接触数量

numContacts(=*0*)与此刚体接触的数量

ori
当前方向。

pos
当前位置。

refOri(=*Quaternionr::Identity()*)
参考方向

refPos(=Vector3r::Zero())参考位置**rot**((State)辐角 *1*) → Vector3：由参考方向得到的旋转（以旋转向量表示）**se3**(=Se3r(Vector3r::Zero(), Quaternionr::Identity()))作为一个对象的位置和方向。**stress**(=Matrix3r::Zero())球形颗粒的应力张量（假设颗粒体积为 = pi*r*r*r*4/3.），堆积率为 0.62**updateAttrs**((Serializable)参数1，（字典）辐角 *2*) → None：根据给定字典更新对象属性**vel**(=Vector3r::Zero())当前线速度。**class** yade.wrapper.**JCFpmState**(继承 State → 可序列化对象)每个刚体的 JCFpm 状态信息。**angMom**(=Vector3r::Zero())当前角动量**angVel**(=Vector3r::Zero())当前角速度

blockedDOFs
线速度/角速度始终保持恒定（等于零或用户定义的值）的自由度，不受施加的力/力矩影响。可包含 'xyzXYZ' 的字符串（平移和旋转）。**damageIndex**(=*0*)断裂键与初始键的比率。 [-]
densityScaling(=*-1*)(自动更新)参见 *GlobalStiffnessTimeStepper::targetDt*。**dict**((可序列化对象)辐角 *1*) →字典：返回属性字典。

dispHierarchy((状态)辐角 *1*[, (布尔值)名称=真]) → 列表：返回分派类列表（从下至上），从类实例本身开始，最后是顶层可索引对象。如果 names 为真（默认值），则返回类名，而不是数值索引。

dispIndex
返回此实例的类索引。**displ**（状态）辐角 *1*) → 三维向量：相对于参考位置（位置- 参考位置）**inertia**(=*Vector3r::Zero()*)关联刚体在局部坐标系中的惯性。**isDamped**(=真)阻尼 *NewtonIntegrator* 可以通过将此变量设为 FALSE 来针对单个颗粒停用。例如，在重力作用下自由飞行的颗粒不适合使用阻尼，但同一模拟中的其他颗粒仍可能适用阻尼。**joint**(=*0*)表示该颗粒所属的节理面的数量（0 表示无节理，1 表示一个节理，等等）。 [-]**jointNormal1**(=*Vector3r::Zero()*)指定节理平面 1 的法向。注：理想情况下，这里应创建一个

其尺寸由节理整数定义的向量数组（节理法向量的数量与节理数量相同）。然而，由于节理检测通过 Python 脚本完成，因此需要让回推函数能够与 Python 配合使用。所以应调整 C++ 文件中的第 272 至 312 行。 [-]

`jointNormal2`(=Vector3r::Zero())指定节理平面2的法向。 [-] `jointNormal3`(=Vector3r::Zero())指定节理平面3的法向。 [-] `mass`(=*0*)该刚体的质量`nbBrokenBonds`(=*0*)断裂键的数量。 [-] `nbInitBonds`(=*0*)初始键的数量。 [-] `onJoint`(=false)标识颗粒是否位于节理面上。`ori`当前方向。`pos`当前位置。`refOri`(=Quaternionr::Identity())参考方向`refPos`(=Vector3r::Zero())参考位置`rot`((State)arg*1*) → Vector3：从参考方向开始的旋转（以旋转向量表示）`se3`(=Se3r(Vector3r::Zero(), Quaternionr::Identity()))将位置和方向表示为一个对象。`updateAttrs`((Serializable)arg1, (dict)arg*2*) → None：根据给定字典更新对象属性`vel`(=Vector3r::Zero())当前线速度。

`class yade.wrapper.WireState`(继承 状态 →可序列化对象)每个刚体的线体状态信息。这些信息均不用于计算（至少目前不会），仅用于后处理。

`angMom`(=*Vector3r::Zero()*)当前角动量`angVel`(=*Vector3r::Zero()*)当前角速度`blockedDOFs`线速度/角速度将始终保持恒定（等于零，或

用户定义的值），不受施加的力/力矩影响（平移和旋转）。`densityScaling`(=*-1*)（自动更新）参见 *GlobalStiffnessTimeStepper::targetDt*。`dict`（可序列化对象）辐角*1*) →字典：返回属性字典。

始终恒定（等于零，或为。可能包含“xyzXYZ”的字符串

dispHierarchy（状态）辐角*1*[，（布尔值）名称=真]）→ 列表：返回分派类的列表（从下至上），从类实例本身开始，顶层可索引对象位于最后。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。**displ**（状态）参数*1*）→
Vector3：从参考位置（位置- 参考位置）

inertia(=*Vector3r::Zero()*)关联刚体在局部坐标系中的惯性。

isDamped(=*true*)NewtonIntegrator 中的阻尼*NewtonIntegrator* 可通过设置此变量为
FALSE 来针对单个颗粒停用。例如，在重力作用下自由飞行的颗粒不适合使用阻尼，但它 t
可能仍适用于同一模拟中的其他颗粒。**mass**(=*0*)此刚体的质量**numBrokenLinks**(=*0*)断裂连接的数量（例如，与该刚体相连且已断裂的线缆数量）。 [-]**ori**当前方向。**pos**当前位置。**refOri**(=Quaternionr::Identity())参考方向**refPos**(=Vector3r::Zero())参考位置**rot**（状态）参数*1*）→ Vector3：从参考方向的旋转（以旋转向量表示）**se3**(=Se3r(Vector3r::Zero(), Quaternionr::Identity()))作为一个对象的位置和方向。**updateAttrs**（可序列化对象）参数1，（字典）参数*2*）→ 无：从给定字典更新对象属性**vel**(=Vector3r::Zero())当前线速度。

材料

class yade.wrapper.Material(继承可序列化对象)一个刚体的材料属性。**density**(=*1000*)材料密度 [$\mathrm{kg/m^3}$]**dict**(*(*可序列化对象*)*辐角*1*）→字典：返回属性字典。**dispHierarchy**(*(*材料*)*辐角*1*[, *(*布尔值*)*名称=真]）→ 列表：返回分派类列表（从下至上），从类实例本身开始，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。

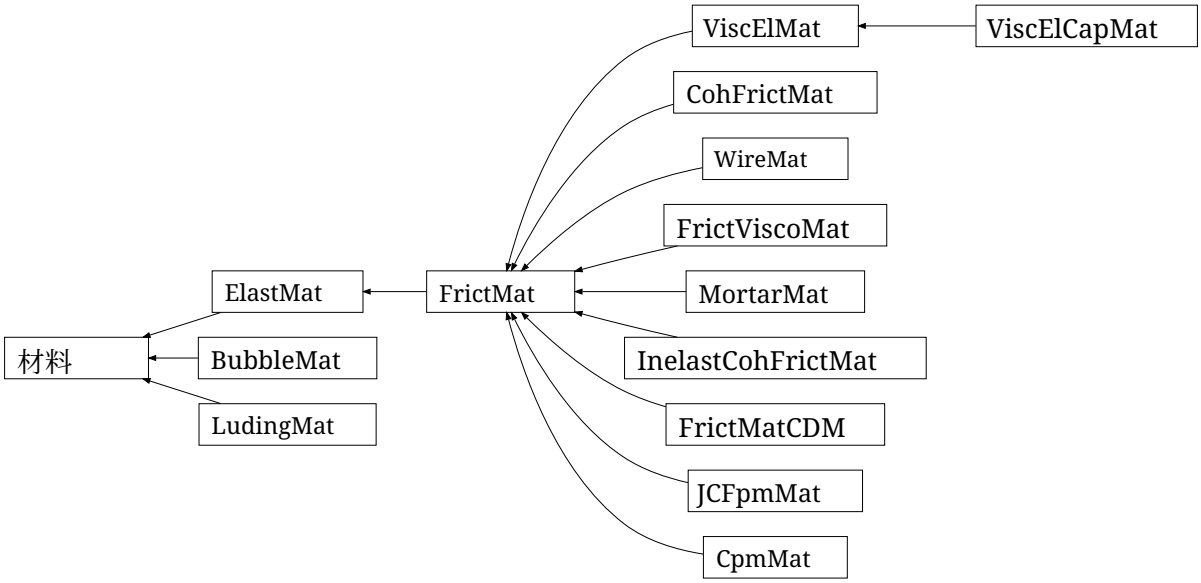


图 21：材料的继承图。另请参阅：*BubbleMat*、*CohFrictMat*、*CpmMat*、*ElastMat*、*Frict-Mat*、*FrictMatCDM*、*FrictViscoMat*、*InelastCohFrictMat*、*JCFpmMat*、*LudingMat*、*MortarMat*、*ViscEl-CapMat*，*ViscElMat*、*WireMat*。

dispIndex
返回此实例的类索引。

id(=*-1*，未共享)
此材料的数字 ID；仅当此材料为共享材料（即位于 O.materials 中）时，该值才为非负数，否则为 -1。通过 *O.materials.append* 将材料插入模拟时，会自动设置此值。（在使用 boost::serialization 之前，需要此 ID，因为当时共享指针无法正确跟踪；此 ID 未来可能会被移除）

label(=未初始化)
此材料的文本标识符；可用于在 *MaterialCon-tainer* 中查找共享材料。

newAssocState(*(材料)辐角 1*) →状态：返回新的 状态实例，该实例与此材料相关联。某些材料对 *Body::state* 类型有特殊要求；在创建刚体时调用此函数，可确保两者匹配。（如果使用 Python 中的 utils.sphere 等函数，此操作会自动完成。）**updateAttrs**(*(可序列化对象)参数1*，*(字典)辐角 2*) → 无：根据给定字典更新对象属性**class yade.wrapper.BubbleMat**(继承材料 →可序列化对象)用于气泡相互作用的材料，可与其他气泡类配合使用**density**(=*1000*)材料密度 [千克/米^3]**dict**(*(可序列化对象)辐角 1*) →字典：返回属性字典。**dispHierarchy**(*(材料)辐角 1*[, *(布尔值)名称=真*]) → 列表：返回分派类的列表（从下至上），以类实例本身开头，最后为顶层可索引对象。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

id(=*-1*，不共享)
此材料的数字 ID；仅当该材料为共享材料（即位于 O.materials 中）时才为非负数，否则为 -1。通过 *O.materials.append* 将材料插入模拟时，会自动设置此值。（在使用 boost::serialization 之前，此 ID 是必需的，因为当时共享指针无法正确跟踪；未来可能会移除此 ID）

label(=未初始化)
此材料的文本标识符；可用于在 *MaterialCon-tainer* 中进行共享材料查找。

newAssocState(*(材料)辐角 1*) →状态:
返回一个新的 状态 实例，它与此 材料 相关联。某些材料对 *Body::state* 类型有特殊要求；在创建刚体时调用此函数可确保两者匹配。（如果使用 Python 中的 utils.sphere 等函数，这将自动完成。）

surfaceTension(=*0.07197*)
气泡周围流体的表面张力。默认值为 25 摄氏度时水的表面张力。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角 2*) →
None：根据给定字典更新对象属性

class yade.wrapper.CohFrictMat(继承 *FrictMat* → *ElastMat* → 材料 → 可序列化对象)扩展 *FrictMat* 的材料描述，包含黏聚特性和转动刚度。可与 *定律2_ScGeom6D_CohFrictPhys_CohesionMoment* 。

alphaKr(=*2。0*)无量纲滚动刚度。

alphaKtw(=*2.0*)
无量纲扭转刚度。

density(=*1000*)
材料密度 [kg/m^3]

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

dispHierarchy(*(材料)辐角 1*[, *(布尔值)名称=真*]) → 列表：返回分派类列表（从下至上），从类实例本身开始，顶层可索引对象位于最后。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

etaRoll(=*-1.*)
无量纲滚动（又称“弯曲”）强度。如果为负，滚动力矩将是弹性的。

etaTwist(=*-1.*)
无量纲扭转强度。如果为负，扭矩将是弹性的。

fragile(=*true*)
接触强度超出时，黏聚力是否会消失

frictionAngle(=*.5*)
接触摩擦角（以弧度表示）。提示：在 Python 脚本中使用 ‘radians(degreesValue)’ .

id(=*-1*，不共享)
该材料的数字 ID；仅当此材料为共享材料（即位于 O.materials 中）时，ID 才为非负数，否则为 -1。通过 *O.materials.append* 将材料添加到模拟中时，会自动设置此值。（此前由于使用 boost::serialization 时未能正确跟踪共享指针，因此需要此 ID；将来它可能会被移除。）

isCohesive(=真)
此刚体是否可以形成可能的黏聚相互作用（如果为真，且取决于其他参数，例如 *Ip2_CohFrictMat_CohFrictMat_CohFrictPhys.setCohesionNow*）。

label(=未初始化)
此材料的文本标识符；可用于在 *MaterialContainer* 中进行共享材料查找。

momentRotationLaw(=假)
在接触处使用弯曲／扭转力矩。只有当两个刚体的此标志均为真时，接触才可能具有力矩。详见 定律2_ScGeom6D_CohFrictPhys_CohesionMoment.always_使用_- 力矩定律 。_

newAssocState((材料)辐角 *1*) →状态:
返回新的状态实例，该实例与此材料相关。有些材料对*Body::state* 类型有特殊要求，在创建刚体时调用此函数可确保两者匹配。（如果在 Python 中使用 utils.sphere 等函数，此操作会自动完成。）

normalCohesion(=*-1*)
抗拉强度，与压力具有相同量纲。如果为负值，则法向力为纯弹性。

poisson(= *25*)泊松比，或剪切刚度与法向刚度之比 [-]。其含义因 Ip 函子而异。

shearCohesion(=*-1*)
剪切强度，与压力同量纲。如果为负值，则剪切力完全为弹性力。

updateAttrs（可序列化对象）参数*1*，（字典）辐角 *2*) → 无：从给定字典更新对象属性

young(=*1e9*)弹性模量 [帕]。根据 Ip 函子的不同，其含义也不同。

class yade.wrapper.CpmMat(继承 *FrictMat* → *ElastMat* → 材料 → 可序列化对象)混凝土材料，用于其他 Cpm 类。

注意： 密度 初始化为 4800 kgm ^{3}自动设置为约 2800 kgm ^{3} 对应 0.5 密度堆积体。

混凝土颗粒模型（CPM）

CpmMat 是颗粒材料，*Ip2_CpmMat_CpmMat_CpmPhys* 对两个颗粒的材料取平均，创建 *CpmPhys*，随后在相互作用求解中由*Law2_ScGeom_CpmPhys_Cpm*中使用。*CpmState* 与 *CpmMat* 关联，并保存定义在颗粒而非相互作用上的状态（例如完全损伤的相互作用数量）。

该模型包含在外部定义的宏 CPM_MATERIAL_MODEL 中，具有拉伸损伤、剪切和压缩塑性以及率相关性。出于商业原因，当未定义 CPM MATERIAL MODEL 时使用的模型简化版本不包含率相关性和压缩塑性。完整模型将在我的（Václav Šmilauer）学位论文中详细介绍，并附有标定程序（刚度、泊松比、抗压强度与抗拉强度之比、断裂能、围压下的行为、率相关行为）。

即使是公开模型，也足以用于混凝土试样的模拟，例如单轴拉压试验。

damLaw(=*1*)
单轴拉伸下的损伤演化定律。0 表示线性应力-应变软化分支，（默认）表示指数损伤演化定律 1

density(=*1000*)材料密度 [千克/米^3]

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

dispHierarchy(*(*材料*)*辐角 *1*$\left[$, *(*布尔值*)*名称=*True*$\right]$) → 列表：返回分派类的列表（从下至上），从类实例本身开始，顶层可索引对象位于最后。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

dmgRateExp(=*0*)
法向黏度函数的指数。 [-]

dmgTau(=*-1*，若为负值则停用)法向黏度的特征时间。 [s]

epsCrackOnset(=非数)弹性应变上限 [-]

equivStrainShearContrib(=*0*)
等效应变的剪切贡献系数

frictionAngle(=*.5*)
接触摩擦角（单位为弧度）。提示：在 Python 脚本中使用 'radians(degreesValue)'。

id(=*-1*，未共享)此材料的数字 ID；仅当此材料被共享时（即位于 O.materials 中）才为非负值，否则为 -1。通过 *O.materials.append* 将材料插入模拟时，此值会自动设置。（在使用 boost::serialization 之前需要此 ID，因为当时共享指针无法正确跟踪；此 ID 将来可能会被移除）

isoPrestress(=*0*)
整个试样的各向同性预应力。 [帕]

label(=*uninitalized*)
此材料的文本标识符；可用于在 *MaterialCon-tainer* 中进行共享材料查找。

neverDamage(=否)
如果为 true，则不会发生损伤（仅用于测试）。

newAssocState（（材料）辐角 *1*）→状态：返回新的 状态 实例，该实例与此材料相关。某些材料对 *Body::state* 类型有特殊要求，在创建刚体时调用此函数可确保两者匹配。（如果使用 Python 中的 utils.sphere 等函数，则会自动完成此操作。）

plRateExp(=*0*)
黏塑性函数的指数。 [-]

plTau(=*-1*，若为负数则停用)黏塑性的特征时间。 [s]

poisson(=。*25*)泊松比，或剪切刚度与法向刚度之比 [-]。其含义取决于 Ip 函子。

relDuctility(=非数)
键在法向上的相对延性

sigmaT(=非数)初始黏聚力 [帕]

updateAttrs(*(可序列化对象)*参数*1*，*(字典)*辐角*2*) → 无：根据给定字典更新对象属性young(=*1e9*)弹性模量 [帕]。它在不同的 Ip 函子中含义不同。

class yade.wrapper.ElastMat(继承 材料 → 可序列化对象)

纯弹性材料。材料参数的具体含义可能取决于所使用的*IPhysFunctor*：true 表示在 *Ip2_FrictMat_FrictMat_MindlinPhys*中使用杨氏模量和泊松比，或在 *Ip2_FrictMat_FrictMat_FrictPhys*中使用接触刚度。

density(=*1000*)材料密度 [千克/米^3]

dict(*(可序列化对象)*辐角*1*) →字典:

返回属性字典。

dispHierarchy(*(材料)*辐角*1*[, *(布尔值)*名称=*True*]) → 列表：返回分派类列表（从下到上），以类实例本身开头，最后为顶层可索引对象。如果 names 为 true（默认值），则返回类名而不是数值索引。

dispIndex

返回此实例的类索引。

id(=*-1*，未共享)此材料的数字 ID；仅当此材料为共享材料（即位于 O.materials 中）时，该值才为非负数，否则为 -1。通过 *O.materials.append* 将材料插入模拟时，会自动设置此值。（此前由于尚未使用 boost::serialization，此 ID 是必要的，因为共享指针未能正确跟踪；未来可能会移除）

label(=未初始化)

此材料的文本标识符；可用于在 *MaterialCo* 中查找共享材料 *n-tainer*。

newAssocState(*(材料)*辐角*1*) →状态:

返回新的状态实例，该实例与此材料相关。某些材料对*Body::state* 类型有特殊要求，在创建刚体时调用此函数可确保两者匹配。（如果在 Python 中使用 utils.sphere 等函数，此操作会自动完成。）

poisson(=。*25*)泊松比，或剪切刚度与法向刚度之比 [-]。其含义因 Ip 函子而异。updateAttrs(*(可序列化对象)*参数*1*、*(字典)*辐角*2*) → None：根据给定字典更新对象属性young(=*1e9*)弹性模量 [帕]。其含义因 Ip 函子而异。

class yade.wrapper.FrictMat(继承自*ElastMat* →材料 →可序列化对象)

带接触摩擦的弹性材料。另请参见*ElastMat*。

density(=*1000*)材料密度 [千克/米^3]

dict(*(可序列化对象)*辐角*1*) →字典:

返回属性字典。

dispHierarchy(*(材料)*辐角*1*[, *(布尔值)*名称=*True*]) → 列表:

返回分派类列表（从下到上），从类实例本身开始，最后为顶层可索引对象。如果 names 为 true（默认值），则返回类名，而不是数值索引。

dispIndex
返回此实例的类索引。

frictionAngle(=*.5*)
接触摩擦角（以弧度为单位）。提示：在 Python 脚本中使用 'radians(degreesValue)'。

id(=*-1*，非共享)
此材料的数字 ID；仅当此材料为共享材料（即位于 O.materials 中）时，该值才为非负数，否则为 -1。通过 *O.materials.append* 将材料插入模拟时，会自动设置此值。（在使用 boost::serialization 之前，需要此 ID，因为当时无法正确跟踪共享指针；此 ID 将来可能会被移除）

label(=未初始化)
此材料的文本标识符；可用于 *MaterialCon-tainer* 中的共享材料查找。

newAssocState(*(Material)*辐角 *1*) →状态：
返回新的 状态 实例，该实例与此 材料相关联。某些材料对 *Body::state* 类型有特殊要求，在刚体创建时调用此函数可确保二者匹配。（如果在 Python 中使用 utils.sphere 等函数，此操作会自动完成。）

poisson(=。*25*)泊松比，或剪切刚度与法向刚度之比 [-]。其含义因 Ip 函子而异。

updateAttrs(*(Serializable)*参数*1*，*(dict)*辐角 *2*) → 无：
根据给定字典更新对象属性

young(=*1e9*)弹性模量 [帕]。其含义因 Ip 函子而异。

class yade.wrapper.FrictMatCDM(继承 *FrictMat* → *ElastMat* → 材料 → 可序列化对象)
用于扩展赫兹-明德林接触定律的材料。法向：锥形损伤模型的参数（Harkness 等，2016；Suhr 和 Six，2017）。切向：应力相关的颗粒间摩擦系数的参数（Suhr 和 Six，2016）。两个模型均可分别启用或关闭。

alpha(=*1e-6*)
[rad] 锥形凸起的角度，alpha 介于 (0, pi/2)

c1(=*0.0*)
[-] 压力相关摩擦模型参数 c1，若使用恒定颗粒间
摩擦系数

c2(=*0.0*)[-] 压力相关摩擦模型参数 c2，若使用恒定颗粒间摩擦系数，请设为 0

density(=*1000*)
材料密度 [kg/m^3]

dict(*(Serializable)arg1*) →字典：返
回属性字典。

dispHierarchy(*(Material)arg1*[, *(bool)names=True*]) → 列表：R
返回 l[列表 o] f d 分派 h cl 类 (from d向上 ds)，从类实例本身开始，最后一个顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

frictionAngle(=*.5*)
接触摩擦角（单位为弧度）。提示：在 Python 脚本中使用 'radians(degreesValue)'。

id(=*-1*，未共享)
此材料的数字 ID；仅当此材料为共享材料（即位于 O.materials 中）时，该值才为非负数，否则为 -1。通过 *O.materials.append* 将材料添加到模拟时，会自动设置此值。（由于在使用 boost::serialization 之前，共享指针无法得到正确跟踪，因此需要此 ID；未来可能会移除此 ID）

label(=未初始化)
此材料的文本标识符；可用于在 *MaterialCon-tainer* 中查找共享材料。

newAssocState(*(*材料 *)arg1*) →状态:
返回新的状态实例，该实例与此材料相关。某些材料对 *Body::state* 类型有特殊要求，在创建刚体时调用此函数可确保类型匹配。（如果通过 Python 使用 utils.sphere 等函数，则会自动完成此操作。）

poisson(=。*25*)
泊松比或剪切刚度与法向刚度的比值 [-]。其含义因 Ip 函子而异。

sigmaMax(=*1e99*)
>0 [帕] 材料的最大抗压强度，选择 1e99 可关闭锥形损伤模型

updateAttrs(*(*可序列化对象*)arg1*，*(*字典*)arg2*) → 无:
根据给定字典更新对象属性

young(=*1e9*)弹性模量 [帕]。其含义因 Ip 函子而异。

class yade.wrapper.**FrictViscoMat**(继承 *FrictMat* → *ElastMat* → 材料 → 可序列化对象)供 FrictViscoPM 类使用的材料

betan(=*0*。)
法向黏性阻尼系数中等于以下数值的比例 $\frac{c_n}{c_{n,crit}}$.

density(=*1000*)材料密度 [千克/米^3]

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

dispHierarchy(*(*材料*)*辐角 *1*[, *(*布尔值*)*名称=真]) → 列表：返回分派类列表（从下至上），以类实例本身开头，以顶层可索引对象结尾。如果名称为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

frictionAngle(=*.5*)
接触摩擦角（单位为弧度）。提示：在 Python 脚本中使用 'radians(degreesValue)'。

id(=*-1*,非共享)此材料的数字 ID；仅当此材料为共享材料（即位于 O.materials 中）时，该值才为非负数；否则为 -1。通过以下方式将材料插入模拟时，会自动设置此值：*O.materials.append*。（在使用 boost::serialization 之前，需要此 ID，因为当时共享指针无法正确跟踪；此功能将来可能会被移除。）

label(=未初始化)
此材料的文本标识符；可用于在 材料容- 器中进行共享材料查找。

newAssocState(*(*材料*)*辐角 *1*) →状态:
返回新的状态实例，它与此材料相关联。某些材料对 *Body::state* 类型有特殊要求，在创建刚体时调用此函数

将确保它们匹配。（如果使用 Python 中的 utils.sphere 等函数，则会自动完成。）

poisson(=。*25*)泊松比，或剪切刚度与法向刚度之比 [-]。其含义取决于 Ip 函子。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无：根据给定字典更新对象属性**young**(=*1e9*)弹性模量 [帕]。其含义取决于 Ip 函子。

class yade.wrapper.InelastCohFrictMat(继承 *FrictMat* → *ElastMat* → 材料 → 可序列化对象)

alphaKr(=*2*。*0*)用于滚动刚度的无量纲系数。

alphaKtw(=*2*。*0*)用于扭转刚度的无量纲系数。
compressionModulus(=*0*。*0*)压缩弹性模量

creepBending(=*0*。*0*)弯曲蠕变系数。通常取值介于 0 和 1 之间。

creepTension(=*0*。*0*)拉伸/压缩蠕变系数。通常取值介于 0 和 1 之间。

creepTwist(=*0*。*0*)扭转蠕变系数。通常取值介于 0 和 1 之间。

density(=*1000*)材料密度 [kg/m^3]

dict(*(可序列化对象)辐角1*) →字典：返回属性字典。**dispHierarchy**(*(材料)辐角1*[, *(布尔值)名称*=*真*]) → 列表：返回分派类列表（从下到上），以类实例本身开头，最后为顶层可索引对象。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。**epsilonMaxCompression**(=*0*。*0*)最大压缩塑性应变
epsilonMaxTension(=*0*。*0*)最大拉伸塑性应变**etaMaxBending**(=*0*。*0*)最大弯曲塑性应变
etaMaxTwist(=*0*。*0*)最大扭转塑性应变**frictionAngle**(=。*5*)接触摩擦角（单位为弧度）。
提示：在 Python 脚本中使用 'radians(degreesValue)'。

id(=*-1*，非共享)此材料的数字 ID；仅当该材料为共享材料（即位于 O.materials 中）时才为非负数，否则为 -1。通过以下方式将材料插入模拟时，会自动设置此值：*O.materials.append*。（在使用 boost::serialization 之前，必须有此 ID，因为当时共享指针无法正确跟踪；它将来可能会被移除）

label(=未初始化)
此材料的文本标识符；可用于在 *MaterialCon-tainer* 中进行共享材料查找。

newAssocState(*(材料)辐角 1*) →状态：返回新的状态实例，该实例与此材料相关。某些材料有

e

对*Body::state* 类型有特殊要求，且在创建刚体时调用此函数 d
将确保它们匹配。（如果使用 utils.sphere、.....函数，这会自动完成） ns
来自 Python）。nuBending(*=0。0*)弯曲弹性应力极限nuTwist(*=0。0*)扭转弹性应力极限
poisson(=。*25*)泊松比，或剪切刚度与法向刚度之比 [-]。它有不同的含义

s

具体含义取决于 Ip 函子。shearCohesion(*=0。0*)剪切弹性应力极限
shearModulus(*=0。0*)剪切弹性模量sigmaCompression(*=0。0*)压缩弹性应力极限
sigmaTension(*=0。0*)拉伸弹性应力极限tensionModulus(*=0。0*)拉伸弹性模量
unloadBending(*=0。0*)弯曲塑性卸载系数。通常取值介于 0 和 +无穷
unloadTension(*=0。0*)拉伸/压缩塑性卸载系数。通常取值介于 0 和 +无穷
unloadTwist(*=0。0*)扭转塑性卸载系数。通常取值介于 0 和 +无穷updateAttrs(*(可序列化对象)arg1*、*(字典)arg2*) → None：根据给定字典更新对象属性young(*=1e9*)弹性模量 [帕]。其具体含义取决于 Ip 函子。

class yade.wrapper.JCFpmMat(继承 *FrictMat* → *ElastMat* → 材料 → 可序列化对象)可能含节理的黏聚摩擦材料，供其他 JCFpm 类使用cohesion(*=0。*)定义剪切过程中最大允许切向力（Fn 对应的切向力）=0，位于矩阵中（*FsMa*

x

=黏聚力 * 横截面)。 [帕]density(*=1000*)材料密度 $[\mathrm{kg/m^3}]$dict(*(可序列化对象)辐角 1*) → 字典：返回属性字典。dispHierarchy(*(材料)辐角 1*[, *(布尔值)名称=真*]) → 列表：返回分派类列表（从下至上），从类实例本身开始

,

最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值 l
索引 ·

`dispIndex`
返回此实例的类索引。

`frictionAngle`(=*.5*)
接触摩擦角（弧度）。提示：在 Python 脚本中使用 'radians(degreesValue)'。`id`(=-*1*，未共享)
此材料的数字 ID；仅当此材料为共享材料（即位于 O.materials 中）时，该值才为非负数，否则为 -1。将材料通过 *O.materials.append* 插入模拟时，会自动设置此值。（在使用 boost::serialization 之前需要此 ID，因为当时无法正确跟踪共享指针；此 ID 将来可能会被移除。）

`jointCohesion`(=*0.*)定义节理面上的剪切最大允许切向力，适用于 Fn=0。 [帕]
`jointDilationAngle`(=*0*)定义节理面的剪胀（仅适用于光滑接触逻辑）。 [弧度]
`jointFrictionAngle`(=*-1*)定义节理面上的库仑摩擦。 [弧度]`jointNormalStiffness`(=*0.*)定义节理面上的法向刚度。 [帕/米]`jointShearStiffness`(=*0.*)定义节理面上的剪切刚度。 [帕/米]
`jointTensileStrength`(=*0.*)定义节理面上的拉伸最大允许法向力。 [帕]`label`(=未初始化)此材料的文本标识符；可用于在 MaterialContainer 中查找共享材料。`newAssocState`((材料)辐角 *1*) →状态：返回新的状态实例，该实例与此材料关联。某些材料对 Body::state 类型有特殊要求；在创建刚体时调用此函数可确保两者类型匹配。（如果在 Python 中使用 utils.sphere 等函数，此操作会自动完成。）`poisson`(=*.25*)泊松比，或剪切刚度与法向刚度之比 [-]。其含义取决于 Ip 函子。`residualFrictionAngle`(=-1.)定义残余摩擦角（接触无黏聚力时）。若未指定 residualFrictionAngle，则使用 frictionAngle。=摩擦角 [弧度]`tensileStrength`(=*0.*)定义矩阵中的拉伸最大允许法向力（FnMax = 抗拉强度 * 横截面）。 [帕]`type`(=*0*)如果两种不同类型的颗粒相互作用，则仅存在摩擦（无黏聚力）。[-]`updateAttrs`((可序列化对象)参数1，(字典)辐角 *2*) → 无：根据给定字典更新对象属性`young`(=*1e9*)弹性模量 [帕]。其含义取决于 Ip 函子。

`class yade.wrapper.LudingMat`(继承材料 →可序列化对象)简单 Luding模型接触材料 [Luding2008] ,[Singh2013]_ 。
`G0`(=非数)黏性阻尼

PhiF(=非数)无量纲塑性深度

density(=*1000*)材料密度 [千克/米^3]

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

dispHierarchy((材料)辐角 *1*[, (布尔值)名称=真]) → 列表：返回分派类列表（从下至上），从类实例本身开始，最后为顶层可索引对象。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

frictionAngle(=非数)
摩擦角 [弧度]

id(=*-1*，非共享)此材料的数字 ID；仅当此材料为共享材料（即位于 O.materials 中）时才为非负数，否则为 -1。通过 *O.materials.append* 将材料插入模拟时，会自动设置此值。（在使用 boost::serialization 之前，需要此 ID，因为当时共享指针无法被正确跟踪；此 ID 未来可能会被移除。）

k1(=非数)加载塑性分支的斜率

kc(=非数)不可逆拉伸黏结分支的斜率

kp(=非数)卸载与再加载极限弹性分支的斜率

ks(=非数)剪切刚度

label(=未初始化)此材料的文本标识符；可用于在 材料容-器 中进行共享材料查找。

newAssocState((材料)辐角 *1*) →状态：返回新的状态实例，该实例与此材料相关。某些材料对 *Body::state* 类型有特殊要求，在创建刚体时调用此函数可确保类型匹配。（如果通过 Python 使用 utils.sphere 等函数，此操作会自动完成。）

updateAttrs((可序列化对象)参数*1*，(字典)辐角 *2*) →
None：根据给定字典更新对象属性

class yade.wrapper.MortarMat(继承自 *FrictMat* → *ElastMat* → 材料 → 可序列化对象)砂浆界面材料，用于 Ip2_MortarMat_MortarMat_MortarPhys 和定律2_Sc-Geom_MortarPhys_Lourenco。默认值依据

cohesion(=*1e6*)黏
聚力 [帕]

compressiveStrength(=*10e6*)抗压
强度 [帕]density(=*1000*)材料密度 [
千克/米^3]dict((可序列化对象)辐角
1) →字典：返回属性字典。

dispHierarchy((材料)辐角 *1*[, (布尔值)名称=*True*]) → 列表：返回分派类列表（从下到上），以类实例本身开头，以顶层可索引对象结尾。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

ellAspect(=*3*)
椭圆形“帽”的长宽比。值 >1 表示椭圆沿法向应力轴方向较长。frictionAngle(=。*25*)摩擦角

id(=*-1*，未共享)此材料的数字 ID；仅当此材料为共享材料（即存在于 O.materials 中）时，该值才为非负数，否则为 -1。通过 *O.materials.append* 将材料插入模拟时，会自动设置此值。（在使用 boost::serialization 之前需要此 ID，因为当时共享指针无法被正确跟踪；此 ID 将来可能会被移除）

label(=未初始化)此材料的文本标识符；可用于在 MaterialContainer 中查找共享材料。neverDamage(=否)若为 true，则无论应力如何，相互作用都保持弹性newAssocState((材料)辐角 *1*) →状态：返回一个新的 State 实例，该实例与此材料关联。某些材料对 Body::state 的类型有特殊要求；在创建刚体时调用此函数可确保两者匹配。（如果使用 Python 中的 utils.sphere 等函数，则会自动完成此操作。）poisson(=*1*)剪切模量与法向模量之比tensileStrength(=*1*e*6*)抗拉强度 [帕]updateAttrs((可序列化对象)参数1，(字典)辐角 *2*) → 无：根据给定字典更新对象属性young(=*1*e*9*)法向弹性模量 [帕]

class yade.wrapper.ViscElCapMat(继承自 ViscElMat → FrictMat → ElastMat → 材料 →可序列化对象)用于带毛细参数的扩展接触黏弹性模型的材料。Capillar(=否)如果需要添加毛细力，则设为 True。CapillarType(=””)毛细相互作用的不同类型：Willett_ 数值型，Willett_ 解析型 [Willett2000] ,Weigert [Weigert1999] , Rabinovich [Rabinov2005] , Lambert（简化修正的 Rabinovich 模型） [Lambert2008]Vb(=*0*。*0*)液桥体积 [m^3]cn(=非数)法向黏性常数。注意：如果定义了 tc、en 或 es，则无法设置此参数！cs(=非数)剪切黏性常数。注意：如果定义了 tc、en 或 es，则无法设置此参数！

dcap(=*0*。*0*)毛细相的阻尼系数 [-]材料密度
density(=*1000*)kg/m [

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

dispHierarchy((材料)辐角 *1*[, (布尔值)名称=真]) → 列表：返回分派类的列表（从下至上），从类实例本身开始，顶层可索引对象位于最后。如果名称为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

en(=非数)法向的恢复系数

et(=非数)切向的恢复系数

frictionAngle(=。*5*)接触摩擦角（单位：弧度）。提示：在 Python 脚本中使用 ‘radians(degreesValue)’。**gamma**(=*0*。*0*)表面张力 [牛/米]

id(=*-1*，未共享)此材料的数字 ID；仅当此材料为共享材料（即位于 O.materials 中）时才为非负数，否则为 -1。通过 *O.materials.append* 将材料插入模拟时，此值会自动设置。（由于在使用 boost::serialization 之前，共享指针无法被正确追踪，因此需要此 ID；将来可能会移除。）

kn(=非数)法向弹性刚度。注意：如果定义了 tc、en 或 es，则不能设置此参数
!

ks(=非数)剪切弹性刚度。注意：如果定义了 tc、en 或 es，则不能设置此参数!

label(=未初始化)此材料的文本标识符；可用于在 *MaterialCon-tainer* 中查找共享材料。

lubrication(=*false*)当材料由杨氏模量、泊松比和恢复系数定义时，应用润滑力的选项。

mR(=*0*。*0*)滚动阻力，参见 [Zhou1999536]。

mRtype(=*1*)
滚动阻力类型，参见 [Zhou1999536]。mRtype=1 - 方程（3），见 [Zhou1999536]；m R-类型=2 - 方程（4），见 [Zhou1999536]。

newAssocState((材料)辐角 *1*) →状态:
返回新的 状态 实例，该实例与此材料相关。某些材料对 *Body::state* 类型有特殊要求，在创建刚体时调用此函数可确保二者匹配。（如果在 Python 中使用 utils.sphere 等函数，此操作会自动完成。）

poisson(=*.25*)
泊松比或剪切刚度与法向刚度之比 [-]。它有不同的含义 gs
取决于 Ip 函子。

roughnessScale(=*1e-3*)如果启用了润滑，则考虑颗粒的粗糙度尺度，以计算有效恢复系数。
tc(=非数)接触时间theta(=*0*。*0*)接触角 [°]updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → 无：根据给定的字典更新对象属性viscoDyn(=*1e-3*)如果启用了润滑，则考虑周围流体的动力黏度，根据碰撞的局部斯托克斯数计算有效恢复系数。

young(=*1e9*)弹性模量 [帕]。具体含义取决于 Ip 函子。

class yade.wrapper.ViscElMat(继承 *FrictMat* → *ElastMat* → 材料 → 可序列化对象)
根据一对球体接触问题的解析解得出的简单黏弹性模型的材料 [Pournin2001] 。

cn(=非数)法向黏性常数。注意，如果定义了 tc、en 或 es，则不能设置此参数！cs(=非数)剪切黏性常数。注意，如果定义了 tc、en 或 es，则不能设置此参数！density(=*1000*)材料密度 [kg/m^3]dict(*(*可序列化对象*)*辐角*1*) →字典：返回属性字典。

dispHierarchy(*(*材料*)*辐角*1*[, *(*布尔值*)*类名=真]) → 列表：返回分派类列表（从下到上），以类实例本身开头，以顶层可索引对象结尾。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。en(=非数)法向的
恢复系数

et(=非数)切向的恢复系数

frictionAngle(=*.5*)
接触摩擦角（单位为弧度）。提示：在 Python 脚本中使用 'radians(degreesValue)' .
id(=*-1*，未共享)此材料的数字 ID；仅当此材料已共享（即位于 O.materials 中）时才为非负数，否则为 -1。通过 *O.materials.append*将材料插入模拟时会自动设置此值。（此前使用 boost::serialization 时，由于共享指针未能正确跟踪，因此需要此 ID；今后可能会移除。）

kn(=非数)法向弹性刚度。注意，若已定义 tc、en 或 es，则不能设置此参数！

ks(=非数)剪切弹性刚度。注意，如果定义了 tc、en 或 es，则无法设置此参数！

label(=取消初始化)此材料的文本标识符；可用于共享材料查找，位于 *MaterialCon-tainer* 中。
lubrication(=否)

材料由杨氏模量、泊松比和 en（恢复系数）定义时应用润滑力的选项。

mR(=*0*。*0*)滚动阻力，参见 [Zhou1999536]。mRtype(=*1*)滚动阻力类型，参见 [Zhou1999536]。mRtype=1 — [Zhou1999536]中的方程（3）；mRtype=2 — [Zhou1999536]中的方程（4）。
newAssocState（（材料）辐角 *1*）→状态：

返回新的状态实例，该实例与此材料相关联。某些材料对 *Body::state* 类型有特殊要求，在创建刚体时调用此函数可确保两者匹配。（如果在 Python 中使用 utils.sphere 等函数，则会自动完成此操作。）

poisson(=。*25*)
泊松比，或剪切刚度与法向刚度之比 [-]。其含义取决于 Ip 函子。

roughnessScale(=*1e-3*)
如果启用了润滑，则会考虑颗粒的粗糙度尺度，以评估有效恢复系数。

tc(=非数)接触时间updateAttrs(*(*可序列化对象*)*
arg1, *(*字典*)arg2*）→ 无：根据给定字典更新对象属性

viscoDyn(=*1e-3*)如果启用了润滑，则会考虑周围流体的动力黏度，以将有效恢复系数作为碰撞局部斯托克斯数的函数进行评估。

young(=*1e9*)弹性模量 [帕]。其含义因 Ip 函子而异。

class yade.wrapper.WireMat(继承 *FrictMat* → *ElastMat* → 材料 → 可序列化对象)供 Wire 类使用的材料。与相应的函子结合使用时，可用于模拟钢丝网 [Thoeni2014]，土工织物 [Cheng2016] 等更多材料。as(=*0*。)用于将应力转换为力的单根导线横截面积。 [米^2]density(=*1000*)材料密度 [千克/米^3]diameter(=*0*。*0027*)

[米] 中单根导线的直径（该直径用于计算导线的横截面积）。

dict（（可序列化对象）辐角 *1*）→字典：返回属性字典。dispHierarchy（（材料）辐角 *1*[，（布尔值）名称=*True*]）→ 列表：返回分派类的列表（从下到上），从类实例本身开始，顶层可索引对象位于最后。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

frictionAngle(=*.5*)
接触摩擦角（单位为弧度）。提示：在 Python 脚本中使用 'radians(degreesValue)'。

id(=*-1*，非共享)
该材料的数字 ID；仅当此材料为共享材料（即位于 O.materials 中）时，该值才为非负数，否则为 -1。通过 *O.materials.append* 将材料插入模拟时，会自动设置此值。（在使用 boost::serialization 之前，此 ID 是必需的，因为当时无法正确跟踪共享指针；未来可能会移除）

isDoubleTwist(=否)
网格的类型。如果为 true，那么两个材料相同且刚体 ID 相差 1 的颗粒将被视为双扭结相互作用。

label(=未初始化)
此材料的文本标识符；可用于 *MaterialCon-tainer* 中的共享材料查找。

lambdaEps(=*0。47*)
介于 0 和 1 之间的参数，用于降低双扭结线材破坏时的应变（如 [Bertrand2008] 所用）。 [-]

lambdaF(=*1。0*)
由 [Thoeni2013] 引入的介于 0 和 1 之间的参数，用于定义平移后的力-位移曲线与新初始刚度的交点： $F^* = \lambda_F F_{elastic}$。 [-]

lambdak(=*0.73*)
用于计算双扭结线材弹性刚度的参数，取值范围为 0 到 1（Bertrand 使用）： [Bertrand2008]）：$k^D = 2(\lambda_k k_h + (1-\lambda_k)k^S)$。 [-]

lambdau(=*0.2*)
由 [Thoeni2013] 引入的参数，取值范围为 0 到 1，用于定义力-位移曲线的最大偏移，以考虑额外的初始伸长（例如线材扭曲/缺陷、滑移、系统柔度）： $\Delta l^* = \lambda_u l_0 rnd(seed)$。 [-]

newAssocState(*(材料)辐角 1*) →状态:
返回新的状态实例，该实例与此材料相关。某些材料对 *Body::state* 类型有特殊要求；在创建刚体时调用此函数可确保两者匹配。（如果使用 Python 中的 utils.sphere 等函数，则会自动完成此操作。）

poisson(=。*25*)泊松比，或剪切刚度与法向刚度之比 [-]。其含义因 Ip 函子而异。

seed(=*12345*)
用于初始化随机数生成器以计算变形的整数。如果该整数等于 0，则会根据时间计算一个内部随机种子。 [-]

strainStressValues(=未初始化)
通过一组点（应变[-]>0，应力[帕]>0）分段线性定义单根线框的应力-应变曲线。仅考虑拉伸，无需包含点 (0,0)！注意：向量需要先初始化！

strainStressValuesDT(=未初始化)
通过一组点（应变[-]>0，应力[帕]>0）分段线性定义双绞的应力-应变曲线。仅考虑拉伸，无需包含点 (0,0)！如果给定此值，计算将基于两条不同的应力-应变曲线进行，而不考虑 [Bertrand2008] 引入的参数（参见 [Thoeni2013]）。

type
三种不同类型：

0	对应 Bertrand 的方法（参见 [Bertrand2008]）：仅使用一条应力-应变曲线 被使用
1	新方法：可以使用两条独立的应力-应变曲线（参见 [Thoeni2013]）
2	采用随机畸变接触模型的新方法：两条独立的应力-应变 初始刚度和平移量已更改的曲线（若随机种子为 ≥ 0，平移量为随机值， 更多详情请参见 [Thoeni2013]）

默认类型为 0。**updateAttrs**（可序列化对象）参数*1*，（字典）辐角*2*）→否：根据给定字典更新对象属性**young**(=*1e9*)弹性模量 [帕]。其含义取决于 Ip 函子。

Bound

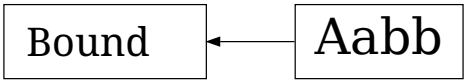


图 22：Bound 的继承图。另请参见：*Aabb*。

class yade.wrapper.Bound(继承自可序列化对象)关联刚体所占空间部分的包围区域；可能更大，用于优化碰撞检测**color**(=*Vector3r(1，1，1)*)用于渲染此对象的颜色**dict**(*(*可序列化对象*)*辐角*1*）→字典：返回属性字典。**dispHierarchy**(*(Bound)*辐角*1*[, *(*布尔值*)*名称=*True*]）→ 列表：返回分派类列表（从下至上），以类实例自身开头，最后为顶层可索引对象。如果 names 为 true（默认值），则返回类名，而非数值索引。

dispIndex

返回此实例的类索引。**lastUpdateIter**(=*0*)记录上次参考位置更新的迭代次数 *(*自动更新*)* **max**(=*Vector3r(*非数，非数，非数*)*)包含此边界（以及 刚体 ）的盒子的上角点**min**(=*Vector3r(*非数，非数，非数*)*)包含此边界（以及 刚体 ）的盒子的下角点**refPos**(=*Vector3r(*非数，非数，非数*)*)参考位置；每次边界分发器更新边界时，都会更新为当前刚体位置 *(*自动更新*)*

sweepLength(=*0*)

用于增大包围盒尺寸的长度，可根据之前的位移进行调整，条件是 *BoundDispatcher::targetInterv*>0。（自动更新）**updateAttrs**（可序列化对象）参数*1*，（字典）辐角*2*）→ None：根据给定字典更新对象属性

class yade.wrapper.Aabb(继承 *Bound* → 可序列化对象)
轴对齐包围盒，用于 *InsertionSortCollider*。（此类近乎冗余，因为 min、max 已包含在 *Bound* 本身中。不过，这种情况将来可能会改变。）

color(=*Vector3r(1, 1, 1)*)用于渲染
此对象的颜色

dict((可序列化对象)*arg1*) →字典:
返回属性字典。

dispHierarchy(*(Bound)arg1*[, *(bool)names=True*]) → 列表：返回分派类列表（从下至上），以类实例本身开头，以顶层可索引对象结尾。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。lastUpdateIter(=*0*)记录上次参考位置
更新的迭代 *(自动更新)*

max(=*Vector3r(*非数，非数，非数*)*)包含此边界（以及刚体）的盒子的上
角点

min(=*Vector3r(*非数，非数，非数*)*)包含此边界（以及刚体）的盒子的下
角点

refPos(=*Vector3r(*非数，非数，非数*)*)参考位置；每次边界分发器更新边界时，都会根据当前刚体位置更新（自动更新）

sweepLength(=*0*)用于增大包围盒尺寸的长度，可根据之前的位移进行调整，前提是 *BoundDispatcher::targetInterv*>0。（自动更新）

updateAttrs((可序列化对象)*参数1*，*(*字典*)*辐角*2*) →
None：根据给定字典更新对象属性

2.3.2 相互作用

相互作用

class yade.wrapper.Interaction(继承可序列化对象)两个
刚体之间的相互作用。

cellDist
如果使用周期性边界条件，刚体之间的距离以单元格尺寸为单位；id2 相对于其 *State::pos* 坐标平移了这么多个单元格后，此相互作用才会存在。由碰撞器赋值。

> **警告：** （内部）单元格距离必须在 Interaction::reset() 后保留，它只在构造函数中初始化。被本构定律取消、经过 reset() 重置后仅成为势的相互作用，如果几何函子再次使其成为实数，就必须具有周期信息。经过几天调试后才弄明白这一点，真是值得记住 :-)

dict((可序列化对象)辐角*1*) →字典:
返回属性字典。

geom(=未初始化)
相互作用的几何体部分。

id1(=0)标识符 此相互作用中第一个刚体的标识符。

id2(=0)标识符 此相互作用中第二个刚体的标识符。

isActive
: 如果此相互作用处于活动状态，则为 True。否则，此相互作用产生的力不会被计入。默认值为 True。

isReal
: 若此相互作用同时具有 几何 和 物理，则为 True；否则为 False。

iterBorn(=-1)
: 相互作用被添加到模拟中的步数。

iterMadeReal(=-1)
: 相互作用完全创建（即几何和物理均已创建）时的步数。（应仅由 *IPhysDispatcher* 和 *InteractionLoop*修改，因此它们被设为 Interaction 的友元

phys(=未初始化)
: 相互作用的物理（材料）部分。

updateAttrs（可序列化对象）参数*1*，（字典）辐角*2*) → 无:
: 根据给定字典更新对象属性

IGeom

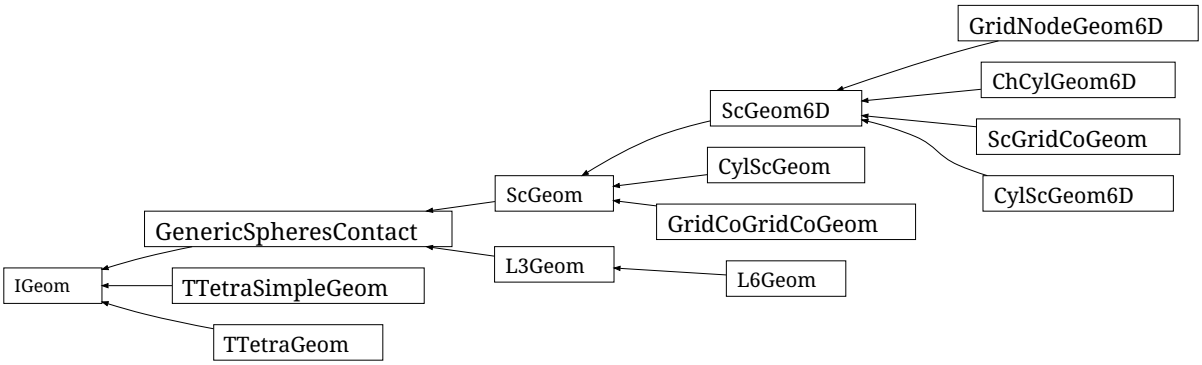


图 23：IGeom 的继承图。另请参阅：*ChCylGeom6D*、*CylScGeom*、*CylScGeom6D*、*Gener-icSpheresContact*、*GridCoGridCoGeom*、*GridNodeGeom6D*、*L3Geom*、*L6Geom*、*ScGeom*、*ScGeom6D*、*ScGridCoGeom*、*TTetraGeom*、*TTetraSimpleGeom*。

class yade.wrapper.IGeom(继承可序列化对象)相互作用的几何构型dict（（可序列化对象）辐角*1*）→ 字典：返回属性字典。dispHierarchy（（*IGeom*）辐角*1*[，（布尔值）名称=真]）→ 列表：返回分派类列表（从下至上），以类实例本身开头，顶层可索引对象位于最后。若 names 为真（默认值），则返回类名而非数值索引。

dispIndex
: 返回此实例的类索引。updateAttrs((可序列化对象)参数*1*，(字典)辐角*2*) → 无：根据给定字典更新对象属性

class yade.wrapper.ChCylGeom6D(继承 *ScGeom6D* → *ScGeom* → *GenericSpheresContact* → *IGeom* → 可序列化对象)testbending(=*Vector3r::Zero()*)接触处的弯曲，以向量表示旋转轴和角度（角度=范数）。contactPoint(=未初始化)相互作用的某个参考点（通常位于中间）。*(*自动计算*)*dict(*(*可序列化对象*)*辐角 *1*) →字典：返回属性字典。dispHierarchy(*(IGeom)*参数 *1*[, *(*布尔值*)*名称=真]) → 列表：返回分派类列表（从下至上），以类实例本身开头，顶层可索引对象位于最后。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex

返回此实例的类索引。incidentVel((ScGeom)arg1, (Interaction)i[, (bool)避免颗粒棘轮效应=True]) → Vec- tor3 :返回相互作用的入射速度（另请参阅 Ig*2*_Sphere_Sphere_ScGeom.避免颗粒棘轮效应，了解棘轮参数的说明）。initialOrientation1(=Quaternionr(1.0, 0.0, 0.0, 0.0))刚体 1 在初始化时的方向（自动更新）initialOrientation2(=Quaternionr(1.0, 0.0, 0.0, 0.0))刚体 2 在初始化时的方向（自动更新）normal(=未初始化)沿相互作用方向，从粒子 #1 指向粒子 #2 的单位向量。（自动更新）penetrationDepth(=非数)球体的穿透距离（重叠时为正）refR1(=未初始化)粒子 #1 的参考半径。（自动计算）refR2(=未初始化)粒子 #2 的参考半径。（自动计算）relAngVel((ScGeom)arg1, (Interaction)i) → Vector3 :返回相互作用的相对角速度。shearInc(=Vector3r::Zero())上一步的剪切位移增量twist(=*0*)接触的弹性扭转角（绕法向轴）。twistCreep(=Quaternionr(1.0, 0.0, 0.0, 0.0))存储的蠕变，在计算弹性力矩时从总相对转动中减去（自动更新）updateAttrs((Serializable)arg1, (dict)arg*2*) → None :根据给定字典更新对象属性

class yade.wrapper.CylScGeom(继承 *ScGeom* → *GenericSpheresContact* → *IGeom* → 可序列化对象)圆柱-球体接触的几何结构。contactPoint(=未初始化)相互作用的某个参考点（通常位于中间）。（自动计算）

dict（（可序列化对象）辐角*1*）→字典：返回属性字典。dispHierarchy（（*IGeom*）辐角*1*[，（布尔值）名称=*True*]）→ 列表：返回分派类列表（从下至上），以类实例本身开头，最后为顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

end(=*Vector3r::Zero()*)第 2 个节点的位置（自动更新）

id3(=*0*)下一个链式圆柱体的标识符 *(自动更新)*

incidentVel(*(ScGeom)*参数*1*，*(Interaction)i*[，*(bool)*避免颗粒棘轮效应=真]）→ Vec- tor3：返回相互作用的入射速度（另请参阅*Ig2_球体_球体_Sc-Geom.avoidGranularRatcheting* 以了解棘轮效应参数的说明）。

isDuplicate(=*0*)如果接触由两个链式圆柱体共享，此标志会自动设为 true (1)。本构定律会跳过一次重复的相互作用，因此同一时间只有一个接触有效。如果 isDuplicate=2，则表示两个重复项中的一个已不再具有几何相互作用，应由本构定律将其删除。normal(=未初始化)沿相互作用方向、从粒子 #1 指向粒子 #2 的单位向量。（自动更新）onNode(=假)节点上的接触? penetrationDepth(=非数)球体的穿透距离（重叠时为正）refR1(=未初始化)粒子 #1 的参考半径。（自动计算）

refR2(=未初始化)粒子 #2 的参考半径。（自动计算）

relAngVel(*(ScGeom)*参数*1*，*(Interaction)i*）→ Vector3：返回相互作用的相对角速度。relPos(=*0*)圆柱体上接触点的位置（0：节点-，1：节点+）（自动更新）shearInc(=*Vector3r::Zero()*)上一步的剪切位移增量start(=*Vector3r::Zero()*)第一个节点的位置（自动更新）trueInt(=*-1*)定义接触实际发生时圆柱体的刚体 ID，适用于 *CylScGeom::isDuplicate*>0。updateAttrs(*(Serializable)*参数*1*，*(dict)*辐角*2*）→ None：根据给定字典更新对象属性

class yade.wrapper.CylScGeom6D(继承 *ScGeom6D* → *ScGeom* → *GenericSpheresContact* → *IGeom* → 可序列化对象)表示 几何体 的 刚体 之间接触的类。接触具有 6 个自由度（法向量、 2× 剪切、扭转、2 个弯曲自由度），并使用 *ScGeom* 增量算法更新剪切。

bending(=*Vector3r::Zero()*)接触处的弯曲，用一个定义旋转轴和角度的向量表示（角度=范数）。**contactPoint**(=取消初始化)相互作用的某个参考点（通常位于中间）。*(自动计算)*

dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

dispHierarchy(（*IGeom*）辐角*1*[,（布尔值）名称=真]) → 列表：返回分派类列表（从下向上），以类实例本身开头，顶层可索引对象位于末尾。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

end(=*Vector3r::Zero()*)第二个节点的位置（自动更新）

id3(=*0*)下一个链式圆柱体的标识符 （自动更新）

incidentVel(*(ScGeom)*参数*1*，*(Interaction)i*[,*(bool)*避免颗粒棘轮效应=是]) → Vec- tor3：返回该相互作用的入射速度（另请参阅*Ig2_球体_球体_Sc Geom.*避免颗粒棘轮效应，了解棘轮参数的说明。

initialOrientation1(=*Quaternionr(1.0, 0.0, 0.0, 0.0)*)刚体 1 在初始化时刻的方向（自动更新）

initialOrientation2(=*Quaternionr(1.0, 0.0, 0.0, 0.0)*)刚体 2 在初始化时刻的方向（自动更新）

isDuplicate(=*0*)如果两个链式圆柱体共享接触，此标志会自动设为 true (1)。本构定律会跳过一次重复的相互作用，因此每次只有一个接触有效。如果 isDuplicate=2，则表示两个重复项之一已不再具有几何相互作用，应由本构定律将其删除。

normal(=取消初始化)沿相互作用方向，从粒子 #1 指向粒子 #2 的单位向量。*(自动更新)*

onNode(=否)接触是否位于节点上？ **penetrationDepth**(=非数)球体的穿透距离（重叠时为正）**refR1**(=未初始化)粒子 #1 的参考半径。*(自动计算)* **refR2**(=未初始化)粒子 #2 的参考半径。*(自动计算)***relAngVel**(*(ScGeom)*参数*1*，*(Interaction)i*) → 三维向量：返回相互作用的相对角速度。**relPos**(=*0*)圆柱体上接触点的位置（0：节点-，1：节点+）*(自动更新)***shearInc**(=*Vector3r::Zero()*)上一步的剪切位移增量**start**(=*Vector3r::Zero()*)第一个节点的位置*(自动更新)*

trueInt(=*-1*)定义发生实际接触时圆柱体的刚体 ID，其中 CylScGeom::isDuplicate>0。**twist**(=*0*)接触的弹性扭转角（绕法向轴）。**twistCreep**(=Quaternionr(1.0, 0.0, 0.0, 0.0))存储的蠕变量，在计算弹性力矩时从总相对转动中减去（自动更新）**updateAttrs**((Serializable)参数1，(dict)辐角*2*) → 无：从给定字典更新对象属性**class** yade.wrapper.**GenericSpheresContact**(继承 I Geom → 可序列化对象)将 ScGeom 和 L3Geom 合并的类，用于 GlobalStiffnessTimeStepper。（该类将来可能会被移除。）请勿直接使用此类。**contactPoint**(=未初始化)相互作用的某个参考点（通常位于中间）。（自动计算）**dict**((Serializable)辐角*1*) →字典：返回属性字典。**dispHierarchy**((IGeom)辐角*1*[, (bool)名称=真]) → 列表：返回分派类的列表（从下到上），从类实例本身开始，最后为顶层可索引对象。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex

返回此实例的类索引。**normal**(=未初始化)沿相互作用方向、从粒子 #1 指向粒子 #2 的单位向量。（自动更新）**refR1**(=未初始化)粒子 #1 的参考半径。（自动计算）**refR2**(=未初始化)粒子 #2 的参考半径。（自动计算）**updateAttrs**((可序列化对象)参数1, (字典)辐角*2*) → 无 : 从给定字典更新对象属性**class** yade.wrapper.**GridCoGridCoGeom**(继承 ScGeom → GenericSpheresContact → IGeom→ 可序列化对象)网格连接-网格连接接触的几何体。**contactPoint**(=未初始化)相互作用的某个参考点（通常位于中间）。（自动计算）**dict**((可序列化对象)辐角*1*) →字典：返回属性字典。**dispHierarchy**((IGeom)辐角*1*[, (布尔值)名称=真]) → 列表：返回分派类列表（从下至上），从类实例本身开始，最后是顶层可索引对象。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex

返回此实例的类索引。**incidentVel**(***(ScGeom)***参数***1***, *(Interaction)i*[, ***(bool)***避免颗粒棘轮效应=是]) → Vec- tor3：返回相互作用的入射速度（另请参阅*Ig2_球体_球体_Sc-Geom.avoidGranularRatcheting* 以了解棘轮效应参数的说明）。

normal(=未初始化)沿相互作用方向、从粒子 #1 指向粒子 #2 的单位向量。（自动更新）**penetrationDepth**(=非数)球体的穿透距离（重叠时为正）**refR1**(=未初始化)粒子 #1 的参考半径。（自动计算）**refR2**(=未初始化)粒子 #2 的参考半径。（自动计算）**relAngVel**((ScGeom)参数1, (Interaction)i）→ Vector3：返回相互作用的相对角速度。**relPos1**(=*0*)第一个连接处接触点的位置（0：节点-，1：节点+）（自动更新）**relPos2**(=*0*)第一个连接处接触点的位置（0：节点-，1：节点+）（自动更新）**shearInc**(=Vector3r::Zero())上一步中的剪切位移增量**updateAttrs**((Serializable)参数1，(dict)参数*2*）→ None：根据给定的字典更新对象属性**class yade.wrapper.GridNodeGeom6D**(继承 ScGeom6D → ScGeom → 通用球体接触 → IGeom → 可序列化对象)GridNode-GridNode 接触的几何体。几乎所有内容均继承自 ScGeom6D。**bending**(=Vector3r::Zero())接触处的弯曲，以定义旋转轴和角度的向量表示（角度=范数）。**connectionBody**(=未初始化)对链接两个网格节点的 GridNode 刚体的引用。**contactPoint**(=未初始化)相互作用的某个参考点（通常位于中间）。（自动计算）**dict**((Serializable)参数*1*）→字典：返回属性字典。**dispHierarchy**((IGeom)参数*1*[, (bool)名称=是]）→ 列表：返回分派类列表（从下至上），从类实例本身开始，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex

返回此实例的类索引。**incidentVel**((ScGeom)arg1, (Interaction)i[, (bool)avoidGranularRatcheting=True]）→ Vec- tor3： 返回相互作用的入射速度（另请参阅 Ig*2*_Sphere_Sphere_Sc- Geom.avoidGranularRatcheting，了解棘轮效应参数的说明）。**initialOrientation1**(=Quaternionr(1.0, 0.0, 0.0, 0.0))刚体 1 在初始化时的方向（自动更新）**initialOrientation2**(=Quaternionr(1.0, 0.0, 0.0, 0.0))刚体 2 在初始化时的方向（自动更新）**normal**(=未初始化)沿相互作用方向的单位向量，从粒子 #1 指向粒子 #2。（自动更新）

penetrationDepth(=非数)球体的穿透距离（重叠时为正）refR1(=未初始化)粒子 #1 的参考半径。（自动计算）refR2(=未初始化)粒子 #2 的参考半径。（自动计算）relAngVel((ScGeom)arg1, (Interaction)i) → Vector3：返回相互作用的相对角速度。shearInc(=Vector3r::Zero())上一步的剪切位移增量twist(=*0*)接触法向轴周围的弹性扭转角。twistCreep(=Quaternionr(1.0, 0.0, 0.0, 0.0))存储的蠕变；计算弹性力矩时，会从总相对转动中减去该值（自动更新）updateAttrs((Serializable)arg1, (dict)arg*2*) → None：根据给定字典更新对象属性class yade.wrapper.L3Geom(继承 GenericSpheresContact → IGeom → Serializable)以局部坐标表示的接触几何，具有 3 个自由度：法向一个自由度，剪切平面内两个自由度。 [实验性]F(=Vector3r::Zero())局部坐标系中的施加力 [仅用于调试，将会移除]contactPoint(=未初始化)相互作用的某个参考点（通常位于中间）。（自动计算）dict((Serializable)arg*1*) →dict：返回属性字典。dispHierarchy((IGeom)arg*1*[, (bool)names=True]) → list：返回分派类列表（从下到上），以类实例自身开头，最后为顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。dispIndex返回此实例的类索引。normal(=未初始化)沿相互作用方向的单位向量，从粒子 #1 指向粒子 #2。（自动更新）refR1(=未初始化)粒子 #1 的参考半径。（自动计算）refR2(=未初始化)粒子 #2 的参考半径。（自动计算）trsf(=Matrix3r::Identity())从全局坐标到局部坐标的变换（旋转）。（平移部分位于 GenericSpheresContact.contactPoint 中）u(=Vector3r::Zero())局部坐标系中的位移分量。（自动更新）u0零位移值；计算得到的几何位移 u 应始终减去 u0，从而得到 u。此值可以更改，例如

1. 通过*IGeomFunctor*（例如，用于考虑基于四元数的底层几何算法无法表示的大剪切位移值）；2. 通过*LawFunctor*，用于考虑法向平衡位置不等于零几何重叠的情况（在接触创建后立即设置一次）；3. 通过*LawFunctor*，用于考虑塑性滑移。

注意：切勿设置*u0*的绝对值，只设置增量，因为*IGeomFunctor* 和*LawFunctor* 都会使用它。如果需要跟踪塑性变形，请将其存储在*IPhys* 中（而非此处；这可能会改变：让*u0* 专供*LawFunctor* 使用，并在需要时将单独的值存储在派生自*L3Geom*的类中。

---

updateAttrs（可序列化对象）参数*1*，（字典）辐角*2*）→ 无：根据给定字典更新对象属性

class yade.wrapper.L6Geom(继承自*L3Geom* → *GenericSpheresContact* → *IGeom* → 可序列化对象)局部坐标系中的接触几何，具有 6 个自由度。 [实验性]

F(=*Vector3r::Zero()*)局部坐标中的施加力 [仅用于调试，将被移除]

contactPoint(=未初始化)相互作用的某个参考点（通常位于中间）。（自动计算）

dict（可序列化对象）辐角*1*）→字典：返回属性字典。

dispHierarchy（*IGeom*）辐角*1*[，（布尔值）名称=真]）→ 列表：返回分派类列表（从下至上），从类实例本身开始，最后是顶层可索引对象。如果名称为真（默认值），则返回类名而非数值索引。

dispIndex

返回此实例的类索引。**normal**(=未初始化)沿相互作用方向，从粒子 #1 指向粒子 #2 的单位向量。（自动更新）**phi**(=*Vector3r::Zero()*)局部坐标中的旋转分量。（自动更新）**phi0**(=*Vector3r::Zero()*)零旋转，应始终从*phi* 中减去以得到其值。参见*L3Geom.u0*。

refR1(=未初始化)粒子 #1 的参考半径。（自动计算）

refR2(=未初始化)粒子 #2 的参考半径。（自动计算）

trsf(=*Matrix3r::Identity()*)从全局坐标到局部坐标的变换（旋转）。（平移部分位于*Gener-icSpheresContact.contactPoint*中）

u(=*Vector3r::Zero()*)局部坐标中的位移分量。（自动更新）

u0零位移值；始终应从几何位移*u* 中减去 u0，该位移由相应的*IGeomFunctor*计算得出，结果为*u*。例如，可以更改此值

1. 通过*IGeomFunctor*，例如，考虑无法用基于四元数的底层几何算法表示的大剪切位移值）

2. 由*LawFunctor*处理，以考虑法向平衡位置与零几何重叠不同的情况（设置一次，在创建相互作用后立即设置）；3. 由*LawFunctor* 处理塑性滑移。

注意： 切勿设置 *u0* 的绝对值，只设置增量，因为 *IGeomFunctor* 和*LawFunctor* 都会使用它。如果需要跟踪塑性变形，请将其存储在 *IPhys* 中（这可能会更改：让 *u0* 专供*LawFunctor* 使用，并在派生自 *L3Geom* 的类中单独存储一个值（需要时））。

`updateAttrs`(*(可序列化对象)参数1*，*(字典)辐角2*) → 无：
根据给定字典更新对象属性

`class yade.wrapper.ScGeom`(继承 *GenericSpheresContact* → *IGeom* → 可序列化对象)

表示两个刚体之间接触点的几何体的类。尽管它主要针对球体接触相互作用（因此命名为“Sc”），但其适用范围比球体-球体接触更广；它也用于表示球体与非球形刚体（面片、平面、盒子、链式圆柱体）之间，或两个非球形刚体（链式圆柱体）之间的接触。该接触有 3 个自由度（法向量和 2×剪切），并使用增量算法更新剪切。

我们分别用 $\mathbf{x}$、$\mathbf{v}$、$\boldsymbol{\omega}$ 表示位置、线速度和角速度（均以全局坐标表示），用 r 表示颗粒半径；下标 1 或 2 用于区分接触的两个球体。然后定义分支长度和单位接触法向量

$$\mathrm{l} = \|\mathbf{x}_2 - \mathbf{x}_1\|, \mathbf{n} = \frac{\mathbf{x}_2 - \mathbf{x}_1}{\|\mathbf{x}_2 - \mathbf{x}_1\|}$$

球体的相对速度为

$$\boldsymbol{\nu}_{12} = \frac{\mathrm{r}_1 + \mathrm{r}_2}{\mathrm{l}}(\boldsymbol{\nu}_2 - \boldsymbol{\nu}_1) - (\mathrm{r}_2\boldsymbol{\omega}_2 + \mathrm{r}_1\boldsymbol{\omega}_1) \times \mathbf{n}$$

乘以平动速度的分数项用于使定义具有客观性并避免棘轮效应（参见 *Ig2_球体_球体_ScGeom.avoidGranularRatcheting*）。剪切分量为

$$\boldsymbol{\nu}_{12}^{\mathrm{s}} = \boldsymbol{\nu}_{12} - (\mathbf{n} \cdot \boldsymbol{\nu}_{12})\mathbf{n}.$$

上一步的切向位移增量为

$$\Delta\boldsymbol{x}_{12}^{\mathrm{s}} = \Delta\mathrm{t}\boldsymbol{\nu}_{12}^{\mathrm{s}}.$$

`contactPoint`(=未初始化)
相互作用的某个参考点（通常位于中间）。（自动计算）

`dict`(*(可序列化对象)辐角1*) →字典：
返回属性字典。

`dispHierarchy`(*(IGeom)辐角1*[, *(布尔值)名称=True*]) → 列表：
返回分派类列表（从下到上），从类实例本身开始，顶层可索引对象位于最后。如果 names 为 true（默认值），则返回类名而不是数值索引。

`dispIndex`
返回此实例的类索引。

`incidentVel`(*(ScGeom)参数1*，*(Interaction)i*[, *(bool)avoidGranularRatcheting=True*]) → 三维向量：返回相互作用的入射速度（另请参阅*Ig2_Sphere_Sphere_Sc-Geom.avoidGranularRatcheting* 以了解棘轮效应参数的说明）。

normal(=未初始化)沿相互作用方向、从粒子 #1 指向粒子 #2 的单位向量。（自动更新）**penetrationDepth**(=非数)球体的穿透距离（重叠时为正值）**refR1**(=未初始化)粒子 #1 的参考半径。（自动计算）**refR2**(=未初始化)粒子 #2 的参考半径。（自动计算）**relAngVel**((ScGeom)参数1, (Interaction)i）→ 三维向量：返回相互作用的相对角速度。**shearInc**(=Vector3r::Zero())上一步的剪切位移增量**updateAttrs**((Serializable)参数1, (dict)参数*2*）→ None：根据给定字典更新对象属性

class yade.wrapper.**ScGeom6D**(继承 ScGeom → GenericSpheresContact → IGeom → 可序列化对象)表示两个接触刚体几何形状的类。该接触有 6 个自由度（法向、 2×剪切、扭转、2 个弯曲），并使用 ScGeom 增量算法更新剪切。**bending**(=Vector3r::Zero())接触处的弯曲，以定义旋转轴和角度的向量表示（角度=范数）。**contactPoint**(=未初始化)相互作用的某个参考点（通常位于中间）。（自动计算）**dict**((Serializable)参数*1*）→字典：返回属性字典。**dispHierarchy**((IGeom)参数*1*[, (bool)names=True]）→ 列表：返回分派类列表（从下至上），以类实例本身开头，以顶层可索引对象结尾。如果 names 为 true（默认值），则返回类名，而不是数值索引。

dispIndex

返回此实例的类索引。**incidentVel**((ScGeom)参数1, (Interaction)i[, (bool)避免颗粒棘轮效应=True]）→ Vec- tor3：返回相互作用的入射速度（关于棘轮效应参数的说明，另请参阅 Ig*2*_Sphere_Sphere_Sc- Geom.avoidGranularRatcheting）。**initialOrientation1**(=Quaternionr(1.0, 0.0, 0.0, 0.0))刚体 1 在初始化时的方向（自动更新）**initialOrientation2**(=Quaternionr(1.0, 0.0, 0.0, 0.0))刚体 2 在初始化时的方向（自动更新）**normal**(=未初始化)沿相互作用方向从粒子 #1 指向粒子 #2 的单位向量。（自动更新）**penetrationDepth**(=非数)球体的穿透距离（重叠时为正值）**refR1**(=未初始化)粒子 #1 的参考半径。（自动计算）**refR2**(=未初始化)粒子 #2 的参考半径。（自动计算）

relAngVel(*ScGeom*参数*1*，*(Interaction)i*) → Vector3：返回相互作用的相对角速度。
shearInc(=*Vector3r::Zero()*)上一步的剪切位移增量twist(=*0*)弹性扭转角（绕法向轴）的接触。
twistCreep(=*Quaternionr(1.0，0.0，0.0，0.0)*)存储的蠕变，在计算弹性力矩时从总相对转动中减去（自动更新）updateAttrs(*Serializable*参数*1*，*(dict)*参数*2*) → 无：根据给定字典更新对象属性

class yade.wrapper.ScGridCoGeom(继承 *ScGeom6D* → *ScGeom* → *GenericSpheresContact* → *IGeom* → 可序列化对象)几何体： 网格连接-球体接触。bending(=*Vector3r::Zero()*)接触处的弯曲，以定义旋转轴和角度的向量表示（角度范数）。contactPoint(=未初始化)相互作用的某个参考点（通常位于中间）。（自动计算）dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

dispHierarchy(（*IGeom*）辐角*1*[，（布尔值）名称=真]) → 列表：返回分派类列表（从下到上），从类实例本身开始，最后是顶层可索引对象。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

id3(=*0*)第一个 网格节点的标识符。 （自动更新）

id4(=*0*)第二个 网格节点的标识符。 （自动更新）

id5(=*-1*)第三个网格节点的标识符。*(自动更新)*

incidentVel(*ScGeom*参数*1*，*(Interaction)i*[，*(bool)avoidGranularRatcheting=True*]) → Vec- tor3：返回相互作用的入射速度（另请参阅*Ig2_球体_球体_ Sc-Geom.avoidGranularRatcheting*，了解棘轮效应参数的说明）。

initialOrientation1(=*Quaternionr(1.0，0.0，0.0，0.0)*)刚体 1 在初始化时刻的方向*(自动更新)*
initialOrientation2(=*Quaternionr(1.0，0.0，0.0，0.0)*)刚体 2 在初始化时刻的方向*(自动更新)*
isDuplicate(=*0*)如果接触由两个连接共享，此标志会自动设为 true (1)。本构定律会跳过一次重复的相互作用，因此同一时刻只有一个接触有效。如果 isDuplicate=2，则表示两个重复项之一已不再具有几何相互作用，应由本构定律将其删除。

normal(=未初始化)沿相互作用方向的单位向量，从粒子 #1 指向粒子 #2。*(自动更新)*

penetrationDepth(=非数)球体的穿透距离（重叠时为正）**refR1**(=未初始化)粒子 #1 的参考半径。（自动计算）**refR2**(=未初始化)粒子 #2 的参考半径。（自动计算）**relAngVel**((ScGeom)参数1, (Interaction)i）→ Vector3：返回相互作用的相对角速度。**relPos**(=*0*)连接上的接触位置（0：节点-, 1：节点+）（自动更新）**shearInc**(=Vector3r::Zero())上一步中的剪切位移增量**trueInt**(=*-1*)定义发生真实接触的 GridConnection 的刚体 ID，条件是 ScGridCo-Geom::isDuplicate>0。**twist**(=*0*)接触点绕法向轴的弹性扭转角。**twistCreep**(=Quaternionr(1.0, 0.0, 0.0, 0.0))存储的蠕变，在计算弹性力矩时从总相对转动中扣除（自动更新）**updateAttrs**((Serializable)参数1, (dict)参数*2*）→ None：根据给定字典更新对象属性**weight**(=Vector3r(0, 0, 0))投影点的重心坐标（自动更新）

class yade.wrapper.**TTetraGeom**(继承 IGeom → 可序列化对象)两个四面体之间的相互作用几何，包括体积特性**contactPoint**(=未初始化)接触点（全局坐标）**dict**((Serializable)参数*1*）→字典：返回属性字典。**dispHierarchy**((IGeom)参数*1*[, (bool)名称=True]）→ 列表：返回分派类列表（从下到上），以类实例本身开头，顶层可索引对象位于最后。如果名称为 true（默认值），则返回类名而非数值索引。**dispIndex**返回此实例的类索引。**equivalentCrossSection**(=非数)重叠的截面（垂直于最小惯性轴**equivalentPenetrationDepth**(=非数)??**maxPenetrationDepthA**(=非数)??**maxPenetrationDepthB**(=非数)??**normal**(=未初始化)相互作用的法向量，方向沿重叠体积的最小惯性方向

penetrationVolume(=非数)重叠体积 [米^3]**updateAttrs**((可序列化对象)参数**1**，（字典）辐角 *2*) → 无：从给定字典更新对象属性**class yade.wrapper.TTetraSimpleGeom**(继承自 IGeom → 可序列化对象)实验性功能。两个四面体之间的相互作用几何**contactPoint**(=未初始化)接触点（全局坐标）**dict**((可序列化对象)辐角 *1*) →字典：返回属性字典。**dispHierarchy**((IGeom)辐角 *1*[, (布尔值)名称=真]) → 列表：返回分派类列表（从下到上），从类实例本身开始，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而不是数值索引。**dispIndex**返回此实例的类索引。**flag**(=*0*) 待办事项**normal**(=未初始化)相互作用的法向量，待办事项**penetrationVolume**(=非数)重叠体积 [米^3] **updateAttrs**((可序列化对象)参数**1**，（字典）辐角*2*) → 无：从给定字典更新对象属性

IPhys

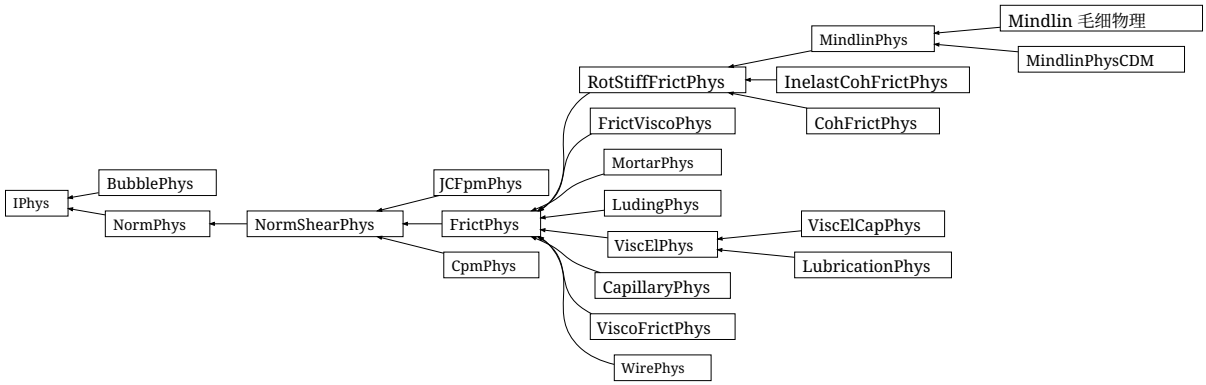


图 *24*：*IPhys* 的继承图。另请参阅：*BubblePhys*、*CapillaryPhys*、*CohFrictPhys*、*CpmPhys*、*FrictPhys*、*FrictViscoPhys*、*InelastCohFrictPhys*、*JCFpmPhys*、*LubricationPhys*、*LudingPhys*、*Mindlin-CapillaryPhys*、*MindlinPhys*、*MindlinPhysCDM*、*MortarPhys*、*NormPhys*、*NormShearPhys*、*RotStiff-FrictPhys*、*ViscElCapPhys*、*ViscElPhys*、*ViscoFrictPhys*、*WirePhys*。

class yade.wrapper.IPhys(继承可序列化对象)
相互作用的物理（材料）属性相互作用。**dict**(
（可序列化对象）辐角 *1*) →字典：返回属性字典。

dispHierarchy（*IPhys*）辐角*1*[，（布尔值）名称=真]）→ 列表：返回分派类的列表（由下至上），从类实例本身开始，最后是顶层可索引对象。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。updateAttrs（（*Serializable*）参数*1*，（字典）辐角*2*）→ 无：从给定字典更新对象属性

class yade.wrapper.BubblePhys(继承 *IPhys* → *Serializable*)用于 BubbleMat 的气泡间相互作用物理模型

Dmax(=非数)气泡在力—位移曲线转变为人工指数曲线之前的最大穿透深度。设置此值不会产生任何效果。更多信息请参阅定律2_ScGeom_- BubblePhys_Bubble::pctMaxForce

static computeForce（（浮点数）参数*1*，（浮点数）参数*2*，（浮点数）参数*3*，（整数）参数*4*，（浮点数）参数*5*，（浮点数）参数*6*，（浮点数）参数*7*，（*BubblePhys*）辐角*8*）→ 浮点数：使用 Newton-Rhapson 方法计算两个相互作用气泡之间的法向力dict（（*Serializable*）辐角*1*）→字典：返回属性字典。dispHierarchy（（*IPhys*）辐角*1*[，（布尔值）名称=真]）→ 列表：返回分派类列表（从下到上），以类实例本身开头，最后为顶层可索引对象。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。fN(=非数)接触法向力newtonIter(=*50*)允许的最大力迭代次数newtonTol(=*1*e-*6*)力迭代的收敛准则normalForce(=Vector3r::Zero())法向力rAvg(=非数)两个相互作用气泡的平均半径surfaceTension(=非数)周围液体的表面张力updateAttrs((Serializable)参数1, (字典)辐角*2*）→ 无：根据给定字典更新对象属性class yade.wrapper.CapillaryPhys(继承 FrictPhys → NormShearPhys → NormPhys →IPhys → Serializable)Law 的物理（相互作用）*2*_ScGeom_CapillaryPhys_毛细作用。Delta1(=*0*。)定义弯月面在半径为 R1 的较小颗粒（R1<R2）上润湿的表面积Delta2(=*0*。)定义弯月面在半径为 R2 的较大颗粒（R1<R2）上润湿的表面积

capillaryPressure(=*0*。)毛细压力 Uc 的值。定义为 Ugas-Uliquid，由以下内容获得 相应的 *Law2* 参数dict(*(Serializable)*辐角 *1*）→字典：返回属性字典。dispHierarchy(*(IPhys)*辐角 *1*[, *(*布尔值*)*名称=真])→ 列表：返回分派类列表（从下至上），从类实例本身开始，最后为顶层可索引对象。若 names 为真（默认值），则返回类名而非数值索引。

dispIndex

返回此实例的类索引。fCap(=*Vector3r::Zero()*)弯月面产生的毛细力。此力作用于粒子 #2 fusionNumber(=*0*。)表示与此弯月面重叠的弯月面数量isBroken(=*false*)用户可将其设为 true，使液桥处于非活动状态（毛细力为零）kn(=*0*)法向刚度ks(=*0*)剪切刚度meniscus(=*false*)当已为此相互作用计算出液体体积非零（弯月面体积）的弯月面时为 True

nn11(=*0*。)$\iint_A n_1 n_1 \, dS = \iint_A n_2 n_2 \, dS$，A 作为弯月面的液气界面，$\mathbf{n}$ 为关联的法向量，而 $(1,2,3)$ 为局部基，其中 3 表示弯月面方向（*ScGeom.normal*）。注意：A= 2 *nn11* + *nn33*。

nn33(=*0.*)$\iint_A n_3 n_3 \, dS$，A 表示弯月面的液气界面，$\mathbf{n}$ 表示相关法向量，而$(1,2,3)$ 表示局部基，其中 3 表示弯月面方向（ScGeom.normal）。注意：$A = 2$nn*11* +nn33.normalForce(=Vector3r::Zero())前一步后的法向力（全局坐标系下），由粒子 #2 承受（来自粒子 #1）。shearForce(=Vector3r::Zero())前一步后的剪切力（全局坐标系下），由粒子 #2 承受（来自粒子 #1）。tangensOfFrictionAngle(=非数)摩擦角的正切值updateAttrs((可序列化对象) arg1, (字典)arg*2*）→ 无：根据给定字典更新对象属性vMeniscus(=*0.*)弯月面体积

class yade.wrapper.CohFrictPhys(继承*RotStiffFrictPhys* → *FrictPhys* → *NormShearPhys*→ *NormPhys* → *IPhys* → 可序列化对象)一种相互作用物理，扩展了*RotStiffFrictPhys* 并添加了可断裂的黏聚特性。例如由定律*2_ScGeom6D_CohFrictPhys_CohesionMoment*。

cohesionBroken(=是)黏聚力是否处于活动状态？创建黏聚接触时设为 false，脆性接触断裂时设为 true**cohesionDisablesFriction**(=否)剪切强度是摩擦与黏附之和，还是仅指黏附？

creep_viscosity(=*-1*)蠕变黏度 [帕·秒/米]。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

dispHierarchy(*(IPhys)*辐角 *1*[, *(*布尔值*)*名称=真]) → 列表：返回分派类列表（从下到上），从类实例本身开始，最后是顶层可索引对象。如果名称为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

fragile(=真)接触强度超过阈值时，黏聚力会消失吗？

initCohesion(=否)使用当前状态将黏聚行为初始化为平衡状态（与 *Ip2_Co-hFrictMat_CohFrictMat_CohFrictPhys::setCohesionNow* 相同，但仅作用于一个相互作用）

kn(=*0*)法向刚度

kr(=*0*)转动刚度 [N.m/rad]

ks(=*0*)剪切刚度

ktw(=*0*)扭转刚度 [N.m/rad]

maxRollPl(=*0.0*)滚动摩擦系数（负值表示弹性）。**maxTwistPl**(=*0.0*)扭转摩擦系数（负值表示弹性）。**momentRotationLaw**(=false)设为 CohFrictMat::momentRotationLaw 中的值，以便在接触处使用弯曲／扭转力矩（如果为 true）。详见 Law2_ScGeom6D_CohFrictPhys_CohesionMoment::always_use_- 力矩定律。**moment_bending**(=Vector3r(0, 0, 0))弯矩 **moment_twist**(=Vector3r(0, 0, 0))扭矩**normalAdhesion**(=*0*)抗拉强度**normalForce**(=Vector3r::Zero())前一步的法向力（全局坐标），由粒子 #2（来自粒子 #1）承受。**shearAdhesion**(=*0*)剪切强度的黏聚部分（根据 CohFrict- Phys::cohesionDisablesFriction，可能会添加摩擦项）

shearForce(=*Vector3r::Zero()*)前一步的剪切力（全局坐标），由粒子 #2（来自粒子 #1）承受。

tangensOfFrictionAngle(=非数)
摩擦角的正切

unp(=*0*)塑性法向位移，仅用于拉伸行为且 *CohFrictPhys::fragile*=为 false。

unpMax(=*0*)塑性法向位移的最大值（按正值计），超过该值后，即使 *CohFrictPhys::fragile* =为 false，相互作用也会断裂。负值（即 -1）表示无最大值。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) → None：根据给定字典更新对象属性

class yade.wrapper.**CpmPhys**(继承 *NormShearPhys* → *NormPhys* → *IPhys* → *Serializable*)

Cpm 类型单个相互作用的表示方式：用于存储相关参数。

接触的演化由 *定律2_ScGeom_CpmPhys_Cpm* 控制，其中包含损伤效应以及 CpmPhys 内部参数的变化。详情请参见 *CPM 模型* 。

E(=非数)法向模量（刚度 / 横截面） [帕]

Fn法向力的模 *(*自动更新*)***Fs**剪切力的模 *(*自动更新*)***G**(=非数)
剪切模量 [帕]**crossSection**(=非数)与此接触关联的等效横
截面 [米^2]

cummBetaCount = 0

cummBetaIter = 0

damLaw(=*1*)单轴拉伸软化部分的定律。0 表示线性，1 表示指数函数（默认）

dict(*(*可序列化对象*)*辐角 *1*) →字典：
返回属性字典。

dispHierarchy(*(IPhys)*辐角 *1*[, *(*布尔值*)*名称=真])→ 列表：返回分派类列表（从下至上），从类实例本身开始，最后是顶层可索引对象。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

dmgOverstress(=*0*)
损伤黏性超应力（前一步或当前步）

dmgRateExp(=*0*)
率相关损伤演化中的指数

dmgStrain(=*0*)
损伤应变（前一步或当前步）

dmgTau(=*-1*)损伤特征时间（若为非正值，则使用不含率相关性的定律）

epsCrackOnset(=非数)材料开始呈现非线性行为时的应变

epsFracture(=非数)键完全断裂时的应变 [-]

epsN当前法向应变 *(自动更新)*

epsNPl法向塑性应变（初始为零向量） *(自动更新)*

epsT当前剪切应变 *(自动更新)*

epsTPl剪切塑性应变（初始为零向量） *(自动更新)*

equivStrainShearContrib(=非数)等效应变的剪切贡献系数

static funcG(*(浮点数)kappaD*，*(浮点数)epsCrackOnset*，*(浮点数)epsFracture*[,*(布尔值)neverDamage*=否[,*(整数)damLaw*=*1*]]) → 浮点数：损伤演化定律，用于计算 ω 参数。 κ_D 历史上表示最大应变，*ep-sCrackOnset* (ε_0) = *CpmPhys.epsCrackOnset*，*epsFracture* = *CpmPhys.epsFracture*；如果 *never-Damage* 为 True，则返回值始终为 0（无损伤）。TODO

static funcGInv(*(浮点数)omega*，*(浮点数)epsCrackOnset*，*(浮点数)epsFracture*[,*(布尔值)neverDamage*=否[,*(整数)damLaw*=*1*]]) → 浮点数：损伤演化定律的逆序，用于计算 κ_D 参数。 ω 表示损伤，其他参数请参见 funcG 函数

isCohesive(=否)如果不是黏结状态，当距离大于零时，相互作用将被删除。

isoPrestress(=*0*)此链接的“预应力”（用于模拟各向同性应力）

kappaD
截至目前的最大法向应变（半范数），随时间不减（自动更新） kn(=*0*)法向刚度ks(=*0*)剪切刚度neverDamage(=否)损伤演化函数始终会返回初始状态normalForce(=*Vector3r::Zero()*)前一步之后的法向力（全局坐标系中），由粒子 #2 承受（来自粒子 #1）。omega损伤内部变量（自动更新） plRateExp(=*0*)率相关黏塑性的指数plTau(=*-1*)黏塑性的特征时间（若非正值，则剪切不具有率相关性） refLength(=非数)相互作用的初始长度 [米]

refPD(=非数)相互作用的初始穿透深度 [米] （与 ScGeom 配合使用）relDuctility(=非数)键在法向上的相对延性 relResidualStrength相对残余强度（自动更新）setDamage(（*CpmPhys*）参数*1*，（浮点数）辐角*2*) → 无：待办

setRelResidualStrength(*(CpmPhys)*参数*1*，*(*浮点数*)*参数*2*) → 无：待办

shearForce(=*Vector3r::Zero()*)前一步之后的剪切力（以全局坐标表示），由粒子 #2（来自粒子 #1 ）承担。

sigmaN
: 当前法向应力（自动更新）

sigmaT
当前剪切应力（自动更新）tanFrictionAngle(=非数)内摩擦角的正切 [-]undamagedCohesion(=非数)原始材料黏聚力 [帕]updateAttrs((Serializable)参数1, (字典)参数*2*) → 无：根据给定字典更新对象属性class yade.wrapper.FrictPhys(继承自 NormShearPhys → NormPhys → IPhys → Serializable)具有摩擦角的简单线弹塑性相互作用，类似于传统的[CundallStrack1979]dict((Serializable)参数*1*) →字典：返回属性字典。dispHierarchy((IPhys)参数*1*[, (布尔值)名称=是]) → 列表：返回分派类列表（从下至上），以类实例自身开头，最后为顶层可索引对象。如果 names 为真（默认），则返回类名而非数值索引。

dispIndex
: 返回此实例的类索引。kn(=*0*)法向刚度ks(=*0*)剪切刚度normalForce(=*Vector3r::Zero()*)前一步结束时粒子 #2（由粒子 #1 施加）承受的法向力（以全局坐标表示）。shearForce(=*Vector3r::Zero()*)前一步结束时粒子 #2（由粒子 #1 施加）承受的剪切力（以全局坐标表示）。tangensOfFrictionAngle(=非数)摩擦角的正切值updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → 无：根据给定字典更新对象属性

class yade.wrapper.FrictViscoPhys(继承 *FrictPhys* → *NormShearPhys* → *NormPhys* → *IPhys* → 可序列化对象)FrictViscoPM 类型的单个相互作用的表示方式，用于存储相关参数**cn**(=非数)法向黏性常数定义为 $\eta = c_{n,\ crit}\beta_n$。**cn_crit**(=非数)临界阻尼对应的法向黏性常数定义为 $\eta = C_{n,\ crit}\beta_n$。**dict**(*(*可序列化对象*)*辐角 *1*) →字典：返回属性字典。**dispHierarchy**(*(IPhys)*参数 *1*[, *(*布尔值*)* 名称=真]) → 列表：返回分派类列表（从下至上），以类实例本身开头，以顶层可索引对象结尾。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex

返回此实例的类索引。**kn**(=*0*)法向刚度**ks**(=*0*)剪切刚度**normalForce**(=Vector3r::Zero())前一步的法向力（全局坐标），由粒子 #2 承受（来自粒子 #1）。**normalViscous**(=Vector3r::Zero())法向黏性分量**shearForce**(=Vector3r::Zero())前一步的剪切力（全局坐标），由粒子 #2 承受（来自粒子 #1）。**tangensOfFrictionAngle**(=非数)摩擦角的正切**updateAttrs**((Serializable)参数1, (dict)辐角*2*) → None：根据给定字典更新对象属性**class yade.wrapper.InelastCohFrictPhys**(继承 RotStiffFrictPhys → FrictPhys → NormS- hearPhys → NormPhys → IPhys → 可序列化对象) **cohesionBroken**(=false)黏聚力是否处于活动状态？脆性接触断裂时将设为 false**dict**((Serializable) 辐角 *1*) →字典：返回属性字典。**dispHierarchy**((IPhys)参数 *1*[, (bool)名称=True]) → 列表：返回分派类列表（从下到上），从类实例本身开始，顶层可索引对象位于最后。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex

返回此实例的类索引。**isBroken**(=*false*)是否已达到压缩塑性断裂**kDam**(=*0*)弯曲损伤系数，根据达到的最大弯矩和纯

蠕变行为。其值将在 *InelastCohFrictPhys::kr* 和 *InelastCohFrict-Phys::kRCrp* 之间变化。

kRCrp(=*0.0*)弯曲蠕变刚度**kRUnld**(=*0.0*)弯曲塑性卸载刚度**kTCrp**(=*0.0*)拉压蠕变刚度**kTUnld**(=*0.0*)拉压塑性卸载刚度

kTwCrp(=*0.0*)扭转蠕变刚度

kTwUnld(=*0.0*)扭转塑性卸载刚度

kn(=*0*)法向刚度

knC(=*0*)压缩刚度

knT(=*0*)拉伸刚度

kr(=*0*)转动刚度 [N.m/rad]

ks(=*0*)剪切刚度**ktw**(=*0*)扭转刚度 [牛·米/弧度]**maxBendMom**(=*0。 0*)塑性失效弯矩。**maxContract**(=*0。 0*)塑性失效收缩。**maxCrpRchdB**(=Vector3r(0, 0, 0))塑性变形过程中达到的最大弯矩。**maxCrpRchdC**(=Vector2r(0, 0))塑性变形过程中达到的最大压缩量。maxCrpRchdC[0]存储未达到值和最大值。CrpRchdC[1]存储法向力。**maxCrpRchdT**(=Vector2r(0, 0))塑性变形过程中达到的最大伸长量。maxCrpRchdT[0] 存储未达到值和最大值。Cr-pRchdT[1]存储法向力。**maxCrpRchdTw**(=Vector2r(0, 0))塑性变形过程中达到的最大扭转量。maxCrpRchdTw[0]存储扭转角，maxCrpRchdTw[1]存储扭矩。**maxElB**(=*0。 0*)最大弹性弯矩。**maxElC**(=*0。 0*)最大弹性压缩力。**maxElT**(=*0。 0*)最大弹性拉力。

maxElTw(=*0*。*0*)最大扭转弹性力矩。maxExten(=*0*。*0*)塑性破坏伸长量。maxTwist(=*0*。*0*)塑性破坏扭转角moment_bending(=Vector3r(0, 0, 0))弯矩moment_twist(=Vector3r(0, 0, 0))扭矩normalForce(=Vector3r::Zero())前一步后的法向力（在全局坐标中），由粒子 #2 承受（来自粒子 #1）。onPlastB(=false)弯曲达到塑性状态时为 trueonPlastC(=false)压缩达到塑性状态时为 trueonPlastT(=false)拉伸达到塑性状态时为 trueonPlastTw(=false)扭转达到塑性状态时为 truepureCreep(=Vector3r(0, 0, 0))纯蠕变曲线，用于计算中的比较。shearAdhesion(=*0*)最大弹性剪切力（黏聚力）。shearForce(=Vector3r::Zero())前一步后的剪切力（在全局坐标中），由粒子 #2 承受（来自粒子 #1）。tangensOfFrictionAngle(=非数)摩擦角的正切值twp(=*0*)描述平衡状态的塑性扭转穿透深度。unp(=*0*)描述平衡状态的塑性法向穿透深度。updateAttrs((可序列化对象)参数1，（字典）参数*2*) → 无：从给定字典更新对象属性

class yade.wrapper.JCFpmPhys(继承 NormShearPhys → NormPhys → IPhys → Serializable)

JCFpm 类型单个相互作用的表示方式，用于存储相关参数FnMax(=*0*。)根据抗拉强度（或节理变体）计算出的正值，用于定义拉伸状态下的最大允许法向力：Fn >= -FnMax。 [N]FsMax(=*0*。)根据黏聚力（或节理黏聚力）计算，用于定义剪切状态下的最大允许切向力，适用于 Fn=0。 [N]checkedForCluster(=否)我们是否已检查此相互作用是否属于簇？clusterInts(=取消初始化)指向附近断裂相互作用的指针向量，这些相互作用共同构成一个簇

clusteredEvent(=否)此相互作用是否
属于一个簇? **computedCentroid**(=否)
用于力矩计算的标志

crackJointAperture(=*0*。)两个球体之间的相对位移（对于裂纹而言，它等同于裂纹张开度）

crossSection(=*0*。)横截面=pi*Rmin^2。
[m2]

dict(*(Serializable)*辐角 *1*) →字典:
返回属性字典。

dilation(=*0*。)定义滑动阈值之后节理中的法向位移。 [m]

dispHierarchy(*(IPhys)*辐角 *1*[, *(bool)*名称=*True*]) → 列表：返回分派类列表（从下到上），从类实例本身开始，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名，而非数值索引。

dispIndex
返回此实例的类索引。

elapsedIter(=*0*)用于力矩计算的已经过迭代次数

eventBeginTime(=*0*)事件启动的时间

eventNumber(=*0*)聚类事件编号

firstMomentCalc(=*true*)力矩计算标志 （自动更新）

initD(=*0*。)相互作用粒子的平衡距离。计算为首次接触检测时的颗粒间距。

interactionsAdded(=*false*)我们是否已添加与此事件相关的交互作用?

isBroken(=*false*)断裂交互作用的标志

isCohesive(=*false*)如果为 false，颗粒将以摩擦方式相互作用。如果为 true，颗粒将根据给定的黏聚力和抗拉强度（或其节理变体）进行黏结。

isOnJoint(=*false*)
当两个相互作用的颗粒都位于节理上且处于节理面的两侧时，定义为 true。在这种情况下，相互作用的力学参数由颗粒的“joint...”材料属性推导得出。此外，相互作用的法向量可能会重新定向（参见定律*2_ScGeom_JCFpmPhys_JointedCohesiveFrictionalPM.smoothJoint*）。

isOnSlot(=*false*)
当相互作用位于穿孔槽（表面）中时，定义为 true。

jointCumulativeSliding(=*0*。)节理上相互作用颗粒的滑移距离。当为 true 时，用于考虑剪切引起的剪胀。 [-]

jointNormal(=*Vector3r::Zero()*)节理的法向量，例如可由此推导：。**kineticEnergy**(=*0*)参与相互作用的两个球体的动能（最简便的做法是将此值与相互作用一起存储，而非存储在球体中，因为我们利用此信息估算力矩模，并进行相关的相互作用搜索）

kn(=*0*)法向刚度

ks(=*0*)剪切刚度**momentBroken**(=否)力矩计算标志**momentCalculated**(=否)用于避免重复执行两次操作的力矩计算标志 *(*自动更新*)***momentCentroid**(=*Vector3r::Zero()*)声发射事件的质心（聚类破裂的平均位置）

momentEnergy(=*0*)周围相互作用（颗粒）的参考应变（或动能）

momentEnergyChange(=*0*)存储周围相互作用（颗粒）的最大应变（或动能）变化量

momentMagnitude(=*0*)失效相互作用的矩震级

more(=*false*)指定相互作用是否被超过 3 个节理穿过。如果为 true，则删除相互作用（临时求解）。

nearbyFound(=*0*)用于调试力矩计算的计数**nearbyInts**(=未初始化)用于力矩计算的邻近相互作用指针向量**normalForce**(=*Vector3r::Zero()*)前一步之后的法向力（在全局坐标中），由粒子 #2 承受（来自粒子 #1）。**originalClusterEvent**(=否)簇的原始声发射事件**originalEvent**(=未初始化)指向簇原始相互作用的指针**shearForce**(=*Vector3r::Zero()*)前一步之后的剪切力（在全局坐标中），由粒子 #2 承受（来自粒子 #1）。**strainEnergy**(=*0*)相互作用的应变能 **tanDilationAngle**(=*0*。)定义节理面剪胀的角度的正切值（根据 *JCFp-mMat.jointDilationAngle* 自动计算）。 [-]**tanFrictionAngle**(=*0*。)此相互作用的库仑摩擦角的正切值（自动计算）。 [-]

temporalWindow(=0)聚类算法的时间窗口

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无：根据给定字典更新对象属性

class yade.wrapper.LubricationPhys(继承 ViscElPhys → FrictPhys → NormShearPhys → NormPhys → IPhys → 可序列化对象)用于无 FlowEngine 润滑的 IPhys 类。由 Law2_ScGeom_ImplicitLubricationPhys 使用。Fn(=0。0)接触的法向力Fv(=0。0)接触的黏性力a(=0。)平均半径 [米]cn(=非数)法向黏性常数contact(=否)球体处于接触状态cs(=非数)剪切黏性常数delta(=0)日志(u) - 用于采用 δ =日志(u) 变量变换的方案dict((可序列化对象)辐角1) →字典：返回属性字典。

dispHierarchy((IPhys)辐角1[, (布尔值)名称=真]) → 列表：返回分派类列表（从下至上），以类实例本身开头，最后为顶层可索引对象。如果 names 为真（默认值），则返回类名，而不是数值索引。

dispIndex
返回此实例的类索引。eps(=0。001)粗糙度：用作粗糙度的半径比例 [-]

eta(=1)流体黏度 [帕·秒]

keps(=1)凸体的刚度系数 [N/m]。仅用于分辨率方法=0；对于分辨率>0，它始终等于 kn。kn(=0)法向刚度kno(=0。0)法向刚度系数（赫兹型接触） [N/m^(3/2)]ks(=0)剪切刚度mR(=0。0)滚动阻力，参见 [Zhou1999536]。mRtype(=1)滚动阻力类型，参见 [Zhou1999536]。mRtype =1 — [Zhou1999536]中的方程（3）；mR-type=2 — [Zhou1999536]中的方程（4）

mum(=0。3)摩擦系数 [-]normalContactForce(=Vector3r::Zero())法向接触力 [N] normalForce(=Vector3r::Zero())前一步的法向力（在全局坐标中），由颗粒 #2 承受（来自颗粒 #1 ）。normalLubricationForce(=Vector3r::Zero()) 法向润滑力 [N] normalPotentialForce(=Vector3r::Zero())接触之外由势产生的法向力 [N]nun(=0。0)法向润滑系数 [N.s]prevDotU(=0)前一次积分中的 du/dt——用于梯形格式（参见定律2_ScGeom_ Implic- itLubricationPhys::resolution，以选择分辨率方案）prev_un(=0)t-dt 时刻的未变形距离（un） [m]shearContactForce(=Vector3r::Zero())摩擦接触力 [N]shearForce(= Vector3r::Zero())前一步的剪切力（在全局坐标中），由颗粒 #2 承受（来自颗粒 #1）。 shearLubricationForce(=Vector3r::Zero())剪切润滑力 [N]slip(=false)接触正在滑移 tangensOfFrictionAngle(=非数)摩擦角的正切u(=-1)t-dt 时刻的界面距离（u） [m]ue(=0。) t-dt 时刻的表面偏转（ue） [m]updateAttrs((可序列化对象)arg1，（字典）arg2）→ None：从给定字典更新对象属性

class yade.wrapper.LudingPhys(继承 FrictPhys → NormShearPhys → NormPhys → IPhys→ 可序列化对象)由 LudingMat 创建的 IPhys，用于 Law2_ScGeom_LudingPhys_ 基础。DeltMax(=非数)碰撞时颗粒之间的最大重叠量DeltMin(=非数)delta 的最小值 DeltNull(=非数)无力重叠量、塑性接触变形DeltPMax(=非数)极限情况下颗粒之间的最大重叠量DeltPNull(=非数)最大无力重叠量、塑性接触变形

DeltPrev(=非数)delta 的先前值G0(=非数)黏性阻尼PhiF(=非数)无量纲塑性深度dict((可序列化对象)辐角 *1*) →字典：返回属性字典。dispHierarchy((*IPhys*)辐角 *1*[, (布尔值)名称=真]) → 列表：返回分派类列表（从下至上），以类实例本身开头，顶层可索引对象位于末尾。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。k1(=非数)加载塑性分支的斜率k2(=非数)卸载与再加载弹性分支的斜率kc(=非数)不可逆拉伸黏结分支的斜率

kn(=*0*)法向刚度

kp(=非数)卸载与再加载极限弹性分支的斜率ks(=*0*)剪切刚度normalForce(=*Vector3r::Zero()*)前一步的法向力（以全局坐标表示），由粒子 #2 承受（来自粒子 #1）。shearForce(=*Vector3r::Zero()*)前一步的剪切力（以全局坐标表示），由粒子 #2 承受（来自粒子 #1）。tangensOfFrictionAngle(=非数)摩擦角的正切值updateAttrs((可序列化对象)参数*1*, (字典)辐角 *2*) → 无：根据给定字典更新对象属性

class yade.wrapper.MindlinCapillaryPhys(继承 *MindlinPhys* → *RotStiffFrictPhys* → *FrictPhys* → *NormShearPhys* → *NormPhys* →*IPhys* →可序列化对象)为 Mindlin 相互作用物理添加毛细管物理。Delta1(=*0*。)定义半径为 R1 的较小颗粒上被弯月面润湿的表面积（R1< R2）Delta2(=*0*。)定义半径为 R2 的较大颗粒上被弯月面润湿的表面积（R1<R2）Fs(=*Vector2r::Zero()*)局部坐标轴中的剪切力（增量计算）adhesionForce(=*0*。*0*)DMT 预测的黏附力

alpha(=*0*。*0*)用于定义非线性弹性力—位移关系中接触黏性阻尼的恒定系数。**betan**(=*0*。*0*)法向阻尼比。法向（方向）黏性阻尼系数中等于 c_{n} $c_{n,crit}$。**betas**(=*0*。*0*)剪切阻尼比。剪切（方向）黏性阻尼系数中等于$c_s c_{s,crit}$。**capillaryPressure**(=*0*。)毛细压力 Uc 的值。定义为 Ugas-Uliquid，由 对应的 *Law2* 参数**dict**(*(*可序列化对象*)*辐角*1*）→字典：返回属性字典。**dispHierarchy**(*(IPhys)*辐角*1*[, *(*布尔值*)*名称=真])→ 列表：返回分派类列表（从下至上），以类实例开头，顶层可索引对象位于末尾。如果 names 为真（默认值），则返回类名而非数值索引。

———

dispIndex
返回此实例的类索引。

fCap(=*Vector3r::Zero()*)弯月面的存在会产生毛细力。这是作用于粒子 #2 的力。

fusionNumber(=*0*。)表示与此弯月面重叠的弯月面数量**isAdhesive**(=否)用于标识接触是否具有黏附性，即接触力是否为吸引力的布尔值**isBroken**(=否)用户可将其设为 true，使液桥失效（毛细力为零）**isSliding**(=否)检查接触是否处于滑动状态（便于计算滑动接触的比例）**kn**(=*0*)法向刚度**kno**(=*0*。*0*)法向刚度公式中的常数值**kr**(=*0*)转动刚度 [N.m/rad]**ks**(=*0*)剪切刚度**kso**(=*0*。*0*)切向刚度公式中的常数值**ktw**(=*0*)扭转刚度 [N.m/rad]**maxBendPl**(=*0*。*0*)用于确定接触处所施加最大塑性力矩的系数**meniscus**(=否)当此相互作用已计算出具有非零液体体积（弯月面体积）的弯月面时为 True

momentBend(=Vector3r::Zero())人工弯矩，用于提供滚动阻力，以考虑颗粒间一定程度的嵌锁 momentTwist(=Vector3r::Zero())人工扭转力矩（目前无法应用塑性条件）normalForce(=Vector3r::Zero())前一步的法向力（全局坐标），由粒子 #2 承受（来自粒子 #1）。normalViscous(=Vector3r::Zero())法向黏性分量prevU(=Vector3r::Zero())前一步的局部位移；仅与定律*2*_L3Geom_FrictPhys_HertzMindlin.radius(=非数)接触半径（仅在定律*2*_ScGeom_MindlinPhys_Mindlin::calcEnergy中计算）shearElastic(=Vector3r::Zero())总弹性剪切力shearForce(=Vector3r::Zero())前一步的剪切力（全局坐标），由粒子 #2 承受（来自粒子 #1）。shearViscous(=Vector3r::Zero())剪切黏性分量tangensOfFrictionAngle(=非数)摩擦角的正切值updateAttrs((可序列化对象)参数1, (字典)辐角*2*) → 无：根据给定字典更新对象属性usElastic(=Vector3r::Zero())总弹性剪切位移（仅弹性部分）usTotal(=Vector3r::Zero())总弹性剪切位移（弹性+塑性部分）vMeniscus(=*0*。)弯月面的体积class yade.wrapper.MindlinPhys(继承 RotStiffFrictPhys → FrictPhys → NormShearPhys→ NormPhys → IPhys → 可序列化对象)Hertz-Mindlin 类型相互作用的表示方式。Fs(=Vector2r::Zero())局部坐标轴中的剪切力（增量计算）adhesionForce(=*0*。*0*)DMT 预测的黏附力alpha(=*0*。*0*)用于定义非线性弹性力—位移关系的接触黏性阻尼的常数系数。betan(=*0*。*0*)法向阻尼比。法向黏性阻尼系数中等于 c_n 的比例，$c_{n_{crit}}$。betas(=*0*。*0*)剪切阻尼比。剪切方向黏性阻尼系数中等于$c_s$$C_s$的比例，$c_{s_{crit}}$。

dict((可序列化对象)辐角*1*) →字典：返回属性字典。dispHierarchy((*IPhys*)辐角*1*[, (布尔值)名称=真]) → 列表：返回分派类列表（从下到上），从类实例本身开始，最后是顶层可索引对象。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex

返回此实例的类索引。isAdhesive(=false)布尔值，用于标识接触是否具有黏附性，也就是说接触力是否具有吸引性isSliding(=false)检查接触是否处于滑动状态（可用于计算滑动接触的比例）kn(=*0*)法向刚度kno(=*0*。*0*)法向刚度公式中的常数值kr(=*0*)转动刚度 [牛·米/弧度]ks(=*0*)剪切刚度kso(=*0*。*0*)切向刚度公式中的常数值ktw(=*0*)扭转刚度 [牛·米/弧度]maxBendPl(=*0*。*0*)用于确定接触处所施加最大塑性力矩的系数momentBend(=Vector3r::Zero())用于提供滚动阻力的人工弯矩，以考虑颗粒间一定程度的嵌锁momentTwist(=Vector3r::Zero())人工扭转力矩（目前无法应用塑性条件）normalForce(=Vector3r::Zero())前一步后的法向力（全局坐标系中），由粒子 #2 承受（来自粒子 #1）。normalViscous(=Vector3r::Zero())法向黏性分量prevU(=Vector3r::Zero())前一局部位移；仅与 Law 一起使用*2*_L3Geom__FrictPhys__HertzMindlin。radius(=非数)接触半径（仅在使用 Law 时计算*2*_ScGeom__MindlinPhys__Mindlin::calcEnergy)shearElastic(=Vector3r::Zero())总弹性剪切力shearForce(=Vector3r::Zero())前一步后的剪切力（全局坐标系中），由粒子 #2 承受（来自粒子 #1）。shearViscous(=Vector3r::Zero())剪切黏性分量

tangensOfFrictionAngle(=非数)摩擦角的正切值
updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
根据给定字典更新对象属性usElastic(=
Vector3r::Zero())总弹性剪切位移（仅弹性部分）
usTotal(=*Vector3r::Zero()*)总弹性剪切位移（弹性+塑性
部分）

class yade.wrapper.MindlinPhysCDM(继承 *MindlinPhys* → *RotStiffFrictPhys* → *FrictPhys* → *NormShearPhys* → *NormPhys* → *IPhys* → *Serializable*)

扩展赫兹-明德林类型相互作用的表示方式。法向：锥形损伤模型的参数（Harkness 等，2016；Suhr 和 Six，2017）。切向：应力相关的颗粒间摩擦系数的参数（Suhr 和 Six，2016）。这两个模型可以分别启用或关闭，参见 FrictMatCDM。

E(=*0.0*)[帕]等效杨氏模量Fs(=*Vector2r::Zero()*)局部坐
标轴中的剪切力（逐步计算）

G(=*0.0*)[帕]等效剪切模量

R(=*0.0*)[米] 锥形损伤模型中的接触半径

adhesionForce(=*0.0*)
DMT 预测的黏附力

alpha(=*0.0*)用于定义非线性弹性力—位移关系中接触黏性阻尼的常数系数。

alphaFac(=*0.0*)
考虑锥形凸起角度的因子betan(=*0.0*)法向阻尼比。黏性阻尼系数（法向）的比例，等于c_n，$_{crit}$.betas(=*0.0*)剪切阻尼比。黏性阻尼系数（剪切方向）的比例，等于$c_s/c_{s,crit}$.c1(=*0.0*)[-] 压力相关摩擦模型的参数 c1c2(=*0.0*)[-] 压力相关摩擦模型的参数 c2dict((可序列化对象)辐角1）→字典：返回属性字典。

---

dispHierarchy((*IPhys*)辐角*1*[, (布尔值)名称=*True*]) → 列表:
返回分派类列表（从下到上），从类实例本身开始，顶层可索引对象最后。如果 names 为 true（默认值），则返回类名而不是数值索引。

dispIndex
返回此实例的类索引。

isAdhesive(=false)用于标识接触是否具有黏附性的布尔值，也就是说，接触力是否具有吸引性 **isSliding**(=false)检查接触是否正在滑动（用于计算滑动接触的比例）**isYielding**(=false)布尔值：接触当前是否正在屈服？**kn**(=*0*)法向刚度**kno**(=*0*。*0*)法向刚度公式中的常数值**kr**(=*0*)转动刚度 [N.m/rad]**ks**(=*0*)剪切刚度**kso**(=*0*。*0*)切向刚度公式中的常数值**ktw**(=*0*)扭转刚度 [N.m/rad]**maxBendPl**(=*0*。*0*)用于确定接触处所施加最大塑性力矩的系数**momentBend**(=Vector3r::Zero())用于提供滚动阻力的人工弯矩，以考虑颗粒间一定程度的嵌锁**momentTwist**(=Vector3r::Zero())人工扭转力矩（目前无法应用塑性条件）**mu0**(=*0*。*0*)[-] 压力相关摩擦模型参数 mu0**normalForce**(=Vector3r::Zero())前一步的法向力（全局坐标系中），由粒子 #2 承受（来自粒子 #1）。**normalViscous**(=Vector3r::Zero())法向黏性分量**prevU**(=Vector3r::Zero())前一步的局部位移；仅与 Law 配合使用*2*_L3Geom_FrictPhys_HertzMindlin。**radius**(=非数)接触半径（仅在 Law*2*_ScGeom_MindlinPhys_Mindlin::calcEnergy 中计算）**shearElastic**(=Vector3r::Zero())总弹性剪切力**shearForce**(=Vector3r::Zero())前一步的剪切力（全局坐标系中），由粒子 #2 承受（来自粒子 #1）。**shearViscous**(=Vector3r::Zero())剪切黏性分量**sigmaMax**(=*0*。*0*)[帕]材料的最大抗压强度**tangensOfFrictionAngle**(=非数)摩擦角的正切值

updateAttrs(*(可序列化对象)arg1*，*(字典)*辐角*2*）→ 无：
根据给定字典更新对象属性usElastic(=
Vector3r::Zero())总弹性剪切位移（仅弹性部分）
usTotal(=*Vector3r::Zero()*)总弹性剪切位移（弹性+塑性
部分）

class yade.wrapper.MortarPhys(继承 *FrictPhys* → *NormShearPhys* → *NormPhys* → *IPhys*→ 可序列化对象)包含 MortarMat 参数的 IPhys 类。由定律使用2_ScGeom_MortarPhys_Lourenco。

cohesion(=非数)黏聚力 [帕]
compressiveStrength(=非数)抗压
强度 [帕]crossSection(=非数)相
互作用的横截面dict(*(可序列化对象)*
arg1）→字典：返回属性字典。

dispHierarchy(*(IPhys)arg1*[, *(布尔值)*名称=真])→ 列表：返回分派类列表（从下到上），以类实例本身开头，以顶层可索引对象结尾。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。
ellAspect(=非数)椭圆形“帽”的长宽比。值 >1 表示椭圆沿法向应力轴方向更长。

failureCondition(*MortarPhys 参数1*，浮点数参数*2*，浮点数辐角*3*）→ 布尔值：由法向应力和剪切应力范数确定的破坏条件（false=弹性，true=损伤）kn(=*0*)法向刚度ks(=*0*)剪切刚度
neverDamage(=*false*)若为 true，则无论应力如何，相互作用始终保持弹性normalForce(=
Vector3r::Zero())前一步后的法向力（全局坐标），由粒子 #2 承受（来自粒子 #1）。

shearForce(=*Vector3r::Zero()*)前一步后的剪切力（全局坐标），由粒子 #2 承受（来自粒子 #1）。

sigmaN
当前法向应力 *(自动更新)*
sigmaT
当前剪切应力*(自动更新)*
tangensOfFrictionAngle(=非数)摩擦
角的正切tensileStrength(=非数)抗
拉强度 [帕]

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无：根据给定字典更新对象属性
class yade.wrapper.NormPhys(继承IPhys → Serializ)具有法向刚度的相互作用的抽象类。dict((可序列化对象)辐角1) →字典：返回属性字典。dispHierarchy((IPhys)辐角1[, (布尔值)名称=真]) → 列表：返回分派类列表（从下至上），以类实例本身开头，并以顶层可索引对象结尾。如果 names 为真（默认值），则返回类名而不是数值索引。

dispIndex
返回此实例的类索引。kn(=0)法向刚度normalForce(=Vector3r::Zero())前一步后的法向力（全局坐标），由粒子 #2 承受（来自粒子 #1）。updateAttrs((Serializable)arg1、(dict)arg2) → None：根据给定字典更新对象属性class yade.wrapper.NormShearPhys(继承 NormPhys → IPhys → Serializable)除法向刚度外，还具有剪切刚度的相互作用抽象类。此类用于 PFC3d 风格刚度时间步进器。dict((Serializable)arg1) →dict：返回属性字典。dispHierarchy((IPhys)arg1[, (bool)names=True]) → list：返回分派类列表（从下至上），以类实例本身开头，顶层可索引对象位于最后。如果 names 为 true（默认值），则返回类名而非数值索引。dispIndex返回此实例的类索引。kn(=0)法向刚度ks(=0)剪切刚度normalForce(=Vector3r::Zero())前一步后的法向力（全局坐标），由粒子 #2 承受（来自粒子 #1）。shearForce(=Vector3r::Zero())前一步后的剪切力（全局坐标），由粒子 #2 承受（来自粒子 #1）。updateAttrs((Serializable)arg1、(dict)arg2) → None：根据给定字典更新对象属性class yade.wrapper.RotStiffFrictPhys(继承 FrictPhys → NormShearPhys → NormPhys→ IPhys → Serializable)具有转动刚度的 FrictPhys 版本dict((Serializable)arg1) → dict：返回属性字典。

dispHierarchy(*(IPhys)arg1*[, *(bool)names=True*]) → list：返回分派类列表（从下至上），以类实例本身开头，顶层可索引对象位于最后。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex

返回此实例的类索引。**kn**(=*0*)法向刚度**kr**(=*0*)转动刚度 [牛·米/弧度]**ks**(=*0*)剪切刚度**ktw**(=*0*)扭转刚度 [牛·米/弧度]**normalForce**(=Vector3r::Zero())前一步之后的法向力（以全局坐标表示），由粒子 #2 承受（来自粒子 #1）。**shearForce**(=Vector3r::Zero())前一步之后的剪切力（以全局坐标表示），由粒子 #2 承受（来自粒子 #1）。**tangensOfFrictionAngle**(=非数)摩擦角的正切**updateAttrs**((Serializable)参数1, (字典)辐角*2*) → 无 : 根据给定字典更新对象属性

class yade.wrapper.**ViscElCapPhys**(继承自 ViscElPhys → FrictPhys → NormShearPhys → NormPhys → IPhys → 可序列化对象)由 ViscElCapMat 创建的 IPhys，供定律使用*2*_ScGeom_ViscElCapPhys_基本。**Capillar**(=假)若需要添加毛细力，则为 True。**CapillarType**(=无_毛细)不同类型的毛细相互作用：Willett_数值，Willett_解析，Weigert，Rabinovich，Lambert，Soulie **Fn**(=*0*。*0*)接触法向力**Fv**(=*0*。*0*)接触黏性力**Vb**(=*0*。*0*)液桥体积 [米^3]**cn**(=非数)法向黏性常数**cs**(=非数)剪切黏性常数**dcap**(=*0*。*0*)毛细相的阻尼系数 [-]**dict**((可序列化对象)辐角*1*) →字典：返回属性字典。**dispHierarchy**((IPhys)参数*1*[, (布尔值)名称=True]) → 列表：返回分派类列表（从下至上），从类实例本身开始，

最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而不是数值索引。

dispIndex
返回此实例的类索引。**gamma**(=*0*。
0)表面张力 [N/m]

kn(=*0*)法向刚度

ks(=*0*)剪切刚度**liqBridgeActive**(=否)液桥当前是否处于活动状态**liqBridgeCreated**(=否)液桥是否已创建，且仅在球体发生法向接触后创建**mR**(=*0*。*0*)滚动阻力，参见 [Zhou1999536]。
mRtype(=*1*)滚动阻力类型，参见 [Zhou1999536]。mRtype=1 - Zhou 中的方程 (3) [Zhou1999536]; mR- type=2 - Zhou 中的方程 (4) [Zhou1999536]

normalForce(=*Vector3r::Zero()*)前一步的法向力（以全局坐标表示），由粒子 #2（来自粒子 #1）承受。

sCrit(=否)临界液桥长度 [m]**shearForce**(=*Vector3r::Zero()*)前一步的剪切力（以全局坐标表示），由粒子 #2（来自粒子 #1）承受。**tangensOfFrictionAngle**(=非数)摩擦角的正切值**theta**(=*0*。*0*)接触角 [rad]**updateAttrs**(*(*可序列化对象*)arg1, (*字典*)arg2*) → None：根据给定字典更新对象属性

class yade.wrapper.**ViscElPhys**(继承 FrictPhys → NormShearPhys → NormPhys → IPhys→ 可序列化对象)IPhys，由 ViscElMat 创建，供 Law2_ScGeom_ViscElPhys_Basic 使用。**Fn**(=*0*。*0*)接触的法向力**Fv**(=*0*。*0*)接触的黏性力**cn**(=非数)法向黏性常数**cs**(=非数)剪切黏性常数**dict**((可序列化对象)辐角 *1*) →字典：返回属性字典。

dispHierarchy(*(IPhys)*辐角 *1*[, *(*布尔值*)*名称=真]) → 列表：返回分派类列表（从下到上），以类实例本身为起点，顶层可索引对象位于末尾。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。**kn**(=*0*)法向刚度**ks**(=*0*)剪切刚度

mR(=*0*。*0*)滚动阻力，参见 [Zhou1999536]。

mRtype(=*1*)滚动阻力类型，参见 [Zhou1999536]。mRtype=1 - Zhou 中的方程 (3) [Zhou1999536]；mR- 类型=2 - Zhou 中的方程 (4) [Zhou1999536]

normalForce(=*Vector3r::Zero()*)前一步之后的法向力（在全局坐标中），由粒子 #2（来自粒子 #1）承受。

shearForce(=*Vector3r::Zero()*)前一步之后的剪切力（在全局坐标中），由粒子 #2（来自粒子 #1）承受。

tangensOfFrictionAngle(=非数)
摩擦角的正切

updateAttrs(*(*可序列化对象*)arg1*，*(*字典*)arg2*) →
None：根据给定字典更新对象属性

class yade.wrapper.**ViscoFrictPhys**(继承 *FrictPhys* → *NormShearPhys* → *NormPhys* → *IPhys*[*y*s → 可序列化对象])
出于兼容性考虑的 *FrictPhys* 临时版本

creepedShear(=*Vector3r(0*，*0*，*0)*)蠕变力（并行）

dict(*(*可序列化对象*)*辐角 *1*) →字典：返回属性字典。**dispHierarchy**(*(IPhys)*辐角 *1*[, *(*布尔值*)*名称=真]) → 列表：返回分派类的列表（从下到上），从类实例本身开始，最后是顶层可索引对象。如果 names 为真（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。**kn**(=*0*)法向刚度**ks**(=*0*)剪切刚度**normalForce**(=*Vector3r::Zero()*)前一步结束后粒子 #2 所承受的法向力（全局坐标系下，来自粒子 #1）。**shearForce**(=*Vector3r::Zero()*)前一步结束后粒子 #2 所承受的剪切力（全局坐标系下，来自粒子 #1）。

tangensOfFrictionAngle(=非数)
摩擦角的正切值

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → None：根据给定的字典更新对象属性

class yade.wrapper.**WirePhys**(继承 *FrictPhys* → *NormShearPhys* → *NormPhys* → *IPhys* → 可序列化对象)

WirePM 类型的单个相互作用的表示方式，用于存储相关参数

sdL(=*0.*)用于考虑 *WireMat* 类型=2 (参见 [Thoeni2013]) 的额外线材长度，用于考虑变形。

dict(*(可序列化对象)辐角1*) →字典:
返回属性字典。

dispHierarchy(*(IPhys)辐角1*[, *(bool)名称*=*True*]) → 列表：返回分派类列表（从下往上），从类实例本身开始，最后是顶层可索引对象。如果 names 为 true（默认值），则返回类名而非数值索引。

dispIndex
返回此实例的类索引。

displForceValues(=未初始化)
定义力-位移曲线的值。

initD(=*0*。)颗粒的平衡距离。计算方式为颗粒连接时的初始颗粒间距。**isDoubleTwist**(=*false*) 如果为 true，相互作用的属性将被定义为双绞结线框。**i**

sLinked(=*false*)如果为 true，颗粒将连接并发生相互作用。根据相应的相互作用半径定义自动建立连接。线材断裂（不再发生相互作用）时，该值为 false。

isShifted(=否)
若为真， *WireMat* 类型=2 力-位移曲线将发生平移。

kn(=*0*)法向刚度
ks(=*0*)剪切刚度**l**

imitFactor(=*0*。)此值表示线框距离失效还有多远，例如，实际法向位移除以允许法向位移。

normalForce(=*Vector3r::Zero()*)
粒子 #2 承受的前一步之后的法向力（在全局坐标中，来自粒子 #1）。

plastD
前一步颗粒间距的塑性部分。

注意： 只有弹性位移是可逆的（卸载时使用弹性刚度），且不允许存在压力。假定压缩刚度为零。

---

shearForce(=*Vector3r::Zero()*)前一步之后的剪切力（以全局坐标表示），由颗粒 #2（来自粒子 #1 ）承受。**stiffnessValues**(= 未初始化)定义各种刚度的值（弹性刚度存储为 kn ）。**tangensOfFrictionAngle**(=非数)摩擦角的正切**updateAttrs**(*(*可序列化对象*)*参数*1*, *(*字典*)*辐角 *2*) → 无：根据给定字典更新对象属性

2.3.3 全局引擎

全局引擎

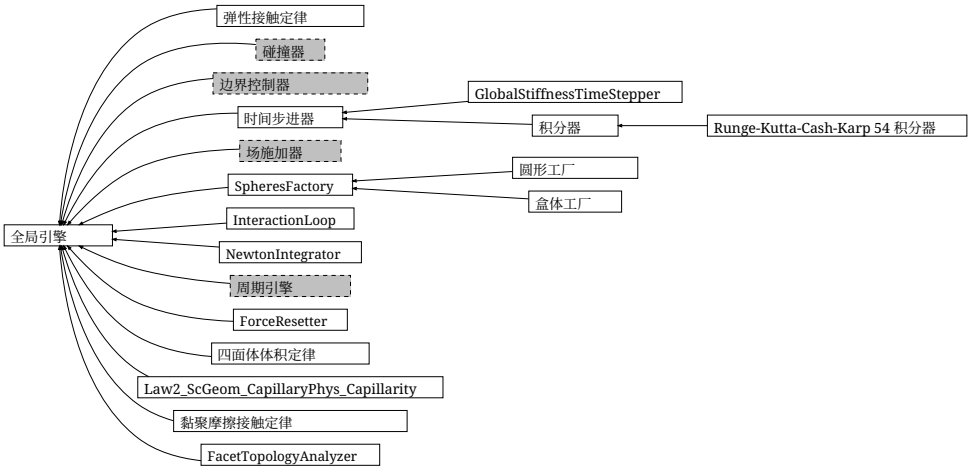


图 *25*：*GlobalEngine* 的继承图，灰色虚线框表示在各自章节中讨论的类：碰撞器、边界控制器、场施加器、周期引擎。另请参阅：盒体工厂、圆形工厂、黏聚摩擦接触定律、弹性接触定律、*FacetTopologyAnalyzer*、*ForceResetter*、*GlobalStiffnessTimeStepper*、*Integrator*、*InteractionLoop*、*Law2_ScGeom_CapillaryPhys_*毛细作用、*NewtonIntegrator*、*Runge-Kutta-Cash-Karp 54* 积分器、*SpheresFactory*、四面体体积定律、时间步进器。

class yade.wrapper.**GlobalEngine**(继承 引擎 →可序列化对象)通常会影响整个模拟的引擎（与 部分引擎相反）。**dead**(=否)若为真，此引擎将完全不会运行；可用于暂时停用引擎，并在之后的某个时间点重新启用。**dict**(*(*可序列化对象*)*辐角 *1*) →字典：返回属性字典。

execCount
: 此引擎的累计运行次数（仅在 *O.timingEnabled*==True时使用）。

execTime
: 此引擎运行所用的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

label(=未初始化)
: 此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

ompThreads(=*-1*)
引擎使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或用于将 *ParallelEngine* 与运行并行区域的引擎组合使用，从而在不同层级形成线程数各异的嵌套 OMP 循环。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → None：根据给定字典更新对象属性

class yade.wrapper.BoxFactory(继承 *SpheresFactory* → 全局引擎 → 引擎 → 可序列化对象)
由半尺寸和中心定义的 SpheresFactory 区域的盒体几何

PSDcalculateMass(=*true*)粒径分布输入采用质量（true）；否则将按颗粒数量处理。

PSDcum(=未初始化)PSD 分布，累积百分比含义 [-]

PSDsizes(=未初始化)PSD 分布，单元尺寸，直径 [米]

blockedDOFs(=*""*)被阻塞的自由度

center(=*Vector3r(NaN, NaN, NaN)*)
区域中心

color(=*Vector3r(-1, -1, -1)*)若指定，则将此颜色用于新创建的颗粒

dead(=*false*)
若为真，则此引擎将完全不运行；可用于暂时停用某个引擎并仅在之后的某个时刻重新启用它。

dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

exactDiam(=真)
若为真，则只会创建直径在 PSDsizes 中定义的颗粒。否则，直径将在 [PSDsizes[i-1]: PSDsizes[i]], 范围内随机选取；在这种情况下，PSDsizes 的长度应比 PSDcum 的长度多 1。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所花费的累计时间（纳秒）（仅在 if *O.timingEnabled*==True)。

extents(=*Vector3r(*非数，非数，非数*)*)
区域的半尺寸

goalMass(=*0*)
当前步结束时应达到的总质量。（自动更新）

ids(=未初始化)已创建刚体的标识符

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

mask(=*-1*)应用于新创建球体的组掩码

massFlowRate(=非数)
质量流率 [千克/秒]

materialId(=*-1*)
新创建球体要使用的共享材料 ID（可以为负数，表示从末尾开始计数）

maxAttempt(=*5000*)随机放置新球体的最大尝试次数。

maxMass(=*-1*)停止生成新颗粒时的最大质量，无论质量流率为何。如果 maxMass=-1，则忽略此参数。

maxParticles(=*100*)
停止生成新颗粒时的颗粒数量，无论质量流率为何。如果 maxParticles=-1，则忽略此参数。

normal(=*Vector3r(*非数，非数，非数*)*)区域几何体的方向；若未设置 normalVel，则为颗粒速度的方向。

normalVel(=*Vector3r(*非数，非数，非数*)*)
颗粒速度的方向。

numParticles(=*0*)
目前已生成的颗粒累计数量*(*自动更新*)*

ompThreads(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 （默认值），则该数量通常为 OMP_NUM_THREADS，或由 'yade -jN' 定义的数量 N（不过此行为可能取决于引擎）。此属性仅影响其代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或用于将 *ParallelEngine* 与运行并行区域的引擎组合，从而产生嵌套 OMP 循环，并使每一层使用不同的线程数。

rMax(=非数)生成球体的最大半径（均匀分布）

rMin(=非数)生成球体的最小半径（均匀分布）

silent(=*false*)
如果为 true，则不再因超出最大尝试次数而发出警告，但会禁用生成器（通过设置 mass- FlowRate =0）。

stopIfFailed(=真)
如果为真，当插入颗粒的最大尝试次数超出时，SpheresFactory 会停止（设置 massFlowRate=0）。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用并设置 *O.timingEnabled*==True。

totalMass(=*0*)
目前为止已生成的球体质量。（自动更新）

totalVolume(=*0*)
目前为止已生成的球体体积。（自动更新）

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → 无:
根据给定字典更新对象属性

vAngle(=非数)初始球体速度偏离法向量的最大角度。

vMax(=非数)生成球体的最大速度范数（均匀分布）

vMin(=非数)生成球体的最小速度范数（均匀分布）

class yade.wrapper.CircularFactory(继承 *SpheresFactory* → 全局引擎 → 引擎 →可序列化对象)

SpheresFactory 区域的圆形几何。它可以是圆盘（由半径和中心定义），也可以是圆柱体（由半径、长度和中心定义）。

PSDcalculateMass(=*true*)PSD 输入以质量为单位（true）；否则将考虑颗粒数量。

PSDcum(=未初始化)PSD 分布，累计百分比含义 [-]

PSDsizes(=未初始化)PSD 分布、单元尺寸，直径 [米]

blockedDOFs(=*""*)被阻塞的自由度

center(=*Vector3r(*非数, 非数, 非数*)*)
区域中心

color(=*Vector3r(-1, -1, -1)*)如果指定，则使用该颜色为新
创建的颗粒着色

dead(=*false*)如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并在之后某个时刻重新启用它。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

exactDiam(=是)
如果为是，则仅创建 PSD尺寸 中定义的直径颗粒。否则，将在范围 [PSD尺寸[i-1]:PSD尺寸[i]], 内随机选择直径；在这种情况下，PSD尺寸的长度应比PSDcum的长度多 1。

execCount
该引擎运行的累计次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
该引擎运行所耗费的累计时间（以纳秒计，仅在*O.timingEnabled*==True时使用）。

goalMass(=*0*)
当前步结束时应达到的总质量。（自动更新）

ids(=未初始化)已创建刚体的标
识符

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

length(=*0*)
圆柱区域的长度（默认为 0）

mask(=*-1*)应用于新创建球体的组掩码

massFlowRate(=非数)质量流率 [千克/秒]materialId(=-1)新创建球体所使用的共享材料 ID（可以为负数，以从末尾开始计数）maxAttempt(=5000)随机放置新球体的最大尝试次数。
maxMass(=-1)停止生成新颗粒的最大质量，无论质量流率如何。如果 maxMass=-1，则忽略此参数。maxParticles(=100)

无论质量流率如何，达到此颗粒数量后将停止生成新颗粒。如果 maxParticles=-1，则忽略此参数。

normal(=Vector3r(非数，非数，非数))区域几何体的方向；未设置 normalVel 时，颗粒速度的方向。

normalVel(=Vector3r(非数，非数，非数))
颗粒速度的方向。

numParticles(=0)迄今为止生成的颗粒累计数量(自动更新)

ompThreads(=-1)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_数量_THREADS，或由“yade -jN”定义的 N（不过，此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时；这会导致嵌套 OMP 循环在各层级使用不同的线程数。

rMax(=非数)生成球体的最大半径（均匀分布）

rMin(=非数)生成球体的最小半径（均匀分布）

radius(=非数)区域半径

silent(=false)若为 true，则不会因超过最大尝试次数 maxAttempt 而报错，但会禁用生成器（通过设置 mass- FlowRate=0）。

stopIfFailed(=true)若为 true，当插入颗粒的尝试次数超过最大值时，SpheresFactorystops（设置 massFlowRate=0）。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用*O.timingEnabled*==True，否则为空。

totalMass(=0)
目前已生成球体的质量。(自动更新)totalVolume(=0)目前已生成球体的体积。
(自动更新)updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无：根据给定字典更新对象属性vAngle(=非数)初始球体速度偏离法向量的最大角度。

vMax(=非数)生成球体速度范数的最大值（均匀分布）vMin(=非数)生成球体速度范数的最小值（均匀分布）

class yade.wrapper.CohesiveFrictionalContactLaw(继承全局引擎 → 引擎 → 可序列化)

[已弃用] 遍历相互作用并应用定律2_ScGeom6D_CohFrictPhys_黏聚力矩_CohFrictPhys_黏聚力矩_黏聚力矩于全部相互作用。

注意：出于性能考虑，请使用*InteractionLoop*和定律*2_ScGeom6D_CohFrictPhys_Cohesion Moment*，而不要使用此类。

always_use_moment_law(=*false*)
如果为 true，则在所有接触处使用弯曲/扭转力矩。如果为 false，则仅计算力矩 r 黏聚接触。

creep_viscosity(=否)蠕变黏度 [帕·秒/米]。可能应该移至 Ip2_CohFrictMat_CohFrictMat_-CohFrictPhys...

dead(=否)
如果为真，此引擎将完全不会运行；可用于暂时停用引擎并仅在稍后的某个时间点重新启用它。

dict((可序列化对象)辐角*1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所耗费的累计时间（以纳秒为单位，仅在 if *O.timingEnabled*==True时使用）。

label(=取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，你可以直接从 Python 中引用它。

neverErase(=否)
即使颗粒彼此分离，也保留相互作用（仅当场景中存在其他本构定律时才适用，例如 定律*2_ScGeom_CapillaryPhys_*毛细作用）

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_数量_线程，或由“yade -jN”定义的数量 N（不过此行为可能取决于引擎）。此属性只影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合时，从而形成每层线程数不同的嵌套 OMP 循环。

shear_creep(=否)
使用 黏聚摩擦接触定律::蠕变_黏度。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用并设置 *O.timingEnabled*==True。

twist_creep(=*false*)
启用扭转力矩的蠕变，使用 *CohesiveFrictionalContactLaw::creep_*黏度。

updateAttrs((可序列化对象) 参数*1*, (字典) 辐角*2*) → 无: 根据给定字典更新对象属性

class yade.wrapper.ElasticContactLaw(继承 全局引擎 → 引擎 → 可序列化对象)[已弃用]循环遍历相互作用并应用 *定律2_ScGeom_FrictPhys_CundallStrack* 于全部相互作用。

> **注意：** 出于性能考虑，请使用 *InteractionLoop*和 *Law2_ScGeom_FrictPhys_CundallStrack* 代替此类。

---

dead(=*false*)
如果为 true，此引擎将完全不运行；可用于暂时停用某个引擎，并在之后的某个时刻重新启用它。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所用的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

label(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

neverErase(=否)
即使颗粒彼此分离，也保留相互作用（仅当场景中存在其他本构定律时才适用，例如 *定律2_ScGeom_CapillaryPhys_毛细作用*）

ompThreads(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或者由“yade -jN”定义的 N（不过，这种行为可能取决于引擎）。此属性只会影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时；这样会形成嵌套 OMP 循环，并且每一层的线程数不同。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用并设置 *O.timingEnabled*==True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) →
None：从给定字典更新对象属性

class yade.wrapper.FacetTopologyAnalyzer(继承自 全局引擎 → 引擎 → 可序列化对象)用于填充面片邻接几何数据的初始化器。

识别公共顶点和公共边，并将面片面之间的相互夹角写入 Facet 实例。如果面片彼此之间不发生移动，则只需在开始时执行一次。

commonEdgesFound(=*0*)
上次运行时识别出的公共边数量。*(*自动更新*)*

commonVerticesFound(=*0*)
上次运行时识别出的公共顶点数量。*(*自动更新*)*

dead(=否)
如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并在之后的某个时间点重新启用。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

execCount
: 此引擎的累计运行次数（仅在*O.timingEnabled*==True已启用时使用）。

execTime
: 该引擎运行所用的累计时间（纳秒；仅在*O.timingEnabled*==True时使用）。

label(=未初始化)
: 此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

ompThreads(=*-1*)
: 引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由'yade -jN'定义的 N 值（不过，此行为可能取决于引擎）。此属性仅影响代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时，这会产生每一层线程数不同的嵌套 OMP 循环。

projectionAxis(=*Vector3r::UnitX()*)用于初始顶点排序
的轴

relTolerance(=*1e-4*)
: "相同"顶点之间的最大距离，相对于最小面片尺寸

timingDeltas
: 引擎本身的详细计时信息。除非在源代码中启用*O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)arg1*，*(字典)arg2*) →
None：从给定字典更新对象属性

class yade.wrapper.ForceResetter(继承自全局引擎 → 引擎 → 可序列化对象)
: 重置存储在 Scene::forces 中的所有力（0.forces 在 Python 中）。通常，这是每一步运行的第一个引擎。此外，如果启用了能量追踪，还会重置那些需要重置的能量。

dead(=*false*)
: 如果为 true，此引擎将完全不运行；可用于暂时停用某个引擎，并仅在之后的某个时间点重新启用它。

dict(*(可序列化对象)辐角 1*) →字典：
返回属性字典。

execCount
: 此引擎运行的累计次数（仅在*O.timingEnabled*==True时使用）。

execTime
: 此引擎运行所花费的累计时间（以纳秒为单位）（仅在*O.timingEnabled*==True时使用）。

label(=未初始化)
: 此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

ompThreads(=*-1*)
: 引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由"yade -jN"定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在使用 *ParallelEngine* 与运行并行区域的引擎组合时，此时会形成嵌套 OMP 循环，且每一层的线程数不同。

timingDeltas
: 引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs((可序列化对象)参数1, (字典)辐角2) → 无:
根据给定字典更新对象属性

class yade.wrapper.GlobalStiffnessTimeStepper(继承 时间步进器 → 全局引擎 →引擎 → 可序列化对象)

一个将时间步长设为问题中最小特征周期的一定比例的引擎。其推导过程详见 离散元法公式 章节。viscEl 选项可采用类似方式，为黏弹性接触定律 定律2_ScGeom_ViscElPhys_Basic 评估时间步长，详情请参阅 *GlobalStiffnessTimestepper::viscEl*。

active(=真)该引擎是否处于
活动状态?

dead(=假)
如果为真，该引擎将完全不运行；可用来暂时停用引擎，并在稍后的某个时间点重新启用。

defaultDt(=-1)
用作时间步长的初始值（在没有接触的最初几步中特别有用）。如果为负值，则由 *utils.PWaveTimeStep* * *GlobalStiffnessTimeStep- per::timestepSafetyCoefficient*确定

densityScaling(=*false*)
(自动更新) 如果不打算手动修改某些刚体的缩放因子，就不要修改此值。在大多数情况下，只需设置 *NewtonIntegrator::densityScaling* 即可让此值自动调整。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所用的纳秒数累计时间（仅在*O.timingEnabled*==True时使用）。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

maxDt(=*Mathr::MAX*_实数)若为正数，则用作时间步长的最大值，不论计算所得的值是多少

ompThreads(=-1)
引擎要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_线程数_，或由'yade -jN'定义的 N（不过此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要用于实验，或在 *ParallelEngine* 与运行并行区域的引擎组合使用时，从而形成每一层线程数不同的嵌套 OMP 循环。

previousDt(=*Mathr::MAX_REAL*)最后计
算出的时间步长 （自动更新）

targetDt(=1)
如果 *NewtonIntegrator::densityScaling* 处于活动状态，此值将用作模拟时间步长，缩放也会将此 dt 值用作目标值。targetDt 的值是任意的，通常不会影响结果。不过，如果某些刚体具有指定速度，例如，它们在每一步中的移动量会因该值而有所不同。

timeStepUpdateInterval(=1)
时间步长更新间隔

timestepSafetyCoefficient(=0.8)
最小特征周期与最终指定的时间步长之间的安全系数（小于 1）

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用并且 *O.timingEnabled*==True。

updateAttrs(*(可序列化对象)*参数*1*，*(字典)*辐角*2*) → 无:
根据给定字典更新对象属性

viscEl(=假)
与 *ViscElPhys*一起使用。如果为 True，则分别计算问题中的最小特征周期：一方面仅考虑弹性贡献（仅弹簧），另一方面仅考虑黏性贡献（仅阻尼器）。然后取两者中的最小值，并使用安全系数*GlobalStiffnessTimestepper::timestepSafetyCoefficient* 来考虑两种贡献之间可能存在的耦合。

class yade.wrapper.Integrator(继承自 时间步进器 → 全局引擎 → 引擎 →可序列化对象)

积分引擎接口。

active(=是)
引擎处于活动状态吗?

dead(=否)如果为真，此引擎将完全不会运行；可用于暂时停用引擎，并在稍后将其重新启用。

dict(*(可序列化对象)*辐角*1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所花费的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

integrationsteps(=未初始化)
所有积分步数均计为成功的子步

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

maxVelocitySq(=非数)
存储最大速度的平方，仅供参考；每一步都会重新计算。（自动更新）

ompThreads(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_数量_THREADS，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性只会影响代码中包含 openMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与 *ParallelEngine* 结合使用会运行并行区域的引擎，从而形成每一层线程数各不相同的嵌套 OMP 循环。

slaves
用于计算作用于颗粒的力的引擎列表的列表；为获得状态的导数，其中的引擎将按顺序运行。

timeStepUpdateInterval(=*1*)
时间步长更新间隔

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用，否则为空，且 *O.timingEnabled*==True。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无：
根据给定字典更新对象属性

class yade.wrapper.InteractionLoop(继承 全局引擎 → 引擎 → 可序列化对象)用于在每一步处理相互作用循环的统一分派器，旨在提升并行性能。

特殊构造函数

由 3 个列表构造，分别包含 *Ig2*、*Ip2*、*Law2* 函子；这些函子将传递给内部的分派器，您可以分别通过 *geomDispatcher*、*physDispatcher*、*lawDispatcher* 获取这些分派器。

callbacks(=未初始化)
回调 将在每个相互作用时调用（若已激活）。

dead(=*false*)若为 true，则此引擎完全不会运行；可用于暂时停用引擎，并仅在稍后重新启用。

dict((*Serializable*)*arg1*) →字典：返
回属性字典。

execCount
此引擎运行的累计次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所耗费的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

geomDispatcher(=*new IGeomDispatcher*)
IGeomDispatcher 用于分派的对象。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

lawDispatcher(=新的 *LawDispatcher*)
LawDispatcher 对象，用于分派。

loopOnSortedInteractions(=否)
如果为 true，主相互作用循环将按已排序的相互作用列表执行。这种方式很慢，但在调试 Yade 的并行实现时，可用于解决浮点力加法不可重复的问题。

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的 N（不过，具体行为可能取决于引擎）。此属性只会影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合时使用；这样会产生嵌套 OMP 循环，且每一层的线程数不同。

physDispatcher(=新的 *IPhysDispatcher*)
IPhysDispatcher 用于分派的对象。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则内容为空。

updateAttrs((可序列化对象)*arg1*，(字典)*arg2*) →
None：根据给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_CapillaryPhys_Capillarity(继承 全局引擎 → 引擎 →可序列化对象)
该定律可用于考虑因存在颗粒间液桥（弯月面）而在球体之间产生的毛细力/效应。

控制参数为毛细压力（或吸力）Uc = Ugas - Uliquid。液桥属性（体积 V、在相互作用颗粒上的延伸量 delta1 和 delta2）根据定义的毛细压力和相互作用几何体（球体半径和颗粒间距）计算得出。

引用：英文 [Scholtes2009b]；更为详细，但为法文 [Scholtes2009d]。

该定律需要 ASCII 文件 M(r=i)，其中 i=R1/R2，才能运行（参见 https://yade-dem.org/wiki/CapillaryTriaxialTest）。这些 ASCII 文件包含在假设润湿角为零的情况下，对不同相互作用几何体配置求解拉普拉斯-杨氏方程所得的一组结果。

为了使远距离球体之间产生毛细力，有必要使用 *Bo1_Sphere_Aabb::AABB* 扩大因子 扩大包围盒，并让 Ig2 通过 *interactionDetectionFactor*定义远距离相互作用。还必须通过本构定律禁用相互作用移除（*Law2*）。目前已支持的定律组合仅有毛细定律+ *Law2_ScGeom_FrictPhys_CundallStrack* 和毛细定律 + *Law2_ScGeom_MindlinPhys_*- 明德林 （以及其他赫兹-明德林变体）。

请参阅 CapillaryPhys-example.py，了解示例脚本。

`binaryFusion`(=是)
: 如果为真，一旦检测到至少 1 个重叠（弯月面融合），毛细力就会被设为零。否则 毛细力 = 毛细力 / (融合数 + 1)

`capillaryPressure`(=*0*。)
: 毛细压力 Uc 的值定义为 Uc=Ugas-Uliquid

`createDistantMeniscii`(=否)
: 在远距离球体之间生成弯月面？否则仅维护现有弯月面。对于润湿路径建模，此标志应始终为否。对于干燥路径，应在一个步骤（初始化）中将其设为是，然后设为否，正如 [Scholtes2009c]

`dead`(=否)
: 如果为真，此引擎将完全不运行；可用于将引擎暂时停用，并仅在稍后某个时间点重新启用。

`dict`(（可序列化对象）辐角 *1*) →字典：返回属性字典。

`execCount`
: 此引擎运行的累计次数（仅在 *O.timingEnabled*==`True` 时使用）。

`execTime`
: 此引擎运行所用的累计时间（以纳秒计，仅在*O.timingEnabled*==`True`时使用）。

`fusionDetection`(=假)
: 如果为 true，则会检查潜在的弯月面重叠，并计算融合数（针对每个毛细管相互作用），同时根据毛细力进行相应调整二元融合

`label`(=未初始化)
: 此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

`ompThreads`(=*-1*)
: 引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_数量_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如*InteractionLoop*）。此属性主要适用于实验，或在将*ParallelEngine* 与运行并行区域的引擎结合使用时，此时会形成嵌套 OMP 循环，并且每一层的线程数各不相同。

`suffCapFiles`(=“”)
: 毛细管文件后缀：M(r=X)毛细管文件后缀

surfaceTension(=*0.073*)
所考虑的表面张力值

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.NewtonIntegrator(继承自全局引擎 →引擎 →可序列化对象)集成牛顿运动方程的引擎。

dampGravity(=*true*)
默认情况下，数值阻尼会作用于全部力，包括重力。如果将此选项设为 false，则基于 NewtonIntegrator.gravity 计算的重力将不计入阻尼计算。此选项不影响由 GravityEngine 添加的重力。

damping(=*0.2*)
Cundall 非黏性阻尼的阻尼系数（参见 数值阻尼 和[Chareyre2005]）

dead(=*false*)
如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并在之后的某个时间点重新启用它。

densityScaling
如果为 True，则会应用密度缩放 [Pfc3dManual30]，使所有刚体的临界时间步长等于 *GlobalStiffnessTimeStepper::targetDt*。从动态角度来看，此选项会使模拟失真，但可能加快准静态模拟。在少数情况下，可能不希望为每个刚体自动设置缩放因子（时间步进器会自动执行此操作）。此时，可将 *GlobalStiffnessTimeStepper.densityScaling* 设为 False。

dict(*(*可序列化对象*)*辐角*1*) →字典:
返回属性字典。

exactAsphericalRot(=真)
仅对非球形体启用更精确的刚体旋转积分器，采用以下公式 m [Allen1989],第 89 页。

execCount
此引擎运行的累计次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行耗时的累计纳秒数（仅在*O.timingEnabled*==True时使用）。

gravity(=*Vector3r::Zero()*)重力加速度（实际上替代了 GravityEngine）。

kinSplit(=*false*)是否分别跟踪平动动能和转动动能。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

mask(=*-1*)
如果定义了掩码，且掩码与刚体的组掩码按位与的结果为 0，则刚体不会移动/旋转。计算速度和加速度时不会考虑此参数。

maxVelocitySq(=*0*)
存储最大位移，并据此触发碰撞检测。（自动更新）

ompThreads(=*-1*)
引擎中要使用的线程数。如果为 ompThreads<0 （默认值），线程数将为 e

通常为 OMP_NUM_THREADS，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性只影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或用于将 *ParallelEngine* 与运行并行区域的引擎组合，从而形成嵌套 OMP 循环，且每一层的线程数不同。

prevVelGrad(=*Matrix3r::Zero()*)保存先前的速度梯度（*Cell::velGrad*），以跟踪加速度。（自动更新）

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用并设置*O.timingEnabled*==True，否则内容为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) →
None：根据给定字典更新对象属性

warnNoForceReset(=*true*)如果此步中力未由*ForceResetter*重置，则发出警告；这通常意味着 *ForceResetter* 被意外遗漏，只有在有充分理由时才应禁用此警告。

class yade.wrapper.**RungeKuttaCashKarp54Integrator**(继承 积分器 → 时间步进器 →全局引擎 → 引擎 → 可序列化对象)

Runge-Kutta-Cash-Karp 54 积分器引擎。

__init__(*(object)arg1*) →无对象__初始化__(tuple args, dict kwds)__**初始化**__(**(object)arg1, (list)arg2) -> 对象：** 由（可能嵌套的）从属引擎列表构造。**a_dxdt**(=*1。 0*) **a_x**(=*1。 0*)**abs_err**(=*1e-6*)相对积分容差**active**(=*true*)引擎是否处于活动状态？**dead**(=*false*)若为 true，此引擎将完全不运行；可用于暂时停用引擎，并在之后某个时间点重新启用。

dict(*(Serializable)arg1*) →字典：返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所用的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

integrationsteps(=未初始化)所有积分步数均计为成功的子步

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，才能在 Python 中直接引用它。

maxVelocitySq(=非数)存储最大速度的平方，仅供参考；每一步都会重新计算。（自动更新）
ompThreads(=*-1*)引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数将为

通常为 OMP_线程数_，或由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时；这样会产生嵌套 OMP 循环，并且每一层的线程数不同。

rel_err(=*1e-6*)绝对积分容差

slaves
用于计算作用于颗粒上的力的引擎列表；为获得状态的导数，内部的引擎将按顺序运行。

stepsize(=*1e-6*)
这对于自适应积分并不重要，但对于观测器在积分后设置所找到的状态很重要

timeStepUpdateInterval(=*1*)
时间步长更新间隔

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用并设置 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)*参数*1*，*(字典)*辐角*2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.**SpheresFactory**(继承 全局引擎 → 引擎 → 可序列化对象)
用于根据质量流率、粒径分布等喷射球体的引擎。颗粒的初速度由 最小速度、最大速度给出，质量流率 决定每步生成的颗粒数量。当达到目标质量 或达到正值最大粒子数 时，引擎将不再生成颗粒。区域的几何体应在派生引擎中通过重写 SpheresFactory::pickRandomPosition() 来定义。

此引擎的示例脚本位于 scripts/spheresFactory.py。

PSDcalculateMass(=真)PSD 输入以质量表示（真），否则将考虑颗粒数量。

PSDcum(=未初始化)PSD 分布，累积百分比含义 [-]

PSDsizes(=未初始化)PSD 分布、单元尺寸、直径 [米]

blockedDOFs(=*""*)被阻塞的自由度

color(=*Vector3r(-1，-1，-1)*)如果指定，则使用此颜色作
为新创建颗粒的颜色

dead(=*false*)
如果为 true，则此引擎将完全不运行；可用于暂时停用引擎，并在之后某个时间点重新启用它。

dict(*(可序列化对象)*辐角*1*) →字典:
返回属性字典。

exactDiam(=真)
如果为真，则只创建 PSD尺寸 中定义了直径的颗粒。否则，将在范围 [PSD尺寸[i-1]:PSD尺寸[i]], 内随机选择直径。在这种情况下，PSD尺寸的长度应比累积粒径分布的长度多 1。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所用的累计时间（以纳秒为单位，仅在*O.timingEnabled*==True时使用）。

goalMass(=*0*)
当前步结束时应达到的总质量。（自动更新）

ids(=未初始化)已创建刚体的标识符

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

mask(=*-1*)应用于新创建球体的组掩码

massFlowRate(=非数)质量流率 [千克/秒]**materialId**(=*-1*)新创建球体使用的共享材料 ID（可以为负数，以从末尾开始计数）**maxAttempt**(=*5000*)随机放置新球体的最大尝试次数。

maxMass(=*-1*)停止生成新颗粒的最大质量，不受 massFlowRate 影响。如果 maxMass=-1，则忽略此参数。

maxParticles(=*100*)
停止生成新颗粒的颗粒数量，不受 massFlowRate 影响。如果 maxParticles=-1，则忽略此参数。

normal(=*Vector3r(NaN, NaN, NaN)*)区域几何体的方向；如果未设置 normalVel，则为颗粒速度的方向。

normalVel(=*Vector3r(NaN, NaN, NaN)*)颗粒速度的方向。

numParticles(=*0*)
迄今产生的颗粒累计数量 *(自动更新)*

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），则线程数通常为 OMP_数量_线程数，或由“yade -jN”定义的数量 N（不过此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要用于实验，或用于将 *ParallelEngine* 与运行并行区域的引擎组合，从而在不同层级产生线程数不同的嵌套 OMP 循环。

rMax(=非数)生成球体的最大半径（均匀分布）**rMin**(=非数)生成球体的最小半径（均匀分布）**silent**(=否)

如果为 true，则不会因超过最大尝试次数而报错，但会禁用生成器（通过设置 massFlowRate=0）。

stopIfFailed(=是)
如果为真，当插入颗粒的尝试次数超过最大值时，SpheresFactorystops（设置 massFlowRate=0）。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

totalMass(=*0*)
迄今生成的球体质量。（自动更新）

totalVolume(=*0*)迄今生成的球体体积。（自动更新）

updateAttrs（可序列化对象）参数*1*，（字典）辐角
2) → 无：根据给定字典更新对象属性

vAngle(=非数)初始球体速度偏离法向量的最大角度。

vMax(=非数)生成球体的最大速度范数（均匀分布）

vMin(=非数)生成球体的最小速度范数（均匀分布）

class yade.wrapper.TetraVolumetricLaw(继承 全局引擎 → 引擎 → 可序列化对象)根据穿透构型 gi 计算相互作用中 2 个 四面体 的物理响应 ven
作者：*TTetraGeom*。

dead(=否)如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并在之后的某个时刻重新启用。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所用的累计纳秒数（仅在*O.timingEnabled*==True时使用）。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

ompThreads(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与*ParallelEngine* 结合使用包含并行区域的引擎，从而形成每层线程数不同的嵌套 OMP 循环。

timingDeltas
引擎本身内部计时的详细信息。除非在源代码中启用且 *O.timingEnabled*==True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.TimeStepper(继承 全局引擎 →引擎 →可序列化对象)定义时间步长的引擎（基础类）

active(=*true*)引擎是否处于活
动状态?

`dead`(=*false*)
如果为 true，此引擎将完全不会运行；可用于暂时停用引擎，并仅在之后的某个时刻重新启用它。

`dict`(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

`execCount`
此引擎的累计运行次数（仅在 *O.timingEnabled*==`True` 时使用）。

`execTime`
该引擎运行所需的纳秒级累计时间（仅在*O.timingEnabled*==`True`时使用）。

`label`(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，你可以直接从 Python 中引用它。

`ompThreads`(=*-1*)
引擎中要使用的线程数。如果为 ompThreads<0 （默认值），则该数量通常为 OMP_线程数_，或者为由“yade -jN”定义的线程数 N（不过具体行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或用于将 *ParallelEngine* 与运行并行区域的引擎组合使用，从而在不同层级产生线程数不同的嵌套 OMP 循环。

`timeStepUpdateInterval`(=*1*)
时间步长更新间隔

`timingDeltas`
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==`True`，否则为空。

`updateAttrs`*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) →
None：根据给定字典更新对象属性

周期引擎

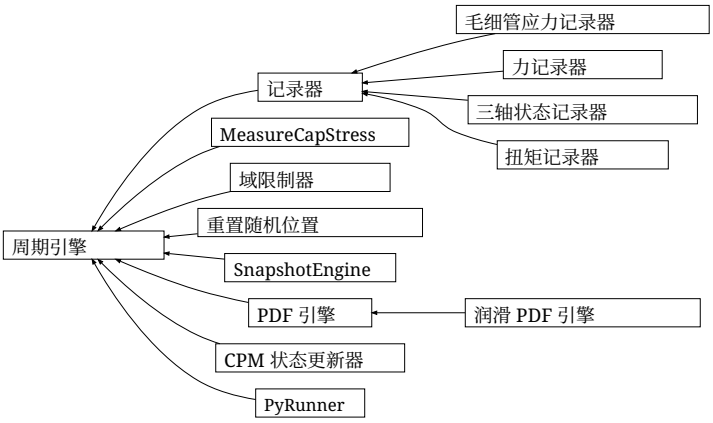


图 26：周期引擎的继承图。另请参阅：毛细管应力记录器、*CPM* 状态更新器、域限制器、力记录器、润滑 *PDF* 引擎、*MeasureCapStress*、*PDF* 引擎、*PyRunner*、记录器、重置随机位置、*SnapshotEngine*、扭矩记录器、三轴状态记录器。

`class yade.wrapper.PeriodicEngine`(继承 全局引擎 → 引擎 → 可序列化对象)通过将任一条件（virtPeriod、realPeriod、iterPeriod）设为正值，按给定的固定周期性实时（=墙上时钟时间、计算时间）、虚拟时间（模拟时间）、迭代次数）运行 Engine::action。默认情况下，它们均为负值（非活动状态）。

可以通过设置 nDo>0来限制此引擎的激活次数。如果已达到激活次数上限，即使活动周期已经过去，也不会调用任何操作。

如果设置了 initRun（默认为 false），引擎将在首次调用时运行；否则，它只会从该时刻开始计数周期（realLast 等间隔变量），但不会实际运行，并且只有自初始运行以来经过一个周期后才会运行。

不应直接使用此类；应派生出你希望周期性运行的引擎。

派生引擎应重写 Engine::action()，该函数将被周期性调用。如果派生引擎还重写了 Engine::isActivated，则也应考虑 PeriodicEngine::isActivated 的返回值，否则周期性功能将无法正常工作。

使用 *PyRunner*的示例。PyRunner 派生自周期引擎；在 Python 脚本中可能会遇到这种用法：

```
PyRunner(realPeriod=5,iterPeriod=10000,command='print O.iter')
```

自上次运行以来，每经过 10000 次迭代或每经过 5 秒墙钟时间（以先到者为准），就会打印迭代次数。

dead(=否)
若为 true，则此引擎完全不会运行；可用于暂时停用某个引擎，并仅在之后的某个时刻重新启用它。

dict(（可序列化对象）辐角 *1*) →字典：返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所用的累计时间（以纳秒为单位；仅在*O.timingEnabled*==True 时使用）。

firstIterRun(=*0*)
设置引擎首次执行时的步数（默认停用）。

initRun(=*false*)
首次调用时也运行。

iterLast(=*0*)记录上次运行的步数（自动更新）。

iterPeriod(=*0*, 停用)基于步数的周期性判据（若 $<= 0$ 则停用）

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

nDo(=*-1*，停用)
将执行次数限制为该数量（若为负数则停用）

nDone(=*0*)
跟踪执行次数（累计）（自动更新）。

ompThreads(=*-1*)
引擎中使用的线程数。如果为 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS 或由“yade -jN”定义的 N（不过，此行为可能因引擎而异）。此属性只会影响代码中包含

OpenMP 并行区域（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时；这会产生嵌套 OMP 循环，且每一层的线程数不同。

realLast(=*0*)
记录上次运行的实际时间 （自动更新）。

realPeriod(=*0*, 停用)使用实数（墙上时钟、计算、人类）时间（秒）的周期性判据（当 <=0 时停用）

timingDeltas
引擎本身内部的详细计时信息。除非在源码中启用，否则为空 e
代码和 *O.timingEnabled*==True。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) →
None：根据给定字典更新对象属性

virtLast(=*0*)记录上次运行的虚拟时间 *(*自动更新*)*。

virtPeriod(=*0*, 停用)采用虚拟（模拟）时间的周期性判据（当 <= 0 时停用）

class yade.wrapper.**CapillaryStressRecorder**(继承 记录器 → 周期引擎 → *Glob-alEngine* → 引擎 → 可序列化对象)
记录三轴压缩试样中的毛细液桥信息。毛细应力采用经典符号约定（拉伸为正）。-> 新形式体系需要测试！！！

addIterNum(=*false*)
创建文件时，会在文件名中添加迭代次数。可用于每次调用时创建新文件（默认为 false）

dead(=*false*)若为 true，则此引擎完全不会运行；可用于暂时停用引擎，并仅在稍后重新启用。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所用的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

file(=未初始化)要保存到的文件名；不能为空。

firstIterRun(=*0*)
设置引擎首次执行的步数（默认停用）。

initRun(=*false*)首次调用时也运行。

iterLast(=*0*)
记录上次运行的步数 （自动更新）。

iterPeriod(=*0*, 停用)
基于步数的周期性判据（若 <= 0 则停用）

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

nDo(=*-1*，已停用)
: 将执行次数限制为此数值（若为负数则停用）

nDone(=*0*)
: 跟踪执行次数（累计）（自动更新）。

ompThreads(=*-1*)
: 引擎要使用的线程数。如果 ompThreads<0 （默认值），则线程数通常为 OMP_NUM_线程数，或由“yade -jN”定义的 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时；这会产生不同层级线程数不同的嵌套 OMP 循环。

realLast(=*0*)
: 跟踪上次运行的实际时间（自动更新）。

realPeriod(=*0*, 停用)
: 以实数（墙上时钟、计算、人）秒为单位的周期性判据（若 <=0 则停用）

timingDeltas
: 引擎本身内部的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True。

truncate(=*false*)
: 打开时是否删除当前文件内容（如果有；默认为 false）

updateAttrs(*(*可序列化对象*)arg1*，*(*字典*)arg2*) →
None：从给定字典更新对象属性

virtLast(=*0*)
: 记录上次运行的虚拟时间 （自动更新）。

virtPeriod(=*0*, 已停用)使用虚拟（模拟）时间的周期性判据（当 <= 0 时停用）

class yade.wrapper.CpmStateUpdater(继承 周期引擎 → 全局引擎 → 引擎 →可序列化对象)

更新 *CpmState* 刚体的状态，依据 *CpmPhys* 中与此刚体相互作用的状态变量。具体而言，会更新刚体的颜色以及 *CpmState::normDmg*，其取值取决于相互作用的平均损伤 以及已完全断裂并消失的相互作用数量。此引擎包含自己的循环（更准确地说是两个循环），遍历所有刚体；如有需要，应定期运行以在模拟过程中更新颜色。

avgRelResidual(=非数)
: 上次运行时的平均残余强度。

dead(=否)
: 若为真，此引擎将完全不运行；可用于暂时停用某个引擎，并仅在稍后某个时间点重新启用它。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

execCount
: 此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
: 此引擎运行所用的累计时间（以纳秒为单位；仅在*O.timingEnabled*==True时使用）。

firstIterRun(=*0*)
: 设置引擎应首次执行的步数（默认禁用）。

`initRun`(=*false*)在我们首次被调用时也运行。

`iterLast`(=*0*)记录上次运行的步数 *(自动更新)*。

`iterPeriod`(=*0*, 停用)使用步数的周期性判据（若 <= 0 则停用）

`label`(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

`maxOmega`(=非数)上次运行时全局最大的损伤参数。

`nDo`(=*-1*，停用)将执行次数限制为此数值（若为负数则停用）

`nDone`(=*0*)跟踪执行次数（累计）（自动更新）。

`ompThreads`(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与 *ParallelEngine* 结合使用包含并行区域的引擎，从而形成嵌套 OMP 循环，且每一层的线程数不同。

`realLast`(=*0*)
记录上次运行的实际时间（自动更新）。

`realPeriod`(=*0*, 已停用)以秒为单位、基于实际时间（墙上时钟、计算、人类时间）的周期性判据（若为 <=0 则停用）

`timingDeltas`
引擎本身内部的详细计时信息。除非在源代码中启用且 *O.timingEnabled*==`True`，否则为空。

`updateAttrs`(*(可序列化对象)参数1*, *(字典)辐角2*) → 无:
根据给定的字典更新对象属性

`virtLast`(=*0*)记录上次运行的虚拟时间*(自动更新)*。

`virtPeriod`(=*0*, 停用)使用虚拟（模拟）时间的周期性判据（当 <= 0 时停用）

`class yade.wrapper.DomainLimiter`(继承 周期引擎 → 全局引擎 → 引擎 → 可序列化对象)删除位于以下轴对齐包围盒之外的颗粒：*lo* 和 *hi*。

`dead`(=假)如果为真，此引擎将完全不运行；可用于暂时停用引擎，并仅在稍后的某个时间点将其重新启用。

`dict`(*(可序列化对象)辐角1*) →字典:
返回属性字典。

`execCount`
此引擎运行的累计次数（仅在*O.timingEnabled*==`True`时使用）。

`execTime`
此引擎运行所用的累计纳秒数（仅在*O.timingEnabled*==`True`时使用）。

firstIterRun(=*0*)
设置步数，指定引擎首次执行的步（默认禁用）。

hi(=*Vector3r(0,0, 0)*)域的上角点。

initRun(=*false*)在首次调用时也运行。

iterLast(=*0*)记录上次运行的步数*(*自动更新*)*。

iterPeriod(=*0*, 已停用)基于步数的周期性判据（当 <= 0 时停用）

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

lo(=*Vector3r(0, 0, 0)*)域的下角点。

mDeleted(=*0*)已删除颗粒的质量。

mask(=*-1*)若定义了掩码，则仅删除具有相应组掩码的颗粒。

nDeleted(=*0*)
已删除颗粒的累计数量。

nDo(=*-1*，已停用)将执行次数限制为此数值（若为负数则停用）

nDone(=*0*)跟踪执行次数（累计）（自动更新）。

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或者由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性只会影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时，此时会产生嵌套 OMP 循环，且每一层的线程数不同。

realLast(=*0*)记录上次运行的实际时间（自动更新）。

realPeriod(=*0*, 停用)以秒为单位使用实际时间（墙上时钟、计算、人为时间）的周期性判据（若 <=0 则停用）

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用且 *O.timingEnabled*==True，否则为空。

updateAttrs(*(*可序列化对象*)param1*，*(*字典*)arg2*) →
None：根据给定字典更新对象属性

vDeleted(=*0*)已删除球体的体积（不计团簇；若计入，请检查 *mDeleted*）
virtLast(=*0*)记录上次运行的虚拟时间 *(*自动更新*)*。**virtPeriod**(=*0*, 已停用)P

eriodicity criteri由我们负责ing virtual (simulation) time (deactivated if <= 0)

class yade.wrapper.ForceRecorder(继承自 记录器 → 周期引擎 → 全局引擎 →引擎 → 可序列化对象)

引擎保存作用于刚体的合力，刚体由标识符列出。例如，这可用于定义 _推土机_ 工作时受到的力。

addIterNum(=否)
创建文件时，会将迭代次数添加到文件名中。此选项适用于每次调用时创建新文件（默认值为 false）

dead(=否)如果为 true，此引擎将完全停止运行；可用此选项暂时停用引擎，并在之后的某个时刻重新启用。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

execCount
引擎运行的累计次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所花费的累计时间（以纳秒为单位，仅在*O.timingEnabled*==True时使用）。

file(=未初始化)要保存到的文件名；不能为空。

firstIterRun(=*0*)
设置步数，以确定引擎首次执行的时机（默认禁用）。

ids(=未初始化)将测量其状态的刚体列表

initRun(=否)首次调用时也运行。

iterLast(=*0*)跟踪上次运行的步数 （自动更新）。

iterPeriod(=*0*, 停用)基于步数的周期性判据（如果 <= 0 则停用）

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

nDo(=*-1*，停用)将执行次数限制为此数值（若为负数则停用）

nDone(=*0*)
跟踪执行次数（累计）（自动更新）。

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_数量_线程数，或由'yade -jN'指定的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时，此时会形成嵌套 OMP 循环，且各层使用的线程数不同。

realLast(=*0*)
记录上次运行的实际时间 （自动更新）。

realPeriod(=*0*, 停用)以实数（墙上时钟、计算、人）秒为单位的周期性判据（当 <=0 时停用）

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

totalForce(=*Vector3r::Zero()*)由函数返回的合力。

truncate(=*false*)打开时是否删除当前文件内容（如果有），默认为 false

updateAttrs(*(Serializable)arg1*, *(dict)arg2*) → 无：从
给定字典更新对象属性

virtLast(=*0*)记录上次运行的虚拟时间 （自动更新）。

virtPeriod(=*0*, 停用)使用虚拟（模拟）时间的周期性判据（如果 <= 0，则停用）

class yade.wrapper.LubricationPDFEngine(继承 *PDF 引擎* → 周期引擎 → 全局引擎 → 引擎 → 可序列化对象)

润滑定律的 *PDF 引擎* 实现

dead(=否)如果为真，此引擎将完全不会运行；可用于暂时停用引擎，并在之后的某个时间点重新启用。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所用时间的累计纳秒数（仅在*O.timingEnabled*==True时使用）。

filename(=*"PDF.txt"*)文件名

firstIterRun(=*0*)设置引擎首次执行的步数（默认停用）。

initRun(=*false*)在首次调用时也运行。

iterLast(=*0*)记录上次运行的步数 （自动更新）。

iterPeriod(=*0*, 停用)使用步数的周期性判据（若 <= 0 则停用）

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

nDo(=*-1*，停用)将执行次数限制为此数值（若为负数则停用）

nDone(=*0*)执行次数（累计）跟踪 （自动更新）。
numDiscretizeAnglePhi(=*20*)φ角扇区数量
numDiscretizeAngleTheta(=*20*)θ角扇区数量

ompThreads(=-1)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由 'yade -jN' 定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与 *ParallelEngine* 结合使用会运行并行区域的引擎，从而产生嵌套 OMP 循环，且每一层的线程数不同。

realLast(=0)
记录上次运行的实际时间（自动更新）。

realPeriod(=0, 已停用)
以秒为单位的周期性判据，使用实际时间（墙上时钟、计算时间、人类时间）（如果 <=0，则停用）

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用，否则为空*O.timingEnabled*==True。

updateAttrs(*(*可序列化对象*)*参数*1*, *(*字典*)*辐角*2*) → None：根据给定字典更新对象属性

virtLast(=0)
记录上次运行的虚拟时间 *(*自动更新*)*。

virtPeriod(=0, 停用)使用虚拟（模拟）时间的周期性判据（当 <= 0 时停用）

warnedOnce(=*false*)用于一次性警告。可能会触发有用的警告

class yade.wrapper.MeasureCapStress(继承 周期引擎 → 全局引擎 → 引擎 →可序列化对象)

用于提供毛细应力张量（非饱和条件，即三相条件下，流体混合物对总应力的贡献）这一后处理引擎，其表达式依据 μUNSAT 表达式详见[Duriez2017c]。尽管该表达式与 utils.getCapillaryStress 的表达式性质不同（此处考虑分布积分，而后者考虑合毛细力），两者等价 [Duriez2016b], [Duriez2017], [Duriez2017c]。表达式中使用的代表性体积单元体积 V 会自动测量：周期性条件下取自单元，否则使用 utils.aabbExtrema 函数。

capillaryPressure(=0)
毛细压力 u_c，定义为等于定律*2_ScGeom_CapillaryPhys_Capillar-ity.capillaryPressure*。

dead(=否)
如果为 true，此引擎将完全不会运行；可用于暂时停用引擎，并在之后的某个时间点重新启用它。

debug(=0)
输出一些调试消息。

dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

execCount
此引擎运行的累计次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所用的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

firstIterRun(=0)
设置引擎首次执行时对应的步数（默认禁用）。

`initRun`(=*false*)首次调用时也运行。

`iterLast`(=*0*)
记录上次运行的步数 *(自动更新)*。

`iterPeriod`(=*0*, 已停用)
基于步数的周期性判据（如果 <= 0 则停用）

`label`(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

`muGamma`(=*Matrix3r::Zero()*)对*sigmaCap*中接触线的张量贡献 Γ: $\mu_\Gamma = \int_\Gamma \nu_{nw} \otimes \mathbf{x}\, dl$ 其中 ν_{nw} 流体-流体界面余法向量 [Duriez2017c], 和 $\mathbf{x}$ 位置。（自动更新）

`muSnw`(=*Matrix3r::Zero()*)对*sigmaCap* 润湿/非润湿（例如液体/气体）界面的张量贡献Snw: $\mu_{Snw} = \int_{Snw}(\delta - \mathbf{n} \otimes \mathbf{n})dS$ 其中 $\mathbf{n}$ 外法向量和 δ 单位张量。（自动更新）

`muSsw`(=*Matrix3r::Zero()*)对*sigmaCap* 润湿固体表面的张量贡献 Ssw: $\mu_{Ssw} = \int_{Ssw} \mathbf{n} \otimes \mathbf{x} dS$其中 $\mathbf{n}$ 外法向量和 $\mathbf{x}$ 位置。（自动更新）

`muVw`(=*Matrix3r::Zero()*)对*sigmaCap* 润湿流体体积的张量贡献（球形，即各向同性）: $\mu_{Vw} = V_w\, \delta$ 其中 $V_w =$ *vW* 和 δ 单位张量。（自动更新）

`nDo`(=*-1*，已停用)
将执行次数限制为此数值（若为负数则停用）

`nDone`(=*0*)
跟踪执行次数（累计）（自动更新）。

`ompThreads`(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），该数量通常为 OMP_NUM_THREADS，或由 'yade -jN' 定义的数量 N（不过，此行为可能取决于引擎）。此属性只会影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会形成嵌套 OMP 循环，且每个层级的线程数不同。

`realLast`(=*0*)跟踪上次运行的实际时间（自动更新）。

`realPeriod`(=*0*, 停用)以实数时间（墙上时钟时间、计算时间或人工时间，单位为秒）为依据的周期性判据（若满足 <=0 则停用）

`sigmaCap`(=*Matrix3r::Zero()*)毛细应力张量 σ^{cap} 本身，表示为 $\sigma^{cap} = 1/V\,[u_c(\mu_{Vw} + \mu_{Ssw}) + \gamma_{nw}(\mu_{Snw} + \mu_\Gamma)]$ 其中，四个微观结构张量 μ_{Vw}，μ_{Ssw}，μ_{Snw}，μ_Γ分别对应*muVw*、*muSsw*、*muSnw* 和*muGamma* 属性。（自动更新）

`surfaceTension`(=*0.073*)
流体间表面张力 γ_{nw}，定义为定律*2_ScGeom_CapillaryPhys_Capillarity*.*surfaceTension*。

`timingDeltas`
引擎本身内部的详细计时信息。除非在源代码中启用并设置了 *O.timingEnabled*==`True`。

`updateAttrs`(*(可序列化对象)参数1*, *(字典)辐角2*) → 无:
根据给定字典更新对象属性

vW(=*0*)润湿流体体积，求和 弯月面体积 (这里比通过 Python 循环更快)。*(自动更新)*

virtLast(=*0*)
记录上次运行的虚拟时间 （自动更新）。

virtPeriod(=*0*, 停用)
基于虚拟（模拟）时间的周期性判据（如果 <= 0 则停用）

wettAngle(=*0*)润湿角，即接触角的值（弧度）。其值应与毛细管文件所依据的值保持一致（这些文件由 *定律2_ScGeom_CapillaryPhys_毛细作用*使用）。

class yade.wrapper.**PDFEngine**(继承 周期引擎 → 全局引擎 → 引擎 → 可序列化对象)

用于频谱计算的基类。以球坐标计算法向应力、剪切应力、距离、速度和相互作用的概率密度函数，并将结果写入文件。列名格式为：Data(theta, phi)。采用的约定：x：phi = 0，y：theta = 0，z：phi =pi/2

dead(=*false*)若为 true，此引擎将完全不会运行；可用于暂时停用引擎，并在之后的某个时刻重新启用。

dict(*(Serializable)arg1*) →字典：返
回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所耗费的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

filename(=*"PDF.txt"*)文件名

firstIterRun(=*0*)设置引擎首次执行时的步数（默认停用）。

initRun(=*false*)每次调用时也在首次运行。

iterLast(=*0*)跟踪上次运行的步数 （自动更新）。

iterPeriod(=*0*, 停用)使用步数的周期性判据（若 <= 0 则停用）

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

nDo(=*-1*，停用)将执行次数限制为此数量（若为负数则停用）

nDone(=*0*)跟踪执行次数（累计）（自动更新）。

numDiscretizeAnglePhi(=*20*)
φ角的扇区数量

numDiscretizeAngleTheta(=*20*)θ角的扇区数
量

ompThreads(=-1)
引擎中要使用的线程数。如果为 ompThreads<0 （默认值），该数量通常为 OMP_数量_线程，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性只会影响代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要用于实验，或用于将 *ParallelEngine* 与运行并行区域的引擎组合，从而形成嵌套 OMP 循环，使每个层级使用不同数量的线程。

realLast(=0)
记录上次运行的实际时间（自动更新）。

realPeriod(=0, 停用)使用实数（墙上时钟、计算、人类）时间（秒）的周期性判据（如果 <=0 则停用）

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用，否则为空；并且 *O.timingEnabled*==True。

updateAttrs（可序列化对象）参数*1*，（字典）辐角*2*）→ 无：从给定字典更新对象属性

virtLast(=0)
记录上次运行的虚拟时间（自动更新）。

virtPeriod(=0, 停用)基于虚拟（模拟）时间的周期性判据（若 <= 0 则停用）

warnedOnce(=*false*)用于一次性警告。可能会触发有用的警告

class yade.wrapper.PyRunner(继承 周期引擎 → 全局引擎 → 引擎 → 可序列化对象)

定期执行 Python 命令，周期性已定义（且可调整）。详情请参阅 周期引擎 文档。

command(="")
由 Python 解释器运行的命令。若为空则不运行。

dead(=*false*)若为真，则此引擎完全不会运行；可用于暂时停用引擎，并仅在之后的某个时间点重新启用它。

dict(*(Serializable)*辐角*1*）→字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所耗费的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

firstIterRun(=0)
设置引擎首次执行时的步数（默认禁用）。

ignoreErrors(=否)
仅用于调试：将此值设为 true，可让 PyRunner 忽略命令执行期间遇到的任何错误。

initRun(=否)
首次调用时也运行。

iterLast(=0)
记录上次运行的步数 （自动更新）。

`iterPeriod`(=*0*, 停用)
使用步数的周期性判据（如果 <= 0 则停用）

`label`(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

`nDo`(=*-1*，停用)
限制执行次数为此数值（若为负数则停用）

`nDone`(=*0*)
跟踪执行次数（累计）（自动更新）。

`ompThreads`(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性只会影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或用于将 *ParallelEngine* 与运行并行区域的引擎结合使用，从而形成嵌套 OMP 循环，且每个层级的线程数不同。

`realLast`(=*0*)
跟踪上次运行的实际时间（自动更新）。

`realPeriod`(=*0*, 已停用)
使用实数（墙上时钟、计算、人类）时间（秒）的周期性判据（如果 <=0 则停用）

`timingDeltas`
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==`True`，否则为空。

`updateAttrs`(*(可序列化对象)arg1*，*(字典)arg2*) →
None：根据给定的字典更新对象属性

`updateGlobals`
是否通过调用 IPython 的方式解决其无法识别局部变量的问题： `globals().update(locals())`。如果 `true`，那么 PyRunner 就能在运行中的 **活动** Yade 会话内调用稍后声明的函数。由于 `PyRunner` 会更新 `globals()`，使其包含最近声明的 Python 函数，因此调用速度会稍慢。

> **警告：**
> 当 `updateGlobals==False` 且函数是在 活动Yade 会话（IPython）中声明时，除非 `NameError: name 'command' is notdefined` 使用 Python 函数 `globals()` 通过 command 更新，否则会出错。
>
> ```
> globals().update(locals())
> ```

`virtLast`(=*0*)
跟踪上次运行的虚拟时间 （自动更新） .

`virtPeriod`(=*0*, 停用)
使用虚拟（模拟）时间的周期性判据（当 <= 0 时停用）

`class yade.wrapper.Recorder`(继承 周期引擎 → 全局引擎 → 引擎 → 可序列化对象)定期将一些数据存储到（一个）外部文件的引擎。除了周期引擎外，它还会根据需要处理文件的打开。参见 周期引擎 以控制周期性。

`addIterNum`(=*false*)
创建文件时，会在文件名中添加迭代次数。适用于每次调用时创建新文件（默认值为 false）

dead(=*false*)如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并在稍后的某个时间点重新启用。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

execCount
: 此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
: 此引擎运行所用的累计时间（以纳秒为单位，仅在*O.timingEnabled*==True时使用）。

file(=未初始化)要保存到的文件名；不能为空。

firstIterRun(=*0*)
: 设置步数，使引擎在该步首次执行（默认禁用）。

initRun(=*false*)首次调用时也运行。

iterLast(=*0*)跟踪上次运行的步数*(*自动更新*)*。

iterPeriod(=*0*, 停用)使用步数的周期性判据（若 <= 0 则停用）

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

nDo(=*-1*，停用)将执行次数限制为此数值（若为负数则停用）

nDone(=*0*)跟踪执行次数（累计）*(*自动更新*)*。

ompThreads(=*-1*)
: 引擎中要使用的线程数。如果 ompThreads<0 （默认值），则线程数通常为 OMP_NUM_THREADS，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响源代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与 *ParallelEngine* 结合使用包含并行区域的引擎，从而形成嵌套 OMP 循环，并且每一层使用不同数量的线程。

realLast(=*0*)
: 记录上次运行的实际时间 （自动更新）。

realPeriod(=*0*, 停用)以实际时间（墙上时钟、计算时间、人类时间）秒数为依据的周期性判据（如果 <=0 则停用）

timingDeltas
: 引擎本身内部的详细计时信息。除非在源代码中启用且 *O.timingEnabled*==True，否则为空。

truncate(=*false*)
: 打开时是否删除当前文件内容（如有；默认为 false）

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) →
None：根据给定的字典更新对象属性

virtLast(=*0*)记录上次运行的虚拟时间 （自动更新）。

virtPeriod(=*0*, 已停用)使用虚拟（模拟）时间的周期性判据（如果 <= 0，则停用）

class yade.wrapper.ResetRandomPosition(继承 周期引擎 → 全局引擎 → 引擎→ 可序列化对象)

在模拟过程中创建球体，并将其放置在随机位置。每次调用时，都会创建一个新球体并将其插入模拟中。

angularVelocity(=*Vector3r::Zero()*)球体的平均角速度。

angularVelocityRange(=*Vector3r::Zero()*)角速度分布区间的半尺寸。新球体的角速度将在角速度±角速度范围内随机取值。

dead(=*false*)若为 true，则此引擎完全不会运行；可用于暂时停用引擎，并在之后的某个时间点重新启用。

dict(（可序列化对象）辐角 *1*) →字典：返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎累计运行时间（以纳秒计，仅在*O.timingEnabled*==True时使用）。

factoryFacets(=未初始化*lized*)
球体将放置于该截面的几何体上；根据*volumeSection* 标志，球体将放置在面片上或它们之间的体积中。

firstIterRun(=*0*)
设置步数，使引擎在该步首次执行（默认禁用）。

initRun(=否)首次调用时也运行。

iterLast(=*0*)记录上次运行的步数 （自动更新）。

iterPeriod(=*0*, 停用)使用步数的周期性判据（如果 <= 0 则停用）

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，你可以直接从 Python 中引用它。maxAttempts(=*20*)放置球体的最大尝试次数。如果将球体放置在某个随机位置会导致其与模型中的其他物理体发生重叠，SpheresFactory 将尝试查找另一个位置。nDo(=*-1*，停用)将执行次数限制为此数值（如果为负数则停用）nDone(=*0*)记录执行次数（累计）（自动更新）。

normal(=*Vector3r(0，1，0)*)? ?

ompThreads(=*-1*)引擎中使用的线程数。如果为 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的 N（不过此行为可能

取决于具体引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在组合 *ParallelEngine* 与会运行并行区域的引擎时使用；这种组合会产生嵌套 OMP 循环，且各层的线程数不同。

`point`(=*Vector3r::Zero()*)??

`realLast`(=*0*)上次运行的实际时间（自动更新）。

`realPeriod`(=*0*, 停用)以实际时间（墙上时钟、计算、人类）秒为单位的周期性判据（若 <=0 则停用）

`subscribedBodies`(=未初始化)受影响的刚体。

`timingDeltas`
引擎本身内部的详细计时信息。除非在源代码中启用，且 *O.timingEnabled*==`True`。

`updateAttrs`(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) → None：根据给定字典更新对象属性

`velocity`(=*Vector3r::Zero()*)球体的平均速度。

`velocityRange`(=*Vector3r::Zero()*)速度分布区间的半尺寸。新球体的速度将在速度范围± velocityRange内随机取值。

`virtLast`(=*0*)记录上次运行的虚拟时间*(*自动更新*)*。

`virtPeriod`(=*0*, 已停用)使用虚拟（模拟）时间的周期性判据（在 <= 0 时停用）

`volumeSection`(=*false*，按面片定义生成器。)在工厂体积内而非其表面上创建新球体。

`class yade.wrapper.SnapshotEngine`(继承周期引擎 → 全局引擎 → 引擎 →可序列化对象)定期将 GLView(s) 快照保存为 .png 文件。文件命名格式为文件基名 + 计数器 + '.png'（计数器左侧补 0，例如 snap00004.png）。

`counter`(=*0*)下次保存快照时追加到文件基名的数字（每次保存后递增）。*(*自动更新*)*

`dead`(=*false*)若为 true，此引擎将完全不运行；可用于暂时停用引擎，并在之后重新启用。

`deadTimeout`(=*3*)
三维操作（打开新视图、保存快照）的超时时间；超时后抛出异常（若忽略错误则仅报告错误），并将自身设为停用。 [秒]

`dict`(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

`execCount`
此引擎运行的累计次数（仅在*O.timingEnabled*==`True`时使用）。

`execTime`
此引擎运行所花费的纳秒累计时间（仅在*O.timingEnabled*==`True`时使用）。

`fileBase`(=””)快照的基本文件名

`firstIterRun`(=*0*)
设置步数，指定引擎首次执行的步数（默认禁用）。

`format`(=*”PNG”*)快照格式（可选 JPEG、PNG、EPS、PS、PPM、BMP 之一）QGLViewer 文档。文件扩展名将转换为小写格式。不会检查格式是否有效。

`ignoreErrors`(=*true*)超时时仅报告错误，而不抛出异常。

`initRun`(=*false*)首次调用时也运行。

`iterLast`(=*0*)记录上次运行的步数（自动更新）。

`iterPeriod`(=*0*, 停用)基于步数的周期性判据（如果 $<= 0$ 则停用）

`label`(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

`msecSleep`(=*0*)
快照后休眠的毫秒数（以防止 3D 硬件问题） [毫秒]

`nDo`(=*-1*，停用)将执行次数限制为此数值（若为负数则停用）

`nDone`(=*0*)跟踪执行次数（累计）（自动更新）。

`ompThreads`(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由 ‘yade -jN’ 定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时，此时会产生嵌套 OMP 循环，且每一层的线程数不同。

`plot`(=未初始化)将快照自动追加到 *plot.imgData* 中的字段名称。

`realLast`(=*0*)记录上次运行的实际时间（自动更新）。

`realPeriod`(=*0*, 停用)使用实际时间（墙上时钟、计算时间、人类时间，单位为秒）的周期性判据（如果为 <=0，则停用）

`snapshots`(=未初始化)迄今创建的文件

`timingDeltas`
引擎本身内部的详细计时信息。除非在源代码中启用，否则为空。 ce
代码和 *O.timingEnabled*==`True`。

`updateAttrs`(*(可序列化对象)参数1*, *(字典)辐角2*) →
None：从给定字典中更新对象属性

`virtLast`(=*0*)记录上次运行的虚拟时间（自动更新）。

virtPeriod(=*0*, 已停用)使用虚拟（模拟）时间的周期性判据（若 <= 0 则停用）

class yade.wrapper.TorqueRecorder(继承记录器 → 周期引擎 → 全局引擎 →引擎 →可序列化对象)

引擎根据给定的轴和零点保存总扭矩，力取自刚体，列于标识符中。例如，这可用于定义球磨机运行过程中所受的扭矩。

addIterNum(=*false*)
创建文件时，将迭代次数添加到文件名中。此功能适用于每次调用时创建新文件（默认值为 false）

dead(=*false*)
若为真，则此引擎将完全不会运行；可用于暂时停用引擎，并仅在稍后重新启用。

dict(（可序列化对象）辐角 *1*) →字
典：返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所用时间的累计纳秒数（仅在*O.timingEnabled*==True时使用）。

file(=未初始化)要保存到的文件名；不能为空。

firstIterRun(=*0*)
设置引擎应首次执行的步数（默认停用）。

ids(=未初始化)将测量其状态的刚体列表

initRun(=*false*)也在首次调用时运行。

iterLast(=*0*)记录上次运行的步数 （自动更新）。

iterPeriod(=*0*, 已停用)使用步数的周期性判据（若 <= 0 则停用）

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

nDo(=*-1*，已停用)
将执行次数限制为此数值（若为负数则停用）

nDone(=*0*)
跟踪执行次数（累计）（自动更新）。

ompThreads(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时；由此产生的嵌套 OMP 循环在各层使用不同数量的线程。

realLast(=*0*)
记录上次运行的实际时间（自动更新）。

realPeriod(=*0*, 已停用)以实数（墙上时钟、计算时间、人类时间）秒为单位的周期性判据（若 <=0 则停用）rotationAxis(=*Vector3r::UnitX()*)旋转轴

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

totalTorque(=*0*)函数返回的合力矩。

truncate(=否)打开时是否删除当前文件内容（如果有），默认为否

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*) → 无：根据给定字典更新对象属性

virtLast(=*0*)记录上次运行的虚拟时间（自动更新）。

virtPeriod(=*0*, 停用)使用虚拟（模拟）时间的周期性判据（当 <= 0 时停用）

zeroPoint(=*Vector3r::Zero()*)旋转中心点

class yade.wrapper.TriaxialStateRecorder(继承 记录器 → 周期引擎 → 全局引擎 → 引擎 → 可序列化对象)

记录三轴变量的引擎（参见输出文件第一行中的变量列表）。此记录器需要在模拟中存在 三轴压缩引擎 或 三维三轴引擎 。

addIterNum(=否)创建文件时，会在文件名中添加迭代次数。适用于每次调用时创建新文件（默认为否）

dead(=否)如果为真，此引擎将完全不运行；可用于暂时停用引擎，并在之后的某个时刻重新启用。

dict(*(*可序列化对象*)*辐角*1*) →属性字典：返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所花费的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

file(=未初始化)要保存到的文件名；不能为空。

firstIterRun(=*0*)
设置步数，在该步数时引擎应首次执行（默认停用）。initRun(=*false*)首次调用时也运行。iterLast(=*0*)跟踪上次运行的步数*(*自动更新*)*。iterPeriod(=*0*, 停用)使用步数的周期性判据（当 <= 0 时停用）

`label`(=取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

`nDo`(=*-1*，停用)
将执行次数限制为此数量（若为负数则停用）

`nDone`(=*0*)
跟踪执行次数（累计）（自动更新）。

`ompThreads`(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由 'yade -jN' 定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，从而在不同层级产生线程数不同的嵌套 OMP 循环。

`porosity`(=*1*)堆积体的孔隙率 [-]

`realLast`(=*0*)记录上次运行的实际时间 （自动更新）。

`realPeriod`(=*0*, 已停用)使用实际时间（墙上时钟、计算、人）秒数的周期性判据（如果 <=0 则停用）

`timingDeltas`
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==`True`，否则为空。

`truncate`(=*false*)
打开时是否删除当前文件内容（如果有；默认为 false）

`updateAttrs`(*(可序列化对象)参数1*，*(字典)辐角2*) → None：根据给定字典更新对象属性

`virtLast`(=*0*)记录上次运行的虚拟时间 （自动更新）。

`virtPeriod`(=*0*, 停用)使用虚拟（模拟）时间的周期性判据（当 <= 0 时停用）

边界控制器

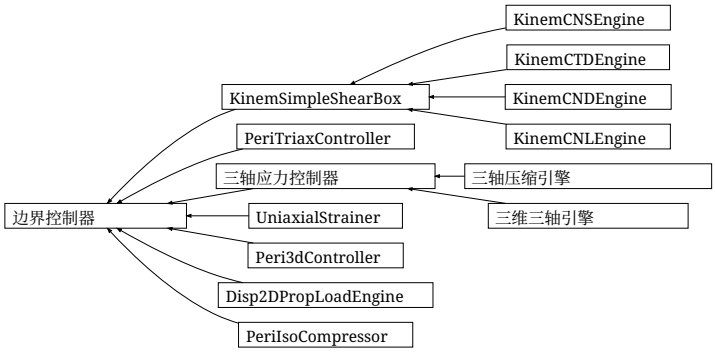


图 27：边界控制器的继承图。另请参阅：*Disp2DPropLoadEngine*，*KinemCNDEngine*，*KinemCNLEngine*，*KinemCNSEngine*，*KinemCTDEngine*，*KinemSimpleShearBox*，*Peri3dController*，*PeriIsoCompressor*，*PeriTriaxController*，*ThreeDTriaxialEngine*，*TriaxialCompressionEngine*，*Triaxial-StressController*，*UniaxialStrainer*。

class yade.wrapper.BoundaryController(继承 全局引擎 → 引擎 → 可序列化对象)用于控制模拟边界条件的引擎基类。请勿直接使用。

dead(=*f*否)
如果为真，则此引擎将完全不运行；可用于暂时停用引擎，并仅在稍后的某个时间点重新启用它。

dict(（可序列化对象）辐角*1*)→字典：返回属性字典。

execCount
此引擎运行的累计次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所耗费的累计纳秒数（仅在 *O.timingEnabled*==True 时使用）。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

ompThreads(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 (默认值)，线程数通常为 OMP_数量_THREADS，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如*InteractionLoop*）。此属性主要适用于实验，或与*ParallelEngine* 结合使用包含并行区域的引擎，从而在不同层级产生线程数各异的嵌套 OMP 循环。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用，否则为空；并且*O.timingEnabled*==True。

updateAttrs(*(*可序列化对象*)arg1*，*(*字典*)*辐角*2*)→None：根据给定字典更新对象属性

class yade.wrapper.Disp2DPropLoadEngine(继承 边界控制器 → 全局引擎 →引擎 →可序列化对象)沿给定的位移方向扰动简单剪切试样

此引擎允许在简单剪切试样上施加由 du/dgamma =恒定控制的载荷，这等效于 du + 恒定 * dgamma = 0 （比例路径载荷）。为此，简单剪切盒的上板沿给定方向移动（对应给定的 du/dgamma），而侧板则移动以使盒子保持闭合。此引擎可轻松用于执行方向探测：通过 Python 脚本，在修改加载方向（参见 θ 属性）后，依次运行包含此引擎的同一个 .xml 文件。因此，该引擎包含一个 *saveData* 过程，可在加载结束时保存试样状态的数据（若通过 Python 脚本依次进行多次加载，即沿不同方向加载，则每一行对应一个加载方向）。

Key(=“”)
添加到保存的文件名称以及由 *saveData* 填充的输出文件名称中的字符串

LOG(=*false*)
控制屏幕上消息输出的布尔值

dead(=*false*)
如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并在稍后的某个时间点重新启用它。

dict(*(*可序列化对象*)*辐角*1*)→字典：返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所花费的累计时间（以纳秒为单位，仅在*O.timingEnabled*==True时使用）。

id_boxback(=*4*)
试样后侧墙的标识符

id_boxbas(=*1*)下壁的标识符

id_boxfront(=*5*)样品前方的壁的标识符

id_boxleft(=*0*)左墙的标识符

id_boxright(=*2*)右墙的标识符

id_topbox(=*3*)上壁的标识符

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

nbre_iter(=*0*)要执行的加载迭代次数

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与包含并行区域的 *ParallelEngine* 引擎结合使用，从而在不同层级产生线程数各异的嵌套 OMP 循环。

theta(=*0.0*)
在 (伽马,h=-u) 平面内，从伽马轴到扰动向量的角度（按三角函数方向） [度]

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs((可序列化对象)参数*1*，(字典)辐角*2*) → None：根据给定字典更新对象属性

v(=*0。0*)施加扰动的速度。对于法向载荷比切向载荷更敏感的试样，一种可能的做法是取 v = V_剪切 - | (V_剪切- V_压缩)*正弦(θ) | => v=V_剪切（剪切时）；V_压缩（压缩时） [米/秒]

class yade.wrapper.KinemCNDEngine(继承 *KinemSimpleShearBox* → 边界控制器 →全局引擎 → 引擎 → 可序列化对象)
对平行四边形箱体施加恒定法向位移（CND）剪切

此引擎专为涉及简单剪切盒（*SimpleShear* 预处理器或 scripts/simpleShear.py）的模拟而设计，可通过水平平移上板来执行恒定法向位移剪切，同时旋转侧板，使其始终与上下壁接触。

Key(=*""*)添加到保存的文件名称中的字符串LOG(=*false*)控制屏幕上消息输出的布尔值

alpha(=*Mathr::PI/2.0*)从下方盒子到左侧盒子的角度（按三角函数方向）。由此引擎测量。必须保存，但不得由用户更改。

dead(=否)如果为真，此引擎将完全不运行；可用于暂时停用引擎，并仅在之后某个时间点重新启用。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

execCount此引擎运行的累计次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所用的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

f0(=*0*。*0*)第一个时间步上作用于上板的（竖直）力（由引擎确定）。在 *KinemCNSEngine* 或 *KinemCNLEngine* 的算例中，载荷控制将依据此初始值进行 [N]。必须保存，但用户不得更改。

firstRun(=真)
引擎运行一次作业后，该布尔值会立即设为 false：用于判断上部箱体的初始高度和所承受的法向力是否已知（从而判断是否需要初始化）。必须保存，但用户不得更改。

gamma(=*0*。*0*)当前切向位移的值

gamma_save(=未初始化)向量，包含执行模拟保存时的伽马值 [m]

gammalim(=*0*。*0*)停止位移时的切向位移值 [m]

id_boxback(=*4*)试样后侧墙的标识符

id_boxbas(=*1*)下壁的标识符

id_boxfront(=*5*)样品前方的壁的标识符

id_boxleft(=*0*)左墙的标识符

id_boxright(=*2*)右墙的标识符

id_topbox(=*3*)上壁的标识符

label(=取消初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

max_vel(=*1*。*0*)以限制用于控制 σ （CNL 或 CNS 算例） [m/s的垂直位移速度
]

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数将为 e
通常为 OMP_NUM_THREADS，或由'yade -jN'定义的 N 值（此行为可能 n
不过，具体取决于引擎）。此属性只会影响代码中包含 s

OpenMP 并行区域（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会产生嵌套 OMP 循环，且每一层的线程数不同。

`shearSpeed`(=*0.0*)
进行剪切的速度：上板的速度为 [米/秒]

`temoin_save`(=未初始化)
向量（例如，与"gamma_save"的长度相同），其值为 0 或 1，具体取决于对应 gamma 值的保存操作是否已完成（1）或尚未完成（0）。必须保存，但用户不得更改。

`timingDeltas`
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==`True`，否则为空。

`updateAttrs`(*(可序列化对象)参数1*，*(字典)辐角2*) → None：根据给定字典更新对象属性

`wallDamping`(=*0.2*)
用于控制 σ （CNL 或 CNS 算例）的垂直位移实际上会通过墙体阻尼进行阻尼处理

`y0`(=*0.0*)第一个时间步中上板的高度：引擎会找到其值 [m]。必须保存，但用户不得更改。

`class yade.wrapper.KinemCNLEngine`(继承 *KinemSimpleShearBox* → 边界控制器 →全局引擎 → 引擎 → 可序列化对象)
对平行四边形箱体施加恒定法向应力剪切（即恒定法向载荷：CNL），适用于（简单剪切盒：简单剪切 预处理器或 scripts/simpleShear.py）
该引擎可以使上板水平移动，同时侧板旋转，从而始终与上下壁保持接触。

事实上，上板不仅可以水平移动，也可以垂直移动，从而使作用于上板的法向应力保持恒定（该恒定值并非由用户设定，而是模拟开始时的应力值）。

允许的右侧垂直位移根据试样相对于壁面的刚度 Kn 计算（因此，为抵消 deltaSigma，会设定法向位移 dplt deltaSigma*S/(Kn)）。
此外，用户还可以通过 剪切速度 控制运动，该速度即上壁的速度；用户还可以设置水平位移的最大值 切向位移限值，达到该值后，剪切便会停止。

注意： 更新的不仅是壁面的位置，还有它们的速度；这绝非无关紧要，因为接触定律会使用物体速度来计算切向相对位移值。

警告： 正因如此，如果你想使用由此引擎执行的模拟的后续存档，但未执行 stop Movement，那么盒体会保留其速度 => 你将不得不在 .xml 中"手动"将其取消。

`Key`(=*""*)添加到保存文件名称中的字符串

`LOG`(=*false*)
控制屏幕上消息输出的布尔值

alpha(=*Mathr::PI/2.0*)下方盒子到左侧盒子的角度（按三角方向）。由此引擎测量。必须保存，但用户不得更改。

dead(=*false*)如果为 true，此引擎将完全不运行；可用于暂时停用某个引擎，并仅在之后的某个时刻重新启用它。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

execCount此引擎运行的累计次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
: 此引擎运行所耗费的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

f0(=*0*。*0*)引擎在第一个时间步作用于上板的（竖直）力。对于 *KinemCNSEngine* 或 *KinemCNLEngine* 算例，载荷将根据此初始值进行控制 [N]。必须保存，但用户不得更改。

firstRun(=真)
: 布尔值，一旦引擎完成一次作业便设为 false：用于判断是否已知上部箱体的初始高度及其承受的法向力（从而判断是否需要初始化）。必须保存，但用户不得更改。

gamma(=*0*。*0*)当前切向位移值 [米]

gamma_save(=未初始化)向量，包含执行模拟保存时对应的伽马值 [米]

gammalim(=*0*。*0*)停止剪切时（上板）的切向位移值 [米]

id_boxback(=*4*)试样后侧墙的标识符

id_boxbas(=*1*)下壁的标识符

id_boxfront(=*5*)样品前方的壁的标识符

id_boxleft(=*0*)左墙的标识符

id_boxright(=*2*)右墙的标识符

id_topbox(=*3*)上壁的标识符

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

max_vel(=*1*。*0*)用于限制控制 σ （CNL 或 CNS 算例）时垂直位移的速度 [m/s]

ompThreads(=*-1*)
: 引擎中要使用的线程数。如果 ompThreads<0 （默认），线程数将为 e
通常为 OMP_NUM_THREADS，或由'yade -jN'定义的数量 N（此行为可 n
不过，这取决于引擎）。此属性只会影响其代码包含 s

OpenMP 并行区域（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会产生嵌套 OMP 循环，且各层的线程数不同。

shearSpeed(=*0.0*)
进行剪切的速度：上板的速度为 [米/秒]

temoin_save(=未初始化)
向量（例如，与“伽马_保存”的长度相同），其值为 0 或 1，具体取决于对应伽马值的保存是否已完成（1）或尚未完成（0）。必须保存，但用户不得更改。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用，否则为空 *O.timingEnabled*==True。

updateAttrs(*(可序列化对象)参数1*, *(字典)辐角2*) → 无:
从给定字典更新对象属性

wallDamping(=*0.2*)
用于控制 σ （CNL 或 CNS 算例）的垂直位移实际上会通过墙体阻尼进行阻尼处理

y0(=*0。0*)第一个时间步中上板的高度：引擎会找出其值 [m]。必须保存，但用户不得更改。

class yade.wrapper.KinemCNSEngine(继承 *KinemSimpleShearBox* → 边界控制器 →全局引擎 → 引擎 → 可序列化对象)
对平行四边形箱体（简单剪切）施加恒定法向刚度（CNS）剪切
该引擎适用于包含一个可变形长方体盒的模拟，可使上板水平平移，同时侧板旋转，以始终与上下壁保持接触。上板不仅可以水平移动，也可以垂直移动，因此可使法向刚度定义为上板的 ΔF/ΔU ＝保持恒定（= *KnC*由用户定义）。

此外，用户还可通过 *shearSpeed* （上壁的水平速度）和最大水平位移 *gammalim* （上板的位移）来控制运动；达到该位移后，剪切将停止。

注意：更新的不仅是壁面位置，还有它们的速度；考虑到接触定律中会使用物体速度来计算切向相对位移，这一点绝非无关紧要。

> **警告：**但是，由于上述原因，如果之后要使用通过此引擎运行的模拟存档，而运行期间未执行 stopMovement，那么刚体仍会保留其速度 =>，必须手动在 .xml 文件中将其清除。

Key(=””)
添加到保存的文件名称中的字符串

KnC(=*10.0e6*)
用户选择的法向刚度 [兆帕/毫米]- 将转换为帕/米

LOG(=*false*)
控制屏幕上消息输出的布尔值

alpha(=*Mathr::PI/2.0*)从下方盒子到左侧盒子的角度（按三角函数方向）。由此引擎测量。必须保存，但用户不能更改。

dead(=*false*)如果为真，此引擎将完全不运行；可用于暂时停用引擎，并在之后某个时间点重新启用。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所用的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

f0(=*0*。*0*)第一个时间步中作用于上板的（竖直）力（由引擎确定）。在 *KinemCNSEngine* 或 *KinemCNLEngine* 算例中，载荷将根据此初始值进行控制 [N]。必须保存，但用户不得更改。

firstRun(=真)
布尔值，一旦引擎完成一次作业便设为 false：用于判断上部箱体的初始高度以及所受法向力是否已知（从而判断是否需要初始化）。必须保存，但用户不得更改。

gamma(=*0.0*)
当前切向位移值 [m]

gammalim(=*0*。*0*)剪切停止时（上板）的切向位移值 [m]

id_boxback(=*4*)试样后侧墙的标识符

id_boxbas(=*1*)下壁的标识符

id_boxfront(=*5*)样品前方的壁的标识符

id_boxleft(=*0*)左墙的标识符

id_boxright(=*2*)右墙的标识符

id_topbox(=*3*)上壁的标识符

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

max_vel(=*1*。*0*)用于限制控制 σ (CNL 或 CNS 算例) 时执行的垂直位移速度 [m/s]

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或用于将 *ParallelEngine* 与运行并行区域的引擎组合，从而形成嵌套 OMP 循环，且各层的线程数不同。

shearSpeed(=*0.0*)
剪切速度：上板的速度为 [米/秒]

temoin_save(=未初始化)
向量（例如，与 'gamma_save' 长度相同），取值为 0 或 1，分别表示对应伽马值的保存是否已完成（1）或尚未完成（0）。必须保存，但用户不得更改。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) →
None：根据给定字典更新对象属性

wallDamping(=*0.2*)
用于控制 σ （CNL 或 CNS 算例）的垂直位移实际上会通过墙体阻尼进行阻尼处理

y0(=*0。0*)第一个时间步中上板的高度：引擎会查找其值 [m]。必须保存，但用户不得更改。

class yade.wrapper.KinemCTDEngine(继承 *KinemSimpleShearBox* → 边界控制器 →全局引擎 → 引擎 可序列化对象)
仅通过垂直移动上部箱体来压缩简单剪切试样，使切向位移（由上箱体与下箱体之间的水平间隙定义）保持恒定（即 CTD = 恒定切向位移）。侧边箱体也会移动，以始终保持接触。此过程持续进行，直至该箱体承受给定应力（目标应力）。此外，在向量 应力_存档中存储的每个应力值处，以及在 目标应力处执行存档

Key(=*""*)要添加到已保存文件名称中的字符串

LOG(=*false*)
控制屏幕消息输出的布尔值

alpha(=*Mathr::PI/2.0*)下方盒子到左侧盒子的角度（按三角函数方向）。由此引擎测量。必须保存，但用户不能更改。

compSpeed(=*0.0*)
上部箱体的（垂直）速度： >0 用于实际压缩， <0 用于卸载 [m/s]

dead(=否)
如果为真，此引擎将完全停止运行；可用于暂时停用引擎 ed
并仅在之后的某个时间点重新启用它。

dict(（可序列化对象）辐角*1*) →字
典：返回属性字典。

execCount
此引擎运行的累计次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所用的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

f0(=*0.0*)
第一个时间步中作用于上板的（垂直）力（由引擎确定）。对于*KinemCNSEngine* 或 *KinemCNLEngine* 算例，载荷将根据此初始值进行控制 [N]。必须保存，但用户不得更改。

firstRun(=真)
布尔值；引擎完成一次作业后立即设为 false：可用于判断上部箱体的初始高度和所承受的法向力是否已知（从而判断是否需要初始化）。必须保存，但不得由用户更改。

id_boxback(=4)试样后侧墙的标识符
id_boxbas(=1)下壁的标识符

id_boxfront(=5)样品前方的壁的标识符

id_boxleft(=0)左墙的标识符

id_boxright(=2)右墙的标识符

id_topbox(=3)上壁的标识符

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

max_vel(=1.0)
限制垂直位移的速度，以控制 σ （CNL 或 CNS 算例） [m/ s]

ompThreads(=-1)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由 'yade -jN' 定义的数量 N（不过此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时；这种情况下会产生嵌套 OMP 循环，且每一层的线程数各不相同。

sigma_save(=未初始化)
包含应力值的向量，在这些应力值处应保存模拟 [kPa]

targetSigma(=0。0)应停止压缩时的应力值 [kPa]

temoin_save(=取消初始化)向量（与'gamma_save'的长度相同），其值根据相应 gamma 的保存操作是否已完成而为 0 或 1：已完成为 1，未完成为 0。必须保存，但用户不得更改。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用，否则为空 ce
代码和 *O.timingEnabled*==True。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角
2) → 无：根据给定字典更新对象属性

wallDamping(=0.2)
用于控制 σ （CNL 或 CNS 算例）的垂直位移实际上会通过墙体阻尼 thiswallDamping 受到阻尼

y0(=0。0)第一个时间步中上板的高度：引擎会查找其数值 [m]。该值必须保存，但用户不得更改。

class yade.wrapper.KinemSimpleShearBox(继承自 边界控制器 → 全局引擎 →引擎 →可序列化对象)

这个类旨在作为所有对 简单剪切 盒子施加载荷的引擎的基类。它本身并非供直接使用，但其声明和实现包含了这些引擎通用的所有内容。脚本 simpleShear.py 展示了如何使用各种相关引擎。

Key(="")要添加到保存的文件名称中的字符串

LOG(=否)控制屏幕上消息输出的布尔值

alpha(=*Mathr::PI/2.0*)下方盒子到左侧盒子的角度（按三角函数方向）。由此引擎测量。必须保存，但用户不得更改。

dead(=否)如果为真，此引擎将完全不运行；可用于暂时停用引擎，并仅在稍后的某个时间点重新启用。

dict(*(Serializable)*辐角 *1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
该引擎运行所用的累计时间（以纳秒为单位；仅在以下条件下使用：*O.timingEnabled*==True）。

f0(=*0*。*0*)第一个时间步中作用于上板的（竖直）力（由引擎确定）。对于 *KinemCNSEngine* 或 *KinemCNLEngine* ，载荷将根据此初始值进行控制 [N]。必须保存，但用户不得更改。

firstRun(=*true*)
布尔值；引擎完成一次作业后立即设为 false：可用于判断上部箱体的初始高度和所承受的法向力是否已知（进而判断是否需要初始化）。必须保存，但用户不得更改。

id_boxback(=*4*)
试样后侧墙的标识符

id_boxbas(=*1*)下壁的标识符

id_boxfront(=*5*)样品前方的壁的标识符

id_boxleft(=*0*)左墙的标识符

id_boxright(=*2*)右墙的标识符

id_topbox(=*3*)上壁的标识符

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

max_vel(=*1.0*)
用于限制为控制 σ （CNL 或 CNS 算例） [m/ 而进行的垂直位移速度 s]

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），通常线程数为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时，此时会形成嵌套 OMP 循环，且各层的线程数不同。

temoin_save(=未初始化)
向量（例如，与“gamma_save”的长度相同），其中的值为 0 或 1，分别表示对应 gamma 值的保存是否已完成（1 表示已完成，0 表示未完成）。必须保存，但用户不得更改。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用，否则为空 *O.timingEnabled*==True。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无:
根据给定的字典更新对象属性

wallDamping(=*0*。*2*)
通过 thiswallDamping 对为控制 σ （CNL 或 CNS 算例）所做的垂直位移进行阻尼

y0(=*0*。*0*)第一个时间步时上板的高度：引擎会找到它的值 [m]。必须保存，但不能由用户更改。

class yade.wrapper.Peri3dController(继承 边界控制器 → 全局引擎 → 引擎 →可序列化对象)

用于独立控制周期性单元的“工程”应力和应变的类。 目标值是目标值，而 应力掩码用于确定哪些分量规定应力，哪些分量规定应变。

如果规定了应变，则直接施加适当的应变率。如果规定了应力，则使用应变预测器：根据前两个时间步的应力值，规定应变率，使下一步的应力值尽可能接近理想值。当前算法极其简单，今后可能会更改，但足够稳健，且大多数情况下都能正常工作。

在当前步和前一步（$d\sigma_i$、$d\sigma_{i-1}$）中评估应力误差（实际应力与理想应力之间的差异）。使用线性外推估算下一步的误差

$$d\sigma_{i+1} = 2d\sigma_i - d\sigma_{i-1}$$

根据此误差，应变率会通过 *mod* 参数进行调整

$$d\sigma_{i+1} \begin{cases} > 0 \rightarrow \dot{\varepsilon}_{i+1} = \dot{\varepsilon}_i - \max(\mathrm{abs}(\dot{\varepsilon}_i)) \cdot \mathrm{mod} \\ < 0 \rightarrow \dot{\varepsilon}_{i+1} = \dot{\varepsilon}_i + \max(\mathrm{abs}(\dot{\varepsilon}_i)) \cdot \mathrm{mod} \end{cases}$$

根据这一事实，规定的应力（几乎）永远不会达到精确的规定值，但差异会非常小（并且随着 *nSteps* 增大而减小）。如果规定了一个主导应变率，此方法效果良好。如果规定了所有应力，或者所有目标应变均规定为零，则需要为第一步提供良好的估算。因此，将估算柔度矩阵（根据用户定义的宏观材料参数估算 *youngEstimation*和 *poissonEstimation*），并根据规定的应力率和柔度矩阵计算相应的应变率（柔度矩阵的估算可以自动计算，从而避免用户输入此类参数）。

也已支持对旋转周期单胞进行模拟。首先，对单胞的变换矩阵 极分解 trsf $\mathcal{T} = \mathbf{U}\mathbf{P}$进行极分解，其中 $\mathbf{U}$ 是表示旋转的正交（酉）矩阵，而 $\mathbf{P}$ 是表示应变的半正定厄米矩阵。应对 $\mathbf{P}$ 取对数，以便在较高应变值下获得符合实际的值（尚未实现）。全局坐标系中规定的应变增量 $dt \cdot \dot{\varepsilon}$ 会适当地旋转到单胞的局部坐标系中，并加到 $\mathbf{P}$

$$\mathbf{P}_{i+1} = \mathbf{P} + \mathbf{U}^T dt \cdot \dot{\varepsilon}\mathbf{U}$$

The 新值变换的计算时刻为 $\mathbf{T}_{i+1} = \mathbf{U}\mathbf{P}_{i+1}$。根据当前和下一个变换单元的速度 y 梯度 其中速度梯度 根据其定义计算如下

$$\mathbf{V} = (\mathbf{T}_{i+1}\mathbf{T}^{-1} - \mathbf{I})/dt$$

当前实现允许用户为每个指定分量定义独立的加载“路径”，即将指定值定义为时间的函数（或 进度 或步）。请参阅 路径。

示例 examples/test/peri3dController_example1.py和 examples/test/peri3dController_-triaxialCompression.py 说明 Peri3dController 的用法和输入，exam-ples/test/peri3dController_shear.py 是一个使用剪切分量以及在旋转单元上进行模拟的示例。

`dead`(=*false*)
如果为 true，此引擎将完全不会运行；可用于暂时停用引擎，并在稍后的某个时间点重新启用它。

`dict`((可序列化对象)辐角 *1*) →字典:
返回属性字典。

`doneHook`(=未初始化)达到 nSteps 时要运行的 Python 命令（字符串形式）*nSteps* 达到时。如果为空，引擎将被设置为停用。

`execCount`
此引擎的累计运行次数（仅在 *O.timingEnabled*==`True` 时使用）。

`execTime`
此引擎运行所用的累计时间（以纳秒为单位，仅在*O.timingEnabled*==`True`时使用）。

`goal`(=*Vector6r::Zero()*)目标状态；仅考虑上三角矩阵；每个分量根据 *stressMask*，表示指定的应力或应变。

`label`(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

`lenPe`(=*0*)Peri3dController 内部变量

`lenPs`(=*0*)Peri3dController 内部变量

`maxStrain`(=*1e6*)模拟中允许的应变最大绝对值。达到该值时，即认为模拟已完成

`maxStrainRate`(=*1e3*)应变率（应变的法向量分量和剪切分量）的最大绝对值 应变）

`mod`(=*.1*)
预测器修正因子，通过反复试验分析，发现 0.1 是最佳值。

`nSteps`(=*1000*)模拟的步数。

`ompThreads`(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会产生嵌套 OMP 循环，且每一层的线程数不同。

`pathSizes`(=*Vector6i::Zero()*)Peri3dController
内部变量

`pathsCounter`(=*Vector6i::Zero()*)
Peri3dController 内部变量

pe(=*Vector6i::Zero()*)Peri3dController 内部变量**poissonEstimation**(=。*25*)宏观泊松比的估算，用于第一个模拟步

progress(=*0*。)使用控制器进行的模拟当前进度。

ps(=*Vector6i::Zero()*)Peri3dController 内部变量

strain(=*Vector6r::Zero()*)当前应变（变形）向量（ε_x，ε_y，ε_z，γ_{yz}，γ_{zx}，γ_{xy}）（自动更新）。

strainGoal(=*Vector6r::Zero()*)
Peri3dController 内部变量

strainRate(=*Vector6r::Zero()*)当前应变率向量。

stress(=*Vector6r::Zero()*)当前应力向量（σ_x,σ_y,σ_z,τ_{yz},τ_{zx},τ_{xy}）| y更新|。

stressGoal(=*Vector6r::Zero()*)
Peri3dController 内部变量

stressIdeal(=*Vector6r::Zero()*)当前时间步的理想应力向量。

stressMask(=*0*,全部应变)用于确定 目标值 各分量是应变（0）还是应力（1）的掩码。顺序从最低有效位开始依次为 00、11、22、12、02、01。（例如，0b000011 表示应力 00 和应力 11）。

stressRate(=*Vector6r::Zero()*)当前应力率向量（即指定的应力率，实际值略有不同）。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用并设置 *O.timingEnabled*==True。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无:
根据给定字典更新对象属性

xxPath
xx方向的“时间函数”（分段线性）。由数字对组成的序列。第一个数字是时间，第二个数字是相应量（应力或应变）的目标值。最后一对数字被视为最终状态（等于 (*nSteps*，目标值)），其他值均相对于此状态。

示例：nSteps=1000，goal[0]=300，xxPath=((2,3),(4,1),(5,2))

在第 400 步（=5*1000/2）时，值为 450（=3*300/2），

在第 800 步（=4*1000/5）时，值为 150（=1*300/2），

在第 1000 步（=5*1000/5=nSteps）时，值为 300（=2*300/2=goal[0]）。

参见示例 scripts/test/peri3dController_example1以供说明。

xyPath(=向量<*Vector2r*>*(1，Vector2r::Ones())*)
xy 方向的时间函数，参见 *xxPath*

youngEstimation(=*1e20*)宏观杨氏模量的估算值，用于第一个模拟步

yyPath(=向量<*Vector2r*>*(1，Vector2r::Ones())*)
yy 方向的时间函数，参见 *xxPath*

yzPath(=向量<*Vector2r*>*(1, Vector2r::Ones())*
yz 方向的时间函数，参见 *xxPath***zxPath**(=向量<
Vector2r>*(1, Vector2r::Ones())*)zx 方向的时间函数，
参见 *xxPath***zzPath**(=向量<*Vector2r*>*(1,*
Vector2r::Ones()))zz 方向的时间函数，参见
xxPath

class yade.wrapper.PeriIsoCompressor(继承边界控制器 → 全局引擎 → 引擎 →可序列化对象)

通过控制周期单元尺寸来压缩/膨胀球体云，直至达到指定的平均应力，然后移动到给定应力序列中的下一个应力值。

charLen(=*-1.*)特征长度，应为类似平均颗粒直径的值（默认值 -1=无效值）

currUnbalanced
当前不平衡力的值

dead(=*false*)如果为 true，此引擎将完全不运行；可用此设置暂时停用引擎，并在稍后重新启用。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

doneHook(=””)
达到最后指定的应力时要运行的 Python 命令

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所耗纳秒数的累计时间（仅在*O.timingEnabled*==True时使用）。

globalUpdateInt(=*20*)
重新计算平均应力、刚度和不平衡力的频率

keepProportions(=是)精确保留单元的比例（应力根据平均值而非各分量进行控制

label(=已取消初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

maxSpan(=*-1.*)以包围盒计的最大物体跨度，以防周期单元过小。（自动计算）

maxUnbalanced(=*1e-4*)如果实际不平衡力小于此数值，则认为堆积体稳定，

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），该数量通常为 OMP_NUM_THREADS，或由 ‘yade -jN’ 定义的数量 N（不过此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会形成嵌套 OMP 循环，且每一层的线程数不同。

sigma
当前应力值

state(=*0*)应力序列进行到哪里了

stresses(=未初始化)应依次达到的应力

timingDeltas
引擎本身的详细计时信息。除非在源码中启用，否则为空 ce
代码和 *O.timingEnabled*==True。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.PeriTriaxController(继承自边界控制器 → 全局引擎 →引擎 →可序列化对象)

用于在周期性模拟中独立控制应力或应变的引擎。

PeriTriaxController.goal 包含受控量的绝对值，而 *Peri-TriaxController.stressMask* 决定这些值的含义（0 表示应变，1 表示应力）：例如，`( 1<<0 | 1<<2 ) = 1 | 4 = 5` 表示 `goal[0]` 和 `goal[2]` 是应力值，而 `goal[1]`是应变。

简单示例请参见 scripts/test/periodic-triax.py。

absStressTol(=*1e3*)应力绝对容差

currUnbalanced(=非数)当前不平衡力（每次 globUpdate 时更新）（自动更新）

dead(=*false*)若为 true，则此引擎将完全不运行；可用于暂时停用引擎，并在之后某个时间点重新启用。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

doneHook(=未初始化)达到目标状态时要运行的 Python 命令

dynCell(=否)施加应力可以通过堆积刚度进行控制，也可以通过应用动力学定律来控制（dynCell=是）。别忘了为单元指定质量。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所花费的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

externalWork(=*0*)
来自边界控制器的功输入。**globUpdate**(=*5*)重新计算平均应力、刚度和不平衡力的频率。

goal
期望的应力或应变值（取决于应力掩码），应变定义为strain(i)=log(Fii)。**警告：** 应变是相对于*O.cell.refSize*（参考单元尺寸）而言的，而非相对于当前尺寸（例如，设置新应变值时的尺寸）。

growDamping(=。*25*)单元缩放阻尼（0=理想控制， 1=完全不控制）；另请参阅 wallDamping 中的三轴应力控制器。**label**(=未初始化)

此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

mass(=非数)单元质量（由用户设置）；若未设置且 动态单元 已启用，则将通过对全部颗粒的质量求和来计算。

maxBodySpan(=*Vector3r::Zero()*)最大物体尺寸 （自动计算）

maxStrainRate(=*Vector3r(1， 1， 1)*)周期单元的最大应变率。

maxUnbalanced(=*1e-4*)最大不平衡力。

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或者由'yade -jN'定义的 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要用于实验，或用于将 *ParallelEngine* 与运行并行区域的引擎组合使用，从而形成每层线程数各不相同的嵌套 OMP 循环。

prevGrow(=*Vector3r::Zero()*)上一次单元增长**relStressTol**(=*3e-5*)相对应力容差

stiff(=*Vector3r::Zero()*)平均刚度（仅每隔 globUpdate 步根据相互作用重新计算）（自动更新）

strain(=*Vector3r::Zero()*)单元应变 （自动更新）

strainRate(=*Vector3r::Zero()*)单元应变率 （自动更新）

stress(=*Vector3r::Zero()*)应力张量的对角项

stressMask(=*0*,全部应变)用于指定目标分量含义（应变/应力，0/1）的掩码

stressTensor(=*Matrix3r::Zero()*)平均应力，每步更新（仅每隔全局更新步数步，在 !dynCell 时根据相互作用重新计算）

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(Serializable)arg1*, *(*字典*)*辐角*2*) → None：根据给定字典更新对象属性

class yade.wrapper.ThreeDTriaxialEngine(继承 三轴应力控制器 → 边界控制器 → 全局引擎 → 引擎 →可序列化对象)

该引擎执行三轴压缩，并在应力控制（如果 stressControl_i）下沿"i"方向进行控制，否则采用应变控制。

对于应力控制，施加应力由“sigma_i”指定，并根据“strainRatei”设置“max_veli”。若要在应力控制下获得与应变控制相同的应变率，需要设置“wallDamping = 0.8”。对于应变控制，施加应变由“strainRatei”指定。使用此引擎还可以通过增大颗粒尺寸来执行内部压实，使用 `TriaxialStressController::controlInternalStress`即可。为此，只需开启“internalCompaction=1”，并将 sigma_iso=设置为内部压实结束时所需的平均压力值。

警告： 此引擎已弃用，如果希望获得长期支持，请切换到三轴应力控制器。

`Key(="")`追加到所有文件末尾的字符串，用于命名模拟。

`UnbalancedForce(=1)`
平均合力除以平均接触力

`boxVolume`
总堆积体积。

`computeStressStrainInterval(=10)`

`currentStrainRate1(=0)`
方向 1 的当前应变率 - 正在趋近 *ThreeDTriaxialEngine::strainRate1* (./s)

`currentStrainRate2(=0)`方向 2 的当前应变率 - 正在收敛至*ThreeDTriaxialEngine::strainRate2* (./s)

`currentStrainRate3(=0)`方向 3 的当前应变率 - 正在收敛至*ThreeDTriaxialEngine::strainRate3* (./s)

`dead(=否)`若为 true，此引擎将完全不运行；可用于暂时停用引擎，并在之后的某个时刻重新启用它。

`depth(=0)`盒子尺寸（2 轴）（自动更新）

`depth0(=0)`应变定义的参考尺寸。参见*TriaxialStressController::depth*

`dict((可序列化对象)辐角1)` →字典:
返回属性字典。

`execCount`
此引擎已运行的累计次数（仅在*O.timingEnabled*==`True`时使用）。

`execTime`
此引擎运行所花费的累计时间（纳秒），仅在*O.timingEnabled*==`True`时使用。

`externalWork(=0)`
与边界条件相关的机械功，即边界处的表面牵引力 **T** 和位移 **u** 所做的功。（自动更新）

`finalMaxMultiplier(=1。00001)`内部压实期间直径的最大乘数（二次精确调整 - *Tri- axial应力控制器::maxMultiplier* 在初始阶段使用）

`frictionAngleDegree(=-1)`
模拟中使用的摩擦值（如果启用 updateFrictionAngle）
`goal1(=0)`轴 1 上指定的应力/应变率，由*TriaxialStressController::stressMask*定义

goal2(=0)轴 2 上指定的应力/应变率，由*TriaxialStressController::stressMask*定义

goal3(=0)
: 轴 3 上指定的应力/应变率，由以下内容定义：*TriaxialStressController::stressMask*

height(=0)
: 箱体尺寸（轴 1）（自动更新）

height0(=0)应变定义的参考尺寸。参见*TriaxialStressController::height*

internalCompaction(=true)在“外部”（壁面）和“内部”（颗粒增长）压实方式之间切换。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

maxMultiplier(=1.001)
: 内部压实时直径的最大乘数（初始快速增大 - 三轴应力应变控制器 *ss-Controller::finalMaxMultiplier* 用于第二阶段）

max_vel(=1)
: 允许的最大壁面速度 [米/秒]。如果应力控制器指定的值更高，则此值会取代该值。最大_速度在许多情况下可以设为无穷，但有时设定有限值有助于稳定堆积体。根据此值，会基于试样的尺寸为每个轴计算不同的最大值，使得每个边界都以最大速度移动时，应变率为各向同性（例如参见 *TriaxialStressController::max_vel1*）。

max_vel1
: 参见*TriaxialStressController::max_速度* （自动计算）

max_vel2
: 参见*TriaxialStressController::max_速度* （自动计算）

max_vel3
: 请参见三轴应力控制器::最大值_速度 （自动计算）

meanStress(=0)
: 堆积体中的平均应力。（自动更新）

ompThreads(=-1)
: 引擎中要使用的线程数。如果 ompThreads<0 （默认值），该数量通常为 OMP_线程数_，或为由 ‘yade -jN’ 定义的数量 N（不过此行为可能取决于引擎）。此属性只会影响其代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时，从而导致不同层级的嵌套 OMP 循环使用不同数量的线程。

particlesVolume
: 颗粒总体积（团簇和 动态 球体）。 （自动计算）

porosity
: 堆积体的孔隙率，根据 颗粒体积 和 盒体体积计算。 （自动更新）

previousMultiplier(=1)(自动更新)

previousStress(=0)(自动更新)

radiusControlInterval(=10)

setContactProperties((三维三轴引擎)参数1，(浮点数)辐角2) → 无:
: 为动力体和相对相互作用指定新的摩擦角（度）

spheresVolume
的简写形式 *TriaxialStressController::颗粒体积*

stiffnessUpdateInterval(=*10*)
用于测量合成堆积边界刚度的迭代周期，以进行应力伺服控制

strain
向量中的当前应变（exx、eyy、ezz）。这些值表示真实的（对数）应变。

strainDamping(=*0.9997*)
用于平滑有效应变率变化的因子。如果目标速率为 TR，则每次迭代都会添加 (1- 阻尼)*(TR-当前速率)。当阻尼为=0时，速率=始终以目标值为准。当阻尼为=1时，速率不会发生变化。

strainRate
向量中的当前应变率 d/dt(exx、eyy、ezz)。

strainRate1(=*0*)
方向 1 的目标应变率（./s，>0 表示压缩）

strainRate2(=*0*)
方向 2 的目标应变率（./s，>0 表示压缩）

strainRate3(=*0*)
方向 3 的目标应变率（./s，>0 表示压缩）

stress(*(三轴应力控制器)参数1*，*(int)*标识符）→ 三维向量：返回边界“标识符”上的平均应力。此处，“标识符”指边界的内部编号，范围为 0 到 5。

stressControl_1(=*true*)用于选择方向 1 上应力控制或应变控制的开关

stressControl_2(=*true*)用于选择方向 2 上应力控制或应变控制的开关

stressControl_3(=*true*)用于选择方向 3 上应力控制或应变控制的开关

stressDamping(=*0.25*)应力控制的壁面阻尼系数 - wallDamping=0 表示理想控制（理论上），wallDamping=1表示不发生移动

stressMask(=*7*)
用于确定施加的目标值是应力（0 表示无，7 表示全部，1 表示方向 1，5 表示方向 1 和 3 等）还是应变率的位掩码

thickness(=*-1*)盒体厚度（某些函数需要）

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用且设置*O.timingEnabled*==True。

updateAttrs(*(可序列化对象)arg1*，*(字典)arg2*）→
None：根据给定的字典更新对象属性

updateFrictionAngle(=假)
启用将颗粒间摩擦更新为该值的开关*ThreeDTriaxi- alEngine::摩擦角度*。

updatePorosity(=假)
如果为 true，固体体积将更新一次（计算一步后会自动重置为 false），例如用于孔隙率计算。当模拟期间颗粒体积发生变化时（例如颗粒被删除或创建团簇），可以使用此功能。

volumetricStrain(=*0*)体积应变（参见*TriaxialStressController::strain*）。（自动更新）
wall_back_activated(=*true*)如果为 true，此壁面将根据目标值（应力或应变率）移动。
wall_back_id(=*4*)边界的标识符；坐标 2（如果在球体之前添加 aabbWalls，默认值即可。）

wall_bottom_activated(=*true*)若为 true，则此壁面将根据目标值（应力或应变率）移动。

wall_bottom_id(=*2*)边界的标识符；坐标 1（如果在球体之前添加 aabbWalls，则默认值即可。）

wall_front_activated(=*true*)若为 true，则此壁面将根据目标值（应力或应变率）移动。

wall_front_id(=*5*)边界的标识符；坐标 $2+$（如果在球体之前添加 aabbWalls，则默认值即可。）

wall_left_activated(=*true*)若为 true，则此壁面将根据目标值（应力或应变率）移动。

wall_left_id(=*0*)边界的标识符；坐标 0（如果在球体之前添加 aabbWalls，则可使用默认值）。

wall_right_activated(=*true*)若为 true，此墙将根据目标值（应力或应变率）移动。

wall_right_id(=*1*)边界的标识符；坐标 $0+$（如果在球体之前添加 aabbWalls，则可使用默认值）。

wall_top_activated(=*true*)若为 true，此墙将根据目标值（应力或应变率）移动。
wall_top_id(=*3*)边界的标识符；坐标 $1+$（如果在球体之前添加 aabbWalls，则可使用默认值。）
width(=*0*)盒子的尺寸（0 轴）（自动更新）**width0**(=*0*)应变定义的参考尺寸。参见*TriaxialStressController::width*

class yade.wrapper.TriaxialCompressionEngine(继承自*TriaxialStressController* →边界控制器 → 全局引擎 → 引擎 →可序列化对象)

引擎是一个状态机，包含以下状态；状态之间可能会自动转换，见下文。1. STATE_ISO_COMPACTION：进行各向同性压实（压缩），直至达到规定的平均压力 sigmaIsoCompaction，且堆积体稳定。压实通过拉伸壁面（!internalCompaction）或增大颗粒尺寸（internalCom-paction）实现。2. STATE_ISO_UNLOADING：从先前达到的状态开始进行各向同性卸载，直至达到平均压力 sigmaLateralConfinement（并稳定下来）。

注意：如果 sigmaLateralConfinement == sigmaIsoCom-paction，则会跳过此状态。

3. 状态_三轴应力应变控制器_加载：围压单轴压缩：侧壁（左侧、右侧、前侧、后侧）上的恒定侧向围压 sigmaLateralConfinement 保持不变，而顶部和底部壁面沿其轴线（通过施加应变）加载堆积体，直到达到 epsilonMax（沿加载轴的变形）值。此时，模拟将停止。

4. 状态_固定_孔隙率_压实：各向同性压实（压缩），直至达到选定的孔隙率值（参数：fixedPorosity）。六个壁面以选定的平移速度移动（参数 StrainRate）。

5. 状态_三轴应力应变控制器_暂置：当前未使用，因为模拟在前一状态中被强制停止。

Tran从压实过渡到卸载会自动完成，只要 autoUnload==true；

从（卸载到加载）或从（压实到加载：如果跳过卸载）的过渡会自动完成，只要 autoCompressionActivation=true；默认情况下，autoUnload 和 autoCompressionActivation 均为 true。

注意： 大多数所用算法最初是为以下文献报告的模拟而开发的[Chareyre2002a]和 [Chareyre2005]。随后，这些算法被移植到 Yade，并用于例如 [Kozicki2008],[Scholtes2009b]_、[Jerier2010b]。

警告： 此引擎已弃用；若希望获得长期支持，请切换到三轴应力控制器。

`Key`(=””)追加到全部文件末尾的字符串，用于命名模拟。

`StabilityCriterion`(=*0。001*)以 三轴压缩引擎::*UnbalancedForce* 为依据的容差，用于判定堆积体是否稳定

`UnbalancedForce`(=*1*)平均合力除以平均接触力

`autoCompressionActivation`(=*true*)从各向同性压实（或卸载状态，如果 sigmaLateralConfine- ment<sigmaIsoCompaction）自动切换到偏应力加载

`autoStopSimulation`(=*false*)当试样达到状态_LIMBO 时停止模拟，或继续运行

`autoUnload`(=*true*)自动从各向同性压实切换到卸载

`boxVolume`
总堆积体积。

`computeStressStrainInterval`(=*10*)

`currentState`(=*1*)
三轴压缩引擎有 5 种可能的状态。 见上文 *wrapper.TriaxialCompressionEngine*

`currentStrainRate`(=*0*)
当前应变率——趋近于 *TriaxialCompressionEngine::strainRate* (./秒)

dead(=否)若为真，此引擎将完全停用；可用于暂时停用引擎，并在之后某个时间点重新启用。

depth(=*0*)盒子尺寸（2 轴）（自动更新）

depth0(=*0*)应变定义的参考尺寸。参见 *TriaxialStressController::depth*

dict（可序列化对象）辐角 *1*）→字
典：返回属性字典。

epsilonMax(=*0。5*)必须停止加载时的轴向变形值

execCount此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所花费的累计时间（纳秒）（仅在 if
O.timingEnabled==True）时使用。

externalWork(=*0*)与边界条件相关的机械功，即 $\int_{\partial\Omega}\mathbf{T}\cdot\mathbf{u}ds$ 边界处的表面牵引力和 $\mathbf{u}$ 位移。（自动更新）

finalMaxMultiplier(=*1。00001*)内部压实时直径的最大乘数（二次精确调整——三轴应力应变控制器*::maxMultiplier* 在初始阶段使用）

fixedPoroCompaction(=否)一种强制设定最终孔隙率的特殊压实类型
TriaxialCompressio-nEngine::fixedPorosity（警告：可能导致不切实际的结果！）

fixedPorosity(=*0*)用户选择的孔隙率值

frictionAngleDegree(=*-1*)偏应力加载前指定的摩擦值

goal1(=*0*)轴 1 上的指定应力/应变率，由以下内容定义：*TriaxialStressController::stressMask*

goal2(=*0*)轴 2 上的指定应力/应变率，由以下内容定义：*TriaxialStressController::stressMask*

goal3(=*0*)轴 3 上的指定应力/应变率，由以下内容定义：*TriaxialStressController::stressMask*

height(=*0*)箱体尺寸（轴 1）（自动更新）

height0(=*0*)应变定义的参考尺寸。参见 *TriaxialStressController::height*

internalCompaction(=*true*)在“外部”（壁面）压实与“内部”（颗粒增长）压实之间切换。

isAxisymetric(=*false*)若为 true，则将 sigma_iso 赋给 sigma1、2 和 3（每次迭代时应用，并覆盖用户设置的 s1、2、3 值）

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

maxMultiplier(=*1.001*)
内部压实时直径的最大乘数（初始快速增加 - *TriaxialStress- Controller::*最终最大倍率 用于第二阶段）

maxStress(=*0*)
模拟期间轴向应力的最大绝对值（用于后处理）

max_vel(=*1*)
允许的最大壁面速度 [米/秒]。如果应力控制器指定的值更高，则此值会覆盖该值。在许多情况下，max_vel 可以设为无穷，但有时将其调低有助于稳定堆积体。根据此值，会依据试样的尺寸为每个轴计算不同的最大值，因此，若每个边界都以其最大速度移动，应变率将是各向同性的（例如，参见 *TriaxialStressController::max_vel1*）。

max_vel1
参见 *TriaxialStressController::max_vel* （自动计算）

max_vel2
参见 *TriaxialStressController::max_vel* （自动计算）

max_vel3
查看 *TriaxialStressController::max_速度* *(自动计算)*

meanStress(=*0*)
堆积体中的平均应力。*(自动更新)*

noFiles(=否)如果为真，则不会生成任何文件（*.xml、*.spheres，.....）

ompThreads(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 (默认值)，线程数通常为 OMP_NUM_THREADS，或由 'yade -jN' 指定的 N（不过，此行为可能取决于引擎）。此属性只影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在组合 *ParallelEngine* 与运行并行区域的引擎时使用；这样会形成嵌套 OMP 循环，且各层的线程数不同。

particlesVolume
颗粒（团簇和动态球体）的总体积。*(自动计算)*

porosity
堆积体的孔隙率，根据 颗粒体积 和 盒体体积计算得出。*(自动更新)*

previousMultiplier(=*1*)*(自动更新)*

previousSigmaIso(=*1*)继承的应力_iso 的前一值（用于检测围压是否被手动更改）

previousState(=*1*)
前一状态（用于检测 .xml 中的状态是否被手动更改）

previousStress(=*0*)*(自动更新)*

radiusControlInterval(=*10*)

setContactProperties(*(三轴压缩引擎)*参数*1*, *(浮点数)*辐角*2*) → 无：为动力体和相对相互作用指定新的摩擦角（度）

sigmaIsoCompaction(=*1*)
压实阶段规定的各向同性压力（< 0 用于实数 - 压缩 - 压实）

sigmaLateralConfinement(=*1*)
偏应力加载过程中的指定围压（< 0 适用于经典压缩算例）；可能与 三轴压缩引擎::压实各向同性应力不同

sigma_iso(=*0*)指定围压应力（参见 :yref:三轴压缩引擎::isAxisymetric‘）

spheresVolume
……的简写形式 *TriaxialStressController::particlesVolume*

stiffnessUpdateInterval(=*10*)
用于测量合成堆积边界刚度的迭代周期，用于应力控制 vo-
控制

strain
向量中的当前应变 (exx,eyy,ezz)。这些值表示真实（对数）应变。

strainDamping(=*0*。*99*)用于使应变率过渡更加平滑的系数。速率达到目标值的方式类似于 d^n 趋近于 0，其中 d 为阻尼系数， n 为步数

strainRate(=*0*)
目标应变率（./s， >0表示压缩）

stress(*(三轴应力控制器)参数1*，*(int)*标识符）→ 三维向量：返回边界‘标识符’上的平均应力。此处，‘标识符’指边界的内部编号，范围为 0 到 5。

stressDamping(=*0.25*)应力控制的壁面阻尼系数 - wallDamping=0 表示（理论上的）理想控制，wallDamping=1表示不发生移动

stressMask(=*7*)
位掩码，用于确定施加的目标值是应力（0 表示无，7 表示全部，1 表示方向 1，5 表示方向 1 和 3 等）还是应变率

testEquilibriumInterval(=*20*)阶段转换检查的区间；大于 1 时可节省计算时间。

thickness(=*-1*)盒子厚度（某些函数需要）

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用且*O.timingEnabled*==True。

translationAxis(=三轴应力控制器::法向量[墙__底部])压缩轴

uniaxialEpsilonCurr(=*1*)受限加载过程中的轴向变形当前值（对应应变[1]）

updateAttrs(*(可序列化对象)参数1*，*(字典)*辐角*2*）→ 无:
根据给定字典更新对象属性

updatePorosity(=*false*)
如果为真，固体体积将更新一次（完成一个计算步后会自动重置为 false），例如用于孔隙率计算。当模拟期间颗粒体积发生变化时（例如颗粒被删除或创建团簇时）可使用。

volumetricStrain(=*0*)体积应变（参见 *TriaxialStressController::strain*）。
（自动更新）

wall_back_activated(=真)
如果为真，则该壁面会根据目标值（应力或应变率）移动。

wall_back_id(=*4*)边界的标识符；坐标 2-（如果在球体之前附加 aabbWalls，则使用默认值即可。）

wall_bottom_activated(=真)
若为 true，此墙将根据目标值（应力或应变率）移动。

`wall_bottom_id`(=*2*)
: 边界标识符；坐标 1（如果在球体之前添加 aabbWalls，则默认值即可球体。）

`wall_front_activated`(=*true*)
: 如果为 true，此墙将根据目标值（应力或应变率）移动。

`wall_front_id`(=*5*)
: 边界标识符；坐标 2+ （如果在球体之前添加 aabbWalls，则默认值即可。）

`wall_left_activated`(=*true*)
: 若为 true，则此墙将根据目标值（应力或应变率）移动。

`wall_left_id`(=*0*)
: 边界的标识符；坐标 0（如果在球体之前添加 aabbWalls，则默认值即可。）

`wall_right_activated`(=*true*)
: 若为 true，则此墙将根据目标值（应力或应变率）移动。

`wall_right_id`(=*1*)边界的标识符；坐标 0+ （如果在球体之前添加 aabbWalls，则默认值即可。）

`wall_top_activated`(=*true*)
: 如果为 true，此墙将根据目标值（应力或应变率）移动。

`wall_top_id`(=*3*)边界的标识符；坐标 1+ （如果在球体之前添加 aabbWalls，则默认值即可。）

`warn`(=*0*)用于仅发送一次弃用警告的计数器

`width`(=*0*)
: 箱体尺寸（0 轴）（自动更新）

`width0`(=*0*)应变定义的参考尺寸。参见 *TriaxialStressController::width*

`class yade.wrapper.TriaxialStressController`(继承自 边界控制器 →全局引擎→引擎 →可序列化对象)

在平行六面体堆积体的某些边界上维持恒定应力或恒定应变率的引擎。对于每个轴，使用 *TriaxialStressController::stressMask*（位掩码）定义应力/应变控制，并通过 goal1、goal2 和 goal3 定义目标值。应变和应力采用连续介质力学的符号约定（拉伸为正）。

注意： 所使用的算法最初是为以下文献中报告的模拟开发的：[Chareyre2002a]和 [Chareyre2005]。随后，它们被移植到 Yade 中，并用于例如 [Kozicki2008],[Scholtes2009b]_、[Jerier2010b]。

`boxVolume`
: 总堆积体积。

`computeStressStrainInterval`(=*10*)

`dead`(=否)如果为 true，该引擎将完全不运行；可用于暂时停用某个引擎，并在稍后时间点重新启用它。

`depth`(=*0*)
: 箱体尺寸（2 轴）（自动更新）

depth0(=0)应变定义的参考尺寸。请参阅 三轴应力控制器::深度

dict((可序列化对象)辐角 1) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所用的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

externalWork(=0)
与边界条件相关的机械功，即 $\int_{\partial\Omega}\mathbf{T}\cdot\mathbf{u}\mathrm{d}s$ 边界处的表面牵引力和 **T**位移。*(自动更新)*

finalMaxMultiplier(=1。00001)内部压实期间的最大直径乘数（二次精确调整——*Tri-axialStressController::maxMultiplier* 在初始阶段使用）

goal1(=0)轴 1 上指定的应力/应变率，定义见*TriaxialStressController::stressMask*

goal2(=0)轴 2 上指定的应力/应变率，定义见*TriaxialStressController::stressMask*

goal3(=0)轴 3 上指定的应力/应变率，定义见*TriaxialStressController::stressMask*

height(=0)箱体尺寸（轴 1）（自动更新）

height0(=0)应变定义的参考尺寸。参见*TriaxialStressController::height*

internalCompaction(=*true*)在“外部”（壁面）和“内部”（颗粒增长）压实之间切换。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，才能直接从 Python 中引用。

maxMultiplier(=1.001)
内部压实期间直径的最大乘数（初始快速增大 - 三轴应力控制器::最终最大倍率 在第二阶段使用）

max_vel(=1)
允许的最大壁面速度 [米/秒]。如果此值更高，则会取代应力控制器设定的值。最大值_速度在许多情况下可以设为无穷大，但有时将其设为有限值有助于稳定堆积体。根据此值，会根据试样的尺寸为每个轴计算不同的最大值，从而使每个边界都以最大速度移动时，应变率保持各向同性（例如参见 三轴应力控制器::最大值_速度1）。

max_vel1
参见三轴应力控制器::最大值_速度 （自动计算）

max_vel2
参见三轴应力控制器::最大值_速度 （自动计算）

max_vel3
参见三轴应力控制器::最大值_速度 *(自动计算)*

meanStress(=0)
堆积体中的平均应力。*(自动更新)*

ompThreads(=-*1*)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性只会影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时，从而产生各层线程数不同的嵌套 OMP 循环。

particlesVolume
颗粒总体积（团簇和动态球体）。（自动计算）

porosity
堆积体的孔隙率，根据颗粒体积和盒体体积计算。（自动更新）

previousMultiplier(=*1*)（自动更新）

previousStress(=*0*)（自动更新）

radiusControlInterval(=*10*)

spheresVolume
.....的简写 *TriaxialStressController::particlesVolume*

stiffnessUpdateInterval(=*10*)用于应力伺服控制时测量合成堆积边界刚度的迭代周期

strain
以向量形式表示的当前应变 (exx,eyy,ezz)。这些值表示真实（对数）应变。

strainDamping(=*0。99*)用于使应变率平滑过渡的系数。应变率达到目标值的方式类似于 d^n 趋近于 0，其中 d 为阻尼系数， n 为步数

strainRate
以向量形式表示的当前应变率 d/dt(exx,eyy,ezz)。

stress(*(TriaxialStressController)arg1*, *(int)id*) → 三维向量：返回边界'id'上的平均应力。此处，'id'指边界的内部编号，范围为 0 到 5。

stressDamping(*arg1*, *arg2*)应力控制的壁面阻尼系数 - wallDamping=0 表示（理论上的）理想控制，wallDamping=1表示无移动

stressMask(=*7*)
确定施加的目标值是应力（0 表示无，7 表示全部，1 表示方向 1，5 表示方向 1 和 3，等等）还是应变率的位掩码

thickness(=*-1*)
盒子厚度（某些函数所需）

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用，否则为空，并且*O.timingEnabled*==True。

updateAttrs(*(可序列化对象)参数1*, *(字典)辐角2*) → None：从给定字典更新对象属性

updatePorosity(=否)
若为 true，固体体积将更新一次（更新后会在一个计算步后自动重置为 false），例如用于孔隙率计算。当模拟过程中颗粒体积发生变化时（例如颗粒被删除或创建团簇时），可以使用此功能。

volumetricStrain(=*0*)体积应变（参见*TriaxialStressController::strain*）。（自动更新）
wall_back_activated(=*true*)若为 true，此墙将根据目标值（应力或应变率）移动。
wall_back_id(=*4*)边界的标识符；坐标 2（若在球体之前添加 aabbWalls，则默认值即可。）

wall_bottom_activated(=*true*)若为 true，此墙将根据目标值（应力或应变率）移动。

wall_bottom_id(=*2*)边界的标识符；坐标 1（如果在球体之前追加 aabbWalls，则默认值即可。）

wall_front_activated(=*true*)若为 true，此墙将根据目标值（应力或应变率）移动。

wall_front_id(=*5*)边界的标识符；坐标 $2+$（如果在球体之前追加 aabbWalls，则默认值即可。）

wall_left_activated(=*true*)若为 true，此墙将根据目标值（应力或应变率）移动。

wall_left_id(=*0*)边界的标识符；坐标 0（如果在球体之前添加 aabbWalls，则默认值即可。）

wall_right_activated(=*true*)若为 true，此墙将根据目标值（应力或应变率）移动。

wall_right_id(=*1*)边界的标识符；坐标 $0+$（如果在球体之前添加 aabbWalls，则默认值即可。）

wall_top_activated(=*true*)若为 true，此墙将根据目标值（应力或应变率）移动。
wall_top_id(=*3*)边界的标识符；坐标 $1+$（如果在球体之前添加 aabbWalls，则默认值即可。）
width(=*0*)箱体尺寸（0 轴）（自动更新）**width0**(=*0*)应变定义的参考尺寸。参见*TriaxialStressController::width*

class yade.wrapper.UniaxialStrainer(继承 边界控制器 → 全局引擎 → 引擎 →可序列化对象)以给定应变率沿相反方向轴向移动两组刚体。**absSpeed**(=非数)或者，可以指定边界运动的绝对速度；此设置仅在开始时生效，且仅当未设置 strainRate 时有效；在模拟过程中直接更改 absSpeed 不会产生任何效果。 [毫秒 [1]]**active**(=是)此引擎是否已启用**asymmetry**(=*0*, 对称)如果为 0，则负向 ID 和正向 ID 的应变对称；如果为 1（或 -1），则仅对正向 ID 施加应变，负向 ID 不移动（反之亦然）

avgStress(=*0*)当前平均应力 *(自动更新)* [帕]

axis(=*2*)受应变的轴（0、1、2 分别对应 x、y、z）

blockDisplacements(=*false*)是否阻止边界体沿受应变轴垂直方向的位移，或允许其自由移动

blockRotations(=*false*)是否阻止边界体旋转。

crossSectionArea(=非数)垂直于受应变轴的横截面；必须显式给出 [m^2]

currentStrainRate(=非数)当前应变率（自动更新）。*(自动更新)*

dead(=否)若为 true，此引擎将完全不运行；可用于暂时停用引擎，并仅在之后某个时刻重新启用。

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

execCount
: 此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
: 此引擎运行所用的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

idleIterations(=*0*)
: 达到 stopStrain 后，在不产生应变活动的情况下继续运行的迭代次数 d

initAccelTime(=*-200*)应变达到请求值所需的时间（线性插值）。如果为负值，则时间为 dt*(-initAccelTime)，其中 dt 是第一次迭代时的时间步长。 [s]

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

limitStrain(=*0*, 已禁用)达到此应变值时，反转应变方向（突然反转，不经过过渡）。值为 0 时无效。

negIds(=未初始化)
: 施加应变的刚体（沿轴的负向端）

notYetReversed(=是)
: 标志应变方向是否已反转（仅供内部使用）。

ompThreads(=*-1*)
: 引擎中使用的线程数。如果 ompThreads<0 （默认值），则该数量通常为 OMP_线程数_，或由‘yade -jN’定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响其代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或用于将 *ParallelEngine* 与运行并行区域的引擎组合使用，从而产生每一层线程数不同的嵌套 OMP 循环。

originalLength(=非数)
: 应变开始前参考刚体沿轴方向的距离（自动计算 o-
（自动） [m]

posIds(=未初始化)应变将施加于这些刚体（沿轴正向的一端）setSpeeds(=否)是否应在开始时直接重置速度，而不是逐步提高应变率？stopStrain(=非数)暂停模拟时的应变值；默认不活动（非数）；必须从下方达到（按绝对值计）strain(=*0*)当前应变值，伸长量/原始长度 *(自动更新)* [-]strainRate(= 非数)应变率，从 0 开始，线性提高至应变率。 [-]
stressUpdateInterval(=*10*)重新计算支撑体应力的频率。timingDeltas

引擎本身内部的详细计时信息。除非在源码中启用，否则为空 ce
代码和 *O.timingEnabled*==True。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) →
None：根据给定字典更新对象属性

碰撞器

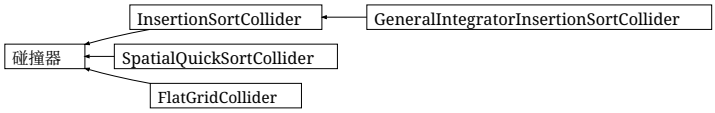


图 28：碰撞器的继承图。另请参阅：*FlatGridCollider*、*GeneralIntegratorInsertionSortCol-lider*、*InsertionSortCollider*、*SpatialQuickSortCollider*。

class yade.wrapper.Collider(继承自 全局引擎 →引擎 → 可序列化对象)用于查找刚体之间空间碰撞的抽象类。

特殊构造函数

派生的碰撞器（除非重写 pyHandleCustomCtorArgs）可以接收 *BoundFunc-tors* 列表，用于初始化内部的 *boundDispatcher* 实例。

avoidSelfInteractionMask(=*0*)
此掩码用于避免粒子组内部发生相互作用。为此，颗粒必须具有完全相同的掩码，并且该掩码在二进制表示中应至少有一位与此*avoidSelfInteractionMask* 掩码相同。

boundDispatcher(=新建 *BoundDispatcher*)
BoundDispatcher 对象，用于根据碰撞器的请求按需创建 边界。

dead(=*false*)
如果为 true，此引擎将完全不会运行；可用于暂时停用引擎并仅在稍后的某个时间点重新启用它。

dict(*(*可序列化对象*)*辐角*1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
: 此引擎运行所用的累计时间（单位：纳秒；仅在*O.timingEnabled*==True时使用）。

label(=未初始化)
: 此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

ompThreads(=*-1*)
: 引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性只会影响其代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在 *ParallelEngine* 与运行并行区域的引擎组合使用时，从而形成每一层线程数各不相同的嵌套 OMP 循环。

timingDeltas
: 引擎本身内部的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(Serializable)arg1*, *(dict)arg2*) → None:
从给定字典更新对象属性

class yade.wrapper.FlatGridCollider(继承 碰撞器 → 全局引擎 → 引擎 → 可序列化对象)

未优化的网格碰撞器，将网格存储为稠密一维数组。每个刚体都会被分配到一个或多个单元格，这些单元格在*aabbMin* 和*aabbMax*之间按规则网格排列，单元格尺寸为步 （各方向相同）。对于超出范围（*aabbMin*，*aabbMax*）的刚体，会妥善处理并分配到最近的单元格（这会产生虚假的潜在相互作用）。*verletDist* 决定每个刚体的扩张幅度，以避免在每一步都进行碰撞检测。

注意： 此碰撞器将所有单元格存储在线性内存数组中，因此对于稀疏模拟而言，内存效率较低。

警告： 对象 *Body::bound* 未使用，*BoundFunctors* 也未使用：将单元格分配给刚体的逻辑在内部是硬编码的。目前支持的形状 有：球体。

注意： 尚未处理周期性边界。

---

aabbMax(=*Vector3r::Zero()*)
: 网格的上角点（近似值，可能会向上舍入到 *minStep*。

aabbMin(=*Vector3r::Zero()*)网
格的下角点。

avoidSelfInteractionMask(=*0*)
: 此掩码用于避免粒子组内部的相互作用。为此，颗粒必须具有完全相同的掩码，并且该掩码在其二进制表示方式中应与此避免自相互作用掩码 有一个共同的位。

boundDispatcher(=新建 *BoundDispatcher*)
: *BoundDispatcher* 对象，用于根据碰撞器的请求按需创建边界。

dead(=*false*)
: 如果为 true，此引擎将完全不运行；可用于暂时停用某个引擎，并仅在之后的某个时间点将其重新启用。

dict(*(可序列化对象)*辐角 *1*) →字典:
返回属性字典。

execCount
: 此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
: 此引擎运行所用的累计时间（纳秒；仅在*O.timingEnabled*==True）

label(=已取消初始化)
: 此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

ompThreads(=*-1*)
: 引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如*InteractionLoop*）。此属性主要用于实验，或在将*ParallelEngine* 与运行并行区域的引擎结合使用时，从而产生每一层线程数不同的嵌套 OMP 循环。

step(=*0*)
: 网格中的步长（单元格尺寸）

timingDeltas
: 引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)arg1*，*(字典)arg2*) → None：根据给定的字典更新对象属性

verletDist(=*0*)每个颗粒所占空间的扩大量；这样可以避免每一步都运行碰撞检测。

class yade.wrapper.GeneralIntegratorInsertionSortCollider(继承 *InsertionSortCollider* →碰撞器 →全局引擎 → 引擎 →可序列化对象)

此类是 InsertionSortCollider 的自适应版本，并将碰撞器算法对 NewtonIntegrator 的依赖改为更通用的 Integrator 接口。

allowBiggerThanPeriod
: 如果为 true，将停用对刚体尺寸的检查；即使发现刚体尺寸大于周期，模拟仍会正常运行。当周期性问题中包含例如用 wall/box/facet 建模的地板时，这可能会很有用。如果要修改此标志，请务必清楚自己在做什么。如果一个大型刚体移出 (0,0,0) 周期，结果将无法确定。

avoidSelfInteractionMask(=*0*)
: 此掩码用于避免粒子组内部的相互作用。为此，粒子必须具有完全相同的掩码，并且该掩码必须与此避免自相互作用掩码 在二进制表示中至少有一个相同的位。

boundDispatcher(=新的 *BoundDispatcher*)
: 用于根据碰撞器的请求按需创建边界的 BoundDispatcher 对象。

dead(=*false*)
: 如果为 true，此引擎将完全不运行；可用来暂时停用引擎，并仅在稍后重新启用它。

dict(*(可序列化对象)*辐角 *1*) →字典：返回属性字典。

doSort(=*false*)
: 强制对相互作用进行重新排序。

dumpBounds(*(InsertionSortCollider)*辐角 *1*) →元 例：
返回内部排序数据的表示方式。对于 3 个坐标轴，其格式为 `([...],[...],[...])`，每个.....都是由条目（边界）组成的列表。条目是一个元组，包含以下项目：

- 坐标（浮点数）
- 刚体 ID（整数），但对于负边界，其值取反
- 周期编号（整数），当碰撞器处于周期状态时。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==`True`时使用）。

execTime
此引擎运行所用的累计时间（纳秒，仅在*O.timingEnabled*==`True`时使用）。

fastestBodyMaxDist(*=0*)
自上次运行以来移动最快的物体的归一化最大位移；如果 >= 1，我们可以移出 et 包围盒，并触发完整运行。（自动更新）

isActivated(*(InsertionSortCollider)*辐角 *1*) →布尔值：若碰撞器需要在下一次迭代时执行，则返回 true。

keepListsShort(*=false*)
若为 true，则从列表中移除不存在或无界刚体的边界（自动更新）；在 MPI 模式下以及使用 *BodyCon-tainer.enableRedirection‘=True. :ydefault:‘false* 擦除刚体时，会自动设为 true

label(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

minSweepDistFactor(*=0.1*)
扩展所有包围盒的最小距离；当该值低于（minSweepDistFactor x verletDist）时，会覆盖计算得出的 verlet-Dist 值。

newton(=共享_指针<*NewtonIntegrator*>*()*)指向活动状态对象的引用*Newton* 积分器。（自动更新）

numAction(*=0*)
碰撞检测的累计次数。

numReinit(*=0*)
边界数组重新初始化的累计次数。

ompThreads(*=-1*)
引擎中要使用的线程数。如果 ompThreads<0 (默认值)，线程数通常为 OMP_NUM_THREADS，或者由 ‘yade -jN’ 定义的数量 N（不过，此行为可能取决于引擎）。此属性只会影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会形成每一层线程数不同的嵌套 OMP 循环。

overlapTolerance(*=1e-7*)
确定重叠时的容差。在少数情况下，代码的不同部分可能会对重叠得出不一致的结果；spatialOverlapPeri 产生的假阴性可能会导致接触检测中出现严重缺陷（假阳性则无害）。此容差用于避免假阴性，其数值可理解为相对于 1 的比例（即与颗粒尺寸或任何其他参考长度无关）。默认值应该合适。

periodic
碰撞器是否处于周期模式（只读；用于调试）*(自动更新)*

smartInsertErase(*=false*)
使用针对大量插入/删除优化的算法（避免使用 initSort）——实验性功能。

sortAxis(=*0*)
用于初始接触检测的轴。

sortThenCollide(=否)
将排序阶段与碰撞阶段分开；这样会慢很多，但每一步都会处理全部相互作用；这实际上使碰撞器变为非持久性，不会记住上一个状态。（默认行为基于这样一个事实：插入排序期间的逆序是指刚开始或刚停止重叠的包围盒，并且只处理这些逆序；这使碰撞器的效率高得多。）

strideActive
跨步是否处于活动状态（只读；用于调试）。*(自动更新)*

targetInterv(=*100*)
（实验性）边界更新之间的目标迭代次数，用于为较慢的颗粒定义较小的扫描距离（如果 >0），否则始终使用 1*verletDist。在慢速刚体与快速刚体之间速度差异很大的模拟中很有用。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用，否则为空 e
代码和 *O.timingEnabled*==True。

updateAttrs(*(可序列化对象)参数1，(字典)辐角2*) →
None：根据给定字典更新对象属性

updatingDispFactor(=*-1*)（
实验性的 ）
用于触发边界更新的位移因子：仅当 updatingDispFactor*disp>sweepDist 满足 >0 时才更新边界，否则更新全部边界。

verletDist(=*-.5*，自动初始化)
用于扩大粒子边界的长度，以避免每一步都运行碰撞器。为零时禁用步长。负值将触发自动计算，因此实数值将为 *verletDist* × 最小球形粒子半径；如果不存在球形颗粒，则禁用此功能。当*InsertionSortCollider::targetInterv* 为 > 0时。

class yade.wrapper.**InsertionSortCollider**(继承自 碰撞器 → 全局引擎 → 引擎 →可序列化对象)

复杂度为 O(n log(n)) 的碰撞器，使用 *Aabb* 表示边界。
在初始步，刚体的边界（沿 排序轴）首先沿此（排序轴）使用 std::sort 排序，然后进行碰撞检测。初始排序的复杂度为 $O(n^2)$，相关信息请参阅 碰撞器性能 （可使用 examples/collider-perf 中的脚本进行测量）。

插入排序用于对上一轮迭代中已预排序的边界列表进行排序；每次逆序都会调用 checkOverlap，随后处理重叠（必要时创建相互作用）或无重叠的情况（如果相互作用仅为潜在相互作用，则将其删除）。

没有包围体的刚体（例如团簇）会得到妥善处理，且绝不会发生碰撞。已删除的主体也会得到妥善处理。
此碰撞器用于处理周期性边界条件，但存在一些限制，主要包括：

1. 刚体在相应维度上的 Aabb 尺寸不能大于单元半尺寸。若超过此限制并发生相互作用，就会引发异常。 `allowBiggerThanPeriod` 提供了一种显式绕过此限制的方法：启用该选项后，可插入一个非常大的盒子作为地板（例如参见 examples/periodicSandPile.py）。

2. 刚体在一步内的移动距离不能超过单元的距离；否则意味着模拟出现了数值爆炸，而且这种情况只有在某些情况下才能检测到。

步长 可通过扩大颗粒的边界、追踪其位移，并且仅在颗粒可能超出边界时才重新运行碰撞器，从而避免每一步都运行碰撞器（简要说明和背景请参见 Verlet 列表 ）。这需要 *NewtonIntegrator* 和 *BoundDispatcher*的配合，它们会自动从引擎中查找（如果未找到，则会抛出异常）。

如果要使用步长，请将 verletDist （边界在所有方向上扩大的长度）设为某个值，例如设为 0.05 × 个典型颗粒半径。此参数体现了以下权衡：潜在相互作用较多（较少运行碰撞器，但精确求解相互作用所需的阶段更长）与潜在相互作用较少（更频繁地运行碰撞器，但相互作用的求解精度较低）；这主要取决于堆积密度和粒子半径分布。

如果 targetInterv 为 >1，则并非所有颗粒都会将其边界扩大 verletDist；相反，它们的边界会增加一段长度，以便在 targetInterv次迭代后触发新的碰撞检测，假设它们以近似恒定的速度移动。在这种方法中，理想情况下，所有颗粒都应在同一次迭代时到达其边界。当然，只要速度随时间波动，情况就不再如此。 *Bound::sweepLength* 根据碰撞器最近两次运行期间记录的位移进行调整。在这种情况下， verletDist 定义最大扫描长度。

allowBiggerThanPeriod
如果为 true，将禁用对刚体尺寸的测试，即使发现尺寸大于周期的刚体，模拟仍会正常运行。当周期性问题包含例如由墙、盒子或面片构成的地板模型时，这可能很有用。如果要修改此标志，请务必确认自己了解其作用。当一个大型刚体移出 (0,0,0) 周期时，结果是未定义的。

avoidSelfInteractionMask(=*0*)
此掩码用于避免粒子组内部发生相互作用。为此，组内颗粒必须具有完全相同的掩码，并且该掩码应与*avoidSelfInteractionMask* 的二进制表示至少共享一位。

boundDispatcher(=新建 *BoundDispatcher*)
BoundDispatcher 对象，用于在碰撞器请求时按需创建边界。

dead(=否)
如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并仅在稍后的某个时刻重新启用它。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

doSort(=否)
强制进行相互作用重新排序。

dumpBounds(*(InsertionSortCollider)*辐角 *1*) →元组:
返回内部排序数据的表示方式。对于 ([...],[...],[...]) 个坐标轴，其格式为：每个.....都是由条目（边界）组成的列表。该条目是一个包含以下项目的元组：

- 坐标（浮点数）
- 刚体 ID（整数），但对于负边界取反
- 周期状态下的周期编号（int）。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所用的累计时间（纳秒；仅在*O.timingEnabled*==True 时使用）。

fastestBodyMaxDist(=*0*)
自上次运行以来移动最快的物体的归一化最大位移；如果 >= 1，则可能会移出包围盒，并触发完整运行。*(*自动更新*)*

isActivated(*(InsertionSortCollider)*辐角 *1*) →bool: 如果碰撞器需要
在下一次迭代中执行，则返回 true。

keepListsShort(=*false*)
若为 true，则从列表中移除不存在或无界刚体的边界（自动更新）；在 MPI 模式下，以及使用 *BodyCon-tainer.enableRedirection*‘ 擦除刚体时，会自动设为 true=*True. :ydefault:‘false*

`label`(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

`minSweepDistFactor`(=*0.1*)
扩大全部包围盒的最小距离；当 verlet-Dist 小于 (minSweepDistFactor x verletDist) 时，使用计算值的上限。

`newton`(=共享_指针<*NewtonIntegrator*>())对活动状态的引用
Newton 积分器。 *(自动更新)*

`numAction`(=*0*)
碰撞检测的累计次数。

`numReinit`(=*0*)
边界数组重新初始化的累计次数。

`ompThreads`(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能因引擎而异）。此属性只会影响其代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合时使用；这会产生嵌套 OMP 循环，并且每一层的线程数不同。

`overlapTolerance`(=*1e-7*)
确定重叠时所采用的容差。在少数情况下，代码的不同部分可能会导致重叠判断结果不一致；spatialOverlapPeri 产生的假阴性可能会在接触检测中引发严重缺陷（假阳性则无害）。此容差用于避免假阴性，其数值可视为相对于 1 的值（即与颗粒尺寸或其他任何参考长度无关）。默认值应该合适。

`periodic`
碰撞器是否处于周期模式（只读；用于调试）*(自动更新)*

`smartInsertErase`(=否)
使用针对大量插入/删除优化的算法（避免使用 initSort）——实验性功能。

`sortAxis`(=*0*)
初始接触检测所用的轴。

`sortThenCollide`(=否)
将排序阶段与碰撞阶段分开；速度会慢得多，但每一步都会处理全部相互作用；这实际上会使碰撞器变为非持久型，不再记住上一个状态。（默认行为基于这样一个事实：插入排序期间的逆序对应于刚开始或刚结束重叠的包围盒，并且只处理这些情况；这样能显著提高碰撞器的效率。）

`strideActive`
跨步是否处于活动状态（只读；用于调试）。*(自动更新)*

`targetInterv`(=*100*)
（实验性）边界更新之间的目标迭代次数；如果 >0，用于为较慢的颗粒定义较小的扫描距离，否则始终使用 1*verletDist。适用于慢速刚体与快速刚体之间速度差异很大的模拟。

`timingDeltas`
引擎本身内部的详细计时信息。除非在源代码中启用 *O.timingEnabled*==`True`，否则为空。

`updateAttrs`(*(可序列化对象)参数1*, *(字典)辐角 2*) → None：根据给定字典更新对象属性

`updatingDispFactor`(=*-1*)
（实验性）用于触发边界更新的位移因子：仅当 updatingDispFactor*disp>sweepDist 满足 >0 时才更新边界，否则更新全部边界。

verletDist(=*-.5*，自动初始化)
: 用于扩展粒子边界的长度，以避免每一步都运行碰撞器。为零时禁用步长。负值将触发自动计算，因此实数值将为 *verletDist* × 最小球形粒子半径；如果没有球形颗粒，则会禁用。添加到一个边界的实际长度可能只是 verletDist 的一部分，具体取决于 *InsertionSortCollider::targetInterv* 是否 > 0。

class yade.wrapper.SpatialQuickSortCollider(继承自 碰撞器 → 全局引擎 → 引擎→ 可序列化对象)每一步沿坐标轴使用快速排序的碰撞器，使用 *Aabb* 边界。

其性能低于 *InsertionSortCollider* (参见碰撞器性能)，但该算法足够简单，适合用于检查其他碰撞器的正确性。

avoidSelfInteractionMask(=*0*)
: 此掩码用于避免粒子组内部发生相互作用。为此，粒子必须具有完全相同的掩码，并且该掩码的二进制表示必须与此避免自相互作用掩码 至少共享一个位。

boundDispatcher(=新建 *BoundDispatcher*)
: *BoundDispatcher* 用于按需创建边界的对象，以响应碰撞器的请求。

dead(=否)
: 如果为 true，此引擎将完全停止运行；可用于暂时停用引擎，并在之后的某个时间点将其重新启用。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

execCount
: 此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
: 此引擎运行所用的累计时间（纳秒），仅当*O.timingEnabled*==True时使用。

label(=未初始化)
: 此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

ompThreads(=*-1*)
: 引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_数量_线程数，或由'yade -jN'定义的数量 N（具体行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与 *ParallelEngine* 结合使用包含并行区域的引擎，从而在不同层级形成线程数各异的嵌套 OMP 循环。

timingDeltas
: 引擎本身内部的详细计时信息。除非在源代码中启用且满足*O.timingEnabled*==True，否则为空。

updateAttrs(*(*可序列化对象*)arg1*，*(*字典*)arg2*) → 无:
从给定字典更新对象属性

场施加器

class yade.wrapper.FieldApplier(继承自 全局引擎 → 引擎 → 可序列化对象)用于将力文件施加到颗粒上的引擎基类。请勿直接使用。

dead(=否)
: 如果为真，此引擎将完全不会运行；可用于暂时停用引擎，并仅在之后的某个时间点重新启用它。

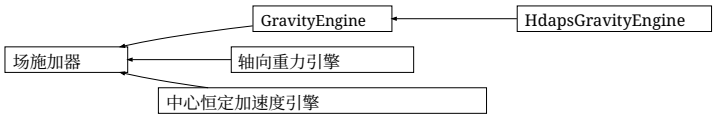


图 29：场施加器的继承图。另请参阅：轴向重力引擎、中央恒定加速度引擎、*GravityEngine*、*HdapsGravityEngine*。

`dict`(*(可序列化对象)辐角 1*) →字典：
返回属性字典。

`execCount`
此引擎的累计运行次数（仅在 *O.timingEnabled*==`True` 时使用）。

`execTime`
此引擎运行所耗费的累计时间（纳秒）（仅在 if *O.timingEnabled*==`True`）时使用。

`label`(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

`ompThreads`(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性仅对代码中包含 OpenMP 并行区域的引擎有效（例如，*InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会产生嵌套 OMP 循环，且每一层的线程数不同。

`timingDeltas`
引擎本身内部的详细计时信息。除非在源代码中启用 *O.timingEnabled*==`True`，否则为空。

`updateAttrs`（可序列化对象）参数*1*，（字典）辐角 *2*) → None：根据给定字典更新对象属性

`class yade.wrapper.AxialGravityEngine`(继承 场施加器 → 全局引擎 → 引擎 →可序列化对象)

施加加速度（与距离无关）指向某个轴。

`acceleration`(=*0*)
加速度大小 [kgms ^{2}]

`axisDirection`(=*Vector3r::UnitX()*)重力轴的方向（将自动归一化）

`axisPoint`(=*Vector3r::Zero()*)轴经过的点。

`dead`(=否)
若为真，此引擎将完全不运行；可用于暂时停用引擎
并仅在稍后的某个时间点重新启用它。

`dict`(*(可序列化对象)辐角 1*) →字典：
返回属性字典。

`execCount`
此引擎的累计运行次数（仅在 *O.timingEnabled*==`True` 时使用）。

`execTime`
此引擎运行所花费的累计纳秒时间（仅在*O.timingEnabled*==`True`时使用）。

label(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

mask(=*0*)
如果定义了掩码，则此引擎只会影响具有相应组掩码的刚体。如果为 0，则会影响全部刚体。

ompThreads(=*-1*)
引擎使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_数量_THREADS，或由'yade -jN'定义的数量 N（不过此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合时，此时不同层级的嵌套 OMP 循环会使用不同的线程数。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无:
从给定字典更新对象属性

class yade.wrapper.CentralConstantAccelerationEngine(继承 场施加器 → 全局引擎 → 引擎 → 可序列化对象)

引擎对全部刚体施加恒定加速度，使其朝向中心体，而不考虑它们之间的距离。

accel(=*0*)
加速度大小 [kgms ^{2}]

centralBody(=*Body::ID*_无)
该刚体 是所有其他刚体被吸引的目标。

dead(=*false*)若为 true，此引擎将完全不会运行；可用于暂时停用引擎，并仅在之后的某个时刻将其重新启用。

dict(*(Serializable)arg1*) →字典：返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所花费时间的累计纳秒数（仅在*O.timingEnabled*==True时使用）。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

mask(=*0*)
如果定义了掩码，则此引擎只会影响具有相应组掩码的刚体。如果为 0，则会影响全部刚体。

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或者由"yade -jN"定义的 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时；这会导致嵌套 OMP 循环的不同层级具有不同的线程数。

reciprocal(=*false*)如果为 true，则也会对中心体施加加速度。

timingDeltas
: 引擎本身的详细计时信息。除非在源代码中启用此功能并设置 *O.timingEnabled*==True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → None：根据给定字典更新对象属性

class yade.wrapper.GravityEngine(继承 场施加器 → 全局引擎 → 引擎 → *Serializ- able*)将恒定加速度施加于全部刚体的引擎。已弃用，除非需要能量跟踪或使用 groupMask 进行选择性重力施加，否则请改用 *Newton::gravity* 。

dead(=*false*)
: 如果为 true，此引擎将完全不运行；可用来暂时停用引擎，并在之后的某个时刻重新启用。

dict(*(*可序列化对象*)*辐角*1*) →字典：
返回属性字典。

execCount
: 此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
: 此引擎运行所用的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

gravity(=*Vector3r::Zero()*)
加速度 [kgms ^{2}]

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

mask(=*0*)
: 如果定义了掩码，则只有具有相应组掩码的刚体会受到此引擎的影响。如果为 0，则所有刚体都会受到影响。

ompThreads(=*-1*)
: 引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或者是由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会形成嵌套 OMP 循环，且每一层的线程数不同。

timingDeltas
: 引擎本身内部计时的详细信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*) → 无：根据给定字典更新对象属性

warnOnce(=*true*)用于弃用警告，仅提示一次。

class yade.wrapper.HdapsGravityEngine(继承自 *GravityEngine* → 场施加器 → 全局引擎 → 引擎 → 可序列化对象)读取 ThinkPad 笔记本电脑中的加速度计（HDAPS 并据此在模拟中设置重力。此代码借鉴自 hdaps-gl 。示例请参见 scripts/test/hdaps.py 。

accel(=*Vector2i::Zero()*)从 sysfs 文件读取

`calibrate`(=*Vector2i::Zero()*)
零向量位置；如果为非数，则从 *hdapsDir* / 校准中读取。

`calibrated`(=否)
是否已更新 校准 。除非同时为 校准提供有意义的值，否则不要手动设为 `True` 。

`dead`(=否)
如果为 true，此引擎将完全不运行；可用来暂时停用某个引擎，并在之后的某个时间点重新启用它。

`dict`(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

`execCount`
此引擎的累计运行次数（仅在*O.timingEnabled*==`True`时使用）。

`execTime`
此引擎运行所用的累计时间（以纳秒为单位；仅在*O.timingEnabled*==`True`时使用）。

`gravity`(=*Vector3r::Zero()*)
加速度 [kgms ^{2}]

`hdapsDir`(=*"/sys/devices/platform/hdaps"*)Hdaps 目录；包含 `position` （含加速度计读数）和 `calibration` （零加速度）。

`label`(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

`mask`(=*0*)
如果定义了掩码，则只有具有相应组掩码的刚体会受此引擎影响。如果为 0，则所有刚体都会受到影响。

`msecUpdate`(=*50*)
更新读数的频率。

`ompThreads`(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会产生嵌套 OMP 循环，各层的线程数不同。

`timingDeltas`
引擎本身内部的详细计时信息。除非在源代码中启用，并且 *O.timingEnabled*==`True`，否则为空。

`updateAttrs`(*(*可序列化对象*)arg1*, *(*字典*)arg2*) →
None：根据给定字典更新对象属性

`updateThreshold`(=*4*)
更新重力前从文件读取读数的最小差异，以避免抖动。

`warnOnce`(=*true*)
用于仅显示一次的弃用警告。

`zeroGravity`(=*Vector3r(0, 0, -1)*)加速度计处于水平（零）
位置时的重力。

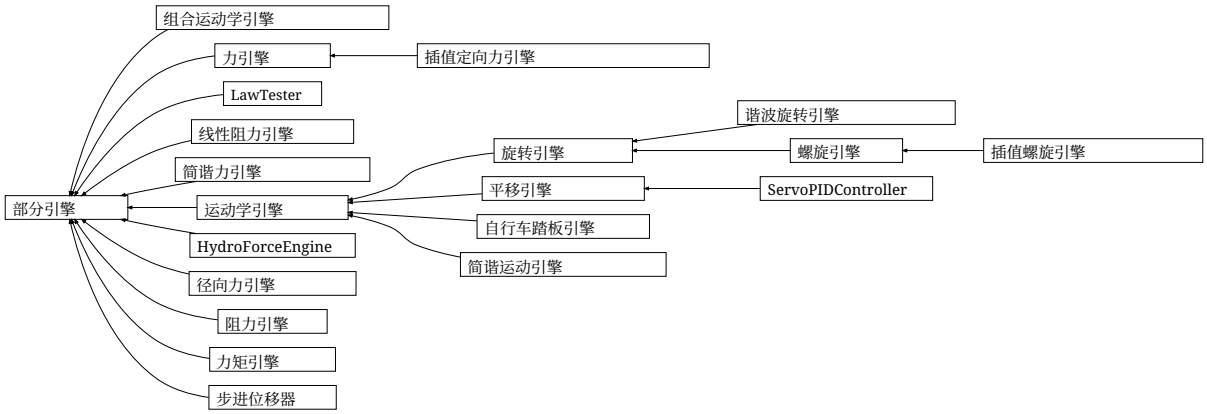


图 *30*：部分引擎的继承图。另请参阅：自行车踏板引擎、*CombinedKinematicEngine*、阻力引擎、力引擎、简谐力引擎、简谐运动引擎、*HarmonicRotationEngine*、螺旋引擎、*HydroForceEngine*、插值定向力引擎、插值螺旋引擎、运动学引擎、*LawTester*、线性阻力引擎、径向力引擎、旋转引擎、*ServoPIDController*、步进位移器、力矩引擎、平移引擎。

2.3.4 部分引擎

class yade.wrapper.**PartialEngine**(继承 引擎 → 可序列化对象)

仅作用于模拟中特定刚体的引擎，即作用于*ids* 属性中定义的刚体。另请参阅全局引擎。

dead(=否)
: 如果为真，则此引擎将完全不运行；可用于暂时停用引擎，并在稍后的某个时间点重新启用它。

dict(*(可序列化对象)*辐角 *1*) →字典:

返回属性字典。

execCount
: 此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
: 此引擎运行所用的累计时间（以纳秒为单位，仅在*O.timingEnabled*==True时使用）。

ids(=未初始化)标识符 受此部分引擎影响的刚体列表。

label(=未初始化)
: 此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

ompThreads(=*-1*)
: 引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会产生嵌套 OMP 循环，各层的线程数不同。

timingDeltas
: 引擎本身内部的详细计时信息。除非在源代码中启用并且 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)*参数*1*，*(字典)*辐角 *2*) → 无:

根据给定字典更新对象属性

class yade.wrapper.BicyclePedalEngine(继承 运动学引擎 → 部分引擎 → 引擎 → 可序列化对象)

对 bicycle pedal 应用线性运动的引擎，例如，使点绕轴移动而不旋转

angularVelocity(=0)角速度。 [弧度/秒]

dead(=否)若为 true，此引擎将完全不运行；可用于暂时停用某个引擎，并在之后某个时刻重新启用。

dict((可序列化对象)辐角) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所用的纳秒数累计时间（仅在*O.timingEnabled*==True时使用）。

fi(=*Mathr::PI/2*。0)初始相位 [弧度]

ids(=未初始化)标识符受此部分引擎影响的刚体列表。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

ompThreads(=-1)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_数量_线程数，或由“yade -jN”定义的 N（具体行为可能取决于引擎）。此属性仅影响代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时，以便在不同层级使用不同线程数运行嵌套 OMP 循环。

radius(=-1.0)旋转半径。 [m]

rotationAxis(=*Vector3r::UnitX()*)旋转轴（方向）；将自动归一化。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs((可序列化对象)*arg1*，(字典)*arg2*) →
None：根据给定字典更新对象属性

class yade.wrapper.CombinedKinematicEngine(继承 部分引擎 →引擎 →可序列化对象)

引擎使用一个对象（comb）执行操作，该对象将 ids 操作的是组合中第一个引擎的操作对象（自动分配）。

comb(=未初始化)
将由此引擎组合的运动学引擎，按给定顺序运行。

dead(=否)
若为 true，则此引擎完全不会运行；可用于暂时停用某个引擎并仅在稍后的某个时间点将其恢复。

我应用程序 运动学引擎 执行操作 ids 操作的是组合中第一个引擎的操作对象 构造 序 ed

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

execCount
: 此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
: 此引擎运行所耗费的累计时间（以纳秒为单位，仅在 if *O.timingEnabled*==True 时使用）。

ids(=已取消初始化)
: 标识符 受此部分引擎影响的刚体列表。

label(=未初始化)
: 此对象的文本标签；必须是有效的 Python 标识符，以便可以直接在 Python 中引用它。

ompThreads(=*-1*)
: 引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（具体行为可能取决于引擎）。此属性只影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会产生嵌套 OMP 循环，且每一层的线程数不同。

timingDeltas
: 引擎本身内部的详细计时信息。除非在源代码中启用并设置 *O.timingEnabled*==True，否则为空。

updateAttrs（可序列化对象）参数*1*，（字典）辐角 *2*) → None：根据给定字典更新对象属性

class yade.wrapper.DragEngine(继承部分引擎 → 引擎 → 可序列化对象)
在每一步对部分颗粒施加阻力，使其线速度成比例地减小。施加的力为

$$F_d = -\frac{v}{|v|}\frac{1}{2}\rho|v|^2 C_d A$$

其中 ρ 为介质密度（密度），v 为颗粒速度，A 为颗粒投影面积（圆盘），C_d 为阻力系数（球体的阻力系数为 0.47），

注意：阻力仅作用于 ids 中列出的球形颗粒。

Cd(=*0.47*)
: 阻力系数 <http://en.wikipedia.org/wiki/Drag_系数>‘_。

Rho(=*1.225*)
: 介质（流体或空气）的密度，默认为空气的密度。

dead(=*false*)
: 如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并仅在之后的某个时间点重新启用它。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

execCount
: 此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
: 此引擎运行所用的累计时间（纳秒），仅在*O.timingEnabled*==True时使用。

ids(=已取消初始化)
标识符 受此部分引擎影响的刚体列表。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

ompThreads(=-1)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与 *ParallelEngine* 结合使用包含并行区域的引擎，从而形成嵌套 OMP 循环，并使各层的线程数不同。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) →
None：根据给定字典更新对象属性

class yade.wrapper.ForceEngine(继承 部分引擎 → 引擎 → 可序列化对象)在每一步对一些颗粒施加接触力。

dead(=*false*)如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并仅在之后的某个时刻将其重新启用。

dict(*(可序列化对象)辐角1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True时使用）。

execTime
此引擎运行所用的纳秒数累计时间（仅在*O.timingEnabled*==True时使用）。

force(=*Vector3r::Zero()*)要施加的力。

ids(=未初始化)标识符 受此部分引擎影响的刚体列表。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

ompThreads(=-1)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与 *ParallelEngine* 结合使用包含并行区域的引擎，从而形成嵌套 OMP 循环，使每一层的线程数不同。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用并设置 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) →
None：根据给定字典更新对象属性

class yade.wrapper.HarmonicForceEngine(继承 部分引擎 →引擎 →可序列化对象)

此引擎会向一组刚体施加谐波（正弦）力。它与 *Harmonic-MotionEngine* 完全相同，但指定的是力振幅而非运动；另请参阅简谐运动动力学

A(=*Vector3r::Zero()*)振幅 [N]

dead(=否)
若为 true，此引擎将完全不运行；可用于暂时停用引擎，并在之后某个时间点重新启用它。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所用的累计时间（以纳秒为单位；仅在*O.timingEnabled*==True时使用）。

f(=*Vector3r::Zero()*)频率 [赫兹]

fi(=*Vector3r::Zero()*)
初始相位 [弧度]。默认情况下，相位为零，因此力从零开始。

ids(=未初始化)标识符受此部分引擎影响的刚体列表。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由'yade -jN'定义的数量 N（不过此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时，以致产生每层线程数不同的嵌套 OMP 循环。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用且 *O.timingEnabled*==True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) →
None：根据给定字典更新对象属性

class yade.wrapper.HarmonicMotionEngine(继承 运动学引擎 → 部分引擎 → *E n-gine* → 可序列化对象)

此引擎实现刚体的简谐振动。另请参见 简谐力引擎，它施加简谐力；另请参见 简谐运动动力学

A(=*Vector3r::Zero()*)振幅 [米]

dead(=否)如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并在之后的某个时间点重新启用它。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所花费的累计时间（以纳秒计，仅在*O.timingEnabled*==True时使用）。

f(=*Vector3r::Zero()*)频率 [赫兹]

fi(=*Vector3r(Mathr::PI/2.0,Mathr::PI/2.0, Mathr::PI/2.0)*)初始相位 [弧度]。默认情况下，刚体围绕初始位置振荡。

ids(=未初始化)标识符 受此部分引擎影响的刚体列表。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_数量_线程，或由“yade -jN”定义的 N（不过具体行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时；这会导致嵌套 OMP 循环，并且每个层级的线程数不同。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用并设置 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)arg1*，*(字典)arg2*) →
None：根据给定字典更新对象属性

class yade.wrapper.HarmonicRotationEngine(继承自 旋转引擎 → 运动学引擎 → 部分引擎 →引擎 →可序列化对象)
此引擎实现刚体的简谐旋转振荡，另请参阅简谐运动的动力学；请将由此引擎驱动的刚体的动态=设为 False，否则振幅将是预期值的 2 倍。

A(=*0*)
振幅 [rad]

angularVelocity(=*0*)
角速度。 [rad/s]

dead(=*false*)若为 true，此引擎将完全不会运行；可用于暂时停用引擎，并在之后的某个时间点重新启用。

dict(*(可序列化对象)辐角1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
该引擎运行所用的累计时间（纳秒）（仅在 if *O.timingEnabled*==True）。

f(=*0*)频率 [赫兹]

fi(=*Mathr::PI/2.0*)初始相位 [弧度]。默认情况下，刚体会围绕初始位置振荡。

ids(=已取消初始化)标识符 受此部分引擎影响的刚体列表。

label(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

ompThreads(=-1)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响其代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与 *ParallelEngine* 结合使用包含并行区域的引擎，从而形成嵌套 OMP 循环，且各层的线程数不同。

rotateAroundZero(=否)
如果为 True，刚体将不会绕其质心旋转，而是绕 zeroPoint 旋转。

rotationAxis(=*Vector3r::UnitX()*)旋转轴（方向）；将自动归一化。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 ce
代码和 *O.timingEnabled*==True。

updateAttrs(*(可序列化对象)arg1*，*(字典)arg2*) →
None：根据给定字典更新对象属性

zeroPoint(=*Vector3r::Zero()*)如果 rotateAroundZero 为 True，刚体将围绕此点旋转

class yade.wrapper.HelixEngine(继承 旋转引擎 → 运动学引擎 → 部分引擎 →引擎 →可序列化对象)同时沿同一轴执行旋转和平移的引擎，因此得名螺旋引擎
e

angleTurned(=*0*)目前已经转过了多少。（自动更新） [弧度]

angularVelocity(=*0*)角速度。 [
弧度/秒]

dead(=否)如果为 true，此引擎将完全不会运行；可用于暂时停用引擎，并仅在稍后某个时刻重新启用它。

dict(（可序列化对象）辐角*1*) →字
典：返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所用的累计时间（以纳秒计；仅在*O.timingEnabled*==True 时使用）。ids(=未初始化)标识符 受此部分引擎影响的刚体列表。label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

linearVelocity(=*0*)
线速度 [米/秒]

ompThreads(=-1)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由‘yade -jN’定义的数量 N（此行为可能取决于
p引擎 g数量 gh)。此属性仅 y 影响 en g其代码包含

OpenMP 并行区域（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会产生嵌套 OMP 循环，且每一层的线程数不同。

`rotateAroundZero`(=否)
如果为 True，刚体将不会绕其质心旋转，而是绕 `zeroPoint` 旋转。

`rotationAxis`(=*Vector3r::UnitX()*)旋转轴（方向）；将自动归一化。

`timingDeltas`
引擎本身的详细计时信息。除非在源代码中启用并设置 *O.timingEnabled*==`True`。

`updateAttrs`(*(*可序列化对象*)arg1*，*(*字典*)*辐角*2*) → None：根据给定字典更新对象属性

`zeroPoint`(=*Vector3r::Zero()*)如果 `rotateAroundZero` 为 True，刚体将绕此点旋转

`class yade.wrapper.HydroForceEngine`(继承 部分引擎 → 引擎 → 可序列化对象)

通过体积平均一维流体求解实现离散元法耦合的引擎，用于模拟稳态

该引擎可分为三个部分：（i）根据流体速度剖面和流体性质，对颗粒施加流体力；（ii）计算流体力施加和流体求解所需的平均固相深度剖面；（iii）求解体积平均一维流体动量平衡。

下面详细介绍这三个不同的函数：

(i) 作用于颗粒的流体力：对每个颗粒施加由一维流体流动产生的浮力、阻力和升力，并可在两个颗粒之间施加润滑力。施加的阻力表达式为

$$F_d = \frac{1}{2} C_d A \rho^f |\boldsymbol{v}_f - \boldsymbol{v}| \boldsymbol{v}_f - \boldsymbol{v}$$

其中，ρ 为流体密度（densFluid），v 为颗粒速度，v_f 为颗粒中心处的流体速度（取自流体速度剖面 vxFluid），$A = \pi d^2/4$为颗粒的投影面积（圆盘），C_d 为阻力系数。阻力系数的计算公式取决于局部颗粒雷诺数和固相体积分数。阻力的计算公式为 [Dallavalle1948] [RevilBaudard2013]，并通过 Richardson-Zaki [Richardson 1954] 进行修正，以考虑受阻效应。该定律在泥沙输运中十分常见。可能启用的润滑力（将参数 yref:lubrication<HydroForceEngine.lubrication> 设为 True）表达式为：$F_{lubrication} = \frac{6\pi\eta^f v_r el^n}{\delta^n + \varepsilon_r}$，其中 η^f 为流体动力黏度 viscoDyn，$v_r el^n$ 为两个颗粒的法向相对速度，δ^n 为两个颗粒表面之间的距离，ε_r 为颗粒的粗糙度尺度（roughnessPartScale）。

可以为每个颗粒启用阻力的脉动，以考虑湍

升力的计算公式取自 [Wiberg1985]，且形式如下：

$$F_L = \frac{1}{2} C_L A \rho^f ((v_f - v)^2_{top} - (v_f - v)^2_{bottom})$$

下标 top 和 bottom 分别表示在所考虑球体的顶部和底部处求得的值。这种升力的表述考虑了湍流剪切流中颗粒顶部与底部的压力差。由于这种表述在接近起动阈值时存在争议 [Schmeeckle2007] 因此可以通过变量 升力停用该项。浮力通过浮重计入：

$$F_{buoyancy} = -\rho^f V^p g$$

，其中 g 是沿垂直方向的重力向量，V^p 是颗粒的体积。在流体流动稳态且均匀的情况下，浮力会简化为其壁法向分量

（参见 [Maurin2018] 以了解完整说明），并应将*steadyFlow* 设为 true，以消除流向分量。

(ii) 平均固相深度剖面 averageProfile 函数会计算颗粒速度、颗粒固相体积分数和颗粒阻力的体积加权平均深度剖面（一维）。它采用体积加权平均方法，参见 [Maurin2015PhD]_[Maurin2015b]_；也就是说，给定离散化壁法向位置 z 处与颗粒相关联的变量 A^p 的平均值为：

$$\langle A\rangle^s(z) = \frac{\sum_{p|z^p\in[z-dz/2,z+dz/2]} A^p(t)V_z^p}{\sum_{p|z^p\in[z-dz/2,z+dz/2]} V_z^p}$$

求和范围为位于壁法向位置 $z-dz/2$ 和 $z+dz/2$之间切片内的颗粒，V^p 表示给定颗粒的体积中实际包含在该切片内的部分。更多详情请参见 [Maurin2015PhD]。

(iii) 一维体积平均流体分辨率 流体求解基于一维体积平均流体动量平衡的求解。根据定义，它假设流体流动是稳态且均匀的（单向）。这与 [RevilBaudard2013]所采用的流体求解相同。详情请参见该论文以及 [Maurin2015PhD][Maurin2015b]。

这三个不同的组件可以独立使用，例如，根据给定的流体剖面施加流体力，或针对给定颗粒浓度求解流体动量平衡。

`Cl`(=*0.2*)
升力系数的取值取自 [Wiberg 1985]

`ReynoldStresses`(=未初始化)
尺寸等于 *nCell* 的向量，包含雷诺应力随深度变化的函数。ReynoldStresses(z) $=\rho^f < u_x' u_z' > (z)^2$

`averageDrag`(=未初始化)
离散化的平均阻力深度剖面。在引擎中不起作用，是输出参数。出于实用考虑，可以直接在引擎内部计算：从 Python 调用引擎的 averageProfile() 方法。

`averageDrag1`(=取消初始化)
1 型颗粒的离散化平均阻力深度剖面。在双尺寸设为 o
真。

`averageDrag2`(=取消初始化)
类型 2 颗粒的离散化平均阻力深度剖面。在双尺寸设为真时计算。

`averageProfile`(*(HydroForceEngine)*辐角 *1*) → 无：
计算并存储颗粒速度（*vxPart*、*vyPart*、*vzPart*）以及固相体积分数（*phiPart*）的深度分布。对于每个已定义的单元 z，平均颗粒速度的 k 分量为：

$< v_k >^z = \sum_p V^p v_k^p / \sum_p V^p$,

其中，总和是对单元内的颗粒求得的，v_k^p 是与颗粒 p 相关联的速度的 k 分量，V^p 是颗粒 p 位于单元内的那部分体积。此定义可使平均过程更加平滑；当每个单元内的颗粒数量较多时，它等价于仅考虑颗粒中心。至于固相体积分数，也采用相同方式计算：对于每个已定义的单元 z，其表达式为：

$< \varphi >^z = \frac{1}{V_{cell}} \sum_p V^p$，其中 V_{cell} 是所考虑单元的体积，V^p 是单元 z 中所含颗粒 p 的体积。此函数生成平均速度和固相体积分数的深度剖面，返回从高度为 *zRef* 的参考水平面（输入参数）到高度为 *zRef* 加上

nCell 个 *deltaZ* （输入参数）。当选项 *twoSize* 设置为 True 时，还会计算平均阻力（*averageDrag1* 和 *averageDrag2*）和固相体积分数（*phiPart1* 和 *phiPart2*）的深度剖面；计算平均值时分别仅考虑半径为 *radiusPart1*和 *radiusPart2* 的颗粒。

bedElevation(=*0.*)
床面高度，高于该高度时流体流动呈湍流，颗粒会发生湍流速度脉动。

channelWidth(=*1.*)
流体求解：用于在流体求解过程中计算流体壁面摩擦的通道宽度。

compatibilityOldVersion(=*false*)使 HydroForceEngine 兼容旧版脚本的选项。会略微降低计算速度，最终将被移除。

computeRadiusParts(*(HydroForceEngine)*辐角 *1*) →无：计算模拟中存在的不同半径类别。

convAcc(=未初始化)与深度相关的对流加速度

convAccOption(=否)启用对流加速度选项，以便在动量平衡中考虑对流加速度项。

dead(=否)若为真，此引擎将完全不运行；可用于暂时停用引擎，并在之后某个时刻将其重新启用。

deltaZ(=未初始化)离散单元的高度。

densFluid(=*1000*)流体密度，默认值为水的密度

dict(*(*可序列化对象*)*辐角 *1*) →字典：
返回属性字典。

dpdx(=*0*。)沿流向的压力梯度

dtFluct(=未初始化)湍流脉动模型的执行时间步。

enableMultiClassAverage(=否)为所有不同的颗粒尺寸启用特定的平均处理。若使用多种不同的颗粒尺寸，会占用大量内存

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所用的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

^{e}**xpoRZ**(=*3.1*)受阻效应引起的阻力修正所用的 Richardson-Zaki 指数值

fluctTime(=取消初始化)向量，包含与每个颗粒相关的脉动的持续时间。

fluidFrictionCoef(=*1*。)流体求解：用于拟合流体壁面摩擦的系数

`fluidResolution`(*(HydroForceEngine)*参数*1*，*(*浮点数*)*参数*2*，*(*浮点数*)*辐角*3*) → 无:
在已定义的网格（*nCell*、*deltaZ*）上求解一维体积平均流体动量平衡，使用体积平均固体剖面（*phiPart*,:yref:*vxPart*<*HydroForceEngine.vxPart*>,:yref:*averageDrag*<*HydroForceEngine.averageDrag*>），这些剖面可通过 averageProfile 函数求得。

`fluidWallFriction`(=*false*)
流体求解：若设为 true，则引入一个汇项以计入墙面处的流体摩擦，详情请参见 [Maurin 2015] 。需要设置通道宽度。这可能会显著减慢计算速度。

`gravity`(=*Vector3r(0, 0, -9.81)*)重力向量

`ids`(=未初始化)
受此部分引擎影响的刚体 ID 列表。`ilm`(=*2*)流体求解：混合长度求解类型：0：经典普朗特混合长度，1：考虑自由表面效应的普朗特混合长度，2：考虑颗粒存在影响的阻尼湍流 [Li1995], 详见 [RevilBaudard2013]。`initialization`((HydroForceEngine)辐角*1*) → None：初始化使 HydroForceEngine 运行所需的参数。必须在运行任何模拟之前执行，否则会崩溃`irheolf`(=*0*)流体求解：有效流体黏度选项：0：纯流体黏度，1：爱因斯坦黏度。`iturbu`(=*1*)流体求解：启用湍流求解（1），或不启用（0）`iusl`(=*1*)流体求解：设置流体顶部边界条件的选项，0：狄利克雷，固定（u = uTop en z = h），1：诺伊曼，自由表面（du/dz = 0 en z = h）。`kappa`(=*0.41*)流体求解：冯·卡门常数。可调整此值以考虑颗粒对流体湍流的影响，参见例如 [RevilBaudard2015]`label`(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。`lift`(=false)是否启用升力计算`lubrication`(=false)启用润滑力计算的条件。`multiDragPart`(=未初始化)每类颗粒的空间平均曳力。未使用？还是仅用于调试。`multiPhiPart`(=未初始化)每类颗粒的空间平均固体体积分数。`multiVxPart`(=未初始化)每类颗粒的 x 方向空间平均速度。`multiVyPart`(=未初始化)每类颗粒的 y 方向空间平均速度。`multiVzPart`(=未初始化)每类颗粒的 z 方向空间平均速度。`nCell`(=*1*)深度方向上的单元数量

`nbAverageT`(=*0*)
如果为 >0，则在空间平均之外，再对 nbAverage 步进行时间平均。

`ompThreads`(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与 *ParallelEngine* 结合使用包含并行区域的引擎，从而产生嵌套 OMP 循环，且每层的线程数不同。

`phiBed`(=*0.08*)
湍流建模参数。与混合长度建模 ilm $=5$ 相关。

`phiMax`(=*0.64*)
流体求解：最大固相体积分数。

`phiPart`(=未初始化)
离散化的固相体积分数深度分布。可作为输入参数，也可在引擎内部直接计算，方法是在 Python 中调用 averageProfile() 函数

`phiPart1`(=未初始化)
1 型颗粒的离散化固相体积分数深度分布。在 双尺寸设为 True 时计算。

`phiPart2`(=未初始化)
类型 2 颗粒的离散固相体积分数深度分布。在 双尺寸设为 True 时进行评估。

`pointParticleAverage`(=*false*)
使用点颗粒法计算平均值。如果为 False，则考虑颗粒范围，并根据每个平均单元中包含的体积对平均值进行加权。

`radiusPart`(=*0.*)
参考颗粒半径

`radiusPart1`(=*0.*)
1 型颗粒的半径。仅当 双尺寸 设置为 True 时才有用。

`radiusPart2`(=*0.*)类型 2 颗粒的半径。仅当 双尺寸 设置为 True 时才有用。

`radiusParts`(=未初始化)用于记录模拟中不同颗粒半径数量的变量。允许按颗粒尺寸进行分组平均。

`roughnessPartScale`(=*1e-3*)
颗粒的粗糙度长度尺度。实际上，当两个颗粒之间的距离达到 roughnessPartScale 时，润滑力会被截断。

`steadyFlow`(=*true*)
根据静止流体与稳态流动中的流体之间的物理差异，修改浮力的条件。更多详情请参见 [Maurin2018]

`taufsi`(=未初始化)
流体求解：创建 Taufsi/rhof $=$ 阻力项/(rhof(vf-vxp))，以传递给流体 co de

`timingDeltas`
引擎本身内部的详细计时信息。除非在源代码中启用并设置 *O.timingEnabled*==`True`。

`turbulentFluctuation`(*(HydroForceEngine)*辐角 *1*) →无：
在计算阻力时应用离散随机游走模型，以考虑流体速度的湍流脉动。该模型非常简单，通过雷诺应力的值施加脉动，以恢复属性 $<u'_x u'_z>(z) =< R^f_{xz} > (z)/\rho^f$。随机脉动会在由涡周转时间确定的时间尺度内发生变化。

turbulentFluctuationZDep(*(HydroForceEngine)*辐角 *1*) → 无：
以类似 turbulentFluctuation 的方式将湍流脉动应用于问题，但根据颗粒位置更新脉动。

turbulentViscosity(=未初始化)
流体求解：湍流黏度是深度的函数

twoSize(=否)
不再维护。用于在模拟中考虑两种颗粒尺寸时启用的选项。启用后，将分别计算直径为 diameterPart1 和 diameterPart2 的两种颗粒的平均固相体积分数和阻力。

uTop(=*1*。)流体求解：当 iusl $= 0$ 时顶部边界处的流体速度

unCorrelatedFluctuations(=否)
生成不相关流体脉动的条件。默认算例表示自由表面流，其中垂向与流向流体速度脉动相关（参见 Nezu 与 Nagakawa，1992 年的参考著作《明渠流中的湍流》）。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角 *2*) → None：根据给定字典更新对象属性

vCell(=未初始化)平均单元体积

vFluctX(=未初始化)将流向流体速度脉动与各颗粒关联的向量。脉动在 C++ 代码中根据离散随机游走模型计算

vFluctY(=未初始化)将展向流体速度脉动与各颗粒关联的向量。该脉动由 C++ 代码根据离散随机游走模型计算

vFluctZ(=未初始化)
将法向流体速度脉动与各颗粒关联的向量。该脉动由 C++ 代码根据离散随机游走模型计算

vPart(=未初始化)
离散化的流向固体速度深度剖面，分别沿 x、y 和 z 方向。在一维流体耦合求解中，仅考虑 x 方向的测量值。另两个方向可用作输出参数。x 分量可作为输入参数，也可在引擎内部直接求值，方法是从 Python 调用 averageProfile() 函数

velFluct(=*false*)
若为 true，则仅在每个颗粒所在位置，使用基于雷诺应力剖面的简单离散随机游走（DRW）模型，激活下一时间步的湍流流体速度脉动计算（*ReynoldStresses*）

viscoDyn(=*1e-3*)
流体的动力黏度，默认为水的黏度

viscousSubLayer(=*0*)
流体求解：若设为 1，则求解靠近底部边界的黏性底层

vxFluid(=未初始化)
t 时刻离散化的流向流体速度深度剖面

vxPart(=未初始化)
离散化固体流向速度深度剖面。可作为输入参数提供，或在引擎内部直接调用 Python 的 averageProfile() 函数进行计算。

vxPart1(=未初始化)
1 型颗粒的离散固体流向速度深度剖面。在双尺寸 设置为 True 时进行评估。

vxPart2(=取消初始化)
类型 2 颗粒的离散固体流向速度深度剖面。仅当双尺寸设为 True 时计算。

vyPart(=未初始化)
离散固体展向速度深度剖面。可作为输入参数，也可在引擎内部直接调用 averageProfile() 函数计算。

vyPart1(=未初始化)
1 型颗粒的离散固体展向速度深度剖面。仅当双尺寸设为 True 时计算。

vyPart2(=未初始化)
类型 2 颗粒的离散化固体展向速度深度剖面。在双尺寸设为 True 时进行计算。

vzPart(=未初始化)
离散化的壁法向固体速度深度分布。可作为输入参数传入，或 r
在引擎内部直接计算，也可从 Python 调用 averageProfile() 函数

vzPart1(未初始化)1 型颗粒的离散化固体壁面法向速度深度分布。在双尺寸 设为 True 时进行计算。

vzPart2(=未初始化)
类型 2 颗粒的离散化的固体壁面法向速度深度分布。在双尺寸 设为 True 时进行评估。

zRef(=*0*。)参考点的位置，该点对应于流体速度的第一个值，即地面。

class yade.wrapper.InterpolatingDirectedForceEngine(继承 力引擎 → 部分引擎 →引擎 → 可序列化对象)

用于对已订阅刚体施加大小可变但方向恒定之力的引擎。时间与大小必须具有相同的长度；方向（会自动归一化）决定力的朝向。

与插值引擎一样：在第一个时间点之前使用第一个力的大小，在最后一个时间点之后使用最后一个力的大小。wrap 指定时间是否会在最后一个时间点之后回绕到第一个时间点。

dead(=*false*)
如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并仅在之后的某个时间点重新启用它。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

direction(=*Vector3r::UnitX()*)接触力方向（自动归一化）

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所耗费的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

force(=*Vector3r::Zero()*)要
施加的力。

ids(=已取消初始化)标识符 此部分引擎所影响的刚体列表。

label(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，你可以直接在 Python 中引用它。

magnitudes(=取消初始化)
力的大小读数 [N]

ompThreads(=-1)
引擎中使用的线程数。如果 ompThreads<0 （默认值），该数量通常为 OMP_数量_线程数，或由“yade -jN”定义的数量 N（不过此行为可能取决于具体引擎）。此属性只会影响其代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或用于组合 *ParallelEngine* 与运行并行区域的引擎，从而形成嵌套 OMP 循环，并使每一层使用不同的线程数。

times(=未初始化)时间
读数 [s]

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用，否则为空 e
代码和 *O.timingEnabled*==True。

updateAttrs(*(可序列化对象)*参数*1*，*(字典)*辐角*2*) → 无:
根据给定字典更新对象属性

wrap(=否)若超出最后一个时间点，则 wrap 回序列开头

class yade.wrapper.InterpolatingHelixEngine(继承 螺旋引擎 → 旋转引擎 →运动学引擎 → 部分引擎 →引擎→ 可序列化对象)

应用螺旋运动的引擎，通过对时间和值进行线性插值来确定当前角速度，并使用斜率参数计算平移。

插值会在第一个时间点之前采用边界值，在最后一个时间点之后采用最后一个值。如果指定了 wrap，时间会从最后一个时间值绕回第一个时间值（请注意，不会对最后一个值和第一个值之间进行插值）。

angleTurned(=*0*)
到目前为止我们转过了多少。 （自动更新） [弧度]

angularVelocities(=未初始化)
角速度列表；长度必须与时间相同。 [弧度/秒]

angularVelocity(=*0*)角速度。 [
弧度/秒]

dead(=*false*)若为真，此引擎将完全不运行；可用于暂时停用引擎，并仅在之后某个时间点将其重新启用。

dict(*(可序列化对象)*辐角*1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所耗费的累计时间（以纳秒计，仅在*O.timingEnabled*==True时使用）。

ids(=未初始化)
标识符 受此部分引擎影响的刚体列表。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

`linearVelocity`(=*0*)
线速度 [米/秒]

`ompThreads`(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_数量_线程数，或由 'yade -jN' 定义的数量 N（不过，此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在组合 *ParallelEngine* 与运行并行区域的引擎时使用，这会导致嵌套 OMP 循环在各层级使用不同的线程数。

`rotateAroundZero`(=否)
如果为 True，刚体不会绕其质心旋转，而是围绕 `zeroPoint`。

`rotationAxis`(=*Vector3r::UnitX()*)旋转轴（方向）；将自动归一化。

`slope`(=*0*)
每弧度转角的轴向平移（可以为负） [m/rad]

`times`(=未初始化)给出速度的时间点列表；必须按递增顺序排列 [s]

`timingDeltas`
引擎本身内部的详细计时信息。除非在源代码中启用，否则为空；且 *O.timingEnabled*==`True`。

`updateAttrs`(*(可序列化对象)arg1*, *(字典)arg2*) → None：根据给定字典更新对象属性

`wrap`(=否)若 t>时间_n，则对 t 进行 wrap，即 t_wrap=t-N*(时间_n-时间_0)

`zeroPoint`(=*Vector3r::Zero()*)若 `rotateAroundZero` 为真，刚体将围绕此点旋转

`class yade.wrapper.KinematicEngine`(继承部分引擎 → 引擎 → 可序列化对象)用于施加预设位移的抽象引擎。

注意： 派生类应重写 `apply` 并使用给定的 `ids` （而非 `action` 使用部分引擎.*ids*），以便它们组合使用时也能正常工作；所有已订阅刚体的速度和角速度会在调用 `apply` 方法前重置，因此该方法应仅对这些物理量进行增量更新。

`dead`(=否)
如果为真，此引擎将完全不运行；可用于暂时停用引擎，并在之后某个时间点重新启用。

`dict`(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

`execCount`
此引擎的累计运行次数（仅在*O.timingEnabled*==`True`时使用）。

`execTime`
此引擎运行所用的累计时间（纳秒），仅在*O.timingEnabled*==`True`时使用）。

`ids`(=未初始化)标识符受此部分引擎影响的刚体列表。

`label`(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，您可以直接在 Python 中引用它。

`ompThreads`(=*-1*)
: 引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，从而在不同层级形成线程数不同的嵌套 OMP 循环。

`timingDeltas`
: 引擎本身内部的详细计时信息。除非在源代码中启用 *O.timingEnabled*==`True`，否则为空。

`updateAttrs`(*(可序列化对象)参数1*, *(字典)辐角2*) → None：根据给定字典更新对象属性

`class yade.wrapper.LawTester`(继承自部分引擎 →引擎 →可序列化对象)
: 指定并应用相互作用的变形，即局部相对位移和旋转。加载路径既可以使用 path（由包含广义位移的 6 维向量组成的序列 u_x、 u_y、 u_z、 φ_x、 φ_y、 φ_z）指定，也可以使用 disPath（u_x、 u_y、 u_z）和 rotPath（φ_x、 φ_y、φ_z）指定。pathSteps 用于给出时间函数及其时间值（步数），这些值对应加载路径上的各点。加载值在给定加载路径点之间线性插值，并假定 path 和 pathSteps 均从零值（初始构型）开始。hooks 可以指定在到达路径上相应点时运行的 Python 代码；路径结束后，将运行 doneHook。

  LawTester 应放置在 *InteractionLoop* 和 *NewtonIntegrator* 之间的模拟循环中，因为它通过设置颗粒的线速度/角速度来控制运动；这些速度由 *NewtonIntegrator* 积分，从而产生实际的位置变化，进而使 *IGeom* 得到更新（并在执行 接触定律 时应用），而这发生在执行 *InteractionLoop* 时。本构定律对颗粒产生的力不会影响预设的颗粒运动，因为使用 LawTester 时，两个颗粒的所有 自由度都会被锁定 。

  LawTester 尽可能利用 *IGeom* 提供有用的数据（例如局部坐标系），但如果相应的 *IGeom* 中不存在这些数据，它也能够独立计算：

IGeom	#自由度	LawTester 支持级别
L3Geom	3	full
L6Geom	6	full
ScGeom	3	模拟局部坐标系
ScGeom6D	6	模拟局部坐标系

  根据 *IGeom*，3 个（u_x， u_y， u_z）或 6 个（u_x， u_y， u_z， φ_x， φ_y， φ_z）自由度由 LawTester 控制，方法是指定接触的两个颗粒的线速度和角速度。LawTester 控制的所有自由度彼此正交（完全解耦），并且独立控制。

  控制 3 个自由度时，*rotWeight* 用于控制局部剪切是通过使颗粒沿另一颗粒周围的圆弧移动来施加，还是通过旋转而不改变位置来施加；尽管这种旋转会使相互作用发生相互转动，但对于只有 3 个自由度的 *IGeom* ，该转动会被忽略。控制 6 个自由度时，剪切仅通过圆弧位移施加，否则会发生相互转动。

  idWeight 在两个颗粒之间分配规定的运动（如果 `id1` 朝 `id2` 或 `id2` 朝 `id1`移动，产生的局部变形相同）。这仅适用于 u_x， u_y， u_z， φ_x；不过，弯曲转动 φ_y， φ_z 无论 `idWeight` 为何值，始终按与半径成反比的比例分配给两个球体，因此不会引起剪切。

  LawTester 可从两个来源获取当前接触变形：一是其自身的内部数据（用于在每一步指定位移），可在 *uTest* 中访问；二是来自*IGeom* 本身（取决于它提供哪些数据），这些数据存储在 *uGeom* 中。这两个值应当相同（忽略数值精度误差），这也是一种测试方式，用于检查 *IGeom* 及相关函子是否按预期进行计算。

由 LawTester 操作的相互作用可通过 *GlExtra_LawTester* 渲染器进行渲染。

参见 scripts/test/law-test.py示例。

dead(=*false*)
如果为 true，此引擎将完全不运行；可用于暂时停用某个引擎，并仅在稍后的某个时间点重新启用它。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

disPath(=未初始化)
加载路径，其中每个三维向量包含所需的法向位移和两个剪切位移分量（位于自动跟踪的局部坐标系中）。如果其长度短于 *rotPath*，则重复使用最后一个值。

displIsRel(=是)
是否按参考接触长度归一化*disPath* 中的位移值（对于两个球体为 r1+r2）。

doneHook(=未初始化)
当达到路径末端时要运行的 Python 命令（字符串形式）。如果为空，则将引擎设为 停用。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所花费的纳秒数累计时间（仅在*O.timingEnabled*==True时使用）。

hooks(=未初始化)
到达路径中的相应点时、执行该步中的其他操作之前要运行的 Python 命令。另请参见完成钩子。

idWeight(=*1*)
浮点数，通常为 〈0,1〉，用于确定位移如何在颗粒之间分配（id1 对应 0，id2 对应 1）；中间值会将相应部分分别应用于每个颗粒。对于具有 6 个自由度的*IGeom*，此参数将被忽略。

ids(=未初始化)受此部分引擎影响的刚体标识符列表。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，你可以直接从 Python 中引用它。

ompThreads(=*-1*)
引擎中要使用的线程数。如果为 ompThreads<0 （默认值），则该数量通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过此行为可能取决于引擎）。此属性只会影响其代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或用于将 *ParallelEngine* 与运行并行区域的引擎组合使用，从而形成嵌套 OMP 循环，并在每个层级使用不同数量的线程。

pathSteps(=向量<整数>（*1*，*1*），（固定步长）)对应值在路径中的步数；如果比路径短，则其余部分使用最后两个值之间的距离。

refLength(=*0*)
参考接触长度，仅用于渲染。

renderLength(=*0*)
用于渲染的特征长度，设为较小的半径。

rotPath(=已取消初始化)
加载路径的旋转分量，其中每个条目都包含扭转和两个局部坐标系中的弯曲转动。如果长度短于路径，则重复最后一个值。

rotWeight(=*1*)
浮点数〈0,1〉，用于确定剪切位移是以圆弧上的旋转还是位移形式施加（0 表示仅位移，1 表示仅旋转）。指定相互转动时，此设置无效。

step(=*1*)
此引擎处于活动状态的步数；使用 pathSteps 确定其在路径中的位置。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用，否则为空 e
代码和 *O.timingEnabled*==True。

trsf(=未初始化)局部坐标系的变换矩阵。（自动更新）

uGeom(=*Vector6r::Zero()*)当前广义位移（3 个位移、3 个旋转），存储在交互本身中。它们应与 *uTest* 相对应，否则表示存在缺陷。

uTest(=*Vector6r::Zero()*)根据此 *LawTester*，当前广义位移（3 个位移、3 个旋转）应为这些值。应与 *uGeom* 相对应。

updateAttrs(*(可序列化对象)arg1*，*(字典)arg2*) → 无:
从给定字典更新对象属性

uuPrev(=*Vector6r::Zero()*)前一步达到的广义位移值，用于确定当前步应应用哪个增量。

class yade.wrapper.LinearDragEngine(继承 部分引擎 → 引擎 → 可序列化对象)
对 某些颗粒施加黏性阻力或线性阻力，每步都会按其线速度的比例使其减速。施加力的计算式为

$$F_d = -bv$$

其中， b 为线性阻力， v 为颗粒速度。

$$b = 6\pi\nu r$$

其中 ν 为介质黏度， r 为斯托克斯半径，即颗粒的斯托克斯半径（但为简化起见，此处将其视为等于球体半径），

注意： 线性阻力仅作用于 ids 中列出的球形颗粒。

dead(=否)若为 true，此引擎将完全不运行；可用于暂时停用引擎，并在之后某个时刻重新启用。

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所用的累计时间（纳秒，仅在*O.timingEnabled*==True时使用）。

ids(=未初始化)
标识符 受此部分引擎影响的刚体列表。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，才能直接在 Python 中引用。

nu(=*0。001*)介质的黏度。

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响源代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在组合 *ParallelEngine* 与运行并行区域的引擎时使用；这样会产生每一层线程数不同的嵌套 OMP 循环。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无:
从给定字典更新对象属性

class yade.wrapper.RadialForceEngine(继承 部分引擎 →引擎 →可序列化对象)施加大小为给定值、方向背离空间轴的力。

axisDir(=*Vector3r::UnitX()*)轴方向（自动归一化）

axisPt(=*Vector3r::Zero()*)轴上的点

dead(=否)若为 true，该引擎将完全不运行；可用于暂时停用引擎，并在之后的某个时刻重新启用。

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

execCount
此引擎运行的累计次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所用的累计时间（纳秒，仅在*O.timingEnabled*==True时使用）。

fNorm(=*0*)
施加力的大小

ids(=已取消初始化)标识符受此部分引擎影响的刚体列表。

label(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由‘yade -jN’定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在组合 *ParallelEngine* 与运行并行区域的引擎时使用，这会导致嵌套 OMP 循环的各层线程数不同。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用*O.timingEnabled*==True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → None：从给定字典更新对象属性

class yade.wrapper.RotationEngine(继承 运动学引擎 → 部分引擎 → 引擎 →可序列化对象)

通过设置角速度对已订阅刚体施加旋转的引擎。如果 绕零点旋转已设置，则每个刚体还会绕 零点移动。

angularVelocity(=0)角速度。 [弧度/秒]

dead(=否)如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并在稍后的某个时刻重新启用。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

execCount
该引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
该引擎运行所花费的累计时间（纳秒）（仅在 *O.timingEnabled*==True 时使用）。

ids(=已取消初始化)标识符 受此部分引擎影响的刚体列表。

label(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

ompThreads(=-1)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由 'yade -jN' 定义的 N（不过，此行为可能取决于引擎）。此属性只会影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在与 *ParallelEngine* 结合使用包含并行区域的引擎时；这会导致嵌套 OMP 循环的各层使用不同数量的线程。

rotateAroundZero(=否)若为 True，刚体将不会绕其质心旋转，而是绕 zeroPoint旋转。

rotationAxis(=*Vector3r::UnitX()*)旋转轴（方向）；将自动归一化。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用并设置 *O.timingEnabled*==True，否则为空。

updateAttrs((可序列化对象)*arg1*，(字典)*arg2*) → None：从给定字典更新对象属性

zeroPoint(=*Vector3r::Zero()*)若 rotateAroundZero 为 True，刚体将绕此点旋转

class yade.wrapper.ServoPIDController(继承 平移引擎 → 运动学引擎 →部分引擎 → 引擎 → 可序列化对象)PID 控制器伺服引擎，用于对刚体施加预设力。 http://en.wikipedia.org/wiki/PID 控制器_axis(=*Vector3r::Zero()*)用于施加速度的单位向量 [-]curVel(=*0*。*0*)当前施加的速度 [米/秒]

current(=*Vector3r::Zero()*)控制器的当前值 [N]

dead(=否)若为真，该引擎将完全不运行；可用于暂时停用引擎，并在之后的某个时间点重新启用。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

errorCur(=*0*。*0*)当前错误 [N]

errorPrev(=*0*。*0*)先前错误 [N]

execCount
该引擎运行的累计次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所花费的累计时间（纳秒）（仅用于 if *O.timingEnabled*==True）。

iTerm(=*0*。*0*)积分项 [N]

ids(=未初始化)标识符 受此部分引擎影响的刚体列表。

iterPeriod(=*100*。*0*)速度相关性的周期性判据 [-]

iterPrevStart(=*-1*。*0*)速度相关性的前一次迭代 [-]

kD(=*0.0*)PID 控制器的微分增益/系数 [-]

kI(=*0.0*)PID 控制器的积分增益/系数 [-]

kP(=*0.0*)PID 控制器的比例增益/系数 [-]

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

maxVelocity(=*0.0*)速度 [米/秒]

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），该数量通常为 OMP_数量_线程数，或由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合时使用，从而在不同层级产生线程数不同的嵌套 OMP 循环。

target(=*0*。*0*)
控制器 [N] 的目标值

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

translationAxis(=未初始化)
施加平移的方向 [向量3]

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
从给定字典更新对象属性

velocity(=未初始化)施加速度的标量值 [米/秒]。施加的矢量速度为 速度 * 轴

class yade.wrapper.StepDisplacer(继承 部分引擎 → 引擎 → 可序列化对象)逐步对已订阅刚体施加广义位移（位移或旋转）。可用于接触定律测试（通过移动一个球体，使其相对于另一个球体发生位移）；但在这种情况下，请参阅 *LawTester*

dead(=否)
如果为 true，此引擎将完全不运行；可用于暂时停用引擎 d
并仅在稍后的某个时间点重新启用它。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

execCount
该引擎运行的累计次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
该引擎运行所耗费的累计时间（以纳秒为单位；仅在*O.timingEnabled*==True 时使用）。

ids(=未初始化)标识符 受此部分引擎影响的刚体列表。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

mov(=*Vector3r::Zero()*)每次迭代要应用的线性位移步长，通过将其加到 *State.pos* 上实现。

ompThreads(=-1)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或者由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与 *ParallelEngine* 组合使用会运行并行区域的引擎，从而形成嵌套 OMP 循环，且各层的线程数不同。

rot(=*Quaternionr::Identity()*)每次迭代要应用的旋转步长（通过与 *State.ori*进行旋转复合）。

setVelocities(=*false*)
如果为 false，则直接更新位置和取向，而不改变相关刚体的速度。如果为 true，则仅修改速度和角速度。在后一种情况下，应使用 积分器 ，这样借助此引擎，刚体便会在一次迭代（dt）内完成指定的跳跃。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
根据给定字典更新对象属性

class yade.wrapper.TorqueEngine(继承 部分引擎 →引擎 →可序列化对象)在每一步，
将给定的力矩（动量）值施加到每个已订阅的颗粒上。

dead(=否)如果为真，此引擎将完全停用；可用于暂时停用引擎，并在之后的某个时间点重新启用。

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

execCount
: 此引擎运行的累计次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
: 此引擎运行所用的累计时间（以纳秒为单位）（仅在 if *O.timingEnabled*==True 时使用）。

ids(=未初始化)标识符 受此部分引擎影响的刚体列表。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

moment(=*Vector3r::Zero()*)要施加的
力矩值。

ompThreads(=*-1*)
: 引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性只会影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在组合 *ParallelEngine* 与运行并行区域的引擎时使用，从而形成嵌套 OMP 循环，并且每一层的线程数各不相同。

timingDeltas
: 引擎本身内部的详细计时信息。除非在源代码中启用，否则为空；且需设置 *O.timingEnabled*==True。

updateAttrs(*(可序列化对象)参数1*, *(字典)辐角 2*) →
None：根据给定字典更新对象属性

class yade.wrapper.TranslationEngine(继承自 运动学引擎 → 部分引擎 → 引擎→ 可序列化对象)通过设置线速度来对已订阅刚体施加平移运动的引擎。

dead(=*false*)若为真，此引擎将完全不会运行；可用于暂时停用引擎，并仅在之后的某个时刻重新启用它。

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

execCount
: 此引擎运行的累计次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
: 此引擎运行所花费的累计时间（纳秒）（仅在*O.timingEnabled*==True时使用）。

ids(=未初始化)标识符 受此部分引擎影响的刚体列表。

label(=未初始化)
: 此对象的文本标签；必须是有效的 Python 标识符，您可以直接从 Python 中引用它。

ompThreads(=*-1*)
: 引擎中要使用的线程数。如果为 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由'yade -jN'定义的数量 N（不过，此行为可能取决于引擎）。此属性只会影响代码中包含

OpenMP 并行区域（例如 *InteractionLoop*）。此属性主要适用于实验，或在组合 *ParallelEngine* 与运行并行区域的引擎时，这会导致不同层级的嵌套 OMP 循环使用不同的线程数。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

translationAxis(=未初始化)
施加平移的方向 [向量3]

updateAttrs(*(可序列化对象)参数1*, *(字典)辐角2*) → None：根据给定字典更新对象属性

velocity(=未初始化)
施加速度的标量值 [米/秒]。施加的矢量速度为 速度 * 轴

2.3.5 分派器

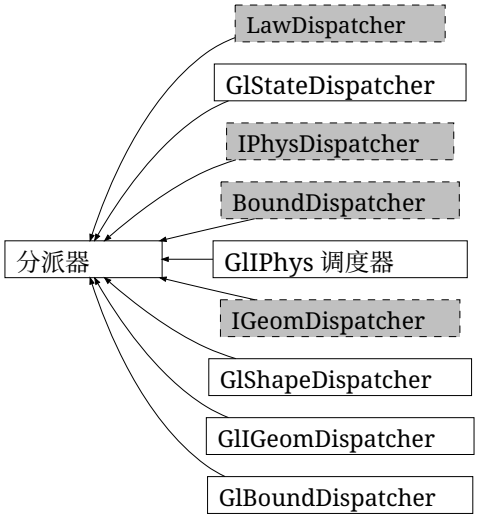


图 31：分派器的继承图，灰色虚线框中的类将在各自的章节中讨论：*LawDis-patcher*、*IPhysDispatcher*、*BoundDispatcher*、*IGeomDispatcher*。另请参阅：*GlBoundDispatcher*、*GlIGe-omDispatcher*、*GlIPhysDispatcher*、*GlShapeDispatcher*、*GlStateDispatcher*。

class yade.wrapper.Dispatcher(继承自 *Engine* → 可序列化对象)
Engine 根据接收到的参数类型，将控制权分派给关联的函子。这个抽象基类本身不提供任何功能。

dead(=否)若为 true，此引擎将完全不运行；可用于暂时停用引擎，并在之后的某个时间点重新启用。**dict**(*(可序列化对象参数)1*) →字典：返回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所耗费的累计时间（以纳秒为单位；仅在*O.timingEnabled*==True时使用）。

label(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，您可以直接引用它 m
Python。

`ompThreads`(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时；这会产生嵌套 OMP 循环，且每一层的线程数不同。

`timingDeltas`
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==`True`。

`updateAttrs`(*(可序列化对象)参数1*，*(字典)辐角2*) → None：根据给定字典更新对象属性

`class yade.wrapper.GlBoundDispatcher`(继承 分派器 → 引擎 → 可序列化对象)根据收到的参数类型分派 函子 。

`dead`(=*false*)
若为真，此引擎将完全停止运行；可用于暂时停用引擎 d
并仅在之后某个时间点将其重新启用。

`dict`(*(可序列化对象)辐角1*) →字典:
返回属性字典。

`dispFunctor`(*(GlBoundDispatcher)参数1*，*(Bound)辐角2*) → GlBoundFunctor：返回针对给定参数将要分派的函子；若不分派则返回 None；若分派存在歧义则抛出异常。

`dispMatrix`(*(GlBoundDispatcher)辐角1*[, *(bool)名称*=真]) → 字典：返回包含分派矩阵内容的字典。

`execCount`
此引擎的累计运行次数（仅在 *O.timingEnabled*==`True` 时使用）。

`execTime`
该引擎运行所用的累计时间（纳秒）;仅在*O.timingEnabled*==`True`时使用。

`functors`
与此分派器关联的函子。

`label`(=取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

`ompThreads`(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，从而产生每一层线程数不同的嵌套 OMP 循环。

`timingDeltas`
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==`True`，否则为空。

`updateAttrs`(*(可序列化对象)arg1*，*(字典)arg2*) → None：根据给定字典更新对象属性

`class yade.wrapper.GlIGeomDispatcher`(继承 分派器 → 引擎 → 可序列化对象)根据收到的 函子 参数类型进行分派。

dead(=*false*)
若为 true，此引擎将完全不运行；可用于暂时停用引擎，并仅在之后的某个时刻将其重新启用。

dict(*(可序列化对象)参数1*) →字典:
返回属性字典。

dispFunctor(*(GlIGeomDispatcher)参数1*，*(IGeom)参数2*) → GlIGeomFunctor：返回针对给定参数分派的函子；若不分派则返回 None；若分派存在歧义则抛出异常。

dispMatrix(*(GlIGeomDispatcher)参数1*[, *(布尔值)names=True*]) → 字典：返回包含分派矩阵内容的字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所用的累计时间（纳秒；仅在*O.timingEnabled*==True时使用）。

functors
与此分派器关联的函子。

label(=取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

ompThreads(=*-1*)
引擎中使用的线程数。如果 ompThreads<0 (默认值)，线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能因引擎而异）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，此时会形成嵌套 OMP 循环，且各层的线程数不同。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.GlIPhysDispatcher(继承 分派器 → 引擎 → 可序列化对象)根据接收到的参数类型调用 函子 的分派器。

dead(=*false*)
如果为 true，则此引擎将完全不运行；可用于将引擎暂时停用，并仅在稍后某个时间点重新启用。

dict(*(可序列化对象)辐角1*) →字典:
返回属性字典。

dispFunctor(*(GlIPhysDispatcher)参数1*，*(IPhys)辐角2*) → GlIPhysFunctor：返回针对给定参数进行分派的函子；如果不分派则返回 None；存在歧义的分派会引发异常。

dispMatrix(*(GlIPhysDispatcher)辐角1*[, *(bool)names=True*]) → 字典:
返回包含分派矩阵内容的字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所用的累计时间（单位为纳秒，仅在*O.timingEnabled*==True时使用）。

`functors`
与此分派器关联的函子。

`label`(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，才能在 Python 中直接引用。

`ompThreads`(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS 或由 'yade -jN' 定义的数量 N（不过此行为可能取决于引擎）。此属性只会影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要用于实验，或在组合 *ParallelEngine* 与运行并行区域的引擎时使用，这会导致不同层级出现线程数不同的嵌套 OMP 循环。

`timingDeltas`
引擎本身内部的详细计时信息。除非在源代码中启用并且设置了 *O.timingEnabled*==`True`，否则为空。

`updateAttrs`(*(可序列化对象)*参数*1*，*(字典)*辐角*2*) → None：根据给定的字典更新对象属性

`class yade.wrapper.GlShapeDispatcher`(继承 分派器 → 引擎 → 可序列化对象)根据收到的参数类型调用 函子 。

`dead`(=*false*)
若为 true，此引擎将完全不运行；可用于暂时停用引擎，并在稍后重新启用。

`dict`(*(可序列化对象)*辐角*1*) →字典：
返回属性字典。

`dispFunctor`(*(GlShapeDispatcher)*参数*1*，*(形状)*辐角*2*) → GlShapeFunctor：返回针对给定参数分派的函子；若不分派则返回 None；若分派存在歧义则抛出异常。

`dispMatrix`(*(GlShapeDispatcher)*辐角*1*[, *(bool)*名称=真]) → 字典：返回包含分派矩阵内容的字典。

`execCount`
此引擎的累计运行次数（仅在*O.timingEnabled*==`True`时使用）。

`execTime`
此引擎运行所用的累计时间（以纳秒为单位，仅在*O.timingEnabled*==`True`时使用）。

`functors`
与此分派器关联的函子。

`label`(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

`ompThreads`(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由 'yade -jN' 定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时；这会导致嵌套 OMP 循环的各层使用不同的线程数。

`timingDeltas`
引擎本身内部的详细计时信息。除非在源代码中启用，否则为空 e
代码和 *O.timingEnabled*==`True`。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.GlStateDispatcher(继承 分派器 → 引擎 → 可序列化对象)分派器调用 函子 根据收到的参数类型进行分派。

dead(=*false*)
如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并仅在之后的某个时间点重新启用它。 d

dict(*(可序列化对象)辐角1*) →字典:
返回属性字典。

dispFunctor(*(GlStateDispatcher)参数1*，*(状态)辐角2*) → GlStateFunctor：返回针对给定参数进行分派的函子；如果没有分派则返回 None；如果分派不明确，则抛出异常。

dispMatrix(*(GlStateDispatcher)辐角1*[, *(布尔值)名称=真*]) → 字典:
返回包含分派矩阵内容的字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所耗费的累计时间（以纳秒计，仅在*O.timingEnabled*==True时使用）。

functors
与此分派器关联的函子。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可以在 Python 中直接引用。

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或者由'yade -jN'定义的数量 N（不过此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，从而产生每层线程数不同的嵌套 OMP 循环。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用并且 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无:
根据给定字典更新对象属性

2.3.6 函子

class yade.wrapper.Functor(继承可序列化对象)
一种类似函数的对象；当参数类型与函子声明接受的类型相匹配时，由分派器调用。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

label(=未初始化*lized*)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

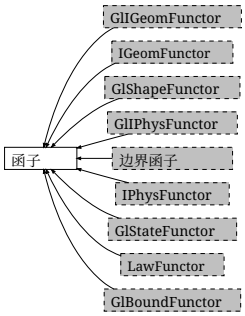


图 32：函子的继承图，灰色虚线类将在各自的章节中讨论：*GlIGeom-函子*、*IGeomFunctor*、*GlShapeFunctor*、*GlIPhysFunctor*、*边界函子*、*IPhysFunctor*、*GlStateFunc-函子*、*LawFunctor*、*GlBoundFunctor*。

timingDeltas
分派器内部的详细计时信息。除非在 t he
源代码和 O.timingEnabled==True。

updateAttrs(*(可序列化对象)arg1*，*(字典)arg2*) → None：从给定字典更新对象属性

2.3.7 包围体创建

边界函子

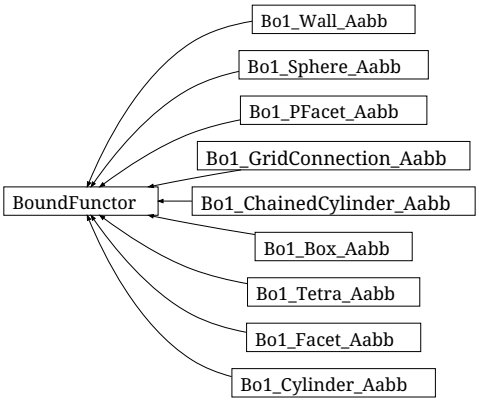


图 *33*：边界函子的继承图。另请参见：*Bo1_盒子_Aabb*、*Bo1_链式圆柱体_-_Aabb*、*Bo1_圆柱体_Aabb*、*Bo1_面片_Aabb*、*Bo1_网格连接_Aabb*、*Bo1_PFacet_Aabb*、*Bo1_-_球体_Aabb*、*Bo1_四面体_Aabb*、*Bo1_墙_Aabb*。

class yade.wrapper.BoundFunctor(继承 函子 → 可序列化对象)用于创建/更新 *Body::bound*。

bases
此函子接受的类型有序列表（以字符串形式）。**dict**（（可序列化对象）辐角 *1*）→字典：返回属性字典。**label**(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.Bo1_Box_Aabb(继承 边界函子 → 函子 → 可序列化对象)创建/更新 *Aabb* 的 盒子。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict(*(可序列化对象)辐角1*) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs(*(可序列化对象)arg1*，*(字典)arg2*) →
None：从给定字典更新对象属性

class yade.wrapper.Bo1_ChainedCylinder_Aabb(继承 边界函子 → 函子 → 可序列化函子)创建 *Aabb* 的函子，来自 链式圆柱体。

aabbEnlargeFactor
包围盒的相对扩大量；若为负值则停用。

注意： 此属性用于创建远程相互作用，但只有与不会直接丢弃此类相互作用的 *IGeomFunctor* 搭配时才有意义： *Ig2_*圆柱体*_*圆柱体*_-ScGeom::interactionDetectionFactor* 应与 *AABB* 扩大因子的值相同。

bases
Or排序后的类型列表（以字符串形式），此函子可 概念。

dict(*(Serializable)辐角1*) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(Serializable)参数1*，*(字典)辐角2*) →
None：使用给定字典更新对象属性

class yade.wrapper.Bo1_Cylinder_Aabb(继承 边界函子 → 函子 → 可序列化对象)创建 *Aabb* 的函子，来源于 圆柱体。

aabbEnlargeFactor
包围盒的相对扩大幅度；若为负值则停用。

注意：此属性用于创建远程相互作用，但仅在.....时有意义 h

一个 *IGeomFunctor* 不会简单地丢弃此类相互作用： *Ig2_圆柱体_圆柱体_-ScGeom::interactionDetectionFactor* 应与 *AABB* 扩大因子相同。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用此功能并将 O.timingEnabled==设为 True，否则为空。

updateAttrs((可序列化对象)参数 *1*，(字典)辐角 *2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.Bo1_Facet_Aabb(继承 边界函子 → 函子 → 可序列化对象)创建/更新 *Aabb*的 面片。

bases
此函子接受的有序类型列表（以字符串表示）。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用该功能且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数 *1*、(字典)辐角 *2*) →
None：根据给定字典更新对象属性

class yade.wrapper.Bo1_GridConnection_Aabb(继承 边界函子 → 函子 →*Serializable*)
从 网格连接创建 *Aabb* 的函子。

aabbEnlargeFactor(=*-1*，停用)包围盒的相对扩大量；若为负值则停用。

bases
O按序排列的类型列表（以字符串表示），此函子接受。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用，且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数 *1*，(字典)辐角 *2*) →
None：根据给定的字典更新对象属性

class yade.wrapper.Bo1_PFacet_Aabb(继承 边界函子 → 函子 → 可序列化对象)创建 *Aabb* 的 *PFacet*。

aabbEnlargeFactor(=-1，停用)包围盒的相对扩张量；若为负值则停用。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict(（可序列化对象）辐角*1*）→字典：返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*）→ 无：根据给定字典更新对象属性

class yade.wrapper.Bo1_Sphere_Aabb(继承 边界函子 → 函子 → 可序列化对象)创建 *Aabb* 的球体。

aabbEnlargeFactor
包围盒的相对扩展量；若为负值则停用。

> **注意：** 此属性用于创建远程相互作用，但只有与不会直接丢弃此类相互作用的 *IGeomFunctor* 配合使用时才有意义： *Ig2_Sphere_Sphere_-ScGeom::interactionDetectionFactor* 的值应与 *AABB* 扩大因子相同。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(*(Serializable)arg1*）→字典：返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用 O.timingEnabled==True。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*）→ 无：根据给定字典更新对象属性

class yade.wrapper.Bo1_Tetra_Aabb(继承 边界函子 → 函子 → 可序列化对象)创建/更新 *Aabb* 一个 四面体

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(（可序列化对象）辐角*1*）→字典：返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用该功能并将 O.timingEnabled==设为 True，否则内容为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*）→ None：根据给定字典更新对象属性

class yade.wrapper.Bo1_Wall_Aabb(继承 边界函子 → 函子 → 可序列化对象)
创建/更新 *Aabb* 墙

bases
此函子接受的类型有序列表（以字符串表示）。

dict(*(可序列化对象)*辐角 *1*) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

timingDeltas
分派器自身的详细计时信息。除非在源代码中启用并且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(Serializable)*参数*1*，*(*字典*)*辐角 *2*) →
None：根据给定的字典更新对象属性

BoundDispatcher

class yade.wrapper.BoundDispatcher(继承 分派器 → 引擎 → 可序列化对象)根据接收到的参数类型分派 函子 。

activated(=*true*)引擎是否已激活（只能由碰撞器更改）

dead(=*false*)如果为 true，此引擎将完全不会运行；可用于暂时停用引擎，并在之后的某个时间点重新启用它。

dict(*(Serializable)*辐角 *1*) →字典:
返回属性字典。

dispFunctor(*(BoundDispatcher)*参数*1*，*(*形状*)*辐角 *2*) → 边界函子：返回针对给定参数进行分派的函子；如果没有可分派的函子，则返回 None；若分派存在歧义，则抛出异常。

dispMatrix(*(BoundDispatcher)*辐角 *1*[, *(*布尔值*)*名称=真]) → 字典:
返回包含分派矩阵内容的字典。

execCount此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所用时间的累计纳秒数（仅在*O.timingEnabled*==True时使用）。

functors
与此分派器关联的函子。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

minSweepDistFactor(=*0.2*)
扩大全部包围盒的最小距离；当计算出的 sweep- Dist 小于（minSweepDistFactor x sweepDist）时，此值将覆盖计算结果。由碰撞器更新。（自动更新）。

ompThreads(=*-1*)
引擎中要使用的线程数。若为 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含

OpenMP 并行区域（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时，从而形成每一层线程数不同的嵌套 OMP 循环。

sweepDist(=*0*)
扩大所有包围盒的距离，以防止碰撞器每一步都运行（此值仅应由碰撞器更改）。

targetInterv(=*-1*)
参见 *InsertionSortCollider::targetInterv* (自动更新)

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用，否则为空 e
代码和 *O.timingEnabled*==True。

updateAttrs((可序列化对象)参数*1*，(字典)辐角*2*) →
None：根据给定字典更新对象属性
updatingDispFactor(=*-1*)参见
InsertionSortCollider::updatingDispFactor (自动更新)

2.3.8 相互作用几何创建

IGeomFunctor

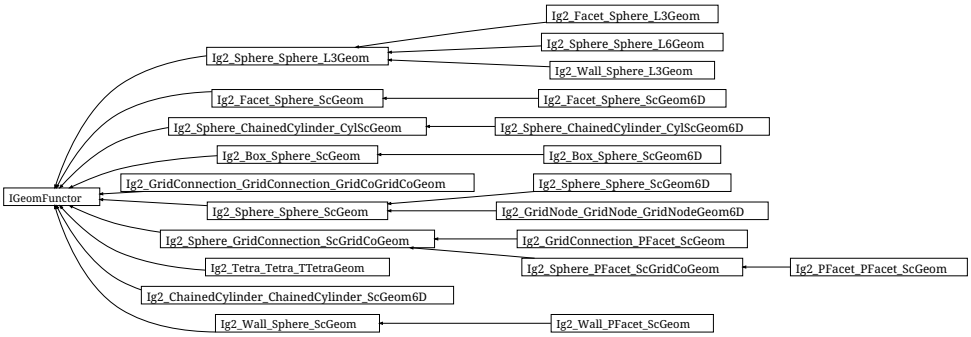


图 *34*：*IGeomFunctor* 的继承图。另请参见：*Ig2_Box_Sphere_ScGeom*、*Ig2_Box_-Sphere_ScGeom6D*、*Ig2_*链式圆柱体*_ChainedCylinder_ScGeom6D*、*Ig2_*面片*_Sphere_L3Geom*、*Ig2_*面片*_Sphere_ScGeom*、*Ig2_*面片*_Sphere_ScGeom6D*、*Ig2_*网格连接*_GridConnection_-GridCoGridCoGeom*、*Ig2_*网格连接*_PFacet_ScGeom*、*Ig2_*网格节点*_GridNode_GridNode-Geom6D*、*Ig2_PFacet_PFacet_ScGeom*、*Ig2_Sphere_*链式圆柱体*_CylScGeom*、*Ig2_Sphere_-ChainedCylinder_CylScGeom6D*、*Ig2_Sphere_*网格连接*_ScGridCoGeom*、*Ig2_Sphere_PFacet_-ScGridCoGeom*、*Ig2_Sphere_Sphere_L3Geom*、*Ig2_Sphere_Sphere_L6Geom*、*Ig2_Sphere_Sphere_Sc-Geom*、*Ig2_Sphere_Sphere_ScGeom6D*、*Ig2_*四面体*_Tetra_TTetraGeom*、*Ig2_*墙*_PFacet_ScGeom*、*Ig2_*墙*_Sphere_L3Geom*、*Ig2_*墙*_Sphere_ScGeom*。

class yade.wrapper.IGeomFunctor(继承 函子 → 可序列化对象)
用于创建/更新 *Interaction::geom* 对象。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict((可序列化对象)辐角*1*) →字典:
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接引用它 ly
来自 Python。

timingDeltas
分派器自身的详细计时信息。除非在源代码中启用并将 O.timingEnabled==设为 True，否则为空。

updateAttrs（可序列化对象）参数*1*，（字典）辐角*2*）→ 无：根据给定的字典更新对象属性

class yade.wrapper.Ig2_Box_Sphere_ScGeom(继承 *IGeomFunctor* → 函子 → 可序列化对象)创建相互作用几何 *ScGeom*，由 盒子 和 球体构成，用具有相同半径的投影虚拟球体表示盒子。

bases
此函子接受的类型有序列表（以字符串形式）。

dict（可序列化对象）辐角*1*）→字典：返回属性字典。

interactionDetectionFactor
将球体半径扩大此因子（如果 >1），以允许创建远程相互作用。

当 interactionDetectionFactor*(rad) > distance 时，将计算 InteractionGeometry。

注意： 此参数在功能上与 *Bo1_Sphere_Aabb::aabbEnlargeFactor*相关；该参数会创建更大的包围盒，且其值应与后者相同。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

timingDeltas
分派器本身内部的详细计时信息。除非在源代码中启用该功能且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(可序列化对象)arg1*，*(字典)arg2*）→ None：根据给定字典更新对象属性

class yade.wrapper.Ig2_Box_Sphere_ScGeom6D(继承 *Ig2_盒子_球体_ScGeom* → *IGeom*- 函子 → 函子 → 可序列化对象)
创建相互作用几何 *ScGeom6D*，由 盒子 和 球体表示，其中盒子由一个投影得到的同半径虚拟球体表示。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(*(可序列化对象)辐角*1*）→字典：返回属性字典。

interactionDetectionFactor
将球体半径乘以此因子进行扩大（如果 >1），以允许创建远距离相互作用。

当 interactionDetectionFactor*(rad) > distance 时，将计算相互作用几何。

注意： 此参数在功能上与 *Bo1_球体_Aabb::AABB 扩大因子*相关，它会创建更大的包围盒，因此应使用相同的值。

label(=取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，您可以直接从 Python 中引用它。

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → None：从给定字典更新对象属性

class yade.wrapper.Ig2_ChainedCylinder_ChainedCylinder_ScGeom6D(继承IGeomFunctor → 函子 → 可序列化对象)创建/更新一个ScGeom 实例，表示串联圆柱体之间的连接。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict((可序列化对象)辐角1) →字典：
返回属性字典。

halfLengthContacts(=真)
若为真，圆柱体节点之间的相互作用方式与半径为 0.5*长度的球体相同；否则，一个节点的尺寸为长度，另一个节点的尺寸为 0。两者的差异主要在于旋转轨迹的定义。

interactionDetectionFactor(=1)
将两个半径都乘以此因子（若为 >1），以便创建远距离相互作用。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为真，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无：
从给定字典更新对象属性

class yade.wrapper.Ig2_Facet_Sphere_L3Geom(继承 Ig2_球体_球体_L3Geom → IGeom函子 → 函子 → 可序列化对象)
增量计算 L3Geom f或接触面片 和 S球体。使用属性 o f Ig2_球体_球体_L3Geom。

approxMask
选择性启用几何近似（位掩码）；将要启用的近似对应的值相加 s 以启用。

1	使用前一个变换来变换速度（速度已知于中间步）， 而不是使用在 t=0.5 处通过四元数球面线性插值计算的中间步变换。
2	计算相对剪切位移时，不取平均（中间步）法向量， 而是使用前一个值
4	如果使用平均（步中）法向量，请勿重新归一化.....

默认情况下，掩码为零，因此不会使用这些近似。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict（可序列化对象）辐角1）→字典：返回属性字典。

distFactor(=1)
如果球体之间的距离不超过距离因子 *(r1+r2)，则创建相互作用。如果为负值，则将零向量法向变形设为初始值（否则，几何距离就是“零”距离）。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，您可以直接引用它 ly 来自 Python。

noRatch(=真)
查看*Ig2_球体_球体_ScGeom.avoidGranularRatcheting*。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

trsfRenorm(=*100*)
重新归一化变换的频率；如果为非正数，则永不重新归一化（模拟可能不稳定）

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → None：根据给定字典更新对象属性

class yade.wrapper.Ig2_Facet_Sphere_ScGeom(继承自*IGeomFunctor* → 函子 → 可序列化对象)

创建/更新一个 *ScGeom* 实例，用于表示 面片 与 球体的交集。面片的等效半径（*ScGeom.refR1*）取为球体半径的两倍。

bases
此函子接受的类型有序列表（以字符串表示）。

dict(*(*可序列化对象*)*辐角*1*) →字典:
返回属性字典。

hertzian(=*false*)
面片的等效半径（*ScGeom.refR1*）取为球体半径的 1e8 倍（更接近赫兹接触理论，其中该半径为无穷大）。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用 ly
中引用。

shrinkFactor(=*0*,不收缩)
面片内切圆的半径会减小，减小量为球体半径乘以收缩因子。根据由面片构成的表面上接触点的定义，给定表面并不连续，实际上会变成由三角形瓦片覆盖的表面，各个瓦片之间的间隙等于球体半径乘以2×*shrinkFactor*。如果为零，则不进行收缩。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用该功能且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(*可序列化对象*)arg1*，*(*字典*)arg2*) → None：从给定字典更新对象属性

class yade.wrapper.Ig2_Facet_Sphere_ScGeom6D(继承 *Ig2_面片_球体_ScGeom* →*IGeomFunctor* → 函子 可序列化对象)
创建一个相互作用几何 *ScGeom6D* ，由 面片 和 球体构成，用具有相同半径的投影虚拟球体表示该面片。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(*(*可序列化对象*)*辐角*1*) →字典:
返回属性字典。

hertzian(=*false*)
面片的等效半径（*ScGeom.refR1*）设为球体半径的 1e8 倍（更接近赫兹接触理论，其中该半径为无穷大）。

label(=取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，您可以直接引用它 ly
来自 Python。

shrinkFactor(=0,不收缩)
面片内切圆的半径会减小，减小量为球体半径乘以 收缩因子。根据由面片构成的表面上接触点的定义，给定表面并不连续，实际上会变成由三角形瓦片覆盖的表面，各瓦片之间的间隙等于球体半径乘以2×*shrinkFactor*。如果为零，则不进行收缩。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) →
None：根据给定字典更新对象属性

class yade.wrapper.Ig2_GridConnection_GridConnection_GridCoGridCoGeom(继承 IGe - om函子 → 函子 →可序列化对象)
创建/更新一个*GridCoGridCoGeom* 实例，用于表示两个网格连接之间接触点的几何体，包括相对旋转。

bases
O此函子接受的有序类型列表（以字符串形式） 收。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) →
None：根据给定字典更新对象属性

class yade.wrapper.Ig2_GridConnection_PFacet_ScGeom(继承 *Ig2_球体_网格连接 ScGridCoGeom* → *IGeom-_函子*→ 函子 可序列化对象)创建/更新一个 *ScGeom* 表示 面片 与 网格连接交集的实例。

bases
此函子接受的类型有序列表（以字符串表示）。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

interactionDetectionFactor(=1)
将两个半径都扩大此因子（如果为 >1），以便创建远距离相互作用。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

shrinkFactor(=0,不收缩)
面片内切圆的半径将减小球体半径乘以收缩因子所得的值。根据由面片构成的表面上接触点的定义，给定的表面并不连续，实际上会变成由三角形瓦片覆盖的表面，各瓦片之间的间隙等于球体半径乘以2×*收缩因子*。如果为零，则不进行收缩。

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → None：根据给定字典更新对象属性

class yade.wrapper.Ig2_GridNode_GridNode_GridNodeGeom6D(继承*Ig2_球体_球体_-ScGeom* → *IGeomFunctor* →函子 → 可序列化对象)

创建/更新一个 *GridNodeGeom6D* 实例，用于表示两个 网格节点之间接触点的几何特征，包括相对旋转。

avoidGranularRatcheting
定义相对速度以避免棘轮效应。此方法适用于球-球接触。若虚拟球的运动定义正确，它最终也适用于模拟球体的相互作用（即可转换为 ScGeom 类型的相互作用；例如参见 *Ig2_球体_链式圆柱体_-CylScGeom*）。

简要说明我们想要避免的内容：

考虑接触处的一个小弹性循环，有助于理解数值棘轮效应 n
两个颗粒：假设 b1 固定，对 b2 施加以下位移：

1. 沿法向平移 dx
2. 旋转a
3. 平移 -dx （回到初始位置）
4. 旋转 -a （回到初始方向）

如果用于定义旋转×分支中相对剪切的分支向量不是常量（通常是根据向量中心→接触点定义时），那么此循环结束时的剪切位移不为零：旋转 a 和 -a 分别与长度不同的分支相乘。

尽管位置和取向均未改变，循环结束时仍会产生有限的接触力，这与问题的弹性本质完全矛盾。也可以将其视为不一致的能量产生或损失。鉴于离散元模拟往往会在平衡附近产生振荡（阻尼质量弹簧系统），这可能会显著影响堆积体的演化，例如在恒定载荷下经过多次迭代后出现缓慢蠕变。

此处采用的方法旨在避免棘轮效应，遵循的是 McNamara 及其同事提出的方法。他们详细分析了棘轮效应问题——尽管他们评论时依据的循环与上文所示的循环不同。例如，可以在以下文献中找到有趣的讨论：[McNamara2008], 尽管该方法所建议的方案在此处并未完全采用（运动方程未纳入 alpha，这与 McNamara 等人的建议相矛盾）。

bases
此函子接受的类型有序列表（以字符串形式表示）。

creep(=否)
从相对旋转中减去旋转蠕变。旋转蠕变 *ScGeom6D::twistCreep*是一个四元数，必须在本构定律中更新，例如参见 定律2_-*ScGeom6D_CohFrictPhys_CohesionMoment*。

dict(*(可序列化对象)辐角1*) →字典：
返回属性字典。

interactionDetectionFactor
将两个半径都扩大 此因子（如果 >1），以便创建相距较远的 相互作用。

当 interactionDetectionFactor*(rad1+rad2) > dis- tance 时，将计算 InteractionGeometry。

注意：此参数在功能上与 *Bo1_Sphere_Aabb::aabbEnlargeFactor* 耦合，后者会生成更大的包围盒，两者的值应相同。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用并且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(可序列化对象)参数1*, *(字典)辐角2*) → 无:
根据给定字典更新对象属性

updateRotations(=真)
预先计算相对旋转。如果本构定律不需要旋转（例如，在三轴试验早期阶段，不带黏聚力和力矩地压缩球体时），将此项设为 false 可以加快模拟速度，但并非万无一失。只有在清楚自己在做什么时才更改此值。

class yade.wrapper.Ig2_PFacet_PFacet_ScGeom(继承 *Ig2_Sphere_PFacet_ScGridCo-Geom* → *Ig2_Sphere_网格连接_Sc-GridCoGeom* → *IGeomFunctor* → 函子→ 可序列化对象)

创建/更新一个 *ScGridCoGeom* 实例，用于表示以下对象的交集： 面片 和 球体。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(*(可序列化对象)辐角1*) →字典:
返回属性字典。

interactionDetectionFactor(=*1*)
将两个半径都扩大此因子（如果 >1），以便创建远距离相互作用。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可以直接从 Python 中引用。

shrinkFactor(=*0*,不收缩)
面片内切圆的半径会减小，减小值为球体半径乘以收缩因子。根据由面片构成的表面上接触点的定义，给定表面并非连续表面，实际上会变成由三角形瓦片覆盖的表面，各独立瓦片之间的间隙等于球体半径乘以2×*收缩因子*。如果为零，则不进行收缩。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(可序列化对象)arg1*, *(字典)arg2*) →
None：根据给定字典更新对象属性

class yade.wrapper.Ig2_Sphere_ChainedCylinder_CylScGeom(继承 *IGeomFunctor* →函子 → 可序列化对象)

创建/更新一个 *ScGeom* 实例，用于表示两个 球体的交集。

bases
此函子接受的类型有序列表（以字符串表示）。

dict(*(可序列化对象)辐角1*) →字典:
返回属性字典。

interactionDetectionFactor(=*1*)
将两个半径都扩大此因子（如果 >1），以允许创建远距离相互作用。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.**Ig2_Sphere_ChainedCylinder_CylScGeom6D**(继承 *Ig2_球体_- 链式圆柱体_CylSc- 几何* → *IGeomFunctor* →函子 → 可序列化对象)

创建/更新一个*ScGeom6D* 实例，用于表示两个球体之间接触点的几何信息，包括相对旋转。

bases
此函子接受的类型有序列表（以字符串表示）。

creep(=*false*)从相对旋转中减去旋转蠕变。旋转蠕变*ScGeom6D::twistCreep*是一个四元数，必须在本构定律中更新，例如参见定律*2_-ScGeom6D_CohFrictPhys_CohesionMoment*。

dict(*(Serializable)*辐角 *1*) →字典:
返回属性字典。

interactionDetectionFactor(=*1*)
将两个半径都扩大此因子（若为 >1），以便创建远程相互作用。

label(= 未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器本身的详细计时信息。除非在 e
源代码中启用 O.timingEnabled==True。

updateAttrs(*(Serializable)arg1*、*(dict)arg2*) → None:
根据给定字典更新对象属性

updateRotations(=*false*)
预先计算相对旋转。如果本构定律中不需要旋转（例如，在三轴试验初期，球体在没有黏聚力和力矩的情况下受压时），将此项设为 false 可以加快模拟速度，但并非万无一失。只有在了解自己所做操作的情况下，才更改此值。

class yade.wrapper.**Ig2_Sphere_GridConnection_ScGridCoGeom**(继承 *IGeomFunctor* →函子 → 可序列化对象)

创建/更新一个*ScGridCoGeom6D* 实例，用于表示*GricConnection* 与*Sphere* 之间接触点的几何体，包括相对旋转。

bases
此函子接受的类型有序列表（以字符串形式）。

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

interactionDetectionFactor(=*1*)
将两个半径都扩大此因子（如果为 >1），以便创建远距离相互作用。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器自身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
从给定字典更新对象属性

class yade.wrapper.Ig2_Sphere_PFacet_ScGridCoGeom(继承 *Ig2_球体_网格连接 ScGridCoGeom → IGeom函子_ →函子 →可序列化对象*)创建/更新一个 *ScGridCoGeom* 表示 *PFacet* 与 球体交集的实例。

bases
此函子接受的类型有序列表（以字符串形式）。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

interactionDetectionFactor(=1)
将两个半径都乘以此因子（如果为 >1），以允许创建远距离相互作用。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

shrinkFactor(=0,不收缩)
面片内切圆的半径会减小球体半径乘以收缩因子所得的值。根据由面片构成的表面上接触点的定义，给定表面并不连续，实际上会变成由三角形瓷砖覆盖的表面，各个瓷砖之间的间隙等于球体半径乘以2×*收缩因子*。如果为零，则不进行收缩。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)arg1，(字典)arg2) →
None：根据给定字典更新对象属性

class yade.wrapper.Ig2_Sphere_Sphere_L3Geom(继承 *IGeomFunctor* → 函子 → *Serial*)用于增量计算存储在 *L3Geom*中的 2 个 球体(或大小的配置的函子；该配置通过局部原点 $\mathbf{c}$ （接触点）和旋转矩阵 $\mathbf{T}$ （正交变换矩阵）定位于全局空间，其自由度为局部位移 $\mathbf{u}$ （沿一个法向量方向和两个剪切方向）；对于 *Ig2_Sphere_Sphere_L6Geom* 和 *L6Geom*，还额外存储 $\boldsymbol{\varphi}$。 $\mathbf{T}$的第一行，即局部 x轴，是接触法线，为简洁起见记作 $\mathbf{n}$ 。此外， $\mathbf{u}_0$ （以及 $\boldsymbol{\varphi}_0$）的准恒定值存储为 $\mathbf{u}$ （以及 $\boldsymbol{\varphi}$）的平移原点；因此，位移的当前值始终为 $\mathbf{u}^\circ - \mathbf{u}_0$。

设有两个球体，其半径为 r_i，位置为 $\mathbf{x}_i$，速度为 $\mathbf{v}_i$，角速度为 $\boldsymbol{\omega}_i$。

如果尚未发生接触，则当 $u_N = |\mathbf{x}_2^\circ - \mathbf{x}_1^\circ| - |f_d|(r_1 + r2) < 0$ 时，将创建接触，其中 f_d 是距离因子 （有时也称为“相互作用半径”）。如果 $f_d > 0$，则 $\mathbf{u}_{0x}$ 将初始化为u_N，否则初始化为 0。换句话说，如果将球体扩大 $|f_d|$ 后彼此接触，就会创建接触；如果 $\mathbf{u}_x - \mathbf{u} - 0x$ 为正，则“平衡距离”（即 f_d 为零时的距离）将设为当前距离；如果为负，则设为几何接触距离。

局部坐标轴（ $\mathbf{T}$的各行）最初定义如下：

- 局部 x 轴为 $\mathbf{n} = \mathbf{x}_l = \widehat{\mathbf{x}_2 - \mathbf{x}_1}$;
- 局部 y轴以任意但确定的方式定位：与 xz 平面对齐（如果是 $\mathbf{n}_y < \mathbf{n}_z$），否则与 xy 平面对齐；

- 局部 z轴 $\mathbf{z}_l = \mathbf{x}_l \times \mathbf{y}_l$。

如果两个球体之间已经发生接触，则会更新接触信息，以跟踪接触的刚体运动（不会改变球体的相互构型）以及相互构型的变化。刚体运动会变换局部坐标系，并可分解为刚性平移（影响 $\mathbf{c}$）和刚性旋转（影响 $\mathbf{T}$）；刚性旋转又可分解为垂直于法向量的旋转 $\mathbf{o}_r$ 和平行于法向量的旋转 $\mathbf{o}_t$（“扭转”）：

$$\mathbf{o}_r^{\ominus} = \mathbf{n}^- \times \mathbf{n}^\circ.$$

由于已知前一中间步（$t - \Delta t/2$）的速度，因此我们采用中间步法向量

$$\mathbf{n}^{\ominus} = \frac{\mathbf{n}^- + \mathbf{n}^\circ}{2}.$$

为确保数值稳定性， $\mathbf{n}^{\ominus}$ 计算完成后会重新归一化，除非*approxMask*禁止此操作。如果 *approxMask* 设置了相应的位，则不计算中间法向量，而是直接使用 $\mathbf{n}^{\ominus} \approx \mathbf{n}^-$。

平行于法向量的刚性旋转为

$$\mathbf{o}_t^{\ominus} = \mathbf{n}^{\ominus} \left(\mathbf{n}^{\ominus} \cdot \frac{\boldsymbol{\omega}_1^{\ominus} + \boldsymbol{\omega}_2^{\ominus}}{2} \right) \Delta t.$$

*Bran*这些向量 $\mathbf{b}_1$、 $\mathbf{b}_2$ （连接 $\mathbf{x}_1^\circ$、 $\mathbf{x}_2^\circ$ 与 $\mathbf{c}^\circ$ ）的计算取决于 *noRatch* （参见此处 ）.

$$\mathbf{b}_1 = \begin{cases} r_1 \mathbf{n}^\circ & \text{with noRatch} \\ \mathbf{c}^\circ - \mathbf{x}_1^\circ & \text{otherwise} \end{cases}$$

$$\mathbf{b}_2 = \begin{cases} -r_2 \mathbf{n}^\circ & \text{with noRatch} \\ \mathbf{c}^\circ - \mathbf{x}_2^\circ & \text{otherwise} \end{cases}$$

$\mathbf{c}^\circ$ 处的相对速度可计算为

$$\boldsymbol{\nu}_r^{\ominus} = (\tilde{\boldsymbol{\nu}}_2^{\ominus} + \boldsymbol{\omega}_2 \times \mathbf{b}_2) - (\boldsymbol{\nu}_1 + \boldsymbol{\omega}_1 \times \mathbf{b}_1)$$

其中 $\tilde{\boldsymbol{\nu}}_2$ 是 $\boldsymbol{\nu}_2$ 在周期性边界条件下不含平均场速度梯度的情况（参见*Cell.homoDeform*）。在数值实现中，使用移除入射速度的法向分量（因为该分量是直接计算的）。

用全局坐标表示的任意向量 $\mathbf{a}$ 在一个时间步长内的变换为

$$\mathbf{a}^\circ = \mathbf{a}^- + \boldsymbol{\nu}_r^{\ominus} \Delta t - \mathbf{a}^- \times \mathbf{o}_r^{\ominus} - \mathbf{a}^- \times \mathbf{t}_r^{\ominus}$$

其中这些增量分别表示相对剪切、垂直于 $\mathbf{n}$ 的刚性旋转，以及平行于 $\mathbf{n}$的刚性旋转。局部坐标系的方向（旋转矩阵 $\mathbf{T}$）按行更新，即

$$\mathbf{T}^\circ = \begin{pmatrix} \mathbf{n}_x^\circ & \mathbf{n}_y^\circ & \mathbf{n}_z^\circ \\ \mathbf{T}_{1,\bullet}^- - \mathbf{T}_{1,\bullet}^- \times \mathbf{o}_r^{\ominus} - \mathbf{T}_{1,\bullet}^- \times \mathbf{o}_t^{\ominus} \\ \mathbf{T}_{2,\bullet}^- - \mathbf{T}_{2,\bullet}^- \times \mathbf{o}_r^{\ominus} - \mathbf{T}_{\bullet}^- \times \mathbf{o}_t^{\ominus} \end{pmatrix}$$

除非通过*approxMask*禁止，否则该矩阵会重新归一化，并使用四元数球面插值计算步中变换，如下所示

$$\mathbf{T}^{\ominus} = \mathrm{Slerp}\left(\mathbf{T}^-; \mathbf{T}^\circ; t = 1/2\right).$$

根据*approxMask*，可以通过近似 $\mathbf{T}^{\ominus} = \mathbf{T}^-$来避免此项计算。

最后，当前位移计算如下

$$\mathbf{u}^\circ = u^- + \mathbf{T}^{\ominus} \boldsymbol{\nu}_r^{\ominus} \Delta t.$$

对于法向 com 对于分量，优先采用非增量求值 ，从而得到

$$\mathbf{u}_x^\circ = |\mathbf{x}_2^\circ - \mathbf{x}_1^\circ| - (r_1 + r_2)$$

如果对此函子调用 *L6Geom*，则局部旋转更新为

$$\boldsymbol{\varphi}^\circ = \boldsymbol{\varphi}^- + \mathbf{T}^\ominus \Delta t(\boldsymbol{\omega}_2 - \boldsymbol{\omega}_1)$$

approxMask
有选择地启用几何近似（位掩码）；将要启用的近似对应的值相加。

1	使用前一次变换来变换速度（这些速度已知于中间步）， 而不是将 t=0.5 处的四元数球面线性插值计算得到的步中变换。
2	计算相对剪切位移时，不要取平均（步中）法向量， 而应使用前一个值
4	如果使用平均（步中）法向量，则不要对其重新归一化.....

默认情况下，掩码为零，因此不会使用这些近似。

bases
此函子接受的类型有序列表（以字符串形式）。

dict（可序列化对象）辐角*1*）→字
典：返回属性字典。

distFactor(=*1*)
如果球体之间的距离不超过距离因子 *(r1+r2)，则创建相互作用。如果为负值，则将零向量法向变形设为初始值（否则，几何距离就是“零”值）。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

noRatch(=真)
参见 *Ig2_球体_球体_ScGeom.avoidGranularRatcheting*。

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

trsfRenorm(=*100*)
重新归一化 变换的频率；如果为非正值，则从不重新归一化（模拟可能不稳定）

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) →
None：根据给定字典更新对象属性

class yade.wrapper.Ig2_Sphere_Sphere_L6Geom(继承 *Ig2_球体_球体_L3Geom* → *IGeomFunctor* → 函子 → 可序列化对象)增量计算 *L6Geom* 两个球体的接触。

approxMask
选择性启用几何近似（位掩码）；将要启用的近似对应的值相加。

1	使用前一步的变换来变换速度（这些速度在中间步已知）， 而不是使用四元数球面线性插值在 t=0.5 处计算的中间步变换。
2	计算相对剪切位移时，不要采用平均（步中）法向量， 改用前一个值
4	如果使用平均（步中）法向量，则不要再次归一化.....

默认情况下，掩码为零向量，因此不会使用这些近似。 .

bases
此函子接受的有序类型列表（以字符串表示）。

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

distFactor(*=1*)
如果球体之间的距离不超过 距离因子 *(r1+r2)，则创建相互作用。如果该值为负，则将零向量法向变形设为初始值（否则，几何距离为“零”值）。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

noRatch(=真)
参见 *Ig2_球体_球体_ScGeom.avoidGranularRatcheting*。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用并且 O.timingEnabled==为 True，否则为空。

trsfRenorm(*=100*)
重新归一化 变换的频率；如果小于或等于零，则永不重新归一化（模拟可能不稳定）

updateAttrs(*(可序列化对象)参数1，(字典)辐角 2*) →
None：根据给定字典更新对象属性

class yade.wrapper.**Ig2_Sphere_Sphere_ScGeom**(继承自 *IGeomFunctor* → 函子 → 可序列化对象)

创建/更新表示两个*ScGeom* 之间接触点几何体的实例球体。

avoidGranularRatcheting
定义相对速度以避免棘轮效应。此定义适用于球-球接触。如果虚拟球体的运动定义正确，它最终也适用于模拟球体的相互作用（即可转换为 ScGeom 类型的相互作用；例如参见 *Ig2_球体_链式圆柱体_-CylScGeom*）。

简要说明我们要避免的内容:

要理解数值棘轮效应，最好考虑两个颗粒接触时的一个小弹性循环：假设 b1 固定，对 b2 施加以下位移:

1. 沿法向平移dx
2. 旋转a
3. 平移-dx （回到初始位置）
4. 旋转-a （回到初始方向）

如果用于定义 rotation×branch 中相对剪切的分支向量不是常量（通常是根据向量中心→接触点定义的），那么此循环结束时的剪切位移不为零：旋转a 和-a 分别乘以长度不同的分支。

这会导致循环结束时仍存在有限的接触力，尽管位置和取向均未发生变化，这与该问题的弹性本质完全矛盾。这也可以看作是<样式 id='1'>不一致的能量产生或损失</样式>。鉴于离散元模拟往往会在平衡状态附近产生振荡（阻尼质量弹簧系统），这可能会对堆积体的演化产生显著影响，例如在恒定载荷下的迭代过程中导致缓慢蠕变。

<样式 id='1'>这里为避免棘轮效应而采用的求解方案，是 McNamara 及其同事提出的方案。他们详细分析了棘轮效应问题——尽管他们所依据的循环与上文所示的循环不同。读者可以在例如 </样式> [<样式 id='4'>McNamara</样式>2008], <样式 id='7'>中找到有趣的讨论，尽管其建议的方案并未在此处完全采用（运动方程中未纳入 alpha，这与 McNamara 等人的建议相矛盾）。</样式>

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

interactionDetectionFactor
将两个半径都乘以此因子（如果 >1），以便创建远程相互作用。

当 interactionDetectionFactor*(rad1+rad2) > 距离时，将计算 InteractionGeometry。

注意：此参数在功能上与*Bo1_Sphere_Aabb::AABB 扩大因子*相关，它会生成更大的包围盒，两者的值应相同。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用该功能且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)*arg1*，(字典)*arg2*) →
None：从给定字典更新对象属性

class yade.wrapper.**Ig2_Sphere_Sphere_ScGeom6D**(继承 *Ig2_球体_球体_ScGeom* →*IGeomFunctor* → 函子 → 可序列化对象)
创建/更新一个 *ScGeom6D* 实例，用于表示两个球体接触点的几何体，包括相对旋转。

avoidGranularRatcheting
定义相对速度以避免棘轮效应。它适用于球-球接触。最终也适用于模拟球体的相互作用（即可以转换为 ScGeom 类型的相互作用），前提是虚拟球的运动定义正确（例如参见 *Ig2_Sphere_链式圆柱体_-CylScGeom*）。

对我们希望避免的问题的简要说明 :

要理解数值棘轮效应，最好考虑两个颗粒接触时的一个小弹性循环：假设 b1 固定，对 b2 施加以下位移：

1. 沿法向平移 dx
2. 旋转a
3. 平移 -dx （回到初始位置）
4. 旋转 -a （回到初始方向）

如果用于定义 rotation×branch 中相对剪切的分支向量不是恒定的（通常是根据 vector center→contactPoint 定义的情况），那么此循环结束时的剪切位移不为零：旋转 a 和 -a 分别乘以长度不同的分支。

尽管位置和取向没有变化，但在循环结束时仍会产生有限的接触力，这与问题的弹性本质完全矛盾。它

也可视为不一致的能量产生或损失。鉴于离散元模拟往往会在平衡状态附近产生振荡（阻尼质量弹簧系统），这可能会对堆积体的演化产生显著影响，例如在恒定载荷下，迭代过程中会出现缓慢蠕变。

这里采用的避免棘轮效应的方法，是 McNamara 及其同事提出的方法。他们详细分析了棘轮效应问题——尽管他们是基于一个不同于上文所示周期的周期进行注释的。例如，在 [McNamara2008]，中可以找到有趣的讨论，尽管其建议的求解方案在此处并未完全应用（运动方程没有纳入 alpha，这与 McNamara 等人所建议的内容相矛盾）。

bases
此函子接受的有序类型列表（以字符串形式表示）。

creep(=否)
从相对旋转中减去旋转蠕变。旋转蠕变 *ScGeom6D::twistCreep* 是一个四元数，必须在本构定律内部进行更新，例如参见 定律*2_-ScGeom6D_CohFrictPhys_CohesionMoment*。

dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

interactionDetectionFactor
将两个半径都乘以此因子（如果 >1），以便创建远距离相互作用。

当 interactionDetectionFactor*(rad1+rad2) > dis- tance 时，将计算 InteractionGeometry。

注意： 此参数在功能上与 *Bo1_Sphere_Aabb::aabbEnlargeFacto* 耦合 *r*，它会创建更大的包围盒，因此应使用相同的值。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接引用它 ly 来自 Python。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用，且 O.timingEnabled==True，否则为空。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*) → 无：根据给定字典更新对象属性

updateRotations(=真)
预先计算相对旋转。如果本构定律中不需要旋转，将此项设为 false 可以加快模拟速度（例如，在三轴试验早期阶段，无黏聚力和力矩的情况下压缩球体），但这并非万无一失。只有在清楚自己在做什么时才更改此值。

class yade.wrapper.Ig2_Tetra_Tetra_TTetraGeom(继承 *IGeomFunctor* → 函子 → 可序列化对象)创建/更新两个 四面体 （*TTetraGeom* 实例）

bases
函子接受的有序类型列表（以字符串形式表示）。

dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

label(=取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.Ig2_Wall_PFacet_ScGeom(继承 *Ig2_墙_球体_ScGeom* → *IGeom-* 函子 → 函子 → 可序列化对象)创建/更新一个*ScGeom* 实例，表示以下对象的交集：墙 和 *PFacet*。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(*(可序列化对象)辐角1*) →字典:
返回属性字典。

hertzian(=假)
墙的等效半径（*ScGeom.refR1*）选为球体半径的 1e8 倍（更接近赫兹接触理论，其中半径为无穷大）。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

noRatch(=真)
避免颗粒棘轮效应

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无:
从给定字典更新对象属性

class yade.wrapper.Ig2_Wall_Sphere_L3Geom(继承 *Ig2_球体_球体_L3Geom* → *IGe- om*函子→ 函子 → 可序列化对象)
增量计算 *L3Geom* 用于 墙 与 球体之间接触的几何信息。使用 *Ig2_* - 球体_球体_*L3Geom*。

approxMask
选择性启用几何近似（位掩码）；将要启用的近似对应的值相加。

1	使用之前的变换来变换速度（速度在中间步已知）， 而不是使用四元数球面线性插值在 t=0.5 处计算出的中间步变换。
2	计算相对剪切位移时，不要使用平均（半步）法向量， 改用前一个值
4	如果使用平均（半步）法向量，则不要重新归一化.....

默认情况下，掩码为零向量，因此不会使用这些近似。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict（可序列化对象）辐角*1*）→字
典：返回属性字典。

distFactor(=*1*)
如果球体之间的距离不超过 距离因子 *(r1+r2)，则创建相互作用。若该值为负，则将零法向变形设为初始值（否则，几何距离为“零”）。

label(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用它中引用。

noRatch(=*true*)
请参阅 *Ig2_球体_球体_ScGeom.*避免颗粒棘轮效应。

timingDeltas
分派器本身内部的详细计时信息。除非在源代码中启用并将 O.timingEnabled==设为 True，否则为空。

trsfRenorm(=*100*)
重新归一化变换的频率；若为非正值，则永不重新归一化（模拟可能不稳定）

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*) → None：根据给定字典更新对象属性

class yade.wrapper.Ig2_Wall_Sphere_ScGeom(继承自*IGeomFunctor* → 函子 → 可序列化对象)

创建/更新一个*ScGeom* 实例，用于表示墙与球体的交集。墙的等效半径（*ScGeom.refR1*）取为球体的半径。

bases
此函子接受的类型有序列表（以字符串表示）。

dict(*(*可序列化对象*)*辐角*1*) →字典:
返回属性字典。

hertzian(=否)墙的等效半径（*ScGeom.refR1*）选为球体半径的 1e8 倍（更接近赫兹接触理论，其中半径为无穷大）。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

noRatch(=是)避免颗粒棘轮效应

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → None：根据给定字典更新对象属性

IGeomDispatcher

class yade.wrapper.IGeomDispatcher(继承分派器 → 引擎 → 可序列化对象)根据收到的参数类型调用函子。dead(=否)如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并在稍后将其重新启用。dict(*(*可序列化对象*)*辐角*1*) →字典：返回属性字典。dispFunctor(*(IGeomDispatcher)*参数*1*、*(*形状*)*参数*2*、*(*形状*)*辐角*3*) → IGeomFunctor：返回针对给定参数分派的函子；如果没有分派，则返回 None；存在歧义分派时会抛出异常。dispMatrix(*(IGeomDispatcher)*辐角*1*[, *(*布尔值*)*名称=*True*]) → 字典：返回包含分派矩阵内容的字典。

execCount
此引擎的累计运行次数（仅在*O.timingEnabled*==True时使用）。

execTime
此引擎运行所花费的累计时间（纳秒），仅在*O.timingEnabled*==True时使用。

functors
与此分派器关联的函子。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由 'yade -jN' 定义的 N（不过此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎组合使用时；这会导致嵌套 OMP 循环在每一层使用不同数量的线程。

timingDeltas
引擎本身内部的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)arg1*, *(字典)*辐角*2*) → None：根据给定字典更新对象属性

2.3.9 相互作用物理创建

IPhysFunctor

class yade.wrapper.IPhysFunctor(继承自 函子 → 可序列化对象)用于创建/更新 *Interaction::phys* 对象的函子，其对象来自 刚体的材料属性。

bases
该函子接受的有序类型列表（以字符串形式表示）。

dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用并且 O.timingEnabled==为 True，否则为空。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*) → None：根据给定字典更新对象属性

class yade.wrapper.Ip2_2xInelastCohFrictMat_InelastCohFrictPhys(继承 *IPhysFunc-tor* → 函子 → 可序列化对象)

生成带力矩的黏聚摩擦相互作用。用于接触定律 *Law2_Sc-_Geom6D_InelastCohFrictPhys_CohesionMoment*。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(*(可序列化对象)*辐角*1*) →字典：返回属性字典。



图 *35*：*IPhysFunctor* 的继承图。另请参阅：*Ip2__2xInelastCohFrictMat__Inelast- CohFrictPhys*、*Ip2__BubbleMat__BubbleMat__BubblePhys*、*Ip2__CohFrictMat__CohFrictMat__CohFrict- Phys*、*Ip2__CpmMat__CpmMat__CpmPhys*、*Ip2__ElastMat__ElastMat__NormPhys*、*Ip2__ElastMat__Elast- Mat__NormShearPhys*、*Ip2__FrictMatCDM__FrictMatCDM__MindlinPhysCDM*、*Ip2__FrictMat__Cpm- Mat__FrictPhys*、*Ip2__FrictMat__FrictMatCDM__MindlinPhysCDM*、*Ip2__FrictMat__FrictMat__Capillary- Phys*、*Ip2__FrictMat__FrictMat__FrictPhys*、*Ip2__FrictMat__FrictMat__LubricationPhys*、*Ip2__FrictMat__- FrictMat__MindlinCapillaryPhys*、*Ip2__FrictMat__FrictMat__MindlinPhys*、*Ip2__FrictMat__FrictMat__- ViscoFrictPhys*、*Ip2__FrictMat__FrictViscoMat__FrictViscoPhys*、*Ip2__FrictViscoMat__FrictViscoMat__- FrictViscoPhys*、*Ip2__JCFpmMat__JCFpmMat__JCFpmPhys*、*Ip2__LudingMat__LudingMat__LudingPhys*、*Ip2__MortarMat__MortarMat__MortarPhys*、*Ip2__ViscElCapMat__ViscElCapMat__ViscElCapPhys*、*Ip2__- ViscElMat__ViscElMat__ViscElPhys*、*Ip2__WireMat__WireMat__WirePhys*。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器自身的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
从给定字典更新对象属性

class yade.wrapper.Ip2_BubbleMat_BubbleMat_BubblePhys(继承IPhysFunctor →函子→ 可序列化对象)生成气泡相互作用。用于接触定律 Law2_ScGeom_BubblePhys_Bubble。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器内部计时的详细信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
从给定字典更新对象属性

class yade.wrapper.Ip2_CohFrictMat_CohFrictMat_CohFrictPhys(继承 IPhysFunctor →函子 → 可序列化对象)

生成带力矩的黏聚摩擦相互作用，用于接触定律 定律2_Sc-Geom6D_CohFrictPhys_CohesionMoment。法向和剪切刚度及摩擦的定义与 Ip2_FrictMat_FrictMat_FrictPhys相同，详情请参阅该处的文档。

法向和剪切分量相关的黏附力由 CohFrict-Mat::normalCohesion (C_n) 和 CohFrictMat::shearCohesion (C_s) 计算。对于尺寸为 R_1 的颗粒，R_2 黏附力为 $a_i = C_i \min(R_1, R_2)^2$，$i = n$，$s$。

扭转和滚动刚度与剪切刚度成比例，其无量纲因子分别为 alphaKtw 和 alphaKr，因此转动刚度定义为 $k_s \alpha_i R_1 R_2$, $i = tw\,r$

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict（可序列化对象）辐角1) →字典:
返回属性字典。

frictAngle(=未初始化)
决定如何计算相互作用摩擦角的MatchMaker 实例。如果 None，则使用最小值。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接引用它 ly
来自 Python。

normalCohesion(=已取消初始化)
用于确定拉伸强度的 MatchMaker 实例

setCohesionNow(=假)
若为 true，则在当前时间步长内为全部现有接触赋予黏聚力。该标志会自动设为 false，因此只会在当前时间步长内进行赋值。

setCohesionOnNewContacts(=假)
若为真，则为所有新接触赋予黏聚力。若为假，则只有现有接触可以具有黏聚力（另 o

参见 *Ip2_CohFrictMat_CohFrictMat_CohFrictPhys::setCohesionNow*），且新接触只有摩擦作用。

shearCohesion(=未初始化)
实例：*MatchMaker* 用于确定剪切强度中的黏聚部分（可能会根据 *CohFrictPhys::cohesionDisablesFriction* 添加摩擦项）

timingDeltas
分派器自身的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs(*(*可序列化对象*)arg1*，*(*字典*)arg2*) → None：根据给定字典更新对象属性

class yade.wrapper.Ip2_CpmMat_CpmMat_CpmPhys(继承自 *IPhysFunctor* → 函子 → 可序列化)

将 2 个 *CpmMat* 实例转换为 *CpmPhys*，并使用相应的参数。如果材料不同，则使用简单的（算术）平均值。如果两个颗粒共享同一材料，则直接复制参数。详情请参阅 *CPM* 模型。

E(=未初始化)MatchMaker 的实例，用于确定如何计算相互作用的法向模量。如果 None，则使用平均值。

bases
此函子接受的类型有序列表（以字符串形式表示）。

cohesiveThresholdIter(*=10*)
新接触是否应具有黏聚性？在此迭代编号之前创建的接触具有黏聚性，此后则不具有黏聚性。如果为 0，则它们永远不会具有黏聚性。如果为负数，则它们始终会以黏聚状态创建（默认为 10）。

dict(*(*可序列化对象*)*辐角 *1*) →字典：
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接通过它引用 ly
在 Python 中。

timingDeltas
Dispatcher 内部的详细计时信息。除非在源代码中启用且 O.timingEnabled== 为 True，否则为空。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角 *2*) → None：从给定字典更新对象属性

class yade.wrapper.Ip2_ElastMat_ElastMat_NormPhys(继承*IPhysFunctor* → 函子 → 可序列化对象)创建一个*NormPhys*，由两个弹性材料。待办事项。实验性

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict(*(*可序列化对象*)*辐角 *1*) →字典：
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接引用它 ly
（来自 Python）。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角 *2*) → 无：根据给定的字典更新对象属性

class yade.wrapper.Ip2_ElastMat_ElastMat_NormShearPhys(继承 *IPhysFunctor* → 函子-器 →*Serializa ble*)

创建一个 *NormShearPhys*，由两个 弹性材料生成。待办。实验性

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器内部的详细计时信息。除非在 e
源代码和 O.timingEnabled==True 中启用。

updateAttrs（可序列化对象）参数*1*，（字典）辐角 *2*) → None：根据给定字典更新对象属性

class yade.wrapper.Ip2_FrictMatCDM_FrictMatCDM_MindlinPhysCDM(继承 *IPhysFunctor*→函子 →可序列化对象)

创建一个 *MindlinPhysCDM*，由两个 *FrictMatCDMsExts*构成。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict((可序列化对象)参数*1*) →字典:
返回属性字典。

frictAngle(=未初始化)用于*MatchMaker* 确定如何计算相互作用摩擦角的实例。如果 None，则使用最小值。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用该功能且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数*1*，(字典)参数*2*) →
None：根据给定的字典更新对象属性

class yade.wrapper.Ip2_FrictMat_CpmMat_FrictPhys(继承 *IPhysFunctor* → 函子 →可序列化对象)

将 *CpmMat* 实例和 *FrictMat* 实例转换为 *FrictPhys*，并使用相应的参数（杨氏模量、泊松比、摩擦角）。如果材料参数不同，则使用简单的（算术）平均值。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

frictAngle(=未初始化)
参见 *Ip2_FrictMat_FrictMat_FrictPhys*。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用此功能且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → None：根据给定字典更新对象属性

class yade.wrapper.Ip2_FrictMat_FrictMatCDM_MindlinPhysCDM(继承 *IPhysFunctor* → 函子 → *Serializable*)

创建一个 *MindlinPhysCDM* 由一个 FrictMat 实例和一个 FrictMatCDM 实例构成。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict（（可序列化对象）辐角1）→字典：返回属性字典。

frictAngle(=未初始化)
的实例*MatchMaker*，用于确定如何计算相互作用摩擦角。如果 None，则使用最小值。

label(= 未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
Dispatcher 内部的详细计时信息。除非在源代码中启用该功能且 O.timingEnabled==为 True，否则为空。

updateAttrs（（可序列化对象）参数1，（字典）辐角2）→ None：根据给定字典更新对象属性

class yade.wrapper.Ip2_FrictMat_FrictMat_CapillaryPhys(继承 *IPhysFunctor* → 函子 → 可序列化对象)适用于 *定律2_ScGeom_CapillaryPhys_毛细作用*。

这些关系中会计算所有交互属性。

警告： 与其他 *Ip2* 函子一样，大多数属性仅在新建相互作用时计算一次。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict((可序列化对象)辐角1）→字典：返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
Dispatcher 内部计时的详细信息。除非在源代码中启用，并且 O.timingEnabled== 为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → None：根据给定的字典更新对象属性

class yade.wrapper.Ip2_FrictMat_FrictMat_FrictPhys(继承 *IPhysFunctor* → 函子 → 可序列化对象)

根据两个 *FrictPhys* 创建一个 *FrictMats*。这里将单个球体在点载荷作用下的柔度定义为 $1/(E.D)$，其中 E 是球体的刚度，D 是其直径。接触本身的柔度取为每个球体柔度的总和，即 $1/(E_1.D_1) + 1/(E_2.D_2)$适用于一般情况，而 $2/(E.D)$ 适用于尺寸和刚度均相等的特殊情况。请注意，柔度求和等价于对刚度取调和平均后求和。这一推理

应用于法向量和切向（例如 [Scholtes2009a] 中所述），因此接触刚度的一般形式为：

$k = \frac{E_1 D_1 * E_2 D_2}{E_1 D_1 + E_2 D_2} = \frac{k_1 * k_2}{k_1 + k_2}$, with $k_i = E_i D_i$.
在上述方程中， E_i 取为*FrictMat::young* 球体 i 的法向刚度，以及*FrictMat::young* × *ElastMat::poisson* 作为剪切刚度。对于*ViscElMat* 与*FrictMat*之间的接触，请务必设置 *FrictMat::young* 和*FrictMat::poisson*，否则将使用默认值。

接触摩擦根据*Ip2_FrictMat_FrictMat_FrictPhys::frictAngle* （默认取两种材料中的较小值）。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict（可序列化对象）辐角*1*）→字典：返回属性字典。

frictAngle(=未初始化)
用于*MatchMaker* 实例，确定如何计算相互作用摩擦角。如果 None，使用最小值。

kn(=取消初始化)
用于*MatchMaker* 实例，确定如何计算相互作用法向刚度。如果 None，则使用调和平均。

ks(=取消初始化)
用于*MatchMaker* 实例，确定如何计算相互作用剪切刚度。如果 None，则使用调和平均。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用该功能且 O.timingEnabled==True，否则为空。

updateAttrs(*(可序列化对象)arg1*，*(字典)arg2*）→ None：根据给定字典更新对象属性

class yade.wrapper.**Ip2_FrictMat_FrictMat_LubricationPhys**(继承 *IPhysFunctor* →函子 →可序列化对象)

Ip2 根据两个 Material 实例创建 LubricationPhys。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict(*(可序列化对象)*辐角*1*）→字典：返回属性字典。

eps(=*0。001*)粗糙度：接触凸体半径增大的比例

eta(=*1*)流体黏度 [帕·秒]

keps(=*1*)
凸体的无量纲刚度系数，相对于表面刚度（最终刚度为 keps*kn）。仅在使用分辨率方法=0 时使用；使用分辨率>0 时，该系数始终为 1。 [-]

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用该功能且 O.timingEnabled==True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → None：从给定字典更新对象属性

class yade.wrapper.Ip2_FrictMat_FrictMat_MindlinCapillaryPhys(继承 *IPhysFunctor* → 函子 → *Serializ-* 可序列化对象)

适用于以下定律的关系2_ScGeom_CapillaryPhys_毛细作用
在这些关系中，会计算所有交互属性。

> **警告：** 与其他*Ip2* 函子一样，大多数属性仅在交互新建时计算一次。

bases
此函子接受的类型有序列表（以字符串表示）。

betan(=未初始化)法向黏性阻尼比 β_n。

betas(=未初始化)
切向黏性阻尼比 β_s。

dict(*(可序列化对象)辐角1*) →字典:
返回属性字典。

en(=未初始化)法向恢复系数 e_n。

es(=未初始化)切向恢复系数 e_s。

eta(*=0。0*)用于确定塑性弯矩的系数

gamma(*=0。0*)
每单位接触表面的表面能参数 [J/m^2]，用于推导 DMT公式
来自 HM

krot(*=0。0*)力矩接触定律的转动刚度

ktwist(*=0。0*)力矩接触定律的扭转刚度

label(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接引用它
来自 Python。

timingDeltas
分派器自身的详细计时信息。除非在源代码中启用此功能且 O.timingEnabled==True，否则此项为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → None：根据给定字典更新对象属性

class yade.wrapper.Ip2_FrictMat_FrictMat_MindlinPhys(继承 *IPhysFunctor* → 函子→ 可序列化对象)

根据赫兹-明德林公式（在 PFC 中实现），计算获得法向和剪切刚度所需的物理参数。黏性阻尼系数 c_n、 c_s 可以通过黏性阻尼比（β_n、 β_s）或恢复系数（e_n、 e_s）指定。

如果给出了黏性阻尼比 β_n (β_s)，则将其直接赋给 *MindlinPhys.betan*(*MindlinPhys.betas*)，并将黏性阻尼系数计算为 $c_n = 2 \cdot \beta_n \cdot \sqrt{m_{bar} \cdot k_n}$(
$c_s = 2 \cdot \beta_s \cdot \sqrt{m_{bar} \cdot k_s}$ k_n k_s $k_n =$)，其中
$3/2 \cdot k_{no} \cdot u_N^{0.5}$ $k_s = k_{so} \cdot u_N^{0.5}$ $k_{no} = 4/3 \cdot E \cdot \sqrt{R}$ $k_{so} = 2 \cdot \sqrt{4 \cdot R \cdot G/(2-\nu)}$
（）为切向刚度，法向（剪切）刚度。代入（）和（），可得$c_n = 2 \cdot \beta_n \cdot \sqrt{m_{bar} \cdot \sqrt{2 \cdot E \cdot \sqrt{R \cdot u_N}^{0.25}}}$(
$c_s = 2 \cdot \beta_s \cdot \sqrt{m_{bar} \cdot \sqrt{4 \cdot \sqrt{R \cdot G/(2-\nu) \cdot u_N}^{0.25}}}$)，其中 m_{bar}， R， E， G 分别为相互作用颗粒的有效质量和平均半径、弹性模量和剪切模量。

如果给出的是恢复系数 e_n，则使用 $\beta_n = -(\log e_n)/\sqrt{\pi^2 + (\log e_n)^2}$ 计算法向黏性阻尼比。e_s 切向恢复系数视为 e_n $c_n = c_s = \alpha \cdot \sqrt{m_{bar} \cdot u_N}^{0.25}$，并将黏性阻尼系数计算为，其中$\alpha = 2 \cdot \sqrt{5/6 \cdot \beta_n} \cdot \sqrt{2 \cdot E \cdot \sqrt{R}}$，即 $c_n = c_s = 2 \cdot \sqrt{5/6 \cdot \beta_n \cdot \sqrt{m_{bar} \cdot \sqrt{2 \cdot E}} \cdot \sqrt{R \cdot u_N}^{0.25}}$。

在这两种情况下，黏性力计算为 $F_{n,\ viscous} = c_n \cdot v_n$ ($F_{s,\ viscous} = c_s \cdot v_s$)，其中 v_n (v_s) 相对速度的法向（切向）分量。适用以下规则：

如果 β_n 和 β_s 被使用，那么 *MindlinPhys.alpha* =0；如果 e_n 被定义，那么 *Mindlin-Phys.betan* =*MindlinPhys.betan* =0.0。

同时指定 e_n 和 β_n （e_s 和 β_s）会导致错误（异常）。

如果既未指定 e_n 也未指定 β_n，则 *MindlinPhys.betan* 将使用零值；不会产生黏性效应。

如果既未指定 e_s 也未指定 β_s，则 *MindlinPhys.betan* 的值也将用于 *MindlinPhys.betas* 。

若要在法向和剪切接触方向上考虑不同的黏性系数，请使用 β_n、 β_s，而不是 e_n。

这些 e_n、 β_n、 e_s、 β_s 是 *MatchMaker* 对象；它们可以由浮点值构造，从而始终返回常量值。示例请参见 scripts/examples/spheresFactory.py，其中展示了如何根据参数组合，为不同的接触材料指定 e_n。

bases
　此函子接受的有序类型列表（以字符串形式表示）。

betan(=取消初始化)法向黏性阻尼比 β_n。

betas(=取消初始化)
　切向黏性阻尼比 β_s。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

en(=取消初始化)法向恢复系数 e_n。

es(=取消初始化)
　切向恢复系数 e_s。

eta(=*0*。*0*)用于确定塑性弯矩的系数

frictAngle(=未初始化)用于*MatchMaker* 确定如何计算相互作用的摩擦角的实例。如果None，则使用最小值。

gamma(=*0*。*0*)
　表面能参数 [J/m^2]，用于推导 DMT 公式的每单位接触表面 n
　来自 HM

krot(=*0*。*0*)力矩接触定律的转动刚度

ktwist(=0.0)
力矩接触定律的扭转刚度

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用此功能且 O.timingEnabled==True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) →
None：根据给定字典更新对象属性

class yade.wrapper.Ip2_FrictMat_FrictMat_ViscoFrictPhys(继承 *Ip2_FrictMat_FrictMat_FrictPhys* → *IPhys函-* 子 → 函子 → 可序列化对象)

创建一个 *FrictPhys* 由两个 *FrictMats*。这里将球体在对称点载荷作用下的柔度定义为 1/(E.r)，其中 E 是球体的刚度，r 是其半径；对应于柔度 1/(2.E.r)=1/(E.D)，来自每个接触点。接触本身的柔度是每个球体柔度的总和，即一般情况下为 1/(E.D1)+1/(E.D2)，特殊情况下尺寸相等时为 1/(E.r)。注意，将柔度相加相当于对刚度取调和平均，这正是 kn 在 *Ip2_FrictMat_FrictMat_FrictPhys* 函子中

单个球体的剪切刚度 ks 由材料参数 *ElastMat::poisson*定义，即 ks=poisson*kn，而相互作用的剪切刚度也将采用调和平均计算。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

frictAngle(=取消初始化)
用于确定如何计算相互作用摩擦角的 *MatchMaker* 实例。如果 None，则使用最小值。

kn(=取消初始化)
.....的实例*MatchMaker* 用于确定如何计算相互作用的法向刚度。如果 None，则使用调和平均值。

ks(=未初始化)
.....的实例*MatchMaker* 用于确定如何计算相互作用的剪切刚度。如果 None，则使用调和平均值。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器内部的详细计时信息。除非启用，否则为空 e
源代码和 O.timingEnabled==为 True。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
根据给定字典更新对象属性

clas s yade.wrapper.Ip2_FrictMat_FrictViscoMat_FrictViscoPhys(继承*IPhysFunctor* 函子 → 可序列化对象)

将 *FrictMat* 和 *FrictViscoMat* 实例转换为 *FrictViscoPhys*，并使用相应的参数。基本上，该函子对应于 *Ip2_FrictMat_FrictMat_FrictPhys* 唯一的区别是可以考虑法向方向上的阻尼。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

frictAngle(=未初始化)
实例 *MatchMaker* 用于确定如何计算相互作用摩擦角。如果 None，则使用最小值。

kRatio(=未初始化)
实例 *MatchMaker* 用于确定如何计算相互作用的剪切接触刚度。如果未提供此值，则使用两个碰撞材料的弹性属性（即泊松比）来计算刚度。

kn(=未初始化)
实例 *MatchMaker*，用于确定如何计算相互作用的法向接触刚度。如果未指定此值，则使用两种碰撞材料的弹性属性（即杨氏模量）来计算刚度。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

timingDeltas
分派器本身内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs(*(可序列化对象)参数1*, *(字典)辐角 2*) →
None：根据给定字典更新对象属性

class yade.wrapper.Ip2_FrictViscoMat_FrictViscoMat_FrictViscoPhys(继承 *IPhys-* 函子 → 函- 子→ 可序列化对- 象)

将 2 个 *FrictViscoMat* 实例转换为具有相应参数的 *FrictViscoPhys* 。基本上，此函子对应于 *Ip2_FrictMat_FrictMat_FrictPhys*，唯一的区别是可以考虑法向阻尼。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

frictAngle(=未初始化)
用于确定如何计算相互作用摩擦角的 *MatchMaker* 实例。如果 None，则使用最小值。

kRatio(=未初始化)
实例 *MatchMaker*，用于确定如何计算相互作用的剪切接触刚度。如果未提供此值，则使用两种碰撞材料的弹性属性（即泊松比）来计算刚度。

kn(=未初始化)
实例 *MatchMaker*，用于确定如何计算相互作用的法向接触刚度。如果未提供此值，则使用两种碰撞材料的弹性属性（即杨氏模量）来计算刚度。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用该功能且 O.timingEnabled==为 True，否则内容为空。

updateAttrs(*(可序列化对象)参数1*, *(字典)辐角 2*) →
None：根据给定字典更新对象属性

class yade.wrapper.Ip2_JCFpmMat_JCFpmMat_JCFpmPhys(继承 *IPhysFunctor* → 函子 →可序列化对象)将 2 个 *JCFpmMat* 实例转换为一个 *JCFpmPhys* 实例，并使用相应参数。详情请参阅 *JCFpmMat* 和 [Duriez2016]。

bases
此函子接受的有序类型列表（以字符串形式表示）。

cohesiveTresholdIteration(=*1*)
新接触是否应具有黏聚性？如果严格为负，则无论如何都会具有黏聚性。如果为正，则在此迭代之前会具有黏聚性，此后则不会。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

label(=未初始化)的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

timingDeltas
分派器本身的详细计时信息。除非在 t he 源代码和 O.timingEnabled==True。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) →
None：根据给定字典更新对象属性

weibullCutOffMax(=*10*)
用于截去威布尔分布的相互作用面积中最大值的因子。

weibullCutOffMin(=*0*。)用于截去威布尔分布的相互作用面积中最小值的因子。

xSectionWeibullScaleParameter(=*1*)
根据威布尔分布，为横截面积生成相互作用半径时使用的尺度参数（仅更改强度准则）。任何非 0 值均可激活。需要与形状参数结合使用

xSectionWeibullShapeParameter(=*0*)
根据威布尔分布，为横截面积生成相互作用半径时使用的形状参数（仅改变强度准则）。取任何非 0 值时启用。需要与 尺度参数) 配合使用。

class yade.wrapper.Ip2_LudingMat_LudingMat_LudingPhys(继承 *IPhysFunctor* →函子→ 可序列化对象)

使用连续连接规则将 2 个 *LudingMat* 转换为 *LudingPhys* 。

bases
此函子接受的类型有序列表（以字符串表示）。

dict(*(*可序列化对象*)**arg1*) →字典:
返回属性字典。

label(=未初始化)的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs(*(*可序列化对象*)**arg1*，*(*字典*)**arg2*) →
None：从给定字典更新对象属性

class yade.wrapper.Ip2_MortarMat_MortarMat_MortarPhys(继承 *IPhysFunctor* →*Functor*→ 可序列化对象)
Ip2 根据两个 MortarMat 实例创建 MortarPhys。

bases此函子接受的类型有序列表（以字符串形式表示）。

c
ohesiveThresholdIter(=*2*)
新的接触是否应具有黏聚性？在此迭代#之前，它们会具有黏聚性；之后则不会。如果<=0，它们将永远不会具有黏聚性。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接引用它

y

来自 Python。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.Ip2_ViscElCapMat_ViscElCapMat_ViscElCapPhys(继承 *Ip2_*
ViscEl-Mat ViscElMat Vis-_ _cElPhys→ *IPhys-* 函子→ 函子→ 可序列化对象)

使用连续连接规则，将 2 个 *ViscElCapMat* 转换为 *ViscElCapPhys* 。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

en(=未初始化)MatchMaker 的 实例 用于确定法向方向上的恢复系数

et(=未初始化)的实例 *MatchMaker* 用于确定切向方向上的恢复系数

n

frictAngle(=未初始化)的实例 *MatchMaker* 用于确定如何计算相互作用摩擦角。如果 None，则使用最小值。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

tc(=未初始化)的实例 *MatchMaker* 用于确定接触时间

timingDeltas
分派器内部计时的详细信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(*可序列化对象*)arg1*，*(*字典*)arg2*) →
None：根据给定字典更新对象属性

class yade.wrapper.Ip2_ViscElMat_ViscElMat_ViscElPhys(继承 *IPhysFunctor* →函子→ 可序列化对象)根据连续连接规则，将 2 个 *ViscElMat* 转换为 *ViscElPhys* 。bases此函子接受的类型有序列表（以字符串形式表示）。dict(*(*可序列化对象*)*辐角 *1*) →字典：返回属性字典。

en(=未初始化)
实例*MatchMaker* 用于确定法向恢复系数

et(=未初始化)
实例*MatchMaker* 用于确定切向恢复系数

frictAngle(=未初始化)实例*MatchMaker* 用于确定如何计算相互作用摩擦角。如果 None，则使用最小值。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

tc(=未初始化)实例 *MatchMaker* 确定接触时间

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用并将 O.timingEnabled==设置为 True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*, *(*字典*)*辐角*2*) →
None：从给定字典更新对象属性

class yade.wrapper.Ip2_WireMat_WireMat_WirePhys(继承 *IPhysFunctor* → 函子 → 可序列化对象)将 2 个 *WireMat* 实例转换为 *WireMat* ，并使用相应的参数。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(*(*可序列化对象*)*辐角*1*) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

linkThresholdIteration(=*1*)用于创建链接的迭代。

timingDeltas
分派器本身内部的详细计时信息。除非在源代码中启用并且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(*可序列化对象*)**arg1*，*(*字典*)**arg2*) → 无:
根据给定字典更新对象属性

IPhysDispatcher

class yade.wrapper.IPhysDispatcher(继承分派器 → 引擎 → 可序列化对象)根据接收到的参数类型调用函子。**dead**(=否)如果为真，此引擎将完全不运行；可用于暂时停用引擎，并在稍后某个时间点重新启用。**dict**（可序列化对象）辐角*1*）→字典：返回属性字典。**dispFunctor**（*IPhysDispatcher*）参数*1*，（材料）参数*2*，（材料）辐角*3*）→ IPhysFunctor：返回针对给定参数进行分派的函子；如果不进行分派则返回 None；分派不明确时抛出异常。**dispMatrix**（*IPhysDispatcher*）辐角*1*[，（布尔值）名称=真]）→ 字典：返回包含分派矩阵内容的字典。

execCount
此引擎运行的累计次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所耗费的累计时间（纳秒），仅在*O.timingEnabled*==True时使用。

functors
与此分派器关联的函子。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

ompThreads(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_数量_线程数，或由 'yade -jN' 定义的 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与会运行并行区域的引擎组合使用时；这样会导致嵌套 OMP 循环的各层使用不同的线程数。

timingDeltas
引擎本身内部计时的详细信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → None：根据给定字典更新对象属性

2.3.10 本构定律

LawFunctor

class yade.wrapper.LawFunctor(继承自 函子 → 可序列化对象)用于将本构定律应用于相互作用的函子。

bases
有序类型列表（以字符串形式）此函子接受。

dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

label(=取消初始化)此对象的文本标签；必须是有效的 Python 标识符，你可以直接从 Python 中引用它。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用计时并将 O.timingEnabled==设置为 True，否则为空。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*) → 无：根据给定字典更新对象属性

class yade.wrapper.Law2_ChCylGeom6D_CohFrictPhys_CohesionMoment(继承 *LawFunctor*→函子 → 可序列化对象)

用于线性压缩的定律，以及无黏聚力的莫尔-库仑塑性面。该定律实现了经典的线弹塑性定律，源自 [CundallStrack1979]（另请参见 [Pfc3dManual30]）。法向力为（按照拉力为正的约定）$F_n = \text{最小值}(k_n u_n, 0)$。剪切力为 $F_s = k_s u_s$，塑性条件定义了剪切力的最大值：$F_s^{max} = F_n$正切(φ)，其中 φ 为摩擦角。

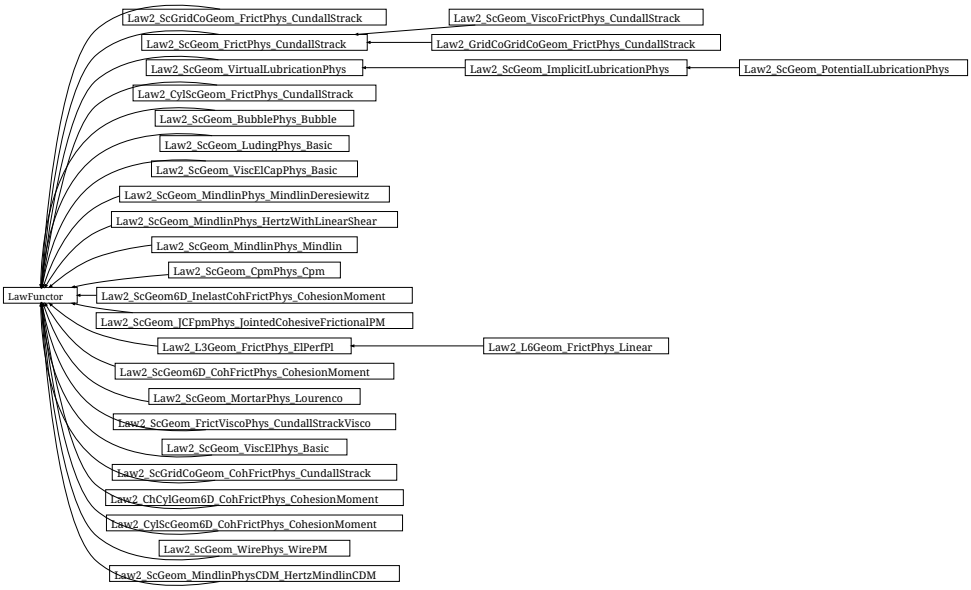


图 *36*：*LawFunctor* 的继承图。另请参阅：*Law2_ChCylGeom6D_CohFrictPhys_CohesionMo- ment*、*Law2_CylScGeom6D_CohFrictPhys_CohesionMoment*、*Law2_CylScGeom_FrictPhys_Cundall- Strack*、*Law2_GridCoGridCoGeom_FrictPhys_CundallStrack*、*Law2_L3Geom_FrictPhys_ElPerfPl*、*Law2_L6Geom_FrictPhys_Linear*、*Law2_ScGeom6D_CohFrictPhys_CohesionMoment*、*Law2_Sc- Geom6D_InelastCohFrictPhys_CohesionMoment*、*Law2_ScGeom_BubblePhys_Bubble*、*Law2_Sc- Geom_CpmPhys_Cpm*、*Law2_ScGeom_FrictPhys_CundallStrack*、*Law2_ScGeom_FrictViscoPhys_- CundallStrackVisco*、*Law2_ScGeom_ImplicitLubricationPhys*、*Law2_ScGeom_JCFpmPhys_Jointed- CohesiveFrictionalPM*、*Law2_ScGeom_LudingPhys_Basic*、*Law2_ScGeom_MindlinPhysCDM_Hertz- MindlinCDM* 、*Law2_ScGeom_MindlinPhys_HertzWithLinearShear*、*Law2_ScGeom_MindlinPhys_- Mindlin*、*Law2_ScGeom_MindlinPhys_MindlinDeresiewitz*、*Law2_ScGeom_MortarPhys_Lourenco*、*Law2_ScGeom_PotentialLubricationPhys*、*Law2_ScGeom_VirtualLubricationPhys*、*Law2_ScGeom_- ViscElCapPhys_Basic*、*Law2_ScGeom_ViscElPhys_Basic*、*Law2_ScGeom_ViscoFrictPhys_Cundall- Strack*、*Law2_ScGeom_WirePhys_WirePM* 、*Law2_ScGridCoGeom_CohFrictPhys_CundallStrack*、*Law2_ScGridCoGeom_FrictPhys_CundallStrack*。

注意：此定律在三轴模拟的上下文中经过了充分测试，并已用于多项已发表的结果（例如，参见 [Scholtes2009b] 以及同一作者的其他论文）。它通过 *定律2_ScGeom6D_CohFrictPhys_CohesionMoment*进行了推广，后者会增加黏聚力和接触力矩。

always_use_moment_law(=*false*)
若为 true，则在全部接触处使用弯曲／扭转力矩。若为 false，则仅对黏聚接触计算力矩。

bases
O此函子接受的类型列表（以字符串形式）。

creep_viscosity(=*1*)蠕变黏度 [帕·秒/米]。可能应该移至 Ip2_CohFrictMat_CohFrictMat_CohFrictPhys...

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

neverErase(=假)
即使颗粒彼此远离，也保留相互作用（仅当场景中存在其他本构定律时，例如 *定律2_ScGeom_CapillaryPhys_毛细作用*）

shear_creep(=否)使用*CohesiveFrictionalContactLaw::creep_黏度*在剪切力上启用蠕变。

timingDeltas
分派器本身内部的详细计时信息。除非在源代码中启用并且 O.timingEnabled==为 True，否则为空。

twist_creep(=否)使用*CohesiveFrictionalContactLaw::creep_黏度*在扭转力矩上启用蠕变。

updateAttrs(*(可序列化对象)arg1*、*(字典)arg2*) →
None：从给定字典更新对象属性

useIncrementalForm(=否)使用增量形式计算弯曲力矩和扭转力矩。在这种情况下，不包含扭转力矩上的蠕变。

class yade.wrapper.Law2_CylScGeom6D_CohFrictPhys_CohesionMoment(继承自 *LawFunctor* → 函子 → 可序列化对象)

此定律对*定律2_CylScGeom_FrictPhys_CundallStrack* 进行了推广，在接触处增加了黏聚力和力矩。

always_use_moment_law(=*false*)若为 true，则在所有接触处使用弯曲/扭转力矩。若为 false，则仅为黏聚接触计算力矩。

bases
此函子接受的类型列表（以字符串表示）。

creep_viscosity(=*1*)蠕变黏度 [Pa.s/m]。或许应该移到 Ip2_CohFrictMat_CohFrictMat_CohFrictPhys...

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

label(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

neverErase(=否)
即使颗粒彼此分离，也保留相互作用（仅当场景中存在其他本构定律时才适用，例如 定律2_ScGeom_CapillaryPhys_毛细作用）

shear_creep(=否)
使用 *CohesiveFrictionalContactLaw::creep_黏度*。

timingDeltas
分派器本身内部的详细计时信息。除非在 e
源代码中启用且 O.timingEnabled==为 True。

twist_creep(=*false*)
对扭转力矩启用蠕变，使用 *CohesiveFrictionalContactLaw::creep_黏度*。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) →
None：根据给定字典更新对象属性

useIncrementalForm(=*false*)
使用增量形式计算弯曲力矩和扭转力矩。在这种情况下，扭转力矩中不包含蠕变。

class yade.wrapper.Law2_CylScGeom_FrictPhys_CundallStrack(继承 *LawFunctor* → 函子 → 可序列化对象)
用于线性压缩和无黏聚力莫尔-库仑塑性面的定律。该定律实现了 [CundallStrack1979] 中的经典线弹塑性定律（另见 [Pfc3dManual30]）。法向力（约定拉力为正）为 $F_n = \text{最小值}(k_n u_n, 0)$。剪切力为 $F_s = k_s u_s$，塑性条件定义了剪切力的最大值：$F_s^{max} = F_n \text{正切}(\varphi)$，其中 φ 为摩擦角。

注意：该定律使用 *ScGeom*。

注意：该定律在三轴模拟中经过了充分测试，并已用于多项已发表的研究成果（例如 [Scholtes2009b] 及同一作者的其他论文）。其推广形式为 *Law2_ScGeom6D_CohFrictPhys_CohesionMoment*，该形式增加了黏聚力和接触力矩。

---

bases
该函子接受的有序类型列表（以字符串形式表示）。

dict(（可序列化对象）辐角*1*) →字
典：返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

neverErase(=否)
即使颗粒彼此分离，也保留相互作用（仅当场景中存在其他本构定律时，例如 定律2_ScGeom_CapillaryPhys_毛细作用）

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用该功能且 O.timingEnabled== 为 True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → 无：
根据给定字典更新对象属性

class yade.wrapper.Law2_GridCoGridCoGeom_FrictPhys_CundallStrack(继承 定律2_Sc-Geom_FrictPhys_-CundallStrack →LawFunctor →函子 → 可序列化对象)

两个 网格连接 之间的摩擦弹性接触定律。参见 定律2_ScGeom_FrictPhys_Cun-dallStrack 了解详情。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict((可序列化对象)参数1) →字典:
返回属性字典。

elasticEnergy((定律2_ScGeom_FrictPhys_CundallStrack)参数1) → 浮点数:
计算并返回所有"FrictPhys"接触中的总弹性势能

initPlasticDissipation((定律2_ScGeom_FrictPhys_CundallStrack)参数1，(浮点数)参数2) →无：将累积塑性耗散初始化为指定值（默认为 0）。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

neverErase(=否)
即使颗粒彼此远离，也保留相互作用（仅当场景中存在其他本构定律时才适用，例如 定律2_ScGeom_CapillaryPhys_毛细作用）

plasticDissipation((定律2_ScGeom_FrictPhys_CundallStrack)辐角1) → 浮点数：所有 FrictPhys 接触中塑性滑移所耗散的总能量。仅当 定律2_-ScGeom_FrictPhys_CundallStrack::traceEnergy 为 true 时计算。

sphericalBodies(=true)
如果为 true，则根据半径计算分支向量（速度更快）；否则使用接触点位置。将此标志设为 true 对球-球接触及其他少数特定情况是安全的，但会导致面片或盒体上的力矩值错误。

timingDeltas
分派器自身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

traceEnergy(=false)
定义所有接触处塑性滑移耗散的总能量。此项仅追踪该定律中的塑性能量；如需更完整地追踪能量，请参见 O.trackEnergy

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
根据给定字典更新对象属性

class yade.wrapper.Law2_L3Geom_FrictPhys_ElPerfPl(继承 LawFunctor → 函子 → Serializable)

用于测试的基本定律 L3Geom；它不具有黏聚力（除非 不破坏 为 True），且塑性滑移遵循莫尔-库仑准则（除非 无滑移 为 True）。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(（可序列化对象）辐角1) →字典：返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

noBreak(=假)
颗粒分离时不要断开接触 e.

noSlip(=否)无塑性滑移。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*) → None：根据给定字典更新对象属性

class yade.wrapper.Law2_L6Geom_FrictPhys_Linear(继承 定律*2_L3Geom_FrictPhys_ElPerfPl* *LawFunctor* → 函子 →可序列化对象)用于测试的基本定律 *L6Geom* – 法向和剪切方向均为线性，不发生滑移或破坏

bases
此函子接受的类型有序列表（以字符串形式表示）。

charLen(=*1*)
特征长度，其含义与弯曲/剪切和扭转/法向刚度比有关。

dict(*(*可序列化对象*)*辐角*1*) →字典：
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

noBreak(=否)
颗粒分离时不要断开接触。

noSlip(=否)无塑性滑移。

timingDeltas
Dispatcher 内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs(*(*可序列化对象*)arg1*，*(*字典*)arg2*) → None：根据给定字典更新对象属性

class yade.wrapper.Law2_ScGeom6D_CohFrictPhys_CohesionMoment(继承 *LawFunctor* →函子 → 可序列化对象)

具有黏聚力的线性拉伸-压缩-弯曲-扭转定律+摩擦和莫尔-库仑塑性面。该定律为 定律*2_ScGeom_FrictPhys_Cundall-Strack*。

法向力为（约定拉力为正） $F_n = \min(k_n * (u_n - u_n^p), a_n)$，其中 a_n 为法向黏附力，u_n^p 为法向位移的塑性部分。剪切力为$F_s = k_s * u_s$，塑性条件定义了剪切力的最大值，默认值为 $F_s^{max} = F_n * \tan(\varphi) + a_s$，其中 φ 为摩擦角，a_s 为剪切黏附力。如果 *CohFrict-Phys::cohesionDisablesFriction* 为 True，则只要黏附处于活动状态，就会忽略摩擦，剪切力的最大值仅为 $F_s^{max} = a_s$。

如果达到最大拉伸力或最大剪切力，且 *CohFrictPhys::fragile* =为 True（默认值），黏聚连接就会断裂，并且 a_n，a_s 会重置为零。如果存在拉伸力，则接触会丢失；否则，剪切强度为 $F_s^{max} = F_n * \tan(\varphi)$。如果 *CohFrictPhys::fragile* =为 False，则行为为理想塑性，剪切强度保持恒定。

如果定律*2_ScGeom6D_CohFrictPhys_CohesionMoment::momentRotationLaw* =为 True，则使用线性定律计算弯曲力矩和扭转力矩，其模量分别为 k_t 和 k_r，因此力矩为：$M_b = k_b * \Theta_b$ 和 $M_t = k_t * \Theta_t$，其中$\Theta_{b,\ t}$ 表示相互作用体之间的相对旋转（详情参见 [Bourrier2013]）。可以定义力矩的最大值，其形式为滚动摩擦。未来也可能加入黏聚型力矩。

该定律实现了接触蠕变，定义见 [Hassan2010]。若启用，则剪切分量和扭转分量会表现出黏性行为，而剪切位移的弹性部分和相对扭转的演变由 $du_{s,\ e}/dt = -F_s/\nu_s$ 和 $d\Theta_{t,\ e}/dt = -M_t/\nu_t$给出。

always_use_moment_law(=假)如果为真，则在所有接触处使用弯曲/扭转力矩。如果为假，则仅为黏聚接触计算力矩。两种情况还都要求 *CohFrictPhys::momentRotationLaw* 为真。

bases
此函子接受的类型有序列表（以字符串表示）。

bendingElastEnergy(*(定律2_ScGeom6D_CohFrictPhys_CohesionMoment)*辐角*1*）→浮点数：计算弯曲弹性势能。

creep_viscosity(=*1*)
蠕变黏度 [帕·秒/米]。可能应该移至 Ip2_CohFrictMat_CohFrictMat_- CohFrictPhys。

dict(*(可序列化对象)*辐角*1*）→字典：返回属性字典。

elasticEnergy(*(Law2_ScGeom6D_CohFrictPhys_黏聚力力矩)*辐角*1*）→浮点数：计算总弹性势能。

initPlasticDissipation(*(Law2_ScGeom6D_CohFrictPhys_黏聚力力矩)*参数*1*,*(浮点数)*辐角*2*）→ None：将累积塑性耗散初始化为指定值（默认为 0）。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

false)
即使颗粒彼此分离，也保留相互作用（仅当场景中存在另一种本构定律时适用，例如 *Law2_ScGeom_CapillaryPhys_毛细作用*）

normElastEnergy(*(Law2_ScGeom6D_CohFrictPhys_黏聚力力矩)*辐角*1*）→浮点数：计算法向弹性能。

plasticDissipation(*(Law2_ScGeom6D_CohFrictPhys_CohesionMoment)*辐角*1*）→浮点数：所有 CohFrictPhys 接触中塑性滑移耗散的总能量。仅在*Law2_-ScGeom_FrictPhys_CundallStrack::traceEnergy* 为 true 时计算。

shearElastEnergy(*(Law2_ScGeom6D_CohFrictPhys_CohesionMoment)*辐角*1*）→ 浮点数：计算剪切弹性能。

shear_creep(=*false*)启用剪切力蠕变，使用*CohesiveFrictionalContactLaw::creep_黏度*。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

traceEnergy(=*false*)
Defi 计算接触处塑性滑移耗散的总能量。请注意，它不会反映任何与脱粘相关的能量（脆性接触中可能发生脱粘），也不包括拉伸中的塑性耗散。

twistElastEnergy(*(定律2_ScGeom6D_CohFrictPhys_黏聚力力矩)*辐角*1*）→ 浮点数：计算扭转弹性能。

twist_creep(=*false*)
使用*CohesiveFrictionalContactLaw::creep_黏度*。对扭转力矩启用蠕变

updateAttrs(*(可序列化对象)*参数*1*，*(字典)*辐角*2*）→ None：根据给定字典更新对象属性

useIncrementalForm(=*false*)
使用增量形式计算弯曲力矩和扭矩。在这种算例中，不考虑扭转力矩的蠕变。

class yade.wrapper.Law2_ScGeom6D_InelastCohFrictPhys_CohesionMoment(继承 定律- 函子 →函子 →可序列化对象)

此定律目前仍在开发中。最终版本和文档将在2014年年底前发布。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict((可序列化对象)辐角*1*) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

normElastEnergy((*Law2_ScGeom6D_InelastCohFrictPhys_CohesionMoment*)*arg 1*) →浮点数：计算法向弹性能。

shearElastEnergy((*Law2_ScGeom6D_InelastCohFrictPhys_CohesionMoment*)*arg 1*) →浮点数：计算剪切弹性能。

timingDeltas
分派器自身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)*arg1*，(字典)*arg2*) → None：从给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_BubblePhys_Bubble(继承 *LawFunctor* → 函子 → 可序列化对象)

气泡模型的本构定律。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict（（可序列化对象）辐角*1*）→字典：返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

pctMaxForce(=*0。1*)
Chan[2011] 指出，该接触定律仅在干涉较小时有效；因此，在超出此范围（穿透较大）时，使用指数型力-位移曲线来模拟接触刚度。这种人为增加刚度的做法可确保气泡在模拟过程中不会彼此穿过或完全重叠。最大力为 Fmax = (2*pi*surfaceTension*rAvg)。pctMaxForce 是最大力的百分比，它决定了每个接触的分离阈值 Dmax。穿透量小于 Dmax 时，根据推导出的接触定律计算反作用力；穿透量等于或大于 Dmax 时，则根据人为设定的指数曲线计算反作用力。

surfaceTension(=*0.07197*)
气泡周围液体的表面张力。默认值为 25 摄氏度时水的表面张力。

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用并且 O.timingEnabled==为 True，否则为空。

updateAttrs（可序列化对象）参数*1*，（字典）辐角*2*）→ None：根据给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_CpmPhys_Cpm(继承 *LawFunctor* → 函子 → 可序列化对象)

用于 *CPM* 模型的本构定律。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(*(*可序列化对象*)*辐角*1*）→字典:
返回属性字典。

elasticEnergy(*(*定律*2_ScGeom_CpmPhys_Cpm)*辐角*1*）→ 浮点数：计算并返回所有“CpmPhys”接触的弹性势能总和

epsSoft(=*1.*，对围压的近似值为：*-3e-3* 时精确达到 *-20*兆帕，*-100*兆帕略高，*-200* 和 *-400* 尚可（割线）)压缩软化开始时的应变（设为非负值即可停用）。默认值设为不会发生塑性变形

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

omegaThreshold(=*1*。， >=*1*。要停用，即永不删除任何接触)接触消失时的损伤值（<1），因为只有当应变达到 →+∞relKnSoft(= 时，omega 才会达到 1。。*3*)压缩软化分支的相对刚度（0=理想弹塑性， <0 软化， >0硬化）

timingDeltas
分派器内部的详细计时信息。除非在 t 中启用，否则为空
源代码和 O.timingEnabled==True。

updateAttrs(*(*可序列化对象*)arg1*，*(*字典*)arg2*）→ None：根据给定字典更新对象属性

yieldEllipseShift(=非数)椭圆的水平缩放（与 +y 发生相互作用时，沿 +x 轴平移）

yieldLogSpeed(=。*1*)对数屈服面中的缩放（若要获得 <1 符合实际的结果，应为；若要获得 >=0 有意义的结果，应为）

yieldSigmaTMagnitude(*(*定律*2_ScGeom_CpmPhys_Cpm)*参数*1*, *(*浮点数*)*法向应力,*(*浮点数*) omega,(*浮点数*)*未损伤黏聚力, *(*浮点数*)*正切摩擦角）→ 浮点数：返回给定材料和状态参数对应的屈服面半径；使用当前实例的属性（如 屈服面类型 等），如有需要，请在调用前更改这些属性。

yieldSurfType(=*2*)
屈服函数：0：莫尔-库仑（原始）；1：抛物线型；2：对数型，3：对数+线性_拉伸，4：椭圆型，5：椭圆型+对数型

class yade.wrapper.Law2_ScGeom_FrictPhys_CundallStrack(继承 *LawFunctor* → 函子→ 可序列化对象)

线性压缩定律，以及无黏聚力的莫尔-库仑塑性面。该定律实现了 [CundallStrack1979] 提出的经典线弹塑性定律（另见 [Pfc3dManual30]）。法向力为（约定拉力为正） $F_n = \text{最小值}(k_n u_n, 0)$。剪切力为 $F_s = k_s u_s$，塑性条件定义了剪切力的最大值：$F_s^{max} = F_n \text{正切}(\varphi)$，其中 φ 为摩擦角。

该定律在三轴模拟中经过了充分测试，并已用于多项已发表的研究成果（例如参见 [Scholtes2009b] 以及同一作者的其他论文）。它由 *Law2_ScGeom6D_CohFrictPhys_CohesionMoment* 推广而来，后者增加了黏聚力和接触力矩。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

elasticEnergy(*(定律2_ScGeom_FrictPhys_CundallStrack)辐角 1*) → 浮点数:
计算并返回所有“FrictPhys”接触的弹性势能总和

initPlasticDissipation(*(定律2_ScGeom_FrictPhys_CundallStrack)参数1*，*(浮点数)辐角 2*) →无：将累积塑性耗散初始化为指定值（默认值为 0）。

已取消初始化）
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

neverErase(=否)
即使颗粒彼此远离，也保留相互作用（仅当场景中存在另一种本构定律时，例如 定律2_ScGeom_CapillaryPhys_毛细作用）

plasticDissipation(*(定律2_ScGeom_FrictPhys_CundallStrack)辐角 1*) → 浮点数：所有 FrictPhys 接触中塑性滑移耗散的总能量。仅当 定律2_-ScGeom_FrictPhys_CundallStrack::traceEnergy 为 true 时才计算。

sphericalBodies(=是)
如果为 true，则根据半径计算分支向量（更快）；否则使用接触点位置。将此标志设为 true 对球-球接触及其他少数特定情形是安全的，但会导致面片或盒体上的力矩值错误。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用，且 O.timingEnabled==为 True，否则为空。

traceEnergy(=否)
定义所有接触处塑性滑移所耗散的总能量。此项仅追踪本定律中的塑性 ic
能量；如需更完整地追踪能量，请参见 O.trackEnergy

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角 2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_FrictViscoPhys_CundallStrackVisco(继承 *LawFunctor* → 函子 → 可序列化对象)
FrictViscoPM 的本构定律。对应于定律2_ScGeom_FrictPhys_CundallStrack唯一的区别是可以考虑法向上的黏性阻尼。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(*(可序列化对象)辐角 1*) →字典:
返回属性字典。

elasticEnergy(*(定律2_ScGeom_FrictViscoPhys_CundallStrackVisco)辐角 1*) → 浮点数：计算并返回所有“FrictViscoPhys”接触中的总弹性势能

initPlasticDissipation(*(定律2_ScGeom_FrictViscoPhys_CundallStrackVisco)arg1*，*(浮点数)arg2*) → None：将累积塑性耗散初始化为某个值（默认为 0）。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。 ly
。

neverErase(=*false*)
即使颗粒彼此远离，也保留相互作用（仅当场景中存在另一种本构定律时，例如 定律2_ScGeom_CapillaryPhys_毛细作用）

plasticDissipation((定律2_ScGeom_FrictViscoPhys_CundallStrackVisco)辐角1) → 浮点数：所有 FrictPhys 接触中塑性滑移耗散的总能量。仅当 :yref:Law2_ScGeom_FrictViscoPhys_CundallStrackVisco::traceEnergy'为 true 时才计算。

sphericalBodies(=真)
: 若为真，则根据半径计算分支向量（速度更快）；否则使用接触点位置。将此标志设为真，对于球-球接触以及其他少数特定情况是安全的。但这会导致面片或盒体上的力矩值错误。

timingDeltas
: 有关分派器本身内部计时的详细信息。除非在源代码中启用此功能且 O.timingEnabled==为真，否则为空。

traceEnergy(=假)
: 定义所有接触处塑性滑移所耗散的总能量。此项仅跟踪该定律中的塑性能；如需更全面地跟踪能量，请参见 O.trackEnergy

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无：
从给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_ImplicitLubricationPhys(继承 定律2_ScGeom_Virtu-alLubricationPhys → LawFunc-tor → 函子 → 可序列化对象)

用于两个球体之间润滑和接触的材料定律，采用隐式方法求解。此接触定律的完整描述见 [Chevremont2020] 。有多种分辨率方法可供选择：迭代精确法，通过求解二阶多项式得出结果；其他求解方法采用数值方法（牛顿-拉夫森法和二分法），并进行变量变换 δ = 对数(u)，在无量纲坐标中求解。

MaxDist(=2.)
: 相互作用的最大距离（d/a）

MaxIter(=30)
: 数值求解的最大迭代次数（二分法和牛顿-拉夫森法）

SolutionTol(=1.e-8)数值求解容差（二分法和牛顿-拉夫森法）

activateRollLubrication(=true)启用滚动润滑
（默认：true）

activateTangencialLubrication(=true)启用切向润滑
（默认：true）

activateTwistLubrication(=true)启用扭转润滑
（默认：true）

bases
: O此函子接受的类型（以字符串表示）的有序列表 。

dict((可序列化对象)辐角1) →字典：
返回属性字典。

static getStressForEachBody() → 元组：获取每个刚体的应力张量：法向接触应力、剪切接触应力、法向润滑应力、剪切润滑应力，以及附加势力产生的应力。

static getTotalStresses() →元组：
: 获取总应力张量：法向接触应力、剪切接触应力、法向润滑应力、剪切润滑应力，以及附加势力产生的应力。

label(=未初始化)
: 此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

maxSubSteps(=*4*)
: theta 方法中自适应时间步进的最大递归深度，因此最小时间间隔为 $Omega::dt/2^{\mathrm{depth}}$。如果仍未收敛，积分器将切换为后向欧拉法。

resolution(=*0*)
: 更改法向分量求解方法，0：使用子步进的迭代精确求解（θ 方法，线性接触），1：牛顿-拉夫森无量纲求解（θ 方法，线性接触），2：（默认）二分无量纲求解（θ 方法，线性接触），3：带接触预测的精确无量纲解（θ 方法，线性接触）。如果体积分数不是太高，方法 3 更好；否则使用方法 2。

theta(=*0.55*)
: “theta”方法的参数，1：后向欧拉法，0.5：梯形法则，0：不使用，0.55：建议的最优值）

timingDeltas
: 分派器内部的详细计时信息。除非在源代码中启用并将 O.timingEnabled==设置为 True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_JCFpmPhys_JointedCohesiveFrictionalPM(继承 定律-函子 → 函子 → 可序列化对象)

适用于黏聚摩擦材料（例如岩石）的相互作用定律；这类材料可能具有节理面，且可通过光滑接触逻辑进行力学描述 [实例变量2011] (在 Yade 中由[Scholtes2012]实现）。脚本示例请参见 examples/jointedCohesiveFrictionalPM。节理面定义（通过 STL 网格或使用 GTS 模块直接定义）也在其中进行了说明。

Key(=*""*)指定保存文件名称的字符串 ‘cracks_____.txt’，当 记录裂纹 为 true 时。

bases
: 此函子接受的类型有序列表（以字符串表示）。

clusterMoments(=*true*)计算聚类矩? （默认开启

computedCentroid(=*false*)计算聚类矩?

cracksFileExist(=*false*)iftrue (andif 记录裂纹), 数据会追加到现有的“裂纹键”文本文件中；否则，其内容将被重置。

dict(*(*可序列化对象*)*辐角*1*) →字典:
返回属性字典。

eventNumber(=*0*)
: 聚类事件编号（用于聚类以及在 ParaView 中可视化各组）。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

momentFudgeFactor(=*1.*)
: Hazzard 和 Damjanac 2013年使用的修正因子，用于提高矩大小精度（默认设为 1，表示不产生影响）

momentRadiusFactor(=*5.*)
: 用于矩震级计算的平均颗粒直径乘数

momentsFileExist(=否)
如果为真（且如果 记录裂纹），则数据将追加到现有的“力矩键”文本文件中；否则，其内容将被重置。

nbShearCracks(=0)
剪切微裂纹的数量。

nbTensCracks(=0)
拉伸微裂纹的数量。

neverErase(=否)
即使颗粒彼此分离，也保留相互作用（仅当场景中存在另一种本构定律时

recordCracks(=否)
如果为 true，则失去黏聚特性的相互作用数据会存储在文本文件 cracksKey.txt 中（参见 键 和 裂纹文件是否存在）。该文件包含 9 列：断裂迭代次数、接触点的 3 个坐标、类型（ 1 表示剪切断裂，0 表示拉伸断裂）、“横截面”（两个球体的平均半径）以及接触法线的 3 个坐标。

recordMoments(=否)
与 :yref: 键<*Law2ScGeom_JCFpmPhys_JointedCohesiveFrictionalPM.Key*>组合，根据簇集断键法计算声发射？（默认关闭）

smoothJoint(=否)
如果为 true，则节理面上的颗粒相互作用（*JCFpmPhys.isOnJoint*）按照光滑接触逻辑处理 [Ivars 2011], [Scholtes2012]。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

totalCracksSurface(=0。)计算总裂纹表面积。

totalShearCracksE(=0。)计算剪切作用下颗粒间微裂纹造成的总耗散能。

totalTensCracksE(=0。)计算拉伸作用下颗粒间微裂纹造成的总耗散能。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → 无:
根据给定字典更新对象属性

useStrainEnergy(=真)使用应变能进行力矩大小估算（如果为假，则使用动能）

class yade.wrapper.Law2_ScGeom_LudingPhys_Basic(继承 *LawFunctor* → 函子 → 可序列化对象)
作用于以下对象的线性黏弹性模型：*ScGeom* 和 *LudingPhys*。参见 [Luding2008] ,[Singh2013]__-了解更多详情。

bases
此函子接受的类型有序列表（以字符串表示）。

dict(*(*可序列化对象*)arg1*) →字典:
返回属性字典。

label(=取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用计时并将 O.timingEnabled==设置为 True，否则为空。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → None：根据给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_MindlinPhysCDM_HertzMindlinCDM(继承*LawFunctor* →函子→可序列化对象)

扩展的赫兹-明德林模型：法向：采用 Harkness 等，2016./ Suhr 和 Six，2017 的锥形损伤模型。切向：采用 Suhr 和 Six，2016 的应力相关的颗粒间摩擦系数。两种模型均可分别开启或关闭。此版本中没有阻尼（无论是黏性阻尼还是线性阻尼）、没有黏附，也不计算_能量；没有 includeMoment，也没有 preventGranularRatcheting。尚未针对周期性模拟进行测试。

bases
此函子接受的类型列表（以字符串形式表示）。

dict(*(可序列化对象)辐角1*) →字典：
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

neverErase(=否)
即使颗粒彼此远离，也要保留相互作用（仅当场景中存在其他本构定律时才适用，例如 定律2_ScGeom_CapillaryPhys_毛细作用）

ratioSlidingContacts(*(定律2_ScGeom_MindlinPhysCDM_HertzMindlinCDM)辐角1*) → 浮点数：返回给定时间滑动接触数量与接触总数的比值

e.

ratioYieldingContacts(*(定律2_ScGeom_MindlinPhysCDM_HertzMindlinCDM)辐角1*) →浮点数：返回给定时间屈服接触数量与接触总数的比值。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用该功能并且 O.timingEnabled==为 True，否则为空。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*) → 无：根据给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_MindlinPhys_HertzWithLinearShear(继承 *LawFunctor*→函子 → 可序列化对象)

赫兹公式的本构定律（使用 *MindlinPhys.kno*）以及剪切方向上的线性行为（使用 *MindlinPhys.kso* 作为刚度，使用 *FrictPhys.tangensOfFrictionAngle*）。

Note： 无黏度或阻尼。如需这些功能，请参阅 定律2_ScGeom_MindlinPhys_Mindl 在此包括非线性 Mindlin 剪切。

---

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict(*(可序列化对象)辐角1*) →字典：
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

neverErase(=否)
即使颗粒彼此远离，也保留相互作用（仅当另一个本构 e
定律位于场景中，例如 定律2_ScGeom_CapillaryPhys_毛细作用）

nonLin(=0)
剪切力非线性（该值决定考虑非线性的多少种特征）。1：ks 与 HM 中相同；2：弹性剪切力增量按 HM 中的方法计算；3：禁用颗粒棘轮效应。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → None：根据给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_MindlinPhys_Mindlin(继承 LawFunctor → 函子 → 可序列化对象)赫兹-明德林公式的本构定律。该公式在法向方向上包含非线性弹性，正如赫兹对两个非共形弹性接触体的预测。在剪切方向上，它则类似于明德林的论文中讨论的无滑移简化情形，其中剪切力与切向位移呈线性关系。最后，采用莫尔-库仑准则确定接触处能够产生的最大摩擦力。此外，还可以通过定义参数 β_n 和 β_s来考虑线性黏性阻尼的影响。

bases
此函子接受的有序类型列表（以字符串形式表示）。

calcEnergy(=否)
用于计算能量项（剪切势能、摩擦耗散能以及法向和切向阻尼导致的能量耗散）的布尔值

contactsAdhesive((定律2_ScGeom_MindlinPhys_Mindlin)arg1) → 浮点数：
计算黏附接触的总数量。

dict((可序列化对象)arg1) →字典：
返回属性字典。

frictionDissipation(=取消初始化)滑动导致
的能量耗散

includeAdhesion(=否)用于根据 DMT公式考虑黏附力的布尔值。若为真，法向弹性能也会考虑黏附效应。

includeMoment(=否)
用于考虑滚动阻力的布尔值（如果 Ip2_FrictMat_FrictMat_MindlinPhys::eta 为 0.0，则不应用塑性条件。）

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接引用它 ly
从 Python 中。

neverErase(=否)
即使颗粒彼此分离，也保留相互作用（仅当场景中存在其他本构定律时，例如 定律2_ScGeom_CapillaryPhys_毛细作用）

normDampDissip(=未初始化)法向阻尼耗散的能量

normElastEnergy((定律2_ScGeom_MindlinPhys_Mindlin)辐角1) →浮点数：计算法向弹性势能。如果 定律2_ScGeom_-MindlinPhys_Mindlin::includeAdhesion 设置为 true，则处理 DMT公式。

preventGranularRatcheting(=是)
用于避免颗粒棘轮效应的布尔值

ratioSlidingContacts((定律2_ScGeom_MindlinPhys_Mindlin)辐角1) →浮点数：返回给定时间内滑动接触数量与接触总数的比值。

shearDampDissip(=已取消初始化)切向阻尼耗散的能量

shearEnergy(=未初始化)剪切弹性势能

timingDeltas
分派器自身内部的详细计时信息。除非在 th 中启用，否则为空 e
源代码和 O.timingEnabled==True。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
从给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_MindlinPhys_MindlinDeresiewitz(继承 *LawFunctor* →函子 →*Serializable*)

如 [Thornton1991] 所述的带部分滑移解的赫兹–明德林接触定律。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

neverErase(=*false*)
即使颗粒彼此分离，也保留相互作用（仅当场景中存在另一种本构定律时，例如 定律2_ScGeom_CapillaryPhys_毛细作用）

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) →
None：根据给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_MortarPhys_Lourenco(继承 *LawFunctor* → 函子 →可序列化对象)

根据 [Lourenco1994]的砂浆层材料定律。接触在达到强度包络线之前表现为弹性；达到强度包络线时发生脆性破坏。强度包络线分为三部分。

满足条件 $\sigma_N - f_t$ 的拉伸。

满足莫尔-库仑条件 $|\sigma_T| + \sigma_N \tan \varphi - c$ 的剪切部分。

满足条件 $\sigma_N^2 + A^2\sigma_T^2 - f_c^2$ 的压缩部分

主要思路是使用此模型开始模拟，并在接触断开后，使用标准的非黏聚 Law2_PolyhedraGeom_PolyhedraPhys_Volumetric。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → None：根据给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_PotentialLubricationPhys(继承 *Law2_ScGeom_ImplicitLubricationPhys* →*Law2ScGeom* *VirtualLubri-__* *__cationPhys* → *LawFunctor* →函子 → 可序列化对象)

用于两个球体之间润滑势的材料定律 + 。势模型包含接触。此材料定律将使用润滑和给定的势求解系统。

MaxDist(=2。)相互作用的最大距离 (d/a)

MaxIter(=30)
数值求解的最大迭代次数（二分法和牛顿-拉夫森法）

SolutionTol(=1.e-8)数值求解容差（二分法和牛顿-拉夫森法）

activateRollLubrication(=true)启用滚动润滑
（默认值：true）

activateTangencialLubrication(=true)启用切向润滑
（默认值：true）

activateTwistLubrication(=true)启用扭转润滑
（默认值：true）

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict((可序列化对象)辐角1) →dict:
返回属性字典。

static getStressForEachBody() → 元组：获取每个刚体的应力张量：法向接触应力、剪切接触应力、法向润滑应力、剪切润滑应力，以及附加势力产生的应力。

static getTotalStresses() →元组：获取总应力张量：法向接触应力、剪切接触应力、法向润滑应力、剪切润滑应力，以及附加势力产生的应力。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

maxSubSteps(=4)
theta 方法中自适应时间步进的最大递归深度，因此最小时间间隔为 $Omega::dt/2^{\mathrm{depth}}$。如果仍未收敛，积分器将切换为后向欧拉法。

potential(=*new GenericPotential()*)球体之间
的物理势力。

resolution(=0)
更改法向分量求解方法：0：带子步进的迭代精确求解（theta 方法，线性接触）；1：牛顿-拉夫森无量纲求解（theta 方法，线性接触）；2：（默认）二分无量纲求解（theta 方法，线性接触）；3：带接触预测的精确无量纲解（theta 方法，线性接触）。当体积分数不太高时，方法 3 效果更好；否则使用方法 2。

theta(=0.55)
'theta'方法的参数，1：后向欧拉法，0.5：梯形法则，0：未使用，0.55：建议的最优值)

timingDeltas
有关分派器内部计时的详细计时信息。除非在源代码中启用并将 O.timingEnabled==设置为 True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.**Law2_ScGeom_VirtualLubricationPhys**(继承 *LawFunctor* → 函子→ 可序列化对象)

用于剪切润滑函数的虚拟类。它不执行任何计算，不应直接使用！

MaxDist(=*2*。)相互作用的最大距离（d/a）

activateRollLubrication(=真)启用滚动润滑
（默认：真）

activateTangencialLubrication(=是)启用切向润滑（默认：启用）

activateTwistLubrication(=是)启用扭转润滑（默认：启用）

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

static getStressForEachBody() → 元组：获取每个刚体的应力张量：法向接触应力、剪切接触应力、法向润滑应力、剪切润滑应力，以及附加势力产生的应力。

static getTotalStresses() →元组：获取总应力张量：法向接触应力、剪切接触应力、法向润滑应力、剪切润滑应力，以及附加势力产生的应力。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，您可以在 Python 中直接引用它。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) →
None：从给定字典更新对象属性

class yade.wrapper.**Law2_ScGeom_ViscElCapPhys_Basic**(继承 *LawFunctor* → 函子 →可序列化对象)

带毛细参数的线性黏弹性模型扩展版本。

NLiqBridg(=未初始化)液桥的总数

VLiqBridg(=未初始化)液桥的总体积

bases
此函子接受的类型有序列表（以字符串表示）。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

timingDeltas
分派器本身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(可序列化对象)*参数*1*，*(字典)*辐角*2*) → 无:
根据给定的字典更新对象属性

class yade.wrapper.Law2_ScGeom_ViscElPhys_Basic(继承 *LawFunctor* → 函子 → *Serializable*)作用于 ScGeom 和 ViscElPhys 的线性黏弹性模型。接触定律在法向上呈黏弹性，在切向上呈黏弹性摩擦特性。法向接触被建模为刚度为 k_n的弹簧，与黏度为 c_n的黏性阻尼器并联。切向接触则由弹簧—阻尼器系统（与刚度为 k_s 、黏度为 c_s的弹簧—阻尼器系统并联）串联一个摩擦系数为μ = 正切值 φ的滑块组成。

摩擦系数 μ =正切值 φ 始终按 tan(最小值(φ_1, φ_2))计算，其中 φ_1和 φ_2分别是颗粒 1 和颗粒 2 的摩擦角。对于其他参数，根据材料输入的不同，接触的等效参数（K_n、C_n、K_s、C_s、φ）的计算方式也不同。下文括号中的物理量是材料常数，针对每个颗粒分别指定。随后将它们与颗粒 1 和 2 相关联（例如 kn_1、kn_2、cn_1.....），不应与接触的等效参数（K_n、C_n、K_s、C_s、φ）混淆。

- 若指定了接触时间 (tc)、法向和切向恢复系数 (en,et)，则会按照 Pournin 的公式计算等效参数 [Pournin2001]。
- 若指定每个颗粒的法向和切向刚度 (kn, ks) 以及阻尼常数 (cn,cs)，则由颗粒 1 和 2 构成的每个接触的等效刚度和阻尼常数为 $A = 2\frac{a_1 a_2}{a_1+a_2}$，其中 A 为 K_n, K_s, C_n 和 C_s，1 和 2 分别指与颗粒 1 和 2 关联的值。
- 也可以指定杨氏模量 (young) 和泊松比 (pois- son)，以替代法向和弹簧常数 (kn 和 ks)。在这种情况下，等效参数的计算方式与前一算例相同，其中 $kn_x = E_x d_x$, $ks_x = \nu_x kn_x$， E_x, ν_x 和 d_x 分别是颗粒 x 的杨氏模量、泊松比和直径。
- 若已指定杨氏模量（杨氏模量）、泊松比（泊松比）以及法向和切向恢复系数（en、et），则等效刚度按前述方法计算： $K_n = 2\frac{kn_1 kn_2}{kn_1+kn_2}$， $kn_x = E_x d_x$， $K_s = \sqrt{2(ks_1 ks_2)}/(ks_1 + ks_2)$， $ks_x = \nu kn_x$。为满足法向恢复系数$e_n = (en_1 + en_2)/2$，需要在每次接触时计算阻尼常数。具体做法是对[Schwager2007]的方程 21 进行数值求解（事实上，该文章从方程 18 到 19 存在错误，导致符号发生变化）。请注意，在此配置下，切向恢复系数设为 1（无切向阻尼）。该公式直接规定碰撞的法向恢复系数，而不是阻尼常数。

bases
此函子接受的类型有序列表（以字符串表示）。

dict(*(可序列化对象)*辐角*1*) →字典:
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接引用它 ly
来自 Python。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用并且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(Serializable)*参数*1*，*(字典)*辐角*2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.Law2_ScGeom_ViscoFrictPhys_CundallStrack(继承定律2_ScGeom_-FrictPhys_CundallStrack→ LawFunctor →函子→ 可序列化对象)L

aw 类似于定律2_ScGeom_FrictPhys_CundallStrack，并在接触处增加剪切蠕变 -
触 ·

bases
此函子接受的有序类型列表（以字符串表示）。

creepStiffness(=1)

dict((可序列化对象)辐角1) →字典:
返回属性字典。

elasticEnergy((定律2_ScGeom_FrictPhys_CundallStrack)辐角1) → 浮点数:
计算并返回所有"FrictPhys"接触中的总弹性势能

initPlasticDissipation(（定律2_ScGeom_FrictPhys_CundallStrack）参数1，（浮点数）辐角2) →无：将累积塑性耗散初始化为指定值（默认为 0）。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

neverErase(=否)
即使颗粒彼此远离，也保留相互作用（仅当场景中存在其他本构定律时，例如 定律2_ScGeom_CapillaryPhys_毛细作用）

plasticDissipation(（定律2_ScGeom_FrictPhys_CundallStrack）辐角1) → 浮点数：所有 FrictPhys 接触中的塑性滑移所耗散的总能量。仅当定律2_-ScGeom_FrictPhys_CundallStrack::能量追踪 为 true 时才计算。

shearCreep(=false)

sphericalBodies(=true)
若为 true，则根据半径计算分支向量（速度更快）；否则使用接触点位置。对于球-球接触和其他少数特定情况，将此标志设为 true 是安全的。但这会导致面片或盒体上的力矩值错误。

timingDeltas
Dispatcher 内部的详细计时信息。除非在源代码中启用并且 O.timingEnabled==True，否则为空。

traceEnergy(=false)D
efine 在接触处塑性滑移所耗散的能量。此项仅跟踪该定律中的塑性能；如需更全面地跟踪能量，请参阅 O.trackEnergy。

updateAttrs((可序列化对象)参数1，(字典)辐角2) →
None：根据给定字典更新对象属性

viscosity(=1)

class yade.wrapper.Law2_ScGeom_WirePhys_WirePM(继承 LawFunctor → 函子 → 可序列化对 i-
象)
线模型的本构定律。

bases
此函子接受的有序类型列表（以字符串表示）。

dict(（可序列化对象）辐角1) →字
典：返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

linkThresholdIteration(=1)
用于创建链接的迭代。

timingDeltas
分派器自身的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → None：根据给定字典更新对象属性

class yade.wrapper.Law2_ScGridCoGeom_CohFrictPhys_CundallStrack(继承 *LawFunctor*→ 函子 → *Serial-izable*)

黏聚摩擦 网格连接 与黏聚摩擦 球体之间的定律。几乎与 定律2_ScGeom6D_CohFrictPhys_CohesionMoment相同，但不计算旋转力矩。

bases
此函子接受的类型有序列表（以字符串形式）。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

neverErase(=假)
即使颗粒彼此分离，也保留相互作用（仅当场景中存在另一种本构定律时才保留，例如 定律2_ScGeom_CapillaryPhys_毛细作用）

timingDeltas
Dispatcher 内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → None：根据给定字典更新对象属性

class yade.wrapper.Law2_ScGridCoGeom_FrictPhys_CundallStrack(继承自 *LawFunctor* → 函子 → 可序列化对象)摩擦网格连接与摩擦球体之间的定律。几乎与定律2_-ScGeom_FrictPhys_CundallStrack相同，但力会被分开并仅施加到两个网格节点上。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

neverErase(=否)
即使颗粒彼此分离，也保留相互作用（仅当场景中还有其他本构定律时适用，例如 定律2_ScGeom_CapillaryPhys_毛细作用）

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用并且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
根据给定字典更新对象属性

LawDispatcher

class yade.wrapper.LawDispatcher(继承自分派器 → 引擎 → 可序列化对象)分派器调用函子根据接收到的参数类型进行分派。

dead(=*false*)
:)如果为 true，此引擎将完全不运行；可用于暂时停用引擎，并仅在之后的某个时间点重新启用它。

dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

dispFunctor(*(LawDispatcher)*参数*1*，*(IGeom)*参数*2*，*(IPhys)*辐角*3*) → LawFunctor：返回针对给定参数进行分派的函子；如果没有匹配的分派，则返回 None；如果分派不明确，则抛出异常。

dispMatrix(*(LawDispatcher)*辐角*1*[, *(bool)*名称=真]) → 字典：返回包含分派矩阵内容的字典。

execCount
: 此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
: 此引擎运行所用时间的累计值，单位为纳秒（仅在*O.timingEnabled*==True时使用）。

functors
: 与此分派器关联的函子。

label(=未初始化)
: 此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

ompThreads(=*-1*)
: 引擎中使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与*ParallelEngine* 组合使用包含并行区域的引擎，从而在不同层级产生线程数不同的嵌套 OMP 循环。

timingDeltas
: 引擎本身内部计时的详细信息。除非在源代码中启用且 *O.timingEnabled*==True，否则为空。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*) → None：根据给定字典更新对象属性

2.3.11 内力

内力函子

内力分派器

2.3.12 回调

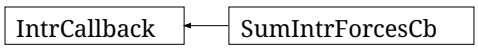


图 37：IntrCallback 的继承图。另请参阅：*SumIntrForcesCb*。

class yade.wrapper.IntrCallback(继承可序列化对象)
抽象回调对象，每个（真实）相互作用在经过以下对象处理后都会调用*InteractionLoop*。

在相互作用循环开始时， stepInit 会被调用以初始化对象；它返回以下任一项：NULL （用于在此时间步停用回调）或指向函数的指针；随后将向该函数传递：(1) 指向回调对象本身的指针，以及 (2) 指向相互作用的指针。

注意： （尚未完成）此功能可通过向 *InteractionLoop* 构造函数传递第 4 个参数，或将回调对象追加到 *InteractionLoop::callbacks* 中，从 Python 访问。

dict(*(Serializable)*辐角 *1*) →字典:
返回属性字典。

updateAttrs(*(Serializable)*参数*1*，*(*字典*)*辐角 *2*) → 无:
根据给定字典更新对象属性

class yade.wrapper.SumIntrForcesCb(继承自 *IntrCallback* → *Serializable*)对全部相互作用中的力大小求和的回调。 相互作用的 *IPhys* 必须派生自 *NormShearPhys* （由用户负责）。

dict(*(Serializable)*辐角 *1*) →字典:
返回属性字典。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) → 无:
根据给定字典更新对象属性

2.3.13 预处理器

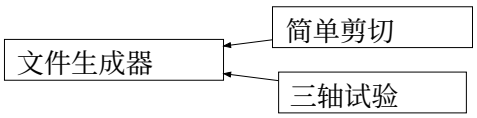


图 38：文件生成器的继承图。另请参阅：简单剪切、三轴试验。

class yade.wrapper.FileGenerator(继承可序列化对象)场景生成器和预处理器的基类。dict(*(*可序列化对象*)*辐角 *1*) →字典：返回属性字典。generate(*(*文件生成器*)*参数*1*，*(*字符串*)*输出) → 无：生成场景并保存到指定文件load(*(*文件生成器*)*辐角 *1*) → 无：生成场景，保存到临时文件并立即加载updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) → 无：根据给定字典更新对象属性

class yade.wrapper.SimpleShear(继承 文件生成器 → 可序列化对象)
简单剪切盒模型的预处理器。堆积体最初呈气体状、非常疏松的状态（参见 utils.makeCloud 函数），也可以稍作修改源代码，从文本文件导入已有的堆积体。当前版本的预处理器会执行固结压缩，直至法向应力达到 2 兆帕（且力学状态稳定）。还可以使用其他引擎，例如 *KinemCNDEngine*、*KinemCNSEngine* 和*KinemCNLEngine*，通过此类简单剪切盒分别施加恒定法向位移、恒定法向刚度和恒定法向应力路径。

density(=*2600*)球体密度 [kg/m^3]**dict**(*(*可序列化对象*)*辐角 *1*) →字典：返回属性字典。**generate**(*(*文件生成器*)*参数*1*，*(*字符串*)out*) → 无：生成场景并保存到指定文件**gravApplied**(=否)根据此项，*GravityEngine* 会被添加到场景中或不添加，以计入颗粒的重量**gravity**(=*Vector3r(0，-9.81，0)*)对应于所用重力的向量（如果*gravApplied*） [m/s^2]

height(=*0.02*)剪切盒沿 y 轴的初始高度 [m]

length(=*0.1*)剪切盒沿 x 轴的初始长度 [m]

load(*(*文件生成器*)*辐角 *1*) → None：生成场景，保存到临时文件并立即加载

matFrictionDeg(=*37*)堆积体内以及两个水平边界的 FrictMat.frictionAngle 值（其他边界上的摩擦为零） [°] （所需的转换会在 [rad] 中自动完成）**matPoissonRatio**(=*0。04*)刚体的 FrictMat.poisson 值 [-]**matYoungModulus**(=*4.0e9*)刚体的 FrictMat.young 值 [Pa]**thickness**(=*0。001*)构成剪切盒的盒体厚度 [m]**timeStepUpdateInterval**(=*50*)此处所用 TimeStepper 的 TimeStepper::timeStepUpdateInterval 值**updateAttrs**((可序列化对象)参数1，(字典)辐角 *2*) → None：根据给定字典更新对象属性

width(=*0。04*)剪切盒的初始宽度（沿 z轴） [m]

class yade.wrapper.**TriaxialTest**(继承 文件生成器 → 可序列化对象)创建三轴试验场景。

简介 Yade 包含用于模拟颗粒集合体三轴试验的工具。此预处理器（及其变体，例如 *CapillaryTriaxialTest*）展示了如何使用这些工具。它会生成一个场景，默认情况下，该场景将依次经过以下步骤：

- 在平行六面体中生成随机疏松堆积体。
- 对堆积体进行各向同性压缩：根据标志 *internalCom-paction*，可以将堆积体夹在移动的刚性盒之间进行挤压，也可以在盒体固定时扩张颗粒。此阶段的围压由 *sigmaIsoCompaction* 定义。
- 当堆积体变得密实且稳定后，模拟加载路径并由此获得力学响应。

默认加载路径对应于两个方向上的恒定侧向应力（侧向围压）以及第三个方向上的恒定应变率。当标志 自动压缩启用 为 True时，将执行此默认加载路径；否则，模拟会在各向同性压缩后停止。

可以执行不同的加载路径。要定义这些路径，用户可以在模拟的任意时刻修改引擎 三轴应力控制器 中的标志（见 c++）。如果 `TriaxialStressController.wall_X_activated` 为 `true`，边界 X 将自动移动，以维持所定义的应力水平 *sigmaN*（见下文的坐标轴约定）。如果 `false`，引擎将完全不控制该边界。在这种情况下，用户可以自行指定固定位置、恒定速度或更复杂的条件。

注意： 坐标轴约定。垂直于x轴的边界称为“左侧”和“右侧”，y对应“顶部”和“底部”，而z轴对应“前侧”和“后侧”。在默认加载路径中，应变率沿y方向施加，并在x轴和z轴上施加恒定应力。

基本引擎

1. 三轴压缩引擎 用于控制试样状态并模拟加载路径。三轴压缩引擎 继承自 三轴应力控制器，该控制器计算堆积体中的应力和类应变量，并在每个边界处维持恒定的应力水平。三轴压缩引擎 还包含一些成员，用于施加恒定应变率，并控制各向同性压缩与三轴试验之间的转换。转换通过更改 *Triaxial-StressController*中的一些标志来定义，即在施加应变率与施加应力之间进行切换。

2. 类 *TriaxialStateRecorder* 用于将应力应变历史写入文件。

3. *TriaxialTest* 使用 *GlobalStiffnessTimeStepper* 计算适用于 Δt 数值格式的参数。

注意： `TriaxialStressController::ComputeUnbalancedForce` 返回的值可用于评估堆积体的稳定性。其定义为（作用于颗粒的平均力）/（平均接触力），因此在稳定堆积中趋近于0。*TriaxialCompressionEngine* 会检查此参数，以便在模拟的不同阶段之间切换（例如，停止各向同性围压并开始轴向加载）

常见问题

1. **如何生成堆积体？如何更改颗粒尺寸分布？为什么我会看到 m**

   球体的初始定位通过在盒子中随机生成 (x,y,z)，并检查能否在该位置插入一个半径为 R 的球体（R 也按照均匀分布随机生成，范围为 mean*(1-std_dev) 到 mean*(1+std_dev)），且不与其他球体重叠。

   如果球体发生重叠，就会重新生成 (x,y,z)，直到为新球体找到一个空闲位置。这解释了你看到的消息：经过 3000 次反复试探后，仍无法放置该球体，于是算法停止。

   如果尝试生成初始密堆积体，就会出现上述消息，因为该算法无法实现这一点。它只能生成云团。应保留孔隙率的默认值（n~0.7），如果某些情况下仍然过低，也可以调高。密实状态将在第二步获得（压实，见下文）。

2. **如何进行压　　实，参数 最大墙体速度 和 *finalMaxMultiplie***

   压实已完成

   1. 移动刚性盒，或 2. 增大颗粒的尺寸（通过选项 *internalCompaction*　size increase 决定）。

两种算法都需要数值参数来防止不稳定性。例如，方法（1）中，*maxWallVelocity* 是最大墙体速度；方法（2）中，*final-MaxMultiplier* 是每次迭代时应用于尺寸的乘数的最大值（通常接近 1.00001）。

3. **在三轴压缩试验的模拟过程中，一个方向上的墙以应变增量移动，同时其他两个方向上的应力被调整为** **an incr**
   边界上的应力控制基于堆积体与该边界之间所有接触的总刚度 K 。简而言之，在每一步中，位移=应力_- 误差/K。该算法在 三轴应力控制器中实现，而控制逻辑本身位于 `TriaxialStressController::ControlExternalStress`中。可以使用标志 `wall_XXX_activated`，分别关闭各个边界的控制，其中 XXX {顶部、底部、左侧、右侧、后侧、前侧}。如果 三轴应力控制器.*isAxisymetric*，则所有方向施加的应力均为同一个值（*sigma_iso*）；否则为三个独立值*sigma1*、*sigma2*、*sigma3*。

4. **在三轴阶段的压实过程中，您使用的摩擦角值是多少** **est?**
   压实过程中的摩擦角（试样生成时无论使用膨胀法还是压缩法）可以取 0 到三轴阶段最终使用值之间的任意值。请注意，摩擦角高于最终值会导致试验开始时发生体积塌缩。在这一阶段使用不同摩擦角的目的，是因为试样生成结束时得到的最终孔隙率不仅取决于假设的粒径分布，也取决于摩擦角。改变摩擦角的初始值会得到不同的最终孔隙率。

5. **的目标是什么** `bool isRadiusControlIteration`**?** 这个内部变量（自动更新）每隔 N 个时间步（即 N=*radiusControlInterval*）为真。其他时间步不会发生膨胀。不进行膨胀的循环只是一种加快模拟速度的方法，其依据是：每 10 次迭代增加 1%，比每次迭代增加 0.1% 所需的操作更少，但两者会得到相似的结果。

6. **为什么只有经过许多时间步后，压实**
   不平衡力的值（无量纲）预计只有在压实阶段结束时才会降到较低值（即表明试样处于静力平衡状态）。该代码并非旨在模拟准静态各向同性压实过程，而只是在压实结束时生成稳定堆积体。

`Key`(=””)用于添加到输出文件名中的代码。

`StabilityCriterion`(=*0.01*)
系统被视为稳定时的不平衡力值。用于条件语句中 o
在加载阶段之间切换。

`WallStressRecordFile`(=*”./WallStresses”*+键)

`autoCompressionActivation`(=*true*)
我们是只想在各向同性压力下生成稳定堆积体（false），还是希望在压实阶段结束后立即自动开始三轴加载（true）？

`autoStopSimulation`(=*false*)
达到条件时冻结模拟（如果希望能够通过 Qt图形用户界面运行/停止，请勿启用此选项）

`autoUnload`(=*true*)
将各向同性应力状态从 *TriaxialTest::sigmaIsoCompaction* 调整为 *Triaxial-Test::sigmaLateralConfinement* （如果二者的值不同）。详见 *TriaxialCompres-sionEngine::autoUnload*

`boxFrictionDeg`(=*0.0*)
边界接触的摩擦角 [°] 。

`boxKsDivKn`(=*0。5*)盒体剪切接触刚度与法向接触刚度之比。`boxYoungModulus`(=*15000000。0*)盒体刚度。`compactionFrictionDeg`(=sphereFrictionDeg)压实过程中球体的摩擦角 [°]（不同的值会产生不同的孔隙率）]。三轴试验前，TriaxialTest::sphereFrictionDeg 会覆盖此值。`dampingForce`(=*0。2*)Cundall-非黏性阻尼系数（施加于力的 3 个分量）`dampingMomentum`(=*0。2*)Cundall-非黏性阻尼系数（施加于力矩的 3 个分量）`defaultDt`(=*-1*)最大时间步长。若已定义，则用作初始值，随后由时间步进器进行调整。`density`(=*2600*)球体密度`dict`(（可序列化对象）辐角*1*) →字典：返回属性字典。`facetWalls`(=否)使用面片作为边界（未测试）`finalMaxMultiplier`(=*1。001*)内部压实过程中直径的最大乘数（二次精确调整）`fixedBoxDims`(=””)包含 {‘x’,’y’,’z’} 的某个子集（最多 2 个）的字符串；包含这些坐标轴时，盒体的尺寸将被硬编码，即使盒体按平均_半径规定进行了缩放：缩放将应用于其余部分。`generate`(（文件生成器）参数1，（字符串）out) → 无：生成场景并保存到指定文件`importFilename`(=””)包含球体位置和尺寸的文件。`internalCompaction`(=否)用于在压实阶段选择移动边界或增大颗粒尺寸的标志。`load`(（文件生成器）辐角*1*) → 无：生成场景，保存到临时文件并立即加载`lowerCorner`(=Vector3r(0, 0, 0))盒子的下角点。`maxMultiplier`(=*1。01*)内部压实过程中直径的最大乘数（初始快速增大）`maxWallVelocity`(=*10*)边界的最大速度。通常没有用处，但在某些情况下有助于稳定系统。`noFiles`(=否)运行期间不创建任何文件（.xml、.spheres、墙面应力记录）`numberOfGrains`(=*400*)生成球体的数量。`radiusControlInterval`(=*10*)球体增大时尺寸变化之间的区间。

radiusMean(=*-1*)平均半径。如果为负值（默认值），则根据盒子尺寸和 *Triaxial-Test::numberOfGrains*__radiusStdDev__(=*0*自动计算。*3*)生成尺寸的归一化标准差。__recordIntervalIter__(=*20*)文件输出间隔__seed__(=*0*)调用 makeCloud 时使用的随机种子 __sigmaIsoCompaction__(=*-50000*)各向同性压实时的围压（< 0 对于实数值，即压缩压实）。

__sigmaLateralConfinement__(=*-50000*)三轴加载期间的侧向应力（< 0 适用于经典压缩算例）。如果该值不等于 *TriaxialTest::sigmaIsoCompaction*，则执行各向同性卸载。

__sphereFrictionDeg__(=*18。0*)三轴试验前分配给球体的摩擦角 [°] 。

__sphereKsDivKn__(=*0。5*)球体的剪切接触刚度与法向接触刚度之比。

__sphereYoungModulus__(=*15000000。0*)球体的刚度。__strainRate__(=*0。1*)三轴加载时的应变率。__thickness__(=*0。001*)边界的厚度。该值任意设定，且不应产生影响 __timeStepUpdateInterval__(=*50*)用于 *GlobalStiffnessTimeStepper*__updateAttrs__(*(可序列化对象) arg1*, *(字典) arg2*) → None：根据给定字典更新对象属性__upperCorner__(=*Vector3r(1, 1, 1)*)盒子的上角点。__wallOversizeFactor__(=*1。3*)将边界设得比堆积体更大，以确保球体在变形过程中不会跑出边界。__wallStiffnessUpdateInterval__(=*10*)更新样品与边界接触刚度的区间

__wallWalls__(=*false*)使用壁面作为边界（未测试）

2.3.14 渲染

OpenGL 渲染器

__class__ yade.wrapper.__OpenGLRenderer__(继承可序列化对象)负责在 OpenGL 设备上渲染场景的类。__bgColor__(=*Vector3r(.2,.2, .2)*)背景画布的颜色（RGB） __blinkHighlight__(=*BlinkHighlight::NORMAL*)调整在“模拟检查”窗口中选中的刚体的闪烁效果。

`bound`(=否)渲染物体 Bound`cellColor`(=Vector3r(1, 1, 0))周期单元的颜色（RGB）。`clipPlaneActive`(=向量<布尔值>(numClipPlanes, false))启用/停用相应的裁剪平面`clipPlaneSe3`(=向量<Se3r>(numClipPlanes, Se3r(Vector3r::Zero(), Quaternionr::Identity()))裁剪平面的位置和方向`dict`((可序列化对象)辐角 *1*）→字典：返回属性字典。`dispScale`(=Vector3r::Ones()，禁用缩放)将物体参考位置的位移按此相对量人为放大（缩放），使其更易于观察（在三个维度上分别缩放）。当值为 (1,1,1) 时禁用。`dof`(=否)显示每个刚体被约束的自由度`extraDrawers`(=未初始化)附加渲染组件 (GlExtraDrawer)。`ghosts`(=是)通过在多个位置克隆对象，渲染跨越周期单元边缘的对象（仅适用于周期性模拟）。`hideBody`((OpenGLRenderer)参数1, (int)标识符）→ 无：隐藏具有指定标识符的刚体（参见 OpenGLRenderer::showBody）`id`(=否)显示刚体标识符`intrAllWire`(=否)为所有相互作用绘制线缆，势相互作用显示为蓝色，真实相互作用显示为绿色（主要用于调试）`intrGeom`(=否)渲染 Interaction::geom 对象。`intrPhys`(=否)渲染 Interaction::phys 对象`intrWire`(=否)渲染相互作用时，仅使用线框表示。`light1`(=是)打开光源 1。`light2`(=是)打开光源 2。`light2Color`(=Vector3r(0.5, 0.5, 0.1))次级光源各颜色的强度（RGB）。`light2Pos`(=Vector3r(-130, 75, 30))场景中次级 OpenGL 光源的位置。`lightColor`(=Vector3r(0.6, 0.6, 0.6))主光源各颜色的强度（RGB）。`lightPos`(=Vector3r(75, 130, 0))场景中 OpenGL 光源的位置。`mask`(=*~0*,绘制全部内容)位掩码，仅显示满足 ((mask & Body::mask)! 的刚体=0

render((OpenGLRenderer)参数*1*) → 无：在当前 OpenGL 上下文中渲染场景。**rotScale**(=*1*。，禁用缩放)人为放大（缩放）刚体相对于其参考方向的旋转，使其更容易看清。**selId**(=Body::ID_无)用户选中的颗粒标识符。**setRefSe3**((OpenGLRenderer)参数*1*) → 无：将当前位置和方向设为位移缩放和旋转缩放的参考值。**shape**(=是)渲染刚体形状**showBody**((OpenGLRenderer)参数1, (int)标识符) → 无：使刚体可见（参见OpenGLRenderer::hideBody）**updateAttrs**((Serializable)参数1, (dict)参数*2*) → 无：根据给定字典更新对象属性**wire**(=否)仅以线框方式渲染所有刚体（速度更快）

GlShapeFunctor

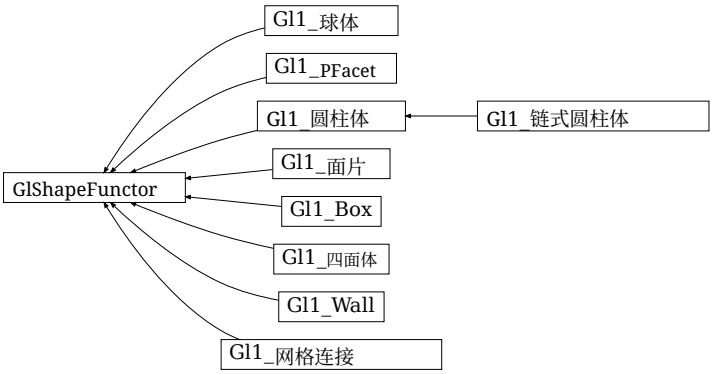


图 39：GlShapeFunctor 的继承图。另请参见：*Gl1_盒子*、*Gl1_链式圆柱体*、*Gl1_圆柱体*、*Gl1_面片*、*Gl1_网格连接*、*Gl1_PFacet*、*Gl1_球体*、*Gl1_四面体*、*Gl1_墙*。

class yade.wrapper.GlShapeFunctor(继承函子 → 可序列化对象)用于渲染的抽象函子形状对象。**bases**此函子接受的有序类型列表（以字符串形式表示）。**dict**((可序列化对象)辐角*1*) →字典：返回属性字典。**label**(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器内部计时的详细信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs((可序列化对象)参数*1*，(字典)辐角*2*) → 无：
从给定字典更新对象属性

class yade.wrapper.Gl1_Box(继承 *GlShapeFunctor* → 函子 → 可序列化对象)渲染盒子 对象

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict(（可序列化对象） 参数*1*) →字典：返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器内部计时的详细信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(（可序列化对象） 参数*1*，（字典） 参数*2*) → None：根据给定字典更新对象属性

class yade.wrapper.Gl1_ChainedCylinder(继承 *Gl1_圆柱体* → *GlShapeFunctor* → 函- 子→ 可序列化对象)渲染 链式圆柱体 对象，其中包含用于补偿弯曲的平移。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(*(*可序列化对象*)*辐角*1*) →字典：返回属性字典。

glutNormalize = True

glutSlices = 8

glutStacks = 4

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
Dispatcher 内部的详细计时信息。除非在源代码中启用该功能且 O.timingEnabled==为 True，否则为空。

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → 无：根据给定字典更新对象属性

wire = False

class yade.wrapper.Gl1_Cylinder(继承*GlShapeFunctor* → 函子 → 可序列化对象)渲染圆柱体对象

wire(=否) [static]仅显示线框（由 glutSlices 和 glutStacks控制）。

glutNormalize(=是) [static]修正非线框渲染的法线

glutSlices(=*8*) [static]球体切片数。

glutStacks(=*4*) [static]球体堆叠数

S.bases此函子接受的类型有序列表（以字符串形式表示）。dict(（可序列化对象） 辐角*1*) →字典：返回属性字典。

glutNormalize = True

glutSlices = 8

glutStacks = 4

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可在 Python 中直接引用。

timingDeltas
分派器本身内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
根据给定字典更新对象属性

wire = False

class yade.wrapper.Gl1_Facet(继承 *GlShapeFunctor* → 函子 → 可序列化对象)渲染面片 对象

normals(=假) [static]在线框模式下，渲染面片和边的法线；此时忽略面片的 颜色 。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict((可序列化对象)辐角1) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

normals = False

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
使用给定字典更新对象属性

class yade.wrapper.Gl1_GridConnection(继承 *GlShapeFunctor* → 函子 → 可序列化对象)渲染圆柱体 对象

wire(=否) [static]仅显示线框（由 glutSlices 和 glutStacks控制。

glutNormalize(=是) [static]修正非线
框渲染的法线
glutSlices(=8) [static]圆柱
切片数。
glutStacks(=4) [static]圆柱层的数量。bases此函子
接受的类型（字符串形式）有序列表。dict((可序列化对
象）辐角1) →字典：返回属性字典。
glutNormalize = TrueglutSlices = 8glutStacks = 4

label(=取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
Dispatcher 内部的详细计时信息。除非在源代码中启用并且 O.timingEnabled==True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无:
根据给定字典更新对象属性

wire = False

class yade.wrapper.Gl1_PFacet(继承 *GlShapeFunctor* → 函子 → 可序列化对象)渲染面片 对象

wire(=否) [static]仅显示线框（由 glutSlices 和 glutStacks 控制。

bases
此函子接受的有序类型列表（以字符串表示）。

dict(（可序列化对象）辐角1) →字
典：返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(（可序列化对象）参数1，（字典）辐角
2) → None：根据给定字典更新对象属性

wire = False

class yade.wrapper.Gl1_Sphere(继承自 *GlShapeFunctor* → 函子 → 可序列化对象)渲染球体 对象

quality(=*1.0*) [static]
更改球体的离散化级别。质量>1 更高可提升图像质量，但会增加 CPU/GPU 使用量， 0<质量<1 更低则可加快渲染速度。如果显示的是单色球体（*Gl1_-Sphere::stripes* = 为 False），质量会乘以 *Gl1_Sphere::glutSlices* 和 *Gl1_Sphere::glutStacks*。如果显示的是条纹球体（*Gl1_Sphere::stripes* = 为 True），只有整数增量才有意义：质量=1 和质量=10.9 的效果相同，质量=2则会得到更精细的效果。

wire(=*false*) [static]
仅显示线框（由 glutSlices 和 glutStacks控制）。

stripes(=*false*) [static]
在非线框渲染中，显示清晰表明颗粒旋转的条纹。

localSpecView(=是) [static]在局部视点坐标系中计算镜面
光。

glutSlices(=*12*) [static]
球体切片的底数，使用前乘以 *Gl1_Sphere::quality* ）；不与stripes (参见 glut{Solid,Wire}Spherereference)

glutStacks(=*6*) [static]
球体堆叠的底数，乘以 *Gl1_Sphere::quality* 使用前；不与stripes （参见 glut{Solid,Wire}Spherereference)

circleView(=否) [static]
对于二维模拟：显示环面而非球体，这样当查看器朝向正确方向时，环面就会呈现为圆形。在这种情况下，请记得在查看器中按下“t”键以禁用透视。

circleRelThickness(=*0.2*) [static]
如果 *Gl1_Sphere::circleView* 已启用，则此值表示环面直径相对于球体半径的比例（即圆形的相对厚度）。

circleAllowedRotationAxis(='z') [static]
如果 *Gl1_Sphere::circleView* 已启用，则这是二维模拟中允许旋转的唯一轴（‘x’、‘y’或‘z’）。它可确保环面以正确的方向呈现为圆形。举例来说，如果 circleAllowedRotationAxis='x' 已设置，则所有颗粒的受阻自由度 blockedDOFs="YZ"也应设置。

bases
此函子接受的有序类型列表（以字符串形式表示）。

circleAllowedRotationAxis = 'z'

circleRelThickness = 0.2

circleView = False

dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

glutSlices = 12

glutStacks = 6

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

localSpecView = True

quality = 1.0

stripes = False

timingDeltas
Dispatcher 内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*) → 无：根据给定字典更新对象属性

wire = False

class yade.wrapper.Gl1_Tetra(继承 *GlShapeFunctor* → 函子 → 可序列化对象)渲染四面体 对象

wire(=是) [static]待办

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用并且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无：
根据给定字典更新对象属性

wire = True

class yade.wrapper.Gl1_Wall(继承 *GlShapeFunctor* → 函子 → 可序列化对象)渲染墙对象

div(=20) [static]可见场景部分中墙的分区数量。

bases此函子接受的类型（以字符串表示）的有序列表。
dict(（可序列化对象） 辐角1) →字典：返回属性字典。

div = 20

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) →
None：根据给定字典更新对象属性

GlStateFunctor

class yade.wrapper.GlStateFunctor(继承 函子 → 可序列化对象)用于渲染的抽象函子 状态 对象。

bases
此函子接受的类型有序列表（以字符串形式表示）。

dict((可序列化对象)辐角1) →字典：
返回属性字典。

label(=取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，您可以直接引用它来自 Python。

timingDeltas
分派器自身内部的详细计时信息。除非在源代码中启用并将 O.timingEnabled==设置为 True，否则为空。

updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无：
从给定字典更新对象属性

GlBoundFunctor

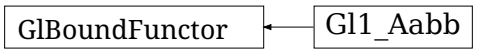


图 40：GlBoundFunctor 的继承图。另请参见：*Gl1_Aabb*。

class yade.wrapper.GlBoundFunctor(继承自函子 → 可序列化对象)用于渲染的抽象函子边界对象。

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。

label(=已取消初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==True，否则为空。

updateAttrs((可序列化对象)参数*1*，(字典)辐角 *2*) →
None：根据给定字典更新对象属性

class yade.wrapper.Gl1_Aabb(继承 *GlBoundFunctor* → 函子 → 可序列化对象)渲染轴对齐包围盒（*Aabb*）。

bases
此函子接受的类型有序列表（以字符串表示）。

dict((可序列化对象)辐角*1*) →字典:
返回属性字典。

label(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。
。

timingDeltas
Dispatcher 内部的详细计时信息。除非在源代码中启用并且 O.timingEnabled==True，否则为空。

updateAttrs((可序列化对象)参数*1*，(字典)辐角 *2*) →
None：从给定字典更新对象属性

GlIGeomFunctor

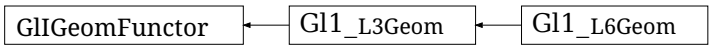


图 41：GlIGeomFunctor 的继承图。另请参阅：*Gl1_L3Geom*，*Gl1_L6Geom*。

class yade.wrapper.GlIGeomFunctor(继承 函子 → 可序列化对象)用于渲染 *IGeom* 对象的抽象函子。

bases
此函子接受的类型有序列表（以字符串表示）。

dict((可序列化对象)辐角 *1*) →字典:
返回属性字典。label(=未初始化)

此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。

updateAttrs((可序列化对象)参数*1*，(字典)辐角 *2*) →
None：根据给定字典更新对象属性

class yade.wrapper.Gl1_L3Geom(继承 *GlIGeomFunctor* → 函子 → 可序列化对象)渲染 *L3Geom* 几何体。

axesLabels(=否) [static]是否显示局部坐标轴（
x、y、z）的标签

axesScale(=*1*。) [static]缩放局部坐标轴，其参考长度为最小半径的一半。

axesWd(=*1*。) [static]坐标轴线的宽度，单位为像素；若为
非正值，则不绘制

uPhiWd(=*2*。) [static]
用于绘制位移（以及旋转）的线宽，适用于*L6Geom*）；若为非
正值。

uScale(=*1.*) [static]
缩放局部位移（*u* - *u0*）；1 表示实际缩放比例，0 表示禁用局部位移的绘制；也允许使用负值。

axesLabels = False

axesScale = 1.0

axesWd = 1.0

bases
此函子接受的有序类型列表（以字符串形式表示）。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器本身内部的详细计时信息。除非在源代码中启用，且 O.timingEnabled==为 True，否则为空。

uPhiWd = 2.0

uScale = 1.0

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角 *2*) →
None：根据给定字典更新对象属性

class yade.wrapper.Gl1_L6Geom(继承 *Gl1_L3Geom* → *GlIGeomFunctor* → 函子 → 可序列化对象)

渲染*L6Geom* 几何体。

phiScale(=*1*。) [static]缩放局部旋转（*phi* - *phi0*）。默认缩放方式是将 π 旋转绘制为长度等于最小半径。

axesLabels = False

axesScale = 1.0

axesWd = 1.0

bases
此函子接受的类型有序列表（以字符串表示）。

dict(*(*可序列化对象*)*辐角 *1*) →字典:
返回属性字典。

label(=取消初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

phiScale = 1.0

timingDeltas
分派器自身的详细计时信息。除非在源代码中启用该功能并将 O.timingEnabled==设为 True，否则为空。

uPhiWd = 2.0uScale = 1.0updateAttrs((可序列化对象)参数1，(字典)辐角2) → 无：根据给定字典更新对象属性

GlIPhysFunctor

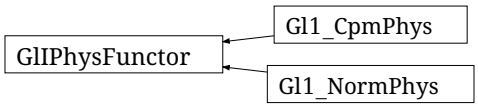


图 42：GlIPhysFunctor 的继承图。另请参阅：*Gl1_CpmPhys*，*Gl1_NormPhys* .

class yade.wrapper.GlIPhysFunctor(继承自 函子 → 可序列化对象)用于渲染 *IPhys* 对象的抽象函子。

bases此函子接受的类型有序列表（以字符串形式表示）。dict((可序列化对象)辐角1) →字典：返回属性字典。label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

timingDeltas
分派器本身内部的详细计时信息。仅在源代码中启用且 O.timingEnabled==为 True 时才显示。
updateAttrs((Serializable)arg1，（dict）arg2) → None：根据给定字典更新对象属性
class yade.wrapper.Gl1_CpmPhys(继承自 GlIPhysFunctor → 函子 → 可序列化对象)渲染相互作用的 CpmPhys 对象。contactLine(=true) [static]显示接触线dmgLabel(=true) [static]以数值形式显示接触损伤参数dmgPlane(=false) [static]这是什么？ }epsT(=false) [static]显示剪切应变epsTAxes(=false) [static]显示剪切平面的坐标轴normal(=false) [static]显示接触法线colorStrainRatio(=-1) [static]如果为正，则根据 ε_N （由 ε_0 x colorStrainRatio (ε_0 = CpmPhys.epsCrackOnset) 归一化）设置相互作用（线框）的颜色。否则，根据残余强度设置颜色。

epsNLabel(=*false*) [static]以数值形式显示法向应变

bases
: 此函子接受的有序类型列表（以字符串形式表示）。

colorStrainRatio = -1.0

contactLine = True

dict(（可序列化对象）辐角*1*) →字典：返回属性字典。

dmgLabel = True

dmgPlane = False

epsNLabel = False

epsT = False

epsTAxes = False

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

normal = False

timingDeltas
: 分派器内部的详细计时信息。除非在源代码中启用该功能且 O.timingEnabled==为 True，否则为空。

updateAttrs(（可序列化对象）参数*1*，（字典）辐角*2*) → 无：根据给定字典更新对象属性

class yade.wrapper.Gl1_NormPhys(继承 *GlIPhysFunctor* → 函子 → 可序列化对象)将 *NormPhys* 对象渲染为圆柱体，其直径和颜色取决于 *Norm-Phys.normalForce* 的模。

maxFn(= *0*) [static]与 *NormPhys.normalForce* 对应的 *maxRadius*值。此值会自动增大（但 不会减小 ）。

signFilter(=*0*) [static]
: 如果非零，则只显示法向力为负（-1）或正（+1）的接触；如果为零，o, 将显示所有接触。

refRadius(=*std::numeric_limits<*实数*>::infinity()*) [static]参考（最小）颗粒半径；仅在 *maxRadius* 为负时使用。此值会自动减小（但 不会增大 ）。（自动更新）

maxRadius(=*-1*) [static]
: 对应最大法向力的圆柱半径。若为负值，则改用自动更新的 *re-fRadius* 。

slices(=*6*) [static]球体切片数；（参见 glutCylinder 参考文档）

stacks(=*1*) [static]球体堆叠数；（参见 glutCylinder 参考文档）

maxWeakFn(=非数) [static]根据接触的法向力将其划分为“弱组构”和“强组构”的阈值。此值由 *utils.fabricTensor* 作为副作用设置。

weakFilter(=*0*) [static]如果非零，则仅显示属于“弱”（-1）或“强”（+1）组构的接触。

weakScale(=*1.*) [static]
如果 最大弱线函数 已设置，则按此值缩放弱线的半径（通常小于 1）。如果为零，则显示 1 像素的直线。颜色不受此值影响。

bases此函子接受的类型有序列表（以字符串形式表示）。dict(*(可序列化对象)*辐角 *1*）→字典：返回属性字典。label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。maxFn = 0.0maxRadius = -1.0maxWeakFn = nanrefRadius = inf signFilter = 0slices = 6stacks = 1timingDeltas分派器内部的详细计时信息。除非在源代码中启用且 O.timingEnabled==为 True，否则为空。updateAttrs(*(可序列化对象)*参数*1*，*(字典)*辐角 *2*）→ None：根据给定字典更新对象属性weakFilter = 0weakScale = 1.0

2.3.15 仿真数据

Omega

class yade.wrapper.Omega

addScene(*(Omega)*辐角 *1*）→ int：向 Omega 添加新场景，并返回其编号bodies当前模拟中的刚体（支持按标识符索引访问和迭代的容器）cell当前场景的周期单元（如果场景为非周期性，则为 None）。childClassesNonrecursive(*(Omega)*参数*1*，*(str)*辐角 *2*）→ 列表：返回所有派生自给定类、并已在类工厂中注册的类的列表disableGdb(*(Omega)*辐角 *1*）→None：将段错误和异常终止处理器恢复为系统默认值。dt当前时间步长（t）值。请参阅 动态时间步，了解如何通过 Δ 时间步进器启用或禁用自动 时间步进器更新。dynDt是否使用 时间步进器（若存在于 *O.engines*中）来动态控制 Δ时间步长。

dynDtAvailable
是否有 时间步进器位于 *O.engines* 中，无论其是否已激活。

energy
能量跟踪器 当前仿真。（仅在启用 *O.trackEnergy* 时有意义）

engines
模拟中的引擎列表（对应 C++ 源代码中的 Scene::engines）。

exitNoBacktrace(*(Omega)*辐角 *1*[, *(int)status=0*]) → None：禁用段错误处理程序并退出，可选择指定状态码。

filename
当前仿真保存到的文件名（如果从未保存，则为 None）。

forceSyncCount
ForceContainer 中同步次数的计数器，用于性能分析。

forces
ForceContainer （当前仿真中的力、力矩、位移）。

interactions
访问 仿真中的相互作用，使用 g

1. 相互作用双方刚体的标识符，例如 刚体 的相互作用，例如 `O.interactions[23,65]`

2. 遍历整个容器：

```
for i in O.interactions: print i.id1,i.id2
```

注意： 遍历时会静默跳过非实数。

---

isChildClassOf(*(Omega)*参数*1*，*(str)*参数*2*，*(str)*辐角*3*) → 布尔值：
判断第一个类是否派生自第二个类（两者均以字符串形式给出）。

iter
获取当前步数

labeledEngine(*(Omega)*参数*1*，*(str)*辐角*2*) →对象：
返回具有给定标签的引擎/函子实例。用户不应直接调用此函数；每次更改 O.engines 时，都会根据标签为相应的全局 Python 变量赋值。

例如：

*O.engines=[InsertionSortCollider(*标签*='碰撞器')]*
collider.nBins=5 # collider 自动赋值给 *O.engines* 后已成为一个变量

load(*(Omega)*参数*1*，*(str)*文件[, *(bool)*静默*=False*]) → None：
从文件加载模拟。该文件应已在相同版本的 Yade 中保存，或使用相同功能构建或编译；否则无法保证兼容性。兼容性也可能受到不同版本的外部库（如 Boost）的影响保存于

loadTmp(*(Omega)*辐角 *1*[, *(str)*标记*='*[, *(bool)*静默*=False*]]) →None：从先前由 saveTmp 保存在内存中的模拟加载。标记选项可用于区分多个已保存的模拟

lsTmp(*(Omega)*辐角 *1*) → 列表：返回所有保存在内存中的模拟的列表。

materials
共享材料；可通过标识符或标签访问

`miscParams`
模拟中的杂项参数（Scene::mistParams），通常用于保存无法归入其他位置的可序列化对象，例如 GL 函子

`numThreads`
获取 OpenMP 可使用的最大线程数。

`pause`(*(Omega)*辐角 *1*) → 无：停止仿真执行。（可在循环内部调用，并且会在当前步结束后停止。）

`periodic`
获取/设置场景是否具有周期性（True/False）。

`plugins`(*(Omega)*辐角 *1*) →列表：返回类工厂中注册的全部插件列表。

`realtime`
返回模拟已运行的时钟时间（现实世界时间）。

`reload`(*(Omega)*辐角 *1*[, *(*布尔值*)*静默=否]) → 无：重新加载当前仿真

`reset`(*(Omega)*辐角 *1*) → 无：
彻底重置所有模拟（包括其他场景！）。

`resetAllScenes`(*(Omega)*辐角 *1*) →无：
重置所有场景。

`resetCurrentScene`(*(Omega)*辐角 *1*) →无：
重置当前场景。

`resetThisScene`(*(Omega)*辐角 *1*) →无：
重置当前场景。

`resetTime`(*(Omega)*辐角*1*) →无：
重置模拟时间：步数、虚拟时间和实际时间。（不影响其他任何内容，包括计时。）

`run`(*(Omega)*辐角 *1*[, *(int)nSteps=-1*[, *(bool)wait=False*]]) → None：运行模拟。*nSteps* 要运行的步数（如果为正数，则运行完后停止）；*wait* 若为真，在模拟停止之前不会返回到 Python。

`runEngine`(*(Omega)*参数*1*，*(*引擎*)*辐角 *2*) → 无：将给定引擎精确运行一次；模拟时间、步数等不会递增（仅在你清楚自己在做什么时使用）。

`running`
后台线程当前是否正在运行模拟。

`save`(*(Omega)*参数*1*，*(*字符串*)*文件[, *(*布尔值*)*安静=否]) → 无：将当前模拟保存到文件（应为 .xml、.xml.bz2、.yade 或 .yade.gz）。.xml 文件比 .yade 文件大，但通常可以相对轻松地（取决于文件大小）打开和编辑，例如使用文本编辑器。.bz2 和 .gz 均对应压缩版本。成功重新加载有软件要求，参见 *O.load*。

`saveTmp`(*(Omega)*辐角 *1*[, *(*字符串*)*标记=”[, *(*布尔值*)*安静=否]]) → 无：将模拟保存到内存中（关闭时会消失），之后可通过 loadTmp 加载。标记 选项可用于区分不同的内存保存模拟。

`sceneToString`（*(Omega)* 辐角 *1*）→ 对象：以字符串形式返回整个场景。相当于使用 O.save(...)，但场景会输出为字符串，而不是文件。（另请参阅 stringToScene()）

`speed`
返回当前计算速度 [次/秒]。

step(*(Omega)*辐角 *1*）→无：将模拟推进一步。该步完成后返回。

stopAtIter
: 获取/设置模拟停止前的迭代次数。

stopAtTime
: 获取/设置模拟停止前的时间。

stringToScene（*Omega*）参数*1*，（字符串）辐角*2*[，（字符串）标记="]）→ None：从作为参数传入的字符串加载模拟（另请参阅 sceneToString）。

subStep
: 获取当前子步数（仅当 O.subStepping==True 时才有意义）；在循环外时为 -1，否则为 0（O.subStepping==False）或将要运行的引擎数量（O.subStepping==True）

subStepping
: 获取/设置子步进是否处于活动状态。

switchScene(*(Omega)*辐角 *1*）→ 无:
: 切换到备用模拟（同时保留旧模拟）。再次调用此函数会切回第一个模拟。请注意，即使切换后，第一个模拟中的大多数变量仍会引用第一个模拟（例如 b=O.bodies[4]; O.switchScene(); [此处的 b 仍指向第一个模拟中的刚体]）

switchToScene(*(Omega)*参数*1*，*(int)*辐角*2*）→ 无：切换到指定场景。默认场景的编号为 0，其他场景必须通过 addScene 方法创建。

tags
: 当前模拟的标签（字符串=字符串字典），即支持字符串索引访问/赋值的容器

thisScene
: 返回当前场景的标识符。

time
: 返回模拟的虚拟时间（模型世界时间）。

timingEnabled
: 全局启用/禁用计时服务（请参阅计时模块的文档）。

tmpFilename(*(Omega)*辐角 *1*）→ 字符串:
: 返回临时目录中文件的唯一名称，该文件将在 Yade 退出时被删除。

tmpToFile(*(Omega)*参数*1*，*(*字符串*)*文件名[，*(*字符串*)*标记="]）→ 无:
: 将 *saveTmp* 的模拟保存为 XML 到 文件名。

tmpToString(*(Omega)*辐角 *1*[，*(*字符串*)*标记="]）→
字符串：将 *saveTmp* 的模拟作为字符串返回 XML。

trackEnergy
: 在此模拟中启用或禁用能量跟踪时。

wait(*Omega*辐角 *1*）→None：直到模拟暂停后才返回。（如果未在运行，则立即返回）。

BodyContainer

class yade.wrapper.BodyContainer

__init__（对象）参数*1*，（*BodyContainer*）辐角*2*）→ 无

addToClump（*BodyContainer*）参数*1*，（对象）参数*2*，（整数）辐角*3*[，（整数）离散化=*0*]）→ 无

：将刚体 b（或刚体列表）添加到现有团簇 c 中。c 必须是团簇，且 b 不能是 c 的团簇成员。团簇质量和惯性会自动调整（详情请参阅*clump()*）。

请参阅examples/clumps/addToClump-example.py获取示例脚本。

注意：如果 b 本身是团簇，则其所有成员都会添加到 c 中，并且 b 将被删除。如果 b 是团簇 d 的团簇成员，则 d 中的所有成员都会添加到 c 中，并且 d 将被删除。如果只需添加团簇成员 b，请先从 d 中释放该成员。

append(*(BodyContainer)*参数*1*，*(*刚体*)*辐角*2*）→ 整数：追加一个 Body 实例，并返回其标识符。

append((BodyContainer)参数1, (对象)参数2) -> 对象：追加刚体列表 i n-姿态，返回标识符列表

appendClumped（*BodyContainer*）参数*1*，（对象）辐角*2*[，（整数）离散化=*0*]）→ 元组：将给定的刚体列表追加为一个团簇（刚性团聚体）；返回一个由以下内容组成的元组：(clumpId, [memberId1,memberId2,...])。团簇质量和惯性会根据离散化自动计算（详情请参见*clump()*）。

clear（*BodyContainer*）辐角*1*）→ None：移除全部刚体（不检查相互作用）

clump(*(BodyContainer)*参数*1*，*(*对象*)*辐角*2*[，*(*整数*)*离散化=*0*]）→ 整数：

将给定刚体合并为一个团簇（创建刚性团聚体）；返回 clumpId。当团簇成员发生重叠时，要精确定义团簇质量和惯性，需要进行离散化>0；此时通过使用规则网格的单元对质点进行积分/求和来实现（网格单元长度定义为 $L_{min}/discretization$，其中 L_{min} 是轴对齐包围盒的最小长度。如果*离散化*<=0则直接使用成员惯性的总和，速度更快，但仅在成员互不重叠时准确）。

deleteClumpBody(*(BodyContainer)*参数*1*，*(*刚体*)*参数*2*）→ None：删除团簇成员。

deleteClumpMember(*(BodyContainer)*参数*1*，*(*刚体*)*参数*2*，*(*刚体*)*参数*3*）→ None：删除团簇成员。

enableRedirection

让碰撞器在刚体被删除时切换到带刚体重定向的优化算法——默认开启

erase(*(BodyContainer)*参数*1*，*(int)*参数*2*[，*(bool)eraseClumpMembers=0*]）→ bool：删除具有给定标识符的刚体；InteractionLoop 将在下一步中删除全部相互作用。如果删除的是团簇，请使用 $O.bodies.erase(clumpId, True)$ 来删除团簇及其成员。

getRoundness(*(BodyContainer)*参数*1*[，*(*列表*)excludeList=[]*]）→ 浮点数：返回圆度系数 RC = R2/R1。R1 是团簇的等效球半径。R2 是能够包住团簇的球体的最小半径。如果只有球体，则 RC = 1。如果存在团簇，则 $0 < RC < 1$。可通过提供标识符列表将刚体排除在计算之外：$O.bodies.getRoundness([ids])$。

请参阅 examples/clumps/replaceByClumps-example.py ，了解示例脚本。

insertAtId(*(BodyContainer)*参数*1*，*(*刚体*)*参数*2*，*(int)insertatid*）→ int：在指定标识符处插入一个刚体（该标识符处不应存在刚体）

releaseFromClump(*(BodyContainer)*参数*1*，*(int)*参数*2*，*(int)*辐角*3*[，*(int)*离散化=*0*]）→无：从团簇 c 中释放刚体 b。b 必须是 c 的团簇成员。团簇质量和惯性

are 自动调整（详情请参见 团簇 *p()*）。
参见examples/clumps/releaseFromClump-example.py，了解示例脚本。

注意： 如果 c 只包含 2 个成员，b 将不会被释放，并会出现警告。 In
在此算例中，团簇 c 应被擦除。

`replace`(*(BodyContainer)*参数*1*，*(*对象*)*辐角*2*) →对象

`replaceByClumps`(*(BodyContainer)*参数*1*，*(list)*参数*2*，*(object)*参数*3*[, *(int)*离散化*=0*]) →列表:

使用团簇模板列表和数量列表将球体替换为团簇；返回一个元组列表：
`[(clumpId1,[memberId1,memberId2,...]),(clumpId2,[memberId1,memberId2,..]), ...]`。
新团簇的体积与被替换的球体相同。团簇质量和惯性会自动调整（详情请参阅*clump()*）。

O.bodies.replaceByClumps([utils.clumpTemplate([1,1],[.5,.5])] , [.9]) #将用“二元团簇”替换全部独立球体中的 *90%*
请参阅examples/clumps/replaceByClumps-example.py 查看示例脚本。

`updateClumpProperties`(*(BodyContainer)*参数*1*[, *(list)*排除列表*=[]*[, *(int)discretization=5*]]) → None：手动强制 Yade 更新团簇属性：质量、体积和惯性（关于“离散化”值的详情，请参阅*clump()*）。当模拟过程中团簇被修改或删除时可使用此功能。提供标识符列表即可将相应团簇排除在计算之外：*O.bodies.updateProperties([*标识符*])*。

`useRedirection`
当场景使用最新的受限物体和真实物体列表时为 true；在以下情况下会自动设为 true：1/ 启用启用重定向*=True* 后移除刚体，以及 2/ 在 MPI 执行期间。*(*自动更新*)*

InteractionContainer

`class yade.wrapper.InteractionContainer`
通过以下方式访问模拟中的相互作用：

1. 相互作用中两个刚体的标识符，例如： `O.interactions[23,65]`
2. 遍历整个容器：

```
for i in O.interactions: print i.id1,i.id2
```

注意： 迭代会静默跳过虚拟的相互作用，即非实数。

`__init__`(*(object)arg1*，*(InteractionContainer)arg2*) → 无

`all`(*(InteractionContainer)arg1*[, *(bool)onlyReal=False*]) → 列表：返回全部相互作用的列表。若 onlyReal=为 True，则筛除虚拟相互作用；否则（默认）会转储全部内容。

`clear`(*(InteractionContainer)arg1*) →无:
移除全部相互作用，并使持久碰撞检测器数据失效（如果碰撞器支撑此功能）。

`countReal`(*(InteractionContainer)arg1*) → 整数:
返回实数相互作用的数量。

`erase`(*(InteractionContainer)*参数*1*，*(int)*参数*2*，*(int)*辐角*3*) → 否：擦除一个由 id1、id2 指定的相互作用（内部会调用 `requestErase` ——如果 碰撞器 作出相应决定，该相互作用仍可能以势的形式存在）。

`eraseNonReal`(*(InteractionContainer)*辐角*1*) →否：擦除所有不属于真实 的相互作用。`has`(*(InteractionContainer)*参数*1*，*(int)id1*，*(int)id2*[, *(bool)*仅真实交互=*False*]) → bool：判断一对标识符 *id1*，*id2* 是否对应一个现有的相互作用（真实 与否取决于仅真实交互）

`nth`(*(InteractionContainer)*参数*1*，*(int)*辐角*2*) → 相互作用：从容器中返回第 n 个相互作用（可用于随机选取相互作用）。不会访问虚拟相互作用。

`serializeSorted`

`withBody`(*(InteractionContainer)*参数*1*，*(int)*辐角*2*) → 列表：返回指定刚体的实数相互作用列表。`withBodyAll`(*(InteractionContainer)*参数*1*，*(int)*辐角*2*) → 列表：返回指定刚体的全部（实数以及非实数）相互作用列表。

ForceContainer

```
class yade.wrapper.ForceContainer
```

`__init__`(*(object)*对象，*(ForceContainer)*辐角*2*) → 无

`addF`(*(ForceContainer)*参数*1*，*(int)*标识符，*(Vector3)*三维向量[, *(bool)*永久=*False*]) →无：对刚体施加力（累积）。该力仅作用一个迭代，之后由 ForceResetter 重置。# permanent 参数已弃用；请使用 setPermF(...) 代替 addF(...,permanent=True)。

`addT`(*(ForceContainer)arg1*，*(int)id*，*(Vector3)t*[, *(bool)permanent=False*]) → None：对刚体施加力矩（累积）。力矩作用一个迭代，随后由 ForceResetter 重置。# permanent 参数已弃用；请使用 setPermT(...)，而不要使用 addT(...,permanent=True)。

`f`(*(ForceContainer)arg1*，*(int)id*[, *(bool)sync=False*]) → Vector3：刚体上的合力，不包括重力。对于使用 OpenMP 的团簇，设置 sync=True以同步力容器，否则该值将不正确。

`getPermForceUsed`(*(ForceContainer)arg1*) →bool：检查是否存在永久力。`m`(*(ForceContainer)arg1*，*(int)id*[, *(bool)sync=False*]) → Vector3：已弃用的 t（力矩）别名。`permF`(*(ForceContainer)arg1*，*(int)id*) → Vector3：读取刚体上的永久力值（通过 setPermF() 设置）。`permT`(*(ForceContainer)arg1*，*(int)id*) → Vector3：读取刚体上的永久力矩值（通过 setPermT() 设置）。

`reset`(*(ForceContainer)*参数*1*[, *(bool)resetAll=True*]) → None：重置力容器，包括用户定义的永久力/力矩。resetAll=False 将保持永久力/力矩不变。

`setPermF`(*(ForceContainer)*参数*1*,*(int)*参数*2*, *(*三维向量*)*参数*3*) → None：设置刚体上永久力的值。

`setPermT`(*(ForceContainer)*参数*1*,*(int)*参数*2*, *(*三维向量*)*参数*3*) → None：设置刚体上永久力矩的值。

`syncCount`
: ForceContainer 的同步次数（累计）；如果远高于步数，可能存在不必要的同步，影响性能。

t（(*ForceContainer*) 参数*1*，（*int*）标识符[，（*bool*）*sync=False*]）→ 三维向量：施加于刚体的力矩。对于 OpenMP 中的团簇，请将力容器与 sync=True 同步，否则该值将不正确。

MaterialContainer

class yade.wrapper.MaterialContainer
材料容器 材料。可使用以下方式访问材料：

1. range(0,len(cont)) 范围内的数值索引，例如 cont[2]；2. 分配给材料的文本标签，例如 cont['钢']。这需要遍历所有材料，因此不应频繁使用。__init__(*(object)arg1*, *(MaterialContainer)arg2*) → 无append(*(MaterialContainer)arg1*，*(Material)arg2*) → 整数：添加新的共享材料；更改其标识符并返回它。**append((MaterialContainer)arg1, (object)arg2) -> 对象：** 追加材料实例列表，并返回标识符列表。index(*(MaterialContainer)arg1*，*(str)arg2*) → 整数：根据材料标签返回其标识符。

场景

class yade.wrapper.Scene(继承自 可序列化对象)包含整个模拟的对象。dict(*(Serializable)arg1*) →字典：返回属性字典。doSort(=否)向场景添加新刚体时使用。dt(*=1e-8*)用于积分的当前时间步长。isPeriodic(=否)周期性边界条件是否处于活动状态。iter(*=0*)当前迭代（计算步）编号

selectedBody(*=-1*)用户选定的刚体标识符

speed(*=0*)当前计算速度 [迭代/秒]

stopAtIter(*=0*)模拟停止前的迭代次数。

stopAtTime(*=0*)模拟停止前的时间

subStep(*=-1*)子步数量；不应直接更改。-1 表示运行循环前导阶段（单元积分），0...n-1 运行相应的引擎（n 为引擎数量），n 运行收尾阶段（增加步数和时间。

subStepping(=否)
我们当前是否在每一步中推进一个引擎（而不是每次运行都遍历所有引擎）。

tags(=未初始化)任意键=值关联（例如 mp3 标签：作者、日期、版本、描述等）time(=*0*)模拟时间（虚拟时间） [秒]trackEnergy(=否)是否正在跟踪能量。updateAttrs(*(可序列化对象)*参数 *1*，*(字典)*辐角 *2*）→ 无：根据给定的字典更新对象属性

Cell

class yade.wrapper.Cell(继承可序列化对象)周期性边界条件的参数。仅当 O.isPeriodic==为 True 时才适用。dict((可序列化对象)*1*）→字典：返回属性字典。getDefGrad((单元)*1*）→Matrix3：返回单元变形的变形梯度张量 $\mathbf{F}$ （http://en.wikipedia.org/wiki/ 有限_应变_理论）
getEulerianAlmansiStrain((单元)*1*）→Matrix3：返回单元的欧拉-阿尔曼西应变张量 $e = \frac{1}{2}(\mathbf{I} - \mathbf{b}^{-1}) = \frac{1}{2}(\mathbf{I} - (\mathbf{F}\mathbf{F}^{\mathrm{T}})^{-1})$ （http: //en.wikipedia.org/wiki/Finite_应变_理论）
getLCauchyGreenDef((单元)*1*）→ Matrix3：返回单元的左柯西-格林变形张量 $\mathbf{b} = \mathbf{F}\mathbf{F}^{\mathrm{T}}$ （ http://en.wikipedia.org/ wiki/Finite_应变_理论）

getLagrangianStrain(*(单元)1*）→Matrix3：返回拉格朗日应变张量 $\mathbf{E} = \frac{1}{2}(\mathbf{C} - \mathbf{I}) = \frac{1}{2}(\mathbf{F}^{\mathrm{T}}\mathbf{F} - \mathbf{I}) = \frac{1}{2}(\mathbf{U}^2 - \mathbf{I})$ 单元的（http://en.wikipedia.org/wiki/Finite_应变_理论）

getLeftStretch(*(Cell)*参数 *1*）→ Matrix3：返回单元的左（空间）伸长张量（由极分解 $\mathbf{U}$ 得到的矩阵 $\mathbf{F} = \mathbf{R}\mathbf{U}$ ）

getPolarDecOfDefGrad(*(Cell)*参数 *1*）→元组：返回正交矩阵 $\mathbf{R}$ 和对称半正定矩阵 $\mathbf{U}$ ，作为单元变形梯度 $\mathbf{F}$ 的极分解（ $\mathbf{F} = \mathbf{R}\mathbf{U}$ ）

getRCauchyGreenDef(*(Cell)*参数 *1*）→ Matrix3：返回单元的右柯西-格林变形张量 $\mathbf{C} = \mathbf{F}^{\mathrm{T}}\mathbf{F}$ （ http://en.wikipedia.org/wiki/Finite_应变_理论）

getRightStretch(*(Cell)*参数 *1*）→Matrix3：返回单元的右（材料）伸长张量（由极分解得到的矩阵 $\mathbf{V}$ $\mathbf{F} = \mathbf{R}\mathbf{U} = \mathbf{V}\mathbf{R} \rightarrow \mathbf{V} = \mathbf{F}\mathbf{R}^{\mathrm{T}}$ ）
getRotation(*(Cell)*参数 *1*）→ Matrix3：返回单元的旋转（由极分解 $\mathbf{R}$ 得到的正交矩阵 $\mathbf{F} = \mathbf{R}\mathbf{U}$ ）

getSmallStrain(*(Cell)*参数 *1*）→Matrix3：返回单元的小应变张量 $\boldsymbol{\varepsilon} = \frac{1}{2}(\mathbf{F} + \mathbf{F}^{\mathrm{T}}) - \mathbf{I}$ （ http://en.wikipedia.org/wiki/Finite_应变_理论）

getSpin(*(单元)辐角 1*）→ 三维向量：返回由 速度梯度 的反对称部分定义的自旋

hSize
: 基准单元向量（矩阵的列），在每一步都根据 速度梯度 （变换 累积应用的 速度梯度 变换）更新。模拟过程中不支持设置 *hSize*

大多数接触定律都不支持此操作；它仅用于在任何相互作用创建之前的第 0 次迭代。

hSize0
相对于当前变换而言，未经变换的 hSize 的值（计算方式为 变换 (计算为 变换 $^{1}\times$ *hSize*。

homoDeform(=*2*)
如果 >0，则通过调整刚体的位置和速度，对单元进行仿射变形（速度梯度）。速度变化可通过对表达式 v= v.x 求导得到，其中 v 为宏观速度梯度，其增量形式为： Δv=Δ v x + v Δx。因此，一旦 velGrad 发生变化，速度就会根据第一项立即改变： Δv(t)=Δ v x(t)，而第二项则体现了对流项： Δv'= v v(t-dt/2)。如果 homoDeform=1，则忽略第二项。若 homoDeform=2 （默认值），则包含所有项。

nextVelGrad(=*Matrix3r::Zero()*)
参见 *Cell.velGrad*。

prevHSize(=*Matrix3r::Identity()*)前一步的 *hSize* 用于定义跨周期的相对速度
.

prevVelGrad(=*Matrix3r::Zero()*)前一步的速度梯度。

refHSize(=*Matrix3r::Identity()*)参考单元配置，仅与 *OpenGLRenderer.dispScale* 配合使用。当直接赋值给 *hSize* 或 *trsf* 时会自动更新；也会由*utils.setRefSe3* 修改（例如由用户界面中的 Reference 按钮调用）。

refSize
单元参考尺寸（初始单元向量的长度，即 *hSize* 的列范数）。

注意： 不建议修改此值，请改用 *setBox* 。

---

setBox(*(单元)参数1*, *(三维向量)*辐角 *2*) → 无：将 单元 形状设为长方体，尺寸沿指定坐标轴取给定参数。此操作等同于将相应值组成的对角矩阵赋给 *hSize*。

setBox((Cell)arg1, (float)arg2, (float)arg3, (float)arg4) -> 无：将 单元 形状设为矩形，其沿 x、 y、 z 的尺寸由参数指定。此操作相当于将对应条目赋值给 *hSize* 对角矩阵。

shearPt(*(Cell)arg1*, *(Vector3)*辐角 *2*) → 三维向量：对该点应用剪切（单元斜切+旋转）

shearTrsf
当前的斜切+旋转变换（不调整尺寸）

size
单元的当前尺寸，即 *Cell.hSize* 列中包含的 3 个单元横向向量的长度 s.
每一步都会自动更新。

trsf
C单元的当前变换矩阵，由对 *Cell.velGrad* 进行时间积分得到。

unshearPt（单元）参数*1*， （三维向量）辐角 *2*) → 三维向量：对该点应用逆剪切变换（消除单元的斜切+旋转）

unshearTrsf
当前斜切+旋转变换的逆矩阵（不调整尺寸）

updateAttrs(*(可序列化对象)参数1*, *(字典)*辐角 *2*) → None：从给定字典更新对象属性

velGrad
变换的速度梯度；用于 *NewtonIntegrator*。 速度梯度 的值会在每一步中累加到变换 中。

注意：在时间步开始时更改速度梯度会导致该步的积分不准确，因为通常应在接触定律之后（但在 Newton 之前）更改它。为避免此问题，赋值会自动延迟。赋予的值会在内部存储于 *Cell.nextVelGrad* 中，并由 Newton 积分器在适当的时间应用。

警告： 无法像 *O.cell.velGrad[0,0]=1* 这样赋值各个分量（这不会返回错误，但也不会产生任何效果）。应像 *O.cell.velGrad=Matrix3(...)*这样对整个矩阵赋值。或者，也可以直接对 *nextVelGrad* 赋值（既可以逐分量赋值，也可以整体赋值），效果应该相同。

velGradChanged(=*false*)
当速度梯度被手动更改时为 true（另请参阅 *Cell.nextVelGrad*）

volume
单元的当前体积。

wrap(*(单元)参数1, (三维向量)辐角2*) → 三维向量：将任意点变换为参考单元中的点

wrapPt(*(单元)参数1, (三维向量)辐角2*) → 三维向量：将点包裹在参考单元内，假设单元没有斜切+旋转。

2.3.16 其他类

class yade.wrapper.TimingDeltas

data获取计时数据，格式为元组列表（标签、执行时间[纳秒], 执行次数）（每个检查点对应一个元组）reset(*(TimingDeltas)辐角1*) →无：重置计时信息

class yade.wrapper.Serializable

dict(*(可序列化对象)辐角1*) →字典：返回属性字典。updateAttrs(*(可序列化对象)参数1、(字典)辐角2*) → 无：根据给定字典更新对象属性

class yade.wrapper.GlExtra_LawTester(继承 *GlExtraDrawer* → 可序列化对象)
查找 *LawTester* 的一个实例，并以可视化方式显示其数据。dead(=否)停用对象（发生错误/异常时）。dict(*(可序列化对象)辐角1*) →字典：返回属性字典。tester(=已取消初始化)关联的*LawTester* 对象。updateAttrs(*(可序列化对象)参数1、(字典)辐角2*) → 无：根据给定字典更新对象属性

class yade.wrapper.MatchMaker(继承*Serializ*能够)

根据一对标识符进行匹配的类，可返回预定义值（针对 匹配项 中定义的一对标识符），也可返回派生值（使用 算法 计算得出）的标量参数。它可以在 Python 和 C++ 中通过 id1， id2， val1=NaN， val2=NaN 调用++。

注意： 此类有一个 转换器，可将 Python 数字转换为该类定义的对象，并创建一个新的 *MatchMaker*，使其返回该数字的值；因此，无需传递对象实例，只需传递数字值，系统便会自动转换。

---

algo
未找到标识符匹配项时用于计算值的算法。可能的值包括

- 'avg'（算术平均值）
- 'min'（最小值）
- 'max'（最大值）
- 'harmAvg'（调和平均）

以下算法否 需要有意义的输入值即可运行：

- 'val'（返回由 *val* 指定的值）
- 'zero'（始终返回 0。）

computeFallback(*(MatchMaker)*参数*1*，*(*浮点数*)val1*，*(*浮点数*)val2*) → 浮点数：
使用*val1* 和*val2*，根据算法计算算法值。

dict(*(*可序列化对象*)*辐角*1*) →字典：
返回属性字典。

matches(=未初始化)包含 (id1,id2,value) 个项目的数组；查询匹配 id1 + id2 或 id2 + id1 将返回 value

updateAttrs(*(*可序列化对象*)*参数*1*，*(*字典*)*辐角*2*) → 无：
从给定字典更新对象属性

val(=非数)无匹配项时返回的常量值，且算法为 val

class yade.wrapper.Engine(继承可序列化对象)
模拟的基本执行单元，由模拟循环（O.engines）调用

dead(=否)若为真，则此引擎完全不会运行；可用于暂时停用某个引擎，并仅在稍后的某个时刻重新启用它。

dict(*(Serializable)arg1*) →字典：返
回属性字典。

execCount
此引擎的累计运行次数（仅在 *O.timingEnabled*==True 时使用）。

execTime
此引擎运行所耗费的累计时间（以纳秒计）（仅在 if *O.timingEnabled*==True）。

label(=未初始化)此对象的文本标签；必须是有效的 Python 标识符，可直接在 Python 中引用。

ompThreads(=-1)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由“yade -jN”定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或在将 *ParallelEngine* 与运行并行区域的引擎结合使用时；这会导致嵌套 OMP 循环在各层使用不同的线程数。

timingDeltas
引擎本身的详细计时信息。除非在源代码中启用 *O.timingEnabled*==True，否则为空。

updateAttrs(*(可序列化对象)arg1*，*(字典)arg2*) → None：根据给定字典更新对象属性

class yade.wrapper.EnergyTracker(继承可序列化对象)
用于跟踪能量的存储空间。仅当 *O.trackEnergy* 为 True 时使用。

clear(*(能量跟踪器)辐角1*) →无：清除所有已存储的值。

dict(*(可序列化对象)辐角1*) →字典：
返回属性字典。

energies(=未初始化)
能量值，以线性数组形式

items(*(能量跟踪器)辐角1*) →列表：
以 (名称, 值) 元组列表的形式返回内容。

keys(*(EnergyTracker)辐角1*) →列表：
返回已定义的能量。

total(*(EnergyTracker)辐角1*) →浮点数：
返回全部能量的总和。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → None：从给定字典更新对象属性

class yade.wrapper.LinExponentialPotential(继承*Cundall-Strack 势* → *GenericPotential* → 可序列化对象)
仅采用 Cundall-Strack 类接触的 LinExponential 势。LinExponential 势的公式为 $F(u) = \frac{k*(x_e-x_0)}{x_e}(u/a-x_0)\exp\left(\frac{-(u/a)}{x_e-x_0}\right)$。其中，k 为原点斜率，$x_0$ 为势能为零的位置，x_e 为极值位置。

F0(=1)
接触力。$F_0 = F(u=0)$ 时的力（LinExponential）

Fe(=1)
极值力。极值处的力值。（LinExponential）

alpha(=1)
体刚度与粗糙度刚度比

computeParametersFromF0(*(LinExponentialPotential)参数1*，*(浮点数)初始力*，*(浮点数)xe*，*(浮点数)k*) →无：设置势能参数，其中 k 根据 F_0计算得出

computeParametersFromF0Fe(*(LinExponentialPotential)参数1*，*(浮点数)xe*，*(浮点数)Fe*，*(浮点数)F0*)→ None：设置势能参数，其中 k 和 x_0 由 F_0 和 F_e计算得出

dict(*(可序列化对象)辐角1*) →字典：
返回属性字典。

k(=1)
原点斜率（刚度）。（LinExponential）

`potential`((LinExponentialPotential)参数1，(浮点数)u）→ 浮点数：获取任意点处的势值。`setParameters`((LinExponentialPotential)参数1，(浮点数)x0，(浮点数)xe，(浮点数)k）→ None：设置势能参数`updateAttrs`((Serializable)参数1，(字典)辐角*2*）→ None：根据给定字典更新对象属性`x0`(=*0*)平衡距离。势力在 x_0 处为 0。(LinExponential)`xe`(=*1*)极值位置。力的局部最大值/最小值所在的位置。(LinExponential)`class yade.wrapper.CundallStrackPotential`(继承 GenericPotential →Serializable)仅包含 Cundall-Strack 类接触的势。`alpha`(=*1*)体刚度与粗糙度刚度比`dict`((Serializable)辐角*1*）→字典：返回属性字典。`updateAttrs`((Serializable)参数1，(字典)辐角*2*）→ None：根据给定字典更新对象属性`class yade.wrapper.GlExtra_OctreeCubes`(继承 GlExtraDrawer → Serializable)渲染从文件读取的盒体`boxesFile`(=未初始化)读取盒体的文件；ASCII 文件包含 `x0 y0 z0 x1 y1 z1 c` 条记录，其中 `c` 是指定填充方式的整数（0 表示线框，1 表示填充）。`dead`(=false)停用对象（发生错误/异常时）。`dict`((Serializable)辐角*1*）→字典：返回属性字典。`fillRangeDraw`(=Vector2i(-2, 2))将要渲染的填充索引范围。`fillRangeFill`(=Vector2i(2, 2))将要填充的填充索引范围。`levelRangeDraw`(=Vector2i(-2, 2))将要渲染的层级范围。`noFillZero`(=true)不填充填充值为 0 的盒体（即进一步细分的盒体）`updateAttrs`((Serializable)参数1，(字典)辐角*2*）→ None：根据给定字典更新对象属性`class yade.wrapper.ParallelEngine`(继承引擎 → Serializable)用于并行运行其他引擎的引擎。`__init__`((对象)辐角*1*）→None 对象.__初始化__(元组 args，字典 kwds).__初始化__((对象)参数1，(列表)参数2) -> 对象：根据（可能嵌套的）从属引擎列表构造。`dead`(=false)若为 true，此引擎将完全不运行；可用于暂时停用引擎，并在之后重新启用。

`dict`((可序列化对象)辐角 *1*) →字典:
返回属性字典。

`execCount`
此引擎的累计运行次数（仅在*O.timingEnabled*==`True`时使用）。

`execTime`
此引擎运行所耗的累计时间（以纳秒为单位，仅在 if *O.timingEnabled*==`True`时使用）。

`label`(=未初始化)
此对象的文本标签；必须是有效的 Python 标识符，可直接从 Python 中引用。

`ompThreads`(=*-1*)
引擎中要使用的线程数。如果 ompThreads<0 （默认值），线程数通常为 OMP_NUM_THREADS，或由"yade -jN"定义的数量 N（不过，此行为可能取决于引擎）。此属性仅影响代码中包含 OpenMP 并行区域的引擎（例如 *InteractionLoop*）。此属性主要适用于实验，或与 *ParallelEngine* 结合使用会运行并行区域的引擎，从而形成嵌套 OMP 循环，且每一层的线程数不同。

`slaves`
引擎列表的列表；每个顶层组将与其他组并行运行，而每组内的引擎将按给定顺序依次运行。

`timingDeltas`
引擎本身内部的详细计时信息。除非在源代码中启用且设置了 *O.timingEnabled*==`True`，否则为空。

`updateAttrs`((可序列化对象)*arg1*，(字典)*arg2*) →
None：根据给定字典更新对象属性

class `yade.wrapper.Cell`(继承可序列化对象)
周期性边界条件的参数。仅当 O.isPeriodic==为 True 时适用。

`dict`((可序列化对象)辐角 *1*) →字典:
返回属性字典。

`getDefGrad`((单元)辐角 *1*) →Matrix3：返回单元变形的变形梯度张量 $\mathbf{F}$ （http://en.wikipedia.org/wiki/ 有限_应变_理论）

`getEulerianAlmansiStrain`((单元)辐角 *1*) →Matrix3：返回单元的欧拉-阿尔曼西应变张量 $\boldsymbol{e} = \frac{1}{2}(\mathbf{I} - \mathbf{b}^{-1}) = \frac{1}{2}(\mathbf{I} - (\mathbf{F}\mathbf{F}^{\mathsf{T}})^{-1})$ （http: //en.wikipedia.org/wiki/有限_应变_理论）

`getLCauchyGreenDef`((单元)辐角 *1*) → Matrix3：返回单元的左柯西-格林变形张量 $\mathbf{b} = \mathbf{F}\mathbf{F}^{\mathsf{T}}$ （http://en.wikipedia.org/ wiki/有限_应变_理论）

`getLagrangianStrain`((单元)辐角 *1*) →Matrix3：返回单元的拉格朗日应变张量 $\mathbf{E} = \frac{1}{2}(\mathbf{C} - \mathbf{I}) = \frac{1}{2}(\mathbf{F}^{\mathsf{T}}\mathbf{F} - \mathbf{I}) = \frac{1}{2}(\mathbf{U}^2 - \mathbf{I})$（http://en.wikipedia.org/wiki/Finite_应变_理论）

`getLeftStretch`((单元)辐角 *1*) → Matrix3：返回单元的左（空间）伸长张量（由极分解得到的矩阵 $\mathbf{U}$ $\mathbf{F} = \mathbf{R}\mathbf{U}$ ）

`getPolarDecOfDefGrad`((单元)辐角 *1*) →元组：返回单元变形梯度 $\mathbf{F}$ 的极分解中的正交矩阵 $\mathbf{R}$ 和对称半正定矩阵 $\mathbf{U}$ （ $\mathbf{F} = \mathbf{R}\mathbf{U}$ ）

`getRCauchyGreenDef`((单元)辐角 *1*) → Matrix3：返回单元的右柯西-格林变形张量 $\mathbf{C} = \mathbf{F}^{\mathsf{T}}\mathbf{F}$ （http://en.wikipedia.org/wiki/Finite_应变_理论）

`getRightStretch`（（单元）辐角*1*）→Matrix3：
返回单元的右侧（材料）拉伸张量（矩阵 $\mathbf{V}$，来自极分解 $\mathbf{F} = \mathbf{RU} = \mathbf{VR} \rightarrow \mathbf{V} = \mathbf{FR}^{T}$）

`getRotation`(*(单元)辐角1*) → Matrix3：
返回单元的旋转（正交矩阵 $\mathbf{R}$，来自极分解 $\mathbf{F} = \mathbf{RU}$）

`getSmallStrain`（（单元）辐角*1*）→Matrix3：返回小应变张量 $\varepsilon = \frac{1}{2}(\mathbf{F}+\mathbf{F}^{T})-\mathbf{I}$ 单元的（http://en.wikipedia.org/wiki/Finite_应变_理论）

`getSpin`(*(单元)辐角1*) → 三维向量：
返回由 速度梯度 的反对称部分定义的自旋

`hSize`
基准单元向量（矩阵的列），每一步都会根据 速度梯度 (变换 累积应用的 速度梯度 变换) 进行更新。在模拟过程中设置 *hSize* 不受大多数接触定律支持；它仅用于在第 0 次迭代、创建任何相互作用之前设置。

`hSize0`
相对于当前 变换 的未变换 hSize 值（计算方式为 变换 $^{1} \times$ *hSize* .

`homoDeform`(*=2*)
如果 >0，则通过调整刚体的位置和速度，对单元进行均匀变形（速度梯度）。速度变化通过对表达式 v 求导得到= v.x，其中 v 是宏观速度梯度，以增量形式表示为： Δv=Δ v x + v Δx。因此，一旦 `velGrad` 发生变化，速度就会根据第一项立即改变： Δv(t)=Δ v x(t)，而第二项则反映了对流项： Δv'= v v(t-dt/2)。如果 homoDeform=1，则忽略第二项。若 homoDeform=2 （默认值），则包含所有项。

`nextVelGrad`(*=Matrix3r::Zero()*)
请参见*Cell.velGrad*。

`prevHSize`(*=Matrix3r::Identity()*)
前一步的 *hSize*，用于定义跨周期的相对速度。

`prevVelGrad`(*=Matrix3r::Zero()*)前一步的速度梯度。

`refHSize`(*=Matrix3r::Identity()*)
参考单元配置，仅与*OpenGLRenderer.dispScale*一起使用。当直接赋值*hSize* 或*trsf* 时会自动更新；也可由*utils.setRefSe3* （例如由用户界面中的 `Reference` 按钮调用）修改。

`refSize`
单元参考尺寸（初始单元向量的长度，即 *hSize* 的列范数）。

> **注意：** 不建议修改此值，请改用 *setBox* 。

`setBox`(*(辐角)参数*) → None：将单元形状设为长方体，各坐标轴上的尺寸由给定参数指定。这是将对角矩阵赋给 *hSize* 的简写形式。

setBox((单元)参数1, (浮点数)参数2, (浮点数)参数3, (浮点数)参数4) -> None： 将单元形状设为长方体，各坐标轴上的尺寸由 x， y， z 指定。这是将相应元素构成的对角矩阵赋给 *hSize* 的简写形式。

`shearPt`（（单元）参数*1*，（三维向量）辐角*2*）→ 三维向量：对点应用剪切（单元斜切+旋转）

`shearTrsf`
当前斜切+旋转变换（不调整尺寸）

size
单元的当前尺寸，即 *Cell.hSize* 列中包含的 3 个单元横向向量的长度。每一步都会自动更新。

trsf
单元的当前变换矩阵，由 *Cell.velGrad* 的时间积分得出。

unshearPt(*(单元)参数1*，*(三维向量)辐角2*) → 三维向量：对该点应用逆剪切（移除单元的斜切+旋转）

unshearTrsf
当前斜切+旋转变换的逆矩阵（不调整尺寸）

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无：根据给定字典更新对象属性

velGrad
变换的速度梯度；用于 *NewtonIntegrator*。每一步中，速度梯度 的值会累积到变换 中。

> 注意：在时间步开始时更改速度梯度会导致该步的积分不准确，因为通常应在接触定律之后（但在 Newton 之前）更改它。为避免此问题，赋值会自动延后。赋予的值会内部存储在 *Cell.nextVelGrad* 中，并由 Newton 积分器在适当的时间应用。

> **警告：** 无法像下面这样赋值给各个分量： *O.cell.velGrad[0,0]=1* （不会返回任何错误，但也不会产生任何效果）。应像下面这样赋值整个矩阵： *O.cell.velGrad=Matrix3(...)*。或者，也可以直接为 *nextVelGrad* 赋值（既可以逐分量赋值，也可以整体赋值），效果应该相同。

velGradChanged(*=false*)
当速度梯度被手动更改时为 true（另请参阅 *Cell.nextVelGrad*）

volume
单元的当前体积。

wrap(*(单元)参数1*，*(三维向量)辐角2*) → 三维向量：将任意点变换为参考单元中的一个点

wrapPt(*(单元)参数1*，*(三维向量)辐角2*) → 三维向量：假设单元没有斜切时，参考单元内的 wrap 点+旋转。

class yade.wrapper.CundallStrackAdhesivePotential(继承 *Cundall-Strack 势* → *GenericPotential* →可序列化对象)
带黏附部分的 Cundall-Strack 模型。当 $u/a - \varepsilon < 0$ 时创建接触，并在以下情况下释放：$u/a - \varepsilon > l_{adh}$，其中 $l_{adh} = f_{adh}/k_n$。这会导致滞回吸引部分。

alpha(*=1*)
体刚度与粗糙度刚度比

dict(*(可序列化对象)辐角1*) →字典：
返回属性字典。

fadh(*=0*)
黏附力。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角2*) → 无：
根据给定字典更新对象属性

class yade.wrapper.GenericPotential(继承可序列化对象)
用于 PotentialLubrication 定律中势表示方式的通用类。不会执行任何操作。如果将其设为势，结果将是仅润滑模拟。

dict(*(可序列化对象)辐角 1*) →字典：
返回属性字典。

updateAttrs(*(可序列化对象)参数1*，*(字典)辐角 2*) →
None：从给定字典更新对象属性

class yade.wrapper.GlExtraDrawer(继承可序列化对象)
Perf执行任意 OpenGL 绘图命令；由 *OpenGL 渲染器* （参见 *OpenGLRenderer.extraDrawers*）在常规渲染例程完成后调用。

T此类本身不执行任何渲染，派生类应重写 *render* 方法。

dead(*=false*)停用该对象（发生错误/异常时）。dict(*(Serializable)arg1*) →字典：返回属性字典。

updateAttrs(*(Serializable)arg1*，*(dict)arg2*) →
None：根据给定字典更新对象属性

2.4 Yade 模块参考

2.4.1 yade.bodiesHandling 模块

用于处理刚体的杂项函数。

yade.bodiesHandling.facetsDimensions(*idFacets=[]*，掩码*=-1*)该函数接受面片 ID 列表或面片列表，并计算最大和最小尺寸以及几何中心。

参数

- **idFacets** (*list*) – 球体列表
- **mask** (*int*)– 已检查刚体的 *Body.mask*

返回字典，包含以下键： `min` （最小维度，三维向量）、 `max` （最大维度，三维向量）、 `minId` （最小维度面片 ID，三维向量）、 `maxId` （最大维度面片 ID，三维向量）、 `center` （包围盒中心点，三维向量）、 `extends`（包围盒尺寸，三维向量）、 `number` （面片数量，整数），

yade.bodiesHandling.sphereDuplicate(球体 *ID*)
该函数会复制球体

yade.bodiesHandling.spheresModify(球体 *ID* 列表*=[]*，掩码*=-1*，平移=三维向量*(0，0，0)*，缩放*=1.0*，方向=四元数*((1，0，0)，0)*，复制*=False*)
该函数接受球体标识符列表或刚体列表，并对其进行修改：旋转、缩放、平移。如果 copy= True，则复制刚体并对其进行修改。此外，还可以指定掩码。如果 idSpheres 不为空，该函数只影响通过掩码的刚体。如果 idSpheres 为空，该函数将搜索通过掩码的刚体。

参数

- **shift** (Vector3) – 移动球体的三维向量(X、Y、Z)参数。
- **scale** (*float*) – 缩放指定球体的因子。
- **orientation** (Quaternion) – 球体的方向
- **mask** (*int*)– *Body.mask* 用于经过检查的刚体

若 copy= 为 True，则返回刚体列表；若 copy= 为 False，则返回布尔值

yade.bodiesHandling.spheresPackDimensions(*idSpheres=[]*, 掩码=*-1*)

该函数接受球体标识符列表或刚体列表，并计算最大和最小尺寸及几何中心。

参数

- **idSpheres** (*list*) – 球体列表
- **mask** （*int*） – *Body.mask* 所选刚体的掩码

返回一个字典，其键为 `min` （最小维度，三维向量）、 `max` （最大维度，三维向量）、 `minId` （最小维度球体标识符，三维向量）、 `maxId` （最大维度球体标识符，三维向量）、 `center` （包围盒中心点，三维向量）、`extends` （包围盒尺寸，三维向量）、 `volume` （球体体积，实数）、 `mass`（球体质量，实数）、 `number` （球体数量，整数），

2.4.2 yade.export 模块

将几何体（以及其他内容）导出为各种格式。

class yade.export.VTKExporter(继承对象)

类 用于将数据导出为 VTK 简单旧式文件 （例如，出于某种原因，您无法够使用 *VTKRecorder*）。已支持导出：

- 球体
- 面片
- 多面体
- PotentialBlocks
- 相互作用
- 接触点
- 周期单元

用法：

- 创建对象 `vtkExporter = VTKExporter('baseFileName')`,
- 添加到 `O.engines` 一个 `PyRunner` ，包含 `command='vtkExporter.exportSomething(...)'`
- 或者，直接在脚本末尾使用 `vtkExporter.exportSomething(...)` ，例如

示例： examples/test/vtk-exporter/vtkExporter.py, examples/test/unv-read/unvReadVTKExport.py。

参数

- **baseName** (*string*) – 导出文件的名称。文件将命名为，例如，baseName-spheres-snapNb.vtk 或 baseName-facets-snapNb.vtk
- **startSnap** (*int*) – 文件编号将从 `startSnap` 开始

exportContactPoints(标识符='全部',内容=*{}*, 使用参考位置=*{}*, 注释='注释', *nu mLa-bel*=无)导出接触点（CPs）和已定义的属性。

参数

- **ids** (*[(int,int)]*) – 参见 *exportInteractions()*
- **what**(*dictionary*) – 参见 *exportInteractions()*
- **useRef**(*bool*) – 参见 *exportInteractions()*
- **comment** (*string*) – 要添加到 VTK 文件中的注释

- **numLabel** (*int*) – 文件数量（例如时间步），如果未指定，则使用上次使用的值 + 1

exportFacets(标识符='全部'，内容=*{}*，注释='注释'，文件编号=无)导出面片（位置）和定义的属性。面片以重复节点的形式导出

参数

- **ids** (*[int]*/*"all"*) – 如果是"全部"，则导出所有面片；否则仅导出整数列表中的面片
- **what**(*dictionary*) – 参见 *exportSpheres()*
- **comment** (*string*) – 要添加到 VTK 文件的注释
- **numLabel** (*int*) – 文件编号（例如时间步）；如果未指定，则使用上次使用的值 + 1

exportFacetsAsMesh(标识符='全部'，连接表=无，内容=*{}*，注释='注释'，文件编号=无)

导出面片（位置）和已定义的属性。面片将作为网格导出（不会复制节点）。因此需要额外指定参数连接表

参数

- **ids** (*[int]*/*"all"*) – 如果为"全部"，则导出所有面片；否则仅导出整数列表中的面片
- **what**(*dictionary*) – 参见 *exportSpheres()*
- **comment** (*string*) – 添加到 VTK 文件中的注释
- **numLabel** (*int*) – 文件的编号（例如时间步）；若未指定，则使用上次使用的值 + 1
- **nodes** (*[(float,float,float)*/*Vector3]*) – 节点坐标列表
- **connectivityTable** (*[(int,int,int)]*) – 各个元素（面片）的节点编号列表

exportInteractions(标识符='全部'，内容=*{}*,顶点导出内容=*{}*,注释='注释',*numLa- bel*=*None*, 使用参考位置=*False*)导出相互作用和已定义的属性。

参数

- **ids** (*[(int,int)]*/*"all"*) – 如果为"全部"，则 export 所有相互作用，否则仅 export (int,int) 列表中的相互作用
- **what** (*dictionary*) – 要 export 的内容。参数是一个名称->command 字典。名称是保存到 VTK 中时使用的字符串，command 是要执行的字符串。注意，在此函数中，相互作用的标签为 i。标量、向量和张量变量均已支持。例如，要 export 与某个值 (1e9) 的刚度差异（命名为 `dStiff`），应写为：
  `what=dict(dStiff='i.phys.kn-1e9', ... )`
- **verticesWhat** (*dictionary*) – 要在相连刚体上 export 的内容。刚体的标签为 `b`（如果需要区别处理两个刚体，则为 `b1` 和 `b2`）
- **comment** (*string*) – 要添加到 VTK 文件的注释
- **numLabel** (*int*) – 文件数量（例如时间步），如果未指定，则使用上次使用的值 + 1
- **useRef** (*bool*) – 若为 False（默认值），则 export 时使用刚体的当前位置，否则使用参考位置

exportPeriodicCell(注释='注释'，文件编号=无)导出单元用于
周期性模拟的几何体。

参数

- **comment** (*string*) – 要添加到 VTK 文件的注释
- **numLabel** (*int*) – 文件编号（例如时间步）；若未指定，则使用最后一次使用的值 + 1

exportPolyhedra(标识符=‘全部’、内容=*{}*、注释=‘注释’、文件编号=无、使用参考位置=否)导出多面体及已定义的属性。

参数

- **ids** (*[int] / "all"*) – 如果为“全部”，则导出所有多面体；否则仅导出整数列表中的多面体
- **what** (*dictionary*) – 要导出的其他物理量（除位置外）。参数是名称-> command 字典。名称是保存到 VTK 文件中的字符串，command 是要执行的字符串。请注意，在此函数中，刚体被标记为 b。标量、向量和张量变量均已支持。例如，要导出速度（命名为 particleVelocity）和到点 (0,0,0) 的距离（命名为 dist），应写为：`what=dict(particleVelocity='b.state.vel',dist='b.state.pos.norm()', ... )`
- **comment** (*string*) – 要添加到 VTK 文件中的注释
- **numLabel** (*int*) – 文件编号（例如时间步），如果未指定，则使用上次使用的值 + 1

exportPotentialBlocks(标识符=’全部’，内容=*{}*，注释=’注释’， 文件编号=无，使用参考位置=否)导出势块和已定义的属性。

参数

- **ids** (*[int] / "all"*) – 如果为“全部”，则导出所有势块；否则仅导出整数列表中的势块
- **what** (*dictionary*) – 要导出的其他物理量（除位置外）。参数是名称-> command 字典。名称是保存到 VTK 文件时使用的字符串，command 是要执行的字符串。注意，在此函数中，刚体的标签为 b。标量、向量和张量变量均已支持。例如，要导出速度（命名为 parti- cleVelocity）以及与点 (0,0,0) 的距离（命名为 dist），应写为：
  `what=dict(particleVelocity='b.state.vel',dist='b.state.pos.norm()', ... )`
- **comment** (*string*) – 要添加到 VTK 文件的注释
- **numLabel** (*int*) – 文件编号（例如时间步）；如果未指定，则使用上次使用的值 + 1

exportSpheres(标识符=’全部’, 内容=*{}*,注释=’注释’,文件编号=*None*,使用参考位置=*False*)导出球体（位置和半径）以及已定义的属性。

参数

- **ids** (*[int]/"all"*) – 如果为“全部”，则导出所有球体；否则仅导出整数列表中的球体
- **what** (*dictionary*) – 要导出的其他物理量（位置和半径除外）。参数是名称->command 字典。名称是保存到 VTK 中的字符串，command 是要执行的字符串。请注意，在此函数中，刚体以 b 标记。标量、向量和张量变量均已支持。例如，要导出速度（名称为 particleVelocity）以及到点 (0,0,0) 的距离（名称为 dist ），你应当

写入： what=dict(particleVelocity='b.state.vel',dist='b.state.pos.norm()', ...)

- **comment** (*string*) – 添加到 VTK 文件的注释
- **numLabel** (*int*) – 文件的数量（例如时间步），如果未指定，则使用上次使用的值 + 1
- **useRef** (*bool*) – 如果为 False（默认值），则使用球体的当前位置进行 export，否则使用参考位置

class yade.export.VTKWriter(继承 *object*)

用法：创建对象 vtk_writer = VTKWriter('base_文件_名称')，将其添加到引擎 PyRunner，并使用 command='vtk_writer.snapshot()'

snapshot()

yade.export.gmshGeo(文件名，注释='', 掩码=*-1*，精度=*-1*)将球体保存到 geo 文件中，以供随后在 GMSH（http://www.geuz.org/gmsh/doc/texinfo/）程序中使用。球体可在其中进行网格划分。

参数

- **filename** (*string*) – 导出球体坐标的文件名。
- **mask** (*int*) – 仅导出具有相应掩码的球体
- **accuracy** (*float*) – 设置 geo 文件中各点的精度参数。默认值：最小球体直径的 1/10。

返回已导出的球体数量。

返回类型 int

yade.export.text(文件名，掩码=*-1*)

将球体坐标保存到文本文件中；直线的格式为：x y z r。非球形刚体将被静默跳过。示例已添加到 examples/regular-sphere-pack/regular-sphere-pack.py

参数

- **filename** (*string*) – 将导出球体坐标的文件名。
- **mask** (*int*) – 仅导出具有对应掩码的球体

返回 已写入的球体数量。

返回类型 int

yade.export.text2vtk(输入文件名，输出文件名，注释='注释')将文本文件（由 *export.textExt* 函数创建）转换为 VTK 文件。请参阅 examples/test/paraview-spheres-solid-section/export_text.py 示例

参数

- **inFileName** (*str*) – 输入文本文件的名称
- **outFileName** (*str*) – 输出 VTK 文件的名称
- **comment** (*str*) – VTK 文件中的可选注释

yade.export.text2vtkSection(输入文件名，输出文件名，点，法向量=*(1，0，0)*)将文本文件（由 *export.textExt* 函数创建）中穿过球体的截面转换为 VTK 文件。请参阅 examples/test/paraview-spheres-solid-section/export_text.py示例

参数

- **inFileName** (*str*) – 输入文本文件的名称
- **outFileName** (*str*) – 输出 VTK 文件的名称

- **point** (*Vector3|(float,float,float)*) – 位于截面平面上的一个点的坐标
- **normal** (*Vector3|(float,float,float)*) – 截面平面的法向量

yade.export.**textClumps**(文件名，格式=*'x_y_z_r_*团簇 *ID'*，注释=*''*，掩码=*-1*)将团簇成员保存到文本文件中。非团簇成员（刚体）会被静默跳过。

参数

- **filename** (*string*) – 将导出球体坐标的文件名。
- **comment** (*string*) – 将作为注释添加到文件顶部的文本。如果要创建多行文本，请在后续行使用'\n#'。
- **mask** (*int*) – 仅导出掩码相应的球体 仅导出掩码相应的球体

返回团簇数量、写入的球体数量。

返回类型 int

yade.export.**textExt**(文件名,格式=*'x_y_z_r'*, 注释=*''*, 掩码=*-1*, 属性=*[]*)将球体坐标及其他参数保存到特定格式的文本文件中。非球形刚体将被静默跳过。用户可以在此添加自己的特定格式，并为其指定有意义的名称。文件的第一行将包含格式名称。请确保在 ymport.textExt 中添加相同的格式规范。

参数

- **filename** (*string*) – 将导出球体坐标的文件名。
- **format** (*string*) – 输出格式的名称。已支持 'x_y_z_r'（默认）、'x_y_z_r_matId'、'x_y_z_r_attrs'（使用适当的注释）
- **comment** (*string*) – 将作为注释添加到文件顶部的文本。若要创建多行文本，请在后续行前使用 '\n#'。对于 'x_y_z_r_attrs' 格式，最后一行（或唯一一行）应由作为 attrs 传入的物理量的列标题组成（标量使用 1 个注释词，向量使用 3 个注释词，矩阵使用 9 个注释词）
- **mask** (*int*) – 仅导出具有相应掩码的球体；仅导出具有相应掩码的球体
- **attrs** (*[str]*) – 要通过 'x_y_z_r_attrs' 格式导出的属性。列表中的每个字符串都会针对导出的每个刚体进行求值，body=b（即 'b.state.pos.norm()' 表示刚体到坐标系原点的距离）

返回 已写入的球体数量。

返回类型 int

yade.export.**textPolyhedra**(文件名,注释=*''*, 掩码=*-1*, 解释性注释=*True*, *at-trs=[]*)将多面体保存到文本文件中。非多面体刚体将被静默跳过。

参数

- **filename** (*string*) – 输出文件的名称
- **comment** (*string*) – 将作为注释添加到文件顶部的文本。若要创建多行文本，请在后续行前使用'\n#'。
- **mask** (*int*) – 仅导出具有相应掩码的多面体
- **explanationComment** (*str*) – 在文件开头添加格式说明

返回写入的多面体数量。

返回类型 int

2.4.3 yade.geom 模块

根据面片创建几何对象。

yade.geom.**facetBox**(中心,半尺寸,方向=*Quaternion((1, 0, 0), 0)*, 壁面掩码=*63*, ***kw*)创建由面片组成的任意对齐盒子，并指定其中心、半尺寸和方向。如果盒子的某个尺寸为零，则不会创建对应的面片。面片的朝向均为从盒子向外。

参数

- **center** (Vector3) – 盒子的中心
- **extents** (Vector3) – 盒子各边的半长度
- **orientation** (Quaternion) – 盒子的方向
- **wallMask** (*bitmask*) – 确定要创建哪些壁面，顺序为 -x (1)、 +x (2)、-y (4)、 +y (8)、-z (16)、 +z (32)。这些数字执行按位与运算；默认值 63 表示创建全部壁面
- ****kw** –（未使用的关键字参数）传递给 *utils.facet*

返回 组成盒子的面片列表

yade.geom.**facetBunker**(中心，料仓直径，出料口直径，料仓高度，出料口高度，料仓管道高度=*0.0*, 方向=四元数*((1，0，0)，0)*，分段数=*10*，壁面掩码=*4*，角度范围=无，闭合间隙=否，**关键字参数)创建一个方向任意的料斗，由面片组成，并指定其中心、半径、高度和方向。返回组成该料斗的面片列表；



参数

- **center** (Vector3) – 所创建料斗的中心
- **dBunker** (*float*) – 料斗直径，顶部
- **dOutput** (*float*) – 料斗出料口直径
- **hBunker** (*float*) – 料斗高度
- **hOutput** (*float*) – 料斗出口高度
- **hPipe** (*float*) – 料斗管道高度

- **orientation** (Quaternion)– 料斗的方向；参考方向的轴沿着 +x 轴。
- **segmentsNumber** (*int*) – 料斗表面的边数（>=5）
- **wallMask** (*bitmask*) – 确定要创建哪些壁面，顺序为上方 (1)、下方 (2)、侧面 (4)。这些数字按位与运算；默认值 7 表示创建所有壁面
- **angleRange** (*(ϑmin,Θmax)*) – 可通过指定角度范围仅创建料斗的一部分；如果为 None，则假定范围为 (0,2*pi)。
- **closeGap** (*bool*) – 在圆柱底面使用三角形面片时，跳过 angleRange 中的闭合范围。
- ****kw** –（未使用的关键字参数）传递给 utils.facet;

yade.geom.facetCone(中心，顶部半径，底部半径，高度，方向=四元数*((1，0，0)，0)*，分段数=*10*，壁面掩码=*7*，角度范围=无，闭合间隙=假，顶部内半径=*-1*，底部内半径=*-1*，**kw*)创建由面片组成、可任意定向的圆锥，指定其中心、半径、高度和方向。返回构成该圆锥的面片列表;

参数

- **center** (Vector3) – 所创建圆柱体的中心
- **radiusTop** (*float*) – 圆锥顶部半径
- **radiusBottom** (*float*) – 圆锥底部半径
- **radiusTopInner** (*float*) – 圆锥顶部的内半径，默认为 -1
- **radiusBottomInner** (*float*) – 圆锥底部的内半径，默认为 -1
- **height** (*float*) – 圆锥高度
- **orientation** (Quaternion) – 圆锥的方向；参考方向为沿 +x 轴的轴向。
- **segmentsNumber** (*int*) – 圆锥表面上的边数（>=5）
- **wallMask** (*bitmask*) – 决定要创建哪些壁面，顺序为上（1）、下（2）、侧面（4）。这些数字按位与；默认值 7 表示创建所有壁面
- **angleRange** (*(ϑmin,Θmax)*) – 通过指定角度范围，仅创建部分圆锥；如果 None，则假定范围为 (0,2*pi)。
- **closeGap** (*bool*) – 使用圆柱底面三角形面片时，角度范围内会跳过近距离。
- ****kw** – 传递给 utils.facet 的（未使用的关键字参数）；

yade.geom.facetCylinder(中心，半径,高度,方向=四元数*((1，0，0)，0)*，分段数量=*10*，壁面掩码=*7*,角度范围=无,闭合间隙=否,顶部内半径=*-1*，底部内半径=*-1*，**kw*）

创建一个由面片组成、方向任意的圆柱，指定中心、半径、高度和方向。返回组成圆柱的面片列表;

参数

- **center** (Vector3) – 所创建圆柱的中心
- **radius** (*float*) – 圆柱半径
- **height** (*float*) – 圆柱体高度
- **radiusTopInner** (*float*) – 圆柱体顶部的内半径，默认为 -1
- **radiusBottomInner** (*float*) – 圆柱体底部的内半径，默认为 -1

- **orientation** （Quaternion）– 圆柱体的方向；参考方 +x 向的轴沿着该轴。
- **segmentsNumber** (*int*) – 圆柱体表面的边数（>=5）
- **wallMask** (*bitmask*) – 决定要创建哪些壁面，顺序为上方 (1)、下方 (2)、侧面 (4)。这些数字按位与运算；默认值 7 表示创建全部壁面
- **angleRange** (*(ϑmin,Θmax)*) – 可通过指定角度范围，仅创建料斗的一部分；如果 None，则假定范围为 (0,2*pi)。
- **closeGap** (*bool*) – 在圆柱底面使用三角形面片，跳过 angleRange 中的封闭范围。
- ****kw** –（未使用的关键字参数）传递给 utils.facet；

yade.geom.**facetCylinderConeGenerator**(中心, 顶部半径,高度,方向=四元数*((1,0, 0), 0)*, 分段数=*10*, 壁面掩码=*7*,*an- gleRange=None*, 闭合间隙=*False*, 底部半径=*-1*,顶部内半径=*-1*, 底部内半径=*-1*, ***kw*)

请勿直接使用此函数！请改用 geom.facetCylinder 和 geom.facetCone。此函数是用于通过面片生成圆柱体和圆锥体的基础函数。

参数

- **radiusTop** (*float*) – 顶部半径
- **radiusBottom** (*float*) – 底部半径
- ****kw** –（传递给 utils.facet 的未使用关键字参数）；

yade.geom.**facetHelix**(中心, 外半径, 螺距, 方向=四元数*((1, 0, 0), 0)*, 分段数量=*10*, 角度范围=无, 内半径=*0*, ***kw*)

创建由面片组成、方向任意的螺旋体，并指定其中心、半径（外半径和内半径）、螺距和方向。返回构成螺旋体的面片列表；

参数

- **center** (Vector3) – 所创建圆柱体的中心
- **radiusOuter** (*float*) – 外半径
- **radiusInner** (*float*) – 内部高度（可以为 0）
- **orientation** (Quaternion) – 螺旋体的方向；参考方向的轴沿着 +x 轴。
- **segmentsNumber** (*int*) – 螺旋体表面上的边数（>=3）
- **angleRange** (*(ϑmin,Θmax)*) – 角度范围；如果是 None，则假定为 (0,2*pi)。
- ****kw** – 传递给 utils.facet 的（未使用的关键字参数）；

yade.geom.**facetParallelepiped**(中心, 半尺寸, 高度, 方向=四元数*((1, 0, 0), 0)*, 壁面掩码=*63*, ***kw*)

创建由面片组成、方向可任意对齐的平行六面体，并指定其中心、半尺寸、高度和方向。如果平行六面体的任一维度为零，则不会创建对应的面片。面片朝向平行六面体的外侧。

参数

- **center** (Vector3) – 平行六面体的中心
- **extents**(Vector3) – 平行六面体各边的半长度
- **height** (*Real*) – 平行六面体的高度（沿 z 轴）
- **orientation** (Quaternion) – 平行六面体的方向

- **wallMask** (*bitmask*) – 决定要创建哪些壁面，顺序为 -x (1)、 +x (2)、-y (4)、+y (8)、-z (16)、 +z (32)。这些数字按位与运算；默认值 63 表示创建全部壁面
- ****kw** – （未使用的关键字参数）传递给 *utils.facet*

返回 构成平行六面体的面片列表

yade.geom.facetPolygon(中心，外半径，方向=四元数*((1，0，0)，0)*，分段数量=*10*，角度范围=无，内半径=*0*，**关键字参数)创建由面片组成、可任意对齐的多边形，并指定其中心、半径（外半径和内半径）和方向。返回构成该多边形的面片列表；

参数

- **center** (Vector3) – 所创建圆柱体的中心
- **radiusOuter** (*float*) – 外半径
- **radiusInner** (*float*) – 内部高度（可以为 0）
- **orientation** (Quaternion) – 多边形的方向；参考方向的轴沿着 +x 轴。
- **segmentsNumber** (*int*) – 多边形表面上的边数（>=3）
- **angleRange** (*(ϑmin,Θmax)*) – 通过指定角度范围，仅创建多边形的一部分；如果为 None，则假定为 (0,2*pi)。
- ****kw** – （未使用的关键字参数）传递给 utils.facet；

yade.geom.facetPolygonHelixGenerator(中心，外半径，螺距=*0*，方向=四元数*((1，0，0)，0)*，分段数=*10*，角度范围=无，内半径=*0*，***kw*)

请勿直接使用此函数！请改用 geom.facetPloygon 和 geom.facetHelix。此基础函数用于根据面片生成多边形和螺旋体。

yade.geom.facetSphere(中心,半径, *θ*分辨率=*8*,*φ*分辨率=*8*, 返回元素映射=否, ***kw*)创建由面片构成、可任意对齐的球体，并指定其中心、半径和方向。返回构成该球体的面片列表。参数参考 ParaView 球体图元。

参数

- **center** (Vector3) – 所创建球体的中心
- **radius** (*float*) – 球体半径
- **thetaResolution** (*int*) – “赤道”周围的面片数量
- **phiResolution** (*int*) – “两极”之间的面片数量 + 1
- **returnElementMap** (*bool*) – 如果为真，还会返回节点元组 ((x1,y1,z1),(x2,y2,z2),...) 和元素 ((id01,id02,id03),(id11,id12,id13),...)；否则仅返回面片
- ****kw** – （未使用的关键字参数）传递给 utils.facet；

2.4.4 yade.gridpfacet 模块

用于创建圆柱体、网格和膜的辅助函数。有关此类元素的更多详细信息，请参见 [Effeindzourou2016], [Effeindzourou2015a], [Bourrier2013]。

有关使用网格连接的示例，请参见

- examples/grids/CohesiveGridConnectionSphere.py
- 示例/网格/网格连接_弹簧.py
- examples/grids/Simple_Grid_Falling.py

- examples/grids/Simple_网格连接_Falling.py

有关 *PFacet* 的示例，请参阅

- examples/pfacet/gts-pfacet.py
- examples/pfacet/mesh-pfacet.py
- examples/pfacet/pfacetcreators.py

yade.gridpfacet.**chainedCylinder**(起点=三维向量*(0，0，0)*，终点=三维向量*(1，0，0)*，半径=*0.2*，动态=无，固定=否，线框=否，颜色=无，高亮=否，材料=*-1*，掩码=*1*)

创建并连接一个具有给定参数的链式圆柱体。重复调用所生成的形状后是折线与球体的闵可夫斯基和。

参数

- **radius** (*Real*) – 闵可夫斯基和中球体的半径。
- **begin** （Vector3）– 在闵可夫斯基和中确定直线位置的第一个点
- **last** (Vector3) – 在闵可夫斯基和中确定直线位置的最后一个点

为了构建正确的链，秩为 N 的元素的最后一个点必须与同一条链中秩为 N+1 的元素的第一个点相对应（允许一定的容差，因为将使用包围盒创建连接）。

返回具有链式圆柱体形状的刚体对象。

注意： 链式圆柱体已弃用，未来将被移除，请改用网格连接。请参阅*gridpfacet.cylinder* 和 *gridpfacet.cylinderConnection*。

---

yade.gridpfacet.**cylinder**(起点=三维向量*(0，0，0)*，终点=三维向量*(1，0，0)*，半径=*0.2*，节点标识符=*[]*，圆柱体标识符=*[]*，动态=无，固定=假，线框=假，颜色=无，高亮=假，内部材料=*-1*，外部材料=*-1*，掩码=*1*)

根据给定参数创建一个圆柱体。该形状对应于直线线段与球体的闵可夫斯基和，因此圆柱体的顶点经过圆角处理。圆柱体（网格连接）及其对应的节点（yref:网格节点<网格节点>）会自动添加到模拟中。包含节点和圆柱体标识符的列表会自动更新。

参数

- **begin** (Vector3) – 全局坐标系中闵可夫斯基和的第一个点。
- **end** (Vector3) – 全局坐标系中闵可夫斯基和的最后一个点。
- **radius** (*Real*) – 闵可夫斯基和中球体的半径。
- **nodesIds** (*list*) – 已存在的网格节点的标识符列表。新标识符将被添加。
- **cylIds** （*list*）– 列出已存在的网格连接的标识符。将添加新的标识符。
- **intMaterial** – *Body.material* 用于创建两个网格节点之间的相互作用物理
- **extMaterial** – *Body.material* 用于创建圆柱体（网格连接）与其他刚体之间的相互作用物理（例如，球体与圆柱体之间的相互作用）

请参阅 *utils.sphere*的文档，了解其他参数的含义。

yade.gridpfacet.cylinderConnection(顶点，半径=*0.2*，*nodesIds=[]*，*cylIds=[]*，动态=无，固定=否，线框=否，颜色=无，*highlight*=否，内部材料=*-1*，外部材料=*-1*，掩码=*1*)

使用给定参数创建一条圆柱体链。圆柱体（网格连接）及其对应的节点（yref:网格节点<网格节点>）会自动添加到模拟中。节点列表和圆柱体标识符列表也会自动更新。

参数 **vertices** (*[*Vector3*]*) – 全局坐标系中待连接顶点的坐标。

其他参数的含义请参阅 *gridpfacet.cylinder* 文档。

yade.gridpfacet.gmshPFacet(*meshfile='file.mesh'*,平移=*Vector3(0,0, 0)*,缩放=*1.0*,方向=*Quaternion((1, 0,0), 0)*,半径=*1.0*, 线框=*True*,固定=*True*,节点材料=*-1*,材料=*-1*, 颜色=*None*)

从 .mesh 文件导入网格几何体，并自动创建相连的 PFacet 单元。示例请参阅 examples/pfacet/mesh-pfacet.py。

参数

- **filename** (*string*) – 要读取的 .gts 文件。
- **shift** (*[float,float,float]*) – [X、Y、Z] 参数用于平移网格。
- **scale** (*float*) – 因子用于缩放网格。
- **orientation** (*quaternion*) – 导入的几何体的方向。
- **radius** (*float*) – 用于创建 *PFacet* 的半径。
- **materialNodes** – 指定 *GridNodes* 的 刚体材料 网格节点。此材料用于生成内部连接。
- **material** – 指定 *PFacets* 的 刚体材料 *PFacet*。此材料用于与外部物体交互。

有关其他参数的含义，请参阅 *utils.sphere* 文档。

返回 列表： 网格节点 标识符 *nodesIds*、网格连接 标识符 *cylIds*以及 *PFacet* 标识符 *pfIds*

可以使用 GMSH轻松创建网格文件。

可从以下网址下载其他网格文件示例： http://www-roc.inria.fr/gamma/download/download.php

yade.gridpfacet.gridConnection(标识符*1*，标识符*2*，半径，线框=否，颜色=无，高亮=否，材料=*-1*，掩码=*1*，单元格距离=无)创建一个网格连接，通过连接两个网格节点。

参数

- **id1, id2** – 构成圆柱体的两个网格节点。
- **radius** (*float*) – 圆柱体的半径。注意，半径必须与网格节点的半径相同。
- **cellDist** (Vector3)– 关于周期性边界条件，请参阅*Interaction.cellDist*。注意：网格连接的周期性边界条件尚未实现！

其他参数的含义请参阅*utils.sphere* 的文档。

返回具有网格连接形状的刚体对象。

注意：将使用网格节点的材料来设置内部连接的本构行为，即圆柱体的本构行为。网格连接的材料用于与其他（外部）刚体之间的相互作用。

---

yade.gridpfacet.gridNode(中心，半径，动态=无，固定=否，线框=否，颜色=无，高亮=否，材料=*-1*)创建一个网格节点，用于设置网格连接。

有关参数含义，请参阅 *utils.sphere* 的文档。

返回一个 Body 对象，其 网格节点 形状。

yade.gridpfacet.gtsPFacet(网格文件，平移=三维向量(0, 0, 0)，缩放=*1*.0，半径=*1*，线框=True，固定=True，节点材料=-1，材料=-1，颜色=None)从 .gts 文件导入网格几何体，并自动创建相互连接的 PFacet3 元素。示例请参见 examples/pfacet/gts-pfacet.py。

参数

- **filename** (*string*) – 要读取的 .gts 文件。
- **shift** (*[float,float,float]*) – [X、Y、Z] 参数用于平移网格。
- **scale** (*float*) – 因子用于缩放网格。
- **radius** （*float*）– 用于创建*PFacet*的半径。
- **materialNodes** – 指定网格节点的 *Body.material* 。该材料用于创建内部连接。
- **material** – 指定*PFacet* 的 *Body.material* 。该材料用于与外部物体的相互作用。

有关其他参数的含义，请参阅*utils.sphere* 文档。

返回列表，其中包含网格节点标识符*nodesIds*、网格连接标识符*cylIds*以及*PFacet* 标识符*pfIds*

yade.gridpfacet.pfacet(*id1*、*id2*、*id3*、线框=是、颜色=无、高亮=否、材料=*-1*、掩码=*1*、单元格距离=无)从 3 个*PFacet* 网格节点创建一个元素，这些节点已通过 3 个网格节点网格连接相互连接：网格连接：

参数

- **id1、id2、id3** – 已通过网格连接连接的网格节点
- **wire** (*bool*) – 如果 True，则顶部和底部面片以线框形式显示；否则面片将被填充。
- **color** (*Vector3-or-None*) – PFacet 的颜色；如果 None，则会分配随机颜色。
- **cellDist** (Vector3) – 关于周期性边界条件，请参见*Interaction.cellDist*。注意：PFacet 尚未实现周期性边界条件！

有关其他参数的含义，请参阅*utils.sphere* 的文档。

返回具有PFacet形状的*Body* 对象。

注意： 网格节点和网格连接必须具有相同的半径。此半径也用于创建*PFacet*

---

yade.gridpfacet.pfacetCreator1(顶点，半径，节点标识符=*[]*，圆柱体标识符=*[]*，*pfIds*=*[]*，线框=否，固定=是，节点材料=*-1*，材料=*-1*，颜色=无)

从 3 个顶点创建一个*PFacet* 单元，并自动将其添加到模拟中。该函数使用这些顶点创建网格节点，并自动检查是否存在已有节点。

参数

- **vertices** (*[*Vector3*,*Vector3*,*Vector3*]*) – 全局坐标系中顶点的坐标。
- **radius** (*float*) – 用于创建*PFacet*的半径。

- **nodesIds** (*list*) – 已存在网格节点的标识符列表。将添加新的标识符。
- **cylIds** （*list*）——包含已存在网格连接的标识符列表。新标识符将被添加。
- **pfIds** （*list*）——包含已存在*PFacet*的标识符列表。新标识符将被添加。
- **materialNodes** – 指定*Body.material* （网格节点）的材料。该材料用于创建内部连接。
- **material** – 指定*Body.material* （*PFacet*）的材料。该材料用于与外部物体进行交互。

有关其他参数含义，请参阅*utils.sphere* 的文档。

yade.gridpfacet.**pfacetCreator2**(*id1,id2,* 顶点， 半径， *nodesIds=[]*， 线框=是,节点材料=*-1*, 材料=*-1*, 颜色=无, 固定=是)从 2 个已存在且相连的 *PFacet* 网格节点网格节点和一个顶点创建一个元素。该元素会自动添加到模拟中。

参数

- **id1,id2** (*int*) – 已通过 *GridConnection* 连接的 *GridNodes* 的标识符。
- **vertex** (Vector3) – 顶点在全局坐标系中的坐标 .

其他参数的含义请参阅 *gridpfacet.pfacetCreator1* 的文档。

yade.gridpfacet.**pfacetCreator3**(*id1*, *id2*, *id3*, *cylIds=[]*,*pfIds=[]*, 线框=是, 材料=*-1*,颜色=无,固定=是, 掩码=*-1*)从 3 个尚未连接的已存在网格节点创建一个 *PFacet* 元素。该元素会自动添加到模拟中。

参数 **id1, id2, id3** （*int*）– 构成 网格节点 的 *PFacet* 的标识符。

有关其他参数的含义，请参阅 *gridpfacet.pfacetCreator1* 的文档。

yade.gridpfacet.**pfacetCreator4**(*id1*、*id2*、*id3*、*pfIds=[]*、线框=*True*、材料=*-1*、颜色=*None*、固定=*True*、掩码=*-1*)从 3 个已存在的 网格连接 元素创建一个 *PFacet*。该元素会自动追加到模拟中。

参数 **id1, id2, id3** （*int*）– 构成 网格连接 的 *PFacet* 的标识符。

有关其他参数的含义，请参阅 *gridpfacet.pfacetCreator1* 的文档。

2.4.5 yade.libVersions 模块

yade.libVersions 模块会记录编译时使用的全部库的版本。用法示例如下：

```
from yade.libVersions import *
if(getVersion('cgal') > (4,9,0)):
        …
else:
        …
```

若要获取全部库的列表，请使用函数 *libVersions.printAllVersions*。

此模块会检测 先决条件 中列出的全部库。

注意：如果需要使用某个未列在 先决条件中的库，则还必须将其添加到该列表中。

添加新版本时，请查看以下三个文件：

1. py/_libVersions.cpp：通过 C 检测 `#include` 文件的版本++。2. py/libVersions.py.in：在编译期间由 CMake 构建的 Python 模块。全部 `*.in`文件均由 CMake 处理。3.

cMake/FindMissingVersions.cmake：强制检测无法检测到版本的库。

提示： 比较版本最稳妥的方法是使用 Python 内置的元组比较，例如 `if(cgalVer > (4,9,0) and cgalVer < (5,1,1))`:。

yade.libVersions.**getAllVersions**(*rst格式=False*)

返回 `str` - 此函数返回 *printAllVersions(rstFormat)*调用在字符串变量中的结果。

yade.libVersions.**getAllVersionsCmake**()

此函数返回编译期间由 CMake 提供的库版本。

返回以下格式的字典：`{ "libName" : [ (major, minor,patchlevel), "versionString" ] }`

例如，下方的字典展示了编译本文档时使用的库（此处仅列出了由CMAKE检测到的库）：

```
Yade [1]: from yade.libVersions import *

Yade [2]: getAllVersionsCmake()
Out[2]:
{'cmake': [(3, 16, 3), '3.16.3'],
 'compiler': [(9, 4, 0), '/usr/bin/c++ 9.4.0'],
 'boost': [(1, 71, 0), '107100'],
 'freeglut': [(2, 8, 1), '2.8.1'],
 'python': [(3, 8, 10), '3.8.10'],
 'eigen': [(3, 3, 7), '3.3.7'],
 'ipython': [(7, 13, 0), '7.13.0'],
 'sphinx': [(1, 8, 5), '1.8.5-final-0'],
 'mpi4py': [(3, 0, 3), '3.0.3'],
 'mpmath': [(1, 1, 0), '1.1.0']}
```

注意： 请在此处添加 Yade 开始使用其他库时的检测代码，或者如果你发现如何从 CMake 中提取我尚未添加的版本，也请添加。

yade.libVersions.**getArchitecture**()

返回 包含处理器架构名称的字符串，该名称由 `uname -m` 调用报告，或来自 `CMAKE_HOST_SYSTEM_PROCESSOR` CMake 变量。_ _ _

yade.libVersions.**getLinuxVersion**()

返回 包含 Linux 发行版和版本的字符串，最好是 `PRETTY_-NAME` 文件 `/etc/os-release`中的值。

yade.libVersions.**getVersion**(*库名称*)

此函数返回包含库版本号的元组 `(major, minor, patchlevel)`。位于`yade --test` 文件 py/tests/libVersions.py 中的 test 用于验证此版本是否与 CMake 和 C++检测到的版本相同。如果只有其中一个能够检测到库版本，则使用该版本号。

参数 `libName`(*string*) – 库的名称

返回 格式为 `(major, minor, patchlevel)` 的元组（如果 `libName` 存在）。否则返回 `None`。

注意： 库 OpenBLAS 的头文件中未正确定义版本，因此此函数将为 OpenBLAS 返回(0,0,0) 。解析版本字符串并不可靠。CMake 检测到的 MPI 版本有时与 C++ 检测到的版本不同，这还需要进一步调查。

yade.libVersions.**printAllVersions**(*rst格式=False*)This function 打印包含库版本的格式美观的表格 信息。

参数 **rstFormat** (*bool*) – 是否使用 reStructuredText 格式打印表格。默认为 False 并使用 GitLab Markdown 规则 打印，以便轻松粘贴到 GitLab 讨论区。

例如，下表实际上反映了编译本文档时所用的库：

```
Yade [1]: printAllVersions()

```
Yade version   :  2021-11-19.git-639e121
Yade features  :  QT5
Yade config dir:  ~/.yadeflip
Yade precision :  53 bits, 15 decimal places, without mpmath, PrecisionDouble
```

Libraries used :

library	cmake	C++
boost	107100	1.71.0
cmake	3.16.3	
compiler	/usr/bin/c++ 9.4.0	gcc 9.4.0
eigen	3.3.7	3.3.7
freeglut	2.8.1	
gl		20190805
ipython	7.13.0	
mpi4py	3.0.3	
mpmath	1.1.0	
python	3.8.10	3.8.10
qglviewer		2.6.3
qt		5.12.8
sphinx	1.8.5-final-0	
sqlite		3.31.1

```
Linux version  :  Ubuntu 20.04.4 LTS
Architecture   :  amd64
Little endian  :  True
```
```

注意： 为方便起见，启动时会执行 `from yade.libVersions import printAllVersions` ，以便随时调用此函数。

yade._libVersions.**getAllVersionsCpp**() → 字典 此函数返回编译期间 C++ 从所有#include 头文件中检测到的库版本。调试时，这有助于检测某些库之间的 .so冲突。

返回以下格式的字典： { "libName" : [(major, minor, patch) , "versionString"] }

例如，下面的字典反映了编译本文档时使用的库（此处仅列出由 C++ 检测到的库）：

```
Yade [1]: from yade.libVersions import *

Yade [2]: getAllVersionsCpp()
Out[2]:
{'compiler': [(9, 4, 0), 'gcc 9.4.0'],
 'boost': [(1, 71, 0), '1.71.0'],
 'qt': [(5, 12, 8), '5.12.8'],
 'gl': [(2019, 8, 5), '20190805'],
 'qglviewer': [(2, 6, 3), '2.6.3'],
 'python': [(3, 8, 10), '3.8.10'],
 'eigen': [(3, 3, 7), '3.3.7'],
 'sqlite': [(3, 31, 1), '3.31.1'],
 'vtk': [],
 'cgal': [],
 'suitesparse': [],
 'openblas': [],
 'metis': [],
 'mpi': [],
 'clp': [],
 'coinutils': [],
 'mpfr': [],
 'mpc': []}
```

注意： 当 Yade 开始使用其他库时，请在此处添加 C++ 对这些库的检测。

2.4.6 yade.linterpolation 模块

该模块为分段线性函数的操作提供基础支持（这些函数通常是高阶函数的插值，因此得名）。插值始终由两个长度相同的列表给出，其中 x 列表必须按递增顺序排列。

通过假设插值可以从最后一个 x 值 wrap 到第一个 x 值（要得到有意义的结果，第一个 x 值应为 0），即可支持周期性。

非周期　字典插值可以通过用常数　t
头部和尾部 填充插值，将其转换为周期性插值，使用 sanitizeInterpolation 函数。

pkg/common/Engine/PartialEngine/LinearInterpolate.hpp 中提供了一个用于在此类序列上进行插值的 c++模板函数（有状态，因此适合顺序读取，速度很快）。

TODO：目前尚不支持在 Python 中进行插值。

`yade.linterpolation.integral`(*x*、*y*)返回由点 x0、x1、... 和 y0、y1、... 给出的分段线性函数的积分

`yade.linterpolation.revIntegrateLinear`(*I*，*x0*，*y0*，*x1*，*y1*)辅助函数：对于经过 (x0,y0)、(x1,y1) 的线性函数 f，返回积分变量 x 的值，使得 1. x [x0，x1] 2。 _x0^x f dx=I；如果这样的数不存在，或者解不唯一（有这种可能吗？），则引发异常。

`yade.linterpolation.sanitizeInterpolation`(*x*，*y*，*x0*，*x1*)扩展分段线性函数，使其至少覆盖 x0...x1 区间：在开头（使用 y0）和/或末尾（使用 y1）添加常数填充，也可以不添加。

`yade.linterpolation.xFractionalFromIntegral`(积分，*x*，*y*)
返回范围 x0...x1 内满足以下条件的 x：积分 x0^x f dx 等于给定的积分值。如果在此范围内未达到积分值，则引发错误。

yade.linterpolation.xFromIntegral(积分值、x、y) 返回使 _x0^x f dx==积分的 x 值。x 在 xn 处循环。因而，为获得有意义的结果，x0 应当 $== 0$

2.4.7 yade.log 模块

该 yade.log 模块提供了一个接口，用于访问基于boost::log实现的 yade 日志框架。完整文档请参阅调试章节。Python 中的用法示例如下：

```
import yade.log
yade.log.setLevel('PeriTriaxController',yade.log.TRACE)
```

C++ 中的用法示例如下：

```
LOG_WARN("Something: "<<something)
```

yade._log.defaultConfigFileName() → str

返回默认日志配置文件；如果该文件存在，则会在启动时加载。

yade._log.getAllLevels() →字典

返回一个包含 Yade 中所有已知日志记录器的 Python 字典。未设置调试级别的日志记录器，其值为 -1，表示将使用 Default 过滤日志级别。

yade._log.getDefaultLogLevel() → 整数

返回当前的 Default 过滤日志级别。

yade._log.getMaxLevel() → 整数

返回当前 yade 构建的最大_日志_级别。

yade._log.getUsedLevels() → 字典

返回Yade 中所有已使用日志级别的 Python 字典。没有调试级别（值为 -1）的项会被省略。

yade._log.readConfigFile(*(str)*辐角 *1*) → 无。
加载给定的配置文件。

参数 **fname** (*str*) – 要加载的配置文件。

yade._log.resetOutputStream() →无。将日志输出流重置为默认状态：全部日志都会打印到 std::clog 通道，该通道通常会重定向到 std::cerr。

yade._log.saveConfigFile(*(str)*辐角 *1*) → 无。
将日志配置保存到指定文件。

参数 **fname** (*str*) – 要保存的配置文件。

yade._log.setDefaultLogLevel(*(int)*辐角 *1*) → 无

参数 **level** (*int*) – 设置 Default 过滤日志级别，与调用 log.setLevel("Default",level) 相同。

yade._log.setLevel(*(str)*参数*1*，*(int)*辐角 *2*) → 无 为指定的日志记录器设置过滤级别（常数 TRACE (6)、 DEBUG (5)、 INFO (4)、 WARN (3)、 ERROR (2)、 FATAL (1)、 NOFILTER(0)）。

参数

- **className** (*str*) – 要设置过滤级别的日志记录器名称。使用名称 Default 更改默认过滤级别。
- **level** (*int*) – 要设置的过滤级别。

> **警告：** 将 Default 日志级别设置得高于编译时提供的 MAX_LOG_LEVEL 级别不会产生任何效果。日志不会被打印，因为它们已在编译期间被移除。

yade._log.setOutputStream(*(str)*参数*1*，*(bool)*辐角*2*) → None

参数

- **streamName** (*str*) – 设置输出流，特殊名称 cout、 cerr、 clog使用 std::cout、 std::cerr、 std::clog 对应的std::clog （它是默认输出流）。其他名称均表示日志将写入参数中指定名称的文件。
- **reset** (*bool*) – 决定是否移除之前设置的所有输出流。设为 false 时：新的输出流会添加到当前输出流。

yade._log.setUseColors(*(bool)*辐角*1*) →None

开启/关闭日志消息中的颜色。默认开启。如果记录到文件，最好将其关闭。

yade._log.testAllLevels() → 无

此函数会打印全部日志级别的测试消息。可用于查看过滤的工作方式，以及日志会写入哪些流。

yade._log.testOnceLevels() →无

此函数使用 LOG_ONCE_* 宏系列打印全部日志级别的测试消息。

yade._log.testTimedLevels() → 无

此函数会在全部日志级别上打印定时测试消息。在此测试中，日志级别 [0...2] 每 2 秒定时打印一次，级别 [3,4] 每 1 秒打印一次，级别 [5,6] 每 0.5 秒打印一次。循环持续 2.1 秒。可用于查看定时过滤的工作方式，以及日志写入哪些流。

yade._log.unsetLevel(*(str)*辐角*1*) →无

参数 **className** (*str*) – 要取消设置过滤级别的日志记录器名称，取消后将改用 Default 。取消设置 Default 级别会将其更改为最大级别，并打印全部内容。

2.4.8 yade.math 模块

此 Python 模块向 Python 提供了适用于实数和复数类型的全部 C++ 数学函数。事实上，它在某种程度上重复了 import math、 import cmath 或 import mpmath。此外，它还便于将旧版 Python 脚本迁移到高精度计算。

此模块有以下用途：

1. 可靠地测试所有++ 任意精度实数和复数类型的 C 数学函数，并与 mpmath 进行对比。2. 作为迁移助手，用于处理调用了无法与 mpmath 良好配合的 Python 数学函数的 Python 脚本。例如，请参见下方的*math.linspace* 以及此合并请求。3. 允许以与 Yade 中的 C 数学代码相似的方式编写 Python 数学代码。另一个好处是，它比 mpmath 更快，因为 mpmath 是一个纯 Python 库（这也是编写lib/high-precision/ToFromPythonConverter.hpp时遇到的主要难题之一）。4. 测试 Eigen NumTraits。5. 测试 CGAL NumTraits。

如果需要其他 C++ 数学函数，应将其添加到以下文件中：

1. lib/high-precision/MathFunctions.hpp

2. py/高精度/_math.cpp3.
py/tests/testMath.py4.
py/tests/testMathHelper.py

如果其他 python math 函数与 mpmath 一起使用时运行不正常，则应在下方添加该函数，并让原本对该函数的调用改为调用此函数，例如，将 numpy.linspace(…) 替换为 yade.math.linspace(…)。

这些 RealHP<n> 更高精度的数学函数可以在 Python 中通过使用 .HPn 模块作用域进行访问。例如：

```
import yade.math as mth
mth.HP2.sqrt(2) # produces square root of 2 using RealHP<2> precision
mth.sqrt(2)     # without using HPn module scope it defaults to RealHP<1>
```

yade.math.Real(辐角)
: 此函数用于兼容如下调用： g = yade.math.toHP1("-9.81")。如果 Yade 使用默认的 Real 精度并将其设置为 double 进行编译，则 Python 不会接受作为数字的字符串参数。然而，在使用更高精度时，只有调用 yade.math.toHP1("1.23456789012345678901234567890123456789012345678901234567890") 不会截断为前 15 位小数。诸如 yade.math.toHP1(1.23456789012345678901234567890123456789012345678901234567890) 这样的调用将使用默认的 python double 转换，并将数字截断为前 15 位。

  如果你在调试高精度 Python 脚本时，难以找到发生此类截断的位置，那么对于模拟中具有物理重要性的所有 Python 浮点数，都应使用 yade.math.toHP1(string) 进行声明。如果即将发生错误的转换，此函数会抛出异常。

  另请参见高精度检查示例：checkGravityRungeKuttaCashKarp54.py。

yade.math.Real1(辐角)
: 此函数用于兼容如下调用： g = yade.math.toHP1("-9.81")。如果 Yade 使用默认的 Real 精度（设为 double）编译，那么 Python 不会将字符串参数识别为数字。不过，使用更高精度时，只有调用 yade.math.toHP1("1.23456789012345678901234567890123456789012345678901234567890") 不会截断到前 15 位小数。诸如 yade.math.toHP1(1.23456789012345678901234567890123456789012345678901234567890) 这样的调用会使用默认的 python double 转换，并将数字截断为前 15 位。

  如果你正在调试高精度 Python 脚本，并且难以找到发生此类精度截断的位置，应使用 yade.math.toHP1(string) 来声明模拟中所有在物理上重要的 Python 浮点数。若即将发生错误转换，此函数会抛出异常。

  另请参见高精度检查示例 checkGravityRungeKuttaCashKarp54.py。

yade.math.degrees(辐角)
: **返回** 使用 yade.math.Real 精度将弧度表示的辐角转换为度。

yade.math.degreesHP1(辐角)
: **返回** 使用 yade.math.Real 精度将弧度表示的辐角转换为度。

yade.math.getRealHPCppDigits10()
: **返回** 一个元组，其中包含 Eigen 和 CGAL 在 C++ 端支持的十进制数字数量。

yade.math.getRealHPPythonDigits10()
: **返回**一个元组，其中包含 yade.minieigenHP 在 Python 端支持的十进制数字数量。

yade.math.linspace(*a*, *b*, 数量)
: 此函数调用 numpy.linspace(…) 或 mpmath.linspace(…)，因为 numpy.linspace 函数无法与 mpmath 一起使用。

yade.math.needsMpmathAtN(*N*)

参数 N – 问题中的 int N 值。必须为 N >= 1。

返回 True 或 False，并提供信息，说明使用 mpmath 是否有助于避免处理 RealHP<N> 时损失精度。

yade.math.radians(辐角)
默认的 Python 函数 import math ; math.radians(arg) 仅支持 15 位精度 double。如果要以更高精度继续进行计算，建议改用此函数yade.math.radiansHP1(arg) 。它使用完整的 Yade Real精度数值。

注意：将来此函数可能会替换 radians(…) 函数。该函数在 Yade 的许多脚本中被调用，实际上调用的是原生 Python math.radians。我们只需找到最佳的向后兼容方案。函数 yade.math.radiansHP1(arg) 仍将作为使用原生 Yade Real 精度的函数保留。

yade.math.radiansHP1(辐角)
默认的 Python 函数 import math ; math.radians(arg) 仅支持 15 位精度。若要继续以更高精度进行计算，建议改用此函数yade.math.radiansHP1(arg) 。它使用 Yade 的完整精度数值。

注意：将来此函数可能会替换 radians(…) 函数。该函数在 Yade 的许多脚本中被调用，实际上调用的是 Python 原生的 math.radians。我们只需找到最佳的向后兼容方案。函数 yade.math.radiansHP1(arg) 将保留为使用 Yade 原生精度的函数。

yade.math.toHP1(辐角)
此函数用于兼容如下调用： g = yade.math.toHP1("-9.81")。如果 Yade 使用默认精度 Real 编译，且精度设为 double，那么 Python 不会接受字符串参数作为数字。不过，使用更高精度时，只有调用 yade.math.toHP1("1.23456789012345678901234567890123456789012345678901234567890") 不会截断到小数点后 15 位。诸如 yade.math.toHP1(1.23456789012345678901234567890123456789012345678901234567890) 这样的调用会使用默认转换，并将数值截断为前 15 位。
如果你正在调试高精度 Python 脚本，并且难以找到发生此类截断的位置，则应使用 yade.math.toHP1(string) 来声明模拟中物理上重要的所有 Python 浮点数。若即将发生不正确的转换，此函数会抛出异常。

另请参阅高精度检查示例 checkGravityRungeKuttaCashKarp54.py。

yade.math.usesHP()

返回 True 若 yade 使用的默认 Real 精度高于 15 位（53 位）double 类型。

yade._math.Catalan(*[(int)Precision=53]*) → 浮点数

返回 Real 向 卡塔兰常数，通过 Python 暴露，用于 测试 Eigen 数值特性的 特征。

yade._math.Euler(*[(int)Precision=53]*) → 浮点数

返回 Real 欧拉–马歇罗尼常数，供 Python 用于测试 Eigen 数值特性。

class yade._math.HP1

AddCost = 1
Catalan(*[(int)Precision=53]*) → 浮点数:

返回 Real 卡塔兰常数，供 Python 用于测试 Eigen 数值特性。

ComplexAddCost = 2

ComplexMulCost = 6

ComplexReadCost = 2

Euler([*(int)Precision=53*]) → 浮点数:

返回 Real 欧拉–马歇罗尼常数，供 Python 使用，用于测试Eigen 数值特征。

IsComplex = 0IsInteger = 0

IsSigned = 1

Log2([*(int)Precision=53*]) → 浮点数:

返回 Real 2 的自然对数，供 Python 使用，用于测试Eigen 数值特征。

MulCost = 1

Pi([*(int)Precision=53*]) → 浮点数:

返回 Real 用于 π 测试Eigen 数值特性的常量，已公开给 Python 使用。

ReadCost = 1

RequireInitialization = 0

class Var

Var 类用于测试任意精度 Real`cpl`一个 Complex 变量的 Python 转换器，以测试对该变量的读写。`val`一个 Real 用于测试的变量。abs(*(复数)x*) →浮点数:

返回 Real Complex 参数的绝对值。根据编译选项，封装 ::boost::multiprecision::abs(…) 或 std::abs(...) 函数。

abs((浮点数)x) ->浮点数：返回 Real Real 参数的绝对值。根据编译选项，封装 ::boost::multiprecision::abs(…) 或 std::abs(...) 函数。

acos(*(复数)x*) → 复数:

返回 Complex 参数的反余弦 Complex 值（以弧度表示）。根据编译选项，封装 ::boost::multiprecision::acos(…) 或 std::acos(...)函数。

acos((float)x) ->浮点数:

返回 Real 参数的反余弦。根据编译选项，wrap ::boost::multiprecision::acos(…) 或std::acos(...)函数。

acosh(*(complex)x*) →复数:

返回 Complex 参数的反双曲余弦 Complex （单位为弧度）。根据编译选项，wrap ::boost::multiprecision::acosh(…)或std::acosh(...)函数。

acosh((float)x) ->浮点数:

返回 Real 参数的反双曲余弦。根据编译选项，wraps ::boost::multiprecision::acosh(…) 或std::acosh(...)函数。

arg(*(复数)x*) →浮点数:

返回 Real 复数的辐角（相位角，单位为弧度）Complex 参数（单位为弧度）。根据编译选项，使用 wrap::boost::multiprecision::arg(…) 或 std::arg(...)函数。

asin(*(复数)x*) → 复数:

返回 Complex 参数的反正弦值 Complex （单位为弧度）。根据编译选项，使用 wrap ::boost::multiprecision::asin(…) 或 std::asin(...)函数。

asin((浮点数)x) -> 浮点数:

返回 Real 参数的反正弦。根据编译选项，wraps ::boost::multiprecision::asin(…) 或std::asin(...) 函数。

asinh(*(复数)x*) →复数:

返回 Complex 参数的反双曲正弦 Complex 弧度值。根据编译选项，wraps ::boost::multiprecision::asinh(…) 或std::asinh(...) 函数。

asinh((浮点数)x) ->浮点数:

返回 Real 参数的双曲反正弦。根据编译选项，wraps ::boost::multiprecision::asinh(…) 或std::asinh(...)函数。

atan(*(复数)x*) → 复数:

返回 Complex 以弧度表示的 Complex 参数的反正切。根据编译选项，封装 ::boost::multiprecision::atan(…) 或std::atan(...) 函数。

atan((float)x) -> 浮点数:

返回 Real 参数的反正切。根据编译选项，封装 ::boost::multiprecision::atan(…) 或std::atan(...) 函数。

atan2(*(float)x*, *(float)y*) →浮点数:

返回 Real 根据参数的符号计算 y/x 的反正切值 x 和 y以确定正确的象限。根据编译选项，封装:::boost::multiprecision::atan2(…) 或std::atan2(...) 函数。

atanh(*(complex)x*) →复数:

返回 Complex 参数的反双曲正切值，单位为 Complex 弧度。根据编译选项，wraps ::boost::multiprecision::atanh(…)或std::atanh(...) 函数。

atanh((float)x) ->浮点数:

返回 Real 参数的反双曲正切值。根据编译选项，wraps ::boost::multiprecision::atanh(…) 或 std::atanh(...)函数。

cbrt（浮点数）*x*）→浮点数:

返回 Real 参数的立方根。根据编译选项，会 wrap ::boost::multiprecision::cbrt(…) 或std::cbrt(...) 函数。

ceil(*(浮点数)x*) →浮点数:

返回 Real 计算不小于辐角的最小整数值。根据编译选项，包装 ::boost::multiprecision::ceil(…) 或std::ceil(...)函数。

conj(*(复数)x*) → 复数:

返回复共轭 Complex 参数。根据编译选项，包装 ::boost::multiprecision::conj(…) 或std::conj(...) 函数。

cos(*(复数)x*) →复数:

返回 Complex 以弧度表示的 Complex 参数的余弦。根据编译选项，封装 ::boost::multiprecision::cos(…) 或std::cos(...)函数。

cos((float)x) ->浮点数:

返回 Real 以弧度表示的 Real 参数的余弦。根据编译选项，封装 ::boost::multiprecision::cos(…) 或std::cos(...)函数。

cosh(*(complex)x*) → 复数:

返回 Complex 以弧度表示的 Complex 参数的双曲余弦。根据编译选项，封装 ::boost::multiprecision::cosh(…) 或std::cosh(...)函数。

cosh((float)x) ->浮点数:

返回 Real 的双曲余弦， Real 参数以弧度表示。根据编译选项，将调用 ::boost::multiprecision::cosh(…) 或std::cosh(...) 函数。

cylBesselJ(*(int)k*, *(float)x*) → 浮点数:

返回 Real 阶数为 k 且 Real 参数为 <https://www.boost.org/doc/libs/1_77_0/libs/math/doc/html/math_toolkit/bessel/bessel_first.html>‘__的第一类贝塞尔函数。参见:

defprec = 53

dummy_precision() → 浮点数:

返回与函数 epsilon 类似，但假设最后 10% 的位仅包含数值误差。Eigen 有时会在调用isEqualFuzzy 时使用此函数，以确定值之间是否存在较大差异，或是否大致接近。

epsilon(*[(int)Precision=53]*) → 浮点数:

返回 Real 返回 1.0 与 Real 类型的下一个可表示值之间的差值。封装了 std::numeric_limits<Real>::epsilon() 函数。

epsilon((float)x) ->浮点数:

返回 Real 返回 1.0 与 Real 类型的下一个可表示值之间的差值。封装了 std::numeric_limits<Real>::epsilon() 函数。

erf(*(float)x*) → 浮点数:

返回 Real 计算参数的 误差函数 。根据编译选项，wrap ::boost::multiprecision::erf(…) 或 std::erf(...) 函数。

`erfc`(*(浮点数)x*) →浮点数:

返回 `Real` 计算参数的互补误差函数，即 `1.0-erf(arg)`。根据编译选项，会包装:`::boost::multiprecision::erfc(…)` 或 std::erfc(...)函数。

`exp`(*(复数)x*) →复数:

返回底数e 的指数函数，计算 `Complex` 参数。根据编译选项，会包装 `::boost::multiprecision::exp(…)` 或 std::exp(...) 函数。

exp((浮点数)x) ->浮点数:

返回底数e 的指数函数，计算 `Real` 参数。根据编译选项，会包装 `::boost::multiprecision::exp(…)` 或 std::exp(...) 函数。

`exp2`(*(浮点数)x*) →浮点数:

返回以 *2* 为底的指数函数，作用于 `Real` 参数。根据编译选项，使用 `::boost::multiprecision::exp2(…)` 或std::exp2(...) 函数。

`expm1`(*(浮点数)x*) → 浮点数:

返回以 e 为底的指数函数，作用于 `Real` 参数后减去 `1.0`。根据编译选项，使用 `::boost::multiprecision::expm1(…)` 或std::expm1(...)函数。

`fabs`(*(浮点数)x*) →浮点数:

返回该 `Real` 参数的绝对值。根据编译选项，使用 `::boost::multiprecision::abs(…)` 或std::abs(...) 函数。

`factorial`(*(int)x*) → 浮点数:

返回 `Real` 参数的阶乘。 `Real` 参见： <https://www.boost.org/doc/libs/1_77_0/libs/math/doc/html/math_toolkit/factorials/sf_factorial.html>‘___

`floor`(*(float)x*) → 浮点数:

返回 `Real` 计算不大于辐角的最大整数值。根据编译选项，使用 wrap `::boost::multiprecision::floor(…)`或std::floor(...) 函数。

`fma`(*(float)x*, *(float)y*, *(float)z*) → 浮点数:

返回 `Real` - 计算 `(x*y) + z` ，如同使用无限精度计算，并且只舍入一次以适配结果类型。根据编译选项，使用:`::boost::multiprecision::fma(…)` 或std::fma(...)函数。

`fmod`(*(float)x*, *(float)y*) → 浮点数:

返回 `Real` 除法运算的浮点余数 `x/y`参数 `x` 和 `y` `::boost::multiprecision::fmod(…)` 或std::fmod(...) 函数，具体取决于编译选项。

`frexp`(*(float)x*) → 元组:

返回元组，包含 `(Real,int)`，将给定的浮点数 `Real` 参数分解为规格化小数和二的整数次幂。具体取决于编译选项，将调用 `::boost::multiprecision::frexp(…)` 或std::frexp(...) 函数。

`fromBits`(*(str)bits*[, *(int)exp=0*[, *(int)sign=1*]]) → float :

参数

- `bits` – `str` - 一个包含字符“0”和“1”的字符串。
- `exp` – `int` - 平移位的二进制指数。

- `sign` – `int` - 符号，应为 -1 或 +1，但不会进行检查。从位构造完成后，它会与结果相乘。

返回 `RealHP<N>` 由包含「0」、「1」位的字符串构造。此函数用于调试，速度较慢。

`getDecomposedReal`(*(浮点数)x*) →字典:

返回 `dict` - 字典，其中包含 DecomposedReal 类如何识别此类型的调试信息。此函数用于调试，速度较慢。其中包括 fpclassify 函数调用的结果、二进制表示以及其他有用信息。另请参阅 *fromBits*。

`getDemangledName`() → 字符串:

返回 `string` - `RealHP<N>`的 C++ 类型名称（已去修饰）。

`getDemangledNameComplex`() →字符串:

返回 `string` - `ComplexHP<N>` 的 C++ 类型名称的反混淆形式。

`getFloatDistanceULP`(*(浮点数)参数1*, *(浮点数)参数2*) → 浮点数:

返回存储在 `RealHP<N>` 中的整数值，ULP 距离由boost::math::float_distance计算得出，另请参阅浮点数比较以及Prof. Ka- han 关于此主题的论文。

警告：返回值是两个参数之间的**有向距离**，这意味着它可能为负数。

`getRawBits`(*(浮点数)x*) →字符串:

返回 `string` - 内存中表示此类型的原始位。请注意：它只检查系统字节序，并据此按相反顺序或原顺序打印字节。它不会进一步解析这些位所表示的符号、指数或有效数（在典型的 x86 处理器上，它们按此顺序打印），而且不同的处理器可能会以不同方式存储它们。对于内部使用指针的类型，此函数并不适用，因为它打印的是指针，而不是浮点数。此功能用于调试。

`hasInfinityNan = True`

`highest`(*[(int)Precision=53]*) → 浮点数:

返回 `Real` `Real` 类型所能表示的最大有限值。封装std::numeric_limits<实数>::max() 函数。

`hypot`(*(float)x*, *(float)y*) →浮点数:

返回 `Real` 的平方和的平方根 `x` 和 `y`，且在计算的中间阶段不会发生过度溢出或下溢。根据编译选项，封装 `::boost::multiprecision::hypot(…)`或std::hypot(...) 函数。

`ilogb`(*(float)x*) → 浮点数:

返回 `Real` 从浮点参数辅读取无偏指数的值，并将其作为有符号整数值返回。根据编译选项，封装为 `::boost::multiprecision::ilogb(…)` 或std::ilogb(...) 函数。

`imag`(*(complex)x*) → 浮点数:

返回复数的 imag 部分 `Complex` 参数。根据编译选项，封装为 `::boost::multiprecision::imag(…)` 或std::imag(...) 函数。

`isApprox`(*(浮点数)a*, *(浮点数)b*, *(浮点数)eps*) → 布尔值:

返回 bool，当 a 与 b 在给定的 eps下近似相等时返回 True，另请参阅 此处

isApproxOrLessThan(*(浮点数)a*，*(浮点数)b*，*(浮点数)eps*）→ 布尔值:

返回 bool，当 a 按照提供的 b 近似小于或等于时返回 True eps，另请参阅 此处

isEqualFuzzy(*(float)参数1*，*(float)参数2*，*(float)辐角3*）→ 布尔值:

返回 bool， True 如果两个数的绝对差小于std::numeric_limits<实数>::epsilon()

isMuchSmallerThan(*(float)a*，*(float)b*，*(float)eps*）→ 布尔值:

返回 bool，当 a 按照提供的 b 小于 eps时返回 True；另请参阅 此处

isfinite(*(float)x*）→ 布尔值:

返回 bool，用于指示 Real 参数是否为无穷。根据编译选项，封装 ::boost::multiprecision::isfinite(…) 或 std::isfinite(...) 函数。

isinf(*(float)x*）→ 布尔值:

返回 bool，用于指示 Real 参数是否为无穷。根据编译选项，封装 ::boost::multiprecision::isinf(…) 或 std::isinf(...) 函数。

isnan(*(float)x*）→ 布尔值:

返回 bool，用于指示 Real 参数是否为非数。根据编译选项封装 ::boost::multiprecision::isnan(…) 或 std::isnan(...) 函数。

laguerre(*(int)n*，*(int)m*，*(float)x*）→ 浮点数:

返回 Real 阶数为 n、 m，自变量为 Real 的拉盖尔多项式。参见：< https://www.boost.org/doc/libs/1_77_0/libs/math/doc/html/math_toolkit/sf_poly/laguerre.html>‘___

ldexp(*(float)x*，*(int)y*）→浮点数:

返回 将浮点值 x 乘以 2 的 exp 次幂。根据编译选项，使用 wrap ::boost::multiprecision::ldexp(…)或std::ldexp(...) 函数。

lgamma(*(float)x*）→ 浮点数:

返回 Real 计算自变量 arg 的伽马函数的绝对值的自然对数。根据编译选项，使用伽马函数的编译选项 wraps: ::boost::multiprecision::lgamma(…) 或std::lgamma(...) 函数。

log(*(complex)x*）→复数:

返回 Complex 自然（底数为e）对数，计算复数值 z 的自然对数，支割沿负实轴。根据编译选项，使用: ::boost::multiprecision::log(…)或std::log(...) 函数。

log((float)x) ->浮点数:

返回 Real 自然（底数为e）对数，计算实数值的自然对数。根据编译选项，使用 ::boost::multiprecision::log(…) 或std::log(...) 函数。

log10(*(complex)x*）→复数:

返回 Complex （底数为*10*）对数，计算复数值 z 的对数，支割沿负实轴。根据编译选项，使用: ::boost::multiprecision::log10(…) 或std::log10(...) 函数。

log10((float)x) -> 浮点数:

返回实 Real 数值的十进制（底数为10）对数。根据编译选项，返回 ::boost::multiprecision::log10(…) 或std::log10(...)函数。

log1p(*(float)x*) → 浮点数:

返回 Real 自然（底数为e）对数 1+argument。根据编译选项，会封装 ::boost::multiprecision::log1p(…) 或std::log1p(...)函数。

log2(*(float)x*) →浮点数:

返回 Real 二进制（底数为2）实数值的对数。根据编译选项，会封装 ::boost::multiprecision::log2(…) 或std::log2(...) 函数。

logb(*(float)x*) →浮点数:

返回 从浮点参数 arg 中提取无偏基数无关指数的值，并将其作为浮点值返回。根据编译选项，wrap ::boost::multiprecision::logb(…) 或std::logb(...) 函数。

lowest(*[(int)Precision=53]*) → 浮点数:

返回 Real 返回 Real 类型的最小（负）有限值。封装为std::numeric_limits<实数>::lowest() 函数。

max(*(浮点数)x*, *(浮点数)y*) → 浮点数:

返回 Real 两个参数中的较大值。根据编译选项，封装为 ::boost::multiprecision::max(…) 或 std::max(...)函数。

max_exp2 = 1024

min(*(浮点数)x*, *(浮点数)y*) → 浮点数:

返回 Real 两个参数中较小的一个。根据编译选项，封装 ::boost::multiprecision::min(…) 或std::min(...) 函数。

modf(*(float)x*) →元组:

返回由 (Real,Real)组成的元组，将给定的浮点数 Real 分解为整数部分和小数部分，两者的类型和符号均与 x 相同。根据编译选项，封装 ::boost::multiprecision::modf(…) 或std::modf(...)函数。

polar(*(float)x*, *(float)y*) →复数:

返回 Complex 由极坐标分量表示的 Real ρ（长度）、 Real θ（角度）参数对应的极坐标形式（极坐标分量构成的复数），单位为弧度。根据编译选项，封装 ::boost::multiprecision::polar(…) 或std::polar(...) 函数。

pow(*(complex)x*, *(complex)pow*) → 复数:

返回 Complex 复数参数1的辐角，其幂为 Complex 参数2。根据编译选项，封装 ::boost::multiprecision::pow(…) 或std::pow(...)函数。

pow((float)x, (float)y) -> 浮点数:

返回 Real base 的值，其幂为 exp。根据编译选项，封装 ::boost::multiprecision::pow(…) 或std::pow(...)函数。

proj(*(复数)x*) → 复数:

返回 Complex 将复数投影到黎曼球面上的投影函数 Complex 辐角，单位为弧度。根据编译选项，调用 ::boost::multiprecision::proj(…) 或std::proj(...) 函数。

random() → 浮点数:

返回 Real (-1,1)区间内的对称随机数。供 Eigen 使用。

random((浮点数)a, (浮点数)b) -> 浮点数:

返回 Real (a,b)区间内的随机数。供 Eigen 使用。

real(*(复数)x*) → 浮点数:

返回实部： Complex 参数。根据编译选项，包装为 ::boost::multiprecision::real(…) 或std::real(...) 函数。

remainder(*(float)x*, *(float)y*) → 浮点数:

返回 Real 浮点除法运算的 IEEE 余数 x/y。根据编译选项，包装为 ::boost::multiprecision::remainder(…) 或std::remainder(...) 函数。

remquo(*(float)x*, *(float)y*) → 元组:

返回元组 (Real,long)，即除法运算的浮点余数 x/y ，与 std::remainder() 函数的行为相同。此外，还会返回 x/y 的符号及其末尾至少三位的信息，足以确定结果在一个周期内所属的卦限。根据编译选项，包装为 ::boost::multiprecision::remquo(…) 或std::remquo(...) 函数。

rint(*(float)x*) →浮点数:

返回使用当前舍入模式，将浮点参数辐角舍入为整数值（以浮点格式表示）。根据编译选项，封装为 ::boost::multiprecision::rint(…) 或std::rint(...) 函数。

round(*(float)x*) → 浮点数:

返回 Real 最接近辐角的整数值（以浮点格式表示）；中点情况向远离零的方向舍入，不受当前舍入模式影响。根据编译选项，封装为 ::boost::multiprecision::round(…)或std::round(...) 函数。

roundTrip(*(float)x*) →浮点数:

返回 Real 返回参数 x。可用于将类型转换为原生 RealHP<N> 精度。

sgn(*(float)x*) → 整数:

返回 int参数的符号： -1、0 或 1。

sign(*(浮点数)x*) → 整数:

返回 int参数的符号： -1、0 或 1。

sin(*(复数)x*) →复数:

返回 Complex 参数的正弦值 Complex （单位为弧度）。根据编译选项，封装为 ::boost::multiprecision::sin(…) 或std::sin(...) 函数。

sin((浮点数)x) ->浮点数:

返回 Real 的正弦值，其中 Real 参数以弧度表示。根据编译选项，使用 wrap ::boost::multiprecision::sin(…) 或std::sin(...) 函数。

sinh(*(复数)x*) → 复数:

返回 Complex 以弧度表示的 Complex 参数的双曲正弦。根据编译选项，wrap ::boost::multiprecision::sinh(…) 或std::sinh(...) 函数。

sinh((float)x) -> 浮点数：

返回 Real 以弧度表示的参数的双曲正弦。 Real 根据编译选项，包装 ::boost::multiprecision::sinh(…) 或std::sinh(...) 函数。

smallest_positive() → 浮点数:

返回 Real 最小正数。包装std::numeric_limits<实数>::min()

sphericalHarmonic(*(int)l*, *(int)m*, *(float)θ*, *(float)phi*) → 复数:

返回 Real 阶数为l （无符号整数）、m （有符号整数）以及 Real 参数 theta 和 phi的球谐多项式。参见：<https://www.boost.org/doc/libs/1_77_0/libs/math/doc/html/math_toolkit/sf_poly/sph_harm.html>‘__

sqrt(*(复数)x*) → 复数:

返回 Complex 的平方根 Complex 参数。根据编译选项，使用 ::boost::multiprecision::sqrt(…) 或std::sqrt(...) 函数。

sqrt((浮点数)x) ->浮点数：返回 Real 参数的平方根。根据编译选项，使用 ::boost::multiprecision::sqrt(…) 或std::sqrt(...) 函数。

squaredNorm(*(复数)x*) →浮点数:

返回 Real 参数的范数（模平方） Complex （以弧度表示）。根据编译选项，使用 ::boost::multiprecision::norm(…)或std::norm(...) 函数。

tan(*(复数)x*) →复数:

返回 Complex 参数的正切值（以 Complex 弧度为单位）。根据编译选项，使用 ::boost::multiprecision::tan(…) 或std::tan(...)函数。

tan((浮点数)x) ->浮点数：

返回 Real 参数的正切值（以 Real 弧度为单位）。根据编译选项，使用 ::boost::multiprecision::tan(…) 或std::tan(...) 函数。

tanh(*(复数)x*) → 复数:

返回 Complex 参数的双曲正切值（以 Complex 弧度为单位）。根据编译选项，使用 ::boost::multiprecision::tanh(…) 或std::tanh(...) 函数。

tanh((float)x) -> 浮点数：

返回 Real 参数的双曲正切值（以 Real 弧度为单位）。根据编译选项，wrap ::boost::multiprecision::tanh(…)或std::tanh(...) 函数。

testArray() → 无:

T此函数会调用 std::vector::data(…) 函数，以提取数组。

testCgalNumTraits = False

testConstants() → 无:
此函数测试 lib/high-precision/Constants.hpp 中的 yade::math::ConstantsHP<N>，它们是早期 yade::Mathr 常数。

tgamma(*(浮点数)x*) → 浮点数:

返回 Real 计算辐角的伽马函数。根据编译选项，封装 ::boost::multiprecision::tgamma(…) 或std::tgamma(...) 函数。

toDouble(*(浮点数)x*) → 浮点数:

返回 float 将 Real 类型转换为 double，并返回原生 Python float。

toHP1(*(浮点数)x*) → 浮点数:

返回 RealHP<1> 将参数 RealHP<1> 转换后的结果，类型为 static_cast<RealHP<1>>(arg)。

toInt(*(浮点数)x*) → 整数:

返回 int 将 Real 类型转换为 int，并返回原生 Python int。

toLong(*(浮点数)x*) → 整数:

返回 int 将 Real 类型转换为 long int，并返回一个原生 Python int。

toLongDouble(*(float)x*) →浮点数:

返回 float 将 Real 类型转换为 long double，并返回一个原生 Pythonfloat。

trunc(*(float)x*) → 浮点数:

返回 Real 不大于辐角模的最近整数。根据编译选项，会 wrap ::boost::multiprecision::trunc(…) 或std::trunc(...) 函数。

yade._math.Log2(*[(int)Precision=53]*) → 浮点数

返回 Real 2 的自然对数，向 Python 暴露，用于 测试 Eigen 数值特性。

yade._math.Pi(*[(int)Precision=53]*) → 浮点数

返回 Real 该 π 常量，向 Python 暴露，用于 测试 Eigen 数值特性。

class yade._math.RealHPConfig
RealHPConfig 类提供有关 RealHP<N> 类型的信息。

变量

- ***extraStringDigits10*** – 此静态变量允许控制转换为十进制字符串时使用的额外位数。为其赋予不同的值，可影响 C++ Python 转换 以及 所有其他转换。请注意，值小于 3 可能导致往返转换测试失败。

- ***isFloat128Broken*** – 如果 Yade 是使用 g++版本 $< 9.2.1$ 编译的，则会提供运行时信息，因此 boost::multiprecision::float128 无法运行。
- ***isEnabledRealHP*** – 提供运行时信息 RealHP<N> 适用于大于 1 的 N。

- ***workaroundSlowBoostBinFloat*** – boost::multiprecision::cpp_bin_float存在一个问题：导出以下函数时，在 Python 中导入它的速度非常慢：erf、erfc、lgamma、tgamma。在这种情况下，Python import yade.math 可能需要一分钟以上。解决方法是使这些函数不可用

以便在 Python 中支持更大的 N 值。请参阅 _math.cpp中对 IfConstexprForSlowFunctions 的调用。此变量包含这些函数可用时的最大 N 值。如果不使用 boost::multiprecision::cpp_bin_float，该值等于最大 N 值。

extraStringDigits10 = 4getDigits10(*(int)N*) → int：这是一个 yade.math.RealHPConfig 诊断函数。**参数** N – int - N 在 RealHP<N> 中的值。

返回 表示 int 的 std::numeric_limits<RealHP<N>>::digits10

getDigits2((int)N) → int：这是一个 yade.math.RealHPConfig 诊断函数。参数 N – int - N 在 RealHP<N>中的值。返回表示 int std::numeric_limits<RealHP<N>>::digits的，该值对应于此类型使用的有效数位数量。getSupportedByEigenCgal() →元组：返回包含 N 的 tuple，这些 N 来自 Eigen 和 CGAL 支持的精度 RealHP<N> getSupportedByMinieigen() →元组：返回包含 N 的 tuple，这些 N 来自 minieigenHP 支持的精度 RealHP<N> isEnabledRealHP = FalseisFloat128Broken = FalseisFloat128Present = False workaroundSlowBoostBinFloat = 1class yade._math.Var使用 Var 类来测试任意精度 Real的 Python 转换器。cpl用于测试从该变量读取和向其写入的变量 Complex 。val用于测试的变量 Real。(complex)xyade._math.abs(float 返回 Real Complex 参数的）→绝对值。根据编译选项，封装 ::boost::multiprecision::abs(…) 或 std::abs(...) 函数。abs((float)x) → float：返回 Real ::boost::multiprecision::abs(…) 参数的 Real 绝对值。根据编译选项，封装 yade._math.acos(或 std::abs(...) 函数。）→(complex)xcomplex

返回 Complex 参数的反余弦值，单位为弧度。根据编译选项，wrap ::boost::multiprecision::acos(…) 或std::acos(...) 函数。

acos((float)x) → **浮点数：返回** Real 参数的反余弦。根据编译选项，wrap ::boost::multiprecision::acos(…) 或std::acos(...) 函数。

yade._math.acosh(*(complex)x*) → 复数

返回 Complex 的反双曲余弦Complex 参数（以弧度表示）。根据编译选项，包装::boost::multiprecision::acosh(…) 或std::acosh(...) 函数。

反双曲余弦((浮点数)x) → 浮点数:

返回 Real 参数的反双曲余弦。根据编译选项，包装 ::boost::multiprecision::acosh(…) 或std::acosh(...) 函数。

yade._math.arg(*(复数)x*) →浮点数

返回 Real 的辐角（复数的相位角，以弧度表示） Complex参数（以弧度表示）。根据编译选项，包装::boost::multiprecision::arg(…) 或std::arg(...) 函数。

yade._math.asin(*(复数)x*) →复数

返回 Complex 参数的反正弦值（以弧度表示）。根据编译选项，封装 Complex 或std::asin(...) 函数。

asin((浮点数)x) → 浮点数:

返回 Real 参数的反正弦值。根据编译选项，封装 ::boost::multiprecision::asin(…) 或std::asin(...) 函数。

yade._math.asinh(*(复数)x*) → 复数

返回 Complex 参数的反双曲正弦值Complex 参数以弧度表示。根据编译选项，封装::boost::multiprecision::asinh(…) 或std::asinh(...) 函数。

asinh((float)x) → 浮点数:

返回 Real 参数的反双曲正弦值。根据编译选项，wrap ::boost::multiprecision::asinh(…) 或std::asinh(...) 函数。

yade._math.atan(*(complex)x*) →复数

返回 Complex 参数的反正切值，单位为弧度。根据编译选项，wrap ::boost::multiprecision::atan(…) 或std::atan(...) 函数。

atan((float)x) → 浮点数:

返回 Real 参数的反正切值。根据编译选项，wrap ::boost::multiprecision::atan(…) 或std::atan(...) 函数。

yade._math.atan2(*(浮点数)x*, *(浮点数)y*) → 浮点数

返回 Real 根据参数的符号计算 y/x 的反正切值 x 和 y以确定正确象限。根据编译选项，这里会封装::boost::multiprecision::atan2(…) 或std::atan2(...) 函数。

yade._math.atanh(*(复数)x*) → 复数

返回 Complex 的反双曲正切值Complex 以弧度表示的参数。根据编译选项，这里会封装::boost::multiprecision::atanh(…) 或std::atanh(...) 函数。

atanh((float)x) → 浮点数:

返回 Real 参数的反双曲正切。根据编译选项，可能会包装为 ::boost::multiprecision::atanh(…) 或std::atanh(...)函数。

yade._math.cbrt(*(浮点数)x*) →浮点数

返回 Real 参数的立方根。根据编译选项，包装::boost::multiprecision::cbrt(…) 或 std::cbrt(...)函数。

yade._math.ceil(*(复数)x*) →复数

返回 Real 计算不小于辐角的最小整数值。根据编译选项，包装 ::boost::multiprecision::ceil(…) 或 std::ceil(...)函数。

yade._math.conj(*(复数)x*) →复数

返回复共轭 Complex 参数。根据编译选项，使用 ::boost::multiprecision::conj(…) 或std::conj(...) 函数。

yade._math.cos(*(复数)x*) →复数

返回 Complex 以下对象的余弦值： Complex 以弧度表示的参数。根据编译选项，使用 ::boost::multiprecision::cos(…) 或std::cos(...) 函数。

cos((浮点数)x) → 浮点数:

返回 Real 以下对象的余弦值： Real 以弧度表示的参数。根据编译选项，使用 ::boost::multiprecision::cos(…) 或 std::cos(...) 函数。

yade._math.cosh(*(复数)x*) →复数

返回 Complex 参数Complex （以弧度为单位）的双曲余弦值。根据编译选项，wrap::boost::multiprecision::cosh(…) 或std::cosh(...) 函数。

cosh((float)x) → 浮点数:

返回 Real 参数 Real （以弧度为单位）的双曲余弦值。根据编译选项，使用 wrap ::boost::multiprecision::cosh(…) 或std::cosh(...)函数。

yade._math.cylBesselJ(*(int)k*,*(float)x*) → 浮点数

返回 Real 阶数 k 和 Real参数对应的第一类贝塞尔函数。参见： < https://www.boost.org/doc/libs/1_77_0/libs/math/doc/html/math_toolkit/bessel/bessel_first.html>‘__

yade._math.dummy_precision() → 浮点数

返回值 与函数 epsilon类似，但假设最后 10 位仅包含数值误差。Eigen 在调用 isEqualFuzzy时有时会使用此函数，以判断值之间的差异是否很大，或者是否大致接近。

yade._math.epsilon(*[(int)Precision=53]*) → 浮点数

返回 Real 返回 1.0 与 Real 类型的下一个可表示值之间的差集。封装 std::numeric_limits<Real>::epsilon() 函数。

epsilon((float)x) → 浮点数:

返回 Real 表示 1.0 与 Real类型的下一个可表示值之间的差集。封装了 std::numeric_limits<实数>::epsilon() 函数。

yade._math.erf(*(float)x*) →浮点数

返回 Real 计算参数的 误差函数 。根据编译选项，封装 ::boost::multiprecision::erf(…) 或std::erf(...) 函数。

yade._math.erfc(*(float)x*) →浮点数

返回 Real 计算参数的 互补误差函数 ，即1.0-erf(arg)。根据编译选项，封装 ::boost::multiprecision::erfc(…) 或 std::erfc(...)函数。

yade._math.exp(*(complex)x*) →复数

返回 底数为 *e* 的 指数函数，其参数为 Complex 。根据编译选项，封装 ::boost::multiprecision::exp(…) 或 std::exp(...)函数。

指数函数((浮点数)x) → 浮点数:

返回底数为e 的指数函数，作用于 Real 参数。根据编译选项，封装 ::boost::multiprecision::exp(…) 或std::exp(...)函数。

yade._math.exp2(*(浮点数)x*) →浮点数

返回底数为2 的指数函数，作用于 Real 参数。根据编译选项，封装 ::boost::multiprecision::exp2(…) 或std::exp2(...)函数。

yade._math.expm1(*(浮点数)x*) →浮点数

返回底数为e 的指数函数，作用于 Real 参数减去1.0。根据编译选项，封装 ::boost::multiprecision::expm1(…) 或std::expm1(...)函数。

yade._math.fabs(*(float)x*) →浮点数

返回参数的 Real 绝对值。根据编译选项，使用 wrap ::boost::multiprecision::abs(…) 或std::abs(...) 函数。

yade._math.factorial(*(int)x*) →浮点数

返回 Real 参数的 Real 阶乘。参见： <https://www.boost.org/doc/libs/1_77_0/libs/math/doc/html/math_toolkit/factorials/sf_factorial.html>‘___

yade._math.floor(*(float)x*) →浮点数

返回 Real 计算不大于辐角的最大整数值。根据编译选项，使用 wrap ::boost::multiprecision::floor(…) 或std::floor(...)函数。

yade._math.fma(*(float)x*, *(float)y*, *(float)z*) → 浮点数

返回 Real - 计算 (x*y) + z 时采用无限精度计算，并且只舍入一次以适配结果类型。根据编译选项，会封装: ::boost::multiprecision::fma(…) 或std::fma(...) 函数。

yade._math.fmod(*(float)x*, *(float)y*) → 浮点数

返回 Real 参数 x/y的浮点余数，即除法运算 x 和 y的结果。根据编译选项，会封装: ::boost::multiprecision::fmod(…)或 std::fmod(...) 函数。

yade._math.frexp(*(float)x*) →元组

返回 包含 (Real,int)的元组，将给定的浮点数 Real 参数分解为规格化小数和二的整数次幂。根据编译选项，会封装 ::boost::multiprecision::frexp(…) 或 std::frexp(...)函数。

yade._math.fromBits((字符串)位[,(整数)指数=0[,(整数)符号=1]]) →浮点数

参数

- **bits** – str - 仅包含'0'和'1'字符的字符串。
- **exp** – int - 二进制指数，用于平移位。
- **sign** – int - 符号，应为 -1 或 +1，但不会对此进行检查。根据位构造完成后，它会乘以结果。

返回 RealHP<N> 由包含'0'和'1'位的字符串构造。此功能用于调试，速度相当慢。

yade._math.getDecomposedReal(*(浮点数)x*) →字典

返回 dict - 一个字典，其中包含 DecomposedReal 类如何看待此类型的调试信息。此信息用于调试，速度较慢。包含 fpclassify 函数调用的结果、二进制表示以及其他有用信息。另请参阅*fromBits*。

yade._math.getDemangledName() →字符串

返回 string - RealHP<N> 的 C++ 类型名称（已反修饰）。

yade._math.getDemangledNameComplex() → 字符串

返回 string - ComplexHP<N> 的 C++ 类型名称（已反修饰）。

yade._math.getEigenFlags() →字典

返回列出所有类型标志的 Python 字典，参见：https://eigen.tuxfamily.org/dox/group__flags.html

yade._math.getEigenStorageOrders() →字典

返回列出所有类型选项的 Python 字典，参见：https://eigen.tuxfamily.org/dox/group__TopicStorageOrders.html

yade._math.getFloatDistanceULP(*(浮点数)参数1*, *(浮点数)参数2*) → 浮点数

返回存储在 RealHP<N>中的整数值，ULP 距离由boost::math::float_distance计算得出，另请参见浮点数比较以及Kahan 教授关于此主题的论文。

返回值是两个参数之间的**有向距离**，这意味着它可能为负值。

yade._math.getRawBits(（浮点数）*x*) →字符串

返回 string ——表示该类型在内存中的原始位。请注意：它只检查系统字节序，并据此按相反顺序打印字节，或者保持原顺序打印。它不会进一步尝试解释位中的符号、指数或有效数（在典型的 x86 处理器上，它们按此顺序打印），而不同的处理器存储这些内容的方式可能不同。对于内部使用指针的类型，此函数并不适用，因为它打印的是指针，而不是浮点数。此函数用于调试。

yade._math.getRealHPErrors(*(list)testLevelsHP*[, *(int)testCount=10*[, *(float)minX=-10.0*[, *(float)maxX=10.0*[, *(bool)useRandomArgs=False*[, *(int)printEveryNth=1000*[, *(bool)collectArgs=False*[, *(bool)extraChecks=False*]]]]]]]) → dict

T 使用最高精度的 n 个参数测试数学函数 testLevelsHPi，并返回 找到的最大 ULP 距离，使用 *getFloatDistanceULP*。执行 testCount 次随机尝试，函数参数的取值范围为 minX ... maxX，针对 RealHP<N> 类型进行；其中 N 来自 testLevelsHP 参数提供的列表。

参数

- **testLevelsHP** – 一个由高精度级别 `N` （以RealHP<N>为单位）的 `int` 值组成的 `list` ，应针对这些值执行测试。至少必须包含两个元素，以便有一种更高精度的类型可用于执行测试。
- **testCount** – `int` - 指定要对每个函数执行多少次随机测试。
- **minX** – `Real` - 随机参数的生成范围起点。
- **maxX**– `Real` - 该范围的终点。
- **useRandomArgs** – 如果 `False` （默认），则将 `minX ... maxX` 划分为 `testCount` 个等距点。如果为 `True` ，则每次调用都会使用一个随机数。这仅适用于函数的第一个参数；如果函数有多个参数，其余参数将随机生成——不会执行二维扫描。
- **printEveryNth** – 将 使用 `LOG_INFO` 每隔 N 步在 `testCount` 循环中打印进度信息。例如，要查看这些信息，请先使用 `yade -f6`，另请参阅 日志记录器文档。
- **collectArgs** – 如果 `True` ，则返回结果中会有更长的参数列表，这些参数会产生错误结果。
- **extraChecks** – 执行此函数时将进行额外检查。仅在调试 *getRealHPErrors*时有用。

返回 一个 Python 字典，其中包含与正确函数值之间 ULP 距离最大的值。对于每个函数名，都有一个字典，其中包含：被测试的 `RealHP<N>` 类型的二进制位数、该函数的最差参数，以及与参考值之间的 ULP 距离。

返回的 ULP 误差是绝对值，与 *getFloatDistanceULP* 不同，后者带有符号。

`yade._math.highest`(`[`*(int)Precision=53*`]`) → 浮点数

返回 `Real` 返回 `Real` 类型的最大有限值。封装 std::numeric_-limits<Real>::max() 函数。

`yade._math.hypot`(*(浮点数)x*, *(浮点数)y*) → 浮点数

返回 `Real` 和 `x` 与 `y`的平方和的平方根，在计算的中间阶段不会发生过大的溢出或下溢。根据编译选项，会包装 `::boost::multiprecision::hypot(…)` 或 std::hypot(...)函数。

`yade._math.ilogb`(*(浮点数)x*) →浮点数

返回 `Real` 从浮点参数 arg 中提取无偏指数的值，并以有符号整数值的形式返回。根据编译选项，会包装 `::boost::multiprecision::ilogb(…)` 或std::ilogb(...) 函数。

`yade._math.imag`(*(复数)x*) →浮点数

返回复数的虚部 `Complex` 参数。根据编译选项，会包装 `::boost::multiprecision::imag(…)` 或std::imag(...) 函数。

`yade._math.isApprox`(*(浮点数)a*, *(浮点数)b*, *(浮点数)eps*) → 布尔值

返回`bool`；当 `a` 与 `b` 在给定 `eps`下近似相等时为 True，另请参见此处

`yade._math.isApproxOrLessThan`(*(浮点数)a*, *(浮点数)b*, *(浮点数)eps*) → 布尔值

返回 `bool`；当 `a` 小于或近似等于 `b` 在给定 `eps`下时为 True，另请参见此处

`yade._math.isEqualFuzzy`(*(浮点数)参数1*, *(浮点数)参数2*, *(浮点数)辐角3*) → 布尔值

如果两个数的绝对差小于， `bool` True 则返回std::numeric_limits<实数>::epsilon()

`yade._math.isMuchSmallerThan`(*(浮点数)a*, *(浮点数)b*, *(浮点数)eps*) → 布尔值

返回 `bool`，当 `a` 小于 `b` 且提供 `eps`时为 True，另请参阅此处

`yade._math.isThisSystemLittleEndian`() → 布尔值

如果此系统采用小端架构，则返回 `True` ，否则不返回。

`yade._math.isfinite`(*(浮点数)x*) →布尔值

返回 `bool` 用于指示 `Real` 参数是否为无穷。根据编译选项，会封装 `::boost::multiprecision::isfinite(…)` 或 std::isfinite(...)函数。

`yade._math.isinf`(*(float)x*) →布尔值

返回 `bool` 用于指示 `Real` 参数是否为无穷。根据编译选项，会封装 `::boost::multiprecision::isinf(…)` 或std::isinf(...) 函数。

`yade._math.isnan`(*(float)x*) →布尔值

返回 `bool` 用于指示 `Real` 参数是否为非数。根据编译选项，会封装 `::boost::multiprecision::isnan(…)` 或std::isnan(...) 函数。

`yade._math.laguerre`(*(int)n*, *(int)m*, *(float)x*) → 浮点数

返回 `Real` 阶数为 `n`、 `m` 参数为 `Real` 的拉盖尔多项式。参见：< https://www.boost.org/doc/libs/1_77_0/libs/math/doc/html/math_toolkit/sf_poly/laguerre.html>‘__

`yade._math.ldexp`(*(float)x*, *(int)y*) → 浮点数

返回 将一个浮点值 `x` 乘以 2 的 `exp` 次幂。根据编译选项，会封装 `::boost::multiprecision::ldexp(…)` 或std::ldexp(...) 函数。

`yade._math.lgamma`(*(float)x*) → 浮点数

返回 `Real` 计算参数的伽马函数绝对值的自然对数 。根据编译选项，会封装 `::boost::multiprecision::lgamma(…)` 或std::lgamma(...) 函数。

`yade._math.log`(*(complex)x*) →复数

返回 `Complex` 复数值 `z` 的自然（底数为 e）对数，其支割沿负实轴。根据编译选项，会包装 `::boost::multiprecision::log(…)` 或std::log(...) 函数。

log((float)x) → 浮点数：返回 `Real` 实数值的自然（底数为 e）对数。根据编译选项，会包装 `::boost::multiprecision::log(…)` 或 std::log(...) 函数。

`yade._math.log10`(*(complex)x*) → 复数

返回 `Complex` (底数为 *10*）复数值 `z` 的对数，其支割沿负实轴。根据编译选项，会包装: `:boost::multiprecision::log10(…)` 或std::log10(...) 函数。

log10((float)x) → 浮点数：

返回 `Real` 常用对数（底数为 10）。实数值的对数。根据编译选项，会包装 `::boost::multiprecision::log10(…)` 或 std::log10(...)函数。

`yade._math.log1p`(*(浮点数)x*) →浮点数

返回 底数 Real 为自然对数（底数为 e） 1+argument。根据编译选项，调用 ::boost::multiprecision::log1p(…) 或 std::log1p(...)函数。

yade._math.log2((浮点数)*x*）→浮点数

返回实数值的 Real 二进制（底数为2）对数。根据编译选项，封装 ::boost::multiprecision::log2(…) 或std::log2(...) 函数。

yade._math.logb((浮点数)*x*）→浮点数

返回从浮点参数辐角中提取无偏基数无关指数的值，并以浮点值形式返回。根据编译选项，封装 ::boost::multiprecision::logb(…) 或std::logb(...)函数。

yade._math.**lowest**(⌈*(int)Precision=53*⌉) → 浮点数

返回值 Real 是 Real 类型的最小（负）有限值。封装为std::numeric_limits<实数>::lowest() 函数。

yade._math.**max**(*(float)x*, *(float)y*) → 浮点数

返回 Real 两个参数中的较大值。根据编译选项，封装为 ::boost::multiprecision::max(…) 或std::max(...) 函数。

yade._math.**min**(*(float)x*, *(float)y*) → 浮点数

返回 Real 两个参数中的较小值。根据编译选项，封装为 ::boost::multiprecision::min(…) 或std::min(...) 函数。

yade._math.**modf**(*(float)x*）→元组

返回由 (Real,Real)组成的元组，将给定的浮点数 Real 分解为整数部分和小数部分，两者的类型和符号均与 x 相同。根据编译选项，封装 ::boost::multiprecision::modf(…) 或 std::modf(...) 函数。

yade._math.**polar**(*(float)x*, *(float)y*) → 复数

返回 Complex 由极坐标分量构成的 Real ρ（长度）、Real θ（角度）参数所表示的极坐标形式复数，单位为弧度。根据编译选项，封装:::boost::multiprecision::polar(…) 或 std::polar(...)函数。

yade._math.**pow**(*(complex)x*, *(complex)pow*) → 复数

返回 Complex 复数参数1 的 Complex 参数2 次幂。根据编译选项，封装 ::boost::multiprecision::pow(…) 或std::pow(...) 函数。

pow((float)x, (float)y) → 浮点数 :

返回 Real 的值 base 的幂 exp。根据编译选项，使用 ::boost::multiprecision::pow(…) 或std::pow(...) 函数。

yade._math.**proj**((复数)*x*）→复数

返回 Complex 的投影函数（复数在黎曼球面上的投影），即 Complex 的辐角，单位为弧度。根据编译选项，使用:::boost::multiprecision::proj(…)或std::proj(...) 函数。

yade._math.**random**() → 浮点数

返回 Real 区间 (-1,1) 内的对称随机数。供 Eigen 使用。

random((浮点数)a, (浮点数)b) → 浮点数:

返回 Real 区间 (a,b) 内的一个随机数。由 Eigen 使用。

yade._math.**real**((复数)*x*）→浮点数

返回一个 Complex 参数的实部。根据编译选项，使用: ::boost::multiprecision::real(…) 或std::real(...)函数。

yade._math.remainder(*(float)x*, *(float)y*) → 浮点数

返回 Real 浮点除法运算的 IEEE 余数 x/y。根据编译选项，会封装为 ::boost::multiprecision::remainder(…)或std::remainder(...)函数。

yade._math.remquo(*(float)x*, *(float)y*) →元组

返回由 (Real,long)组成的元组，即除法运算的浮点余数 x/y ，其行为与 std::remainder() 函数相同。此外，还会返回 x/y 的符号和至少最后三位，用以确定结果在一个周期内所属的卦限。根据编译选项，会封装为 ::boost::multiprecision::remquo(…) 或std::remquo(...) 函数。

yade._math.rint(*(float)x*) →浮点数

返回 将浮点参数辐角舍入为整数值（浮点格式），采用当前舍入模式。根据编译选项，wraps: ::boost::multiprecision::rint(…) 或 std::rint(...)函数。

yade._math.round(*(float)x*) →浮点数

返回 Real 最接近辐角的整数值（浮点格式），中点情况远离零舍入，无论当前舍入模式如何。根据编译选项，wraps ::boost::multiprecision::round(…) 或 std::round(...) 函数。

yade._math.roundTrip(*(float)x*) → 浮点数

返回 Real 返回参数 x。可用于将类型转换为原生 Re- alHP<N> 精度。

yade._math.sgn(*(float)x*) →整数

返回 int参数的符号： -1、0 或 1。

yade._math.sign(*(浮点数)x*) →整数

返回 int参数的符号： -1、0 或 1。

yade._math.sin(*(复数)x*) →复数

返回 Complex 参数的正弦值（单位为 Complex 弧度）。根据编译选项，wrap ::boost::multiprecision::sin(…) 或std::sin(...) 函数。

sin((浮点数)x) → 浮点数:

返回 Real 以弧度表示的 Real 参数的正弦。根据编译选项，wraps ::boost::multiprecision::sin(…) 或std::sin(...) 函数。

yade._math.sinh(*(复数)x*) →复数

返回 Complex 以弧度表示的Complex 参数的双曲正弦。根据编译选项，wraps: ::boost::multiprecision::sinh(…) 或std::sinh(...) 函数。

sinh((浮点数)x) → 浮点数:

返回 Real 以弧度表示的 Real 参数的双曲正弦。根据编译选项，wraps ::boost::multiprecision::sinh(…) 或std::sinh(...)函数。

yade._math.smallest_positive() → 浮点数

返回 Real 最小正数。wrap std::numeric_lim-its<实数>::min()

yade._math.sphericalHarmonic(*(int)l*, *(int)m*, *(float)θ*, *(float)φ*) → 复数

返回 Real 阶数为 l (unsigned int)、m (signed int) 的球谐多项式，其Real 参数为 theta 和 phi。参见：<https://www.boost.org/doc/libs/1_77_0/libs/math/doc/html/math_toolkit/sf_poly/sph_harm.html>‘___

yade._math.sqrt(*(complex)x*) →复数**返回** Complex 平方根 Complex 参数。根据编译选项，可能会调用 ::boost::multiprecision::sqrt(…) 或std::sqrt(...) 函数。**sqrt((float)x) → 浮点数：返回** Real 参数的平方根。根据编译选项，可能会调用: ::boost::multiprecision::sqrt(…) 或 std::sqrt(...) 函数。yade._math.squaredNorm(*(complex)x*) →浮点数

返回 Real Complex 弧度制下参数的范数（模平方）。根据编译选项，可能会调用 ::boost::multiprecision::norm(…) 或std::norm(...) 函数。

yade._math.tan(*(复数)x*) →复数

返回 Complex 以弧度表示的 Complex 参数的正切值。根据编译选项，wrap 会调用 ::boost::multiprecision::tan(…) 或std::tan(...) 函数。**tan((浮点数)x) → 浮点数：返回** Real 以弧度表示的 Real 参数的正切值。根据编译选项，wrap 会调用 ::boost::multiprecision::tan(…) 或 std::tan(...) 函数。yade._math.tanh(*(复数)x*) →复数

返回 Complex 以弧度表示的 Complex 辐角的双曲正切值。根据编译选项，wrap 会调用: ::boost::multiprecision::tanh(…) 或std::tanh(...) 函数。

tanh((浮点数)x) → 浮点数：返回 Real 以弧度表示的 Real 参数的双曲正切值。根据编译选项，wrap 会调用 ::boost::multiprecision::tanh(…) 或 std::tanh(...)函数。
yade._math.testArray() → 无 此函数测试对 std::vector::data(…) 函数的调用，以提取数组。
yade._math.testConstants() →无 此函数测试 lib/high-precision/Constants.hpp、yade::math::ConstantsHP<N>，用于

mer

yade::Mathr 常数。yade._math.testLoopRealHP() → 无 此函数用于测试 lib/high-precision/Constants.hpp，但 C++ 端：全部精度，甚至

ose

无法从 Python 访问yade._math.tgamma(*(浮点数)x*) → 浮点数**返回** Real 计算辐角的伽马函数。根据编译选项，包装 ::boost::multiprecision::tgamma(…) 或std::tgamma(...) 函数。

yade._math.toDouble((浮点数)*x*）→浮点数

返回 float 将 Real 类型转换为 double，并返回原生 Python float。

yade._math.toHP1((*float*)*x*）→浮点数

返回 RealHP<1> 由参数 RealHP<1> 转换而来，转换依据为 static_cast<RealHP<1>>(arg)。

yade._math.toInt((*float*)*x*）→整数

返回 int 将 Real 类型转换为 int，并返回原生 Python int。

yade._math.toLong((浮点数)*x*）→ int

返回 int 将 Real 类型转换为 long int，并返回一个原生 Python int。

yade._math.toLongDouble((浮点数)*x*）→浮点数

返回 float 将 Real 类型转换为 long double，并返回一个原生 Python float。

yade._math.trunc((浮点数)*x*）→浮点数

返回 Real 不大于辐角的模的最近整数。根据编译选项，使用 wrap ::boost::multiprecision::trunc(…) 或 std::trunc(...)函数。

2.4.9 yade.minieigenHP 模块

当 Yade 将高精度数用作 Real 类型时，通常的（旧）写法：

```
from minieigen import *
```

必须替换为：

```
from yade.minieigenHP import *
```

此 command 可确保两者之间的向后兼容。这样便能保证 Python 使用的小数位数与 Yade 在其他所有地方使用的小数位数相同。

请注意，所用精度可能非常任意，因为 cpp_bin_float 或 mpfr 将其视为编译时参数。因此，这类 yade.minieigenHP 无法作为一个包单独预编译。不过，可以针对某些特殊类型进行预编译，例如 boost::multiprecision::float128。

这些 RealHP<n> 更高精度的向量和矩阵可在 Python 中通过 .HPn 模块作用域访问。例如：

```
import yade.minieigenHP as mne
mne.HP2.Vector3(1,2,3) # produces Vector3 using RealHP<2> precision
mne.Vector3(1,2,3)     # without using HPn module scope it defaults to RealHP<1>
```

miniEigen 是Eigen 库的一部分的封装器。详情请参阅其文档。本模块中的所有类均支持序列化。

class yade._minieigenHP.**AlignedBox2**二维轴对齐包围盒对象，由其最小和最大角点定义**__init__**（（对象）辐角*1*）→无___初始化___((对象)参数1, (二维对齐包围盒)other) -> 无___初始化___((对象)参数1, (二维向量)最小值, (二维向量)最大值) -> 无 **center**（（二维对齐包围盒）辐角*1*）→二维向量 **clamp**（（二维对齐包围盒）参数*1*，（二维对齐包围盒）辐角*2*）→ 无**contains**（（二维对齐包围盒）参数*1*，（二维向量）辐角*2*）→ bool contains((二维对齐包围盒)参数1, (二维对齐包围盒)参数2) -> bool

empty((二维对齐包围盒)辐角*1*) → 布尔值

extend((二维对齐包围盒)参数*1*，(二维向量)辐角*2*) → 无

extend((二维对齐包围盒)参数1,(二维对齐包围盒)参数2) -> 无 intersection((二维对齐包围盒)参数1, (二维对齐包围盒)辐角*2*) → 二维对齐包围盒maxmerged((二维对齐包围盒)参数1, (二维对齐包围盒)辐角*2*) → 二维对齐包围盒minsizes((二维对齐包围盒)辐角*1*) → 二维向量volume((二维对齐包围盒)辐角*1*) →浮点数class yade._minieigenHP.AlignedBox3轴对齐包围盒对象，由其最小和最大角点定义__init__((对象)辐角*1*) →无___初始化___((对象)参数1, (三维对齐包围盒)other) -> 无___初始化___((对象)参数1, (三维向量)最小值, (三维向量)最大值) -> 无center((三维对齐包围盒)辐角*1*) →三维向量clamp((三维对齐包围盒)参数1, (三维对齐包围盒)辐角*2*) → 无contains((三维对齐包围盒)参数1, (三维向量)辐角*2*) → bool contains((三维对齐包围盒)参数1, (三维对齐包围盒)参数2) -> boolempty((三维对齐包围盒)辐角*1*) → boolextend((三维对齐包围盒)参数1, (三维向量)辐角*2*) → 无 extend((三维对齐包围盒)参数1,(三维对齐包围盒)参数2) -> 无 intersection((三维对齐包围盒)参数1, (三维对齐包围盒)辐角*2*) → 三维对齐包围盒maxmerged((三维对齐包围盒)参数1, (三维对齐包围盒)辐角*2*) → 三维对齐包围盒minsizes((三维对齐包围盒)辐角*1*) → 三维向量volume((三维对齐包围盒)辐角*1*) →浮点数class yade._minieigenHP.HP1

class AlignedBox2二维中的轴对齐盒子对象，由其最小角点和最大角点定义

__init__（（对象）辐角*1*）→ 无___初始化___（（对象）参数1，（二维对齐包围盒）other）-> 无___初始化___（（对象）参数1，（二维向量）最小值，（二维向量）最大值）-> 无center（（二维对齐包围盒）辐角*1*）→ 二维向量clamp（（二维对齐包围盒）参数*1*，（二维对齐包围盒）辐角*2*）→ 无contains（（二维对齐包围盒）参数*1*，（二维向量）辐角*2*）→ bool contains((二维对齐包围盒)参数1,(二维对齐包围盒)参数2)-> 布尔值empty（（二维对齐包围盒）辐角*1*）→布尔值 extend（（二维对齐包围盒）参数*1*，（二维向量）辐角*2*）→ 无 extend((二维对齐包围盒)参数1,(二维对齐包围盒)参数2)-> 无intersection（（二维对齐包围盒）参数*1*，（二维对齐包围盒）辐角*2*）→ 二维对齐包围盒

maxmerged(*(二维对齐包围盒)参数1*，*(二维对齐包围盒)辐角2*) → 二维对齐包围盒minsizes(*(二维对齐包围盒)辐角1*) →二维向量volume(*(二维对齐包围盒)辐角1*) → 浮点数

class AlignedBox3

轴对齐包围盒对象，由其最小角点和最大角点定义__init__((对象)arg*1*) → 无___初始化___((对象)arg1, (三维对齐包围盒)other) -> 无___初始化___((对象)arg1, (三维向量)最小值,(三维向量)最大值) -> 无center((三维对齐包围盒)arg*1*) → 三维向量clamp((三维对齐包围盒)arg1, (三维对齐包围盒)arg*2*) → 无contains((三维对齐包围盒)arg1, (三维向量)arg*2*) → 布尔值 contains((三维对齐包围盒)arg1,(三维对齐包围盒)arg2) -> 布尔值empty((三维对齐包围盒)arg*1*) →布尔值extend((三维对齐包围盒)arg1, (三维向量)arg*2*) → 无 extend((三维对齐包围盒)arg1,(三维对齐包围盒)arg2)-> 无intersection((三维对齐包围盒)arg1, (三维对齐包围盒)arg*2*) → 三维对齐包围盒maxmerged((三维对齐包围盒)arg1, (三维对齐包围盒)arg*2*) → 三维对齐包围盒minsizes((三维对齐包围盒)arg*1*) →三维向量volume((三维对齐包围盒)arg*1*) → 浮点数

class Matrix33×3 浮点
矩阵。

支持的运算（m 为 Matrix3， f 为浮点数/整数， v 为三维向量）： -m, m+m, m+=m, m-m,m-=m, m*f, f*m, m*=f, m/f, m/=f, m*m, m*=m, m*v, v*m, m==m, m!=m。

静态属性： Zero, Ones, Identity。

```
Identity = Matrix3(1,0,0, 0,1,0, 0,0,1)
Ones = Matrix3(1,1,1, 1,1,1, 1,1,1)
```

static Random() → Matrix3：返回一个对象，其中所有元素均被随机设为 0 到 1 之间的值。

Zero = Matrix3(0,0,0, 0,0,0, 0,0,0)__init__(*(对象)辐角1*) → 无___初始化___((对象)参数1, (四元数)q) -> 无___初始化___((对象)参数1, (Matrix3)other) -> 无___初始化___((对象)参数1, (三维向量)diag)-> 对象___初始化___((对象)参数1, (浮点数)m00, (浮点数)m01, (浮点数)m02, (浮点数)m10, (浮点数)m11, (浮点数)m12, (浮点数)m20, (浮点数)m21, (浮点数)m22) ->对象___初始化___((对象)参数1, (三维向量)r0, (三维向量)r1, (三维向量)r2 [, (bool)列=False]) ->对象col(*(Matrix3)参数1*, *(int)列*) → 三维向量：返回作为向量的列。

cols((*Matrix3)*辐角 *1*) → 整数:
列数。

computeUnitaryPositive((Matrix3)arg *1*) →tuple：计算极分解（酉矩阵 U 和半正定对称矩阵 P，使得 self=U*P）。determinant((Matrix3)arg *1*) → float：返回矩阵行列式。diagonal((Matrix3)arg *1*) → Vector3：以向量形式返回对角线。inverse((Matrix3)arg *1*) →Matrix3：返回逆矩阵。isApprox((Matrix3)arg1, (Matrix3)other[, (float)prec=*1*e-*12*]) → bool：按精度 prec 进行近似比较。jacobiSVD((Matrix3)arg *1*) → tuple：计算方阵的奇异值分解，返回 (U,S,V)，使得 self=U*S*V.transpose()maxAbsCoeff((Matrix3)arg *1*) → float：所有元素中的最大绝对值。maxCoeff((Matrix3)arg *1*) → float：所有元素中的最大值。mean((Matrix3)arg *1*) →float：所有元素的平均值。minCoeff((Matrix3)arg *1*) → float：所有元素中的最小值。norm((Matrix3)arg *1*) →float：欧几里得范数。normalize((Matrix3)arg *1*) → None：就地归一化此对象。normalized((Matrix3)arg *1*) →Matrix3：返回此对象的归一化副本polarDecomposition((Matrix3)arg *1*) →tuple： *computeUnitaryPositive* 的别名。prod((Matrix3)arg *1*) → float：所有元素的乘积。pruned((Matrix3)arg *1*[, (float)absTol=*1*e-*06*]) → Matrix3：将所有绝对值大于 absTol 的元素置零。不会修剪负零。row((Matrix3)arg1, (int)row) → Vector3：以向量形式返回行。rows((Matrix3)arg *1*) → int：行数。selfAdjointEigenDecomposition((Matrix3)arg *1*) → tuple：计算对称矩阵的特征分解（谱分解），返回 (eigVecs,eigVals)。eigVecs 是列为归一化特征向量的正交 Matrix3，eigVals 是包含对应特征值的 Vector3。self=eigVecs*diag(eigVals)*eigVecs.transpose()。spectralDecomposition((Matrix3)arg *1*) → tuple： *selfAdjointEigenDecomposition* 的别名。squaredNorm((Matrix3)arg *1*) → float：欧几里得范数的平方。

`sum`((Matrix3)arg*1*) → float：所有元素的和。`svd`((Matrix3)arg*1*) → tuple： *jacobiSVD* 的别名。`trace`((Matrix3)arg*1*) →float：返回对角元素的总和。`transpose`((Matrix3)arg*1*) → Matrix3：返回转置矩阵。`class Matrix3c`/TODO/
`Identity = Matrix3c(1,0,0, 0,1,0, 0,0,1)``Ones = Matrix3c(1,1,1, 1,1,1, 1,1,1)`
`static Random()` → Matrix3c：返回一个所有元素均随机设为 0 到 1 之间数值的对象。
`Zero = Matrix3c(0,0,0, 0,0,0, 0,0,0)``__init__`((object)arg*1*) → None__初始化__((object)arg1, (Matrix3c)other) -> None__初始化__((object)arg1, (Vector3c)diag) -> object__初始化__((object)arg1, (complex)m00, (complex)m01, (complex)m02, (complex)m10, (complex)m11, (complex)m12, (complex)m20, (complex)m21, (complex)m22) ->object__初始化__((object)arg1, (Vector3c)r0, (Vector3c)r1, (Vector3c)r2 [, (bool)cols=False]) -> object`col`((Matrix3c)arg1, (int)col) → Vector3c：以向量形式返回列。`cols`((Matrix3c)arg*1*) → int：列数。`determinant`((Matrix3c)arg*1*) → complex：返回矩阵行列式。`diagonal`((Matrix3c)arg*1*) → Vector3c：以向量形式返回对角线。`inverse`((Matrix3c)arg*1*) →Matrix3c：返回逆矩阵。`isApprox`((Matrix3c)arg1, (Matrix3c)other[, (float)prec=*1*e-*12*]) → bool：按精度 prec 进行近似比较。`maxAbsCoeff`((Matrix3c)arg*1*) → float：所有元素中的最大绝对值。`mean`((Matrix3c)arg*1*) → complex：所有元素的平均值。`norm`((Matrix3c)arg*1*) → float：欧几里得范数。`normalize`((Matrix3c)arg *1*) → None：就地归一化此对象。`normalized`((Matrix3c)arg*1*) →Matrix3c：返回此对象的归一化副本`prod`((Matrix3c)arg*1*) → complex：所有元素的乘积。

`pruned`(*(Matrix3c)*辐角 *1*[, *(*浮点数*)*绝对容差 =*1e-06*]) → Matrix3c：将所有大于绝对容差的元素置零。不会修剪负零。`row`(*(Matrix3c)*参数*1*，*(int)*行) → Vector3c：以向量形式返回该行。`rows`(*(Matrix3c)*辐角*1*) → int：行数。`squaredNorm`(*(Matrix3c)*辐角 *1*) → float：欧几里得范数的平方。`sum`(*(Matrix3c)*辐角 *1*) →复数：所有元素的和。`trace`(*(Matrix3c)*辐角 *1*) →复数：返回对角元素的总和。`transpose`(*(Matrix3c)*辐角 *1*) → Matrix3c：返回转置矩阵。

`class Matrix6`

6×6 浮点矩阵。由 4 个 3×3 子矩阵或 6 个 Vector6（行）构造。

支持的运算（`m` 为 Matrix6， `f` 为浮点数/int， `v` 为 Vector6）： `-m`、 `m+m`、 `m+=m`、 `m-m`、 `m-=m`、 `m*f`、 `f*m`、 `m*=f`、 `m/f`、 `m/=f`、 `m*m`、 `m*=m`、 `m*v`、 `v*m`、 `m==m`、 `m!=m`。

静态属性： `Zero`, `Ones`, `Identity`。

`Identity = Matrix6( (1,0,0,0,0,0), (0,1,0,0,0,0), (0,0,1,0,0,0), (0,0,0,1,0,0), (0,0,0,0,1,`

`Ones = Matrix6( (1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1,1,1,1),`

`static Random`() → Matrix6：返回一个对象，其中全部元素均随机设为 0 到 1 之间的值。

`Zero = Matrix6( (0,0,0,0,0,0), (0,0,0,0,0,0), (0,0,0,0,0,0), (0,0,0,0,0,0), (0,0,0,0,0,0),`

`__init__`(*(*对象*)*辐角 *1*) → 无___构造函数___((对象)参数1, (Matrix6)other) -> 无___构造函数___((对象)参数1, (Vector6)diag)-> 对象___构造函数___((对象)参数1, (Matrix3)ul, (Matrix3)ur, (Matrix3)ll, (Matrix3)lr) -> 对象___构造函数___((对象)参数1, (Vector6)l0, (Vector6)l1, (Vector6)l2, (Vector6)l3, (Vector6)l4, (Vector6)l5 [, (bool)列= False]) -> 对象

`col`(*(Matrix6)*参数*1*，*(int)col*) → Vector6：将列作为向量返回。

`cols`(*(Matrix6)arg1*) → 整数：列数。

`computeUnitaryPositive`(*(Matrix6)*辐角 *1*) →元组：

计算极分解（酉矩阵 U 和半正定对称矩阵 P，使得 self=U*P）。

`determinant`(*(*六阶矩阵*)*辐角 *1*) → 浮点数：

返回矩阵行列式。

`diagonal`(*(*六阶矩阵*)*辐角 *1*) → 六维向量：

以向量形式返回对角线。

`inverse`(*(*六阶矩阵*)*辐角 *1*) →六阶矩阵：

返回逆矩阵。

`isApprox`(*(*六阶矩阵*)*参数*1*，*(*六阶矩阵*)*其他[, *(*浮点数*)*精度 =*1e-12*]) → 布尔值：

以精度精度进行近似比较。

jacobiSVD((Matrix6)arg*1*) → 元组：对方阵进行奇异值分解，返回 (U,S,V)，使得 self=U*S*V.transpose()**ll**((Matrix6)arg*1*) →Matrix3：返回左下角的 3×3 块**lr**((Matrix6)arg*1*) →Matrix3：返回右下角的 3×3 块**maxAbsCoeff**((Matrix6)arg*1*) → 浮点数：所有元素中的最大绝对值。**maxCoeff**((Matrix6)arg*1*) → 浮点数：所有元素中的最大值。**mean**((Matrix6)arg*1*) →浮点数：所有元素的平均值。**minCoeff**((Matrix6)arg*1*) → 浮点数：所有元素中的最小值。**norm**((Matrix6)arg*1*) →浮点数：欧几里得范数。**normalize**((Matrix6)arg*1*) → 无：就地归一化此对象。**normalized**((Matrix6)arg*1*) →Matrix6：返回此对象的归一化副本**polarDecomposition**((Matrix6)arg*1*) →元组：*computeUnitaryPositive*的别名。**prod**((Matrix6)arg*1*) →浮点数：所有元素的乘积。**pruned**((Matrix6)arg*1*[, (浮点数)absTol=*1*e-*06*]) → Matrix6：将所有绝对值大于绝对容差的元素置零。负零不会被修剪。**row**((Matrix6)arg1, (整数)row) → Vector6：将行作为向量返回。**rows**((Matrix6)arg*1*) → 整数：行数。**selfAdjointEigenDecomposition**((Matrix6)arg*1*) → 元组：对称矩阵进行特征分解（谱分解），返回 (eigVecs,eigVals)。eigVecs 是正交 Matrix3，其列为归一化特征向量；eigVals 是包含对应特征值的 Vector3。self=eigVecs*diag(eigVals)*eigVecs.transpose().**spectralDecomposition**((Matrix6)arg*1*) → 元组：*selfAdjointEigenDecomposition*的别名。**squaredNorm**((Matrix6)arg*1*) → 浮点数：欧几里得范数的平方。**sum**((Matrix6)arg*1*) → 浮点数：所有元素的和。**svd**((Matrix6)arg*1*) → 元组：*jacobiSVD*的别名。**trace**((Matrix6)arg*1*) →浮点数：返回对角元素之和。**transpose**((Matrix6)arg*1*) → Matrix6：返回转置矩阵。

`ul`（六阶矩阵）辐角 *1*）→
Matrix3：返回左上角 3×3 块`ur`(
（六阶矩阵）辐角 *1*）→Matrix3：
返回右上角 3×3 块

`class Matrix6c`/待办/
`Identity = Matrix6c( (1,0,0,0,0,0), (0,1,0,0,0,0), (0,0,1,0,0,0), (0,0,0,1,0,0`
`), (0 0,0,0,1,0`
`Ones = Matrix6c( (1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1 1,1,1) ,`
`static Random()` → Matrix6c：返回一个对象，其中全部元素均随机设为 0 到 1 之间的值。
`Zero = Matrix6c( (0,0,0,0,0,0), (0,0,0,0,0,0), (0,0,0,0,0,0), (0,0,0,0,0,0), (`
`0,0,0 0,0,0) ,`
`__init__`((对象)arg *1*）→ 无___初始化___((对象)arg1, (Matrix6c)other) -> 无___初始化___((对象)arg1, (Vector6c)diag) -> 对象___初始化___((对象)arg1, (Matrix3c)ul, (Matrix3c)ur, (Matrix3c)ll, (Matrix3c)lr) ->对- 象___初始化___((对象)arg1, (Vector6c)l0, (Vector6c)l1, (Vector6c)l2, (Vector6c)l3, (Vec- tor6c)l4, (Vector6c)l5 [, (bool)列=否]) -> 对象`col`((Matrix6c)arg1, (int)列）→ Vector6c：将列作为向量返回。`cols`((Matrix6c)arg *1*）→ int：列数。`determinant`((Matrix6c)arg *1*）→ 复数：返回矩阵行列式。`diagonal`((Matrix6c)arg *1*）→ Vector6c：将对角线作为向量返回。`inverse`(
(Matrix6c)arg *1*）→Matrix6c：返回逆矩阵。`isApprox`((Matrix6c)arg1, (
Matrix6c)other [, (float)精度=*1e-12*]）→ bool：按精度 prec 进行近似比较。`ll`(
(Matrix6c)arg *1*）→Matrix3c：返回左下角 3x3 块`lr`((Matrix6c)arg *1*）→Matrix3c：返回右下角 3x3 块`maxAbsCoeff`((Matrix6c)arg *1*）→ 浮点数：所有元素的绝对值最大值。`mean`(
(Matrix6c)arg *1*）→ 复数：所有元素的平均值。`norm`((Matrix6c)arg *1*）→ 浮点数：欧几里得范数。`normalize`((Matrix6c)arg *1*）→ 无：就地归一化此对象。`normalized`(
(Matrix6c)arg *1*）→Matrix6c：返回此对象的归一化副本`prod`((Matrix6c)arg *1*）→ 复数：所有元素的乘积。

`pruned`(*(Matrix6c)arg1*[, *(浮点数)绝对容差=1e-06*]) → Matrix6c：将所有大于绝对容差的元素置零。负零不会被保留 ned.

`row`(*(Matrix6c)arg1*, *(int)row*) → Vector6c：返回作为向量的行。`rows`(*(Matrix6c)arg1*) → int：行数。`squaredNorm`(*(Matrix6c)arg1*) → float：欧几里得范数的平方。`sum`(*(Matrix6c)arg1*) → complex：所有元素的和。`trace`(*(Matrix6c)arg1*) →complex：返回对角元素的和。`transpose`(*(Matrix6c)arg1*) → Matrix6c：返回转置矩阵。`ul`(*(Matrix6c)arg1*) →Matrix3c：返回左上角 3x3 块`ur`(*(Matrix6c)arg1*) →Matrix3c：返回右上角 3x3 块

`class MatrixX`

XxX（动态尺寸）浮点矩阵。由行列表（作为 VectorX）构造。

支持的运算（`m` 是 MatrixX， `f` 是浮点数/整数， `v` 是 VectorX）： `-m`, `m+m`, `m+=m`, `m-m`,`m-=m`, `m*f`, `f*m`, `m*=f`, `m/f`, `m/=f`, `m*m`, `m*=m`, `m*v`, `v*m`, `m==m`, `m!=m`。

`static Identity`(*(int)arg1*,*(int)rank*) → MatrixX: 创建给定秩的单位矩阵（平方）。

`static Ones`(*(int)rows*, *(int)cols*) → MatrixX：创建具有给定尺寸的矩阵，其中全部元素均设为 1。

`static Random`(*(int)rows*, *(int)cols*) → MatrixX：创建具有给定尺寸的矩阵，其中全部元素均设为 0 到 1 之间的数（均匀分布）。

`static Zero`(***(int)***行, ***(int)***列) → 矩阵 :
创建给定尺寸的零矩阵

`__init__`(***(object)arg1***) → 否
___初始化___((对象)参数1, (MatrixX)其他) -> 无

___初始化___((对象)参数1, (VectorX)对角线) -> 对象

___初始化___((对象)辐角1 [, (VectorX)r0=VectorX() [, (VectorX)r1=VectorX() [, (VectorX)r2=VectorX() [, (VectorX)r3=VectorX() [, (VectorX)r4=VectorX() [, (VectorX)r5=VectorX() [, (VectorX)r6=VectorX() [, (VectorX)r7=VectorX() [, (VectorX)r8=VectorX() [, (VectorX)r9=VectorX() [, (bool)列=False]]]]]]]]]]) -> 对象

___初始化___(对象)参数1, (对象)行 [, (布尔值)列=False]) -> 对象

`col`(***(MatrixX)***参数*1*, ***(***整数*)***列) → VectorX:
将列作为向量返回。

`cols`(***(MatrixX)***辐角*1*) → 整数:
列数。

`computeUnitaryPositive`(***(MatrixX)***辐角*1*) →元组:
计算极分解（酉矩阵 U 和半正定对称矩阵 P，使得 self=U*P）。

`determinant`(*(MatrixX)*辐角 *1*）→浮点数：返回矩阵行列式。`diagonal`(*(MatrixX)*辐角 *1*）→ VectorX：以向量形式返回对角线。`inverse`(*(MatrixX)*辐角 *1*）→ MatrixX：返回逆矩阵。`isApprox`(*(MatrixX)*参数*1*，*(MatrixX)other*[, *(*浮点数*)prec=1e-12*]）→ 布尔值：以精度 *prec* 进行近似比较。`jacobiSVD`(*(MatrixX)*辐角 *1*）→元组：对方阵进行奇异值分解，返回 (U,S,V)，使得

at

self=U*S*V.transpose()`maxAbsCoeff`((MatrixX)arg *1*）→浮点数：所有元素的最大绝对值。`maxCoeff`((MatrixX)arg *1*）→ 浮点数：所有元素的最大值。`mean`((MatrixX)arg *1*）→浮点数：所有元素的平均值。`minCoeff`((MatrixX)arg *1*）→ 浮点数：所有元素的最小值。`norm`((MatrixX)arg *1*）→浮点数：欧几里得范数。`normalize`((MatrixX)arg *1*）→ None：就地归一化此对象。`normalized`((MatrixX)arg *1*）→ MatrixX：返回此对象的归一化副本`polarDecomposition`((MatrixX)arg *1*）→ 元组： *computeUnitaryPositive*的别名。`prod`((MatrixX)arg *1*）→浮点数：所有元素的乘积。`pruned`((MatrixX)arg *1*[, (浮点数) absTol=*1e-06*]）→ MatrixX：将所有大于 absTol 的元素置为零。负零不会被修剪。`resize`((MatrixX)arg1, (int)行数, (int)列）→ None：更改矩阵的尺寸，保留新矩阵中存在的元素值`row`((MatrixX)arg1, (int)行）→ VectorX：将行作为向量返回。`rows`((MatrixX)arg *1*）→ int：行数。`selfAdjointEigenDecomposition`((MatrixX)arg *1*）→元组：计算对称矩阵的特征分解（谱分解），返回 (eigVecs,eigVals

).

r3

eigVecs 是由归一化特征向量构成列的正交 Matrix3，eigVals 是向量以及相应的特征值。self=eigVecs*diag(eigVals)*eigVecs.transpose().

`spectralDecomposition`(*(MatrixX)*辐角 *1*）→ 元组：*selfAdjointEigenDecomposition*的别名。`squaredNorm`(*(MatrixX)*辐角 *1*）→浮点数：欧几里得范数的平方。`sum`(*(MatrixX)*辐角 *1*）→ 浮点数：所有元素的和。

`svd`(*(MatrixX)*辐角 *1*）→元组：别名 *jacobiSVD*。`trace`(*(MatrixX)*辐角 *1*）→ 浮点数：返回对角元素的总和。`transpose`(*(MatrixX)*辐角 *1*）→ MatrixX：返回转置矩阵。

`class MatrixXc`/TODO/`static Identity`((int)arg1, (int)rank）→ MatrixXc：创建给定秩的单位矩阵（方阵）。`static Ones`((int)rows, (int)cols）→ MatrixXc：创建给定尺寸的矩阵，所有元素均设为 1。`static Random`((int)rows, (int)cols）→ MatrixXc：创建给定尺寸的矩阵，所有元素均设为 0 到 1 之间的数（均匀分布）。`static Zero`((int)rows, (int)cols）→ MatrixXc：创建给定尺寸的零矩阵`__init__`((object)arg*1*）→ None___初始化___((object)arg1, (MatrixXc)other) -> None___初始化___((object)arg1, (VectorXc)diag) -> object___初始化___((object)arg1 [, (VectorXc)r0=VectorXc() [, (VectorXc)r1=VectorXc() [, (Vec- torXc)r2=VectorXc() [, (VectorXc)r3=VectorXc() [, (VectorXc)r4=VectorXc() [, (Vec- torXc)r5=VectorXc() [, (VectorXc)r6=VectorXc() [, (VectorXc)r7=VectorXc() [, (Vec- torXc)r8=VectorXc() [, (VectorXc)r9=VectorXc() [, (bool)cols=False]]]]]]]]]) -> object___初始化___((object)arg1, (object)rows [, (bool)cols=False]) -> object`col`((MatrixXc)arg1, (int)col）→ VectorXc：将列作为向量返回。`cols`((MatrixXc)arg*1*）→ int：列数。`determinant`((MatrixXc)arg*1*）→ complex：返回矩阵行列式。`diagonal`((MatrixXc)arg*1*）→ VectorXc：将对角线作为向量返回。`inverse`((MatrixXc)arg*1*）→ MatrixXc：返回逆矩阵。`isApprox`((MatrixXc)arg1, (MatrixXc)other[, (float)prec=*1e-12*]）→ bool：以精度 prec 进行近似比较。`maxAbsCoeff`((MatrixXc)arg*1*）→ float：所有元素中的最大绝对值。`mean`((MatrixXc)arg*1*）→complex：所有元素的平均值。`norm`((MatrixXc)arg*1*）→float：欧几里得范数。`normalize`((MatrixXc)arg*1*）→None：就地归一化此对象。`normalized`((MatrixXc)arg*1*）→ MatrixXc：返回此对象的归一化副本

prod(*(MatrixXc)arg1*) →complex：所有元素的乘积。**pruned**(*(MatrixXc)arg1*[, *(float)absTol=1e-06*]) → MatrixXc：将所有大于 *absTol* 的元素置零。负零不会被修剪。**resize**(*(MatrixXc)arg1*, *(int)rows*, *(int)cols*) → None：更改矩阵尺寸，并保留新矩阵中原有元素的值**row**(*(MatrixXc)arg1*, *(int)row*) → VectorXc：将行作为向量返回。**rows**(*(MatrixXc)arg1*) → int：行数。**squaredNorm**(*(MatrixXc)arg1*) → float：欧几里得范数的平方。**sum**(*(MatrixXc)arg1*) → complex：所有元素的和。**trace**(*(MatrixXc)arg1*) → complex：返回对角元素的总和。**transpose**(*(MatrixXc)arg1*) →MatrixXc：返回转置矩阵。

class Quaternion

表示旋转的四元数。

支持的运算（q 是四元数， v 是三维向量）： q*q （旋转复合）、 q*=q、q*v （将 v 旋转 q）、 q==q、 q!=q。

静态属性： Identity。

注意：四元数打印时以轴角表示（例如 Identity 是Quaternion((1,0,0),0)，也可以通过轴角表示方式构造。不过，这与内部存储的数据不同；可通过索引访问这些数据[0]（x）、 [1]（y）、 [2]（z）、 [3]（w）。要以编程方式获取轴角，请使用 *Quaternion.toAxisAngle*，它会返回该元组。

Identity = Quaternion((1,0,0),0)

Rotate(*(四元数)arg1*, *(三维向量)v*) → 三维向量

__init__（（对象）辐角 *1*）→ 无___初始化___（（对象）参数1，（三维向量）轴，（浮点数）角度）-> 对象___初始化___（（对象）参数1，（浮点数）角度，（三维向量）轴）-> 对象___初始化___（（对象）参数1，（三维向量）u，（三维向量）v）-> 对象**___初始化___（（对象）参数1，（浮点数）w，（浮点数）x，（浮点数）y，（浮点数）z）-> 无：**从系数初始化。

> **注意：** 系数的顺序为 w、x、y、z。 [] 运算符对它们的编号方式不同，针对 $x\ y\ z\ w$ 的编号为 0...4！

___初始化___（（对象）参数1，（Matrix3）旋转矩阵）-> 无

___初始化___（（对象）参数1，（四元数）other）-> 无

angularDistance（（四元数）参数*1*，（四元数）辐角 *2*）→ 浮点数

conjugate（（四元数）辐角 *1*）→四元数

inverse（（四元数）辐角 *1*）→四元数

norm(*(四元数)辐角 1*) → 浮点数normalize(*(四元数)辐角 1*) → 否
normalized(*(四元数)辐角 1*) → 四元数setFromTwoVectors(*(四元数)参数 1*, *(三维向量)u*, *(三维向量)v*) → 否slerp(*(四元数)参数1*, *(浮点数)t*, *(四元数)other*) → 四元数toAngleAxis(*(四元数)辐角 1*) → 元组
toAxisAngle(*(四元数)辐角 1*) → 元组toRotationMatrix(*(四元数)辐角 1*) → Matrix3toRotationVector(*(四元数)辐角 1*) → 三维向量

class Vector2

三维浮点向量。

支持的运算（f 如果是浮点数/整数， v 是三维向量）： -v, v+v, v+=v, v-v, v-=v, v*f, f*v,v*=f, v/f, v/=f, v==v, v!=v。

从由 2 个浮点数组成的序列（列表、元组等）进行隐式转换。

静态属性： Zero, Ones, UnitX, UnitY。

Identity = Vector2(1,0)

Ones = Vector2(1,1)

static Random() → Vector2：返回一个对象，其所有元素均随机设为 0 到 1 之间的值。static Unit((int)arg*1*) → Vector2UnitX = Vector2(1,0)
UnitY = Vector2(0,1)Zero = Vector2(0,0)__init__((object)arg*1*) → None__init__((object)arg1, (Vector2)other) -> None__init__((object)arg1, (float)x, (float)y) -> NoneasDiagonal((Vector2)arg*1*) →object：返回以此向量为对角线元素的对角矩阵。cols((Vector2)arg*1*) → int：列数。dot((Vector2)arg1, (Vector2)other) → float：与 other 的点积。isApprox((Vector2)arg1, (Vector2)other[, (float)prec=*1*e-*12*]) → bool：以精度 prec 进行近似比较。maxAbsCoeff((Vector2)arg*1*) → float：所有元素绝对值的最大值。maxCoeff((Vector2)arg*1*) → float：所有元素的最大值。mean((Vector2)arg*1*) →float：所有元素的平均值。minCoeff((Vector2)arg*1*) → float：所有元素的最小值。

`norm`((Vector2)arg*1*) →float：欧几里得范数。`normalize`((Vector2)arg*1*) → None：原地归一化此对象。`normalized`((Vector2)arg*1*) → Vector2：返回此对象的归一化副本`outer`((Vector2)arg1, (Vector2)other) → object：与 other 的外积。`prod`((Vector2)arg*1*) →float：所有元素的乘积。`pruned`((Vector2)arg*1*[, (float)absTol=*1e-06*]) → Vector2：将所有大于 absTol 的元素置零。负零不会被修剪。`rows`((Vector2)arg*1*) → int：行数。`squaredNorm`((Vector2)arg*1*) → float：欧几里得范数的平方。`sum`((Vector2)arg*1*) → float：所有元素的和。`class Vector2c`/TODO/ `Identity = Vector2c(1,0)``Ones = Vector2c(1,1)``static Random`() → Vector2c：返回一个对象，其所有元素均随机设为 0 到 1 之间的值。`static Unit`((int)arg*1*) → Vector2c`UnitX = Vector2c(1,0)``UnitY = Vector2c(0,1)``Zero = Vector2c(0,0)` `__init__`((object)arg*1*) → None__init__( (object)arg1, (Vector2c)other) -> None __init__( (object)arg1, (complex)x, (complex)y) -> None`asDiagonal`((Vector2c)arg*1*) →object：返回以此向量为对角线元素的对角矩阵。`cols`((Vector2c)arg*1*) → int：列数。`dot`((Vector2c)arg1, (Vector2c)other) → complex：与 other 的点积。`isApprox`((Vector2c)arg1, (Vector2c)other[, (float)prec=*1e-12*]) → bool：以精度 prec 进行近似比较。`maxAbsCoeff`((Vector2c)arg*1*) → float：所有元素绝对值的最大值。`mean`((Vector2c)arg*1*) → complex：所有元素的平均值。`norm`((Vector2c)arg*1*) → float：欧几里得范数。

`normalize`(*(Vector2c)*辐角 *1*) → 无：将此对象原地归一化。`normalized`(*(Vector2c)*辐角 *1*) → Vector2c：返回此对象的归一化副本`outer`(*(Vector2c)*参数 *1*,*(Vector2c)other*) → 对象：与*other*的外积。`prod`(*(Vector2c)*辐角 *1*) → 复数：所有元素的乘积。`pruned`(*(Vector2c)*辐角 *1*[, *(float)*绝对容差 *=1e-06*]) → Vector2c：将所有大于*absTo*的元素置零。负零不会被修剪。`rows`(*(Vector2c)*辐角 *1*) → 整数：行数。`squaredNorm`(*(Vector2c)*辐角 *1*) → 浮点数：欧几里得范数的平方。`sum`(*(Vector2c)*辐角 *1*) →复数：所有元素的和。

`class Vector2i`

二维整数向量。

支持的运算（i 如果是整数， v 则是 Vector2i）： `-v`, `v+v`, `v+=v`, `v-v`, `v-=v`, `v*i`, `i*v`, `v*=i`,`v==v`, `v!=v`。

从包含 2 个整数的序列（列表、元组等）进行隐式转换。

静态属性： `Zero`, `Ones`, `UnitX`, `UnitY`。

`Identity = Vector2i(1,0)`

`Ones = Vector2i(1,1)`

`static Random`() → 二维向量 Vector2i：返回一个对象，其中所有元素都随机设为 0 到 1 之间的值。`static Unit`(*(int)arg1*) → Vector2i`UnitX = Vector2i(1,0)` `UnitY = Vector2i(0,1)``Zero = Vector2i(0,0)``__init__`(*(object)arg1*) → 无__构造函数__( (object)arg1, (Vector2i)other) -> 无__构造函数__( (object)arg1, (int)x, (int)y) -> 无`asDiagonal`(*(Vector2i)arg1*) →对象：返回一个以此向量为对角线的对角矩阵。`cols`(*(Vector2i)arg1*) → 整数：列数。`dot`(*(Vector2i)arg1*, *(Vector2i)other*) → 整数：与*other*的点积。`isApprox`(*(Vector2i)arg1*, *(Vector2i)other*[, *(int)prec=0*]) → 布尔值：以精度*prec*进行近似比较。`maxAbsCoeff`(*(Vector2i)arg1*) → 整数：所有元素中的最大绝对值。`maxCoeff`(*(Vector2i)arg1*) → 整数：所有元素中的最大值。

`mean`(*(Vector2i)arg1*) → 整数：所有元素的平均值。`minCoeff`(*(Vector2i)arg1*) → 整数：所有元素中的最小值。`outer`(*(Vector2i)arg1*,*(Vector2i)other*) → 对象：与*other*的外积。`prod`(*(Vector2i)arg1*) → 整数：所有元素的乘积。`rows`(*(Vector2i)arg1*) → 整数：行数。`sum`(*(Vector2i)arg1*) → 整数：所有元素的和。

`class Vector3`

三维浮点向量。

已支持的运算（`f` 为浮点数/整数， `v` 为三维向量）： `-v`, `v+v`, `v+=v`, `v-v`, `v-=v`, `v*f`, `f*v`,`v*=f`, `v/f`, `v/=f`, `v==v`, `v!=v`， 以及与 `Matrix3` 和 `Quaternion`的运算。

从由 3 个浮点数构成的序列（列表、元组等）进行隐式转换。

静态属性： `Zero`, `Ones`, `UnitX`, `UnitY`, `UnitZ`。

`Identity = Vector3(1,0,0)`

`Ones = Vector3(1,1,1)`

`static Random`() → Vector3：返回一个对象，其中所有元素均随机设为 0 到 1 之间的值。

`static Unit`((int)arg*1*) → Vector3`UnitX = Vector3(1,0,0)`
`UnitY = Vector3(0,1,0)``UnitZ = Vector3(0,0,1)``Zero = Vector3(0,0,0)`
`__init__`((object)arg*1*) → 无__初始化__((object)arg1, (Vector3)other) -> 无__初始化__((object)arg1 [,(float)x=0.0 [, (float)y=0.0 [, (float)z=0.0]]]) -> 无
`asDiagonal`((Vector3)arg*1*) →Matrix3：返回以此向量为对角线的对角矩阵。`cols`((Vector3)arg*1*) → 整数：列数。`cross`((Vector3)arg1, (Vector3)arg*2*) → Vector3`dot`((Vector3)arg1, (Vector3)other) → 浮点数：与 other 的点积。`isApprox`((Vector3)arg1, (Vector3)other[, (float)prec=*1*e-*12*]) → 布尔值：以精度 prec 进行近似比较。`maxAbsCoeff`((Vector3)arg*1*) → 浮点数：所有元素的最大绝对值。`maxCoeff`((Vector3)arg*1*) → 浮点数：所有元素中的最大值。

mean((Vector3)arg*1*) → 浮点数：所有元素的平均值。minCoeff((Vector3)arg*1*) → 浮点数：所有元素中的最小值。norm((Vector3)arg*1*) → 浮点数：欧几里得范数。normalize((Vector3)arg*1*) → 无：就地归一化此对象。normalized((Vector3)arg*1*) → Vector3：返回此对象的归一化副本outer((Vector3)arg1, (Vector3)other) → Matrix3：与 other 的外积。prod((Vector3)arg*1*) → 浮点数：所有元素的乘积。pruned((Vector3)arg*1*[, (float)absTol=*1*e-*06*]) → Vector3：将所有大于 absTol 的元素归零。负零不会被修剪。rows((Vector3)arg*1*) → 整数：行数。squaredNorm((Vector3)arg*1*) → 浮点数：欧几里得范数的平方。sum((Vector3)arg*1*) → 浮点数：所有元素的和。xy((Vector3)arg*1*) → Vector2xz((Vector3)arg*1*) → Vector2yx((Vector3)arg*1*) → Vector2yz((Vector3)arg*1*) → Vector2zx((Vector3)arg*1*) → Vector2zy((Vector3)arg*1*) → Vector2class Vector3c/TODO/Identity = Vector3c(1,0,0)Ones = Vector3c(1,1,1)static Random() → Vector3c：返回一个对象，其中所有元素都被随机设为 0 到 1 之间的值。static Unit((int)arg*1*) → Vector3cUnitX = Vector3c(1,0,0)UnitY = Vector3c(0,1,0)UnitZ = Vector3c(0,0,1)Zero = Vector3c(0,0,0)__init__((object)arg*1*) → 无__初始化__((object)arg1, (Vector3c)other) -> 无__初始化__((object)arg1 [, (complex)x=0j [, (complex)y=0j [, (complex)z=0j]]]) ->

None

asDiagonal(*(Vector3c)arg1*) → Matrix3c：返回以此向量为对角线元素的对角矩阵。

cols((Vector3c)arg*1*) → int：列数。cross((Vector3c)arg1, (Vector3c)arg*2*) → Vector3cdot((Vector3c)arg1, (Vector3c)other) → 复数：与 other 的点积。isApprox((Vector3c)arg1, (Vector3c)other[, (float)prec=*1*e-*12*]) → bool：以精度 prec 进行近似比较。maxAbsCoeff((Vector3c)arg*1*) → float：所有元素的最大绝对值。mean((Vector3c)arg*1*) → 复数：所有元素的平均值。norm((Vector3c)arg*1*) → float：欧几里得范数。normalize((Vector3c)arg*1*) → None：原地归一化此对象。normalized((Vector3c)arg*1*) → Vector3c：返回此对象的归一化副本outer((Vector3c)arg1, (Vector3c)other) → Matrix3c：与 other 的外积。prod((Vector3c)arg*1*) → 复数：所有元素的乘积。pruned((Vector3c)arg*1*[, (float)absTol=*1*e-*06*]) → Vector3c：将所有大于 absTol 的元素置零。不会修剪负零。rows((Vector3c)arg*1*) → int：行数。squaredNorm((Vector3c)arg*1*) → float：欧几里得范数的平方。sum((Vector3c)arg*1*) →复数：所有元素的和。xy((Vector3c)arg*1*) →Vector2cxz((Vector3c)arg*1*) →Vector2cyx((Vector3c)arg*1*) →Vector2cyz((Vector3c)arg*1*) →Vector2czx((Vector3c)arg*1*) →Vector2czy((Vector3c)arg*1*) →Vector2c

class Vector3i

三维整数向量。

已支持的运算（i 如果是整数， v 是 Vector3i）： -v, v+v, v+=v, v-v, v-=v, v*i, i*v, v*=i,v==v, v!=v。

从包含 3 个整数的序列（列表、元组等）进行隐式转换。

静态属性： Zero, Ones, UnitX, UnitY, UnitZ。

Identity = Vector3i(1,0,0)

Ones = Vector3i(1,1,1)

static Random() → Vector3i：返回一个所有元素均随机设为值 b 的对象 介于 0 和 1 之间。

static Unit((int)辐角1) → Vector3iUnitX = Vector3i(1,0,0) UnitY = Vector3i(0,1,0)UnitZ = Vector3i(0,0,1) Zero = Vector3i(0,0,0)__init__((object)辐角1) → None__初始化__((object)参数1, (Vector3i)other) -> None__初始化__((object)辐角1 [, (int)x=0 [, (int)y=0 [, (int)z=0]]])->

None

asDiagonal((Vector3i)辐角1) →对象：返回以此向量为对角线的对角矩阵。cols((Vector3i)辐角1) → 整数：列数。cross((Vector3i)参数1, (Vector3i)辐角2) → Vector3idot((Vector3i)参数1, (Vector3i)other) → 整数：与 other 的点积。isApprox((Vector3i)参数1, (Vector3i)other[, (int)精度=0]) → 布尔值：以精度精度进行近似比较。maxAbsCoeff((Vector3i)辐角1) → 整数：所有元素中的最大绝对值。maxCoeff((Vector3i)辐角1) → 整数：所有元素中的最大值。mean((Vector3i)辐角1) → 整数：所有元素的平均值。minCoeff((Vector3i)辐角1) → 整数：所有元素中的最小值。outer((Vector3i)参数1, (Vector3i)other) → 对象：与 other 的外积。prod((Vector3i)辐角1) → 整数：所有元素的乘积。rows((Vector3i)辐角1) → 整数：行数。sum((Vector3i)辐角1) → 整数：所有元素的总和。xy((Vector3i)辐角1) →Vector2ixz((Vector3i)辐角1) →Vector2iyx((Vector3i)辐角1) →Vector2iyz((Vector3i)辐角1) →Vector2izx((Vector3i)辐角1) →Vector2izy((Vector3i)辐角1) →Vector2i

class Vector4四维浮点向量。

支持的运算（f 若为浮点数/整数， v 为三维向量）： -v, v+v, v+=v, v-v, v-=v, v*f, f*v,v*=f, v/f, v/=f, v==v, v!=v。
从包含 4 个浮点数的序列（列表、元组等）进行隐式转换。

静态属性： Zero, Ones。

Identity = Vector4(1,0,0, 0)

Ones = Vector4(1,1,1, 1)

static Random() → 四维向量：返回一个对象，其中所有元素都被随机设置为 0 到 1 之间的值。

static Unit（（整型）辐角 *1*）→ 四维向量

Zero = Vector4(0,0,0, 0)

__init__（（对象）辐角 *1*）→ 无___初始化___((对象)参数1, (四维向量)其他)-> 无___初始化___((对象)参数1, (浮点数)v0, (浮点数)v1, (浮点数)v2, (浮点数)v3) -> 无

asDiagonal(*(Vector4)*辐角 *1*）→对象：返回以此向量为对角线元素的对角矩阵。

cols(*(Vector4)*辐角 *1*）→ 整数：列数。

dot(*(Vector4)*参数*1*，*(Vector4)other*）→ 浮点数：与 *other* 的点积。

isApprox((Vector4)参数1, (Vector4)other[, (浮点数)prec=*1*e-*12*]）→ 布尔值：以精度 prec 进行近似比较。maxAbsCoeff((Vector4)辐角 *1*）→ 浮点数：所有元素中的最大绝对值。maxCoeff((Vector4)辐角 *1*）→ 浮点数：所有元素中的最大值。mean((Vector4)辐角 *1*）→浮点数：所有元素的平均值。minCoeff((Vector4)辐角 *1*）→ 浮点数：所有元素中的最小值。norm((Vector4)辐角 *1*）→浮点数：欧几里得范数。normalize((Vector4)辐角 *1*）→ 无：原地归一化此对象。normalized((Vector4)辐角 *1*）→ Vector4：返回此对象的归一化副本outer((Vector4)参数1, (Vector4)other）→ 对象：与 other 的外积。prod((Vector4)辐角 *1*）→浮点数：所有元素的乘积。pruned((Vector4)辐角 *1*[, (浮点数)绝对容差=*1*e-*06*]）→ Vector4：将所有大于绝对容差的元素置零。负零不会

已修剪。

rows(*(*四维向量*)*辐角 *1*）→ 整数：行数。

squaredNorm(*(*四维向量*)*辐角 *1*）→ 浮点数：欧几里得范数的平方。

sum((四维向量)辐角 *1*) → 浮点数:
所有元素的和。

class Vector6
六维浮点向量。

已支持的运算（f 如果是浮点数/整数， v 是一个六维向量）： -v, v+v, v+=v, v-v, v-=v, v*f, f*v,v*=f, v/f, v/=f, v==v, v!=v。

从包含 6 个浮点数的序列（列表、元组等）进行隐式转换。

静态属性： Zero, Ones。

Identity = Vector6(1,0,0, 0,0,0)

Ones = Vector6(1,1,1, 1,1,1)

static Random() → Vector6：返回一个对象，其中全部元素都随机设为 0 到 1 之间的值。

static Unit((整数)辐角 *1*) → 六维向量

Zero = Vector6(0,0,0, 0,0,0)

__init__((对象)辐角 *1*) → 无___初始化___((对象)参数1, (六维向量)其他)-> 无___初始化___((对象)参数1, (浮点数)v0, (浮点数)v1, (浮点数)v2, (浮点数)v3, (浮点数)v4, (浮点数)v5) ->对象___初始化___((对象)参数1, (三维向量)头部, (三维向量)尾部)-> 对象

asDiagonal((六维向量)辐角 *1*) →六阶矩阵：返回以此向量为对角线元素的对角矩阵。

cols((Vector6)arg *1*) → int：列数。dot((Vector6)arg1, (Vector6)other) → float：与 other 的点积。head((Vector6)arg *1*) → Vector3isApprox((Vector6)arg1, (Vector6)other[, (float)prec=*1*e-*12*]) → bool：以精度 prec 进行近似比较。maxAbsCoeff((Vector6)arg *1*) → float：所有元素的最大绝对值。maxCoeff((Vector6)arg *1*) → float：所有元素的最大值。mean((Vector6)arg *1*) → float：所有元素的平均值。minCoeff((Vector6)arg *1*) → float：所有元素的最小值。norm((Vector6)arg *1*) →float：欧几里得范数。
normalize((Vector6)arg *1*) → None：就地归一化此对象。
normalized((Vector6)arg *1*) → Vector6：返回此对象的归一化副本
outer((Vector6)arg1, (Vector6)other) → Matrix6：与 other 的外积。

`prod`((Vector6)arg*1*) → float：所有元素的乘积。`pruned`((Vector6)arg*1*[, (float)absTol=*1e-06*]) → Vector6：将所有绝对值大于 absTol 的元素置为零。负零不会被修剪。`rows`((Vector6)arg*1*) → int：行数。`squaredNorm`((Vector6)arg*1*) → float：欧几里得范数的平方。`sum`((Vector6)arg*1*) → float：所有元素的和。`tail`((Vector6)arg*1*) → Vector3`class Vector6c`/TODO/`Identity = Vector6c(1,0,0, 0,0,0)``Ones = Vector6c(1,1,1, 1,1,1)`
`static Random`() → Vector6c：返回一个对象，其中所有元素均被随机设置为介于 0 和 1 之间的值。`static Unit`((int)arg*1*) → Vector6c`Zero = Vector6c(0,0,0, 0,0,0)``__init__`((object)arg*1*) → None__构造函数__( (object)arg1, (Vector6c)other) -> None__构造函数__( (object)arg1, (complex)v0, (complex)v1, (complex)v2, (complex)v3, (com- plex)v4, (complex)v5) -> object__构造函数__( (object)arg1, (Vector3c)head, (Vector3c)tail) -> object`asDiagonal`((Vector6c)arg*1*) → Matrix6c：返回一个以此向量为对角线元素的对角矩阵。`cols`((Vector6c)arg*1*) → int：列数。`dot`((Vector6c)arg1, (Vector6c)other) → complex：与 other 的点积。`head`((Vector6c)arg*1*) → Vector3c`isApprox`((Vector6c)arg1, (Vector6c)other[, (float)prec=*1e-12*]) → bool：以精度 prec 进行近似比较。`maxAbsCoeff`((Vector6c)arg*1*) → float：所有元素的最大绝对值。`mean`((Vector6c)arg*1*) → complex：所有元素的平均值。`norm`((Vector6c)arg*1*) → float：欧几里得范数。`normalize`((Vector6c)arg*1*) → None：就地归一化此对象。`normalized`((Vector6c)arg*1*) → Vector6c：返回此对象的归一化副本`outer`((Vector6c)arg1, (Vector6c)other) → Matrix6c：与 other 的外积。

`prod`(*(Vector6c)*辐角 *1*) → 复数：所有元素的乘积。`pruned`(*(Vector6c)*辐角 *1*[, *(*浮点数*)*绝对容差 *=1e-06*]) → Vector6c：将所有大于绝对容差的元素置为零。不会修剪负零。`rows`(*(Vector6c)*辐角 *1*) → 整数：行数。`squaredNorm`(*(Vector6c)*辐角 *1*) → 浮点数：欧几里得范数的平方。`sum`(*(Vector6c)*辐角 *1*) →复数：所有元素的和。`tail`(*(Vector6c)*辐角 *1*) →Vector3c

`class Vector6i`

六维浮点向量。

支持的运算（`f` 若 a 为浮点数/整数， `v` 为 Vector6）： `-v`, `v+v`, `v+=v`, `v-v`, `v-=v`, `v*f`, `f*v`,`v*=f`, `v/f`, `v/=f`, `v==v`, `v!=v`。

从包含 6 个整数的序列（列表、元组等）进行隐式转换。

静态属性： `Zero`, `Ones`。

`Identity = Vector6i(1,0,0, 0,0,0)`

`Ones = Vector6i(1,1,1, 1,1,1)`

`static Random`() → Vector6i：返回一个对象，其中所有元素都随机设为 0 到 1 之间的值。

`static Unit`(*(int)*辐角 *1*) → Vector6i

`Zero = Vector6i(0,0,0, 0,0,0)`

`__init__`(*(object)*辐角 *1*) → None__初始化__((object)参数1, (Vector6i)other) -> None__初始化__((object)参数1, (int)v0, (int)v1,(int)v2,(int)v3, (int)v4,(int)v5)-> 对象__初始化__((object)参数1, (Vector3i)head,(Vector3i)tail) -> 对象

`asDiagonal`(*(Vector6i)*辐角 *1*) →对象：返回一个以此向量为对角线的对角矩阵。

`cols`(*(Vector6i)arg1*) → int：列数。`dot`(*(Vector6i)arg1*, *(Vector6i)other*) → int：与*other*的点积。`head`(*(Vector6i)arg 1*) → Vector3i`isApprox`(*(Vector6i)arg1*, *(Vector6i)other*[, *(int)prec=0*]) → bool：以*prec*为精度进行近似比较。

`maxAbsCoeff`(*(Vector6i)arg1*) → int：所有元素中的最大绝对值。

`maxCoeff`(*(Vector6i)arg1*) → int：所有元素中的最大值。`mean`(*(Vector6i)arg1*) → int：所有元素的平均值。`minCoeff`(*(Vector6i)arg1*) → int：所有元素中的最小值。

`outer`(*(Vector6i)arg1*, *(Vector6i)other*) → object：与*other*的外积。`prod`(*(Vector6i)arg1*) → int：所有元素的乘积。`rows`(*(Vector6i)arg1*) → int：行数。`sum`(*(Vector6i)arg1*) → int：所有元素的和。`tail`(*(Vector6i)arg1*) → Vector3i

`class VectorX`

动态大小浮点向量。

支持的运算（`f` 若为浮点数/整数， `v` 为 VectorX）： `-v`、 `v+v`、 `v+=v`、 `v-v`、 `v-=v`、 `v*f`、 `f*v`、`v*=f`、 `v/f`、 `v/=f`、 `v==v`、 `v!=v`。

从由 X 个浮点数构成的序列（列表、元组等）进行隐式转换。

`static Ones`(*(int)*辐角*1*) → VectorX

`static Random`(*(int)len*) → VectorX：返回给定长度的向量，所有元素均随机设为 0 到 1 之间的值。

`static Unit`(*(int)*参数*1*, *(int)*辐角*2*) → VectorX

`static Zero`(*(int)*辐角*1*) → VectorX

`__init__`(*(object)*辐角*1*) → None___初始化___((object)arg1, (VectorX)other)-> None___初始化___ ((object)arg1, (object)vv) -> 对象

`asDiagonal`(*(VectorX)*辐角*1*) → MatrixX：返回以此向量为对角线的对角矩阵。

`cols`(*(VectorX)*辐角*1*) → int：列数。

`dot`(*(VectorX)arg1*, *(VectorX)other*) → 浮点数：与*other*的点积。

`isApprox`(*(VectorX)arg1*, *(VectorX)other*[, *(float)prec=1e-12*]) → 布尔值：以精度*prec*进行近似比较。

`maxAbsCoeff`(*(VectorX)*辐角*1*) →浮点数:
全部元素中的最大绝对值。

`maxCoeff`(*(VectorX)*辐角*1*) → 浮点数:
全部元素中的最大值。

`mean`(*(VectorX)*辐角*1*) →浮点数:
全部元素的平均值。

`minCoeff`(*(VectorX)*辐角*1*) → 浮点数:
所有元素中的最小值。

`norm`(*(VectorX)*辐角*1*) →浮点数:
欧几里得范数。

`normalize`(*(VectorX)*辐角*1*) →无:
就地归一化此对象。

`normalized`(*(VectorX)*辐角*1*) → VectorX:
返回此对象的归一化副本

`outer`((VectorX)arg1，(VectorX)other）→ MatrixX：与 other 的外积。`prod`(
(VectorX)arg*1*）→float：所有元素的乘积。`pruned`((VectorX)arg*1*[, (float)absTol=*1*e-*06*]）→
VectorX：将所有绝对值大于 absTol 的元素置零。负零不会被修剪。`resize`((VectorX)arg1,
(int)arg*2*）→ 无`rows`((VectorX)arg*1*）→ int：行数。`squaredNorm`((VectorX)arg*1*）→float：
欧几里得范数的平方。`sum`((VectorX)arg*1*）→ float：所有元素的和。`class VectorXc`/待办/
`static Ones`((int)arg*1*）→ VectorXc`static Random`((int)len）→ VectorXc：返回给定长度的
向量，所有元素均为 0 到 1 之间的随机值。`static Unit`((int)arg1，(int)arg*2*）→ VectorXc
`static Zero`((int)arg*1*）→ VectorXc`__init__`((object)arg*1*）→ 无__初始化__(
(object)arg1, (VectorXc)other) -> 无__初始化__((object)arg1, (object)vv) -> 对象
`asDiagonal`((VectorXc)arg*1*）→MatrixXc：返回以此向量为对角线的对角矩阵。`cols`(
(VectorXc)arg*1*）→ int：列数。`dot`((VectorXc)arg1，(VectorXc)other）→ 复数：与 other
的点积。`isApprox`((VectorXc)arg1，(VectorXc)other[, (float)prec=*1*e-*12*]）→ 布尔值：以精
度 prec 进行近似比较。`maxAbsCoeff`((VectorXc)arg*1*）→ float：所有元素绝对值的最大值。
`mean`((VectorXc)arg*1*）→复数：所有元素的平均值。`norm`((VectorXc)arg*1*）→float：欧几里得
范数。`normalize`((VectorXc)arg*1*）→无：就地归一化此对象。`normalized`((VectorXc)arg
1）→ VectorXc：返回此对象的归一化副本`outer`((VectorXc)arg1，(VectorXc)other）→
MatrixXc：与 other 的外积。`prod`((VectorXc)arg*1*）→复数：所有元素的乘积。

pruned（（*VectorXc*）辐角 *1*[，（浮点数）绝对容差=*1e-06*]）→ VectorXc：将所有大于绝对容差的元素置零。不会修剪负零。resize（（*VectorXc*）参数*1*，（*int*）参数*2*）→ 无rows（（*VectorXc*）参数*1*）→ int：行数。squaredNorm（（*VectorXc*）参数*1*）→ 浮点数：欧几里得范数的平方。sum（（*VectorXc*）参数*1*）→ 复数：所有元素的和。

vectorize = False

class yade._minieigenHP.Matrix3$_{3\times3}$

浮点矩阵。

支持的运算（m 为 Matrix3， f 为浮点数/整数， v 为三维向量）： -m, m+m, m+=m, m-m, m-=m,m*f, f*m, m*=f, m/f, m/=f, m*m, m*=m, m*v, v*m, m==m, m!=m。

静态属性： Zero, Ones, Identity。

Identity = Matrix3(1,0,0, 0,1,0, 0,0,1)

Ones = Matrix3(1,1,1, 1,1,1, 1,1,1)

static Random() → Matrix3：返回一个对象，其中全部元素都随机设为 0 到 1 之间的值。

Zero = Matrix3(0,0,0, 0,0,0, 0,0,0)

__init__（（对象）辐角 *1*）→无___初始化___（（对象）参数1，（四元数）q) ->无___初始化___（（对象）参数1，（Matrix3）other) -> 无___初始化___（（对象）参数1，（三维向量）diag) -> 对象___初始化___（（对象）参数1，（浮点数）m00，（浮点数）m01，（浮点数）m02，（浮点数）m10，（浮点数）m11，（浮点数）m12，（浮点数）m20，（浮点数）m21，（浮点数）m22) ->对象___初始化___（（对象）参数1，（三维向量）r0，（三维向量）r1，（三维向量）r2 [，(bool)列=False])-> 对象

col（（*Matrix3*）参数*1*，（*int*）列）→ 三维向量：返回作为向量的列。

cols（（*Matrix3*）辐角 *1*）→ 整数：列数。

computeUnitaryPositive（（*Matrix3*）辐角 *1*）→元组：
计算极分解（酉矩阵 U 和半正定对称矩阵 P，使得 self=U*P）。

determinant（（*Matrix3*）辐角 *1*）→ 浮点数：
返回矩阵行列式。

diagonal(*(Matrix3)*辐角 *1*）→ 三维向量：
以向量形式返回对角线。

inverse(*(Matrix3)*辐角 *1*）→Matrix3：
返回逆矩阵。

isApprox(*(Matrix3)*参数*1*，*(Matrix3)other*[，*(*浮点数*)*精度=*1e-12*]）→ 布尔值：
以精度 精度进行近似比较。

jacobiSVD(*(Matrix3)arg1*）→元组：对方阵进行奇异值分解，返回 (U,S,V)，使得 self=U*S*V.transpose()

`maxAbsCoeff`((Matrix3)arg*1*) → 浮点数：所有元素中的最大绝对值。`maxCoeff`((Matrix3)arg*1*) → 浮点数：所有元素中的最大值。`mean`((Matrix3)arg*1*) →浮点数：所有元素的平均值。`minCoeff`((Matrix3)arg*1*) → 浮点数：所有元素中的最小值。`norm`((Matrix3)arg*1*) →浮点数：欧几里得范数。`normalize`((Matrix3)arg*1*) →无：就地归一化此对象。`normalized`((Matrix3)arg*1*) → Matrix3：返回此对象的归一化副本`polarDecomposition`((Matrix3)arg*1*) → 元组： *computeUnitaryPositive*的别名。`prod`((Matrix3)arg*1*) →浮点数：所有元素的乘积。`pruned`((Matrix3)arg*1*[, (float)absTol=*1e-06*]) → Matrix3：将所有大于绝对容差的元素置零。负零不会被修剪。`row`((Matrix3)arg1, (int)row) → Vector3：以向量形式返回指定行。`rows`((Matrix3)arg*1*) → int：行数。`selfAdjointEigenDecomposition`((Matrix3)arg*1*) → 元组：对称矩阵进行特征分解（谱分解），返回 (eigVecs,eigVals)。eigVecs 是由归一化特征向量组成的列向量构成的正交 Matrix3，eigVals 是包含对应特征值的 Vector3。self=eigVecs*diag(eigVals)*eigVecs.transpose()。`spectralDecomposition`((Matrix3)arg*1*) → 元组： *selfAdjointEigenDecomposition*的别名。`squaredNorm`((Matrix3)arg*1*) → 浮点数：欧几里得范数的平方。`sum`((Matrix3)arg*1*) → 浮点数：所有元素的和。`svd`((Matrix3)arg*1*) → 元组： *jacobiSVD*的别名。`trace`((Matrix3)arg*1*) → 浮点数：返回对角元素之和。`transpose`((Matrix3)arg*1*) →Matrix3：返回转置矩阵。

`class yade._minieigenHP.Matrix3c`/待办/`Identity = Matrix3c(1,0,0, 0,1,0, 0,0,1)`
`Ones = Matrix3c(1,1,1, 1,1,1, 1,1,1)``static Random()` → Matrix3c：返回一个所有元素均随机设为 0 到 1 之间值的对象。`Zero = Matrix3c(0,0,0, 0,0,0, 0,0,0)`

__init__((对象)辐角 *1*）→无___初始化___((对象)参数1, (Matrix3c)other) -> 无___初始化___((对象)参数1, (Vector3c)diag) -> 对象___初始化___((对象)参数1, (复数)m00, (复数)m01, (复数)m02, (复数)m10, (复数)m11, (复数)m12, (复数)m20, (复数)m21, (复数)m22) -> 对象___初始化___((对象)参数1, (Vector3c)r0, (Vector3c)r1, (Vector3c)r2 [, (布尔值)col s=False]) -> 对象col((Matrix3c)参数1，(int)列）→ Vector3c：返回列向量。cols((Matrix3c)参数*1*）→ int：列数。determinant((Matrix3c)参数*1*）→ complex：返回矩阵行列式。diagonal((Matrix3c)参数*1*）→ Vector3c：以向量形式返回对角线。inverse((Matrix3c)参数*1*）→Matrix3c：返回逆矩阵。isApprox((Matrix3c)参数1，(Matrix3c)其他[, (float)精度=*1*e-*12*]）→ bool：按精度精度进行近似比较。maxAbsCoeff((Matrix3c)参数*1*）→ float：所有元素中的最大绝对值。mean((Matrix3c)参数*1*）→complex：所有元素的平均值。norm((Matrix3c)参数*1*）→ float：欧几里得范数。normalize((Matrix3c)参数*1*）→None：就地归一化此对象。normalized((Matrix3c)参数*1*）→Matrix3c：返回此对象的归一化副本prod((Matrix3c)参数*1*）→complex：所有元素的乘积。pruned((Matrix3c)参数*1*[, (float)绝对容差=*1*e-*06*]）→ Matrix3c：将所有大于绝对容差的元素置零。不会修剪负零。row((Matrix3c)参数1，(int)行）→ Vector3c：返回行向量。rows((Matrix3c)参数*1*）→ int：行数。squaredNorm((Matrix3c)参数*1*）→ float：欧几里得范数的平方。sum((Matrix3c)参数*1*）→ complex：所有元素的和。trace((Matrix3c)参数*1*）→complex：返回对角元素的总和。transpose((Matrix3c)参数*1*）→Matrix3c：返回转置矩阵。

class yade._minieigenHP.Matrix6 6×6 浮点矩阵。由 4 个 3×3 子矩阵构造，来自 6×Vector6（行）。

已支持的运算（m 为六阶矩阵， f 为浮点数/整数， v 为六维向量）： -m, m+m, m+=m, m-m, m-=m,m*f, f*m, m*=f, m/f, m/=f, m*m, m*=m, m*v, v*m, m==m, m!=m。

静态属性： Zero, Ones, Identity。

Identity = Matrix6((1,0,0,0,0,0), (0,1,0,0,0,0), (0,0,1,0,0,0), (0,0,0,1,0,0), (0,0,0,0,1,0), (0,0,0,0,0,1))

Ones = Matrix6((1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1,1,1,1))

static Random() → 六阶矩阵：返回一个对象，其中全部元素均随机设为介于 0 和 1 之间的值。

Zero = Matrix6((0,0,0,0,0,0), (0,0,0,0,0,0), (0,0,0,0,0,0), (0,0,0,0,0,0), (0,0,0,0,0,0), (0,0,0,0,0,0))

__init__((object)arg1) →None___初始化___((object)arg1, (Matrix6)other) -> None___初始化___((object)arg1, (Vector6)diag) -> object___初始化___((object)arg1, (Matrix3)ul, (Matrix3)ur, (Matrix3)ll, (Matrix3)lr) -> object___初始化___((object)arg1, (Vector6)l0, (Vector6)l1, (Vector6)l2, (Vector6)l3, (Vector6)l4, (Vec- tor6)l5 [, (bool)列=False]) -> object

col((Matrix6)arg1, (int)列) → Vector6：返回列向量。cols((Matrix6)arg1) → int：列数。computeUnitaryPositive((Matrix6)arg1) →tuple：计算极分解（酉矩阵 U 和半正定对称矩阵 P，使得 self=U*P）。determinant((Matrix6)arg1) → float：返回矩阵行列式。diagonal((Matrix6)arg1) → Vector6：以向量形式返回对角线。inverse((Matrix6)arg1) →Matrix6：返回逆矩阵。isApprox((Matrix6)arg1, (Matrix6)other[, (float)精度=1e-12]) → bool：以精度 prec 进行近似比较。jacobiSVD((Matrix6)arg1) →tuple：对方阵进行奇异值分解，返回 (U,S,V)，使得 self=U*S*V.transpose()ll((Matrix6)arg1) → Matrix3：返回左下角 3×3 块lr((Matrix6)arg1) → Matrix3：返回右下角 3×3 块maxAbsCoeff((Matrix6)arg1) → float：所有元素的最大绝对值。maxCoeff((Matrix6)arg1) → float：所有元素的最大值。mean((Matrix6)arg1) →float：所有元素的平均值。minCoeff((Matrix6)arg1) → float：所有元素的最小值。

norm((Matrix6)arg*1*) →float：欧几里得范数。normalize((Matrix6)arg*1*) →None：就地归一化此对象。normalized((Matrix6)arg*1*) → Matrix6：返回此对象的归一化副本 polarDecomposition((Matrix6)arg*1*) → tuple： *computeUnitaryPositive*的别名。prod((Matrix6)arg*1*) →float：所有元素的乘积。pruned((Matrix6)arg*1*[, (float)绝对容差=*1e-06*]) → Matrix6：将所有大于绝对容差的元素置零。负零不会被修剪。row((Matrix6)arg1, (int)行) → Vector6：以向量形式返回行。rows((Matrix6)arg*1*) → int：行数。 selfAdjointEigenDecomposition((Matrix6)arg*1*) → tuple：对称矩阵进行特征分解（谱分解），返回 (eigVecs,eigVals)。eigVecs 是列为已归一化特征向量的正交 Matrix3，eigVals 是包含对应特征值的 Vector3。self=eigVecs*diag(eigVals)*eigVecs.transpose()。 spectralDecomposition((Matrix6)arg*1*) → tuple： *selfAdjointEigenDecomposition*的别名。squaredNorm((Matrix6)arg*1*) → float：欧几里得范数的平方。sum((Matrix6)arg*1*) → float：所有元素的和。svd((Matrix6)arg*1*) → tuple： *jacobiSVD*的别名。trace((Matrix6)arg*1*) → float：返回对角元素的总和。transpose((Matrix6)arg*1*) →Matrix6：返回转置矩阵。ul((Matrix6)arg*1*) → Matrix3：返回左上角 3×3 块ur((Matrix6)arg*1*) → Matrix3：返回右上角 3×3 块

class yade._minieigenHP.Matrix6c/待办/

Identity = Matrix6c((1,0,0,0,0,0), (0,1,0,0,0,0), (0,0,1,0,0,0), (0,0,0,1,0,0), (0,0,0,1,0),

Ones = Matrix6c((1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1,1,1,1), (1,1,1,1,1,1) , (1,

static Random() → Matrix6c：返回一个对象，其全部元素均被随机设置为 0 到 1 之间的值。

Zero = Matrix6c((0,0,0,0,0,0), (0,0,0,0,0,0), (0,0,0,0,0,0), (0,0,0,0,0,0), (0,0,0,0,0,0) , (0,

__init__（（对象）辐角*1*）→无___初始化___（（对象）
参数1, (Matrix6c)other) -> 无___初始化___（（对象）
参数1, (Vector6c)diag) -> 对象

__初始化__(（对象）参数1, (Matrix3c)ul, (Matrix3c)ur, (Matrix3c)ll, (Matrix3c)lr) -> 对象

__初始化__((object)参数1, (Vector6c)l0, (Vector6c)l1, (Vector6c)l2, (Vector6c)l3, (Vector6c)l4, (Vector6c)l5 [, (bool)列=否]) -> 对象**col**((Matrix6c)参数1, (int)列）→ Vector6c：将列作为向量返回。**cols**((Matrix6c)参数*1*）→ int：列数。**determinant**((Matrix6c)参数*1*）→ 复数：返回矩阵行列式。**diagonal**((Matrix6c)参数*1*）→ Vector6c：将对角线作为向量返回。**inverse**((Matrix6c)参数*1*）→Matrix6c：返回逆矩阵。**isApprox**((Matrix6c)参数1, (Matrix6c)其他[, (float)精度=*1*e-*12*]）→ bool：按精度 prec 进行近似比较。**ll**((Matrix6c)参数*1*）→ Matrix3c：返回左下角 3×3 块**lr**((Matrix6c)参数*1*）→Matrix3c：返回右下角 3×3 块**maxAbsCoeff**((Matrix6c)参数*1*）→ float：所有元素的最大绝对值。**mean**((Matrix6c)参数*1*）→复数：所有元素的平均值。**norm**((Matrix6c)参数*1*）→float：欧几里得范数。**normalize**((Matrix6c)参数*1*）→None：就地归一化此对象。**normalized**((Matrix6c)参数*1*）→Matrix6c：返回此对象的归一化副本**prod**((Matrix6c)参数*1*）→复数：所有元素的乘积。**pruned**((Matrix6c)参数*1*[, (float)绝对容差=*1*e-*06*]）→ Matrix6c：将所有大于绝对容差的元素置零。负零不会被修剪。**row**((Matrix6c)参数1, (int)行）→ Vector6c：将行作为向量返回。**rows**((Matrix6c)参数*1*）→ int：行数。**squaredNorm**((Matrix6c)参数*1*）→ float：欧几里得范数的平方。**sum**((Matrix6c)参数*1*）→ 复数：所有元素的和。**trace**((Matrix6c)参数*1*）→复数：返回对角元素的和。**transpose**((Matrix6c)参数*1*）→ Matrix6c：返回转置矩阵。**ul**((Matrix6c)参数*1*）→Matrix3c：返回左上角 3×3 块

ur(*(Matrix6c)*参数*1*) →
Matrix3c：返回右上角 3×3 块

class yade._minieigenHP.MatrixXXxX（动态尺寸）浮点矩阵。由行列表（作为 VectorX）构造。

支持的运算（m 是 MatrixX， f 是浮点数/整数， v 是 VectorX）： -m、 m+m、 m+=m、 m-m、 m-=m、 m*f、 f*m、 m*=f、 m/f、 m/=f、 m*m、 m*=m、 m*v、 v*m、 m==m、 m!=m。

static Identity((int)arg1, (int)rank) → MatrixX：创建指定秩的单位矩阵（方阵）。

static Ones((int)rows, (int)cols) → MatrixX：创建指定尺寸且所有元素均设为 1 的矩阵。

static Random((int)rows, (int)cols) → MatrixX：创建指定尺寸的矩阵，所有元素均设为 0 到 1 之间的均匀分布随机数。static Zero((int)rows, (int)cols) → MatrixX：创建指定尺寸的零矩阵__init__((object)arg*1*) →None__初始化__((object)arg1, (MatrixX)other) -> None__初始化__((object)arg1, (VectorX)diag) -> object__初始化__((object)arg1 [, (VectorX)r0=VectorX() [, (VectorX)r1=VectorX() [, (Vec- torX)r2=VectorX()[, (VectorX)r3=VectorX()[, (VectorX)r4=VectorX()[, (Vec- torX)r5=VectorX()[, (VectorX)r6=VectorX()[, (VectorX)r7=VectorX()[, (Vec- torX)r8=VectorX() [, (VectorX)r9=VectorX() [, (bool)cols=False]]]]]]]]]) -> object__初始化__((object)arg1, (object)rows [, (bool)cols=False]) -> objectcol((MatrixX)arg1, (int)col) → VectorX：返回指定列的向量。cols((MatrixX)arg*1*) → int：列数。computeUnitaryPositive((MatrixX)arg*1*) → tuple：计算极分解，得到酉矩阵 U 和半正定对称矩阵 P，使得 self=U*P）。determinant((MatrixX)arg*1*) → float：返回矩阵行列式。diagonal((MatrixX)arg*1*) → VectorX：以向量形式返回对角线。inverse((MatrixX)arg*1*) → MatrixX：返回逆矩阵。isApprox((MatrixX)arg1, (MatrixX)other[, (float)prec=*1*e-*12*]) → bool：以精度 prec 进行近似比较。jacobiSVD((MatrixX)arg*1*) → tuple：对方阵进行奇异值分解，返回 (U,S,V)，满足 self=U*S*V.transpose()maxAbsCoeff((MatrixX)arg*1*) →float：所有元素的最大绝对值。maxCoeff((MatrixX)arg*1*) →float：所有元素的最大值。mean((MatrixX)arg*1*) → float：所有元素的平均值。

`minCoeff`((MatrixX)arg*1*）→float：所有元素的最小值。`norm`((MatrixX)arg*1*）→ float：欧几里得范数。`normalize`((MatrixX)arg*1*）→ None：原地归一化此对象。`normalized`((MatrixX)arg*1*）→ MatrixX：返回此对象的归一化副本`polarDecomposition`((MatrixX)arg*1*）→tuple： *computeUnitaryPositive* 的别名。`prod`((MatrixX)arg*1*）→ float：所有元素的乘积。`pruned`((MatrixX)arg*1*[, (float)absTol=*1*e-*06*]）→ MatrixX：将所有大于绝对容差的元素置零。负零不会被修剪。`resize`((MatrixX)arg1, (int)rows, (int)cols）→ None：更改矩阵尺寸，并保留新矩阵中已有元素的值`row`((MatrixX)arg1, (int)row）→ VectorX：返回指定行的向量。`rows`((MatrixX)arg*1*）→ int：行数。`selfAdjointEigenDecomposition`((MatrixX)arg*1*）→ tuple：计算对称矩阵的特征分解（谱分解），返回 (eigVecs,eigVals)。eigVecs 是正交 Matrix3，其列为归一化特征向量；eigVals 是 Vector3，包含相应的特征值。self= eigVecs*diag(eigVals)*eigVecs.transpose()。`spectralDecomposition`((MatrixX)arg*1*）→ tuple： *selfAdjointEigenDecomposition* 的别名。`squaredNorm`((MatrixX)arg*1*）→float：欧几里得范数的平方。`sum`((MatrixX)arg*1*）→float：所有元素的和。`svd`((MatrixX)arg*1*）→ tuple： *jacobiSVD* 的别名。`trace`((MatrixX)arg*1*）→ float：返回对角元素的总和。`transpose`((MatrixX)arg*1*）→ MatrixX：返回转置矩阵。

c`lass yade._minieigenHP.MatrixXc`/待办/`static Identity`(*(int)*参数*1*,*(int)*秩）→ MatrixXc：使用给定的秩（平方）创建单位矩阵。`static Ones`(*(int)*行,*(int)*列）→ MatrixXc：创建给定尺寸的矩阵，其中全部元素均设为 1。`static Random`(*(int)*行, *(int)*列）→ MatrixXc：创建给定尺寸的矩阵，其中全部元素均设为 0 到 1 之间的数（均匀分布）。`static Zero`(*(int)*行,*(int)*列）→ MatrixXc：创建给定尺寸的零矩阵

`__init__`(*(object)*辐角*1*) → 无___初始化___((object)参数1, (MatrixXc)other) -> 无___初始化___((object)参数1, (VectorXc)diag) -> 对象___初始化___((object)辐角1 [, (VectorXc)r0=VectorXc() [, (VectorXc)r1=VectorXc() [, (VectorXc)r2=向量Xc() [, (向量Xc)r3=向量Xc() [, (向量Xc)r4=向量Xc() [, (VectorXc)r5=VectorXc() [, (VectorXc)r6=VectorXc() [, (VectorXc)r7=Vector向量Xc() [, (VectorXc)r8=VectorXc() [, (VectorXc)r9=VectorXc() [, (bool)列=否]]]]]]]]]]) -> 对象___初始化___((对象)参数1, (对象)行 [, (布尔值)列=否]) -> 对象`col`((MatrixXc)参数1, (整数)列) → VectorXc：将列作为向量返回。`cols`((MatrixXc)参数*1*) → 整数：列数。`determinant`((MatrixXc)参数*1*) →复数：返回矩阵行列式。`diagonal`((MatrixXc)参数*1*) → VectorXc：将对角线作为向量返回。`inverse`((MatrixXc)参数*1*) → MatrixXc：返回逆矩阵。`isApprox`((MatrixXc)参数1, (MatrixXc)其他[, (浮点数)精度=*1*e-*12*]) → 布尔值：以精度 prec 进行近似比较。`maxAbsCoeff`((MatrixXc)参数*1*) →浮点数：全部元素的最大绝对值。`mean`((MatrixXc)参数*1*) →复数：全部元素的平均值。`norm`((MatrixXc)参数*1*) →浮点数：欧几里得范数。`normalize`((MatrixXc)参数*1*) →无：就地归一化此对象。`normalized`((MatrixXc)参数*1*) → MatrixXc：返回此对象的归一化副本`prod`((MatrixXc)参数*1*) →复数：所有元素的乘积。`pruned`((MatrixXc)参数*1*[, (浮点数)绝对容差=*1*e-*06*]) → MatrixXc：将所有大于绝对容差的元素置零。负零不会被修剪

.

`resize`(*(MatrixXc)*参数*1*，*(int)*行，*(int)*列) → 无：更改矩阵的尺寸，保留新矩阵中仍存在的元素值

x

`row`(*(MatrixXc)*参数*1*，*(int)*行) → VectorXc：以向量形式返回该行。`rows`(*(MatrixXc)*参数*1*) → int：行数。`squaredNorm`(*(MatrixXc)*参数*1*) →浮点数：欧几里得范数的平方。`sum`(*(MatrixXc)*参数*1*) →复数：所有元素的和。`trace`(*(MatrixXc)*参数*1*) → 复数：返回对角元素的和。

transpose(*(MatrixXc)*参数*1*) →MatrixXc:

返回转置矩阵。

class yade._minieigenHP.Quaternion

表示旋转的四元数。

支持的运算（q 是四元数， v 是三维向量）： q*q （旋转复合）、 q*=q、 q*v（将 v 绕 q旋转）、 q==q、 q!=q。

静态属性： Identity。

注意： 四元数打印时以轴角形式表示（例如 Identity 为 Quaternion((1,0, 0), 0)，也可以通过轴角表示构造。不过，这与内部存储的数据不同；内部数据可通过索引访问 [0] (x)、 [1] (y)、 [2] (z)、 [3] (w)。若要以编程方式获取轴角，请使用 *Quaternion.toAxisAngle* ，它会返回元组。

Identity = Quaternion((1,0,0),0)

Rotate(*(四元数)arg1*, *(三维向量)v*) → 三维向量

__init__(*(对象)*辐角*1*) →无___初始化___((对象)参数1, (三维向量)轴,(浮点数)角度) -> 对象___初始化___((对象)参数1, (浮点数)角度, (三维向量)轴)-> 对象___初始化___((对象)参数1, (三维向量)u,(三维向量)v)-> 对象**___初始化___((对象)参数1, (浮点数)w, (浮点数)x, (浮点数)y, (浮点数)z) -> 无：** 从系数初始化。

注意： 系数的顺序是w、x、y、z。 [] 运算符对它们的编号方式不同，对于$x\ y\ z\ w$使用 0...4!

___初始化___((对象)参数1, (Matrix3)旋转矩阵) ->无

___初始化___((对象)参数1, (四元数)其他) -> 无

angularDistance(*(四元数)*参数*1*, *(四元数)*参数*2*) → 浮点数

conjugate(*(四元数)*参数*1*) →四元数

inverse(*(Quaternion)*辐角*1*) →四元数

norm(*(Quaternion)*辐角*1*) →浮点数

normalize(*(Quaternion)*辐角*1*) →无

normalized(*(Quaternion)*辐角*1*) → 四元数

setFromTwoVectors(*(Quaternion)*参数*1*,*(Vector3)u*,*(Vector3)v*) → 无

slerp(*(Quaternion)*参数*1*, *(float)t*,*(Quaternion)other*) → 四元数

toAngleAxis(*(四元数)*辐角*1*) →元组

toAxisAngle(*(四元数)*辐角*1*) →元组

toRotationMatrix(*(四元数)*辐角*1*) →Matrix3

toRotationVector(*(四元数)*辐角*1*) →三维向量

三维浮点向量。

Supp支持的运算（f 如果是浮点数/整数， v 是三维向量）： -v, v+v, v+=v, v-v, v-=v, v*f, f*v, v*= f, v/f, v/=f、 v==v、 v!=v。

从包含 2 个浮点数的序列（列表、元组等）进行隐式转换。

静态属性： `Zero`, `Ones`, `UnitX`, `UnitY` .

`Identity = Vector2(1,0)`

`Ones = Vector2(1,1)`

`static Random()` → 二维向量：返回一个对象，其中全部元素均随机设为介于 0 和 1 之间的值

介于 0 和 1 之间。

`static Unit(`*(int)*辐角 *1*) →二维向量

`UnitX = Vector2(1,0)``UnitY = Vector2(0,1)`

`Zero = Vector2(0,0)``__init__`(*(object)*辐角 *1*) →无___初始化___((object)参数1, (Vector2)other) -> 无___初始化___((object)参数1, (float)x, (float)y) -> 无

`asDiagonal`(*(Vector2)*辐角*1*) → 对象：返回以此向量作为对角线的对角矩阵。

`cols`(*(Vector2)*辐角 *1*) → 整数：列数。

`dot`(*(Vector2)arg1*, *(Vector2)other*) → 浮点数：与 *other* 的点积。

`isApprox`(*(Vector2)arg1*, *(Vector2)other*[, *(float)prec=1e-12*]) → 布尔值：以精度 *prec* 进行近似比较。

`maxAbsCoeff`(*(Vector2)arg1*) → 浮点数：所有元素的最大绝对值。

`maxCoeff`(*(Vector2)arg1*) → 浮点数：所有元素中的最大值。

`mean`(*(Vector2)arg1*) →浮点数：所有元素的平均值。`minCoeff`(*(Vector2)arg1*) → 浮点数：所有元素中的最小值。`norm`(*(Vector2)arg1*) →浮点数：欧几里得范数。`normalize`(*(Vector2)arg1*) →无：就地归一化此对象。`normalized`(*(Vector2)arg1*) → 二维向量：返回此对象的归一化副本`outer`(*(Vector2)arg1*, *(Vector2)other*) → 对象：与*other*的外积。`prod`(*(Vector2)arg1*) →浮点数：所有元素的乘积。`pruned`(*(Vector2)arg1*[, *(float)absTol=1e-06*]) → 二维向量：将所有大于*absTol*的元素置零。负零为 n

未修剪。

`rows`(（二维向量）辐角 *1*) → 整数：行数。`squaredNorm`(（二维向量）辐角 *1*) → 浮点数：欧几里得范数的平方。

sum（（二维向量）辐角 *1*）→ 浮点数：所有元素的和。

class yade._minieigenHP.Vector2c/待办/Identity = Vector2c(1,0) Ones = Vector2c(1,1)static Random() → Vector2c：返回一个对象，其中所有元素均随机设为介于

0 和 1 之间。

static Unit((int)arg *1*) →Vector2cUnitX = Vector2c(1,0) UnitY = Vector2c(0,1)Zero = Vector2c(0,0)__init__((object)arg *1*) → None___初始化___((object)arg1, (Vector2c)other) -> None___初始化___((object)arg1, (complex)x, (complex)y) -> NoneasDiagonal((Vector2c)arg *1*) →object：返回以此向量为对角线元素的对角矩阵。cols((Vector2c)arg *1*) → int：列数。dot((Vector2c)arg1, (Vector2c)other）→ complex：与 other 的点积。isApprox((Vector2c)arg1, (Vector2c)other [, (float)prec=*1*e-*12*]）→ bool：以精度 prec 进行近似比较。maxAbsCoeff((Vector2c)arg *1*) → float：所有元素的最大绝对值。mean((Vector2c)arg *1*) →complex：所有元素的平均值。norm((Vector2c)arg *1*) →float：欧几里得范数。normalize((Vector2c)arg *1*) → None：就地归一化此对象。normalized((Vector2c)arg *1*) → Vector2c：返回此对象的归一化副本。outer((Vector2c)arg1, (Vector2c)other）→ object：与 other 的外积。prod((Vector2c)arg *1*) →complex：所有元素的乘积。pruned((Vector2c)arg *1*[, (float)absTol=*1*e-*06*]）→ Vector2c：将所有绝对值大于 absTol 的元素置零。负零不会

已修剪。

rows(*(Vector2c)*辐角 *1*) → 整数：行数。squaredNorm(*(Vector2c)*辐角 *1*) → 浮点数：欧几里得范数的平方。sum(*(Vector2c)*辐角 *1*) → 复数：所有元素的和。

二维整数向量。

支持的运算（i 如果是整数， v 是 Vector2i）： -v, v+v, v+=v, v-v, v-=v, v*i, i*v, v*=i, v==v,v!=v。

从包含 2 个整数的序列（列表、元组等）进行隐式转换。

静态属性： Zero, Ones, UnitX, UnitY
Identity = Vector2i(1,0)Ones = Vector2i(1,1)

static Random() → Vector2i：返回一个对象，其中所有元素都被随机设为某个值

介于 0 和 1 之间。

static Unit((int)arg*1*) →Vector2iUnitX = Vector2i(1,0)
UnitY = Vector2i(0,1)Zero = Vector2i(0,0)__init__(
(object)arg*1*) →None__初始化__((object)arg1,
(Vector2i)other) -> None__初始化__((object)arg1, (int)x,
(int)y) -> NoneasDiagonal((Vector2i)arg*1*) →object：返回以此向量为对角线的对角矩阵。cols((Vector2i)arg*1*) → int：列数。
dot((Vector2i)arg1, (Vector2i)other) → int：与 other 的点积。
isApprox((Vector2i)arg1, (Vector2i)other[, (int)prec=*0*]) → bool：以精度 prec 进行近似比较。maxAbsCoeff((Vector2i)arg*1*) → int：所有元素的最大绝对值。maxCoeff((Vector2i)arg*1*) → int：所有元素中的最大值。mean((Vector2i)arg*1*) → int：所有元素的平均值。minCoeff((Vector2i)arg*1*) → int：所有元素中的最小值。outer((Vector2i)arg1, (Vector2i)other) → object：与 other 的外积。prod((Vector2i)arg*1*) → int：所有元素的乘积。rows((Vector2i)arg*1*) → int：行数。sum((Vector2i)arg*1*) → int：所有元素的和。

class yade._minieigenHP.Vector3三维浮点向量。

支持的运算（f 如果是浮点数/整数， v 是三维向量）： -v, v+v, v+=v, v-v, v-=v, v*f, f*v, v*=f,v/f, v/=f, v==v, v!=v， 以及与 Matrix3 和 Quaternion的运算。

由包含 3 个浮点数的序列（列表、元组等）进行隐式转换。

静态属性： `Zero`, `Ones`, `UnitX`, `UnitY`, `UnitZ`。

`Identity = Vector3(1,0,0)`

`Ones = Vector3(1,1,1)`

`static Random()` → 三维向量：返回一个对象，其所有元素均随机设为 0 到 1 之间的值。

`static Unit(`*(int)arg1*`)` →三维向量`UnitX = Vector3(1,0,0)`
`UnitY = Vector3(0,1,0)UnitZ = Vector3(0,0,1)Zero = Vector3(0,0,0)`
`__init__(`*(object)arg1*`)` →无___构造函数___((object)参数1, (Vector3)other) -> 无___构造函数___((object)arg1 [,(float)x=0.0 [, (float)y=0.0 [, (float)z=0.0]]]) ->无
`asDiagonal(`*(Vector3)arg1*`)` → Matrix3：返回以此向量为对角线的对角矩阵。`cols(`*(Vector3)arg1*`)` → 整数：列数。

`cross(`*(三维向量)参数1*, *(三维向量)辐角2*`)` →三维向量

`dot`((Vector3)arg1, (Vector3)other) → 浮点数：与 other 的点积。`isApprox(`(Vector3)arg1, (Vector3)other[, (float)prec=*1e-12*]) → bool：以精度 prec 进行近似比较。`maxAbsCoeff`((Vector3)arg*1*) → 浮点数：所有元素的最大绝对值。`maxCoeff`((Vector3)arg*1*) → 浮点数：所有元素的最大值。`mean`((Vector3)arg*1*) → 浮点数：所有元素的平均值。`minCoeff`((Vector3)arg*1*) → 浮点数：所有元素的最小值。`norm`((Vector3)arg*1*) →浮点数：欧几里得范数。`normalize`((Vector3)arg*1*) →None：就地归一化此对象。`normalized`((Vector3)arg*1*) → Vector3：返回此对象的归一化副本`outer`((Vector3)arg1, (Vector3)other) → Matrix3：与 other 的外积。`prod`((Vector3)arg*1*) →浮点数：所有元素的乘积。`pruned`((Vector3)arg*1*[, (float)absTol=*1e-06*]) → Vector3：将所有大于 absTol 的元素置零。负零不会被修剪。

rows((Vector3)arg*1*) → 整数：行数。squaredNorm((Vector3)arg*1*) → 浮点数：欧几里得范数的平方。sum((Vector3)arg*1*) → 浮点数：所有元素的和。xy((Vector3)arg*1*) →Vector2xz((Vector3)arg*1*) →Vector2yx((Vector3)arg*1*) →Vector2yz((Vector3)arg*1*) →Vector2zx((Vector3)arg*1*) →Vector2zy((Vector3)arg*1*) →Vector2class yade._minieigenHP.Vector3c/TODO/Identity = Vector3c(1,0,0)Ones = Vector3c(1,1,1)static Random() → Vector3c：返回一个对象，其中所有元素均随机设为 0 到 1 之间的值。static Unit((int)arg*1*) →Vector3c UnitX = Vector3c(1,0,0)UnitY = Vector3c(0,1,0)UnitZ = Vector3c(0,0,1) Zero = Vector3c(0,0,0)__init__((object)arg*1*) →None__初始化__((object)arg1, (Vector3c)other) -> None__初始化__((object)arg1 [,(complex)x=0j [, (complex)y=0j [, (complex)z=0j]]]) ->NoneasDiagonal((Vector3c)arg*1*) →Matrix3c：返回以此向量为对角线元素的对角矩阵。cols((Vector3c)arg*1*) → 整数：列数。cross((Vector3c)arg1, (Vector3c)arg*2*) → Vector3cdot((Vector3c)arg1, (Vector3c)other) →复数：与 other 的点积。isApprox((Vector3c)arg1, (Vector3c)other[, (float)prec=*1*e-*12*]) → bool：以精度 prec 进行近似比较。maxAbsCoeff((Vector3c)arg*1*) → 浮点数：所有元素的最大绝对值。mean((Vector3c)arg*1*) →复数：所有元素的平均值。norm((Vector3c)arg*1*) →浮点数：欧几里得范数。normalize((Vector3c)arg*1*) → None：就地归一化此对象。

`normalized`((Vector3c)辐角 *1*) → Vector3c：返回此对象的归一化副本`outer`((Vector3c)参数1, (Vector3c)other) → Matrix3c：与 other 的外积。`prod`((Vector3c)辐角 *1*) →复数：所有元素的乘积。`pruned`((Vector3c)辐角 *1*[, (浮点数)绝对容差=*1*e-*06*]) → Vector3c：将所有大于绝对容差的元素置零。不会修剪负零。`rows`((Vector3c)辐角 *1*) → 整数：行数。`squaredNorm`((Vector3c)辐角 *1*) → 浮点数：欧几里得范数的平方。`sum`((Vector3c)辐角 *1*) → 复数：所有元素的和。`xy`((Vector3c)辐角 *1*) →Vector2c`xz`((Vector3c)辐角 *1*) →Vector2c`yx`((Vector3c)辐角 *1*) →Vector2c`yz`((Vector3c)辐角 *1*) →Vector2c`zx`((Vector3c)辐角 *1*) →Vector2c`zy`((Vector3c)辐角 *1*) →Vector2c

`class yade._minieigenHP.Vector3i`

三维整数向量。

支持的运算（`i` 如果是一个整数， `v` 是一个 Vector3i）： `-v`, `v+v`, `v+=v`, `v-v`, `v-=v`, `v*i`, `i*v`, `v*=i`, `v==v`,`v!=v`。

从包含 3 个整数的序列（列表、元组等）进行隐式转换。

静态属性： `Zero`, `Ones`, `UnitX`, `UnitY`, `UnitZ`。`Identity = Vector3i(1,0,0)` `Ones = Vector3i(1,1,1)``static Random`() → Vector3i：返回一个对象，其中全部元素均随机设为 0 到 1 之间的值。`static Unit`(*(int)*辐角 *1*) →Vector3i `UnitX = Vector3i(1,0,0)``UnitY = Vector3i(0,1,0)``UnitZ = Vector3i(0,0,1)` `Zero = Vector3i(0,0,0)``__init__`(*(object)*辐角 *1*) →无___初始化___( (object)参数1, (Vector3i)other) -> 无___初始化___( (object)辐角1 [, (int)x=0 [, (int)y=0 [, (int)z=0]]]) -> 无`asDiagonal`(*(Vector3i)*辐角 *1*) →对象：返回以此向量为对角线的对角矩阵。`cols`(*(Vector3i)*辐角 *1*) → 整数：列数。`cross`(*(Vector3i)*参数*1*,*(Vector3i)*辐角 *2*) → Vector3i

`dot`(*(Vector3i)*参数*1*，*(Vector3i)other*）→ 整数：与 *other* 的点积。`isApprox`(*(Vector3i)*参数*1*，*(Vector3i)other*[, *(int)prec=0*])　→布尔值：

精度为 prec 的近似比较。`maxAbsCoeff`(
(Vector3i)arg*1*）→ 整数：所有元素的最大绝对值。
`maxCoeff`((Vector3i)arg*1*）→ 整数：所有元素的最大值。`mean`((Vector3i)arg*1*）→ 整数：所有元素的平均值。`minCoeff`((Vector3i)arg*1*）→ 整数：所有元素的最小值。`outer`((Vector3i)arg1,
(Vector3i)other）→ 对象：与 other 的外积。
`prod`((Vector3i)arg*1*）→ 整数：所有元素的乘积。
`rows`((Vector3i)arg*1*）→ 整数：行数。`sum`(
(Vector3i)arg*1*）→ 整数：所有元素的和。`xy`(
(Vector3i)arg*1*）→Vector2i`xz`((Vector3i)arg
1）→Vector2i`yx`((Vector3i)arg*1*）→Vector2i`yz`(
(Vector3i)arg*1*）→Vector2i`zx`((Vector3i)arg
1）→Vector2i`zy`((Vector3i)arg*1*）→Vector2i

`class yade._minieigenHP.Vector4`四维浮点向量。

支持的运算（`f` 如果是浮点数/整数，　`v` 是三维向量）：　`-v`, `v+v`, `v+=v`, `v-v`, `v-=v`, `v*f`, `f*v`, `v*=f`,`v/f`, `v/=f`, `v==v`, `v!=v`。

从包含 4 个浮点数的序列（列表、元组等）进行隐式转换。

静态属性：　`Zero`, `Ones`。

`Identity = Vector4(1,0,0, 0)`

`Ones = Vector4(1,1,1, 1)`

`static Random`() → 四维向量:
返回一个对象，其中全部元素都随机设为 0 到 1 之间的值。

`static Unit`(*(int)*辐角 *1*）→四维向量

`Zero = Vector4(0,0,0, 0)`

`__init__`(*(object)*辐角 *1*）→无
___初始化___((对象)参数1, (四维向量)其他) -> 无
___初始化___((对象)参数1, (浮点数)v0, (浮点数)v1, (浮点数)v2, (浮点数)v3) ->无

`asDiagonal`(（四维向量）辐角 *1*）→ 对象：返回以此向量为对角线元素的对角矩阵。

cols（（四维向量）辐角*1*）→ 整数：列数。

dot((Vector4)arg1, (Vector4)other）→ 浮点数：与 other 的点积。isApprox(
(Vector4)arg1, (Vector4)other[, (float)prec=*1*e-*12*]）→ 布尔值：与 other 进行近似比较，精度为 prec。maxAbsCoeff((Vector4)arg*1*）→ 浮点数：所有元素中的最大绝对值。maxCoeff((Vector4)arg*1*）→ 浮点数：所有元素中的最大值。mean(
(Vector4)arg*1*）→浮点数：所有元素的平均值。minCoeff((Vector4)arg*1*）→ 浮点数：所有元素中的最小值。norm((Vector4)arg*1*）→浮点数：欧几里得范数。
normalize((Vector4)arg*1*）→None：就地归一化此对象。normalized(
(Vector4)arg*1*）→ Vector4：返回此对象的归一化副本outer((Vector4)arg1,
(Vector4)other）→ 对象：与 other 的外积。prod((Vector4)arg*1*）→浮点数：所有元素的乘积。pruned((Vector4)arg*1*[, (float)absTol=*1*e-*06*]）→ Vector4：将所有大于 absTol 的元素置零。负零不会被修剪。rows((Vector4)arg*1*）→ 整数：行数。
squaredNorm((Vector4)arg*1*）→ 浮点数：欧几里得范数的平方。sum((
Vector4)arg*1*）→ 浮点数：所有元素的和。

class yade._minieigenHP.Vector6六维浮点向量。

支持的运算（f 为浮点数/整数， v 为六维向量）： -v, v+v, v+=v, v-v, v-=v, v*f, f*v, v*=f,v/f, v/=f, v==v, v!=v。

从包含 6 个浮点数的序列（列表、元组等）隐式转换。

静态属性： Zero, Ones。

Identity = Vector6(1,0,0, 0,0,0)

Ones = Vector6(1,1,1, 1,1,1)static Random() → Vector6：返回一个对象，其中全部元素都随机设置为 0 到 1 之间的值

.

static Unit(*(int)*辐角*1*）→Vector6

Zero = Vector6(0,0,0, 0,0,0)

__init__((对象)辐角*1*）→无___初始化___((对象)参数1, (六维向量)other)-> 无___初始化___((对象)参数1, (浮点数)v0, (浮点数)v1, (浮点数)v2, (浮点数)v3, (浮点数)v4, (浮点数)v5) ->对象___初始化___((对象)参数1, (三维向量)HEAD, (三维向量)tail)-> 对象

asDiagonal((Vector6)arg*1*）→ Matrix6：返回以此向量为对角线的对角矩阵。cols((Vector6)arg*1*）→ int：列数。dot((Vector6)arg1, (Vector6)other）→ float：与 other 的点积。head((Vector6)arg*1*）→Vector3isApprox((Vector6)arg1, (Vector6)other[, (float)prec=*1*e-*12*]）→ bool：以精度 prec 进行近似比较。maxAbsCoeff((Vector6)arg*1*）→ float：所有元素的最大绝对值。maxCoeff((Vector6)arg*1*）→ float：所有元素的最大值。mean((Vector6)arg*1*）→ float：所有元素的平均值。minCoeff((Vector6)arg*1*）→ float：所有元素的最小值。norm((Vector6)arg*1*）→float：欧几里得范数。normalize((Vector6)arg*1*）→ None：就地归一化此对象。normalized((Vector6)arg*1*）→ Vector6：返回此对象的归一化副本。outer((Vector6)arg1, (Vector6)other）→ Matrix6：与 other 的外积。prod((Vector6)arg*1*）→float：所有元素的乘积。pruned((Vector6)arg*1*[, (float)absTol=*1*e-*06*]）→ Vector6：将所有绝对值大于 absTol 的元素置零。负零不会被修剪

.

rows((六维向量)辐角*1*）→ 整数：行数。squaredNorm((六维向量)辐角*1*）→ 浮点数：欧几里得范数的平方。sum((六维向量)辐角*1*）→ 浮点数：所有元素的和。tail((六维向量)辐角*1*）→三维向量

class yade._minieigenHP.Vector6c/待办/

```
Identity = Vector6c(1,0,0, 0,0,0)
Ones = Vector6c(1,1,1, 1,1,1)
```

`static Random()` → Vector6c：返回一个对象，其中全部元素的值均随机设为介于 0 和 d 1.

`static Unit(`*(int)*辐角 *1*）→Vector6c

`Zero = Vector6c(0,0,0, 0,0,0)`

`__init__`*((object)*参数 *1*）→None__初始化__((object)参数1, (Vector6c)other) -> None__初始化__((object)参数1, (complex)v0, (complex)v1, (complex)v2, (complex)v 3, (complex)v4, (complex)v5) -> 对象__初始化__((object)参数1, (Vector3c)head, (Vector3c)tail) -> 对象`asDiagonal`((Vector6c)参数 *1*）→Matrix6c：返回以此向量为对角线的对角矩阵。`cols`((Vector6c)参数 *1*）→ int：列数。`dot`((Vector6c)参数 1, (Vector6c)other）→复数：与 other 的点积。`head`((Vector6c)参数 *1*）→ Vector3`cisApprox`((Vector6c)参数1, (Vector6c)other[, (float)精度=*1*e-*12*]）→ bool：以精度 prec 进行近似比较。`maxAbsCoeff`((Vector6c)参数 *1*）→ float：所有元素的最大绝对值。`mean`((Vector6c)参数 *1*）→复数：所有元素的平均值。`norm`((Vector6c)参数 *1*）→float：欧几里得范数。`normalize`((Vector6c)参数 *1*）→ None：就地归一化此对象。`normalized`((Vector6c)参数 *1*）→ Vector6c：返回此对象的归一化副本`outer`((Vector6c)参数1, (Vector6c)other）→ Matrix6c：与 other 的外积。`prod`((Vector6c)参数 *1*）→复数：所有元素的乘积。`pruned`((Vector6c)参数 *1*[, (float)绝对容差=*1*e-*06*]）→ Vector6c：将所有大于绝对容差的元素置零。负零不会被修剪

.

`rows`(*(Vector6c)*辐角 *1*）→ int：行数。
`squaredNorm`(*(Vector6c)*辐角 *1*）→ float：欧几里得范数的平方。`sum`(*(Vector6c)*辐角 *1*）→ complex：所有元素的和。`tail`(*(Vector6c)*辐角 *1*）→ Vector3c

`cla`六维浮点向量。支持的运算（`f` 如果是浮点数/整数， `v` 是 Vector6）： `-v`、 `v+v`、 `v+=v`、 `v-v`、 `v-=v`，

`v*f`, `f*v`, `v*=f`,`v/f`, `v/=f`, `v==v`, `v!=v`。

由包含 6 个交互作用的序列（列表、元组.....）进行隐式转换。

静态属性： `Zero`, `Ones`。

`Identity = Vector6i(1,0,0, 0,0,0)`

`Ones = Vector6i(1,1,1, 1,1,1)`

`static Random()` → Vector6i：返回一个对象，其中所有元素都随机设为 0 到 1 之间的值。

`static Unit`(*(int)*辐角 *1*) →Vector6i

`Zero = Vector6i(0,0,0, 0,0,0)`

`__init__`(*(object)*辐角 *1*) →无___初始化___((object)参数1, (Vector6i)other) -> 无___初始化___((object)参数1, (int)v0, (int)v1, (int)v2, (int)v3, (int)v4, (int)v5) -> 对象___初始化___((object)参数1, (Vector3i)head, (Vector3i)tail) -> 对象

`asDiagonal`(*(Vector6i)*辐角*1*) →对象：返回一个以此向量为对角线的对角矩阵。

`cols`(*(Vector6i)arg1*) → int：列数。

`dot`(*(Vector6i)arg1*, *(Vector6i)other*) → int：与 *other* 的点积。

`head`(*(Vector6i)arg1*) →Vector3i`isApprox`(*(Vector6i)arg1*, *(Vector6i)other*[, *(int)prec=0*]) → bool：与 *prec* 进行近似比较。

`maxAbsCoeff`(*(Vector6i)arg1*) → int：所有元素的最大绝对值。

`maxCoeff`(*(Vector6i)arg1*) → int：所有元素的最大值。`mean`(*(Vector6i)arg1*) → int：所有元素的平均值。`minCoeff`(*(Vector6i)arg1*) → int：所有元素的最小值。`outer`(*(Vector6i)arg1*,*(Vector6i)other*) → object：与 *other* 的外积。`prod`(*(Vector6i)arg1*) → int：所有元素的乘积。`rows`(*(Vector6i)arg1*) → int：行数。`sum`(*(Vector6i)arg1*) → int：所有元素的和。`tail`(*(Vector6i)arg1*) →Vector3i

`class yade._minieigenHP.VectorX`

动态大小浮点向量。

支持的运算（`f` 如果是浮点数/整数， `v` 是 VectorX）： `-v`、 `v+v`、 `v+=v`、 `v-v`、 `v-=v`、 `v*f`、 `f*v`、 `v*=f`、`v/f`、 `v/=f`、 `v==v`、 `v!=v`。

从由 X 个浮点数组成的序列（列表、元组.....）进行隐式转换。

`static Ones`(*(int)*辐角 *1*) →VectorX

static Random(*(int)*长度）→ VectorX：返回指定长度的向量，所有元素均设为 0 到 1 之间的随机值 mly.

static Unit(*(int)*参数*1*，*(int)*辐角*2*）→ VectorX

static Zero(*(int)*辐角*1*）→VectorX

__init__(*(object)*辐角*1*）→None__初始化__(
(object)参数1, (VectorX)other)-> None__初始化__
((object)参数1, (object)vv) ->对象

asDiagonal(*(VectorX)*辐角*1*）→ MatrixX：返回以此向量为对角线的对角矩阵。

cols(*(VectorX)*辐角*1*）→ int：列数。

dot(*(VectorX)arg1*，*(VectorX)other*）→ 浮点数：与 *other* 的点积。

isApprox((VectorX)arg1, (VectorX)other[, (float)prec=*1*e-*12*]）→ 布尔值：以精度 prec 进行近似比较。maxAbsCoeff((VectorX)arg*1*）→浮点数：所有元素中的最大绝对值。maxCoeff((VectorX)arg*1*）→浮点数：所有元素中的最大值。mean((VectorX)arg*1*）→ 浮点数：所有元素的平均值。minCoeff((VectorX)arg*1*）→浮点数：所有元素中的最小值。norm((VectorX)arg*1*）→ 浮点数：欧几里得范数。normalize((VectorX)arg*1*）→无：就地归一化此对象。normalized((VectorX)arg*1*）→ VectorX：返回此对象的归一化副本outer((VectorX)arg1, (VectorX)other）→ MatrixX：与 other 的外积。prod((VectorX)arg*1*）→ 浮点数：所有元素的乘积。pruned((VectorX)arg*1*[, (float)absTol=*1*e-*06*]）→ VectorX：将所有绝对值大于 absTol 的元素置零。不会修剪负零。resize((VectorX)arg1, (int)arg*2*）→ 无
rows((VectorX)arg*1*）→ 整数：行数。squaredNorm((VectorX)arg*1*）→浮点数：欧几里得范数的平方。sum((VectorX)arg*1*）→浮点数：所有元素的和。

class yade._minieigenHP.VectorXc/待办/static Ones(*(int)*辐角*1*）→ VectorXc

`static Random`(*(int)*长度）→ VectorXc：返回指定长度的向量，其中全部元素均设为 0 到 1 之间的值 随机。

`static Unit`(*(int)*参数*1*,*(int)*辐角*2*）→ VectorXc

`static Zero`(*(int)*辐角*1*）→VectorXc

`__init__`((对象)arg*1*）→无___初始化___((对象)arg1, (VectorXc)other) -> 无___初始化___((对象)arg1, (对象)vv) ->对象`asDiagonal`((VectorXc)arg*1*）→ MatrixXc：返回以此向量为对角线的对角矩阵。`cols`((VectorXc)arg*1*）→ 整数：列数。`dot`((VectorXc)arg1, (VectorXc)other）→ 复数：与 other 的点积。`isApprox`(
(VectorXc)arg1, (VectorXc)other[, (浮点数)prec=*1*e-*12*]）→ 布尔值：以精度 prec 进行近似比较。`maxAbsCoeff`((VectorXc)arg*1*）→浮点数：所有元素中的最大绝对值。`mean`((VectorXc)arg*1*）→复数：所有元素的平均值。`norm`((
VectorXc)arg*1*）→浮点数：欧几里得范数。`normalize`((VectorXc)arg*1*）→无：就地归一化此对象。`normalized`((VectorXc)arg*1*）→ VectorXc：返回此对象的归一化副本`outer`((VectorXc)arg1, (VectorXc)other）→ MatrixXc：与 other 的外积。`prod`((VectorXc)arg*1*）→复数：所有元素的乘积。`pruned`((VectorXc)arg*1*[, (浮点数)absTol=*1*e-*06*]）→ VectorXc：将所有大于 absTol 的元素置零。不会修剪负零。`resize`((VectorXc)arg1, (int)arg*2*）→ 无`rows`((VectorXc)arg*1*）→ 整数：行数。`squaredNorm`((VectorXc)arg*1*）→浮点数：欧几里得范数的平方。`sum`(
(VectorXc)arg*1*）→复数：所有元素的和。

2.4.10 yade.mpy 模块

该模块定义了 mpirun()，它是一种采用分布式内存方式实现 run() 的并行实现。消息传递主要通过 mpi4py 完成，不过有些消息也会在 c++ 中处理（使用 OpenMPI）。

注意：本页列出的 mpy 模块的许多内部细节对用户并无帮助。请改为查阅用户手册中的 *mpy* 模块入门资料。

逻辑：

初始为集中式的场景，其逻辑如下：

1. 实例化一个完整的普通 Yade 场景
2. 将子域作为特殊的 Yade 刚体插入。这在某种程度上类似于在团簇成员顶部添加一个团簇刚体
3. 将此场景广播给所有工作进程。在初始化阶段，工作进程将：
   - 定义其分配的刚体的包围盒，并将其返回给其他工作进程
   - 检测哪些分配的刚体与其他域存在虚拟相互作用（基于其包围盒），并将列表发送给相关工作进程
   - 删除既未分配给该子域、也未与其发生虚拟相互作用的刚体
4. 运行若干次“常规”迭代，不再重新运行碰撞检测（Verlet距离机制）。在每次常规迭代中，工作进程将：
   - 计算内部相互作用和跨域相互作用
   - 对分配的刚体执行 Newton（修改后的 Newton 会跳过其他域）
   - 将更新后的位置发送给其他工作进程，并将地板上的部分力发送给主进程
5. 当一个工作进程触发碰撞检测时，所有工作进程都会执行。这样会更新子域之间的交集。
6. 如果启用，刚体可能会在碰撞检测后立即根据筛选器重新分配到不同的域。也可以使用自定义筛选器。此处预定义了一个筛选器（中位数筛选器）。

规则：

#- 交集[0] 包含 0 个刚体（需要向其发送力） #- 交集[thisDomain]包含与当前域重叠的其他域的标识符 #- 交集[otherDomain]包含_当前_ 域中与其他域重叠的刚体的标识符（需要向其发送更新后的位置/速度）

提示：

#- 请谨慎处理 subD.intersections（mirrorIntersections 同样如此）。 subD.intersections.append() 不会作用到 c++ object。subD.intersections 只能被赋值（一个由 int 列表组成的列表）

`yade.mpy.MAX_RANK_OUTPUT = 5`mprint 中

将跳过较大的进程秩

`yade.mpy.REALLOCATE_FILTER`(*i*、*j*、$giveAway$)根据子域的轴对齐包围盒中心点之间的中位数划分，返回“i”中要分配给“j”的刚体。如果 giveAway!=0 为正数或负数，“i”会将该数量的刚体交给“j”或从“j”获取，而不作任何回报（用于负载均衡）

`class yade.mpy.Timing_comm`(继承 *object*)

`Allgather`(计时_名称，*辐角，**_kwargs_)

Gather(计时_名称， *辐角， **kwargs)
Gatherv(计时_名称， *辐角，**kwargs)
allreduce(计时_名称，*辐角， **kwargs)

bcast(计时_名称，*args，**kwargs)

clear()

enable_timing()mpiSendStates(计时_名称,
*args， **kwargs)

mpiWait(计时_名称， *args，**kwargs)

mpiWaitReceived(计时_名称，*args， **kwargs)

print_all()

recv(计时_名称， *args， **kwargs)

send(计时_名称，*参数，**关键字参数)

yade.mpy.bodyErase(标识符)O.bodies.erase(id) 的并行版本，应以集体方式调用，否则分布式场景会彼此不一致（即使不包含该「id」的子域也可以安全调用）。为了提高性能，最好传入列表：bodyErase([i,j,k])。

yade.mpy.checkAndCollide()如果至少一个子域需要启用碰撞检测，则返回 true；否则返回 false。如果 COPY_MIR- ROR_BODIES_WHEN_COLLIDE 运行碰撞器，则在需要时运行，并在这种情况下返回 False。

yade.mpy.colorDomains()根据刚体所属的子域索引为其着色

yade.mpy.configure()导入 MPI 并定义上下文。configure 不会自行生成工作进程，这由 i nitialize() 完成。调用 configure() 前需要设置 OpenMPI 环境变量。

yade.mpy.declareMasterInteractive()
这表示当前处于交互会话，因此 TIMEOUT 将重置为 0（忽略）

yade.mpy.disconnect()
杀死全部 MPI 进程，仅保留进程等级 0 的 Python 解释器，就像单线程执行一样。工作进程中的场景会丢失，因为之后重新连接到 MPI 时只会生成新进程。主线程中的场景保持不变。

yade.mpy.eraseRemote()

yade.mpy.genLocalIntersections(子域)
定义当前域中与其他域重叠的刚体集合。
域 'k' 的数据结构为： [[id1, id2, ...], <———— 交集[0] = 域 k 中与主域交互的刚体标识符（不包括子域 k 本身） [id3, id4, ...], <—— —— 交集[1] = 域 k 中与域进程等级=1交互的刚体标识符（不包括子域 k 本身）..... [domain1, domain2, domain3, ...], <———— 交集[k] = 与域 k 交互的外部域的进程秩（不是标识符！）.....]

yade.mpy.genUpdatedStates(刚体_标识符)
返回要发送给其他工作进程的 [标识符,状态] （或在满足条件时返回 [标识符,状态,形状] ）列表

yade.mpy.initialize(*np*)

yade.mpy.isendRecvForces()
将力从子域传递到主进程。警告：发送端（除主进程外的所有进程）必须等待返回的请求列表

yade.mpy.makeColorScale(否=无)

yade.mpy.makeMpiArgv()

yade.mpy.**maskedConnection**(*b*，布尔数组)
有选择地列出面片内的刚体；boolArray 中标记为“True”的刚体（即已从另一个面片中选出的刚体）将被丢弃

yade.mpy.**maskedPFacet**(*b*，*boolArray*)有选择地列出面片内的刚体；boolArray 中标记为“True”的刚体（即已从另一个面片中选出的刚体）将被丢弃

yade.mpy.**medianFilter**(*i*、*j*、*giveAway*)根据子域轴对齐包围盒中心点之间的中位数划分，将“i”中的刚体分配给“j”。如果 giveAway!=0 为正数或负数，“i”将把相应数量的刚体交给“j”或从“j”获取，且不进行回馈（用于负载均衡）

yade.mpy.**mergeScene**()

yade.mpy.**migrateBodies**(*ids*、*origin*、*destination*)将刚体从原点重新分配到目标。原点（发送方）和目标（接收方）都必须调用此函数。注意：由于 subD.sendBodies() 为非阻塞函数，在一系列重新分配操作之后必须调用 subD.completeSendBodies()

yade.mpy.**mpiStats**()

yade.mpy.**mpirun**(*nSteps*，*np*=无，*withMerge*=否)使用 MPI 域分解的 O.run() 并行版本。

参数
nSteps：要计算的步数 np：MPI 工作进程的数量（主进程+子域），如果=1函数回退到 O.run() withMerge：运行结束时是否应将子域合并到主进程（默认否）。若为 True，主进程中的场景状态将与执行 O.run(nSteps,True) 后完全相同。合并可能非常耗时，建议仅在后处理或其他类似任务需要时启用。

yade.mpy.**mprint**(**args*，*force*=否)无论 mpy.VERBOSE_OUTPUT 如何，始终以反映进程秩的颜色打印，但仍受 rank<=mpy.MAX_RANK_OUTPUT 限制

yade.mpy.**pairOp**(*talkTo*)

yade.mpy.**parallelCollide**()

yade.mpy.**probeRecvMessage**(*source*，*tag*)

yade.mpy.**projectedBounds**(*i*、*j*)返回给定轴上边界投影的排序列表，其中边界取自 i->j 和 j->i 的交集

yade.mpy.**reallocateBodiesPairWiseBlocking**(_筛选器，其他域)
根据“_筛选器”参数，将刚体重新分配到其他域或从其他域移出。要求：“_筛选器”是一个函数，它接收原点和目标的进程秩，并返回要移动的刚体列表（按索引）。分解策略正是在此处定义的。参见示例 medianFilter（默认使用）。

yade.mpy.**reallocateBodiesToSubdomains**(_筛选器=<函数 *medianFilter*>，阻塞=*True*)根据“_筛选器”参数，将刚体重新分配到各子域。要求：“_筛选器”是一个函数，它接收原点和目标的进程秩，并返回要移动的刚体列表（按索引）。分解策略正是在此处定义的。参见示例 medianFilter（默认使用）。此函数必须并行调用，因此若以交互方式运行，必须显式发送 command：mp.sendCommand(“全部”,”reallocateBodiesToSubdomains(medianFilter)”,True)

yade.mpy.**reboundRemoteBodies**(标识符)
更新由其他工作进程处理的刚体的状态，参数‘状态’是一个由 [标识符,状态] （或有条件地为[标识符,状态,形状] ）组成的列表

yade.mpy.**receiveForces**(子域)
累积来自子域的力（仅由主进程执行），应在

ForceResetter 之后、Newton 之前，以及任何其他依赖力的引擎（例如 StressCon- troller）之前执行；可通过 Yade 的 PyRunner 插入。

yade.mpy.**recordMpiTiming**(名称，值)
将值追加到字典 timing.mpi 中由“名称”定义的值列表

yade.mpy.**runOnSynchronouslPairs**(工作进程，*command*)
从本地（单个工作进程的视角）来看，此函数会在其他工作进程列表（通常是交互子域列表）上运行由“command”定义的交互式 MPI 任务。总体而言，系统会建立点对点连接，使“command”在每对工作进程的两端对称且同时执行。也就是说，如果工作进程“i”使用参数“j”（另一个工作进程的索引）执行“command”，那么按设计，工作进程“j”也会使用参数“i”同时执行相同操作。

在许多情况下，可以用如下异步 irecv+send 更简单（且更快）地完成类似的一系列数据交换。

对于 w in 工作进程： m=comm.irecv(w) comm.send(数据,dest=w)

上述做法只有在所有消息都已在本地预先确定、且尚未开始任何通信时才有效。另一方面，如果与工作进程[1] 的相互作用取决于先前与工作进程[0]的一次相互作用的结果，就需要同步执行，因此需要这个函数。如果「command」中存在多个阻塞调用，也必须采用同步执行，否则就会发生显而易见的死锁，就像在那个简单循环中把「irecv」替换为「recv」一样。「medianFilter」算法中这两种情况都会出现，因此我们需要这个同步方法。

在此函数中，各工作进程以非监督且非确定的方式建立成对连接。每当建立一组交互式通信 (i,j) 时，i 和 j 会同时执行「command」。可以保证所有可能的配对都会被访问。

此函数可用于全对全操作（N^2 对），更有趣的是，它也适用于工作进程=交集[秩]（O(N) 对）。可以使用虚拟函数「pairOp」进行测试：runOnSynchronouslPairs(range(线程数量),pairOp)

command: 一个以另一工作进程的索引为参数的函数。由于 runOnSynchronouslPairs 保证另一工作进程会对称地运行该 command，因此可以包含与另一工作进程之间的阻塞通信。

yade.mpy.**sendCommand**(执行器，*command*，等待=真，工作进程间=假)从主进程或另一个工作进程向一个或多个工作进程发送 command。可接受的执行器包括“i”、“[i,j,k]”、“从属引擎”和“全部”（此时主进程也会执行该 command）。

yade.mpy.**sendRecvStates**()

yade.mpy.**shrinkIntersections**()
将交集和镜像交集缩减为与当前子域中另一个有状态刚体实际交互的刚体。这样可以减少发送和接收状态中的更新次数。初始列表已备份；在碰撞检测之前（参见检查并碰撞()），需要恢复这些列表（并更新全部状态）。

yade.mpy.**spawnedProcessWaitCommand**()

yade.mpy.**splitScene**()
将一个单体场景拆分为多个线程上的分布式场景。

前提条件：用户脚本已为刚体设置子域编号

yade.mpy.**unboundRemoteBodies**()根据秩启用或禁用包围盒

yade.mpy.**updateAllIntersections**()

yade.mpy.**updateDomainBounds**(子域)更新当前子域的边界，进行广播，并接收其他子域更新后的边界。前置条件：collider.boundDispatcher.__调用__()

yade.mpy.**updateMirrorOwners**()

yade.mpy.**updateRemoteStates**(状态，*setBounded=False*)更新由其他工作进程处理的刚体状态，参数'states'是由 [标识符、状态] （或[标识符、状态、形状]，视情况而定）组成的列表

yade.mpy.**waitForces**()
等待所有力发送至主进程。如果未设置 ACCUMULATE FORCES_，则主进程和所有线程的 O.freqs 均为空

yade.mpy.**wprint**(**args*)使用反映进程秩的颜色打印，仅当mpy.VERBOSE_OUTPUT=为 True（否则请参阅*mpy.mprint*），限制为秩<=mpy.MAX_RANK_OUTPUT

2.4.11 yade.pack 模块

创建堆积体，并填充由边界表示或构造实体几何定义的体积。

示例参见

- examples/gts-horse/gts-operators.py
- examples/gts-horse/gts-random-pack-obb.py
- examples/gts-horse/gts-random-pack.py
- examples/test/pack-cloud.py
- examples/test/pack-predicates.py
- examples/packs/packs.py
- examples/gts-horse/gts-horse.py
- examples/WireMatPM/wirepackings.py

yade.pack.**SpherePack_toSimulation**(*self*，旋转=*Matrix3(1，0，0，0，1，0，0，0，1)*，***kw*)将球体直接添加到模拟中。除了调用 *O.bodies.append*之外，此方法还会适当地设置模拟的周期单元信息。

```
>>> from yade import pack; from math import *
>>> sp=pack.SpherePack()
```

创建包含 20 个球体的随机周期堆积：

```
>>> sp.makeCloud((0,0,0),(5,5,5),rMean=.5,rRelFuzz=.5,periodic=True,num=20)20
```

初始模拟是非周期性的：

```
>>> O.reset()
>>> O.periodic
False
```

将生成的堆积体添加到模拟中，并沿 +z 旋转 45°

```
>>> sp.toSimulation(rot=Quaternion((0,0,1),pi/4),color=(0,0,1))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

周期性属性（包括旋转）已正确传递到模拟中（可以通过显式传入"hSize=O.cell.hSize"作为参数来避免这种情况）：

```
>>> O.periodic
True
>>> O.cell.refSize
Vector3(5,5,5)
```

（下页继续）

（接上页）

```
>>> O.cell.hSize # doctest: +SKIP
Matrix3(3.53553,-3.53553,0, 3.53553,3.53553,0, 0,0,5)
```

当前状态（即使经过旋转）也被视为机械上未变形的状态，即采用单位矩阵变换：

```
>>> O.cell.trsf
Matrix3(1,0,0, 0,1,0, 0,0,1)
```

参数

- **rot** （*Quaternion/Matrix3*）– 堆积体的旋转，将应用于球体，并用于设置 *Cell.trsf* 。
- ****kw**– 传递给 *utils.sphere*

返回 已添加刚体 ID 的列表（例如 *O.bodies.append*）

yade.pack.**filterSpherePack**(谓词，球体堆积，返回球体堆积=无，***kw*)使用给定的 SpherePack 实例，返回满足谓词条件的球体。返回结果可以是*pack.SpherePack* （如果 returnSpherePack 为真），也可以是一个列表。堆积体会重新居中以匹配谓词；如果谓词范围大于堆积体，则会发出警告。

yade.pack.**gtsSurface2Facets**(曲面，***kw*)根据给定的 GTS 曲面构造面片。
**kw 会传递给 utils.facet。

yade.pack.**gtsSurfaceBestFitOBB**(曲面)

Return（Vector3 center、Vector3 halfSize、Quaternion orientation），用于描述给定表面的最佳拟合定向 dboun包围盒（OBB）。详情请参阅 cloudBestFitOBB。

yade.pack.**hexaNet**(半径，角点坐标=*[0, 0, 0]*，*x* 方向长度=*1.0*，*y* 方向长度=*0.5*，*mos*=*0.08*，*a*=*0.04*，*b*=*0.04*，*startAtCorner*=*True*，*isSymmetric*=*False*，***kw*)用于 WireMatPM 的 x-y 平面六角形金属网颗粒定义。

参数

- **radius** – 颗粒半径
- **cornerCoord** – 网格左下角的坐标
- **xLenght**– 网格在 x 方向的长度
- **yLenght**– 网格在 y 方向的长度
- **mos** – 网孔尺寸（双绞结之间的水平距离）
- **a** – 双绞结的长度
- **b** – 单股线段的高度
- **startAtCorner** – 如果为 true，则从左下角的双绞结开始生成
- **isSymmetric** – 定义网格是否关于 y 轴对称

返回 定义网格（网格）及其精确尺寸（lx,ly）的球体集合。

注意： 此堆积体仅适用于 WireMatPM。角点处的颗粒始终最先生成。有关如何使用此堆积体的示例，请参见 examples/WireMatPM。要为网格创建正确的相互作用，必须在模拟中调整相互作用半径。

---

class yade.pack.**inConvexPolyhedron**(继承谓词)

aabb（（谓词）辐角 *1*）→元组 aabb
（（谓词）参数1）-> 无center(
（谓词）辐角 *1*）→ 三维向量dim(
（谓词）辐角 *1*）→ 三维向量

class yade.pack.inGtsSurface_py(继承谓词)

该类已在 C++ 中重新实现，但应保留在此，作为使用纯 Python 代码实现谓词的参考。C++ 使我们能够在 GTS 中使用 pygts 本身无法实现的巧妙技巧；pygts 的性能开销源于每次调用 gts.Point().is_inside() 时，都会为表面构建并销毁包围盒树。在 C++ 代码中，这会被缓存，前提是对象的生命周期内不对表面进行操作（这是用户的责任）。

—

用于 GTS 曲面的谓词。使用已有的曲面构造，这些曲面必须是封闭的。

导入 gts surf=gts.read(open('horse.gts')) inGtsSurface(surf)

注意： 可通过沿坐标轴在填充距离范围内测试 6 个点来启用填充。必须在构造函数中设置 doSlowPad=True 才能启用此功能。如果未启用且填充不为零，则会发出警告。

---

aabb((谓词)辐角 *1*）→元组 aabb((谓词)参数1) -> 无

center((谓词)辐角 *1*）→ 三维向量

dim((谓词)辐角 *1*）→ 三维向量

class yade.pack.inHalfSpace(继承谓词)

返回 True 的谓词，适用于任意点，且具有无穷大的包围盒。

aabb((谓词)辐角 *1*）→元组
aabb((谓词)参数1) -> 无

center((谓词)辐角 *1*）→ 三维向量

dim((谓词)辐角 *1*）→ 三维向量

class yade.pack.inSpace(继承谓词)

对任意点都返回 True 的谓词，其包围盒为无穷大。

aabb((谓词)辐角 *1*）→元组 aabb((谓词)参数1) -> 无

center((谓词)*arg 1*）→ 三维向量

dim((谓词)*arg 1*）→ 三维向量

yade.pack.randomDensePack(谓词，半径，材料=*-1*，维度=无，裁剪层数=*0*，相对半径-模糊度=*0.0*，单元内球体=*0*，记忆化数据库=无，使用定向包围盒=否，记忆化调试=否，颜色=无，返回球体堆积=无，随机种子=*-1*)

使用指定几何属性生成随机致密堆积的生成器，可通过 TriaxialTest（非周期性）或 PeriIsoCompressor（周期性）实现。周期性取决于是否提供 spheresInCell 参数。

O.switchScene() magic 用于为 TriaxialTest 创建干净的模拟，而不删除原始模拟。因此，此函数绝不能与访问模拟的代码并行运行。

参数

- predicate – 用于生成堆积体的实体定义谓词

- **spheresInCell** – 如果指定，堆积体将具有周期性，且周期单元中的球体数量为指定值。
- **radius** – 球体的平均半径
- **rRelFuzz** – 半径的相对扰动，例如半径为=10、rRelFuzz为=.2时，球体半径将为 $10 \pm (10*.2)$，且服从均匀分布。默认值为 0，表示所有球体的半径完全相同。
- **cropLayers** –（仅适用于非周期性情况）在计算得到的盒子维度上额外添加的球体层数，以消除谓词边界处的边界效应（或至少尽可能减小其影响）。
- **dim** – 堆积体的维度，用于覆盖谓词的尺寸（例如，谓词为无穷时）
- **memoizeDb** – SQLite 数据库的名称（可以是已存在或不存在的数据库），用于查找已生成的堆积体；如果找不到，则存储即将生成的堆积体（缓存高开销计算结果的技术称为记忆化）。此处使用模糊匹配来选择合适的候选项——将对堆积体进行缩放，并比较相对半径扰动和尺寸。过小的堆积体会被舍弃。在剩余候选项中，将加载并返回球体数量最少的堆积体。
- **useOBB** – 仅当给定 inGtsSurface 谓词时才有效。如果为 true（非默认值），将首先计算定向包围盒；这可以大幅减少三轴压缩所需的球体数量（具体取决于刚体的不对称程度，例如 $10\times$ ），参见 examples/gts-horse/gts-random-pack-obb.py
- **memoDbg** – 显示被考虑的堆积体，以及拒绝或接受它们的原因
- **returnSpherePack** – 请参见 *pack.filterSpherePack* 中对应的参数

返回 由谓词筛选出的、包含球体的 SpherePack 对象。

yade.pack.**randomPeriPack**(半径，*initSize*，相对半径扰动=*0.0*，记忆化数据库=*None*，不打印=*False*，随机种子=*-1*)生成周期性密集堆积。

在 initSize 大小的单元中尽可能多地填入球体，然后使用 PeriIsoCompressor 进行周期压缩，就像使用 randomDensePack 一样。

参数

- **radius** – 平均球半径
- **rRelFuzz** – 球半径相对扰动（均匀分布）；参见 randomDensePack 中的相同参数。
- **initSize** – 周期单元初始尺寸。

返回SpherePack 对象，其中还包含周期性信息。

yade.pack.**regularHexa**(谓词、半径、间隙、***kw*)返回由谓词指定的实体内部裁剪得到的正六边形网格中的球体集合。创建的球体将具有给定半径，并且彼此之间由间隙空间隔开。

yade.pack.**regularOrtho**(*predicate, radius*，间隙，***kw*)
返回由谓词定义的实体内部裁剪后的正交规则网格中的球体集合。生成的球体具有给定半径，彼此之间留有间隙空间。

yade.pack.**revolutionSurfaceMeridians**(*sects*，角度，原点=三维向量*(0, 0, 0)*，方向=四元数*((1, 0, 0), 0)*)
给定二维点序列及其对应的角度序列，生成表示旋转曲面子午截面的三维点序列。二维截面绕z轴旋转，但可以使用原点和方向参数进行变换，以得到任意方向。

yade.pack.**sweptPolylines2gtsSurface**(点、阈值=*0*、*capStart=False*、*capEnd=False*)给定长度相同的点序列（以三元组表示），创建扫掠表面（作为 GTS 三角剖分）。

如果指定了阈值（>0），那么

- 将不会生成退化面（边长小于阈值的面）
- 返回前将调用 gts.Surface().cleanup(threshold)，合并彼此距离小于阈值的顶点。如果 pts 是闭合的（最后一个点与第一个点重合），则会生成三角形曲面带。如果另外设置 capStart==True 和 capEnd==True，表面将闭合。

注意： capStart 和 capEnd 会采用最简单的多边形三角剖分方式（对角线），可能无法处理非凸截面。

警告： 该算法会按顺序连接各点；如果两条折线彼此旋转或方向相反，算法不会检测到这一点，而会按给定顺序将它们连接起来。

通用球体堆积的创建、操作与输入/输出。

class yade._packSpheres.**SpherePack**

由中心和半径表示的球体集合。此类由 *pack.randomDensePack*、*pack.randomPeriPack* 等返回。该对象支持对球体进行迭代，例如

```
>>> sp=SpherePack()>>> for center,
radius in sp: print center,radius
```

```
>>> for sphere in sp: print sphere[0], sphere[1]    ## same,
 but without unpacking the␣→tuple automatically
```

```
>>> for i in range(0,len(sp)): print sp[i][0], sp[i][1]    ## same, but accessing spheres␣
 →by index
```

特殊构造函数

从 `[(c1,r1),(c2,r2),…]` 列表构造。要转换两个长度相同的 `centers` 和 `radii` 列表，请使用 `zip(centers,radii)` 构造。

`__init__`(*(object)*辐角 *1*[, *(list)*列表]) → None：空构造函数，可选地接受用于初始数据的列表 [((cx,cy,cz),r), ...]。

`aabb`(*(SpherePack)*辐角 *1*) → 元组：获取轴对齐包围盒坐标，表示为两个三元组。

`add`(*(SpherePack)*参数*1*，*(Vector3)*参数*2*，*(float)*辐角 *3*) → None：向堆积体添加单个球体，给定其中心（三元组）和半径

`appliedPsdScaling`

介于 0 和 1 之间的因子，均匀应用于粒径分布的所有尺寸。

`cellFill`(*(SpherePack)*参数*1*，*(Vector3)*辐角 *2*) → None：重复堆积体（如果是周期性的），使结果具有给定尺寸的 dim() >= 。堆积体保留周期性，但单元尺寸会改变。非周期性堆积会引发异常。

`cellRepeat`(*(SpherePack)*参数*1*，*(Vector3i)*辐角 *2*) → 无：在每个维度上将给定的堆积体重复指定次数。保留周期性，单元尺寸会发生变化。非周期性堆积体会引发异常。

cellSize
周期单元的尺寸；如果不是周期性的，则为 Vector3(0,0,0)。（只有在清楚自己在做什么时才更改此属性。）

center(*(SpherePack)*辐角 *1*) → 三维向量：返回包围盒中心的坐标。

dim(*(SpherePack)*辐角 *1*) → 三维向量：以 aabb() 的形式返回堆积体的尺寸，表示为一个 3 元组。

fromList(*(SpherePack)*参数*1*,*(*列表*)*辐角 *2*) → 无：根据给定列表创建堆积体，格式与构造函数相同。丢弃当前数据。

fromList((SpherePack)参数1, (对象)中心, (对象)半径) -> 无： 根据给定列表创建堆积体，格式与构造函数相同。丢弃当前数据。

fromSimulation(*(SpherePack)*辐角 *1*) → 无：创建与当前仿真相对应的堆积体。丢弃当前数据。

getClumps(*(SpherePack)*辐角 *1*) → 元组：返回按所属团簇排序的球体 ID 列表。返回值为 (独立球体,[团簇1,团簇2,...])，其中每一项都是球体 ID 列表。

hasClumps(*(SpherePack)*辐角 *1*) → 布尔值：
此对象是否包含团簇。

isPeriodic
堆积体是否在周期边界中生成?

load(*(SpherePack)*参数*1*，*(str)*文件名) → 无：
从外部文本文件加载堆积体（当前数据将被丢弃）。

makeCloud(*(SpherePack)*参数*1*[, *(Vector3)minCorner=Vector3(0, 0, 0)*[, *(Vec- tor3)*最大角点=三维向量*(0, 0, 0)*[, *(float)rMean=-1*[, *(*浮点数*)*相对半径扰动*=0*[, *(*整数*)*数量*=-1*[, *(*布尔值*)*周期性*=False*[, *(*浮点数*)*孔隙率*=0.65*[, *(object)psdSizes=[]*[,*(*对象*)*粒径分布累积分数*=[]*[, *(bool)distributeMass=False*[, *(*整数*)*随机种子*=-1*[, *(Matrix3)hSize=Matrix3(0, 0, 0, 0, 0, 0, 0, 0, 0)*]]]]]]]]]]]]) → 整数：
创建一个由平行六面体包围的随机颗粒云。生成的堆积体初始时处于类气态，颗粒之间没有接触。通常用作达到密堆积之前的第一步。

参数

- **minCorner** (Vector3) – 轴对齐包围盒的下角点
- **maxCorner** (Vector3) – 轴对齐包围盒的上角点
- **hSize** (Matrix3) – 广义盒（任意平行六面体）的基向量，通常为*Cell::hSize*；若已定义，则优先于 minCorner 和 maxCorner。仅适用于周期边界。
- **rMean**(*float*) – 球体的平均半径
- **rRelFuzz** (*float*) – 半径相对于平均半径的离散程度
- **num** (*int*) – 要生成的球体数量。如果为负值（默认值），则使用随机尺寸尽可能多地生成球体，并在尝试将球体放入空间达到固定次数后停止；否则，使用确定性尺寸分布，准确生成 num 个球体。
- **periodic** (*bool*) – 要生成的堆积体是否应具有周期性
- **porosity** (*float*) – 迭代生成过程的初始猜测值（如果num>1）。算法会不断重试，直到生成球体的数量达到 num。第一次迭代会使用提供的孔隙率，后续迭代则

在必要时提高孔隙率（因此，较高的初始孔隙率可以加快算法的速度）。如果未定义 `psdSizes`，则会使用 `rRelFuzz` (z) 和 `num` (N)，使给定的孔隙率（ρ）在生成结束时大致达到目标值，$r_m = \sqrt[3]{\frac{V(1-\rho)}{\frac{4}{3}\pi(1+z^2)N}}$。默认值为 ρ=0.5。最佳值取决于rRelFuzz或 psdSizes。

- `psdSizes` – 指定粒径分布 (PSD) 时使用的筛分尺寸（颗粒直径）。
- `psdCumm` – 由 `psdSizes` 给出的颗粒尺寸累积分数；其长度必须与 *PSD*尺寸 相同，且应为非递减序列。
- `distributeMass` (*bool*)– 如果为 `True`，则使用给定的发行版来分配球体质量，而不是分配球体半径。
- `seed` – 用于初始化随机数生成器的数值。

返回 所创建球体的数量；根据所用方法，该数量可能小于 `num`。

注意：

- 当 `minCorner[k]=maxCorner[k]` 针对一个坐标时，可用于二维。
- 如果定义了 `num`，尺寸生成便具有确定性，可实现与目标分布的最佳拟合。它支持按尺寸降序放置球体，因此生成的孔隙率低于随机生成。
- 默认情况下（使用 `distributeMass==False` 时），该分布应用于颗粒数（即通过筛分的颗粒数百分比）。岩土力学中“粒径分布”的典型含义是质量分数 的分布（即通过筛分的质量百分比）；使用 `distributeMass=True` 即可实现这一点。
- 球体半径分布可通过以下任一方式指定：1. `rMean`、 `rRelFuzz` 和 `num` 可在 $rMean\times(1\pm rRelFuzz)$中生成均匀半径分布。如果其值过高，生成的球体可能少于 `num`。2. `rRelFuzz`、 `num` 以及（可选） `porosity`，用于估算平均半径，使最终达到 `porosity`。 `rMean` 必须小于 0（默认值）。 `porosity`只是生成算法的初始猜测值；算法会在孔隙率较高的情况下重试，直到达到指定数量。3. `psdSizes` 和 `psdCumm`，这两个数组用于指定粒径分布函数的点。系统会尽可能多地生成球体。4. `psdSizes`、 `psdCumm`、 `num`以及（可选） `porosity`，与上述方式类似；但如果未达到 `num`，则会将 `psdSizes` 均匀缩小，直至达到 `num` （参见 appliedPsdScaling）。

`makeClumpCloud`(*(SpherePack)*参数*1*，*(*三维向量*)*最小角点，*(*三维向量*)*最大角点，*(*对象*)*团簇[,*(*布尔值*)*周期性=*False*[,*(*整数*)*数量=*-1*[,*(*整数*)*随机种子=*-1*]]]) → 整数

：以与 makeCloud 生成球体相同的方式，创建随机疏松堆积的团簇。参数 `minCorner`、 `maxCorner`、 `periodic`、 `num` 和 `seed` 与 makeCloud中的相同。参数 `clumps` 是一个列表，其中包含所有要追加为 `SpherePack`对象的不同团簇。下面的示例展示如何创建由 10 个相同团簇构成的球体云：

```
clp = SpherePack([((0,0,0), 1e-2), ((1e-2,0,0), 1e-2)]) # The clump we want a cloud␣
→of
sp = SpherePack()
sp.makeClumpCloud((0,0,0), (1,1,1), [clp], num=10, seed=42)
sp.toSimulation() # All the particles in the cloud are now appended to O.bodies
```

psd(*(SpherePack)*辐角 *1*[, *(int)*分箱 *=50*[, *(bool)*质量 *=True*]]) → 元组：
返回堆积体的粒径分布。

参数

- **bins** (*int*) – 最小直径与最大直径之间的分箱数量
- **mass** – 对每个分箱计算相对质量，而非相对颗粒数。对应于 *makeCloud* 的按质量分布参数。

返回 (cumm,edges)元组，其中 cumm 是相应直径的累积分数， edges 是这些直径值。两个数组的维度均为 bins+1。

relDensity(*(SpherePack)*辐角 *1*) →浮点数：相对堆积密度，计算方式为球体体积总和 / 轴对齐包围盒体积。（忽略球体重叠。）

rotate(*(SpherePack)*参数*1*，*(Vector3)*轴，*(float)*角度）→ None：根据给定的旋转轴和角度，围绕堆积体中心（以 aabb() 为准）旋转全部球体。

save(*(SpherePack)*参数*1*，*(str)fileName*) → None：将堆积体保存到外部文本文件（现有内容将被覆盖）。

scale(*(SpherePack)arg2*) → None：围绕堆积体中心（以 aabb() 为准），按给定因子缩放堆积体（因子可以为负数）。

toList(*arg1*) → 列表：以 Python 列表形式返回堆积体数据。

toSimulation(旋转 *=Matrix3(1,0,0,0, 1, 0, 0, 0, 1), **kw*)将球体直接添加到模拟中。除了调用 *O.bodies.append*之外，此方法还会恰当地设置模拟的周期单元信息。

```
>>> from yade import pack; from math import *
>>> sp=pack.SpherePack()
```

创建包含 20 个球体的随机周期堆积：

```
>>> sp.makeCloud((0,0,0),(5,5,5),rMean=.5,rRelFuzz=.5,periodic=True,num=20)20
```

初始模拟是非周期性的：

```
>>> O.reset()
>>> O.periodic
False
```

将生成的堆积体添加到模拟中，并绕 +z 旋转 45°

```
>>> sp.toSimulation(rot=Quaternion((0,0,1),pi/4),color=(0,0,1))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

周期性属性已正确传递到模拟中，包括旋转（通过显式传入“hSize=O.cell.hSize”作为参数即可避免这种情况）：

```
>>> O.periodic
True
>>> O.cell.refSize
Vector3(5,5,5)
>>> O.cell.hSize # doctest: +SKIP
Matrix3(3.53553,-3.53553,0, 3.53553,3.53553,0, 0,0,5)
```

当前状态（即使经过旋转）也被视为机械上未变形的状态，即采用单位矩阵变换：

```
>>> O.cell.trsf
Matrix3(1,0,0, 0,1,0, 0,0,1)
```

参数

- **rot** (*Quaternion/Matrix3*) – 堆积体的旋转，该旋转将应用于球体，并用于设置 *Cell.trsf* 。
- ****kw** – 传递给 *utils.sphere*

返回 所添加刚体的 ID 列表（类似于 *O.bodies.append*）

translate(*(SpherePack)*参数*1*，*(*三维向量*)*辐角*2*）→ 无:
用给定向量平移所有球体。

class yade._packSpheres.SpherePackIterator

__init__（（对象）参数*1*，（球体包迭代器）辐角*2*）→ 否
next()___下一个___((SpherePackIterator)参数1) -> 元组

用于体积（通过解析方式或三角剖分定义）的空间谓词。

class yade._packPredicates.Predicate

aabb（（谓词）辐角*1*）→元组
aabb((谓词)参数1) -> 否center(（谓词）辐角*1*）→ 三维向量dim(（谓词）辐角*1*）→ 三维向量

class yade._packPredicates.PredicateBoolean(继承自 *Predicate*)对两个谓词执行布尔运算（抽象类）

AB__init__()引发异常：无法从 Python 实例化此类aabb（（谓词）辐角*1*）→元组 aabb((谓词)参数1) -> 否center(（谓词）辐角*1*）→ 三维向量dim(（谓词）辐角*1*）→ 三维向量

class yade._packPredicates.PredicateDifference(继承自 *PredicateBoolean* → *Predicate*)两个谓词的差集（与负谓词的合取）。点必须位于第一个谓词内部且位于第二个谓词外部。可以使用谓词上的 - 运算符构造： pred1- pred2。

AB__init__(*(*对象*)*参数*1*，*(*对象*)*参数*2*，*(*对象*)*辐角*3*）→ 无aabb(*(*谓词*)*辐角*1*）→元组 aabb((谓词)参数1) -> 无center(*(*谓词*)*辐角*1*）→ 三维向量

dim(*(*谓词*)*辐角 *1*) → 三维向量

class yade._packPredicates.PredicateIntersection(继承布尔谓词 → 谓词)两个谓词的交集（合取）。点必须同时位于两个谓词内部。可使用谓词上的 & 运算符构造： pred1 & pred2。

AB__init__(*(*对象*)*参数*1*，*(*对象*)*参数*2*，*(*对象*)*辐角*3*) →
无aabb(*(*谓词*)*辐角*1*) →元组 aabb((谓词)参数1) -> 无
center(*(*谓词*)*辐角*1*) → 三维向量dim(*(*谓词*)*辐角*1*) → 三
维向量

class yade._packPredicates.PredicateSymmetricDifference(继承布尔谓词 →谓词)

两个谓词的对称差（异或）。点必须恰好位于这两个谓词中的一个内部。可使用谓词上的 ^ 运算符构造: pred1 ^ pred2。

AB__init__(*(*对象*)*参数*1*，*(*对象*)*参数*2*，*(*对象*)*辐角*3*) →
无aabb(*(*谓词*)*辐角*1*) →元组 aabb((谓词)参数1) -> 无
center(*(*谓词*)*辐角*1*) → 三维向量dim(*(*谓词*)*辐角*1*) → 三
维向量

class yade._packPredicates.PredicateUnion(继承布尔谓词 → 谓词)两个谓词的并集（非互斥析取）。点只要位于两个谓词中的任意一个范围内，就算位于并集内。可使用谓词上的 | 运算符构造： pred1 | pred2。

AB__init__(*(*对象*)*参数*1*、*(*对象*)*参数*2*、*(*对象*)*参数*3*) → 无aabb(*(*谓词*)*参数*1*) →元组 *aabb((*谓词*)*参数*1)* -> 无center(*(*谓词*)*参数*1*) → 三维向量dim(*(*谓词*)*参数*1*) → 三维向量
class yade._packPredicates.inAlignedBox(继承谓词)轴对齐包围盒谓词__init__(*(*对象*)*参数*1*、*(*三维向量*)*最小*AABB*、*(*三维向量*)*最大*AABB*) → 无：构造函数，接受盒子的最小点和最大点（以三元组表示）。aabb(*(*谓词*)*参数*1*) →元组 *aabb((*谓词*)*参数*1)* -> 无center(*(*谓词*)*参数*1*) → 三维向量dim(*(*谓词*)*参数*1*) → 三维向量
class yade._packPredicates.inCylinder(继承谓词)圆柱谓词__init__(*(*对象*)*参数*1*、*(*三维向量*)*底部中心、*(*三维向量*)*顶部中心、*(*浮点数*)*半径) → 否

一：

接收侧壁中心（以三元组表示）和半径的构造函数。

aabb((谓词)arg*1*）→元组 aabb((谓词)arg1) -> 无**center**((谓词)arg*1*）→ 三维向量**dim**((谓词)arg*1*）→ 三维向量**class yade._packPredicates.inEllipsoid**(继承谓词)椭球谓词**__init__**((对象)arg1, (三维向量)中心点, (三维向量)abc）→ 无：接收椭球中心（三元组）及其 3 个半径（三元组）的构造函数。**aabb**((谓词)arg*1*）→元组 aabb((谓词)arg1) -> 无**center**((谓词)arg*1*）→ 三维向量**dim**((谓词)arg*1*）→ 三维向量**class yade._packPredicates.inHyperboloid**(继承谓词)双曲面谓词**__init__**((对象)arg1, (三维向量)底部中心, (三维向量)顶部中心, (浮点数)半径, (浮点数)腰部半径）→无：接收侧壁中心（以三元组表示）、底面半径和腰部半径（中间半径）的构造函数。**aabb**((谓词)arg*1*）→元组 aabb((谓词)arg1) -> 无**center**((谓词)arg*1*）→ 三维向量**dim**((谓词)arg*1*）→ 三维向量**class yade._packPredicates.inParallelepiped**(继承谓词)平行六面体谓词**__init__**((对象)arg1, (三维向量)o, (三维向量)a, (三维向量)b, (三维向量)c）→ 无：接收四个点的构造函数： o （作为原点），然后是 a、 b和 c ， 它们分别定义从 o出发的 3 条边的端点。**aabb**((谓词)arg*1*）→元组 aabb((谓词)arg1) -> 无**center**((谓词)arg*1*）→ 三维向量**dim**((谓词)arg*1*）→ 三维向量**class yade._packPredicates.inSphere**(继承谓词)球体谓词。**__init__**((对象)arg1, (三维向量)中心, (浮点数) 半径 ）→ 无：接收中心（以三元组表示）和半径的构造函数 **aabb**((谓词)arg*1*）→ 元组 aabb((谓词)arg1) -> 无**center**((谓词)arg*1*）→ 三维向量**dim**((谓词)arg*1*）→ 三维向量**class yade._packPredicates.notInNotch**(继承谓词)无限矩形缺口外部谓词**__init__**((对象)arg1, (三维向量)中心点, (三维向量)边, (三维向量)法向量, (浮点数)开口）→无：接收对称面上的点、沿边指向的向量、平面法向量和开口尺寸的构造函数。缺口内侧为边×法向量。法向量会被处理为垂直于边。所有向量都会在构造时进行归一化。

aabb(*(谓词)辐角 1*) →元组 aabb((谓词)参数1) -> 无center(*(谓词)辐角 1*) → 三维向量dim(*(谓词)辐角 1*) → 三维向量

计算点云的定向包围盒。

yade._packObb.cloudBestFitOBB(*(元组)辐角 1*) → 元组 返回给定点元组的最佳拟合定向包围盒 (Vector3 中心, Vector3 半尺寸, Quaternion 方向)（使用蛮力体积最小化算法，不适用于非常大的云团）。

2.4.12 yade.plot 模块

包含用于在 yade 中绘图的实用函数的模块。请参阅 examples/simple-scene/simple-scene-plot.py或 examples/concrete/uniax.py 以查看使用示例。

yade.plot.data = {'eps': [0.0001, 0.001, nan], 'force': [nan, nan, 1000.0], 'sigma': [12, nan, nan]}
包含所有绘图共用的全部数据值的全局字典，格式为 {'name':[value,...],...}。应使用 plot.addData 函数添加数据。所有 [value,...]列的长度都相同；若未指定，则以非数填充。

yade.plot.plots = {'i': ('t',), 'i ': ('z1', 'v1')}字典 x-name -> (yspec,...)，其中 yspec 为 y-name 或 (y-name,'line-specification')。如果(yspec,...) 为 None，则绘图表示图像，图像将从 *plot.imgData* 的相应字段中获取。

yade.plot.labels = {}
将数据中的名称转换为可读名称（例如 TeX 名称）的字典；如果未指定某个变量，则保留其原名。

yade.plot.live = True
启用/禁用实时绘图更新。

yade.plot.liveInterval = 1
实时绘图更新的区间，单位为秒。

yade.plot.setLiveForceAlwaysUpdate(力实时更新)
该*plot.liveInterval* 和*plot.live* 控制计算期间绘图的实时刷新。刷新在独立线程中完成，因此不会干扰计算。如果数据恰好在另一个线程中更新，就无法同时绘制这些数据。若要在绘图更新期间暂停计算，请使用yade.plot.setLiveForceAlwaysUpdate(True) 。此函数在调用成功时返回当前 bool 强制更新的值，否则返回 str 并说明失败原因。如果通过*O.pause()* 调用暂停了模拟，则保证此函数可以正常工作。

yade.plot.autozoom = True
启用/禁用数据更新后的自动重新缩放绘图。有时必须跳过重新缩放，除非调用 *plot.setLiveForceAlwaysUpdate* 强制执行。

yade.plot.plot(不显示=否，子图=是)执行实际绘图：绘图可以显示在屏幕上（此时不返回任何内容；如果不显示为 False，则请确保编译的 Yade 支持 Qt 4 功能，以便显示图形），也可以在不显示为 True时，作为 matplotlib 的 Figure 对象或对象列表返回。

您可以使用

```
>>> from yade import plot
>>> plot.resetData()
>>> plot.plots={'foo':('bar',)}
>>> plot.plot(noShow=True).savefig('someFile.pdf')
>>> import os
```

（下页继续）

（接上一页）

```
>>> os.path.exists('someFile.pdf')
True
>>> os.remove('someFile.pdf')
```

以自动将图形保存到文件。

注意： 出于向后兼容的原因，*noShow* 选项在存在多个图形时会返回图形列表，但如果只有一个图形，则会返回单个图形（而不是包含 1 个元素的列表）。

---

yade.plot.**reset**()
重置所有绘图相关变量（数据、图形、标签）

yade.plot.**resetData**()
重置所有绘图数据；保留图形和标签不变。

yade.plot.**splitData**()
在此点将全部图形设为不连续（向全部数据字段添加非数）

yade.plot.**reverseData**()
反转 yade.plot.data 的顺序。

适用于拉压试验：初始（零）状态会被加载，而为了使数据连续，最后一部分必须结束 于零状态。

yade.plot.**addData**(**d* _在，***kw*)
将参数 name1=value1,name2=value2 中的数据添加到 yade.plot.data。（旧格式 {'name1':value1,' name2':value2} 已弃用，但仍受支持）

新数据将以非数填充，未指定的数据也将设为非数（非数不会出现在图表中）。这样可确保所有数据长度相等，因此可以将它们彼此绘图。

```
>>> from yade import plot
>>> from pprint import pprint
>>> plot.resetData()
>>> plot.addData(a=1)
>>> plot.addData(b=2)
>>> plot.addData(a=3,b=4)
>>> pprint(plot.data)
{'a': [1, nan, 3], 'b': [nan, 2, 4]}
```

可以向 addData 传入某些序列类型；它们将被保存到合成列中，以存储各个分量。

```
>>> plot.resetData()
>>> plot.addData(c=Vector3(5,6,7),d=Matrix3(8,9,10, 11,12,13, 14,15,16))
>>> pprint(plot.data){'c_x': [5.0],'c_y': [6.0],'c_z': [7.0],'d_xx': [8.0],
'd_xy': [9.0],'d_xz': [10.0],'d_yx': [11.0],'d_yy': [12.0],'d_yz': [13.0],
'd_zx': [14.0],'d_zy': [15.0],'d_zz': [16.0]}
```

yade.plot.**addAutoData**()

通过计算*plot.plots*的内容来添加数据。发生异常的表达式将得到妥善处理，但每个表达式都会打印一条警告。

```
>>> from yade import plot
>>> from pprint import pprint
>>> O.reset()
>>> plot.resetData()
>>> plot.plots={'O.iter':('O.time',None,'numParticles=len(O.bodies)')}
>>> plot.addAutoData()
>>> pprint(plot.data)
{'O.iter': [0], 'O.time': [0.0], 'numParticles': [0]}
```

请注意，*plot.plots* 中的每一项都可以是

- 一个要计算的表达式（使用 `eval` 内置函数）；
- `name=expression` 字符串，其中 `name` 将作为图形中的标签，而表达式会在每次绘图时计算；
- 一个类字典对象——当前的键是图形的标签，当前的值会添加到*plot.data*中。字典的内容可以随时间变化，此时会根据需要创建新的直线。

带有绘图功能的简单模拟可以按以下方式编写；请注意能量图的指定方式。

```
>>> from yade import plot,  utils
>>> plot.plots={'i=O.iter':(O.energy,None,'total energy=O.energy.total()')}
>>> # we create a simple simulation with one ball falling down
>>> plot.resetData()>>> O.bodies.append(utils.sphere((0,0,0),1))0
>>> O.dt=utils.PWaveTimeStep()>>> O.engines=[...     ForceResetter(),
...    GravityEngine(gravity=(0,0,-10),warnOnce=False),
...    NewtonIntegrator(damping=.4,kinSplit=True),
...    # get data required by plots at every step
...    PyRunner(command='yade.plot.addAutoData()',iterPeriod=1,initRun=True)
... ]>>> O.trackEnergy=True>>> O.run(2,True)
>>> pprint(plot.data)   #doctest: +ELLIPSIS
{'gravWork': [0.0, -25.13274...],'i': [0, 1], 'kinRot': [0.0, 0.0],
'kinTrans': [0.0, 7.5398...], 'nonviscDamp': [0.0, 10.0530...],
'total energy': [0.0, -7.5398...]}
```

yade.plot.saveGnuplot(基本文件名，终端='wxt'，扩展名=无，时间戳=否，注释=无，标题=无，变量数据=否)

将 *plot.addData* 添加的数据保存到（压缩）文件中，并创建 .gnuplot 文件，尝试模仿 *plot.plots* 中指定的绘图。

参数

- `baseName` – 用于创建 baseName.gnuplot（gnuplot 的命令文件）、相关的数据文件和输出文件（如适用），格式为`baseName.[plot number].extension`
- `term` – 指定 gnuplot 终端；默认值为 `x11`，此时 gnuplot 会在屏幕上绘制持久窗口，然后终止；其他常用终端包括 `png`、`cairopdf` 等

- **extension** – baseName 的扩展名，默认为终端名称，例如 png；不过，如果使用 cairopdf，还应指定 extension='pdf'
- **timestamp** (*bool*) – 将数值时间附加到基本文件名
- **varData** (*bool*) – 指定要绘图的文件是声明为变量，还是直接写入绘图表达式
- **comment** – 用户注释（可以跨多行），将嵌入控制文件中

返回所创建的 gnuplot 文件的名称。

yade.plot.saveDataTxt(文件名，变量=*None*，表头=*None*)

将绘图数据保存到文本文件（可选压缩）中。第一行包含一条注释（以 # 开头），其中为每一列指定变量名。此格式适用于使用 numpy.genfromtxt 函数加载数据以进行后续处理（在 Yade 外部）；该函数可识别这些变量名（创建带有命名条目的 NumPy 数组），并透明地处理解压缩。

```
>>> from yade import plot
>>> from pprint import pprint
>>> plot.reset()
>>> plot.addData(a=1,b=11,c=21,d=31)  # add some data here
>>> plot.addData(a=2,b=12,c=22,d=32)
>>> pprint(plot.data)
{'a': [1, 2], 'b': [11, 12], 'c': [21, 22], 'd': [31, 32]}
>>> plot.saveDataTxt('/tmp/dataFile.txt.tar.gz',vars=('a','b','c'))
>>> import numpy
>>> d=numpy.genfromtxt('/tmp/dataFile.txt.tar.gz',dtype=None,names=True)
>>> d['a']
array([1, 2])
>>> d['b']
array([11, 12])
>>> import os # cleanup
>>> os.remove('/tmp/dataFile.txt.tar.gz')
```

参数

- **fileName** – 保存数据的文件；如果文件名以 .bz2 / .gz 结尾，则会分别使用 bzip2 / gzip 压缩文件。
- **vars** – 要保存的变量名序列（元组/列表/集合）。如果 None （默认值），则会保存 *plot.plot* 中的全部变量。
- **headers**要写入头文件的参数集合

yade.plot.savePlotSequence(文件基名,步长=*1*, 图像宽高比=*(5, 7)*, 标题=无, 标题帧数=*20*,最后帧数=*30*)

保存绘图序列，每幅图对应历史记录中的一条直线。此功能尤其适用于*utils.makeVideo*。

参数

- **stride** – 仅考虑历史记录中每隔 stride 条的直线（默认情况下，每条直线生成一帧）
- **title** – 创建标题帧，标题各行之间以换行符分隔（\n），可选副标题与标题之间以两个换行符分隔。
- **titleFrames** (*int*) – 创建指定数量的标题帧（通过重复其文件名），决定标题在影片中的显示时长。
- **lastFrames** (*int*) – 将最后一帧重复指定次数，使影片结尾不至于过于突然。

返回 包含连续帧的文件名列表。

2.4.13 yade.polyhedra_utils 模块

2.4.14 yade.post2d 模块

用于二维后处理的模块，包含将点从三维投影到二维的各类方法，并提供基本但灵活的框架，用于从刚体/相互作用中提取任意标量值并绘制结果。该模块包含两个基本组件：展平器和提取器。

这些算法根据 post2d.data 的 `intr` 参数，作用于刚体（默认）或相互作用。

展平器

将三维（模型）坐标转换为二维（绘图）坐标的类实例。其接口由 *post2d.Flatten* 类（`__call__`、`planar`、`normal`）定义。

提取器

给定刚体/相互作用对象后，返回标量值或向量值的可调用对象。如果返回的是三维向量，则会调用 Flattener.planar，它应该只返回该向量的平面内分量。

示例

此示例位于 examples/concrete/uniax-post.py

```
from yade import post2d
import pylab # the matlab-like interface of matplotlib

O.load('/tmp/uniax-tension.xml.bz2')

# flattener that project to the xz plane
flattener=post2d.AxisFlatten(useRef=False,axis=1)
# return scalar given a Body instance
extractDmg=lambda b: b.state.normDmg
# will call flattener.planar implicitly
# the same as: extractVelocity=lambda b: flattener.planar(b,b.state.vel)
extractVelocity=lambda b: b.state.vel

# create new figure
pylab.figure()
# plot raw damage
post2d.plot(post2d.data(extractDmg,flattener))

# plot smooth damage into new figure
pylab.figure(); ax,map=post2d.plot(post2d.data(extractDmg,flattener,stDev=2e-3))
# show color scale
pylab.colorbar(map,orientation='horizontal')

# raw velocity (vector field) plot
pylab.figure(); post2d.plot(post2d.data(extractVelocity,flattener))

# smooth velocity plot; data are sampled at regular gridpylab.figure(); ax,
map=post2d.plot(post2d.data(extractVelocity,flattener,stDev=1e-3))
# save last (current) figure to filepylab.gcf().savefig('/tmp/foo.png')


# show the figures
pylab.show()
```

class yade.post2d.AxisFlatten(继承 展平 → 对象)

__init__(使用参考位置=否，轴=*2*)

参数

- **useRef** (*bool*) – 使用参考位置而非实际位置（仅在对刚体进行操作时才有意义）
- **axis** (*{0,1,2}*) – 垂直于平面的法向量；返回值只是删除该分量后的位置。

normal(位置，向量)
给定位置和向量值，返回该向量垂直于平面的长度。

planar(位置，向量)给定位置和向量值，将该向量投影到平面上，并返回其两个平面内分量。

class yade.post2d.CylinderFlatten(继承 *Flatten* →对象)
用于根据从圆出发到平面的投影，将三维点转换为二维点的类。投影中的 y 轴对应旋转轴；x 轴表示到该轴的距离。

__init__(使用参考位置，轴=*2*)

参数

- **useRef**– （布尔值）使用参考位置而非实际位置
- **axis** – 圆柱体的轴， {0,1,2}

normal(*b*，*vec*)
给定位置和向量值，返回垂直于平面的向量的长度。

planar(*b*，*vec*)给定位置和向量值，将向量值投影到平面上，并返回其两个平面内分量。

class yade.post2d.Flatten(继承对象)
用于将三维点转换为二维的抽象类。供 post2d.data2d 使用。

normal(位置，向量)
给定位置和向量值，返回该向量垂直于平面的分量的长度。

planar(位置、向量)给定位置和向量值，将该向量值投影到平面上，并返回其两个平面内分量。

class yade.post2d.HelixFlatten(继承展平 →对象)
基于螺旋体投影将三维点转换为二维的类。投影中的 y 轴对应于旋转轴

__init__(使用参考位置，角度范围，*dH_dTheta*，轴=*2*，螺旋高度=*0*)

参数

- **useRef**(*bool*) – 使用参考位置而非实际位置
- **thetaRange**(*(ϑmin,ϑmax)*) – 超出此范围的刚体将被丢弃
- **dH_dTheta** (*float*) – 螺旋的倾斜度（每弧度）
- **axis** (*{0,1,2}*) – 螺旋的旋转轴
- **periodStart** (*float*) – 零向量角度时螺旋的高度

normal(位置，向量)
给定位置和向量值，返回垂直于平面的向量的长度。

planar(*b*，向量)

给定位置和向量值，将该向量值投影到平面上，并返回其 2 个平面内分量。

yade.post2d.data(提取器，展平器，接触=否，仅动态=真，标准差=无，相对阈值-旧=*3.0*，每单位面积=*0*，*div=(50，50)*，边距=*(0，0)*，半径=*1*)

筛选所有刚体/相互作用，将其投影到二维并提取所需的标量值；根据是否指定标准差值，返回位置和值的离散数组或平滑数据。

该 intr 参数决定我们是对刚体还是相互作用进行操作；提供的提取器应预期接收刚体/相互作用。

参数

- **extractor**（*callable*）– 接收 刚体 (或 相互作用，如果 intr 为 True) 的实例，应返回标量、二元组（向量场）或 None（跳过该刚体/相互作用）
- **flattener**（*callable*）– *post2d.Flatten* 实例，接收刚体/相互作用，返回其二维坐标或 None （跳过该刚体/相互作用）
- **intr** (*bool*) – 对相互作用而非刚体进行操作
- **onlyDynamic** (*bool*) – 跳过所有非动态刚体
- **stDev** (*float/None*) – 用于平均的标准差，启用平滑； None（默认）表示原始模式，返回离散点
- **relThreshold** (*float*) – 高斯权重函数相对于标准差的阈值（仅平滑模式）
- **perArea** (*int*) – 若为 1，则计算加权和/加权面积，而不是加权平均（加权和/权重和）；前者适用于计算平均应力；若为 2，则在细分单元上计算平均值，而不使用权重函数
- **div** (*(int,int)*) – 高斯网格的单元格数量（仅限平滑模式）
- **margin** (*(float,float)*) – 数据包围盒周围的 x、y 边距（仅限平滑模式）
- **radius** (*float/callable*) – 非球形刚体或相互作用的半径回退值（用于原始绘图）；若为可调用对象，则接收刚体/相互作用并返回半径

返回的字典

返回的字典始终包含键'type'（根据 smooth 的值及提取器的返回值，为'rawScalar'、'rawVector'、'smoothScalar'或'smoothVector'之一）、'x'、'y'和'bbox'。

原始数据还包含"半径"。

标量场包含"val"（由提取器返回的值），向量场包含"valX"和"valY"（由提取器返回的两个分量）。

yade.post2d.plot(数据，坐标轴=无，*alpha=0.5*，等高线标签=是，色标=否，纵横比='相等'，***kw*)给定 post2d.data 的输出，将标量绘制为离散图或平滑图。

对于原始离散数据，绘制实心圆，圆的半径对应颗粒半径，并按标量值着色。

对于平滑离散数据，绘制图像，并可选择显示等高线和等高线标签。

For 对于向量数据（原始或平滑），绘制箭矢图（向量场），箭头按其大小着色 e.

参数

- **axes** –将要绘制图形的 matplotlib.axes 实例；如果为 None，则会从头创建。

- **data** –由 *post2d.data* 返回的值
- **clabel** (*bool*) – 显示等高线标签（仅限平滑模式），或在单元格内标注数字（使用 perArea==2）
- **cbar** (*bool*) – 显示颜色条（等同于对返回的可映射对象调用 pylab.colorbar(mappable)）

返回值 (axes,mappable) 元组；可映射对象可用于后续调用 py- lab.colorbar。

2.4.15 yade.qt 模块

yade 的通用初始化核心。

首次从 yade 导入任何内容时，都会执行此文件。它会加载 Yade 插件，并将构造函数注入 c++ class __builtins__ 命名空间（不过将来可能会有所变化），使其在任何地方都可用。

class yade.qt._GLViewer.GLViewer

__init__()引发异常：无法从 Python 实例化此类axes显示坐标轴的箭头。center(*(GLViewer)*辐角 *1*[, *(bool)*中位数=*True*[, *(float)*建议半径=*-1.0*]])→None：居中视图。视图居中后，使全部刚体都能置于其中（中位数 = False），或使 75% 的刚体置于其中（中位数 =True）。如果无法自动确定半径，则使用建议半径。close(*(GLViewer)*辐角 *1*)→None

eyePosition

相机位置。fitAABB(*(GLViewer)*参数*1*,*(*三维向量*)*最小值, *(*三维向量*)*最大值）→ None：调整场景边界，使给定下角点和上角点的轴对齐包围盒最小值, 最大值 适合其中。fitSphere(*(GLViewer)*参数*1*, *(*三维向量*)*中心,*(*浮点数*)*半径）→ None：调整场景边界，使给定的球体中心和半径 适合其中。fps显示每秒帧数指示器。grid在零平面上显示方形网格，以由 3 个布尔值组成的元组指定 yz、xz、xy 平面。loadState(*(GLViewer)*辐角 *1*[, *(str)*状态文件名=*'.qglviewer.xml'*])→ None：从先前保存到文件中的显示参数加载。

lookAt

相机所指向的点。ortho是否使用正交投影；如果为否，则使用透视投影。saveSnapshot(*(GLViewer)*参数*1*，*(str)*文件名）→ None：将当前视图保存为图像文件saveState(*(GLViewer)*辐角 *1*[, *(str)*状态文件名=*'.qglviewer.xml'*])→ None：将显示参数保存到文件中。保存以下两者的状态：*GLViewer* 以及关联的*OpenGLRen-derer*。

scale
视图比例 (?)

sceneRadius
可见场景半径。

screenSize
查看器窗口的尺寸，以屏幕像素为单位

selection

showEntireScene(*(GLViewer)*辐角 *1*) →None

timeDisp
窗口中显示的时间；这是由字符 r、v、i 组成的字符串，分别表示实际时间、虚拟时间和迭代次数。

upVector
将在屏幕上朝上显示的向量。

viewDir
相机方向（以向量表示）。

yade.qt._GLViewer.**Renderer**() → OpenGL 渲染器
返回活动状态的 *OpenGL 渲染器* 对象。

yade.qt._GLViewer.**View**([*(float)timeout=5.0*]) → GLViewer 创建一个新的三维视图。

yade.qt._GLViewer.**center**([*(float)suggestedRadius=-1.0*[, *(三维向量)网格原点=Vector3(0,0,0)*[, *(三维向量)建议中心=Vector3(0, 0, 0)*[,*(整数)网格小数位数=4*]]]]) →无
将全部视图居中。

参数

- **suggestedRadius** – 可选参数；若提供且为正数，则会取代自动半径检测。此参数会影响：(1) 绘制的网格尺寸；(2) OpenGL 中的 Z 轴裁剪距离。这意味着，如果裁剪距离过大，导致场景中的部分内容未被绘制而是被“切掉”或“切片”，则需要增大此参数。
- **gridOrigin** – 可选参数；若提供，则会用作绘制网格的原点。也就是说，三个网格的交点不会位于 0,0,0，而是位于所提供坐标舍入到最接近网格步长后的点。
- **suggestedCenter** – 可选参数；若提供的值不是 (0,0,0)，则会取代使用包围盒自动计算场景中心。此参数会影响绘制的旋转中心。如果尝试旋转视图时发现旋转是围绕某个奇怪的点进行的，则需要更改此参数。
- **gridDecimalPlaces** – 默认值=4，用于确定网格标签显示的小数位数，数值通过字符串流处理（不会显示多余的零）。

注意： 调用以下命令即可获取这四个参数的当前值：*qt.centerValues()*

---

yade.qt._GLViewer.**centerValues**() → 字典

返回 一个包含 yade.qt.center(…) 当前使用的全部参数的字典，参见*qt.center* 或查看类型 yade.qt.center? 了解详情。视图关闭时返回零向量。

yade.qt._GLViewer.**views**() → 列表

返回所有打开的*qt.GLViewer* 对象列表 ts

如果需要在两个不同的 Yade 会话之间精确复制相机位置和设置，可以使用以下命令：

```
v=yade.qt.views()[0]                                    ## to obtain a handle of currently opened␣
→view.v.lookAt,  v.viewDir,  v.eyePosition,
 v.upVector ## to print the current camera parameters␣→of the view.
## Then copy the output of this command into the second yade session to reposition the␣
→camera.v.lookAt,  v.viewDir,  v.eyePosition,
 v.upVector = (Vector3(-0.5,1.6,0.47),Vector3(-0.5,0.6,→0.4), Vector3(0.015,0.98,-0.012),
Vector3(0.84,0.46,0.27))
## Since these parameters depend on each other it might be necessary to execute this␣
→command twice.
```

此外，还可以调用 *qt.centerValues()* 获取轴和场景半径的当前设置（如果未使用默认值），并在第二个 Yade 会话中调用 *qt.center* 应用这些设置。

上述繁琐的方法将来或许可以改进。

2.4.16 yade.timing 模块

用于访问存储在引擎和函子中的计时信息的函数。

请参阅程序员手册中的 计时 章节，以及 Wiki 页面，其中有一些示例。

yade.timing.**reset**()将所有计时数据归零。

yade.timing.**runtime**()
返回总运行时间（与 stats() 输出的最后一行相同），单位为纳秒

yade.timing.**stats**()打印引擎和函子的计时信息汇总表。其中包含绝对时间和百分比。输出示例：

```
Name                                                    Count                   Time␣
 →    Rel. time
-------------------------------------------------------------------------------------------
 →-------------
ForceResetter                                             102                 2150us␣
 →   0.02%
"collider"                                                  5                64200us␣
 →   0.60%
InteractionLoop                                           102              10571887us␣
 →  98.49%
"combEngine"                                              102                 8362us␣
 →   0.08%
"newton"                                                  102                73166us␣
 →   0.68%
"cpmStateUpdater"                                           1                 9605us␣
 →   0.09%
PyRunner                                                    1                  136us␣
 →   0.00%
"plotDataCollector"                                         1                  291us␣
 →   0.00%
TOTAL                                                                     10733564us␣
 →100.00%
```

示例输出（使用 -DENABLE_PROFILING=1 选项编译）：

```
Name                                                  Count                  Time
    Rel. time
-------------------------------------------------------------------------------------------
-------------
ForceResetter                                           102                2150us
  0.02%
"collider"                                                5                64200us
  0.60%
InteractionLoop                                         102            10571887us
 98.49%
  Ig2_Sphere_Sphere_ScGeom                             1222186            1723168us
    16.30%
    Ig2_Sphere_Sphere_ScGeom                           1222186            1723168us
     100.00%
  Ig2_Facet_Sphere_ScGeom                                753                1157us
     0.01%
    Ig2_Facet_Sphere_ScGeom                              753                1157us
     100.00%
  Ip2_CpmMat_CpmMat_CpmPhys                            11712               26015us
     0.25%
    end of Ip2_CpmPhys                                 11712               26015us
     100.00%
  Ip2_FrictMat_CpmMat_FrictPhys                            0                   0us
     0.00%
  Law2_ScGeom_CpmPhys_Cpm                              3583872            4819289us
    45.59%
    GO A                                               1194624            1423738us
      29.54%
    GO B                                               1194624            1801250us
      37.38%
    rest                                               1194624            1594300us
      33.08%
    TOTAL                                              3583872            4819289us
     100.00%
  Law2_ScGeom_FrictPhys_CundallStrack                      0                   0us
     0.00%
"combEngine"                                            102                8362us
  0.08%
"newton"                                                102               73166us
  0.68%
"cpmStateUpdater"                                         1                9605us
  0.09%
PyRunner                                                  1                 136us
  0.00%
"plotDataCollector"                                       1                 291us
  0.00%
TOTAL                                                                  10733564us
100.00%
```

2.4.17 yade.utils 模块

暂时还无法归入其他类别的函数合集。

开发者：请勿在此处添加更多函数，这里已经太拥挤了！

yade.utils.NormalRestitution2DampingRate(*en*)
计算法向阻尼率与法向恢复系数 e_n 的函数关系。对于$e_n \in \langle 0,\ 1\rangle$ 阻尼率等于

$$-\frac{\log e_n}{\sqrt{e_n^2+\pi^2}}$$

yade.utils.**SpherePWaveTimeStep**(半径、密度、杨氏模量)

使用 P 波传播速度公式 $\Delta t_c = \sqrt[r]{E/\rho}$，计算单个（假定具有代表性）球体的 P 波临界时间步长。若要计算模拟中所有球体的最小临界时间步长，请改用 *utils.PWaveTimeStep* 。

```
>>> SpherePWaveTimeStep(1e-3,2400,30e9)
2.8284271247461903e-07
```

class yade.utils.**TableParamReader**(继承自对象)

用于从文本文件读取仿真参数的类。

Each 每个参数由一列表示，每个参数集由一行表示。各列之间以 d[bl 分隔]anks（不加引号）。

第一个非空行包含列标题（不加引号）。可以使用名为“description”的特殊列来描述此参数集；如果没有此列，则会通过拼接列名及其对应的值（param1=34、param2=12.22、param4=foo）生成描述

- 来自以 ! 结尾的列（ ! 不包含在列名中）
- 来自所有列（如果没有列以 !结尾）。

文件中的空行会被忽略（但会计数）； # 会将该行剩余部分标记为注释。所有非空行中的以空格分隔的列数量必须相同。

可以使用特殊值 = 代替参数值；此时将使用前一个非空行中的值（可递归使用）。

此类由 *utils.readParamsFromTable* 使用。

__init__(文件)设置读取器类，将数据读入内存。

paramDict()

返回一个字典，其中包含传递给构造函数的文件中的数据。键为行号（可能不连续，且对应于文本编辑器中可见的实际行号），值为字典，用于将参数名称映射到文件中给出的参数值。特殊值‘=’已被解释， ！（感叹号，如有）已从列标题中移除，description 列已添加（如果原本不存在）。

yade.utils.**aabbDim**(截断值=*0.0*, 中心=否)

返回轴对齐包围盒的尺寸，可选择切除相对部分截断值 。

yade.utils.**aabbExtrema2d**(点)返回由二元组序列构成的二维包围盒。

yade.utils.**aabbWalls**(极值点=无,厚度=*0*, 过大系数=*1.5*, ***kw*)返回 6 个盒体，从各个方向将现有堆积体包裹为墙。

参数

- **extrema** – 堆积体 Aabb 的极值点，作为由两个三维向量组成的列表，或任何等效类型（如果未指定，则会计算）
- **thickness** (*float*) – 壁厚（如果未指定，则为 X 维度的 1/10）
- **oversizeFactor** (*float*) – 用于扩大墙体平面尺寸的因子。

返回按 minX、maxX、minY、maxY、minZ、maxZ 顺序排列的 6 个包围堆积体的壁体列表。

yade.utils.**avgNumInteractions**(截断值=*0.0*, 跳过自由颗粒=否, *considerClumps*=否)返回每个颗粒的平均相互作用数量，也称为配位数 Z。其定义如下：

$$Z = 2C/N$$

其中， C 是接触数量， N 是颗粒数量。存在团簇时，颗粒数量等于独立球体数量与团簇数量之和。如果 cutoff != 0 或 skipFree = True，则计算中会考虑团簇。如果 cutoff=0 （默认值）且 skipFree=False（默认值），则需要将 considerClumps=设为 True，才能在计算中考虑团簇。

使用 跳过自由颗粒时，将跳过对堆积体稳定状态没有贡献的颗粒，依据 [Thornton2000]给出的方程（8）：

$$Z_m = \frac{2C - N_1}{N - N_0 - N_1}$$

参数

- **cutoff** – 从试样的包围盒中裁去一定比例的部分。
- **skipFree** – 参见上文。
- **considerClumps** – 如果 cutoff=0 且 skipFree=为 False，也考虑团簇；进一步说明请参见上文。

yade.utils.box(中心，半尺寸，方向=四元数*((1，0，0)，0)*，动态=无，固定=否，线框=否，颜色=无，高亮=否，材料=*-1*，掩码=*1*)使用给定参数创建盒子（长方体）。

参数

- **extents** (Vector3) – 沿 x、y、z 轴的半尺寸。若盒子相关坐标轴与仿真坐标轴不一致，可使用方向参数
- **orientation** (Quaternion) – 赋予刚体的方向，对应于旋转半尺寸的坐标轴

请参阅*utils.sphere*的文档，了解其他参数的含义。

class yade.utils.clumpTemplate(继承对象)通过相对半径列表和相对位置列表创建团簇模板。两个列表的长度必须相同。

参数

- **relRadii** (*[float,float,..]*) – 相对半径列表（最小长度为 = 2）
- **relPositions** (*[*Vector3,Vector3,..*]*) – 相对位置列表（最小长度为 = 2）

yade.utils.defaultMaterial()

创建刚体时返回默认材料，若使用 *utils.sphere* 等函数且未指定材料，也尚未定义共享材料。默认情况下，此函数返回

```
FrictMat(density=1e3,young=1e7,poisson=.3,frictionAngle=.5,label='defaultMat')
```

yade.utils.facet(顶点，动态=无，固定=是，线框=是，颜色=无，高亮=否，无*Bound*=否，材料=*-1*，掩码=*1*，链=*-1*)使用给定参数创建面片。

参数

- **vertices** (*[*Vector3,Vector3,Vector3*]*) – 全局坐标系中顶点的坐标。
- **wire** (*bool*) – 若 True，面片将以线框形式显示；否则面片将被填充
- **noBound** (*bool*) – 设置 *Body.bounded*
- **color** (*Vector3-or-None*) – 面片的颜色；若None，则将分配随机颜色。

请参阅*utils.sphere*的文档，了解其他参数的含义。

yade.utils.fractionalBox(分数=*1.0*，最小最大值=*N one*)
返回将原始 minMaxbox（如果未指定，则为整个模拟的 aabb）以其中心为基准按比例因子线性缩放后的 (min,max)

yade.utils.levelSetBody(形状=*”*，中心=三维向量*(0，0，0)*，半径=*0*，半尺寸=三维向量*(0，0，0)*，机器精度=二维向量*(0，0)*，团簇=*None*，间距=*0.1*，网格=*None*，距离场=*[]*，表面节点数量=*27*，节点路径=*2*，节点容差=*50*，方向=四元数*((1，0，0)，0)*，动态=*True*，材料=*-1*)

通过多种工作流程创建水平集形状的刚体：可以选择预定义形状（通过 shape 及相关属性），也可以模拟团簇实例（团簇属性，用于比较），或直接在某个网格上赋予离散距离场（distField 和 grid 属性）。: param 字符串 shape: 使用此参数可选用预定义形状，包括‘sphere’（球体）、‘box’（长方体）、‘disk’（二维分析中 (x、y) 平面上的圆盘）或‘superellipsoid’（超椭球）；需同时使用半尺寸或半径属性。超椭球曲面在局部坐标轴（惯性坐标系）中由以下方程定义：
$f(x,y,z) = (|x/r_x|^{2/\varepsilon_e} + |y/r_y|^{2/\varepsilon_e})^{\varepsilon_e/\varepsilon_n} + |z/r_z|^{2/\varepsilon_n} = 1$其距离场通过快速行进法求得。:param 三维向量 center: 刚体的（初始）位置 :param 团簇 clump: 如需模拟团簇，可在此传入一个多球实例 : param 实数 radius: 当形状为 = ‘sphere’或‘disk’时指定的半径 :param 三维向量 extents: 当形状为 = ‘box’或‘superellipsoid’时沿局部坐标轴的半尺寸（对于后者为r_x、r_y、r_z）:param 二维向量 epsilons: 形状为 = ‘superellipsoid’时的（ε_e、ε_n）指数 :param 实数 spacing: 选择预定义形状或团簇时，水平集网格的空间增量 :param 列表 distField: 网格上的离散距离场（如已给定），以列表（由列表嵌套而成的列表）形式表示；如果最初使用三维 NumPy 数组，请使用 .tolist()。其中 distField[i][j][k] 表示网格点 grid.gridPoint(i,j,k) 处的距离值 :param RegularGrid grid: 承载距离场的网格；当距离场通过 distField 直接赋值时使用 :param int nSurfNodes: 边界节点的数量，传递给 LevelSet.nSurfNodes :param int nodesPath: 边界节点的路径，传递给 LevelSet.nodesPath : param 实数 nodesTol: 射线追踪边界节点时的容差，传递给 LevelSet.nodesTol :param 四元数 orientation: 刚体的初始方向 :param bool dynamic: 传递给 Body.dynamic :param 材料 material: 传递给 Body.material :return: 相应的刚体实例

yade.utils.loadVars(标记=无)
从 *utils.saveVars* 加载保存在模拟中的变量。如果 `mark==None`，则加载全部保存的变量。否则仅加载带有传入标记的变量。

yade.utils.makeVideo(帧规格，输出，不覆盖重命名=是，帧率=*24*，千比特每秒=*6000*，比特每秒=无)

使用 mencoder 从外部图像文件创建视频。使用默认的 mencoder 编解码器（MPEG-4）执行两遍编码，并以多线程方式运行，线程数等于为 Yade 分配的 OpenMP 线程数。

参数

- **frameSpec** – 通配符 | 文件名序列。如果是列表或元组，则按给定顺序对文件名进行编码；否则使用 mencoder 的 mf:// URI 选项所理解的通配符（shell 通配符，例如 `/tmp/snap-*.png` 或 printf 风格模式，例如`/tmp/snap-%05d.png`）
- **out** (*str*) – 要保存视频的文件
- **renameNotOverwrite** (*bool*)– 如果为 True，已存在的同名视频文件将附加 -数量；否则将被覆盖。
- **fps** (*int*) – 每秒帧数（`-mf fps=…`）
- **kbps** (*int*) – 比特率（`-lavcopts vbitrate=…`），单位为 kb/s

yade.utils.perpendicularArea(轴)
返回由刚体生成的、垂直于给定轴的面积（0=x，1=y，2=z），这些刚体满足函数 consider 返回 True（默认始终返回 True），且类型为 球体。

yade.utils.phiIniPy(*ioPyFn*，网格)
返回一个适合作为 *FastMarchingMethod.phiIni* 的三维离散场（需要 LS_DEM 特性），并应用用户自定义的 Python 函数 *ioPyFn*

参数

- **ioPyFn** – 一个现有的内外部 Python 函数，它接受三个数字（笛卡尔坐标）作为参数
- **grid** (*RegularGrid*) – 要操作的*RegularGrid* 实例

返回列表 一个合适的三维离散场，以传入*FastMarchingMethod.phiIni*

yade.utils.**plotDirections**(*aabb=()*, 掩码=*0*, 分箱=*20*, *numHist=True*, *noShow=False*, *sph- Sph= False*)

绘制 3 个直方图，显示相互作用方向在 yz、xz 和 xy 平面上的分布，并可选择绘制每个刚体的相互作用数量直方图（默认启用）。如果 sphSph 为真，则 3 个方向直方图仅考虑球-球相互作用。

返回值 如果 *noShow* 为 `False`，则显示图形且不返回任何内容。如果 *noShow*，则返回图形对象但不显示（行为与 *plot.plot*相同）。

yade.utils.**plotNumInteractionsHistogram**(截断值=*0.0*)

绘制每个刚体的相互作用数量直方图，可选择从试样边缘切去一个与轴对齐的盒子。

yade.utils.**polyhedron**(顶点，固定=*False*，线框=*True*，颜色=*None*，高亮=*False*，无边界=*False*，材料=*-1*，掩码=*1*，链=*-1*)使用给定参数创建多面体。

参数 **vertices** (*[*Vector3*]*) – 全局坐标系中顶点的坐标。

参见 *utils.spher* *e*的文档，用于了解其他参数的含义 。

yade.utils.**psd**(分箱=*5*，质量=真，掩码=*-1*)计算颗粒粒径分布。

参数

- **bins** (*int*) – 分箱数量
- **mass**(*bool*) – 如果为真，则会计算质量粒径分布
- **mask** （*int*） – *Body.mask* 刚体的

返回值

- binsSizes：分箱尺寸列表
- binsProc：分箱中所含材料的百分比（累计）
- binsSumCum：分箱中所含材料的单位向量总量（累计）

binsSizes、binsProc、binsSumCum

yade.utils.**randomColor**(随机种子=*None*)

返回一个随机三维向量，各分量位于 0...1 区间内（均匀分布）

yade.utils.**randomOrientation**()

返回（均匀分布的）随机方向。取自Eigen::Quaternion::UnitRandom() 源代码。使用标准 Python random.random() 函数，你可以 `random.seed()` 它

yade.utils.**randomizeColors**(仅动力体=*False*)

为*Shape::color*分配随机颜色。

如果仅动力体为 true，则只有动力体的颜色会改变。

yade.utils.**readParamsFromTable**(表格文件行=无，允许无表格=真，允许未知项=假，***kw*)

从文件中读取参数，并将其赋值给 ___内置___ 变量。

文件的格式如下（允许使用以 # 开头的注释和空行）：

```
# commented lines allowed anywhere
name1 name2 … # first non-blank line are column headings
# empty line is OK, with or without comment
val1    val2    … # 1st parameter set_val2    val2    … # 2nd…
```

A已分配的标签（如果缺少，则会合成 `description` 列，参见 *utils.TableParamReader*）；O.tags[‘描述’]=... # 设置描述列；可能会合成 O.tags[‘参数’]=”名称1=值1，名称2=值2，...” # 所有显式指定的参数 O.tags[‘ 默认参数 ’]=” 未赋值的名称 1= 默认值 1，...” # 保持默认值的参数 O.tags[‘d.id’]=O.tags[‘标识符’]+’.’+O.tags[‘描述’]O.tags[‘id.d’]=O.tags[‘描述’]+’.’+O.tags[‘标识符’]

所有参数（默认参数和可设置参数）均使用 *utils.saveVars*(`'table'`)保存。

参数

- **tableFileLine** – 字符串属性，用于指定从文本文件（每个值位于以空格分隔的列中）中读取值的行号（如文本编辑器中所示）。两条信息之间应有一个“:”，例如“file.table:4”表示从 file.table 文件中读取第 4 行
- **noTableOk** (*bool*) – 如果为 False，则在无法打开文件时引发异常；否则使用默认值
- **unknownOk** (*bool*)– 如果在文件中找到未知列名，则不引发异常，并将其一并赋值

返回 已赋值参数的数量

yade.utils.**replaceCollider**(碰撞引擎)
Repl用提供的引擎访问碰撞器（Collider）引擎。如果引擎中没有碰撞器，则引发错误 s.

yade.utils.**runningInBatch**()判断我们是在批处理中运行还是单独运行。

yade.utils.**saveVars**(标记=*''*, *loadNow=True*, ***kw*)将传递的变量保存到模拟中，以便再次加载模拟时能够恢复。

例如，定义了变量a、b 和c 。要保存它们，请使用：

```
>>> saveVars('something',a=1,b=2,c=3)
>>> from yade.params.something import *
>>> a,b,c(1, 2, 3)
```

当保存模拟本身时，这些变量将保存到 .xml 文件中。要在再次加载 .xml 后恢复这些变量，请使用 `loadVars('something')`，它们便会在 yade.params 中定义。标记 模块。 *loadNow* 参数会在保存后自动调用 *utils.loadVars* 。如果“something”已存在，则会插入给定变量。

yade.utils.**sphere**(中心，半径，动态=*None*，固定=*False*，线框=*False*，颜色=*None*，高光=*False*，材料=*-1*，掩码=*1*)使用给定参数创建球体；质量和惯性将自动计算。

默认使用最后指定的材料（材料 = -1）；如果完全没有定义材料，则使用 utils.defaultMaterial()。

参数

- **center** (Vector3) – 中心
- **radius** (*float*) – 半径

- **dynamic** (*float*) – 已弃用，参见“固定”
- **fixed** (*float*) – 生成所有自由度均被锁定的刚体?
- **material**–**指定 *Body.material*；接受不同类型：** – int：O.materials[材料] 将被使用；作为一种特殊情况，如果材料==- 1 且未定义共享材料，则会将 utils.defaultMaterial() 分配给 O.materials[0]– 字符串：将使用现有材料的标签– 材料实例：将使用此实例 – 可调用对象：将不带参数调用；返回的 Material 值将被使用（也可以说是 Material 工厂对象）

- **mask** (*int*)– 刚体掩码，用于该刚体
- **wire** – 显示为线框球体?
- **highlight** – 在查看器中突出显示此刚体?
- **Vector3-or-None** – 刚体的颜色，以归一化 RGB 表示；如果满足 None，则会分配随机颜色。

Re转化为 具有所需特征的 Body 实例 属性。

如果不存在共享材料且未提供材料，则创建默认共享材料：

```
>>> O.reset()
>>> from yade import utils
>>> len(O.materials)0
>>> s0=utils.sphere([2,0,0],1)
>>> len(O.materials)1
```

可提供材料实例：

```
>>> s1=utils.sphere([0,0,0],1,wire=False,color=(0,1,0),material=ElastMat(young=30e9,
→density=2e3))>>> s1.shape.wireFalse>>> s1.shape.colorVector3(0,1,0)
>>> s1.mat.density2000.0
```

材料可以通过标签指定：

```
>>> O.materials.append(FrictMat(young=10e9,poisson=.11,label='myMaterial'))
1>>> s2=utils.sphere([0,0,2],1,material='myMaterial')>>> s2.mat.label
'myMaterial'>>> s2.mat.poisson0.11
```

最后，材料也可以是一个不接受任何参数、并返回 Material 实例的可调用对象。如果你不直接调用此函数（例如通过 yade.pack.randomDensePack 调用），而只传入 1 个材料 参数，但又不希望材料被共享，请使用这种方式。

例如，可以像这样创建随机材料属性：

```
>>> import random
>>> def matFactory(): return ElastMat(young=1e10*random.random(),density=1e3+1e3*random.
→random())...>>> s3=utils.sphere([0,2,0],1,material=matFactory)
>>> s4=utils.sphere([1,2,0],1,material=matFactory)
```

yade.utils.**tetra**(顶点，严格检查=是，固定=否，线框=是，颜色=无，高光=否，无边界=否，材料=*-1*，掩码=*1*，链=*-1*)使用给定参数创建四面体。

参数

- **vertices** (*[*Vector3,Vector3,Vector3,Vector3*]*) – 全局坐标系中顶点的坐标。
- **strictCheck** (*bool*) – 检查顶点顺序，若体积为负则引发 RuntimeError

请参阅 *utils.sphere*的文档，了解其他参数的含义。

yade.utils.**tetraPoly**(顶点，固定=否，线框=是，颜色=无，高亮=否，无*Bound*=否，材料=*-1*，掩码=*1*，链=*-1*)使用给定参数创建四面体（实际上是简单多面体）。

参数 **vertices** (*[*Vector3,Vector3,Vector3,Vector3*]*) – 全局坐标系中顶点的坐标。

请参阅 *utils.sphere*的文档，以了解其他参数的含义。

yade.utils.**trackPerfomance**(*updateTime=5*)跟踪模拟的性能。（实验性功能）将创建新线程以生成一些图形。对于跟踪长时间运行的模拟（例如在批处理模式下）很有用。

yade.utils.**typedEngine**(名称)
返回当前 O.engines 中按类型（字符串形式）标识的第一个引擎。例如 :

```
>>> from yade import utils
>>> O.engines=[InsertionSortCollider(),NewtonIntegrator(),GravityEngine()]
>>> utils.typedEngine("NewtonIntegrator") == O.engines[1]
True
```

yade.utils.**uniaxialTestFeatures**(文件名=*None*，面积截面=*10*，轴=*-1*，距离因子=*2.2*，***kw*)

获取当前堆积体中有助于进行单轴试验的数据：
1. 找出最长的尺寸（单轴加载轴）。2. 检查垂直于该轴的若干个截面（areaSections），计算每个截面的凸包面积，以此找出试样的最小横截面积。此方法也适用于非棱柱形试样。3. 找出位于负向/正向边界上的刚体，并对其施加应变条件。

参数

- **filename** –如果指定，则从此文件（ASCII 格式）加载球体；否则使用当前模拟。
- **areaSection** (*float*) – 用于估算截面的截面数量
- **axis** (*{0,1,2}*) – 若给定，则强制使用应变轴，而不是根据主要长度计算

返回一个字典，其中包含键 negIds、 posIds、 axis、 area。

警告： 函数 *utils.approxSectionArea* 使用凸包算法来查找面积，但据报告，该实现存在缺陷（bot 在某些算例中可以正常工作）。请务必检查此数值，或修复凸包算法（源代码中有相关说明，参见 py/_utils.cpp）。

yade.utils.**vmData**()
从 Linux 的 /proc/[pid]/status 中的 VmData 行返回内存使用数据。

yade.utils.**voxelPorosityTriaxial**(三轴应力应变控制器，分辨率=*200*，偏移量=*0*)给定三轴压缩引擎，计算试样的孔隙率。

调用函数 *utils.voxelPorosity*，并使用由三轴压缩引擎壁面围成的盒子体积。附加参数偏移量用于在盒子内采用较小的体积，其中体积的每一侧与壁面相距指定的偏移距离。这样可以更精确地求出试样的孔隙率，因为壁面接触处的孔隙率通常较低。

建议将偏移量设为不小于内部球体平均半径的值。

分辨率取决于所用球体的尺寸。可以通过调用带有偏移量=0 的 voxel-PorosityTriaxial，并将结果与 TriaxialCompressionEngine.porosity 进行比较来校准。校准后，可将偏移量设为半径或更大的值，以获得结果。

参数

- **triax** – 三轴压缩引擎句柄
- **resolution** – 体素网格分辨率
- **offset** – 偏移距离

返回给定体积内试样的孔隙率

调用示例：

```
from yade import utilsrAvg=0.03
TriaxialTest(numberOfGrains=200,radiusMean=rAvg).load()
O.dt=-1O.run(1000)O.engines[4].porosity
0.44007807740143889
utils.voxelPorosityTriaxial(O.engines[4],200,0)
0.44055412500000002
utils.voxelPorosityTriaxial(O.engines[4],200,rAvg)
0.36798199999999998
```

yade.utils.**waitIfBatch**()
Bloc在批处理模式下运行时停止模拟。通常用于脚本末尾，以便 es not 在批处理模式下提前结束（这种情况下，执行将会终止）。

yade.utils.**wall**(位置，轴，方向=*0*，颜色=无，材料=*-1*，掩码=*1*)返回预先构建的墙体。

参数

- **position**（*float-or-Vector3*）– 墙的中心。如果是浮点数，则表示沿给定轴的位置，其余两个分量为零
- **axis**（*{0,1,2}*）– 墙面法线的方向（0、1、2）分别对应 x、y、z（即 yz、xz、xy 平面）
- **sense**（*{-1,0,1}*）– 交互方向（0：两者，-1：负方向，+1：正方向；参见 *Wall*）

请参阅 *utils.sphere*的文档，了解其他参数的含义。

yade.utils.xMirror(半)
将一组二维点关于 x 轴镜像（改变 y 坐标的符号）。该序列应从上方开始，然后向下 wrap（反之亦然）。如果最后一个点的 x 坐标为零，则不会重复该点。

yade._utils.PWaveTimeStep() → 浮点数
根据 P 波传播速度获取时间步长；该值根据球体半径、刚度和质量计算。

yade._utils.RayleighWaveTimeStep() →浮点数
根据力传播的瑞利波速度确定时间步长。

yade._utils.TetrahedronCentralInertiaTensor(*(对象)*辐角*1*) →Matrix3 待办事项

yade._utils.TetrahedronInertiaTensor(*(对象)*辐角*1*) →Matrix3 待办事项

yade._utils.TetrahedronSignedVolume(*(对象)*辐角*1*) → 浮点数 待办事项

yade._utils.TetrahedronVolume(*(对象)*辐角*1*) →浮点数 待办事项

yade._utils.TetrahedronWithLocalAxesPrincipal(*(刚体)*辐角*1*) →四元数 待办事项

yade._utils.aabbExtrema(*(float)cutoff=0.0*[,*(布尔值)*中心=*False*]) → 元组 返回包围全部球形物体的盒子的坐标

参数

- **centers** (*bool*) – 不考虑球体半径，只考虑其质心
- **cutoff** (*float 〈0…1〉*) – 盒子边界裁去的相对尺寸。

返回 [下角点，上角点] ，类型为 [三维向量,向量3]

yade._utils.angularMomentum([*(Vector3)origin=Vector3(0, 0, 0)*]) → 三维向量 待办

yade._utils.approxSectionArea(*(浮点数)*参数*1*,*(int)*辐角*2*) → 浮点数
计算扫掠球体在坐标为 coord 的平面上穿过时凸包的面积，该平面垂直于 axis。

yade._utils.bodyNumInteractionsHistogram(*(元组)aabb*) → 元组

yade._utils.bodyStressTensors() →列表
计算并返回一个包含单颗粒应力张量的表格。每个张量表示一个颗粒中的平均应力，其通过对施加载荷进行轮廓积分而得，具体如下文所述。此定义将每个球体视为连续体。对于处于静态平衡、在接触点相互作用且固相体积变化可忽略的球体而言，该定义可视为精确（最后这一假设并不限制堆积尺度上可能发生的变形和体积变化）。

证明：
首先，我们指出单位矩阵：$\sigma_{ij} = \delta_{ik}\sigma_{kj} = x_{i,\ k}\sigma_{kj} = (x_i\sigma_{kj}),\ _k - x_i\sigma_{kj,\ k}$。
在平衡状态下，应力的散度为 NULL：$\sigma_{kj,k} = 0$。因此，根据散度定理：
$\frac{1}{V}\int_V \sigma_{ij}dV = \frac{1}{V}\int_V (x_i\sigma_{kj}),_k dV = \frac{1}{V}\int_{\partial V} x_i\sigma_{kj}n_k dS = \frac{1}{V}\sum_b x_i^b f_j^b$。
最后一个等式隐含地基于将外载荷表示为狄拉克分布，其零点即所谓的接触点：尺寸为零的曲面，接触力 作用于这些曲面上，位置为变形构形中的 x_i 。

对单体应力进行加权平均，可得固相内部的平均应力。在不存在流体压力时，固相内部的应力与等效连续体中的应力之间存在简单关系。对于孔隙率 n，该关系为： $\sigma_{ij}^{equ.} = (1-n)\sigma_{ij}^{solid}$。

如果孔隙率并非均匀分布，这个最后的关系可能不太有用。在这种情况下，可以为每个颗粒分配一个体积，从而在颗粒尺度上定义等效体应力。该体积可通过 *TesselationWrapper* （例如，参见 [Catalano2014a]）

yade._utils.calm([*(int)mask=-1*]) → 无 将物体的平移速度和转动速度设为零。默认情况下应用于全部物体。若只需使部分物体平静下来，请使用掩码参数；它只会使组掩码与给定值兼容的物体平静下来

yade._utils.coordsAndDisplacements（（整数）轴[，（元组）*Aabb=()*]）→ 元组 返回由两个等长列表组成的元组，分别表示给定轴（第一个辐角）上的坐标和位移（坐标减去参考坐标）；如果给定 Aabb=((x_min,y_min,z_min),(x_max,y_max,z_- max)) 盒子，则只考虑该盒子内的物体。

yade._utils.createInteraction(*(int)*刚体标识符*1*，*(int)*标识符*2*[，*(bool)*虚拟相互作用=否]）→ Interaction：手动在给定刚体之间创建相互作用。

如果虚拟相互作用=否，则会在当前引擎中搜索*IGeomDispatcher* 和*IPhysDispatcher* （两者可能都隐藏在*InteractionLoop*中）。几何体由 `force` 参数通过几何分派器创建，因此即使刚体在空间上没有重叠，或者函子在正常情况下会返回 `false` ，相互作用仍然存在。

如果虚拟相互作用=为真，则相互作用将保持在虚拟状态。

警告：此函数在周期性模拟中很可能无法正常工作（不过要扩展它以处理这种情况相当容易）。

yade._utils.fabricTensor([*(float)cutoff=0.0*[，*(bool)*分解张量=否[，*(float)*力阈值=非数]]]）→ 元组：计算全部相互作用的 $F_{ij} = \frac{1}{n_c}\sum_c n_i n_j$ [Satake1982]，织构张量 c。

参数

- **cutoff** (*Real*) – 用于忽略边界效应：在 [0 中定义；1]聚焦于球形堆积 V 中居中的内侧 (1-cutoff)^3*V 部分内的相互作用。
- **splitTensor** (*bool*) – 将织构张量分为两部分，分别对应强接触力（最大压缩法向力）和弱接触力。
- **thresholdForce** (*Real*) – 如果将织构张量分为两部分，可以指定一个阈值，否则默认采用平均接触力。压缩状态使用带负号的值。请注意，如果要区分压缩力和拉力，可以将此值设为零。

yade._utils.flipCell([*(Matrix3)flip=Matrix3(0, 0, 0, 0, 0, 0, 0, 0, 0*<style 标识符='2'>)]）→ Matrix3

翻转周期单元，使 R^3 坐标轴与变换后的坐标轴之间的角度尽可能小，依据以下两条事实：1. 在 R^3 空间中重复周期单元的角点会形成规则网格；2. 通过此过程得到同一网格的两个单元彼此等价，可以相互翻转。翻转需要调整 *Interaction.cellDist* ，以处理跨越边界但之前未跨越（或反之）的相互作用，并重新初始化碰撞器。可使用 *flip* 参数指定所需的翻转：输入整数矩阵，每列对应一个轴；如果矩阵为零矩阵，则自动计算最佳拟合结果（使角度最小）。

在 C++ 中，此函数可通过 `Shop::flipCell` 访问。

yade._utils.forcesOnCoordPlane((浮点数)参数*1*，*(int)*辐角*2*）→ 三维向量

yade._utils.forcesOnPlane(*(三维向量)planePt*，*(三维向量)法向量*）→ 三维向量 查找所有派生自 *NormShearPhys* 且穿过指定平面的相互作用，并对它们的力（法向和剪切力）求和。

参数

- **planePt** (Vector3) – 平面上的一个点
- **normal** (Vector3) – 平面法向量（将进行归一化）。

yade._utils.getBodyIdsContacts(*(int)bodyID=0*) → 列表 G

et a list o f 与给定刚体接触的刚体 ID。

yade._utils.getCapillaryStress(*(float)volume=0*, *(布尔值)Mindlin 毛细=False*) → Matrix3 计算并返回 Love-Weber 毛细应力张量：

$\sigma_{ij}^{cap} = \frac{1}{V}\sum_b l_i^b f_j^{cap\,b}$，其中总和遍历全部相互作用，$l$ 为分支向量（连接刚体中心），f^{cap} 为毛细力。 V 可传递给该函数。如果未传递，在非周期情形下其值为 1，在周期情形下则等于单元体积。目前仅支持 CapillaryPhys 相互作用类型。将此函数用于 Mindlin 毛细物理时，需要将 True 作为第二个参数传递。

yade._utils.getDepthProfiles(*(浮点数)体积*，*(整数)nCell*，*(浮点数)dz*，*(浮点数)参考位置*，*(布尔值)activateCond*，*(浮点数)radiusPy*，*(整数)方向*） →元组

计算并返回沿指定方向（默认为 z； 0=>x、1=>y、2=>z）的颗粒速度和固相体积分数（孔隙率）深度分布。对于每个定义的单元 z，平均颗粒速度的 k 分量为：

$$< v_k >^z = \sum_p V^p v_k^p / \sum_p V^p,$$

其中，总和是对单元内包含的颗粒求得的； v_k^p 是与颗粒 p 相关的速度的 k 分量， V^p 是颗粒 p 位于单元内的体积部分。此定义能够使平均过程平滑；当每个单元中有大量颗粒时，它等效于仅考虑颗粒中心。固相体积分数也以相同方式计算：对于每个定义的单元 z，其表达式为：

$$< \varphi >^z = \frac{1}{V_{cell}} \sum_p V^p,$$ **其中 V_{cell} 是所考虑单元的体积，V^p 是颗粒**

此函数给出平均速度和固相体积分数的深度分布，返回从高度为 zRef 的参考水平面（输入参数）到高度为 zRef+nCell*dz 的平面（输入参数）之间、每个高度为 dz 的单元中的平均物理量。如果参数 activateCond 设为 true，则仅对半径等于 radiusPy（输入参数）的颗粒求平均。

yade._utils.getDepthProfiles_center(*(浮点数)体积*，*(整数)单元数*，*(浮点数)dz*，*(浮点数)参考位置*，*(布尔值)activateCond*，*(浮点数)radiusPy*) → 元组 S

与 get 相同DepthProfiles b纳入考虑k将颗粒 l作为点 l分配 d at th粒子 l中心。

yade._utils.getDynamicStress() → 列表

计算每个刚体的动态应力张量： $\sigma_D^p = -\frac{1}{V^p} m^p u'^p \otimes u'^p$ —

yade._utils.getSpheresMass(*(int)mask=-1*) → 浮点数

计算模拟中球体的总质量，并考虑掩码参数

yade._utils.getSpheresVolume(*(int)mask=-1*) → 浮点数

计算模拟中球体的总体积，并考虑掩码参数

yade._utils.getSpheresVolume2D(*(int)mask=-1*) → 浮点数

计算模拟中圆盘的总体积，并考虑掩码参数

yade._utils.getStress(*(float)volume=0*) → Matrix3

计算并返回 Love-Weber 应力张量 r:

$\sigma_{ij} = \frac{1}{V}\sum_b f_i^b l_j^b$，其中总和是对全部相互作用求和， f 为接触力， l 为分支向量（连接刚体中心）。排斥性接触时，应力为负值

力，即压缩。可以将 V 传递给函数。如果未传递，在周期情形下它将等于单元的体积；在非周期情形下则等于根据 utils.aabbDim() 推导出的体积。

yade._utils.**getStressAndTangent**(*[(float)volume=0[,(bool)*对称性*=True]]*) → 元组 使用与函数 getStress 相同的方程计算周期单元的整体应力。此外，切线算子根据 [Kruyt 和 Rothenburg1998]_ 发表的方程计算：

$$S_{ijkl} = \frac{1}{V}\sum_c (k_n n_i l_j n_k l_l + k_t t_i l_j t_k l_l)$$

参数

- **volume** (*float*) – 与函数 getStress 中的相同
- **symmetry** (*bool*) – 将张量设为对称。

返回 宏观应力张量和切线算子，类型为 py::tuple

yade._utils.**getStressProfile**(*(浮点数)*体积，*(*整数*)*单元数，*(浮点数)dz*，*(浮点数)*参考位置，*(*对象*)vPartAverageX*，*(*对象*)vPartAverageY*，*(*对象*)vPartAverageZ*）→元组

计算并返回应力张量深度分布，其中包括 Love-Weber 应力张量和考虑颗粒惯性影响的动态应力张量的贡献。对于每个定义的单元 z，应力张量为：

$\sigma^z_{ij} = \frac{1}{V}\sum_c f^c_i l^{c,z}_j - \frac{1}{V}\sum_p m^p u_i'^p u_j'^p$,
其中，第一个求和是对包含在单元 z 内或穿过单元 z 的接触进行的，f^c 是颗粒 1 作用于颗粒 2 的接触力，l^{c,z} 是从颗粒 2 指向颗粒 1 的分支向量中包含在该单元内的部分。第二个求和是对颗粒进行的，u'^p 是颗粒 p 相对于该点空间平均颗粒速度的速度波动（作为输入参数给出）。应力张量的表达式与 getStress 中给出的相同，另加惯性贡献。除此之外，它与 getStress 的主要区别在于，它给出应力张量的深度分布：从高度为 zRef 的参考水平面（输入参数）到高度为 zRef+nCell*dz 的平面（输入参数），按高度为 dz 的每个单元计算应力张量。对于 Love-Weber 应力贡献，计算时仅考虑分支向量中位于所考察单元内的部分。为验证这一公式，我们进行了检查：仅启用 Love-Weber 应力张量，并将不同高度处的所有贡献相加后，所得应力张量与使用 getStress 得到的相同。由于固定物体之间存在严重重叠，这一情况给我自己的使用带来了问题，因此我添加了一个条件，以排除固定物体对应力张量的贡献；如有需要，可以轻松停用该条件（与 getStress 比较时也应停用）。

yade._utils.**getStressProfile_contact**(*(float)*体积，*(int)*单元数，*(float)dz*，*(float)*参考位置）→ 与 getStressProfile 完全相同，只包含接触贡献。

yade._utils.**getTotalDynamicStress**(*[(float)volume=0]*) → Matrix3
计算总动态应力张量：$\sigma_D = -\frac{1}{V}\sum_p m^p u'^p \otimes u'^p$。_非周期性模拟必须提供体积。周期性模拟则从单元体积计算体积。

yade._utils.**getViscoelasticFromSpheresInteraction**(*(float)tc*，*(float)en*，*(float)es*）→ 字典
衰减 作用！此函数已弃用！根据分析结果计算黏弹性相互作用参数 cal

一对球体碰撞问题的求解：

$$k_n = \frac{m}{t_c^2}\left(\pi^2 + (\ln e_n)^2\right)$$
$$c_n = -\frac{2m}{t_c}\ln e_n$$
$$k_t = \frac{2}{7}\frac{m}{t_c^2}\left(\pi^2 + (\ln e_t)^2\right)$$
$$c_t = -\frac{2}{7}\frac{m}{t_c}\ln e_t$$

其中 k_n， c_n 是法向弹性系数和黏性系数， k_t， c_t 是剪切弹性系数和黏性系数。详情请参阅 [Pournin2001]。

参数

- **m** (*float*) – 球体质量 m
- **tc** (*float*) – 碰撞时间 t_c
- **en** (*float*) – 法向恢复系数 e_n
- **es** (*float*) – 切向恢复系数 e_s

返回字典，包含键 `kn` （ k_n 的值）、 `cn` （c_n）、 `kt` （k_t）、 `ct` （c_t）。

`yade._utils.growParticle`(*(int)*刚体标识符，*(float)*乘数[, *(bool)*更新质量=*True*]) → None
更改单个球体（待实现：单个团簇）的尺寸。如果 updateMass=为 True，则更新质量。

`yade._utils.growParticles`(*(float)*乘数[, *(bool)updateMass=True*[, *(bool)dynamicOnly=True*]]) → None
按乘数更改球体和团簇的尺寸。如果 updateMass=为 True，则更新质量和惯性。dynamicOnly=为 True 时，将选择动力体。

`yade._utils.highlightNone`() → None
重置所有刚体上的高亮显示。

`yade._utils.initMPI`() →无
初始化用于 Foam 耦合的 MPI 通信器

`yade._utils.inscribedCircleCenter`(*(三维向量)v1*，*(三维向量)v2*，*(三维向量)v3*) → 三维向量r3
返回由其顶点 *v1*、*v2*、*v3* 所确定三角形的内切圆中心。

`yade._utils.interactionAnglesHistogram`(*(int)axis*[, *(int)mask=0*[, *(int)bins=20*[, *(tuple)aabb=()*[, *(bool)sphSph=0*[, *(float)minProjLen=1e-06*]]]]]) → tuple

`yade._utils.intrsOfEachBody`() →列表
返回每个刚体相互作用的列表组成的列表

`yade._utils.kineticEnergy`([*(bool)findMaxId=False*]) → 对象
计算模拟的总动能，公式如下：

$$\sum \frac{1}{2}\left(m_i v_i^2 + \boldsymbol{\omega}(\mathbf{I}\boldsymbol{\omega}^T)\right).$$

对于非球形刚体，其惯性张量 **I** 会先转换到全局坐标系，再乘以 $\boldsymbol{\omega}$，因此该值应当是准确的。

`yade._utils.maxOverlapRatio`() →浮点数
返回 ScGeom中两个球体相互作用的最大重叠比。该比值的计算公式为 $\frac{u_N}{2(r_1 r_2)/r_1+r_2}$，其中 u_N 是当前的重叠距离，而 r_1， r_2 是接触的两个球体的半径。

yade._utils.momentum() → 三维向量 待办

yade._utils.negPosExtremeIds(*(int)*轴，*(*浮点数*)*距离因子）→ 元组
返回沿给定轴位于试样极值端的球体（仅限球体）标识符列表；距离因子将其半径相乘，因此不接触边界坐标的球体也可以被返回。

yade._utils.normalShearStressTensors([*(bool)compressionPositive=False*[, *(bool)*分割法向应力张量=否[, *(*浮点数*)*力阈值=非数]]]）→元组
Comp计算周期单元的总应力张量，并将其分解为两部分，其中一部分由法向力贡献，另一部分由剪切力贡献。其公式见 [Thornton2000], 公式 (3):

$$\sigma_{ij} = \frac{2}{V}\sum RN\mathbf{n}_i\mathbf{n}_j + \frac{2}{V}\sum RT\mathbf{n}_i\mathbf{t}_j$$

其中 V 为单元体积， R 为"接触半径"（在我们的实现中，是颗粒质心之间的当前距离）， **n** 为法向量， **t** 为垂直于 **n**的向量， N 和 T 分别为法向力和剪切力的范数。

参数

- **splitNormalTensor** (*bool*) – 若为真，该函数将根据强、弱力的两个子网络，把法向应力张量分成两部分返回。
- **thresholdForce** (*Real*) – 用于划分法向应力张量的阈值（例如，阈值为零时可区分拉伸力和压力）。

yade._utils.numIntrsOfEachBody() → 列表
返回每个刚体的相互作用数量列表

yade._utils.pointInsidePolygon（（元组）参数*1*，（对象）辐角*2*）→ 布尔值

yade._utils.porosity([*(float)volume=-1*]) → 浮点数 计算堆积孔隙率 $\frac{V-V_s}{V}$其中 V 为总体积， V_s 为球体体积。

参数 **volume** (*float*) – 总体积 V。对于周期性模拟，使用当前单元的体积。对于非周期性模拟，使用从 utils.aabbDim() 得出的值。出于兼容性考虑，在这种情况下也接受用户传入的正值。

yade._utils.ptInAABB(*(*三维向量*)arg1*，*(*三维向量*)arg2*，*(*三维向量*)arg3*）→ 布尔值
返回 True/False，表示点 p 是否位于由最小和最大角点确定的盒子内

yade._utils.scalarOnColorScale(*(*浮点数*)x*[, *(*浮点数*)xmin=0*[, *(*浮点数*)xmax=1*]]）→将标量变量映射到颜色刻度的三维向量。

参数

- **x** (*float*) – 函数所作用的标量值。
- **xmin** (*float*) – 颜色刻度的最小值；当 $x \leq xmin$ 时，返回值为 (0,0,1)，即 RGB 中的蓝色。
- **xmax** (*float*) – 最大值；当 $x \geq xmax$ 时，返回值为 (1,0,0)，即 RGB 中的红色。

返回 一个三维向量，其值取决于x 在 $[xmin$;*xmax*] 缩放范围内的相对位置。

yade._utils.setBodyAngularVelocity(*(int)id*, *(Vector3)angVel*) → None:
根据刚体的标识符和新的 Vector3r 设置刚体角速度。

参数

- **id** (*int*) – the body id.

- **angVel** (Vector3) – 所需更新后的角速度。

yade._utils.**setBodyColor**(*(int)id*,*(Vector3)*颜色）→ 无 根据刚体 ID 和新的 Vector3r 设置刚体颜色。

参数

- **id** (*int*) – 刚体 ID。
- **color** (Vector3) – 要更新为的颜色。

yade._utils.**setBodyOrientation**(*(int)id*,*(Quaternion)ori*）→ 无 根据刚体 ID 和新的 Quaternionr 设置刚体方向。

参数

- **id** (*int*) – 刚体 ID。
- **ori** (Quaternion) – 期望更新后的方向。

yade._utils.**setBodyPosition**(*(int)*标识符，*(Vector3)*位置[, *(str)*轴=*'xyz'*]）→ 无 根据刚体 ID 和新的 Vector3r 设置刚体位置。

参数

- **id** (*int*) – 刚体 ID。
- **pos** (Vector3) – 期望更新后的位置。
- **axis** (*str*) – 需要更新位置所沿的轴（例如：如果 axis==”xy” 且 pos== Vector3r(r0,r1,r2)，则 r2 将被忽略，z 方向的位置不会更新）。

yade._utils.**setBodyVelocity**(*(整数)*标识符，*(三维向量)*速度[, *(字符串)*轴=*'xyz'*]）→ 无 根据刚体 ID 和新的 Vector3r 设置刚体速度。

参数

- **id** (*int*) – 刚体 ID。
- **vel** (Vector3) – 期望更新后的速度。
- **axis** (*str*) – 必须更新速度的轴（例如：如果 axis==”xy”且 vel== Vector3r(r0,r1,r2)，则 r2 将被忽略，z 方向的速度也不会更新）。

yade._utils.**setContactFriction**(*float*) → None 修改材料类和现有接触中的摩擦角（以弧度为单位）。非动态刚体的摩擦不会被修改。

yade._utils.**setNewVerticesOfFacet**(*(刚体)b*，*(三维向量)v1*，*(三维向量)v2*，*(三维向量)v3*）→ None 将给定面片的新顶点（以全局坐标表示）设为指定值。

yade._utils.**setRefSe3**() → None
将所有刚体的参考位置和取向设为其当前的位置和取向。

yade._utils.**shiftBodies**(*(列表)ids*，*(三维向量)*平移）→ 浮点数
平移 ids 列出的刚体，但不更新其速度。

yade._utils.**spiralProject**(*(Vector3)pt*, *(float)dH_dTheta*[, *(int)axis=2*[, *(float)periodStart=nan*[, *(float)theta0=0*]]]）→ tuple

yade._utils.**sumFacetNormalForces**(*(对象)ids*[, *(整数)*轴=*-1*]）→ 浮点数
对指定刚体上的力的大小求和（形状必须为面片类型），仅考虑垂直于每个面片表面的力的分量；如果面片法向量；如果轴的值为正，则使用指定的轴（0=x，1=y， 2=z）代替面片法向量。

yade._utils.**sumForces**(*(列表)*标识符，*(三维向量)*方向）→浮点数 返回作用于具有指定标识符的刚体上的力的摘要，并将其投影到方向向量上。

yade._utils.**sumTorques**（列表）标识符，（三维向量）轴，（三维向量）*axisPt*）→ 浮点数：求给定标识符对应的刚体所受力和力矩之和，计算时以点*axisPt* 及其方向轴所确定的轴为参照。

yade._utils.**totalForceInVolume**() → 元组

Return 对所有相互作用的力求和，并计算平均各向同性刚度，返回为元组 (三维向量,浮点数)

yade._utils.**unbalancedForce**(*(bool)useMaxForce=False*) → 浮点数

计算刚体所受平均力（若*useMaxForce*为真，则取最大力）摘要值与相互作用力模的平均值之比。对于完全静态平衡，所有刚体所受力的摘要值为零（因为相互作用力彼此抵消，不会使颗粒产生加速度）；随着模拟趋于稳定，该比值会趋近于零，但由于有限精度计算，永远不会达到零。足够小的值例如可以是 1e-2 或更小，具体取决于所需的平衡程度。

yade._utils.**voidratio2D**(*(float)zlen=1*) → 浮点数 计算二维堆积空隙比 $\frac{V-V_s}{V_s}$其中 V 为总体积，V_s 为圆盘体积。

参数 **zlen**(*float*) – 第三个方向上的长度。

yade._utils.**voxelPorosity**(*(int)resolution=200, (三维向量)起点=三维向量(0,0,0), (Vector3)end=Vector3(0, 0, 0)*) → float

计算堆积孔隙率 $\frac{V_v}{V}$其中 V为从起点到终点的指定体积，V_v 为落入任意球体内部的体素体积。计算方法是将整个体积划分为由体素组成的密集网格（采用给定分辨率），并统计落入任意球体内部的体素数量。该方法可以计算整个试样中任意给定子体积的孔隙率，并能正确排除不位于指定体积内的球体部分。

参数

- **resolution** (*int*) – 体素网格分辨率，大于分辨率=1600的值需要64位操作系统，因为会使用超过4GB的内存；分辨率=800将使用500MB的内存。
- **start** (Vector3) – 体积的起始角点。
- **end** (Vector3) – 体积的结束角点。

yade._utils.**wireAll**() →无

设置 *Shape::wire* 全部 刚体为 True，仅以线框方式渲染 。

yade._utils.**wireNoSpheres**() → 无 集 *Shape::wire* 对非球形刚体设为 True（面片、墙）。

yade._utils.**wireNone**() → 无

将 *Shape::wire* 对全部刚体设为 False，将其渲染为实体。

2.4.18 yade.ymport模块

从各种格式导入几何体（'import' 是 Python关键字，因此使用名称 'ymport'）。

yade.ymport.**ele**(*nodeFileName*、*eleFileName*、平移*=(0*、*0*、*0)*、缩放*=1.0*、***kw*)从 .ele 文件导入四面体网格，并返回创建的四面体列表。

参数

- **nodeFileName** (*string*) – .node 文件名
- **eleFileName** (*string*) – .ele 文件名
- **shift** (*(float,float,float)|Vector3*) – (X,Y,Z) 参数移动试样。

- **scale** (*float*) – 因子缩放给定的数据。
- ****kw** – （未使用的关键字参数）会传递给*utils.polyhedron*

yade.ympport.gengeo(*mntable*，平移=*Vector3(0,0,0)*，缩放=*1.0*，***kw*)从 LSMGenGeo 库导入几何体并创建球体。自 2012 年起，该软件包已在 Debian/Ubuntu 中提供，名称为 python-demgengeo http://packages.qa.debian.org/p/python-demgengeo.html

参数

mntable*：mntable** 由 LSMGenGeo 库创建的对象，参见示例平移： [**浮点数,浮点数,浮点数**] [X,Y,Z] 参数用于移动试样。缩放：**浮点数** 因子用于缩放给定数据。*kw*：（未使用的关键字参数）**会传递给*utils.sphere*

LSMGenGeo 库允许创建具有给定 [Rmin:Rmax] 的球体堆积，且试样内部为零应力。可用于采矿岩石模拟。

示例：examples/packs/packs.py，在 exam-ples/test/genCylLSM.py 中使用 LSMGenGeo 库。

- https://answers.launchpad.net/esys-particle/+常见问题/877
- http://www.access.edu.au/lsmgengeo_python_doc/current/pythonapi/html/GenGeo 模块.html
- https://svn.esscc.uq.edu.au/svn/esys3/lsm/contrib/LSMGenGeo/

yade.ympport.gengeoFile(文件名=*'file.geo'*，平移=三维向量*(0，0，0)*，缩放=*1.0*，方向=四元数*((1，0，0)，0)*，***kw*)

从 LSMGenGeo .geo 文件导入几何体并创建球体。自 2012 年起，该软件包可在 Debian/Ubuntu 中获取，名称为 python-demgengeo http://packages.qa.debian.org/p/python-demgengeo.html

参数

文件名：**字符串**包含 4 列的文件 [x、y、z、半径]。

平移：**三维向量**三维向量(X,Y,Z)参数用于移动试样。

缩放：**浮点数**因子用于缩放给定的数据。

方向：**四元数**导入的几何体的方向

****kw*：（未使用的关键字参数）**将传递给*utils.sphere*

返回球体列表。

LSMGenGeo 库可以创建内部具有零应力、半径范围为 [Rmin:Rmax] 的球体堆积。可用于采矿岩石模拟。

示例： examples/packs/packs.py，LSMGenGeo 库的用法见 exam-ples/test/genCylLSM.py。

- https://answers.launchpad.net/esys-particle/+常见问题/877
- http://www.access.edu.au/lsmgengeo_python_doc/current/pythonapi/html/GenGeo 模块.html
- https://svn.esscc.uq.edu.au/svn/esys3/lsm/contrib/LSMGenGeo/

yade.ympport.gmsh(*meshfile='file.mesh'*，*shift=Vector3(0，0，0)*，*scale=1.0*，*orienta- tion=Quaternion((1，0，0)，0)*，***kw*)从 .mesh 文件导入几何体并创建面片。

参数

平移： [**浮点数,浮点数,浮点数**] [X、Y、Z] 参数会移动试样。

缩放： **浮点数** 因子用于缩放给定的数据。

方向：**四元数**导入网格的方向

***kw*：（未使用的关键字参数）会传递给*utils.facet*

返回构成试样的面片列表。

网格文件可以使用GMSH轻松创建。示例见examples/packs/packs.py

其他网格文件示例可从以下地址下载：http://www-roc.inria.fr/gamma/download/download.php

yade.ymport.gts(网格文件，平移=三维向量*(0，0，0)*，缩放=*1.0*，
***kw*)读取给定的 GTS 格式网格文件。

参数

网格文件：**字符串** 输入文件的名称。

平移： [**浮点数,浮点数,浮点数**] [X、Y、Z] 参数用于移动试样。

缩放：**浮点数** 因子用于缩放给定的数据。

***kw*：（未使用的关键字参数）会传递给*utils.facet*

返回面片列表。

yade.ymport.iges(文件名，平移=*(0，0，0)*，缩放=*1.0*，返回连通性表=*False*，***kw*)
从 .igs 文件导入三角网格，返回已创建的面片列表。

参数

- **fileName** (*string*) – IGES 文件名
- **shift** (*(float,float,float)|Vector3*) – (X、Y、Z) 参数用于移动试件。
- **scale** (*float*) – 因子用于缩放给定数据。
- ****kw** –（未使用的关键字参数）会传递给*utils.facet*
- **returnConnectivityTable** (*bool*) – 如果为 True，除面片外，还会返回节点（由 (x,y,z) 节点坐标组成的列表）和元素（由 (id1,id2,id3) 元素节点标识符组成的列表）。如果为 False（默认值），则仅返回面片

yade.ymport.stl(文件，动态=*None*，固定=*True*，线框=*True*，颜色=*None*，高亮=*False*，无Bound=*False*，
材料=*-1*，缩放=*1.0*，平移=*Vector3(0, 0, 0)*)导入一个 *.stl* 几何体，并将其表示为一组面片形状的刚体。

参数

- **file** (*string*) – 用作几何输入的 .stl 文件
- **dynamic** (*bool*) – 控制*Body.dynamic*
- **fixed** (*bool*) – 控制 *Body.dynamic* (设为固定 = True 时强制*Body.dynamic* = False) 如果未提供 动态 属性
- **wire**(*bool*) – 渲染选项，传递给 *Facet.wire*
- **color**– 渲染选项，传递给 *Facet.color*
- **highlight** (*bool*) – 渲染选项，传递给 *Facet.highlight*
- **noBound** (*bool*) – 将 *Body.bounded* 设为 False（若为 True 则阻止碰撞检测，反之亦然）
- **material** – 定义材料属性，参见定义材料以了解用法
- **scale** (*float*) – 缩放因子，例如在 > 1 时用于放大几何体
- **shift** (Vector3) – 用于转换几何体

R返回一个对应的面片形状的 b 刚体

yade.ymport.**text**(文件名,平移=三维向量*(0，0，0)*，缩放=*1.0*，***kw*)从文件加载球体坐标，并返回相应刚体的列表；这些刚体可通过 O.bodies.append() 添加到模拟中。

参数

- **filename** (*string*) – 包含 4 列的文件： [x、y、z、半径]。
- **shift** (*[float,float,float]*) – [X、Y、Z] 参数用于平移试样。
- **scale** (*float*) – 因子用于缩放给定的数据。
- ****kw** –（未使用的关键字参数）会传递给 *utils.sphere*

返回 球体列表。

以 # 开头的行会被跳过

yade.ymport.**textClumps**(*fileName*，平移=三维向量*(0，0，0)*，离散化=*0*，朝向=四元数*((1，0，0)，0)*，缩放=*1.0*，***kw*)从文件中加载团簇成员，文件格式由 `format` 参数 选择，并将其插入模拟中。

参数

- **filename** (*str*) – 文件名
- **format** (*str*) – 所选输入格式。已支持 `'x_y_z_r'``(default)、``'x_y_z_r_clumpId'`
- **shift** (*[float,float,float]*) – [X、Y、Z] 参数移动试样。
- **scale** (*float*) – 因子缩放给定的数据。
- ****kw** –（未使用的关键字参数）传递给 *utils.sphere*

返回球体列表。

以 # 开头的行会被跳过

yade.ymport.**textExt**(文件名，格式=*'x_y_z_r'*，平移=三维向量*(0，0，0)*，缩放=*1.0*，属性=*[]*，***kw*)根据 `format` 参数指定的格式从文件中加载球体坐标，并返回相应刚体的列表；该列表可能会通过 O.bodies.append() 添加到模拟中。

参数

- **filename** (*str*) – 文件名
- **format** (*str*) – 选定的输入格式。已支持 `'x_y_z_r'``(default)、``'x_y_z_r_matId'、 'x_y_z_r_attrs'`
- **shift** (*[float,float,float]*) – [X、Y、Z] 参数用于移动试样。
- **scale** (*float*) – 因子用于缩放给定的数据。
- **attrs** (*list*)–如果使用了 export.textExt(format='x_y_z_r_attrs')，则从文件读取属性（采用"按引用传递"样式）
- ****kw** –（未使用的关键字参数）会传递给 *utils.sphere*

返回 球体列表。

以 # 开头的行会被跳过

yade.ymport.**textFacets**(文件名，格式=*'x1_y1_z1_x2_y2_z2_x3_y3_z3'*，平移=三维向量*(0，0，0)*，缩放=*1.0*，属性=*[]*，***kw*)根据 `format` 参数选择的格式从文件加载面片坐标，并返回相应刚体的列表；这些刚体可能会通过 O.bodies.append() 插入到模拟中。

参数

- **filename** (*str*) – 文件名
- **format** (*str*) – 所选输入格式。已支持 'x1_y1_z1_x2_y2_z2_x3_y3_z3'``(default)、 ``'x1_y1_z1_x2_y2_z2_x3_y3_z3_matId'、'id_x1_y1_z1_x2_y2_z2_x3_y3_z3_matId' 或 'x1_y1_z1_x2_y2_z2_x3_y3_-z3 attrs'_
- **shift** (*[float,float,float]*) – [X、Y、Z] 参数用于移动试样。
- **scale** (*float*) – 因子用于缩放给定的数据。
- **attrs** (*list*) – 从文件读取的属性（以“按引用传递”的样式）
- ****kw** –（未使用的关键字参数）会传递给*utils.facet*

返回面片列表。

以 # 开头的行会被跳过

yade.ymport.**textPolyhedra**(文件名，材料，平移=三维向量*(0，0，0)*，缩放=*1.0*，方向=四元数*((1，0，0)，0)*，***kw*)从文本文件加载多面体。

参数

- **filename** (*str*) – 文件名。预期的文件格式与 ex-port.textPolyhedra 输出的格式相同。
- **shift** (*[float,float,float]*) – [X、Y、Z] 参数用于平移试样。
- **scale** (*float*) – 因子用于缩放给定的数据。
- **orientation** (*quaternion*) – 导入多面体的方向
- ****kw** –（未使用的关键字参数）会传递给 多面体_*utils.polyhedra*

返回多面体列表。

以 # 开头的行会被跳过

yade.ymport.**unv**(文件名,平移=*(0,0, 0)*, 缩放=*1.0*, 返回连通性表=*False*, ***kw*)从 UNV 文件导入几何体，并返回创建的面片列表。

参数 字符串 文件名 UNV 文件的名称

参数 (浮点数,浮点数,浮点数)|三维向量平移(X,Y,Z) 参数用于移动试样。

参数 浮点数 缩放因子用于缩放给定数据。

参数 **kw (未使用的关键字参数)会传递给 *utils.facet*

参数 布尔值 returnConnectivityTable 如果为 True，除面片外还会返回节点（由 (x,y,z) 节点坐标组成的列表）和元素（由 (id1,id2,id3) 单元节点标识符组成的列表）。如果为 False（默认值），则仅返回面片

UNV 文件主要用于有限元法分析（由 OOFEM 和 Abaqus使用），但三角形单元可以作为面片导入。这些文件可以使用开源免费软件（例如 Salome）创建。

示例： examples/test/unv-read/unvRead.py。

2.5 安装

- Linux 系统：可以通过软件包（预编译二进制文件）或源代码安装 Yade。具体选择取决于您的需求：如果您不打算修改 Yade 本身，通过软件包安装会更简单。反之，则必须下载并安装源代码。
- 其他操作系统：跳转到本页最后一节。

- 64位操作系统为必需；不支持 32 位（i386）。

2.5.1 软件包

稳定版软件包

自2011年以来，所有 Ubuntu（从 11.10 Oneiric 开始）和 Debian（从 Wheezy 开始）版本都在其主仓库中提供 Yade。这里只有稳定版。要安装 Yade，请运行以下命令：

```
sudo apt-get install yade
```

之后，通常可以使用 command `yade` 或 `yade-batch`启动 Yade。

这张图像 显示了 Yade 在某些代码仓库中的版本及其最新状态。

每日软件包

为目前所有受支持的 Debian 和 Ubuntu 版本提供了比稳定版更新更频繁的预编译软件包，可在 yade-dem.org/packages 获取。
这些是“每日”版本的软件包，会定期更新，因此包含所有新添加的功能。

要安装每日版本，您需要将代码仓库添加到 /etc/apt/sources.list。

- Debian 9 **拉伸**:

```
sudo bash -c 'echo "deb http://www.yade-dem.org/packages/ stretch main" >> /etc/apt/
→sources.list'
```

- Debian 10 **buster**:

```
sudo bash -c 'echo "deb http://www.yade-dem.org/packages/ buster main" >> /etc/apt/
→sources.list'
```

- Debian 11 **bullseye**:

```
sudo bash -c 'echo "deb http://www.yade-dem.org/packages/ bullseye main" >> /etc/apt/
→sources.list'
```

- Debian 12 **bookworm**:

```
sudo bash -c 'echo "deb http://www.yade-dem.org/packages/ bookworm main" >> /etc/apt/
→sources.list'
```

- Ubuntu 16.04 **xenial**:

```
sudo bash -c 'echo "deb http://www.yade-dem.org/packages/ xenial main" >> /etc/apt/
→sources.list'
```

- Ubuntu 18.04 **bionic**:

```
sudo bash -c 'echo "deb http://www.yade-dem.org/packages/ bionic main" >> /etc/apt/
→sources.list'
```

- Ubuntu 20.04 **focal**:

```
sudo bash -c 'echo "deb http://www.yade-dem.org/packages/ focal main" >> /etc/apt/sources.
→list'
```

- Ubuntu 22.04 **jammy**:

```
sudo bash -c 'echo "deb http://www.yade-dem.org/packages/ jammy main" >> /etc/apt/sources.
→list'
```

添加 PGP 将键 AA915EEB 设为受信任并安装 `yade daily`:

```
wget -O - http://www.yade-dem.org/packages/yadedev_pub.gpg | sudo apt-key add -
sudo apt-get update
sudo apt-get install yadedaily
```

之后，通常可以使用 command `yadedaily` 或 `yadedaily-batch`启动 Yade。较旧发行版上的 `yadedaily`可能会因随特定发行版提供的库版本较旧而禁用某些功能。

用于打包内容的 Git 仓库可在 GitLab 上找到。

如果你 不再需要 `yadedaily`-package，只需删除相应的直线在 /etc/apt/sources.list 中以及软件包本身:

```
sudo apt-get remove yadedaily
```

要从密钥环中移除我们的键，请执行以下 command:

```
sudo apt-key remove AA915EEB
```

每日版和稳定版 Yade 版本可以共存而不会发生任何冲突，也就是说，你可以同时使用 `yade` 和 `yadedaily`。

2.5.2 Docker

Yade 可通过 Docker 镜像安装，这些镜像每日构建。镜像包含稳定版和每日版软件包。Docker 镜像基于已支持的发行版:

- Debian 9 **拉伸**:

  ```
  docker run -it registry.gitlab.com/yade-dev/docker-prod:debian-stretch
  ```

- Debian 10 **buster**:

  ```
  docker run -it registry.gitlab.com/yade-dev/docker-prod:debian-buster
  ```

- Debian 11 **bullseye**:

  ```
  docker run -it registry.gitlab.com/yade-dev/docker-prod:debian-bullseye
  ```

- Debian 12 **bookworm**:

  ```
  docker run -it registry.gitlab.com/yade-dev/docker-prod:debian-bookworm
  ```

- Ubuntu 16.04 **Xenial**:

  ```
  docker run -it registry.gitlab.com/yade-dev/docker-prod:ubuntu16.04
  ```

- Ubuntu 18.04 **Bionic**:

  ```
  docker run -it registry.gitlab.com/yade-dev/docker-prod:ubuntu18.04
  ```

- Ubuntu 20.04 **Focal**:

  ```
  docker run -it registry.gitlab.com/yade-dev/docker-prod:ubuntu20.04
  ```

- Ubuntu 22.04 **Jammy**:

  ```
  docker run -it registry.gitlab.com/yade-dev/docker-prod:ubuntu22.04
  ```

容器拉取并运行后，可以检查 Yade 的功能:

```
yade --test
yade --check
yadedaily --test
yadedaily --check
```

2.5.3 源代码

当您想要添加或修改本构定律、引擎、函数等时，从源代码安装是合适的。安装最新的主干版本即可使用新添加的功能，这些功能尚未包含在打包版本中。

下载

如果要从源代码安装，可以选择安装发行版（带编号的固定版本）或当前的开发版本（开发者会频繁更新）。如果您想修改源代码，应下载开发版本（称为 `trunk`），因为您可能会遇到一些问题，而开发者会修复这些问题。发行版本不会更新（关键且易于修复的缺陷所导致的更新除外），但通常比主干版本更稳定。

1. 发行版可从下载页面以压缩归档文件的形式下载。解压归档文件后，您会得到一个包含源代码的目录。

2. 开发版本（`trunk`）可从 GitLab 上的代码仓库获取。

我们使用Git（ `git` command）进行代码管理（在系统上安装 `git` 软件包并创建一个GitLab 账户）：

```
git clone git@gitlab.com:yade-dev/trunk.git
```

将下载 `trunk`的整个代码仓库。请访问*GitLab 上的 Yade*，了解如何使用 `git`进行协作。

或者，也可以通过 HTTPS 只读检出，无需 GitLab 账户（如果您不想修改主干版本，这种方式更简单）：

```
git clone https://gitlab.com/yade-dev/trunk.git
```

对于位于防火墙之后的用户，可以从我们的GitLab 代码仓库下载压缩归档文件。

发行版和主干版本的源代码编译方式完全相同。

先决条件

Yade 运行依赖若干外部软件；在开始编译之前会对其进行检查。其中一些是可选的。

- CMake 构建系统
- GCC 编译器（g++）；其他编译器无法使用；需要 g++>=4.2 才能支持 OpenMP
- Boost 1.47 或更高版本
- Qt库
- freeglut3
- libQGLViewer
- Python，NumPy，IPython，Sphinx，mpi4py
- matplotlib

- Eigen 代数库（最低要求版本 3.2.1）
- GDB调试器
- SQLite3数据库引擎
- VTK库（可选，但建议安装）
- CGAL库（可选）
- SuiteSparse 稀疏代数库（用于流体耦合，可选，需要 Eigen>=3.1）
- OpenBLAS 经过优化和并行化的标准 BLAS 替代方案+LAPACK（流体耦合*FlowEngine*，可选）
- Metis 矩阵预处理（流体耦合，可选）
- OpenMPI 用于并行分布式计算的库（用于 MPI 和 OpenFOAM 耦合，可选）
- python3-mpi4py 适用于 Python 的 MPI（用于 MPI，可选）
- coin-or COIN-OR 线性规划求解器（用于 *PotentialBlock*，可选）
- mpfr 在 `C++` 和 mpmath 在 `python` 中用于高精度 `Real` 或用于 CGAL 精确谓词（可选）
- MPC 是面向复数的 MPFR 扩展。它与 MPFR 配合使用。

上面列表中的大多数软件包很可能已经为你的发行版打包。如果遇到某些软件包不可用的错误（例如，软件包 libmetis-dev 不可用），可能需要添加 Yade 外部 PPA，地址为 https://launchpad.net/~yade-users/+archive/external （见下文），以及 http://www.yade-dem.org/packages （见本页顶部）：

```
sudo add-apt-repository ppa:yade-users/external
sudo apt-get update
```

以下 命令需要在相应发行版的命令行中执行 n.
只需复制 并粘贴到终端中。注意，执行这些命令需要 root 权限。

- **Ubuntu 20.04、18.04、Debian 9、10、11** 及其衍生版本：

```
sudo apt install cmake git freeglut3-dev libloki-dev libboost-all-dev fakeroot \
dpkg-dev build-essential g++ python3-dev python3-ipython python3-matplotlib \
libsqlite3-dev python3-numpy python3-tk gnuplot libgts-dev python3-pygraphviz \
libvtk6-dev libeigen3-dev python3-xlib python3-pyqt5 pyqt5-dev-tools python3-mpi4py \
python3-pyqt5.qtwebkit gtk2-engines-pixbuf python3-pyqt5.qtsvg libqglviewer-dev-qt5 \
python3-pil libjs-jquery python3-sphinx python3-git libxmu-dev libxi-dev libcgal-dev \
help2man libbz2-dev zlib1g-dev libopenblas-dev libsuitesparse-dev \
libmetis-dev python3-bibtexparser python3-future coinor-clp coinor-libclp-dev \
python3-mpmath libmpfr-dev libmpfrc++-dev libmpc-dev
```

•对于 **Ubuntu 16.04** `libqglviewer-dev-qt5` 应替换为 `libqglviewer-dev` ，并将 `python3-ipython` 替换为 `ipython3`。

• 软件包 `python3-mpmath libmpfr-dev libmpfrc++-dev` 仅当需要进行高精度计算时，才需要上述列表中的软件包。只有在使用mpfr 时，才需要后两个软件包。详情请参阅高精度文档。

•构建文档（下文将解释 `make doc` 的调用方式）还需要额外的软件包`texlive-xetex` 。在某些多语言系统上，可能会出现错误 `Building format(s)--all. This may take some time... fmtutil failed.` ；在这种情况下，需要软件包`locales-all`。

某些软件包（例如 CMake、eigen3）是必需的，另一些则是可选的。请留意 `cmake` 在配置步骤中显示的注释和警告/错误。如果缺少的软件包是可选的，Yade 的某些功能将被禁用（请参阅配置结束时显示的消息）。

此处列出的一些软件包较新，您的发行版中可能没有（例如 libmetis-dev）。您可以从 yade-dem.org/packages或我们的 外部个人软件包档案安装。如果未安装，相关功能将自动禁用。

如果您使用的发行版不是 Debian 或其衍生版，则应安装上面列出的软件包。其他发行版中的软件包名称可能与 Debian 软件包的名称不同。

警告： 如果您使用的是 Ubuntu 14.04 Trusty，则需要添加 -DCMAKE_CXX_FLAGS=-frounding-math，以便在编译的配置步骤中使用（见下文），或者安装 libcgal-dev 我们的 外部个人软件包档案。否则，在 AMD64 架构上会出现以下错误：

```
terminate called after throwing an instance of 'CGAL::Assertion_exception'
what():  CGAL ERROR: assertion violation!
Expr: -CGAL_IA_MUL(-1.1, 10.1) != CGAL_IA_MUL(1.1, 10.1)
File: /usr/include/CGAL/Interval_nt.h
Line: 209
Explanation: Wrong rounding: did you forget the  -frounding-math  option if you use GCC (or␣
 → -fp-model strict  for Intel)?
Aborted
```

编译

您应创建一个独立的构建目录，用于配置 Yade 并编译源代码。以下是目录结构示例：

```
myYade/                 ## base directory
        trunk/          ## folder for source code in which you use git
        build/          ## folder in which the sources will be compiled; build-directory; use␣
→cmake here
        install/        ## install folder; contains the executables
```

T然后，在此构建目录中调用 cmake 以配置编译过程：

```
cmake -DCMAKE_INSTALL_PREFIX=/path/to/installfolder /path/to/sources
```

对于上述目录结构，请在“build”目录中调用以下 command：

```
cmake -DCMAKE_INSTALL_PREFIX=../install ../trunk
```

可以使用以下语法在同一行中配置其他选项：

```
cmake -DOPTION1=VALUE1 -DOPTION2=VALUE2
```

例如：

```
cmake -DENABLE_POTENTIAL_BLOCKS=ON
```

以下 CMake 选项可用：（请参阅 源代码 以获取最新列表）

- CMAKE_INSTALL_PREFIX：Yade 的安装路径（默认为 /usr/local）
- LIBRARY_OUTPUT_PATH：库的安装路径（默认为 lib）
- DEBUG：以调试模式编译（默认为 OFF）
- 最大值_日志_级别：设置 日志_* 使用 ENABLE 编译的宏_日志记录器（默认为 5）
- CMAKE_VERBOSE_MAKEFILE：在编译期间输出其他信息（默认为 OFF）
- SUFFIX：后缀，添加在二进制文件名之后（默认使用版本号）

- NOSUFFIX：不要在二进制文件名之后添加后缀（默认关闭）
- YADE_版本：显式设置版本号（默认从 Git 目录中定义）
- 启用_ASAN：地址消毒器可检测内存错误、内存泄漏、堆损坏错误和越界访问（但速度较慢）

- 启用_CGAL：启用 CGAL 选项（默认开启）
- 启用_复数_MP：使用 Boost 多精度复数，否则使用 `ComplexHP<N>`。`std::complex<RealHP<N>>`。请参阅 高精度文档 了解更多详情。（若条件允许则默认开启：需要 Boost >= 1.71）

- 启用_DEFORM：启用恒体积变形引擎（默认关闭）
- 启用_FAST_NATIVE：使用包含 `-Ofast` 和`-mtune=native`的最高级优化编译器标志。注意：`native` 表示代码只能在编译时所用的同一处理器类型上**运行**。实测速度提升为 2%（低于标准差测量误差）；使用 Clang 编译器时，速度提升超过 5%。（默认关闭）

- 启用_FEMLIKE：启用类有限元网格实体（默认启用）
- 启用_GL2PS：启用 GL2PS 选项（默认启用）
- 启用_GTS：启用 GTS 选项（默认启用）
- 启用_GUI：启用图形用户界面选项（默认启用）
- 启用_LBMFLOW：启用 LBMFLOW 选项，格子玻尔兹曼方法_引擎（默认启用）
- 启用_LS_DEM：启用水平集 形状描述（默认开启）
- 启用_LINSOLV：启用 LINSOLV选项（默认开启）
- 启用_LIQMIGRATION：启用 LIQMIGRATION选项，详见 [Mani2013] （默认关闭）

- 启用_LOGGER：使用 boost::log 库进行日志记录，为每个类分别记录（默认开启）
- 启用_MASK_ARBITRARY：启用 MASK_ARBITRARY 选项（默认关闭）
- 启用_MPFR：在 mpfr 中使用 `C++` 和 mpmath 中使用 `python`。可用于更高精度计算`Real` 或用于CGAL精确谓词（默认关闭）
- 启用_MPI：启用 MPI 环境和通信；分布式内存和 Yade-OpenFOAM耦合需要此功能（默认启用）

- 启用_OAR：为基于OAR的任务调度器生成脚本（默认关闭）
- 启用_OPENMP：启用 OpenMP 并行选项（默认启用）
- 启用_PARTIALSAT：启用部分饱和黏土引擎（正在开发中，默认启用）

- 启用_PFVFLOW：启用 PFVFLOW选项，即 FlowEngine（默认启用）
- 启用_POTENTIAL_BLOCKS：启用势块选项（默认启用）
- 启用_势粒子_：启用势粒子选项（默认开启）
- 启用_性能分析：启用性能分析，例如显示更多指标，从而可以确定代码中的性能瓶颈（默认关闭）

- 启用_实数_高精度：允许在 `Real` 的 `RealHP<2>`、`RealHP<4>` 或 `RealHP<N>` 计算量大的截面中使用双精度、四倍精度或更高精度。 `C++` 代码。详情请参阅高精度文档（默认开启）。

- 启用_SPH：启用SPH选项，即平滑粒子流体动力学（默认关闭）
- 启用_热力：启用热引擎（默认开启，实验性）”
- 启用_TWOPHASEFLOW：启用两相流选项、两相流引擎（默认开启）

- 启用_USEFUL_错误：启用实用的编译器错误提示，这对无错误开发大有帮助（默认开启）
- 启用_VTK：启用 VTK 导出选项（默认开启）
- 实数_精度_位数，实数_小数_位数：指定其中任意一项，即可使用自定义的计算精度 `Real` 类型。默认使用 `double` （64 位，15 位小数）精度作为 `Real` 类型。更多详情请参见高精度文档。
- 运行时前缀：用于打包，当安装目录与运行时目录不同时使用（默认值为 /usr/local）
- 向量化：在 Eigen3 库中启用向量化和对齐，属于实验性功能（默认关闭）
- 使用_Qt 5：使用 Qt 5 图形用户界面（默认开启）
- CHOLMOD_GPU 将 Yade 链接到自定义的 SuiteSparse 安装，并启用 GPU 加速的 PFV（默认关闭）
- SUITESPARSEPATH：使用自定义 SuiteSparse 安装时，将此变量定义为相应的路径
- PYTHON_VERSION：强制使用指定的 Python 版本，例如 `-DPYTHON_VERSION=3.5`。设置为 -1 可自动使用系统上的最新版本（默认为 -1）

可以禁用全部选项来创建精简构建：

```
cmake -DDISABLE_ALL=ON
```

在这种情况下，所有可用选项都将被关闭。此时，可以显式启用某些必需的选项：

```
cmake -DDISABLE_ALL=ON -DENABLE_VTK=ON
```

如需使用 CMake 的更多扩展参数，请参阅相应的文档：https://cmake.org/documentation。

警告： 仅支持 Qt 5。在 Debian/Ubuntu 操作系统上，版本为 2.6.3 及更高版本的 libQGLViewer 是针对 Qt 5 编译的（对于其他操作系统，请参阅发行版的软件包档案）。如果混用不同版本的 Qt，启动 Yade 后会立即出现 `Segmentation fault` 。要提供 Qt 5 所需的构建依赖项，请安装 `python-pyqt5 pyqt5-dev-tools`。

如果 CMake 顺利完成且没有错误，您将在最后看到所有已启用和已禁用的选项。然后使用以下命令开始实际的编译过程：

```
make
```

编译过程可能需要相当长的时间，请耐心等待。如果您使用的是多核系统，可以使用参数 `-j` 来加快编译速度，并将编译任务分配到多个核心上。例如，对于 4 核计算机，将参数设为 `-j4` 是比较合理的。请注意，Yade 编译时每个核心大约需要 3 GB 内存，否则将使用交换文件，编译时间会大幅增加。

安装通过以下命令执行：

```
make install
```

`install` command 实际上也会在源文件发生更改时重新编译。因此，没有必要非得分开输入这两条命令。如果您没有权限写入目标文件夹，可能会收到 make 错误。这些错误并不严重，但如果没有写入权限，Yade 将不会安装到 /usr/local/bin/。

编译成功完成后，可进入 /path/to/installfolder/bin 并调用 Yade 来启动新构建的版本（基于版本 yade-2014-02-20.git-a7048f4）：

```
cd /path/to/installfolder/bin
./yade-2014-02-20.git-a7048f4
```

要构建文档，首先应执行 command `make install`，然后执行 `make doc` 来构建。生成的文件将存储在当前安装目录 /path/to/installfolder/share/doc/yade-your-version 中。再次提醒，安装到 /usr/local/share/doc/ 需要写入权限。要打开本地文档，请进入 html 文件夹，并使用浏览器打开文件 index.html。

`make manpage` command 会生成手册页并将其移至标准位置。 `make check` command 会执行标准 test，以检查已编译程序的功能。

Yade 不仅可以使用 GCC 编译器进行编译，还可以使用 Clang LLVM 编译器的前端。为此，请在检测要使用的 C 和 C++++ 编译器时设置环境变量 CC 和 CXX:

```
export CC=/usr/bin/clang
export CXX=/usr/bin/clang++
cmake -DOPTION1=VALUE1 -DOPTION2=VALUE2
```

Clang 目前不支持 OpenMP 并行化，因此该功能将被禁用。

已支持的 Linux 发行版

当前已支持[1] Linux 发行版及其对应的 Docker 文件 如下:

- Ubuntu 16.04 xenial
- Ubuntu 18.04 bionic
- Debian 9 stretch
- Debian 10 buster
- openSUSE 15

以下是用于准备 Linux 发行版和环境，以便安装和测试 Yade 的 bash 命令。每次合并到主进程后，都会通过 GitLab 持续集成服务自动执行这些说明。这可确保 Yade 在这些 Linux 发行版上始终正常运行。事实上，也可以按照以下顺序逐步执行这些说明，手动安装 Yade:

1. 在相应的 Dockerfile 中执行 Bash 命令，以安装必要的软件包；2. 执行 `git clone https://gitlab.com/yade-dev/trunk.git`；3. 然后执行相应发行版的 `cmake_*` 命令，见 .gitlab-ci.yml 文件；4. 接着执行 `make_*` 命令来编译 Yade；5. 最后执行 `--check` 和 `--test` 命令。6. 也可以使用 `make doc` 命令构建文档，但由于 Sphinx的界面经常变更，目前无法保证它能在所有 Linux 发行版上正常工作。

这些说明使用 `ccache` 和 `ld.gold` 来加速编译，如下所述。

Python 2 向后兼容

自 Python 2 支持于 2020 年初结束后，Yade 在 Python 2 生态系统上的编译自 6e097e95 主干版本起不再保证。不过，仍可使用上述 `PYTHON_VERSION` CMake 选项进行编译，但需要先决条件软件包的 Python 2 版本

[1] 如需查看最新构建日志的详细信息，请点击主进程 分支。

软件包列表可在 历史文档中相应的段落（Python 2 向后兼容）中找到。

Yade 的持续开发现在要求使用 Python 3 环境；如有需要，可参考一些关于将 *Python 2* 脚本转换为 *Python 3* 的说明。

2.5.4 加速编译

使用 ccache 编译

使用 ccache 缓存之前的编译结果，可以显著加快重新编译的速度：

```
cmake -DCMAKE_CXX_COMPILER_LAUNCHER=ccache [options as usual]
```

此外，可以使用 command `ccache --show-stats`（`ccache -s` （简写）查看当前 ccache 状态，或更改默认的缓存尺寸（存储于文件 `~/.ccache/ccache.conf`）。

使用 distcc 编译

在将编译任务分配到多个核心时（`make -jN`）， `N` 受可用核心和内存的限制。如果有可用的远程计算机，则可以使用更多核心，通过 distcc分发编译任务（有关配置从属引擎和主进程，请参阅 distcc 文档）：

```
export CC="distcc gcc"
export CXX="distcc g++"
cmake [options as usual]
make -jN
```

这两种工具可以结合使用，在上述 export 命令的基础上添加：

```
export CCACHE_PREFIX="distcc"
```

使用 CMake UNITY_BUILD 编译

此选项会将源文件连接成批次，每批包含多个 `*.cpp` ，以共用包含指令带来的开销（因为通常大多数源文件都包含相同的 Boost 头文件）。它能显著加快从头开始的完整编译。将以下内容添加到 CMake command 即可启用此选项； `CMAKE_UNITY_BUILD_BATCH_SIZE` 用于定义要连接在一起的文件最大数量（越大越好，主要限制可能是可用内存）：

```
-DCMAKE_UNITY_BUILD=ON -DCMAKE_UNITY_BUILD_BATCH_SIZE=18
```

此方法对增量重新编译没有帮助，甚至可能适得其反，因为每次修改单个文件时，都必须重新编译整个批次。如果预计某些特定文件需要进行增量编译，可以将其完整路径赋给 CMake 标志 `NO_UNITY` ，以便将其排除在 UNITY 构建之外（可以是单个文件，也可以是逗号分隔的列表）：

```
-DCMAKE_UNITY_BUILD=ON -DCMAKE_UNITY_BUILD_BATCH_SIZE=18 -DNO_UNITY=../trunk/pkg/dem/
→CohesiveFrictionalContactLaw.cpp
```

链接时间

将默认链接器从 `ld` 切换为 `ld.gold`可以缩短链接时间。两者都包含在同一个软件包 `binutils` 中（在 opensuse15 上，该软件包为 `binutils-gold`）。要执行此切换，请以 root 身份运行以下命令：

```
ld --version
update-alternatives --install "/usr/bin/ld" "ld" "/usr/bin/ld.gold" 20
update-alternatives --install "/usr/bin/ld" "ld" "/usr/bin/ld.bfd" 10
ld --version
```

要切换回去，请使用相反的优先级运行上述命令 `10` `20`。或者，也可以通过命令手动选择：`update-alternatives --config ld`。

注意：`ld.gold` 与编译器封装器 `mpicxx` 在某些发行版中不兼容，这会在 `cmake` 阶段表现为错误。目前，我们的 GitLab 构建不使用 `mpicxx` 。如果您想使用它，请禁用 `ld.gold`。在没有 `mpicxx` 编译器的情况下，也有报告称会出现与 CMake `MPI`相关的破坏；如果发生这种情况，唯一的解决方案是禁用 `ld.gold`或 `MPI` 功能。

2.5.5 云计算

可以利用云计算服务运行 Yade。Yade/亚马逊云科技的组合已被证实运行良好，具体如下。有关迁移到亚马逊的详细说明，请参见章节在亚马逊 *EC2* 上使用 *YADE* 进行云计算。

2.5.6 GPU 加速

FlowEngine 可通过 CHOLMOD 的图形处理器加速求解器进行加速。具体的硬件和软件要求详见使用图形处理器加速 *Yade* 的 *FlowEngine*。

2.5.7 特殊构建版本

可以采用特殊方式编译软件，以便查找其中的一些特定缺陷和问题：内存损坏、数据竞争、未定义行为等。

列出的消毒器都是运行时检测器。只有在代码的特定部分被执行时，它们才能发现该部分代码中的问题。如果你编写了新的 C++ 类（本构定律、引擎等），请尝试使用经过检测的软件下载并运行你的 Python 脚本，以检查代码中是否存在问题。

地址消毒器

地址消毒器 是一种内存错误检测器，可帮助查找堆损坏、越界错误以及许多其他可能导致崩溃甚至产生错误结果的内存错误。

要使用这种类型的消毒器编译 Yade，请使用 ENABLE_ASAN 选项：

```
cmake -DENABLE_ASAN=1
```

与常规构建相比，编译时间、构建期间的内存消耗以及构建文件的尺寸都要高得多。编译期间请监控内存和磁盘使用量，以防出现内存不足问题。

要在特定发行版中找到合适的 libasan 库，请使用 `locate` 或 `find /usr -iname"libasan*so"` command。然后，将 Yade 可执行文件与该 libasan 库一起启动，例如：

```
LD_PRELOAD=/some/path/to/libasan.so yade
```

默认情况下，asan 构建中会启用泄漏检测器。目前，Yade 会产生大量泄漏警告。要屏蔽这些警告并专注于其他内存错误，可以使用 detect_leaks=0 选项。考虑到这一点，在 Debian 10 Buster 上使用 AddressSanitized-Yade 运行测试的完整 command 如下：

```
ASAN_OPTIONS=detect_leaks=0:verify_asan_link_order=false yade --test
```

如果添加了新的检查脚本，它会通过持续集成流水线中的地址消毒器自动运行。

2.5.8 Yubuntu

如果你使用的不是 Linux 系统，可以创建一个 Ubuntu live-usb ，存放在任意 USB 大容量存储设备上（最小尺寸为 10GB）。这样就能在不影响原有系统的情况下，以预装了 Yadedaily 的 Linux 系统启动计算机。有关这种替代方案的更多信息请参见此处 （请先阅读 README 文件）。

或者，也可以下载 Linux 虚拟机的图像，此处同样提供；它们应该能在安装了虚拟化软件的任何系统上运行（已在 VirtualBox 和 VMWare 上测试）。

2.6 致谢Yade

我们恳请 Yade 用户在科学出版物中引用这份完整文档，以此评估 Yade 对其领域的贡献。可使用以下参考文献：

- V. Šmilauer 等（2021年），Yade 文档第3版。Yade 项目。数字对象标识符：10.5281/zenodo.5705394 （http://yade-dem.org/doc/）

除了致谢开发人员的工作之外，追踪 Yade 的学术档案上的引文还有助于发现新的应用场景或吸引新的用户。

第 3 章

面向程序员的 Yade

3.1 程序员手册

3.1.1 构建系统

Yade 使用 CMake 这一跨平台开源构建系统来管理构建过程。它负责配置、编译和安装。CMake 使用简单且独立于平台和编译器的配置文件来控制软件编译过程。CMake 会生成原生 Makefile 文件和工作区，可在您选择的编译器环境中使用。

构建

下面展示了 Yade源代码树的结构。我们将每个顶层组件称为模块 （不包括 `doc`、 `examples` 和 `scripts` ，因为它们不参与构建过程）。为简洁起见，省略了模块 中的一些子目录；更多信息请参阅其中的 `README.rst` 文件：

```
cMake/            ## cmake files used to detect compilation requirements
core/             ## core simulation building blocks
data/             ## data files used by yade,  packaged separately
doc/              ## this documentation
examples/         ## examples directory
gui/              ## user interfaces
   qt5/              ## same,  but for qt5
lib/              ## support libraries,  not specific to simulations
preprocessing/    ## files associated with creation or generation of the simulation
   dem/              ## creating a DEM simulation
   potential/        ## creating a PotentialBlocks or PotentialParticles simulation
   README.rst        ## more information about this directory
pkg/              ## simulation-specific files
   common/           ## generally useful classes
   dem/              ## classes for Discrete Element Method
   README.rst        ## more information about this directory
postprocessing/   ## files associated with extracting results for postprocessing
   dem/              ## general data extraction from DEM,  no particular data target
   image/            ## creating images from simulation
   vtk/              ## extracting data for VTK
   README.rst        ## more information about this directory
py/               ## python modules
scripts/          ## helper scripts including packaging and checks-and-tests
```

头文件安装

CMAKE 使用原始源代码布局，建议使用 `#include <module/Class.hpp>` 包含样式，而不是 `#include "Class.hpp"`，即使它们位于同一目录中。下表给出了一些示例：

原始头文件位置	包含为
core/Scene.hpp	#include <core/Scene.hpp>
lib/base/Logging.hpp	#include <lib/base/Logging.hpp>
lib/serialization/Serializable.hpp	#include <lib/serialization/Serializable.hpp>
pkg/dem/SpherePack.hpp	#include <pkg/dem/SpherePack.hpp>

自动编译

在 `pkg/` 目录中，情况有所不同。为了最大限度地方便向 Yade 添加模块，CMake 会自动递归扫描所有 `*.cpp` 文件，并将其纳入编译。
要启用或禁用某些组件，请使用 cmake 标志 `ENABLE_FEATURE`，这些标志列于：

1. 编译说明.2.
CMakeLists.txt.

启用某个组件后，CMake 会向编译器传递一个额外的 `#define` 标志 `YADE_FEATURE`。然后，在代码中，包含该 `FEATURE` 功能的 `.cpp` 和 `.hpp` 文件都应在开头添加一个 `#ifdef YADE_FEATURE` 保护。

链接

模块之间可能存在的依赖顺序如下：

module	生成的共享库	依赖项
lib	libyade-support.so	可以依赖外部库，但**不能**依赖任何其他 Yade 的组成部分。
core	libcore.so	yade-support；可以依赖外部库。
pkg	libplugins.so	~~core，yade-support~~
gui	libQtGUI.so, libPythonUI.so	lib, core, pkg
py	（多个文件）	lib, core, pkg, 外部

3.1.2 开发工具

集成开发环境和其他工具

常用的集成开发环境之一是 KDevelop。我们推荐使用这款软件浏览源代码、编译和调试。其他有用的调试和性能分析工具包括 Valgrind 和 KCachegrind。维基中有一系列专门介绍这些工具的 Wiki 页面，位于 开发 章节。

托管与版本管理

Yade 项目由以下平台热心托管：Launchpad 和 GitLab：

- GitLab 上的源代码
- 在 GitLab 上进行问题和缺陷跟踪

- 在 Launchpad 上下载发行版
- yade-dev 邮件列表（位于 Launchpad）： yade-dev@lists.launchpad.net
- yade-users 邮件列表（位于 Launchpad）： yade-users@lists.launchpad.net
- Launchpad 上的问题与解答

所使用的版本 的软件是 GIT，其简短教程可参见 *GitLab 上的 Yade*。GIT is 一个分布式版本控制系统。它以软件包形式提供，适用于所有主流 Linux 发行版。

该源代码会定期导入 Launchpad，以构建 PPA 软件包。代码仓库可通过 HTTP 浏览。

构建机器人

由UMS Gricad 托管的构建机器人通过GitLab 流水线机制跟踪源代码的变更。每当通过 Git 将源代码变更提交到主开发分支，或提交合并请求（MR）时，“构建机器人”都会下载并编译新版本，然后开始一系列测试。

如果引入了编译错误，系统会通知 yade-dev 邮件列表和提交者，从而帮助快速修复问题。如果编译成功，构建机器人便会运行单元回归测试和“检查测试”（见下文），并报告结果。如果所有测试都通过，则会生成新版本的文档，并以HTML和PDF格式上传到网站。因此，这两个链接始终指向最近一次成功构建的文档（即你现在正在阅读的文档），提交与文档更新之间的延迟非常短（几分钟）。构建机器人的活动和日志可在线浏览。

点击绿色的 “Passed”按钮，即可直接查看相应构建的输出；然后点击右侧“作业产物”中的“浏览”。

3.1.3 调试

Yade 调试有两种可用工具：

1. 使用调试构建，以便堆栈跟踪提供有关潜在崩溃的完整信息。可通过以下两种方式实现：

   a) 使用 CMake 选项 `-DDEBUG=ON` 编译 Yade，

   b) 安装 `yade-dbgsym` Debian/Ubuntu 软件包（此选项将在此任务完成后提供）。

2. 使用日志记录框架，如下所述。

这些工具可以与其他软件配合使用。关于这些工具的详细讨论请见yade 维基。这些工具包括：KDevelop，Valgrind，Alleyoop，KCachegrind，DDD，GDB，Kompare，KDiff3，Meld。

注意： 在某些 Linux 系统上不会显示堆栈跟踪，而会显示消息 `ptrace: Operationnot permitted` 。要启用堆栈跟踪，请执行命令： `sudo echo 0 > /proc/sys/kernel/yama/ptrace_scope`。要禁用堆栈跟踪，请执行命令 `sudo echo 1 > /proc/sys/kernel/yama/ptrace_scope`。

提示： 调试时，请确保 /tmp 中有足够的可用空间。

日志记录

Yade 使用 boost::log 库来灵活设置日志记录级别并进行按类调试。另请参阅日志模块的描述。编译时必须提供 CMake 编译选项 `-DENABLE_LOGGER=ON` 。[1]

```
In [1]: import log

In [2]: yade.log.setLevel("_log.cpp",5)
<INFO> _log.cpp:101 void setLevel(std::__cxx11::string, int): filter log level for _log.cpp has been set to 5

In [3]: log.setLevel("NewtonIntegrator",4)
<INFO> _log.cpp:101 void setLevel(std::__cxx11::string, int): filter log level for NewtonIntegrator has been set to 4

In [4]: log.getUsedLevels()
Out[4]: {'Default': 3, 'NewtonIntegrator': 4, '_log.cpp': 5}

In [5]: yade.log.testAllLevels()
<NOFILTER> :54 void testAllLevels(): Test log level: LOG_0_NOFILTER, test int: 0
<FATAL ERROR> _log.cpp:55 void testAllLevels(): Test log level: LOG_1_FATAL, test int: 1
<ERROR> _log.cpp:56 void testAllLevels(): Test log level: LOG_2_ERROR, test int: 2
<WARNING> _log.cpp:57 void testAllLevels(): Test log level: LOG_3_WARN, test int: 3
<INFO> _log.cpp:58 void testAllLevels(): Test log level: LOG_4_INFO, test int: 4
<DEBUG> _log.cpp:59 void testAllLevels(): Test log level: LOG_5_DEBUG, test int: 5
<NOFILTER> :62 void testAllLevels(): Below 6 variables are printed at filter level TRACE, then macro TRACE; is used

In [6]:
```

图 *imgLogging* 展示了日志框架的使用示例。通常， `ClassName` 会代替截图中显示的 `_log.cpp` 出现。这是因为 `yade.log` 模块使用 `CREATE_CPP_-LOCAL_LOGGER` 宏，而不是常规的 `DECLARE_LOGGER` 和 `CREATE_LOGGER`，这些内容将在下文讨论。

注意：日志消息的默认格式为：

```
<severity level> ClassName:LineNumber FunctionName: Log Message
```

特殊宏 `LOG_NOFILTER` 会被打印出来，但不会显示 `ClassName` ，因为它缺少一个。

可以通过*readConfigFile* 和*saveConfigFile*保存和加载配置文件。 *defaultConfigFileName*如果存在，会在启动时读取。通过命令行提供的过滤级别设置 `-f` 将覆盖配置文件中的设置。

日志级别

已支持以下调试级别：

[1] 没有 `-DENABLE_LOGGER=ON` CMake 选项时，/lib/base/Logging.hpp 中的调试宏会使用常规 `std::cerr` 进行输出，按类日志记录和日志级别将无法正常工作。

表格 1：Yade 日志详细程度级别。

宏名称	过滤器名称	选项	说明
LOG_NOFILTER	log.NOFILTER	-f0	只会打印未经筛选的消息。 日志 -_NOFILTER 宏仅供开发人员使用，因此基本上 -f0 表示不会打印任何内容。此日志级别 除非需要非常安静的模式，否则没有实用价值。
LOG_FATAL	log.FATAL	-f1	只会打印严重错误。 即使向 Yade 抛出异常，Python 接口也无法从这种情况中恢复。这 通常随后会导致 Yade 退出到 Shell。
LOG_ERROR	log.ERROR	-f2	也会打印无需向 Yade Python 接口抛出异常的错误。 计算将继续，但 结果很可能全都不对。
LOG_WARN	log.WARN	-f3	还会打印有关可恢复问题的警告， 这些问题应当通知你（例如，配置文件中的值无效， 因此 Yade 回退到了默认值）。
LOG_INFO	log.INFO	-f4	还会打印全部信息性消息（例如，某些内容 已加载、某些操作已调用，等等）。
LOG_DEBUG	log.DEBUG	-f5	还会打印调试消息。yade 开发人员会将其 放在各处，而 yade 用户会在 每个 类基础 中启用它们，以提供一些额外的调试信息。
LOG_TRACE	log.TRACE	-f6	跟踪消息会记录关于 yade 行为的所有可能 细节。

Yade 的默认日志级别为 `yade.log.WARN`，与调用 `yade -f3` 相同。

设置过滤级别

警告： 作为参数传递给 `LOG_*` 宏的消息（例如 `a << b << " message."`），仅当消息通过过滤级别筛选时才会使用。**请勿使用此类消息执行关键任务计算**。

过滤级别有两种设置：未设置特定的 `ClassName`（或 `"filename.cpp"`）筛选器时使用的 `Default` 级别，以及为特定 `ClassName`（或 `"filename.cpp"`）设置的过滤级别。可通过以下方式设置：

1. 使用 `yade -fN` command 启动 yade 时， `N` 用于设置 `Default` 过滤级别。默认值为 `yade.log.WARN` (3)。
2. 要在运行时更改 `Default` 过滤级别，请调用命令 `log.setLevel("Default",value)`或 `log.setDefaultLogLevel(value)`:

```
Yade [1]: import log

Yade [2]: log.setLevel("Default",log.WARN)

Yade [3]: log.setLevel("Default",3)

Yade [4]: log.setDefaultLogLevel(log.WARN)

Yade [5]: log.setDefaultLogLevel(3)
```

3. 要更改 `SomeClass` 的过滤级别，请调用命令：

```
Yade [6]: import log
```

（续见下页）

（接上页）

```
Yade [7]: log.setLevel("NewtonIntegrator",log.TRACE)

Yade [8]: log.setLevel("NewtonIntegrator",6)
```

4. 要更改 "filename.cpp" 的过滤级别，请使用创建它时指定的名称。例如，调整 "_log.cpp" 的过滤日志级别可能如下所示：

```
Yade [9]: import log

Yade [10]: log.getUsedLevels()
Out[10]: {}

Yade [11]: log.setLevel("_log.cpp",log.WARN)

Yade [12]: log.getUsedLevels()
Out[12]: {}

Yade [13]: log.getAllLevels()["_log.cpp"]
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
----> 1 log.getAllLevels()["_log.cpp"]

KeyError: '_log.cpp'
```

调试宏

若要启用特定类的调试功能，应将 DECLARE_LOGGER; 宏放在头文件的类定义中，以便为该类创建单独的命名日志记录器。随后必须在类实现 CREATE_LOGGER(ClassName);文件中使用 .cpp 宏来创建静态变量。有时需要在类外使用日志记录器，可以在 .cpp 文件中创建此类命名日志记录器。按照约定，其名称应与文件名对应，并使用宏 CREATE_CPP_LOCAL_LOGGER("filename.cpp"); 。极少数情况下，需要在 .hpp 文件中、类外部进行日志记录（此时无法使用局部类命名日志记录器），那么可使用 LOG_NOFILTER(…) 宏，因为它是唯一一个不依赖命名日志记录器也能正常工作的宏。如有需要，可以改进此方案，详情请参阅Logging.cpp 。

上文截面中列出的全部调试宏（LOG_TRACE、 LOG_DEBUG、 LOG_INFO、 LOG_WARN、 LOG_ERROR、 LOG_FATAL、 LOG_NOFILTER）都支持在括号内使用 std::ostream 语法，例如 LOG_TRACE(a <<b << " text")。 LOG_NOFILTER 比较特殊，因为无论调试级别如何，它始终都会输出，因此应仅用于开发分支。

此外，还有七个用于输出变量的宏，适用于 LOG_TRACE 级别： TRVAR1、 TRVAR2、TRVAR3、 TRVAR4、 TRVAR5、 TRVAR6 和 TRVARn。这些宏会输出变量，例如： TRVAR3(testInt,testStr,testReal); 或 TRVARn((testInt)(testStr)(testReal))。用法示例请参见 函数 testAllLevels 。

宏 TRACE; 会以 TRACE 日志过滤级别输出一条 "Been here" 消息，可用于快速调试。

实用调试宏

LOG_TIMED_* 宏族：

有时，在**非常快速**或**多线程**循环中调试变量会很有用。在这种情况下，可以：

1. 避免过快地向控制台刷屏，并为打印消息设置一定的超时时间，最好以秒或毫秒为单位指定。

2. 确保每个独立线程都有机会打印消息，且其消息不会与其他线程的消息交错。

要使用上述功能，必须在 `#include <lib/base/LoggingUtils.hpp>` 文件中 `.cpp` 提供 `LOG_TIMED_*` 和 `TIMED_TRVAR*` 宏系列。用法示例见函数 testTimedLevels。

为满足第一项要求，全部 `LOG_TIMED_*` 宏都接受**两个参数**，其中第一个参数是等待超时时间，采用标准 C++14 / C++20 时间单位，用法示例如下： `LOG_TIMED_INFO( 2s "test int: " << testInt++);` 每 2 秒打印一次。但只接受秒和毫秒（如有必要，可以更改这一点）。

为满足第二项要求，使用一个线程_局部静态计时器变量。这样，并行循环中的每个线程都可以每隔 `500ms` 或 `10s` 例如在这个并行循环中打印一条消息。上次向控制台打印的时间会按线程分别存储，并会添加一个额外的代码块来检查时间。这意味着，与典型的 `LOG_*` 相比，会进行更多一些的检查；后者只执行整数比较来检查过滤级别。因此，建议仅在进行密集调试时使用。调试完成后，最好将其移除。

注意： `*_TRACE` 系列宏会在发布构建期间由编译器移除，因为默认的 `-DMAX_LOG_LEVEL` 为 5。因此使用这些宏非常安全；但若要在本地使用它们，请确保编译 Yade 时启用 `cmake -DMAX_LOG_LEVEL=6` 选项。

`LOG_ONCE_*` 系列宏：

类似地， `LOG_ONCE_*` 和 `ONCE_TRVAR*` 系列宏也在文件Log-gingUtils.hpp中提供。随后，消息只会打印一次。

所有调试宏汇总如下表：

表格 2：Yade 调试宏。

宏名称	说明
DECLARE_LOGGER;	在类定义中声明日志记录器变量的宏 .hpp 文件中声明类定义内的日志记录器变量。
CREATE_LOGGER(ClassName);	创建静态日志记录器变量（包含 name "ClassName"）位于 . 中的类实现内。cpp 文件。
TEMPLATE_CREATE_-LOGGER(ClassName<OtherClass>);	创建静态日志记录器变量（包含 name "ClassName<OtherClass>"）位于类的实现中——在 .cpp 文件中。用于模板化类。
CREATE_CPP_LOCAL_LOGGER("filename.cpp");	在任意范围之外创建静态日志记录器变量。类之外创建静态日志记录器变量（名称为 "filename.cpp"），位于 filename.cpp 文件中。
LOG_TRACE, LOG_TIMED_TRACE, LOG_ONCE_TRACE, LOG_DEBUG, LOG_TIMED_DEBUG, LOG_ONCE_DEBUG, LOG_INFO, LOG_TIMED_INFO, LOG_ONCE_INFO, LOG_WARN, LOG_TIMED_WARN, LOG_ONCE_WARN, LOG_ERROR, LOG_TIMED_ERROR, LOG_ONCE_ERROR, LOG_FATAL, LOG_TIMED_FATAL, LOG_ONCE_FATAL, LOG_NOFILTER, LOG_TIMED_NOFILTER, LOG_ONCE_NOFILTER	使用 std::ostream 语法打印消息，例如：LOG_TRACE(a << b << " text") LOG_TIMED_TRACE(5s , a << b << " text");，每 5 秒打印一次 LOG_TIMED_DEBUG(500ms , a);，每隔 500 毫秒 LOG_ONCE_TRACE(a << b << " text");，仅打印一次 LOG_ONCE_DEBUG(a);，仅打印一次
TRVAR1, TIMED_TRVAR1, ONCE_TRVAR1, TRVAR2, TIMED_TRVAR2, ONCE_TRVAR2, TRVAR3, TIMED_TRVAR3, ONCE_TRVAR3, TRVAR4, TIMED_TRVAR4, ONCE_TRVAR4, TRVAR5, TIMED_TRVAR5, ONCE_TRVAR5, TRVAR6, TIMED_TRVAR6, ONCE_TRVAR6, TRVARn, TIMED_TRVARn, ONCE_TRVARn	打印提供的变量，例如：TRVAR3(testInt,testStr,testReal); TRVARn((testInt)(testStr)(testReal)); TIMED_TRVAR3(10s , testInt , testStr , testReal); ONCE_TRVARn((testInt)(testStr)(testReal)); 参见文件 py/_log.cpp 以查看使用示例。
TRACE;	在 TRACE 日志筛选器处打印一条 "Been here" 消息级别。
LOG_TIMED_6, LOG_6_TRACE, LOG_ONCE_6, LOG_TIMED_5, LOG_5_DEBUG, LOG_ONCE_5, LOG_TIMED_4, LOG_4_INFO, LOG_ONCE_4, LOG_TIMED_3, LOG_3_WARN, LOG_ONCE_3, LOG_TIMED_2, LOG_2_ERROR, LOG_ONCE_2, LOG_TIMED_1, LOG_1_FATAL, LOG_ONCE_1, LOG_TIMED_0, LOG_0_NOFILTER, LOG_ONCE_0, LOG_TIMED_6_TRACE, LOG_6, LOG_ONCE_6_TRACE, LOG_TIMED_5_DEBUG, LOG_5, LOG_ONCE_5_DEBUG, LOG_TIMED_4_INFO, LOG_4, LOG_ONCE_4_INFO, LOG_TIMED_3_WARN, LOG_3, LOG_ONCE_3_WARN, LOG_TIMED_2_ERROR, LOG_2, LOG_ONCE_2_ERROR, LOG_TIMED_1_FATAL, LOG_1, LOG_ONCE_1_FATAL, LOG_TIMED_0_NOFILTER, LOG_0 LOG_ONCE_0_NOFILTER,	便于在编辑器中使用的额外宏别名支持制表符补全。它们的名称中包含一个过滤级别数字。

最大日志级别

使用 boost::log 进行日志过滤，意味着每次调用 `LOG_*` 宏时，都必须执行一次整数比较，以确定消息是否通过当前过滤级别。对于生产环境使用，计算应尽可能快，而这种过滤方式并非最优，因为这些宏不会被优化掉，因为只需简单调用 `log.setLevel("Default",log.TRACE)` 或 `log.setLevel("Default",6)`即可重新启用它们。解决方法是使用 CMake 编译选项 `MAX_LOG_LEVEL=4` （或 3），该选项会在编译期间移除级别高于指定级别的宏。代码运行速度会略快，而 command `log.setLevel("Default",6)` 只会打印一条警告，说明当前构建不可能达到如此高的日志级别（可通过调用 *log.getMaxLevel()* 进行检查）。

注意： 在将日志记录引入 Yade 时，速度提升幅度非常小，以至于无法通过 `yade -f0 --stdperformance` command 测量。因此，引入此选项 `MAX_LOG_LEVEL` 仅仅是出于原则考虑。

这种方法的优点是，yade 可以在非调试构建中编译，同时仍可使用日志过滤框架。

3.1.4 回归测试

Yade 包含两类回归测试：一类是单元测试，另一类用于测试更复杂的模拟。虽然这两类都可以视为回归测试，但我们的惯例是将前一类简称为“回归测试”，而后一类称为“检查测试”。在 Yade 启动时传入“test”或“checkall”选项，即可运行这两组测试。

```
yade --test
yade --checkall
```

`yade --checkall` 是一次完整检查。若要跳过耗时超过 30 秒的检查，可以使用此命令

```
yade --check
```

单元回归测试

单元向量 回归测试 用于在明确定义的条件下测试各个函子和引擎的输出。它们定义在文件夹 py/tests/中。单元测试旨在确保代码开发过程中最重要的类始终保持正确。由于它们逐个测试类，因此无法检测典型模拟中不同引擎相互作用所引发的问题。这就是引入检查测试的原因。

要添加新的测试，必须执行以下步骤：

1. 放置一个新文件，例如 py/tests/dummyTest.py。2. 将文件名（例如 `dummyTest` ）添加到 py/tests/__init__.py 文件中。3. 如有必要，修改 `import` 和 `allModules` 直线（在 py/tests/__init__.py）中的内容。4. 根据 Python unittest 文档 中的说明，使用类似 `self.assertTrue(…)`、`self.assertFalse(…)` 或 `self.assertRaises(…,…)` 的命令来报告可能出现的错误。

注意： 测试中使用的所有变量都必须存储在类中（使用 `self.` 访问器），并且所有准备工作都必须在函数 `setUp()` 内完成。

检查测试

检查测试 （另请参阅 README）会按照 此处所述，对不同版本 Yade 的仿真结果进行比较。与回归测试相比，它们会模拟更复杂的情形以及不同引擎的组合，而且通常没有数学证明（不过最新版本并无此限制）。它们会将 N 版本中获得的值与先前版本中获得的值或其他“预期”结果进行比较。参考值必须硬编码在脚本本身或随脚本提供的数据文件中。检查测试基于常规 yade 脚本，因此用户可以轻松地将自己的脚本提交到 trunk，以便在其他开发者提交代码后进行自动化测试。

检查失败时，脚本应通过 Python 命令 `raiseYadeCheckError(messageString)` 返回错误消息，说明出了什么问题。如果脚本本身因某种原因失败，无法生成输出，日志中将只包含“scriptName 破坏”。如果脚本定义了已获取数据与预期数据之间的差异，则应输出一些有关问题的有用信息。此情况发生后，自动测试将停止运行并显示错误消息。

一个虚拟检查测试示例 scripts/checks-and-tests/checks/checkTestDummy.py 展示了一个最简单的空测试。一个功能稍多一些的检查测试示例见 scripts/checks-and-tests/checks/checkTestTriax.py。它展示了结果比较、输出，以及如何使用 `checksPath`定义数据文件的路径。鼓励用户将自己的脚本添加到 scripts/checks-and-tests/checks/ 文件夹中。欢迎在 问答 中讨论某些特定检查测试的设计。请注意，必须先 重新编译 ，新添加的脚本才能通过以下方式启动： `yade --check` （或者直接在“lib”子文件夹中进行更改）。检查测试的运行时间不应超过几秒。如果你的常用脚本需要更长时间，请尝试减少元素数量或步数。

如需添加新的检查，必须执行以下步骤：

1. 放置一个新文件，例如 scripts/checks-and-tests/checks/checkTestDummy.py；2. 在新脚本中使用 `checksPath` ，当需要加载某个数据文件时，例如 scripts/checks-and-tests/checks/data/100spheres ；3. 发生错误时，使用以下 command 引发异常 `raise YadeCheckError(messageString)`

GUI 测试

为了添加新的 GUI 测试，需要将一个文件添加到 scripts/checks-and-tests/gui 目录中。文件必须按照以下约定命名： `testGuiName.py` 并放置适当的 test `Name`（该 `testGui.sh` 脚本 会搜索 与此模式匹配的文件）。 scripts/checks-and-tests/gui/testGuiBilliard.py 可以作为样板示例。代码中重要的“额外”部分（例如取自 示例 目录）如下：

1. `from testGuiHelper import TestGUIHelper`2. `scr = TestGUIHelper("Billiard")`，请确保将选定的 test `Name` 放在 `Billiard`的位置。3. 设置合理的 `guiIterPeriod` 值，使 test 在 30 秒内完成。4. 在 `O.engines` 内部，循环末尾必须调用 `PyRunner(iterPeriod=guiIterPeriod, command='scr.screenshotEngine()')`。5. 脚本中的最后一条 command 应为 `O.run(guiIterPeriod * scr.getTestNum() + 1)` ，以启动 test 进程。6. 确保将参考屏幕截图推送到 yade-data 仓库 （有关 `./data` 目录，请参阅 *GitLab 上的 Yade*）。也可以通过在 GitLab 流水线中点击右侧窗格里“Browse”按钮旁边的“Download”按钮，从构建产物中获取这些屏幕截图。

调整 testGui.sh 脚本开头的路径后，即可在本地运行这些测试。支持两种运行模式：

1. 通过 command 在本地桌面启动： `scripts/checks-and-tests/gui/testGui.sh`。在这种情况下，屏幕截图将与测试期间使用的截图不同。

2. 或通过 command 在虚拟 xserver 中启动： `xvfb-run -a -s "-screen 0 1600x1200x24" scripts/checks-and-tests/gui/testGui.sh`，这样生成的屏幕截图将与 test 中使用的截图相似，但字体尺寸可能仍有差异。在这种情况下，建议使用从 GitLab 流水线构建产物中下载的参考截图（参见上文第 6 点）。

注意不要使用 test 中刚体的随机颜色。此外，脚本 `testGuiName.py`中不应打开三维视图或检查器视图等窗口，因为这些窗口会在 test 期间由 `TestGUIHelper` 类打开。

注意： 由于必须在虚拟 xserver 中启动 YADE，因此无法通过诸如 `yade --test` 这样的调用来运行 GUI 测试。

3.1.5 约定

应遵循以下编码规则；文档另行处理。

- 通用

  – C++ 源文件使用 `.hpp` 和 `.cpp` 扩展名（分别用于头文件和实现文件）。少数情况下，纯模板代码会使用 `.ipp`。– 所有头文件都应使用 `#pragma once` 防止重复包含的保护机制。– 不要在头文件中输入 `using namespace …`，这可能导致命名空间污染，从而引发难以察觉的缺陷。– 避免在 `.hpp` 文件中使用 `using std::something`。在.cpp 文件中可以随意使用。但请记住，这种做法仍有常见的问题：可能会使用错误的类型或函数，而不是预期的那个。不过，由于它仅限于单个 `.cpp` 文件，因此更容易调试，而且这种便利性或许超过了相关风险。– 使用制表符缩进。虽然这在 `C++`中仅有视觉效果，但在 Python 中却有语义意义；不慎混用制表符和空格可能导致语法错误。

- 大小写风格

  –类型名称应始终首字母大写。组合类名和类型名应使用驼峰式大小写（`GlobalEngine`）。仅在特殊情况下（例如函子名称）使用下划线。– 类的数据成员和方法名称不得首字母大写；组合名称应使用小驼峰式大小写（`glutSlices`）。Python 模块中的函数也遵循相同规则。– 预处理器宏应使用大写字母，并以下划线分隔；核心以外使用的宏（例外情况除外）采用以下形式 `YADE_*`，例如 *YADE_CLASS_BASE_DOC_* 宏族*。

- 编程样式

  – 如果不会对性能造成显著影响，就应采取防御性措施。应大量使用断言：断言不会影响性能（在优化构建中），而且能让错误状况更容易被发现。– 在调试构建中需要进行类型检查、而在优化构建中需要快速转换时，使用 `YADE_CAST` 和 `YADE_PTR_CAST`。– 在默认构造函数中初始化所有类变量。这样可以避免出现随机发生且难以修复的缺陷。用非数初始化有助于发现原本未初始化的变量。（这由 *YADE_CLASS_BASE_DOC_* 宏系列* （用于用户类的宏）负责处理）

使用 clang-format

文件 .clang-format 包含配置，可确保始终生成相同的结果。它适用于`clang-format --version` >= 10。其目的是避免提交仅更改格式的变更。可以对文件或目录运行脚本scripts/clang-formatter.sh ，以重新设置格式。通常可以将其集成到编辑器中，参见 clang-format 文档（但对于 Vim，`py3f` 必须使用 command），在 KDevelop 中则可将其添加为自定义格式化工具。

脚本 scripts/python-formatter.sh 会按照我们的编码约定对 Python 脚本进行格式化。提交对 Python 脚本的更改前，应运行该脚本。

如需更多帮助，请参阅：

1. clang-format 文档2.
   yapf3 文档

有时需要禁用文件中某个小节的格式化。为此，请在该小节周围添加保护标记：

1. 在 C++ 中使用：

```
// clang-format off
……
// clang-format on
```

2. 在 Python 中使用：

```
# yapf: disable
……
# yapf: enable
```

类命名

尽管由于历史原因，命名方案并不完全一致，但仍应遵守以下规则，尤其是在添加新类时。

全局引擎**和** 部分引擎 全局引擎的命名应体现其执行某种操作的特征（例如 *ForceResetter*、*InsertionSortCollider*、*Recorder*）；如果不适合这样命名，则在表示特征的名称后添加 `Engine` （例如 *GravityEngine* ）。部分引擎没有区别于全局引擎的特殊命名约定。

分派器 所有分派器的名称均以 `Dispatcher`结尾。名称由其创建的类型组成；如果不创建任何对象，则由其主要特征组成。目前定义了以下分派器：[2]：

分派器	元数	分派类型	已创建 type	函子类型	函子前置 fix
BoundDis- 修补者	1	形状	*Bound*	*BoundFunc-tor*	Bo1
IGeomDis- 补丁器	2（对称）	2 ×形状	*IGeom*	*IGeomFunc-tor*	Ig2
*IPhys*物理补丁器	2（对称）	2 × *Mate-rial*	*IPhys*	*IPhys*函数 *tor*	Ip2
LawDispatcher	2（渐近-ric）	*IGeom IPhys*	（无）	*LawFunctor*	Law2

每个分派器各自对应的抽象函子是*BoundFunctor*、*IGeomFunctor*、*IPhys-Functor* 和 *LawFunctor*。

[2] 暂不考虑 OpenGL 分派器；未来它们可能会被常规虚函数替代。

函子 函子名称由 3 部分组成，以下划线分隔。

1. 前缀，由函子类型的缩写和元数组成（见上表） 2. 进入分发器逻辑的类型（一元函子为 1，二元函子为 2） 3. 创建实例的函子的返回类型；不创建实例的函子的简单特征。

举几个例子：

- *Bo1_球体_Aabb*是一个 边界函子 ，它会在 球体上调用，并创建一个 *Aabb*实例。
- *Ig2_面片_球体_ScGeom* 是一个二元函子，在 面片 和 球体上调用，并创建一个 *ScGeom*实例。

•*Law2_ScGeom_CpmPhys_Cpm* 是一个二元函子（*LawFunctor*），其调用对象的类型为 *ScGeom*（几何）和 *CpmPhys*。

文档

妥善编写代码文档是持续开发中最重要的环节之一。

再读一遍。

Yade 等科研软件中的大部分代码不仅会被开发者使用，也会被他们阅读，甚至会被普通用户阅读。因此，添加新类时，务必在文档中说明以下内容：

- 用途
- 功能细节（除非显而易见，例如算法和内部逻辑）
- 局限性（设计方面、实现方面）、缺陷
- 如果使用了非平凡的已发表算法，请提供参考文献（见下文）
- 对其他相关类的引用
- 指向与此特定功能相关的缺陷、蓝图、维基或邮件列表的超链接。

在可行的情况下，你还应该

- 更新所有受影响的其他文档
- 提供一个简单的 Python 脚本，演示 `scripts/test` 中的新功能。

Sphinx 文档

大多数 c++ 类都用 Python 封装，这为自省和交互式文档提供了良好支持（尝试在 `Material?` IPython 提示符中输入；或输入 `help(CpmState)`）。
文档的语法是 ReST （reStructuredText，参见 reStructuredText Primer）。c++ 和 Python 代码也是如此。

- c 类的文档++ 作为第 3 个参数提供给 *YADE_类_- 基础_文档_* 宏系列将在下文介绍。

- Python 类/函数使用常规的 Python 文档字符串进行文档说明。除了说明功能外，还应记录所有参数的含义和类型。简短的代码片段可能会非常有帮助。示例请参见 *utils* 模块。

注意： 在 C++ 字符串字面量 中编写文档字符串时++。按照约定，使用 `R"""(raw text)"""`。例如，请参见 此处 和 此处。

注意： 请记住，在 C++ 文档字符串中，可以调用 Python 命令；在编译文档时，Yade 会执行这些命令。例如，比较此 源文档字符串与 最终效果。

除了标准的 ReST 语法外，yade 还提供了几个简写宏：

`:yref:` 创建指向所引用术语的超链接，例如：

```
:yref:`CpmMat`
```

变为*CpmMat*；链接名称和目标可以不同：

```
:yref:`Material used in the CPM model<CpmMat>`
```

从而得到*CPM 模型中使用的材料*。

`:ysrc:` 创建指向源代码树中文件的超链接（指向其在代码仓库中的最新版本），例如core/Cell.hpp。与 `:yref:`一样，也可以使用替代文本

```
:ysrc:`Link text<target/file>`
```

像 这样。这不能用于链接到指定行号，因为文件一旦更改，行号就会过时。若要链接到某个行号，请使用下文所述的 `:ysrccommit:` 。

`:ysrccommit:` 会在指定的提交哈希处创建指向源代码树中文件的超链接。例如，可以在链接末尾使用 `#L121` 来链接到相应行号。用法与`:ysrc:` 相同，但必须在开头提供提交哈希：

```
:ysrccommit:`Link text<commithash/target/file#Lnumber>`

:ysrccommit:`default engines<775ae7436/py/__init__.py.in#L112>`
```

变为 默认引擎。

链接到 `inheritanceGraph*` 若要链接到某个基类的继承图，会创建一个全局锚点，其名称为 `inheritanceGraph*` 并置于类名前面，例如`:ref:`Shape<inheritanceGraphShape>`` 会生成指向*Shape 的继承图*的链接。

`|ycomp|` 用于属性描述，表示该属性无需由用户提供，而是自动计算得出； `|ycomp|` 会展开为 *(自动计算)*。

`|yupdate|` 标记周期性更新的属性，这些属性是前述属性的子集。 `|yupdate|`展开为 （自动更新）。

`$...$` 用于界定行内数学表达式；它们将被替换为：

```
:math:`...`
```

并通过 LaTeX 渲染。若要输入单个美元符号，请使用反斜杠 `\$`进行转义。

如 Sphinx 中面向 HTML 输出的 math 支持 中所述，可以插入显示数学公式（独立方程）。

提醒一下，在标准 ReST 语法中，引用如下：

`:ref:` 是标准的 reStructuredText 引用，用于指向文本其他位置的锚点。例如，锚点 `.. _NumericalDamping:` 放置在 formulation.rst 中，则可通过`:ref:`NumericalDamping`` 在源代码中链接到它。

`.. _anchor-name:` 用于在文本中放置锚点，以便在文本的其他位置引用。锚点名称中禁止使用符号 `_` ，因为它有特殊含义： `_anchor` 用于指定锚点，而 `anchor_` 用于链接到该锚点，见下文。

anchor-name_ 用于在同一文件中链接到锚点。与适用于不同文件的写法相比，它是更简短的形式：:ref:。锚点的用法示例请见 imgQtGui 此处和此处。

注意： command :scale: NN % (带百分比)与 .html + .pdf 输出配合使用效果不佳，最好指定 :width: NN cm。这样在 .html 和 .pdf 中的尺寸就相同了。例如，请看 此处，它会变成 这张图片。但请注意，.pdf 中的图片最大宽度为 16.2 cm。

书目参考文献

与任何科学文档一样，引用出版物非常重要。要引用一篇文章，首先以 BibTeX 格式将其添加到文件 doc/references.bib 或 doc/yade-*.bib中，具体取决于该参考文献是否使用了 Yade（后者）。（前者则表示未使用 Yade。）请遵循以下约定：

1. 条目应保持以下形式： Author2008 （Author 是第一作者）， Author2008b 等（如果同一作者有多篇文章）；
2. 尽量填写 给定引文类型的必填字段；
3. 不要使用 \'{i} 用于重音符号的奇怪转义，因为它们无法用于 HTML 输出；请直接使用 UTF-8 编码。

I在文档字符串中，之后可以通过 [Author2008]_ 引用这篇 Author2008 文章；例如：

```
According to [Allen1989]_, the integration scheme …
```

将呈现为

> 根据 [Allen1989], 积分方案.....

类/函数单独文档

某些类++ 的文档可能很长或包含大量内容，不便于直接以字符串字面量的形式输入到++ 源代码本身中。Yade 提供了一种在 py/_-extraDocs.py 文件中单独编写文档的方法：该文件会在加载++ 插件后执行，并可直接设置任意对象的 __doc__ 属性，覆盖源代码中的文档字符串++。在这类（特殊）情况下：

1. 不过，仍应在代码中为类提供至少简短的描述++ code，以便只读代码的人了解其内容。

2. 添加说明：“此类的详细文档见 py/_extraDocs.py 文件”。
3. 按照以下方式在 py/_extraDocs.py 中添加文档：

```
module.YourClass.__doc__='''
This is the docstring for YourClass.Class,

 methods and functions can be documented this way.
.. note:: It can use any syntax features you like.'''
```

注意： Boost::python 会将函数签名嵌入文档字符串中（位于用户提供的文档字符串之前）。因此，在为函数创建单独的文档之前，请查看其 __doc__ 属性，并将第一行（以及其后的空行）复制到单独的文档字符串中。随后，第一行将用于创建函数签名（参数和返回值）。

内部 c++文档

doxygen 曾用于自动生成 c++ 代码。由于面向用户的类现在使用 Sphinx 定义，因此使用 doxygen 生成总体文档已无意义。不过，请注意使用常规注释为代码的内部部分编写完善的文档，包括公共和私有数据成员。

3.1.6 支持框架

除了 c++ 标准库（包括 STL）、Boost 和其他依赖项提供的框架外，Yade 还提供了自身的专用服务。

指针

共享指针

Yade 广泛使用共享指针 `shared_ptr`。[3] 虽然这可能会对性能造成一些影响，但它大大简化了内存管理，以及 Python 中 c++ 对象的所有权管理等。要从一个 `shared_ptr`获取原始指针，请使用其 `get()` 方法；如果对象只会短暂使用（例如在函数调用期间），而不会存储在任何地方，则应使用原始指针。

Python 为大多数 c++ Yade 类定义了轻量级封装器（适用于所有通过 *YADE_CLASS_-BASE_DOC_* 宏系列注册的类以及其他一些类），这些封装器可以由 `shared_ptr`构造；通过这种方式，Python 的引用计数与 `shared_ptr` 引用计数模型相结合，可防止 Python 对象指向已被销毁的 c++ 对象而导致崩溃。

类型转换

通常需要对指针进行类型转换；可以选择静态转换或动态转换。

- `dynamic_cast` (`dynamic_pointer_cast` 对于 `shared_ptr`) 会通过检查参数的运行时类型来确保转换有效性；如果转换无效，则返回 NULL。显然，这种检查会耗费时间。通过检查指针是否为 NULL，可以轻松捕获无效转换；即使省略了此类检查（例如 `assert`），崩溃后也能从堆栈跟踪（调试器输出）中轻松发现对 NULL 指针的解引用。此外， `shared_ptr` 会在调试构建中解引用指针前检查其是否非 NULL；如果检查失败，则会中止并显示“断言‘px!=0’失败。”
- `static_cast` 速度很快，但可能存在危险（`static_pointer_cast` 适用于 `shared_ptr`）。即使类型不允许进行强制转换（例如从 `State*`转换为 `Material*`），静态强制转换也会返回非 NULL 指针；这类转换会将垃圾数据解释为目标类的实例，极有可能导致无效内存访问（段错误，简称“崩溃”）。

为了兼顾速度与安全性，Yade 提供了 2 个宏：

`YADE_CAST` 在优化构建中展开为 `static_cast` ，在调试构建中展开为 `dynamic_cast` 。

`YADE_PTR_CAST` 在优化构建中展开为 `static_pointer_cast` ，在调试构建中展开为 `dynamic_pointer_cast`。

基本数值计算

Yade 中使用的浮点类型是 `Real`，默认 typedef 为 `double` （64 位，15 位小数位数）。[4]

[3] 要么使用 `boost::shared_ptr` ，要么使用 `tr1::shared_ptr` ，但始终通过 `using` 语句导入，以便可以使用未限定的 `shared_ptr` 。 [4] 详情请参阅 高精度文档 。

Yade 使用 Eigen 库进行计算。它提供了用于二维和三维向量、四元数以及 3x3 矩阵的类，这些类以数值类型作为模板参数；针对 `Real` 类型的特化版本使用“r”后缀进行 typedef 定义，此外，一些常用的整数类型使用“i”后缀：

- `Vector2r`, `Vector2i`
- `Vector3r`, `Vector3i`
- `Quaternionr`
- `Matrix3r`

Yade 还定义了一个名为 *Se3r* 的类，其中包含空间位置（`Vector3rSe3r::position`）和方向（`Quaternionr Se3r::orientation`）。由于二者经常搭配使用，因此将它们作为单个参数传递给函数会很方便。

Eigen 提供了完整而丰富的线性代数功能。一些代码还使用 [CGAL] 库进行计算几何。

在 Python 中，基本数值类型会经过封装，并从 `yade.minieigenHP` 模块导入；这些类型末尾的 `r` 类型限定符会被去掉，其余语法则类似。 `Se3r` 不会被封装，只会在需要时（这种情况很少见）自动转换为（`Vector3,Quaternion`）元组/列表。更多详情请参阅高精度章节。

```
# cross product
Yade [14]: Vector3(1,2,3).cross(Vector3(0,0,1))
Out[14]: Vector3(2,-1,0)

# construct quaternion from axis and angle
Yade [15]: Quaternion(Vector3(0,0,1),pi/2)
Out[15]: Quaternion((0,0,1),1.5707963267948966)
```

注意：四元数在内部以 4 个数值存储。不过，其通常面向用户的表示方式是旋转轴（归一化后的轴）以及绕该轴旋转的角度，在 Python 中输入/输出时也采用这种表示方式。可以通过 `[0]` ... `[3]` 元素访问方式（或`.W()`、 `.X()`、 `.Y()` 和 `.Z()` 方法）访问原始内部值，在 c++ 和 Python 中均可。

运行时类型识别（RTTI）

由于序列化和分派器需要扩展的类型和继承信息，而标准 RTTI 无法充分提供这些信息，因此每个 yade 类都派生自 `Factorable` ，并且必须使用宏来重写其虚函数，以提供这些扩展的 RTTI：

`YADE_CLASS_BASE_DOC(Foo,Bar Baz,"Docstring")` 会创建以下虚方法（通过 `REGISTER_CLASS_AND_BASE` 宏实现；该宏对用户不可见，不应直接使用）：

- `std::string getClassName()` 返回类名（`Foo`），类型为字符串。（虽然有 `typeid(instanceOrType).name()` 标准 c++ 构造，但返回的名称取决于编译器。）
- `unsigned getBaseClassNumber()` 返回基类的数量（本例中为 2）。
- `std::string getBaseClassName(unsigned i=0)` 返回第 i 个基类的名称（此处， `Bar` 当 i=0 和 `Baz` 当 i=1）。

警告： RTTI 依赖虚函数；要使虚函数正常工作，必须在实现文件（`.cpp`）中至少声明一个虚方法。否则，编译器不会为此类生成虚方法表（vtable），导致虚方法无法正常工作。

So可以从 py 中访问信息 thon:

```
Yade [16]: yade.system.childClasses('Shape')
Out[16]:
{'Box',
 'ChainedCylinder',
 'Clump',
 'Cylinder',
 'Facet',
 'GridConnection',
 'GridNode',
 'PFacet',
 'Sphere',
 'Tetra',
 'Wall'}

Yade [17]: Sphere().__class__.__name__              ## getClassName()
Out[17]: 'Sphere'
```

序列化

序列化用于将模拟保存到文件中，以便稍后恢复。此过程需要满足几个条件：

- 类知道自己有哪些属性（数据成员），以及这些属性的名称（以字符串形式表示）；
- 仅凭类的名称创建类实例；
- 知道特定共享库（插件）中定义了哪些类。

此功能由 3 个宏和 4 个可选方法提供；详情如下。

`Serializable::preLoad`, `Serializable::preSave`, `Serializable::postLoad`, `Serializable::postSave` 在序列化（保存）或反序列化（加载）之前准备属性，或在序列化或反序列化之后处理这些属性。

请参阅属性注册。

`YADE_CLASS_BASE_DOC_*` 在类声明中（即 `.hpp` 文件中的 `class Foo { /*… */};` 代码块）。请参阅属性注册。

枚举应保存和加载的类属性；将每个属性与其字面名称关联，以便用于检索该属性。请参阅 *YADE_CLASS_BASE_DOC_* 宏系列*。

Addi此外，还会在 Python 中为类编写文档，添加从 Python 访问属性的方法，并 d docu为每个属性编写文档。

`REGISTER_SERIALIZABLE` 在头文件中，但在之后的类声明块中。参见类工厂 .

将类的字面名称与用于创建其新实例的函数关联起来（`ClassFactory`）。

必须在 `namespace yade` 内声明。

`YADE_PLUGIN` 在实现 `.cpp` 文件中。参见插件注册。

声明在加载插件时（Yade 启动时）某个特定插件中声明了哪些类。

必须在 `namespace yade` 内声明。

属性注册

Yade 中的所有（可序列化）类型都是以下类型之一：

- 派生自 可序列化对象，这些类型通过重载 `Serializable::registerAttributes` 方法来提供如何序列化自身的信息；该方法会声明需要序列化的数据成员及其字面名称，并通过这些名称对其进行标识。随后，此方法会调用`registerAttributes` 其基类的 `Serializable` （直到到达其自身）；这样，派生类就能正确序列化其基类的数据。

  此功能隐藏在宏*YADE_类_基_文档_*宏族中，该宏用于类声明主体（头文件），接受基类和属性列表：

```
YADE_CLASS_BASE_DOC_ATTRS(ThisClass,BaseClass,"class documentation",((type1,attribute1,
→initValue1, "Documentation for attribute 1"))((type2,attribute2,initValue2,,
→"Documentation for attribute 2")));
```

  请注意，属性用双括号括起，而不是用逗号分隔。空属性列表可以简单地写作 `YADE_CLASS_BASE_DOC_ATTRS(ThisClass,BaseClass,"documentation",)` （末尾的逗号不可省略），也可以从宏名称和最后一个参数中省略 `ATTRS` 。

- 基本类型：字符串、各种数值类型、布尔值、 `Vector3r` 等。它们的“处理器”（序列化器和反序列化器）定义在 `lib/serialization` 中。
- 由任意可序列化对象组成的标准容器。
- 指向可序列化对象的共享指针。

Yade 在内部使用优秀的boost::serialization 库来进行数据序列化。

注意： `YADE_CLASS_BASE_DOC_ATTRS` 也会生成用于从 Python 访问属性的代码；这一点将在后文讨论。由于此宏兼具这两种用途，因此可序列化的属性始终可以从 Python 访问。

Yade 还提供了用于序列化前后回调的虚函数 *Serial-izable::preProcessAttributes* 和 *Serializable::postProcessAttributes*，它们接收一个`bool`deserializing 参数（`true` 反序列化时，`false` 序列化时）。它们在 *Serializable* 中的默认实现不执行任何操作，但通常用于：

- 将类中某些不可序列化的内部数据结构（例如多维数组、哈希表、指针数组）转换为可序列化的结构（预处理），并在反序列化后将这些不可序列化的结构填充回来（后处理）；例如，InteractionContainer 使用这些钩子，在序列化之前要求其具体实现将内容存储到统一存储中（`vector<shared_ptr<Interaction> >`），并在反序列化后从中恢复。

- 根据序列化的值预先计算未序列化的属性；例如，面片根据顶点坐标计算其（局部）边法线和边长。

类工厂

每个可序列化类都必须使用 `REGISTER_SERIALIZABLE`，它定义了一个用于通过`ClassFactory`创建该类的函数。 `ClassFactory` 能够根据类名（字符串）实例化类，这对于反序列化是必需的。

尽管主要由序列化框架在内部使用，程序员也可以使用 `shared_ptr<Factorable> f=ClassFactory::instance().createShared("ClassName");`实例化类，然后将返回的 `shared_ptr<Factorable>` 转换为所需类型。可序列化对象本身派生自 `Factorable`，也就是说，所有可序列化类型同时也是 Factorable 类型。

注意： 宏 `REGISTER_SERIALIZABLE` 和 `YADE_PLUGIN` 都必须在 yade 命名空间内声明。

插件注册

Yade 会在启动时加载包含其全部功能的动态库。必须让 ClassFactory 知道每个特定文件所提供的类。 YADE_PLUGIN 用于此目的；与 *YADE_-CLASS_BASE_DOC_* 宏系列不同，它必须放置在实现文件（.cpp）中，并位于 yade 命名空间内。它只需枚举该文件所提供的类：

```
YADE_PLUGIN((ClassFoo)(ClassBar));
```

注意： 即使只有一个类，也必须在类名外使用括号（预处理器的限制）： YADE_PLUGIN((classFoo));。如果此文件中没有类，则完全不要使用此宏。

在内部，此宏会创建函数 registerThisPluginClasses_ ，并将其特别声明为 __-attribute__((constructor)) （参见 GCC 函数属性）；此属性会使该函数在通过 dlopen 从 ClassFactory::load(...)加载插件时执行。它会将该文件中所有可生成的类注册到 类工厂 中。

注意： 不继承自 Factorable 的类（例如 Shop 或 SpherePack）不会使用YADE PLUGIN声明。_

以下是一个可序列化类的头文件示例：

```
namespace yade {
/*! Homogeneous gravity field; applies gravity×mass force on all bodies. */
class GravityEngine: public GlobalEngine{public:virtual void action();
// registering class and its base for the RTTI system
YADE_CLASS_BASE_DOC_ATTRS(GravityEngine,GlobalEngine,
// documentation visible from python and generated reference documentation
"Homogeneous gravity field; applies gravity×mass force on all bodies."<style 标识符='10'>,
// enumerating attributes here  include documentation                    <style 标识符
((Vector3r,gravity,Vector3r::ZERO,"acceleration, zero by default [kgms$^{2}$]"))
='13'>,);};// registration function for ClassFactoryREGISTER_SERIALIZABLE(GravityEngine);
} // namespace yade
```

下面是具体实现：

```
#include <pkg/common/GravityEngine.hpp>
#include <core/Scene.hpp>

namespace yade {
// registering the plugin
YADE_PLUGIN((GravityEngine));

void GravityEngine::action(){
        /* do the work here */
}
} // namespace yade
```

我们可以创建一个迷你模拟（只包含一个 GravityEngine）：

```
Yade [18]: O.engines=[GravityEngine(gravity=Vector3(0,0,-9.81))]

Yade [19]: O.save('abc.xml')
```

而 XML 保存格式如下：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive" version="17">
<scene class_id="0" tracking_level="0" version="1">
        <px class_id="1" tracking_level="1" version="0" object_id="_0">
                <Serializable class_id="2" tracking_level="1" version="0" object_id="_1"></
→Serializable>
                <dt>1.0000000000000002e-08</dt>
                <iter>0</iter>
                <subStepping>0</subStepping>
                <subStep>-1</subStep>
                <time>0.0000000000000000e+00</time>
                <speed>0.0000000000000000e+00</speed>
                <stopAtIter>0</stopAtIter>
                <stopAtTime>0.0000000000000000e+00</stopAtTime>
                <isPeriodic>0</isPeriodic>
                <trackEnergy>0</trackEnergy>
                <doSort>0</doSort>
                <runInternalConsistencyChecks>1</runInternalConsistencyChecks>
                <selectedBody>-1</selectedBody>
                <tags class_id="3" tracking_level="0" version="0">
                        <count>5</count>
                        <item_version>0</item_version>
                        <item>author=bchareyre~(bchareyre@HP-ZBook-15-G3)</item>
                        <item>isoTime=20220726T141500</item>
                        <item>id=20220726T141500p61547</item>
                        <item>d.id=20220726T141500p61547</item>
                        <item>id.d=20220726T141500p61547</item>
                </tags>
                <engines class_id="4" tracking_level="0" version="0">
                        <count>1</count>
                        <item_version>1</item_version>
                        <item class_id="5" tracking_level="0" version="1">
                                <px class_id="7" class_name="yade::GravityEngine" tracking_
→level="1" version="0" object_id="_2">
                                        <FieldApplier class_id="8" tracking_level="1" version=
→"0" object_id="_3">
                                                <GlobalEngine class_id="9" tracking_level="1"␣
→version="0" object_id="_4">
                                                        <Engine class_id="6" tracking_level="1
→" version="0" object_id="_5">
                                                                <Serializable object_id="_6"></
→Serializable>
                                                                <dead>0</dead>
                                                                <ompThreads>-1</ompThreads>
                                                                <label></label>
                                                        </Engine>
                                                </GlobalEngine>
                                        </FieldApplier>
                                        <gravity class_id="10" tracking_level="0" version="0">
                                                <x>0.0000000000000000e+00</x>
                                                <y>0.0000000000000000e+00</y>
                                                <z>-9.8100000000000050e+00</z>
                                        </gravity>
                                        <mask>0</mask>
                                        <warnOnce>1</warnOnce>
```

（续见下页）

（接上一页）

```
                        </px>
                </item>
        </engines>
        <_nextEngines>
                <count>0</count>
                <item_version>1</item_version>
        </_nextEngines>
        <bodies class_id="11" tracking_level="0" version="1">
                <px class_id="12" tracking_level="1" version="0" object_id="_7">
                        <Serializable object_id="_8"></Serializable>
                        <body class_id="13" tracking_level="0" version="0">
                                <count>0</count>
                                <item_version>1</item_version>
                        </body>
                        <insertedBodies>
                                <count>0</count>
                                <item_version>0</item_version>
                        </insertedBodies>
                        <erasedBodies>
                                <count>0</count>
                                <item_version>0</item_version>
                        </erasedBodies>
                        <realBodies>
                                <count>0</count>
                                <item_version>0</item_version>
                        </realBodies>
                        <useRedirection>0</useRedirection>
                        <enableRedirection>1</enableRedirection>
                </px>
        </bodies>
        <interactions class_id="15" tracking_level="0" version="1">
                <px class_id="16" tracking_level="1" version="0" object_id="_9">
                        <Serializable object_id="_10"></Serializable>
                        <interaction class_id="17" tracking_level="0" version="0">
                                <count>0</count>
                                <item_version>1</item_version>
                        </interaction>
                        <serializeSorted>0</serializeSorted>
                        <dirty>1</dirty>
                </px>
        </interactions>
        <energy class_id="18" tracking_level="0" version="1">
                <px class_id="19" tracking_level="1" version="0" object_id="_11">
                        <Serializable object_id="_12"></Serializable>
                        <energies class_id="20" tracking_level="0" version="0">
                                <size>0</size>
                        </energies>
                        <names class_id="21" tracking_level="0" version="0">
                                <count>0</count>
                                <item_version>0</item_version>
                        </names>
                        <resetStep>
                                <count>0</count>
                        </resetStep>
                </px>
        </energy>
        <materials class_id="23" tracking_level="0" version="0">
                <count>0</count>
                <item_version>1</item_version>
        </materials>
        <bound class_id="24" tracking_level="0" version="1">
```

（续见下一页）

（续前页）

```
                <px class_id="-1"></px>
        </bound>
        <cell class_id="26" tracking_level="0" version="1">
                <px class_id="27" tracking_level="1" version="0" object_id="_13">
                        <Serializable object_id="_14"></Serializable>
                        <trsf class_id="28" tracking_level="0" version="0">
                                <m00>1.0000000000000000e+00</m00>
                                <m01>0.0000000000000000e+00</m01>
                                <m02>0.0000000000000000e+00</m02>
                                <m10>0.0000000000000000e+00</m10>
                                <m11>1.0000000000000000e+00</m11>
                                <m12>0.0000000000000000e+00</m12>
                                <m20>0.0000000000000000e+00</m20>
                                <m21>0.0000000000000000e+00</m21>
                                <m22>1.0000000000000000e+00</m22>
                        </trsf>
                        <refHSize>
                                <m00>1.0000000000000000e+00</m00>
                                <m01>0.0000000000000000e+00</m01>
                                <m02>0.0000000000000000e+00</m02>
                                <m10>0.0000000000000000e+00</m10>
                                <m11>1.0000000000000000e+00</m11>
                                <m12>0.0000000000000000e+00</m12>
                                <m20>0.0000000000000000e+00</m20>
                                <m21>0.0000000000000000e+00</m21>
                                <m22>1.0000000000000000e+00</m22>
                        </refHSize>
                        <hSize>
                                <m00>1.0000000000000000e+00</m00>
                                <m01>0.0000000000000000e+00</m01>
                                <m02>0.0000000000000000e+00</m02>
                                <m10>0.0000000000000000e+00</m10>
                                <m11>1.0000000000000000e+00</m11>
                                <m12>0.0000000000000000e+00</m12>
                                <m20>0.0000000000000000e+00</m20>
                                <m21>0.0000000000000000e+00</m21>
                                <m22>1.0000000000000000e+00</m22>
                        </hSize>
                        <prevHSize>
                                <m00>1.0000000000000000e+00</m00>
                                <m01>0.0000000000000000e+00</m01>
                                <m02>0.0000000000000000e+00</m02>
                                <m10>0.0000000000000000e+00</m10>
                                <m11>1.0000000000000000e+00</m11>
                                <m12>0.0000000000000000e+00</m12>
                                <m20>0.0000000000000000e+00</m20>
                                <m21>0.0000000000000000e+00</m21>
                                <m22>1.0000000000000000e+00</m22>
                        </prevHSize>
                        <velGrad>
                                <m00>0.0000000000000000e+00</m00>
                                <m01>0.0000000000000000e+00</m01>
                                <m02>0.0000000000000000e+00</m02>
                                <m10>0.0000000000000000e+00</m10>
                                <m11>0.0000000000000000e+00</m11>
                                <m12>0.0000000000000000e+00</m12>
                                <m20>0.0000000000000000e+00</m20>
                                <m21>0.0000000000000000e+00</m21>
                                <m22>0.0000000000000000e+00</m22>
                        </velGrad>
                        <nextVelGrad>
```

（下页续）

（接上页）

```
                                        <m00>0.00000000000000000e+00</m00>
                                        <m01>0.00000000000000000e+00</m01>
                                        <m02>0.00000000000000000e+00</m02>
                                        <m10>0.00000000000000000e+00</m10>
                                        <m11>0.00000000000000000e+00</m11>
                                        <m12>0.00000000000000000e+00</m12>
                                        <m20>0.00000000000000000e+00</m20>
                                        <m21>0.00000000000000000e+00</m21>
                                        <m22>0.00000000000000000e+00</m22>
                                </nextVelGrad>
                                <prevVelGrad>
                                        <m00>0.00000000000000000e+00</m00>
                                        <m01>0.00000000000000000e+00</m01>
                                        <m02>0.00000000000000000e+00</m02>
                                        <m10>0.00000000000000000e+00</m10>
                                        <m11>0.00000000000000000e+00</m11>
                                        <m12>0.00000000000000000e+00</m12>
                                        <m20>0.00000000000000000e+00</m20>
                                        <m21>0.00000000000000000e+00</m21>
                                        <m22>0.00000000000000000e+00</m22>
                                </prevVelGrad>
                                <homoDeform>2</homoDeform>
                                <velGradChanged>0</velGradChanged>
                        </px>
                </cell>
                <miscParams class_id="29" tracking_level="0" version="0">
                        <count>0</count>
                        <item_version>1</item_version>
                </miscParams>
                <dispParams class_id="30" tracking_level="0" version="0">
                        <count>0</count>
                        <item_version>1</item_version>
                </dispParams>
        </px>
</scene>
</boost_serialization>
```

警告： 由于 XML 文件与 Yade 的实现细节紧密相关，因此不同版本之间不兼容。仅将它们用于短期保存场景。Python 是 Yade 使用的 高级描述语言。

Python 属性访问

宏 *YADE_类_底数_文档_*宏族如上所述，也用于（在幕后）创建从 Python 访问属性的函数。正如前面所述，可序列化属性集合与可从 Python 访问的属性集合完全相同。除了属性访问之外，这些包装类还模仿了常规 Python 字典的一些功能：

```
Yade [20]: s=Sphere()

Yade [21]: s.radius                ## read-access
Out[21]: nan

Yade [22]: s.radius=4.             ## write access

Yade [23]: s.dict().keys()                ## show all available keys
Out[23]: dict_keys(['radius', 'color', 'wire', 'highlight'])
```

（续见下页）

(续自上一页

YADE_CLASS_BASE_DOC_* 宏族

有几个宏将以下功能封装在其中：*Sphinx* 文档、运行时类型识别（*RTTI*）、属性注册、*Python* 属性访问，以及自动属性初始化和文档生成。它们都是基础宏 `YADE_CLASS_BASE_DOC_ATTRS_INIT_CTOR_PY` 的简写形式，省略了部分参数。必须将它们放在类声明的主体中（`.hpp` 文件）：

```
#define YADE_CLASS_BASE_DOC(klass,base,doc) \
        YADE_CLASS_BASE_DOC_ATTRS(klass,base,doc,)
#define YADE_CLASS_BASE_DOC_ATTRS(klass,base,doc,attrs) \
        YADE_CLASS_BASE_DOC_ATTRS_CTOR(klass,base,doc,attrs,)
#define YADE_CLASS_BASE_DOC_ATTRS_CTOR(klass,base,doc,attrs,ctor) \
        YADE_CLASS_BASE_DOC_ATTRS_CTOR_PY(klass,base,doc,attrs,ctor,)
#define YADE_CLASS_BASE_DOC_ATTRS_CTOR_PY(klass,base,doc,attrs,ctor,py) \
        YADE_CLASS_BASE_DOC_ATTRS_INIT_CTOR_PY(klass,base,doc,attrs,,ctor,py)
#define YADE_CLASS_BASE_DOC_ATTRS_INIT_CTOR_PY(klass,base,doc,attrs,init,ctor,py) \
        YADE_CLASS_BASE_DOC_ATTRS_INIT_CTOR_PY(klass,base,doc,attrs,inits,ctor,py)
```

预期参数由宏名称中以下划线分隔的组成部分表示，其含义如下：

`klass` （不带引号）此类的名称（用于 RTTI 和 Python）

`base` 基类名称（不加引号，用于 RTTI 和 Python）

`doc` 该类的文档字符串，使用 ReST 语法编写。此文档字符串将显示在生成的文档中（例如 *CpmMat*）。长度不限。使用 字符串字面量 来避免被 c++ 编译器解释的序列（这样某些反斜杠就不必重复两次，例如 $\sigma = \varepsilon E$），而不必这样写：

```
":math:`\\sigma=\\epsilon E"
```

请按以下方式编写： `R"""(:math:`\sigma=\epsilon E`)"""`。使用 `R"""(raw text)"""` 时，无须写出转义字符 `\n` 和 `\t`。可以改用换行符和制表符。例如，请参见 此处 和 此处。请充分使用超链接链接到文档 `yref` （所有对其他类的引用都应设为超链接）。有关引用和锚点用法的语法，请参见 上一节 。

`attrs` 属性必须写成括号列表的形式：

```
((type1,attr1,initValue1,attrFlags,"Attribute 1 documentation"))
((type2,attr2,,,"Attribute 2 documentation"))  // initValue and attrFlags unspecified
```

这将展开为

1. 数据成员 d　　在 c++ 中的声明（注意，全部属性都　　e *public*）：

```
public: type1 attr1;
        type2 attr2;
```

2. 默认（无参数）构造函数的初始化器，用于具有非空initValue的属性：

```
Klass(): attr1(initValue1), attr2() { /* constructor body */ }
```

初始值留空的属性不会被赋予初始值（如上例中的 attr2）。请注意，仍需写上逗号。

3. 在序列化系统中注册该属性（除非通过 attrFlags 禁用——见下文）

4. **在 Python 中注册该属性（除非由 attrFlags 禁用），以便可以访问**
该属性默认可读写，参见 attrFlags 可更改此设置。
该属性将附带所提供的文档字符串，以及关于初始值的信息。可以使用 c++ 的逗号运算符，并将 char* 转换为 (void)，为默认值添加文字描述：

```
((Real,dmgTau,((void)"deactivated if negative",-1),,"Characteristic time for␣
→normal viscosity. [s]"))
```

从而得到 *CpmMat::dmgTau*。
该属性通过 `boost::python::add_property` 指定 `return_by_-value` 策略进行注册，而不是使用 `return_internal_reference`；使用def_readwrite时，默认采用后者。原因是我们需要确保这些值使用自定义转换器；详情请参阅 自定义转换器 中的说明。

属性标志

默认情况下，属性会被序列化，并且在 Python 中可读写。可以将一些标志作为第 4 个参数传入（默认为空），以改变这一行为：

- `Attr::noSave` 避免序列化该属性（同时仍保留从 Python 访问该属性的能力）
- `Attr::readonly` 使该属性在 Python 中为只读
- `Attr::triggerPostLoad` 将在从 Python 设置属性值后，触发调用 `postLoad` 函数来处理属性变更；这样可以确保依赖于该值的其他预计算数据（例如 `Cell.trsf` 等）保持一致
- `Attr::hidden` 将完全不向 Python 暴露该属性
- `Attr::noResize` 不允许通过 Python 更改数组的尺寸 [尚未使用]

标志通常可以使用按位析取进行组合 `|`（例如 `Attr::noSave |Attr::readonly`），不过在这种情况下，应将值括在括号中，以避免某些编译器（特别是 g++ ）发出警告，即 `(Attr::noSave | Attr::readonly)`。
目前，标志逻辑是在运行时处理的；这意味着，即使对于具有Attr::noSave的属性，也必须定义其序列化模板（尽管它永远不会被使用）。未来的实现可能会基于模板，从而避免这一要求。

`deprec` 已弃用的属性名称列表。语法为

```
((oldName1,newName1,"Explanation why renamed etc."))
((oldName2,newName2,"! Explanation why removed and what to do instead."))
```

这样访问 `oldName1` 属性 时，*Python* 会返回 `newName`的值，但会显示关于属性名称变更的警告消息，并显示所提供的说明。无论是读取还是写入访问，都会发生这种情况。

如果说明的第一个字符是 `!`(感叹号)，则该消息会在属性访问时显示，但会立即抛出异常。当属性已不再有意义，或无法通过直接替换为其他属性来处理，而需要对用户脚本进行更复杂的调整时，请使用此方式。你仍然必须提供 `newName2`，尽管其值永远不会被使用——出于语法原因，你可以使用任意变量，但必须提供一个变量）。

警告： 由于编译器限制，只有使用 gcc >=4.4 编译 Yade 时，此功能才有效。否则，即使声明了此类属性，弃用属性功能也会被禁用。

`init` 形如以下形式的括号列表：

```
((attr3,value3)) ((attr4,value4))
```

它将在默认构造函数中展开为初始化器：

```
Klass(): /* attributes declared with the attrs argument */ attr4(value4), attr5(value5) {␣
→/* constructor body */ }
```

此参数的作用是使常数和引用能够初始化（它们本身并未使用此宏声明为属性，而是单独声明的），因为在构造函数体中无法完成这项操作。不过，这个参数很少使用。

`ctor` 将直接放入生成的构造函数体中。主要用于在构造函数中调用 createIndex()。

注意： 代码中括号外不得包含逗号（因为预处理器会使用逗号分隔宏参数）。如果需要在构造时执行复杂操作，请创建一个单独的 init() 函数，并在构造函数中调用它。

`py` 将直接追加到用于注册类及其所有属性的生成的 Python 代码之后。你可以用它从 Python 访问类方法，例如覆盖现有的同名属性等：

```
.def_readonly("omega",&CpmPhys::omega,"Damage internal variable")
.def_readonly("Fn",&CpmPhys::Fn,"Magnitude of normal force.")
```

`def_readonly` 对自定义类型（例如 std::vector）不起作用，因为它绕过了转换 n 注册 请尝试；详见 自定义转换器 。

向图形用户界面公开函数属性

通常，要公开更复杂的数据，会使用获取器和设置器函数，例如 *Body::mask*。它们可在 Python 中访问。若要在没有对应变量的情况下让它们显示在图形用户界面中，则必须重写函数 `virtual ::boost::python::dict pyDictCustom() const { …… };`。例如，请参阅 Interaction.hpp，其中有一个特殊属性 `isReal` 会显示在图形用户界面中。若要将此类属性标记为只读，必须在其文档字符串中添加额外信息：`:yattrflags:`2``。通常，这些信息由类属性注册宏写入其中。但由于它不是变量，因此必须手动添加此类属性。

特殊 Python 构造函数

Python 封装器会自动创建一个构造函数，该构造函数接收与实例属性对应的关键字（命名）参数；这些属性会被设为构造函数中提供的值。在某些情况下，我们希望有更大的灵活性（例如 *InteractionLoop*，它接收 3 个函子列表）。对于这类情况，可以重写函数 `Serializable::pyHandleCustomCtorArgs`，以任意方式修改新建的（已存在的）实例。该函数应就地修改传入的参数，因为这些参数会继续传递给设置属性值的例程。在这种情况下，应为构造函数编写文档：

```
.. admonition:: Special constructor

        Constructs from lists of …
```

随后会以类似以下形式出现在文档中：*InteractionLoop*。

枚举

可以在图形用户界面的下拉菜单中公开 `enum` 和 `enum class`（`enum class` 是首选，因为它具有更强的类型安全性，可防止程序员出错）。这种方法向后兼容，在旧版 Python 脚本中对 `int` 值进行赋值，仍会像以前一样正常工作。此外，还可以将 `string` 类型的值赋给枚举。要启用下拉菜单，必须在 `.cpp` 文件中放入宏 `YADE_ENUM( Scope , EnumName , (ValueName1)(ValueName2)(ValueName3)(ValueName4) )`。每个宏参数的含义如下：

1. `Scope` 是枚举所在作用域的完整名称。例如，`yade::OpenGLRenderer::BlinkHighlight`的作用域是 `yade::OpenGLRenderer`。
2. `EnumName` 是要注册的枚举名称 3. `Value`

   ~~Name~~ he C++ 是要向 Python 开放的所有枚举值。如果 t
   枚举声明在 `.hpp` 文件中发生更改。

注册后，例如在 OpenGLRenderer.cpp 中注册后即可使用。此外，已注册的枚举类类型定义也会在 `yade.EnumClass_*` 作用域中公开，例如可以检查 `names` 和 `values` 字典：

```
Yade [27]: yade.EnumClass_BlinkHighlight.namesOut[27]:
{'NEVER': yade.EnumClass_BlinkHighlight.NEVER,
'NORMAL': yade.EnumClass_BlinkHighlight.NORMAL,
'WEAK': yade.EnumClass_BlinkHighlight.WEAK}


Yade [28]: yade.EnumClass_BlinkHighlight.values
Out[28]:
{0: yade.EnumClass_BlinkHighlight.NEVER,
1: yade.EnumClass_BlinkHighlight.NORMAL,
2: yade.EnumClass_BlinkHighlight.WEAK}
```

请注意，这些**并不是变量实例**，因此尝试为它们赋值不会改变图形用户界面中的闪烁高亮设置。若要在 Python 中更改枚举值，必须为相应变量赋值，例如 *yade.qt.Renderer().blinkHighlight*。尝试赋予不正确的值会引发异常。例如：

```
Yade [29]: r = yade.FlowEngine() # this is only a test of enum, not of FlowEngine
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
----> 1 r = yade.FlowEngine() # this is only a test of enum, not of FlowEngine
```

（续见下页）

（接上一页）

```
AttributeError: module 'yade' has no attribute 'FlowEngine'

Yade [30]: r.useSolver
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
----> 1 r.useSolver

NameError: name 'r' is not defined

Yade [31]: r.useSolver = 'GaussSeidel'
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
----> 1 r.useSolver = 'GaussSeidel'

NameError: name 'r' is not defined

Yade [32]: try:
   ....:     r.useSolver = 20    # assigning incorrect value has no effect
   ....: except:
   ....:     print("Error, value is still equal to:",r.useSolver)
   ....:
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
      1 try:
----> 2     r.useSolver = 20    # assigning incorrect value has no effect
      3 except:

NameError: name 'r' is not defined

During handling of the above exception,  another exception occurred:

NameError                                 Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
      2     r.useSolver = 20    # assigning incorrect value has no effect
      3 except:
----> 4     print("Error, value is still equal to:",r.useSolver)

NameError: name 'r' is not defined

Yade [33]: r.useSolver
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
----> 1 r.useSolver

NameError: name 'r' is not defined
```

也可以使用图形用户界面中的下拉菜单实现相同的效果。

静态属性

某些类（例如 OpenGL 函子）会自动实例化；由于我们希望它们的属性在整个会话期间保持不变，因此将其设为静态。若要公开带有静态属性的类，请使用YADE_CLASS_BASE_DOC_STATICATTRS 宏。属性的语法与 YADE_CLASS_BASE_-DOC ATTRS:_ 相同。

```
class SomeClass: public BaseClass{
        YADE_CLASS_BASE_DOC_STATICATTRS(SomeClass,BaseClass,"Documentation of SomeClass",
                ((Type1,attr1,default1,"doc for attr1"))
                ((Type2,attr2,default2,"doc for attr2"))
        );
};
```

此外，你还需要在 `.cpp` 文件中为静态数据成员分配内存（否则，加载插件时会出现未定义符号错误）：

无法使用 `YADE_CLASS_BASE_*`宏公开同时具有静态和非静态属性的类。你必须按常规方式公开非静态属性，并在 `py` 参数中单独包装静态属性。

按值或按引用返回属性

将属性从 c++ 传递到 Python 时，可以采用以下方式之一：

- 按值传递：会构造一个表示原始 c++ 对象的新 Python 对象，但不会将其绑定到原对象；修改该 Python 对象不会改变 c++ 对象，除非显式地将其赋回，此时会进行逆转换，并替换 c++ 对象。

- 引用：仅将底层 c++ 对象的引用返回给 Python；修改 Python 对象时，c++ 对象也会自动修改。

传递给 `YADE_CLASS_BASE_DOC_ATTRS attrs` 参数的属性所采用的传递方式会按以下方式自动确定：

- **`Vector3`、 `Vector3i`、 `Vector2`、 `Vector2i`、 `Matrix3` 和 `Quaternion` 对象通过引用传递。例如**
  O.bodies[0].state.pos[0]=1.33 会将正确的值赋给位置的 `x` 分量，而不会更改其他分量。
- **Yade 类（所有使用 `shared_ptr` 在 Python 中声明时：所有派生自可序列化对象**
  O.engines[4].阻尼=.3 会更改原始引擎对象上的阻尼参数，而不是其副本上的参数。
- **所有其他类型均按值传递。其中最重要的是，在 *C* 中声明的序列类型**
  O.engines[4]=NewtonIntegrator() 将无法按预期工作；它会替换序列副本的第 5 个元素，而此更改不会传播回 c++。

多重分派

多重分派是虚方法的一种泛化： 分派器 根据其参数的类型，决定调用哪个 函子 。参数的数量（目前为 1 或 2）决定了分派器（及函子）的 元数 ：一元或二元。例如：

```
InsertionSortCollider([Bo1_Sphere_Aabb(),Bo1_Facet_Aabb()])
```

创建 *InsertionSortCollider*，它在内部包含 *Collider.boundDispatcher*，一个 *BoundDispatcher* （一个分派器），其中有 2 个函子；它们接收 `Sphere` 或 `Facet` 实例并创建 `Aabb`。这段代码用 C++编写如下：

```
shared_ptr<InsertionSortCollider> collider=(new InsertionSortCollider);
collider->boundDispatcher->add(new Bo1_Sphere_Aabb());
collider->boundDispatcher->add(new Bo1_Facet_Aabb());
```

目前有 4 个预定义的分派器（参见 分派器名称）及其对应的函子类型。它们继承自以下模板的实例化： `Dispatcher1D` 或 `Dispatcher2D` （用于函子，

Functor1D 或 Functor2D）。这些模板本身分别派生自 DynlibDispatcher （用于分派器）和 FunctorWrapper （用于函子）。

示例：IGeomDispatcher

我们来看看（或许是最复杂的）*IGeomDispatcher*。*IGeomFunctor*，它根据两个形状实例（一个函子）进行分派，接收若干参数并返回布尔值。函子的“调用”始终由其重写的 Functor::go 方法提供；它始终将参与分派的实例作为第一个参数（2 × const shared_ptr<Shape>&），并接收它所需的若干其他参数：

```
class IGeomFunctor: public Functor2D<
   bool,                                  //return type
   TYPELIST_7(const shared_ptr<Shape>&,   // 1st class for dispatch
      const shared_ptr<Shape>&,           // 2nd class for dispatch
      const State&,                       // other arguments passed to ::go
      const State&,                       // …
      const Vector3r&,                    // …
      const bool&,                        // …
      const shared_ptr<Interaction>&      // …
   )
>
```

分派器声明如下：

```
class IGeomDispatcher: public Dispatcher2D<
   Shape,                        // 1st class for dispatch
   Shape,                        // 2nd class for dispatch
   IGeomFunctor,   // functor type
   bool,                         // return type of the functor

   // follow argument types for functor call
   // they must be exactly the same as types
   // given to the IGeomFunctor above.
   TYPELIST_7(const shared_ptr<Shape>&,
      const shared_ptr<Shape>&,
      const State&,
      const State&,
      const Vector3r&,
      const bool &,
      const shared_ptr<Interaction>&
   ),

   // handle symetry automatically
   // (if the dispatcher receives Sphere+Facet,
   // the dispatcher might call functor for Facet+Sphere,
   // reversing the arguments)
   false
>
{ /* … */ }
```

继承自 IGeomFunctor 的函子接下来必须

- 使用适当的参数重载 ::go 方法（参数类型必须与传递给TYPELIST_* 宏的类型完全一致）；
- 声明它们应针对哪些类型进行分派，以及在类型不同时采用何种顺序。

```
class Ig2_Facet_Sphere_ScGeom: public IGeomFunctor{
   public:
```

（下页继续）

（接上页）

```
   // override the IGeomFunctor::go
   //   (it is really inherited from FunctorWrapper template,
   //    therefore not declare explicitly in the
   //    IGeomFunctor declaration as such)
   // since dispatcher dispatches only for declared types
   //   (or types derived from them), we can do
   //   static_cast<Facet>(shape1) and static_cast<Sphere>(shape2)
   //   in the ::go body, without worrying about types being wrong.
   virtual bool go(
      // objects for dispatch
      const shared_ptr<Shape>& shape1, const shared_ptr<Shape>& shape2,
      // other arguments
      const State& state1, const State& state2, const Vector3r& shift2,
      const bool& force, const shared_ptr<Interaction>& c
   );
   /* … */

   // this declares the type we want to be dispatched for, matching
   //   first 2 arguments to ::go and first 2 classes in TYPELIST_7 above
   //   shape1 is a Facet and shape2 is a Sphere
   //   (or vice versa, see lines below)
   FUNCTOR2D(Facet,Sphere);

   // declare how to swap the arguments
   //   so that we can receive those as well
   DEFINE_FUNCTOR_ORDER_2D(Facet,Sphere);
   /* … */
};
```

分派解析

分派器并不总是具有与其接收到的实际类型完全匹配的函子。与虚方法类似，它会尝试按以下方式找到最接近的匹配项：

1. 实际实例是函子接受的类型的派生类型，或与接受类型完全相同；
2. 实际类型到接受类型的距离总和达到严格最小值（类层次结构中的每上升一步计为 1）

如果没有函子能够接受给定类型（违反第一个条件），或者多个函子的距离相同（违反条件 2），则会抛出异常。

这种解析机制使得例如可以建立 *ScGeom* 类的层次结构（用于不同的形状组合），但只需提供一个 *LawFunctor*，接受 `ScGeom`，而不必为每种形状组合定义不同的定律。

注意： 分派解析的性能影响相对较低。分派器查找仅执行一次，并使用快速查找矩阵（一维或二维）；随后，为此类型找到的函子会缓存于 `Interaction`（或 `Body`）实例中。因此，常规函子调用的开销与解引用指针并调用虚方法相同。此外，还有 方案 可用于避免调用虚函数。

注意： 一开始，分派矩阵只包含与给定函子完全匹配的条目。只有在必要时（绕过其他类型），才会填充相应的条目。

分派类型索引

参与分派机制的类必须能够快速识别自身及其父类。[5] 这称为类索引，所有此类类都派生自可索引类。共有top-level 种可索引类（分派器接受的类型），每个派生类都会注册其相对于此顶层可索引对象的索引。目前有：

顶层可索引对象	使用者
形状	边界函子，*IGeomDispatcher*
材料	*IPhysDispatcher*
IPhys	*LawDispatcher*
IGeom	*LawDispatcher*

顶层可索引对象必须使用 REGISTER_INDEX_COUNTER 宏，它负责设置识别派生类类型所需的机制；随后，这些类必须在其构造函数中使用 REGISTER_CLASS_INDEX 宏 并 调用createIndex()。例如，以 形状 类（它是顶层可索引对象）为例：

```
// derive from Indexableclass Shape: public Serializable,
 public Indexable {
// never call createIndex() in the top-level Indexable ctor!
/* … */
// allow index registration for classes deriving from ``Shape``
REGISTER_INDEX_COUNTER(Shape);};
```

现在，所有派生类（例如 球体 或 面片）都使用这个：

```
class Sphere: public Shape{
   /* … */
   YADE_CLASS_BASE_DOC_ATTRS_CTOR(Sphere,Shape,"docstring",
      ((Type1,attr1,default1,"docstring1"))
      /* … */,
      // this is the CTOR argument
         // important; assigns index to the class at runtime
         createIndex();
   );
   // register index for this class, and give name of the immediate parent class
   //    (i.e. if there were a class deriving from Sphere, it would use
   //     REGISTER_CLASS_INDEX(SpecialSphere,Sphere),
   //     not REGISTER_CLASS_INDEX(SpecialSphere,Shape)!)
   REGISTER_CLASS_INDEX(Sphere,Shape);
};
```

在运行时，顶层可索引类层次结构中的每个类都有各自唯一的数值索引。这些索引用于为每个分派器构建分派矩阵。

在 Python 中检查分派

如果需要调试或研究多重分派，Python 提供了便捷的接口来使用这一底层功能。

我们可以通过 dispIndex 属性检查索引（注意，顶层可索引对象 Shape 的类索引为负数（无效）；我们有意没有在其构造函数中调用 createIndex ）：

[5] 此处所述的功能运行时类型识别（*RTTI*）用途不同（序列化），而且会影响此处的性能。因此，类除了字符串之外，还会提供数字（索引）。

```
Yade [34]: Sphere().dispIndex, Facet().dispIndex, Wall().dispIndex
Out[34]: (1, 5, 10)

Yade [35]: Shape().dispIndex                    # top-level indexable
Out[35]: -1
```

可以使用 dispHierarchy() 函数显示特定类的分派层次结构，该函数返回类名列表：第 0 个元素是实例本身，最后一个元素是顶层可索引对象（同样使用无效索引）；例如：

```
Yade [36]: ScGeom().dispHierarchy()        # parent class of all other ScGeom_ classes
Out[36]: ['ScGeom', 'GenericSpheresContact', 'IGeom']

Yade [37]: ScGridCoGeom().dispHierarchy(), ScGeom6D().dispHierarchy(), CylScGeom().
→dispHierarchy()
Out[37]:
(['ScGridCoGeom', 'ScGeom6D', 'ScGeom', 'GenericSpheresContact', 'IGeom'],
 ['ScGeom6D', 'ScGeom', 'GenericSpheresContact', 'IGeom'],
 ['CylScGeom', 'ScGeom', 'GenericSpheresContact', 'IGeom'])

Yade [38]: CylScGeom().dispHierarchy(names=False)    # show numeric indices instead
Out[38]: [4, 1, 0, -1]
```

还可以使用 .dispMatrix() 方法检查分派器：

```
Yade [39]: ig=IGeomDispatcher([
....:     Ig2_Sphere_Sphere_ScGeom(),
....:     Ig2_Facet_Sphere_ScGeom(),
....:     Ig2_Wall_Sphere_ScGeom()
....: ])....:

Yade [40]: ig.dispMatrix()
Out[40]:
{('Sphere', 'Sphere'): 'Ig2_Sphere_Sphere_ScGeom',
('Sphere', 'Facet'): 'Ig2_Facet_Sphere_ScGeom',
('Sphere', 'Wall'): 'Ig2_Wall_Sphere_ScGeom',
('Facet', 'Sphere'): 'Ig2_Facet_Sphere_ScGeom',
('Wall', 'Sphere'): 'Ig2_Wall_Sphere_ScGeom'}

Yade [41]: ig.dispMatrix(False)           # don't convert to class names
Out[41]:
{(1, 1): 'Ig2_Sphere_Sphere_ScGeom',
(1, 5): 'Ig2_Facet_Sphere_ScGeom',
(1, 10): 'Ig2_Wall_Sphere_ScGeom',
(5, 1): 'Ig2_Facet_Sphere_ScGeom',
(10, 1): 'Ig2_Wall_Sphere_ScGeom'}
```

我们可以看到，函子利用了对称性（即 Sphere+Wall 和 Wall+Sphere 会被分派给同一个函子）。

最后，可以要求分派器返回适用于给定参数的函子：

```
Yade [42]: ld=LawDispatcher([Law2_ScGeom_CpmPhys_Cpm()])

Yade [43]: ld.dispMatrix()
Out[43]: {('GenericSpheresContact', 'CpmPhys'): 'Law2_ScGeom_CpmPhys_Cpm'}

# see how the entry for ScGridCoGeom will be filled after this request
Yade [44]: ld.dispFunctor(ScGridCoGeom(),CpmPhys())
Out[44]: <Law2_ScGeom_CpmPhys_Cpm instance at 0x397baf0>
```

（续见下一页）

（接上页）

```
Yade [45]: ld.dispMatrix()
Out[45]:
{('GenericSpheresContact', 'CpmPhys'): 'Law2_ScGeom_CpmPhys_Cpm',
 ('ScGridCoGeom', 'CpmPhys'): 'Law2_ScGeom_CpmPhys_Cpm'}
```

OpenGL 函子

OpenGL 渲染也由一维函子完成（根据待渲染的类型进行分派）。由于每种待渲染类型只需一个类，因此会自动找到这些函子。它们的底数函子类型为 `GlShapeFunctor`, `GlBoundFunctor`, `GlIGeomFunctor` 等。这些类使用 `RENDERS`宏注册其渲染的类型：

```
namespace yade { // Cannot have #include directive inside.
class Gl1_Sphere: public GlShapeFunctor {
   public :
      virtual void go(const shared_ptr<Shape>&,
         const shared_ptr<State>&,
         bool wire,
         const GLViewInfo&
      );
   RENDERS(Sphere);
   YADE_CLASS_BASE_DOC_STATICATTRS(Gl1_Sphere,GlShapeFunctor,"docstring",
      ((Type1,staticAttr1,informativeDefault,"docstring"))
      /* … */
   );
};
REGISTER_SERIALIZABLE(Gl1_Sphere);
} // namespace yade
```

通过查询底数函子的子类，可以列出特定类型的可用函子：

```
Yade [46]: yade.system.childClasses('GlShapeFunctor')
Out[46]:{'Gl1_Box','Gl1_ChainedCylinder','Gl1_Cylinder',
'Gl1_Facet','Gl1_GridConnection','Gl1_PFacet',
'Gl1_Sphere','Gl1_Tetra','Gl1_Wall'}
```

注意： OpenGL 函子未来可能会被移除，改由每个可渲染类的虚函数取代。

并行执行

Yade 最初的设计并未考虑并行计算，而是优先追求最大的灵活性（无论好坏）。后来才加入并行执行；为了避免从头重写整个 Yade，采用了一种相对非侵入式的并行化方式： OpenMP。OpenMP 是一种标准化的共享内存并行执行环境，代码中的并行区段会用特殊的 `#pragma` 标记（这意味着代码也能通过不支持 OpenMP 的编译器编译），并提供了一些函数，以便在必要时查询或操作 OpenMP 运行时。

共有 3 个层级的并行性：

- 计算、相互作用（Python、图形用户界面）和渲染线程彼此独立。这通过常规线程（boost::threads）实现，与 OpenMP 无关。
- *ParallelEngine* 可以并行运行多个引擎组（这些引擎组内部仍按串行方式运行）；它很少用于常规模拟，但例如在与独立的高耗时计算耦合时可能会派上用场：

```
ParallelEngine([
        [Engine1(),Engine2()],     # Engine1 will run before Engine2
        [Engine3()]                # Engine3() will run in parallel with the group␣
→[Engine1(),Engine2()]
                                   # arbitrary number of groups can be used
])
```

Engine2 将在 Engine1 之后运行，但与 Engine3 并行运行。

警告： 使用 ParallelEngine 时，避免并发访问数据是您的责任。请确保您充分 理解与其并行运行的引擎的内容。

- 引擎内部的并行性。对刚体或相互作用的某些循环进行了并行化处理（尤其是 交互循环 和 *NewtonIntegrator*，后文将详细介绍（待修复：链接））：

```
#pragma omp parallel for
for(long id=0; id<size; id++){
   const shared_ptr<Body>& b(scene->bodies[id]);
   /* … */
}
```

注意：OpenMP 要求循环遍历连续整数范围（OpenMP 3 也支持使用随机访问迭代器的容器）。

如果考虑在引擎中运行并行循环，务必评估其收益。OpenMP 创建线程和分配工作负载需要一定开销；如果循环体执行速度很快，这些开销所占比例就会更高。结果在很大程度上取决于硬件（CPU 缓存、内存控制器）。

OpenMP 线程的最大数量由 OMP_NUM_THREADS 环境变量决定，并且在程序运行期间保持不变。如果使用 *-j*/--threads 选项，Yade 主程序也会设置此变量（在加载 OpenMP 库之前）。运行时可通过 omp_get_max_threads 函数查询该值。

对于可能被多个线程并发访问的位置，Yade 提供了一些互斥机制，相关内容将在其他地方讨论（待修复）：

- 用于 *ForceContainer* 的可同时写入容器，
- 用于 *Body::state* 的互斥锁。

计时

Yade 提供了 2 种服务，用于测量代码不同部分所耗费的时间。其中一种以引擎为粒度，并且可以在运行时启用。另一种的粒度更细，但需要调整并重新编译要测量的代码。

逐引擎计时

较粗粒度的计时方式仅通过累计每个引擎的调用次数和耗时（精度与 `clock_gettime` 函数 相同）来工作，之后可以使用关联的 Python 模块 `yade.timing` 进行后处理。还有一个静态布尔变量用于控制是否进行此类测量（默认禁用），你可以更改该变量

```
TimingInfo::enabled=True;            // in c++
```

```
O.timingEnabled=True                 ## in python
```

运行模拟后， `yade.timing.stats()` 函数将显示包含结果和百分比的表格:

```
Yade [47]: TriaxialTest(numberOfGrains=100).load()

Yade [48]: O.engines[0].label='firstEngine'   ## labeled engines will show by labels in the␣
 →stats table

Yade [49]: import yade.timing;

Yade [50]: O.timingEnabled=True

Yade [51]: yade.timing.reset()                  ## not necessary if used for the first time

Yade [52]: O.run(50); O.wait()

Yade [53]: yade.timing.stats()
Name                                                Count                Time           ␣
 →Rel. time
-------------------------------------------------------------------------------------------
 →--------
"firstEngine"                                          50             15.278us                0.
 →26%
InsertionSortCollider                                  25           1866.469us               32.
 →16%
InteractionLoop                                        50           2849.897us               49.
 →11%
GlobalStiffnessTimeStepper                              2             15.417us                0.
 →27%
TriaxialCompressionEngine                              50            262.545us                4.
 →52%
TriaxialStateRecorder                                   3            145.883us                2.
 →51%
NewtonIntegrator                                       50            647.859us               11.
 →16%
  forces sync                                          50              7.259us               ␣
 →1.12%
  motion integration                                   50            624.789us               ␣
 →96.44%
  sync max vel                                         50              3.264us               ␣
 →0.50%
  terminate                                            50              2.337us               ␣
 →0.36%
  TOTAL                                               200            637.649us               ␣
 →98.42%
TOTAL                                                                5803.348us              100.
 →00%
```

在 Python 中，可以通过 `Engine::timingInfo` from c++ or`Engine().execCount` and `Engine().execTime` 属性访问和操作执行次数与时间。

引擎内与函子内计时

引擎（以及函子）中的计时基于 *TimingDeltas* 类。该类默认在引擎和函子中实例化为 Engine::timingDeltas 和 Functor::timingDeltas（*Engine.timingDeltas* 和 *Functor.timingDeltas* 在 Python 中）。它用于计时循环（函子的循环位于各自的分派器中），并累积存储各个检查点之间的时间差。

注意： 只有在全局启用计时的情况下，使用 `TimingDeltas` 进行精细计时才会生效（参见上一节）。代码仍会运行，但计时结果和执行次数都为零。

---

1. 最好在构造函数中定义 timingDeltas 属性：

```
// header fileclass Law2_ScGeom_CpmPhys_Cpm: public LawFunctor {/* … */
YADE_CLASS_BASE_DOC_ATTRS_CTOR(Law2_ScGeom_CpmPhys_Cpm,LawFunctor,"docstring",
/* attrs */,/* constructor */
timingDeltas=shared_ptr<TimingDeltas>(new TimingDeltas); // timingDeltas␣
→object is automatically initialized when using -DENABLE_PROFILING=1 cmake␣
→option);// .. };
```

2. 在循环中，通过调用 `timingDeltas->start();` 开始计时

3. 在感兴趣的位置调用 `timingDeltas->checkpoint("label")`。标签仅用于后处理，数据根据检查点的位置存储，而不是根据标签存储。

> **警告：** 必须始终按相同顺序到达检查点，否则计时数据将变得毫无意义。代码仍然可以分支，但必须将检查点放在各分支共有的位置。

```
void Law2_ScGeom_CpmPhys_Cpm::go(shared_ptr<IGeom>& _geom,
                                 shared_ptr<IPhys>& _phys,
                                 Interaction* I,
                                 Scene* scene)
{
   timingDeltas->start();                          // the point at which the first␣
→timing starts
   // prepare some variables etc here
   timingDeltas->checkpoint("setup");
   // find geometrical data (deformations) here
   timingDeltas->checkpoint("geom");
   // compute forces here
   timingDeltas->checkpoint("material");
   // apply forces,  cleanup here
   timingDeltas->checkpoint("rest");
}
```

4. **或者，你可以 使用 -DENABLE_PROFILING=1CMake 选项编译 Yade，并使用预**

```
void Law2_ScGeom_CpmPhys_Cpm::go(shared_ptr<IGeom>& _geom,
                                 shared_ptr<IPhys>& _phys,
                                 Interaction* I,
                                 Scene* scene)
```

（续见下页）

（接上一页）

```
{
   TIMING_DELTAS_START();
   // prepare some variables etc here
   TIMING_DELTAS_CHECKPOINT("setup")
   // find geometrical data (deformations) here
   TIMING_DELTAS_CHECKPOINT("geom")
   // compute forces here
   TIMING_DELTAS_CHECKPOINT("material")
   // apply forces,  cleanup here
   TIMING_DELTAS_CHECKPOINT("rest")
}
```

输出可能如下所示（注意，函子嵌套在分派器内，也嵌套在其引擎/函子 TimingDeltas内）：

```
Name                                 Count                 Time          Rel. time
-------------------------------------------------------------------------------
ForceReseter                           400               9449µs               0.01%
BoundDispatcher                        400            1171770µs               1.15%
InsertionSortCollider                  400            9433093µs               9.24%
IGeomDispatcher        400           15177607µs              14.87%
IPhysDispatcher         400            9518738µs               9.33%
LawDispatcher                          400           64810867µs              63.49%
  Law2_ScGeom_CpmPhys_Cpm
    setup                              4926145            7649131µs              15.25%
    geom                               4926145           23216292µs              46.28%
    material                           4926145            8595686µs              17.14%
    rest                               4926145           10700007µs              21.33%
    TOTAL                                                50161117µs             100.00%
NewtonIntegrator                       400            1866816µs               1.83%
"strainer"                             400              21589µs               0.02%
"plotDataCollector"                    160              64284µs               0.06%
"damageChecker"                          9               3272µs               0.00%
TOTAL                                                102077490µs             100.00%
```

警告： 不要在 *TimingDeltas* 并行区段中使用，结果可能没有意义。尤其应避免在 *InteractionLoop* 使用多个 OpenMP 线程运行时使用计时函子。

TimingDeltas 数据可通过 Python 以由（标签、*时间*、*计数*）组成的元组列表形式访问，每个检查点对应一个元组：

```
deltas=someEngineOrFunctor.timingDeltas.data()
deltas[0][0] # 0th checkpoint label
deltas[0][1] # 0th checkpoint time in nanoseconds
deltas[0][2] # 0th checkpoint execution count
deltas[1][0] # 1st checkpoint label
             # …
deltas.reset()
```

计时开销

较粗粒度的逐引擎计时开销非常小。对于包含数百个以上元素的模拟，其开销低于通常的时间波动范围（百分之几）。

更精细的*TimingDeltas* 计时会对性能产生重大影响，因此只应在调试和性能调优阶段使用。经过文件计时的部分所花费的时间会远远长于引擎的其他部分；在上面的输出中，*LawDispatcher* 几乎占了 的总计

平均模拟时间，但这个数值通常会低两到三倍（请注意，在这个特定算例中，每个检查点都被计时了近 500 万次）。

OpenGL 渲染

Yade 提供基于 QGLViewer 的三维渲染功能。它并非旨在提供功能完备的渲染和后处理，而是为了快速检查场景是否符合预期，或模拟运行是否正常。

注意： 尽管三维渲染在独立线程中运行，但它会影响计算本身的性能，因为交互容器在创建/删除交互时需要互斥。`InteractionContainer::drawloopmutex` 要么由渲染器（*OpenGLRenderingEngine*）持有，要么由插入/删除例程持有。

警告： 崩溃可能由以下两种原因引起。由于可能造成严重的性能损失，因此不会对其进行防范：

1. 访问 *BodyContainer*，尤其是从模拟中删除刚体；这种操作很少发生，不过。
2. 删除 Interaction::phys 或 Interaction::geom。

可渲染实体（形状、状态、*Bound*、*IGeom*、*IPhys*）具有相应的*OpenGL 函子*。如果满足以下条件，实体就会被渲染：

1. 通过 *OpenGLRenderingEngin* 中相应的属性启用此类实体的渲染
   e2. 通过分派机制找到该特定实体类型对应的函子。

`Gl1_*` 对刚体属性进行操作的函子（形状、状态、*Bound*）会根据*State::pos* 和*State::ori*，在经过平移和旋转的 OpenGL 上下文中调用。相互作用函子则在全局坐标中工作。

3.1.7 仿真框架

除了前　一节提到的支持框架之外，还提供了一些与　o
模拟　本身相关的功能。
用于存储刚体、相互作用和（广义）力的专用容器。它们的内部工作机制通常对程序员来说是不透明的，但应当了解，因为这会影响性能。

场景

`Scene` 是包含整个模拟的对象。虽然内存中可以存在多个场景，但其中只有一个处于活动状态。保存和加载（序列化和反序列化） `Scene` 对象，应能让模拟从中断处继续运行。

注意： 全部 引擎 和函子内部都有一个 `Scene* scene` 指针，调用引擎/函子的代码会定期更新它；这样便可在用户代码中访问当前场景。
对于外部函数（例如从 Python 调用的函数，或 `Shop` 中的静态函数），你可以使用 `Omega::instance().getScene()` 来获取当前场景的 `shared_ptr<Scene>` 。

刚体容器

刚体容器是用于存储刚体的线性存储结构。模拟中的每个刚体都有唯一的标识符，必须通过该标识符在*BodyContainer*中找到。尚未加入模拟的刚体通常具有无效值 `Body::ID_NONE`作为标识符，并会在插入容器时获得其 `id` 。对刚体容器的要求如下：

- 以 O(1) 时间访问元素，
- 线性可寻址性（可按 0...n 索引），
- 存储 `shared_ptr`，而非对象本身，
- 否 对插入/移除操作进行互斥（如有需要，必须由调用方确保），
- 为新刚体智能分配 `id` （跟踪已删除的刚体），
- 便于遍历全部刚体。

注意： 目前， `BodyContainer`是一个“抽象”类，具体实现类由它派生而来；最初的设想是能够在运行时选择要使用的实现（以便找出在给定模拟中性能最佳的实现）。这会带来大量虚函数调用的开销，而且将来很可能会改变。与此同时，除了`BodyVector` （内部是一个 `vector<shared_ptr<Body> >`，外加一些相关方法）之外，所有 BodyContainer 的实现都已移除，因为它的速度最快。

插入/删除

刚体插入通常用于文件生成器：

```
shared_ptr<Body> body(new Body);
// … (body setup)
scene->bodies->insert(body); // assigns the id
```

刚体很少被删除：

```
scene->bodies->erase(id);
```

警告： 由于无法保证互斥，切勿在并行区段中插入/删除刚体，除非你明确确保不会发生并发访问。

迭代

可以使用 `for(const auto& …… : …… )` C++ 语法遍历容器：

```
for(const auto& b : *scene->bodies){
   if(!b) continue;                    // skip deleted bodies, nullptr-check
   /* do something here */
}
```

同一个循环也可以通过显式使用类型 `const shared_ptr<Body>&` 来编写 y:

```
for(const shared_ptr<Body>& b : *scene->bodies){
   if(!b) continue;                    // skip deleted bodies, nullptr-check
   /* do something here */
}
```

警告： 先前使用的宏 FOREACH 现已弃用。

请注意以下几点：

1. 始终使用 const shared_ptr<Body>& （常量引用）；这样每次 shared_ptr 都不会增加或减少引用计数。
2. 注意跳过 NULL 刚体（if(!b) continue）：已删除的刚体会从容器中释放，但由于刚体 ID 必须保持不变，其位置会由一个空的 shared_-ptr<Body>() 对象占据，该对象可隐式转换为 false。

在 Python 中，BodyContainer 封装器也支持迭代；为方便起见（这与 c++ 迭代器不同），NULL 刚体会被静默跳过：

```
Yade [54]: O.bodies.append([Body(),Body(),Body()])
Out[54]: [0, 1, 2]

Yade [55]: O.bodies.erase(1)
Out[55]: True

Yade [56]: [b.id for b in O.bodies]
Out[56]: [0, 2]
```

使用 OpenMP 并行化的循环必须遍历整数区间（而不是使用迭代器）：

```
const long size=(long)bodies.size();       // store this value, since it doesn't change during␣
 →the loop
#pragma omp parallel for
for(long _id=0; _id<size; _id++){
   const shared_ptr<Body>& b(bodies[_id]);
   if(!b) continue;
   /* … */
}
```

InteractionContainer

相互作用存储在特殊容器中，并且每个相互作用都必须由一对标识符 (id1,id2) 唯一标识。

- O(1) 访问元素，
- 线性可寻址性（0...n 索引），
- 存储 shared_ptr，而非对象本身，
- 插入/移除时的互斥，
- 便于迭代全部相互作用，
- 处理对称性，即 interaction(id1,id2) interaction(id2,id1)

注意： 与 BodyContainer 一样，这里有一个“抽象”类 InteractionContainer，以及它的具体实现。目前只使用 InteractionVecMap 这一实现，其他实现均已移除。因此，为避免不必要的虚函数调用，抽象类 InteractionContainer 未来可能会被移除。

此外，还有一种蓝图，用于将相互作用存储在刚体内部，因为这样可以更快地获取某个特定刚体的全部相互作用（目前，这需要遍历所有相互作用）；在这种情况下，InteractionContainer 将不复存在。

插入/删除

创建新的相互作用并删除它们是一个棘手的问题，因为模拟中的许多元素都必须保持同步；具体工作流程详见交互处理。您几乎肯定不需要手动从容器中插入/删除相互作用；如果确实需要，请考虑以不同方式设计您的代码。

```
// both insertion and erase are internally protected by a mutex,
// and can be done from parallel sections safely
scene->interactions->insert(shared_ptr<Interaction>(new Interactions(id1,id2)));
scene->interactions->erase(id1,id2);
```

迭代

与 BodyContainer 一样，遍历相互作用时应使用 `for(const auto& …… : …… )` C++ 语法，也可以使用 `const shared_ptr<Interaction>&` 代替 `auto&`：

```
for(const shared_ptr<Interaction>& i : *scene->interactions){
   if(!i->isReal()) continue;
   /* … */
}
```

警告： 此前使用的宏 `FOREACH` 现已弃用。

再次提醒，请注意 `i`使用的是常量引用。检查 `if(!i->isReal())` 会筛选掉仅可能存在的相互作用，即两个刚体之间只有 *Bound* 重叠，而刚体本身尚未发生重叠。函数 `i->isReal()` 等价于 `i->geom && i->phys`。详情请参阅交互处理。

在某些情况下，例如需要整数索引的 OpenMP 循环（OpenMP >= 3.0 也允许使用随机访问迭代器进行并行化），你需要改为遍历相互作用索引：

```
int nIntr=(int)scene->interactions->size(); // hoist container size
#pragma omp parallel for
for(int j=0; j<nIntr; j++){
   const shared_ptr<Interaction>& i=(*scene->interactions)[j];
   if(!i->isReal()) continue;
   /* … */
}
```

ForceContainer

ForceContainer 用于存储“广义力”，即每个刚体的力、力矩、（显式）位移和旋转。

在每个计算步中，通常会有 3 个与力有关的阶段：

1. 将力重置为零（通常由 *ForceResetter* 引擎完成）
2. 从并行区段增量累加力（求解相互作用——来自 *LawFunctor*）
3. 按顺序读取每个刚体的绝对力值：将来自不同相互作用的力相加，得到作用于该刚体的总力（*NewtonIntegrator*，以及其他读取力的各种引擎）

这种情况需要采用特殊设计，以实现快速并行写访问：

- 每个线程都有自己的存储空间（可按需清零），并且只写入自己的存储空间；这样可以避免并发问题。每个线程通过 OpenMP 运行时提供的 omp_get_thread_num() 函数来识别自身。

- 读取绝对值之前，必须先同步容器，即将全部线程中的值求和并分别存储。这是一项相对耗时的操作，我们提供了 Force-Container::syncCount，您可以查看它，以获取同步的累计次数，并将其与步数进行比较。理想情况下，ForceContainer 在每一步只同步一次。

- 每当读取或写入当前范围之外的元素时，容器都会调整尺寸（此时读取操作会返回零向量）；这样就无需跟踪刚体数量，但也存在潜在风险（例如 `scene->forces.getForce(1000000000)`，这可能会耗尽您的内存）。与c++不同，Python 会检查给定标识符是否超出刚体数量范围。

```
// resetting forces (inside ForceResetter)
scene->forces.reset()

// in a parallel section
scene->forces.addForce(id,force); // add force

// container is not synced after we wrote to it, sync before reading
scene->forces.sync();const Vector3r& f=scene->forces.getForce(id);
```

从 Python 读取值时，会自动进行同步：

```
Yade [57]: O.bodies.append(Body())
Out[57]: 3

Yade [58]: O.forces.addF(0,Vector3(1,2,3))

Yade [59]: O.forces.f(0)
Out[59]: Vector3(1,2,3)

Yade [60]: O.forces.f(100)
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~/yade/lib/x86_64-linux-gnu/yadeflip/py/yade/__init__.py in <module>
----> 1 O.forces.f(100)

IndexError: Body id out of range.
```

交互处理

创建和移除相互作用是一个相当微妙的话题，需要多个组件协同工作，才能使整体行为符合预期。

从术语上讲，我们区分

潜在相互作用，既没有几何体也没有物理。*Interaction.isReal* 可用于查询其状态（`Interaction::isReal()` 在 c++中）。
实际相互作用，同时具有几何体和物理。下文将讨论仅有几何体而没有物理属性的相互作用的可能性。

在模拟的每一步中，典型模拟都会对相互作用执行以下操作：

1. 碰撞器根据空间邻近性创建潜在相互作用。并非所有刚体对都可能进入相互作用；这一判断由 Collider::mayCollide 完成：
    - 团簇不能参与相互作用（只有其成员可以）

- 如果团簇成员属于同一个团簇，则它们之间不能发生相互作用
- 两个刚体的掩码按位与 结果 必须非零（即至少有一位共同置位）

2. 碰撞器会擦除已请求擦除的相互作用（见下文）。

3. *InteractionLoop* （通过 *IGeomDispatcher*）根据两个刚体的 *IGeomFunctor* 形状 组合 调用相应的函子（如果存在）。对于实际相互作用，该函子会更新关联的*IGeom*。对于潜在相互作用，该函子会返回

`false` 如果不存在几何重叠，则相互作用仍将仅为潜在相互作用

`true` 如果存在几何重叠，函子便会创建一个 *IGeom* 。

注意： 对于 实际 相互作用，函子 必须 返回 `true`，即使刚体之间不再存在空间重叠也是如此。如果你希望删除不存在几何重叠的相互作用，则必须在 *LawFunctor* 中执行此操作。

这种行为是有意为之，因为不同的 定律 有不同的要求，不过理想情况下，应使用数量相对较少且普遍适用的 几何函子。

注意： 如果没有适用于给定 形状组合的函子，相互作用将保持在势状态，且不会引发任何异常。

---

4. 对于实际相互作用（已存在或在上一步刚创建）， *InteractionLoop* （通过 *IPhys-Dispatcher*）根据两个刚体的 材料 组合调用相应的 ***IPhysFunctor*** 。函子 必须 更新（如果尚不存在，则创建）关联的 *IPhys* 实例。如果找不到合适的函子，则属于错误，并会抛出异常。

5. 对于实际相互作用， *InteractionLoop*（通过 *LawDispatcher*）根据相互作用的 ***IGeom*** 与 ***IPhys*** 的组合调用相应的 ***LawFunctor*** 。同样，如果找不到能够处理该相互作用的函子，则属于错误。

6. *LawDispatcher* 负责删除不再处于活动状态的相互作用（例如，对于非黏聚定律，如果刚体彼此相距过远；或者对于损伤模型，发生完全损伤时）。当 *LawFunctor* 返回 false 时，就会触发此操作。因此， *LawFunctor* 必须始终如一地返回结果，这一点至关重要。

此类相互作用不会立即删除，而是会重置为潜在状态。在碰撞器下次执行时，将调用 `InteractionContainer::conditionalyEraseNonReal` ，只有当包围盒不再重叠时，才会彻底删除相互作用；其余相互作用则会保留在潜在状态。

显式创建相互作用

仍可通过 *utils.createInteraction*显式创建相互作用，且不受任何空间条件限制。此函数会搜索当前引擎中的分派器并使用它们。调用 *IGeomFunctor* 时会传入`force` 参数，并要求它返回 `true` ，即使不存在空间重叠也是如此。

关联材料与状态类型

某些模型会在 状态信息中保存额外数据，该信息位于 *Body.state* 对象中，因此必须严格关联 材料与特定的状态（例如， *CpmMat* 与 *CpmState* 关联，而 ***CPM*** 状态更新器等引擎便假定采用这种组合）。

如果某个材料有此需求，就必须重写两个虚方法：

1. *Material.newAssocState*，该方法会返回一个对应类型的新状态对象。默认实现会返回状态本身。

2. *Material.stateTypeOk*，用于检查给定的状态对象是否属于相应的类型（在模拟开始时会对所有颗粒运行此检查）。

在 C++中，代码如下（适用于*CpmMat*）：

```
class CpmMat: public FrictMat {
   public:
      virtual shared_ptr<State> newAssocState() const { return shared_ptr<State>(new CpmState);
 → }
      virtual bool stateTypeOk(State* s) const { return (bool)dynamic_cast<CpmState*>(s); }
   /* ... */
};
```

这样，只需知道所需的刚体材料类型，就能通过*utils.sphere* 等函数构造对象，从而确保符合模型实现者的预期。材料类型，确保符合模型实现者的预期。

3.1.8 运行时结构

启动序列

Yade 的主程序是一个 Python 脚本，位于 core/main/main.py.in；构建系统会在将其复制到安装位置之前，替换该文件中的若干`${variables}` 。它会执行以下操作：

1. 处理命令行选项，并根据这些选项设置环境变量。2. 导入主 yade 模块（`import yade`），该模块位于 py/__init__.py.in。该模块会定位插件（在 `lib*.so` 安装 `lib` 目录中递归搜索文件）。yade.boot 模块用于设置临时目录等，并且最重要的是加载插件。3. 管理后续操作，例如运行命令行中指定的脚本、打开*qt.Controller* （如果需要），以及启动 `ipython` 提示符。

单例

有几个“全局变量”始终可以从++ 代码中访问；严格来说，它们是单例，即始终只有一个实例存在的类。其用途是让某些通用功能可从代码中的任意位置访问，而无需到处传递指向此类对象的指针。该实例在启动时创建，并且可以使用 `instance()`静态方法（例如 `Omega::instance().getScene()`）始终作为非 const 引用获取。

共有 3 个单例：

`ClassFactory` 从插件注册类，并能够根据类名字符串实例化类（该类必须继承自 `Factorable`）。不向 Python 暴露。

`Omega` 访问仿真；由于其重要性，值得单独设立一个截面。

`Logging` 处理所有命名日志记录器的日志过滤器，参见 日志详细级别。

Omega

该*Omega* 类负责处理所有仿真相关功能：加载/保存、运行、暂停。

在 Python 中，单例的包装类会被实例化[6] 为全局变量 `O`。为方便起见，*Omega* 被用作场景属性的代理：尽管可以实例化多个 `Scene` 对象于 C++中，*Omega* 始终代表当前场景。

[6] 需要注意，在 Python 中实例化 `Omega()` 只会实例化包装类，而不会实例化单例本身。

数据之间的对应关系是字面上的：*Omega.materials* 对应于 `Scene::materials` 当前场景的；同理，材料、刚体、相互作用、标签、单元、引擎、初始化器、杂项参数。
概述（部分）变量：

Python	c++
Omega.迭代	`Scene::iter`
Omega.时间步长	`Scene::dt`
Omega.时间	`Scene::time`
Omega.realtime	`Omega::getRealTime()`
Omega.stopAtIter	`Scene::stopAtIter`

`Omega` 在 c++ 中包含指向当前场景（`Omega::scene`，通过 `Omega::instance().getScene()` 获取）。使用 *Omega.switchScene*，可以将此指针切换为 `Omega::sceneAnother`，一个完全独立的模拟。例如，这在构建模拟时可能很有用（这也是此功能的设计初衷）：如果需要运行另一个模拟来动态生成球体堆积（即通过运行模拟来生成），就可以使用此功能。

引擎循环

运行模拟的过程包括循环遍历 引擎 并按顺序调用它们。此循环定义在 `Scene::moveToNextTimeStep` 函数中，位于 core/Scene.cpp。循环开始前，会调用 *O.initializers*；它们只运行一次。每次迭代时，引擎循环都会对 *O.engines* 执行以下操作：

1. 设置 `Engine::scene` 指针，使其指向当前的 `Scene`。
2. 调用 `Engine::isActivated()`；如果它返回 `false`，则跳过该引擎。
3. 调用 `Engine::action()`
4. 如果*O.timingEnabled*，则将*Engine::execTime* 增加自上次读数以来的时间差（上次读数是在前一个引擎运行后，或者如果这是第一个引擎，则是在循环开始前立即获取的）。将*Engine::execCount* 增加 1。

引擎处理完毕后，虚拟时间增加时间步长，并且迭代次数增加 1。

后台执行

引擎循环通常在后台线程中执行（由SimulationFlow 类处理），使前台线程可以自由地处理用户交互或运行 Python 脚本。后台线程由*O.run()* 和*O.pause()* 命令控制。可以使用*O.wait()*命令阻塞前台线程，直到循环结束。

无需启动额外线程即可运行单次迭代，只需使用*O.step()*。

3.1.9 Python 框架

封装 c++ 类

每个继承自可序列化对象的类都会自动向 Python 暴露，并可访问其（已注册的）属性。这是通过*YADE_CLASS_BASE_DOC_* 宏系列*实现的。类工厂中注册的所有类都会在 `Omega::buildDynlibDatabase`中默认构造。随后，每个可序列化类都会调用 `Serializable::pyRegisterClass` 虚方法，将类封装器注入（最初为空的） `yade.wrapper` 模块中。 `pyRegisterClass` 由 `YADE_CLASS_BASE_DOC` 定义，并且了解类、基类、文档字符串和属性；这些内容随后都会出现在 boost::python 类定义中。

包装类定义了一个特殊构造函数，它接受与类属性对应的关键字参数；因此，以下两种写法是相同的：

```
Yade [61]: f1=ForceEngine()

Yade [62]: f1.ids=[0,4,5]

Yade [63]: f1.force=Vector3(0,-1,-2)
```

and

```
Yade [64]: f2=ForceEngine(ids=[0,4,5],force=Vector3(0,-1,-2))

Yade [65]: print(f1.dict())
{'force': Vector3(0,-1,-2), 'ids': [0, 4, 5], 'dead': False, 'ompThreads': -1, 'label': ''}

Yade [66]: print(f2.dict())
{'force': Vector3(0,-1,-2), 'ids': [0, 4, 5], 'dead': False, 'ompThreads': -1, 'label': ''}
```

包装类还继承了 可序列化对象 提供的几个特殊虚方法： *dict()* 返回所有已注册类属性的字典（如上所示），*clone()* 返回实例的副本（通过复制属性值），*updateAttrs()* 和 *updateExistingAttrs()* 从给定字典中赋值属性（前者会因遇到未知属性而抛出异常，后者则不会）。此外，*pyDictCustom()* 也已在上一节中说明。

只读属性 `name` 封装了 c++ 方法 `getClassName()`，该方法以字符串形式返回类名。（由于 c++ 类和包装类的名称始终相同，因此使用 `__class__` 及其属性 `__name__` 获取 Python 类型会得到相同的值。）

```
Yade [67]: s=Sphere()

Yade [68]: s.__class__.__name__
Out[68]: 'Sphere'
```

在 Python 中对 c++类型进行子类化

在某些（少见的）情况下，从包装的 c 类派生新类可能会很有用++，并使用纯 Python 类型来实现。这可以在 *yade.pack* 模块 模块中看到：*Predicate* 是 c++ 基类；由该类派生出若干 c++ 类（例如 *inGtsSurface*），也派生出 Python 类（例如简单的 *inSpace* 谓词）。 `inSpace`派生自 Python 类 `Predicate`；不过，它并非 c++ `Predicate` 类的直接包装器，因为虚方法无法正常工作。

`boost::python` 为此类情况提供专用 `boost::python::wrapper` 模板，其中每个可重写虚方法都必须显式声明，并请求在存在 Python 重写版本时调用该版本。更多详情请参阅 可重写虚函数 。

从 C++ 调用 Python 代码时，调用线程必须持有 Python“全局解释器锁”（GIL）。初始化脚本以及运行一次调用 `O.step()`的迭代时，运行线程与 Python 使用的是同一线程，因此无需添加额外代码。另一方面，使用 `O.run()` 在模拟循环中调用 Python 代码时，线程必须获取该锁，否则会发生段错误。完整示例请参阅 *pyGenericPotential* () 的实现。

引用计数

Python 内部对其所有对象都使用引用计数，但普通用户通常看不到这一机制。如果直接使用 Python/CAPI 和 `Py_INCREF` 等函数，则必须显式处理引用计数。

`boost::python` 在 Yade 中使用的对象很幸运地由内部处理引用计数。此外，它还会 自动集成 引用计数，适用于 `shared_ptr` 和 Python 对象，只要类 `A` 被包装为`boost::python::class_<A, shared_ptr<A>>`。由于 全部 Yade 类都通过 *YADE_CLASS_-BASE_DOC_* 宏系列进行包装* ，因此返回 `shared_ptr<…>` 对象是将对象从 c 传递到++ Python 的首选方式。

返回 `shared_ptr` 效率高得多，因为只需返回一个指针，并在内部增加引用计数。在 Python 中修改对象会修改 c++ 中的同一个对象，反之亦然。这样还可以确保只要该 c++ 对象仍在 Python 中的某处使用，就不会被删除，从而避免一个（重要的）崩溃源。

自定义转换器

当对象从 c++ 传递到 Python，或反向传递时，

1. 类型是可在 c++ 和 Python 之间透明传递的基本类型（int、bool、std::string 等） 2. 类型由 boost::python 封装（例如 Yade 类、 `Vector3` 等），此时返回的是封装后的对象；[7]

其他类，包括模板容器，例如 `std::vector` ，必须单独编写自定义转换器。其中一些已在 py/wrapper/customConverters.cpp 中提供，尤其是 Python（同质，即所有元素类型相同）序列与对应类型的 c++ `std::vector`之间的转换；请查看该源文件，以添加自己的转换器或获取灵感。

对象跨越 c++/Python 边界时，boost::python 的全局“转换器注册表”会被搜索，以查找能够在相应 c++ 和 Python 类型之间执行转换的类。“转换器注册表”由整个程序实例共用：无需在每个脚本中注册转换器（例如通过导入 `_customConverters`），因为 Yade 已在启动时完成了注册。

注意：自定义转换器仅适用于按值传递给 Python 的值（而非“按引用”传递的值）：某些使用 *YADE_CLASS_BASE_DOC_** 宏族定义的属性是按值传递的，但如果你自行定义，请务必阅读并理解 为什么找不到我的自动转换为 Python 的转换器? 。

简而言之， `def_readwrite` 和 `def_readonly` 的默认行为是返回底层 c++对象的引用，从而避免对其执行转换。因此，必须显式设置返回值策略 `return_by_value` ，并使用稍微复杂一些的 `add_property` 语法，如所引用页面中所述。

目前，此缺陷已在文件 lib/serialization/PyClassCustom.hpp 中的 `.def_readonly(…)` 函数中得到解决。之后如有需要，还可以进一步改进。

3.1.10 添加新的 Python/C++模块

模块放置在 `py/` 目录中，模块的 `C++` 部分以下划线_开头。添加新模块的步骤如下：

1. 创建新文件：

   1. `yourNewModule.py` 文件，格式如下。2. `_yourNewModule.cpp` 文件，格式如下，如果模块的部分内容将使用 `C++`编写。

2. 在以下两个位置更新模块重定向映射：

   1. `mods`在 doc/sphinx/yadeSphinx.py 中。2. `moduleMap` 在 doc/sphinx/conf.py 中，如果新模块有 `C++` 部分（希望很快就能删除这份重复数据）。

[7]创建类封装器时，会自动注册包装类。不过，如果包装类继承自另一个包装类（并且使用 `boost::python::bases` 模板声明了这种依赖关系，Yade 的类会自动采用这种方式），则必须先注册父类，再注册派生类。（这通过 `Omega::buildDynlibDatabase`中的循环来处理，该循环会反复遍历各个类，跳过注册失败的类，直到它们全部成功注册）数学类（三维向量、Matrix3、四元数）通过*minieigenHP*进行包装。详情请参阅高精度文档。

3. 将 C++ 文件添加到 py/CMakeLists.txt 如下所示。

4. 修改 CMakeLists.txt ，但仅当该文件依赖 CMake 编译变量时才这样做，例如像这样。随后，文件需要添加扩展名 .in ，并放在两个位置：

    1. 从以下内容生成文件的 CMake 命令 .in 输入：像这样。2. 安装该文件的 CMake 命令：像这样。

提示：关于 yourNewModule.py.in （或 _yourNewModule.cpp.in）的最后一步仅在极少数情况下才需要，这里提及只是为了完整起见。

Python 与 C++ 之间边界的调试

在正常使用过程中，所有 C++ 异常都会连同相关的完整信息一起传播回 python 接口。唯一可能例外的情况是在执行 import module 中的 python 脚本时。导入新模块时，可能会发生一些难以理解的错误，例如： initialization of module raised unreported exception。这些 unreported exceptions 错误会在以下情况下发生： C++ 代码在其中执行了 python 代码（这称为嵌入），而该 python 代码抛出了异常。处理这种情况的正确方法是将整个模块声明放在 try {} catch(...) {} 块中。也可以处理特定异常（其他 catch 块示例请见此处），但通常的解决方法是明确告知 python 导入此模块失败。在这个 catch 块中，可以执行 PyErr_Print(); 命令查看问题所在，并将异常传播回 Python；不过，在执行 import module 命令期间，只有 SystemError Python 异常能够传递出来。因此， catch(...) 之后的块 BOOST_PYTHON_MODULE(_yourNewModule)应如下所示：

```
#include <lib/base/Logging.hpp>

CREATE_CPP_LOCAL_LOGGER("_yourNewModule.cpp");

BOOST_PYTHON_MODULE(_yourNewModule)
try {
        py::def("foo", foo, R"""(
The description of function foo().

:param arg1: description of first argument
:param arg2: description of second argument
:type arg1: type description
:type arg2: type description
:return: return description
:rtype: the return type description

Example usage of foo:

.. ipython::

   In [1]: from yade.yourNewModule import *

   In [1]: foo()

.. note:: Notes, hints and warnings about how to use foo().


        )""");

} catch (...) {
        LOG_FATAL("Importing this module caused an exception and this module is in an␣
 →inconsistent state now.");
```

（下页继续）

（续前页）

```
        PyErr_Print();
        PyErr_SetString(PyExc_SystemError, __FILE__);
        boost::python::handle_exception();
        throw;
}
```

注意： 请注意， _yourNewModule 中的内容必须与 BOOST_PYTHON_MODULE(…)中的内容一致，也就是与 .cpp 文件的文件名一致。

关于如何处理 Python 异常的更多阅读资料：

1. boost::python 参考手册中的示例。2. boost::python 教程中的示例。3. 当 PyErr_Print(); 不够用时。

3.1.11 维护兼容性

在 Yade 开发过程中，我们发现兼容性是用户非常重视的需求。兼容性涉及 Python 脚本，而非 以 XML 或旧版 C++ 代码保存的模拟。

重命名类

脚本 scripts/rename-class.py用于重命名 c++ 代码中的类。它接受 2 个参数（旧名称和新名称），必须从源代码顶层目录运行：

```
$ scripts/rename-class.py OldClassName NewClassNameReplaced 4 occurences, moved 0 files and 0 directories
Update python scripts (if wanted) by running: perl -pi -e 's/\bOldClassName\b/NewClassName/g'␣
→`ls **/*.py |grep -v py/system.py`
```

这会产生以下效果：

1. 如果 文件或目录的基本文件名为 OldClassName （加上扩展名），则会使用 bz 将其重命名 r.
2. A所有完整单词 OldClassName 都会在 c++ sourc 中替换为 NewClassName es.
3. 在 py/system.py中添加了一条记录，其中包含已弃用类名的映射。在 Yade 启动时，将创建带有 OldClassName 的代理类；实例化该类时会发出 DeprecationWarning ，告知你应该使用的新名称；它会创建 NewClassName 的实例，因此不会影响脚本的正常运行：

```
Yade [3]: SimpleViscoelasticMat()
/usr/local/lib/yade-trunk/py/yade/__init__.py:1: DeprecationWarning: Class␣
→`SimpleViscoelasticMat' was renamed to (or replaced by) `ViscElMat', update your code!␣
→(you can run 'yade --update script.py' to do that automatically)
->  [3]: <ViscElMat instance at 0x2d06770>
```

正如刚才所说，你可以在自己提供的脚本中运行 yade --update ，将所有旧名称替换为新名称：

```
$ yade-trunk --update script1.py some/where/script2.py
```

这样你就可以自由地为类取一个描述性强、直观易懂的类名。

重命名类属性

重命名类属性通过以下方式处理：++ 代码。你可以选择在访问旧属性时仅发出警告（并给出新名称），也可以在此基础上同时抛出异常，二者都可以附带说明。详情请参阅 deprec 参数*YADE_CLASS_BASE_DOC_** 宏系列。

3.2 GitLab 上的 Yade

3.2.1 快速检出（只读）

无需在 GitLab 上注册，即可通过一条命令获取源代码。此方式无法与远程仓库进行相互作用，您只能以只读方式访问：

```
git clone --recurse-submodules https://gitlab.com/yade-dev/trunk.git
```

3.2.2 GitLab 上的分支

最常用的命令列在以下各节中。更多详情，请参阅这些 Git 指南 :

1. ProGit 在线书籍，2. Git 设置指南，3. Git“选择你自己的冒险”，4. 解决冲突指南。

设置

1. 在 gitlab.com 注册
2. 将你的 SSH 密钥 添加到 GitLab
3. 通过终端设置用户名和电子邮件

   ```
   git config --global user.name "Firstname Lastname"
   git config --global user.email "your_email@youremail.com"
   ```

   你可以使用 `git config --list` 查看这些设置。
4. 若要派生代码仓库（可选），请点击GitLab 页面上的“派生”按钮，同时派生YADEdata 文件。

   注意：默认情况下，GitLab 会尝试编译派生的代码仓库；如果你的账户没有关联运行器，编译就会失败。若要避免收到失败通知，请前往代码仓库设置（左侧面板底部->常规->权限），关闭流水线。
5. 通过终端设置本地代码仓库。参数 `--recurse-submodules` 用于确保 `./data` 目录中填充了来自 yade-data 的最新数据（路径相对于你的 GitLab 个人资料）：

   ```
   git clone --recurse-submodules git@gitlab.com:username/trunk.git
   ```

   这会创建一个名为 trunk 的新文件夹，其中包含全部代码（请确保将 username 替换为你的 GitLab 用户名）。如果你已经有一个包含 `./data` 目录的克隆的 yade 代码仓库，则可以使用以下 command 填充现有代码仓库：

```
git submodule update --init --recursive
```

6. 配置远程仓库

```
cd to/newly/created/folder
git remote add upstream git@gitlab.com:yade-dev/trunk.git
git remote update
```

现在，你的“trunk”文件夹已关联到两个托管在 gitlab.com 上的远程仓库：一个是 yade-dev 的原始 trunk（在最后一条命令执行后称为“上游”），另一个是位于你个人账户中的派生仓库（称为“原点”，且默认始终配置）。通过下文介绍的适当命令，你可以更新代码以纳入他人提交的更改，也可以提交自己的更改，供他人获取。

实际上，并非一定要在个人账户下保留一个派生仓库。不过，我们建议这样做，且下文假设你已完成上述步骤。

旧版本

如果你　如果你想使用或编译未打标签的旧版 Yade，可以创建　te
自己的（　本地）分支，对应于相应的每日构建版本。详情请参见 此处 。

提交与更新

检查更改

在本地仓库中更改源代码后，可以先使用一些命令检查更改。对于“diff”命令，直接从“status”命令的输出中复制路径会比较方便，而不必手动输入已修改文件的路径。

```
git status
git diff path/to/modified/file.cpp
```

推送更改

根据要推送到的远程仓库，选择以下一种方法　.

1. 推送到 yade-dev

无法通过直接推送将更改合并到 yade-dev 的 master 分支，只能通过合并请求（见下文）进行合并。不过，属于该组的成员可以将更改推送到 yade-dev 仓库的新分支。这也是目前在实际合并之前，通过 GitLab 持续集成流水线测试合并请求的唯一方法。要推送到新的 yade-dev/分支：目前在实际合并之前，通过 GitLab 持续集成流水线测试合并请求的唯一方法。要推送到新的 yade-dev/分支：

```
git branch localBranch
git checkout localBranch
git add path/to/new/file.cpp  #Version a newly created file
git commit path/to/new_or_modified/file.cpp -m 'Commit message'  #stage (register) change␣
→in the local repository
git pull --rebase upstream master #get updated version of sources from yade-dev repo and␣
→apply your commits on the top of them
git push upstream localBranch:newlyCreatedBranch #Push all commits to a new remote branch.
```

前两行是可选的；如果省略，提交将进入默认分支，称为“主进程”。最后一个 command 中，`localBranch` 是你正在使用的本地分支名称（可能是 `master`），而 `newlyCreatedBranch` 将是该分支在远程上的名称。请尽可能选择一个描述性强的名称（例如“fixBug457895”）。

注意： 如果你在使用 command `git pull --rebase upstream master`时遇到任何问题，始终可以还原，或者更好的做法是解决冲突。

2. 推送到个人仓库

   之后按照前面的步骤通过终端进行提交后，应将“localBranch”替换为一个有意义的名称：

```
git branch localBranch
git checkout localBranch
git add path/to/new/file.cpp  #Version a newly created file
git commit path/to/new_or_modified/file.cpp -m 'Commit message'  #stage (register) change␣
↪in the local repository
git push  #Push all commits to the remote branch
```

   更改将被推送到您的个人 fork。

更新

您可能希望获取其他人完成的更改，以使本地代码仓库和远程仓库与上游保持同步：

```
git pull --rebase upstream master #Pull new updates from the upstream to your branch. Eq. of
↪"bzr update",  updating the local branch from the upstream yade-dev/trunk/master
git push  #Merge changes from upstream into your gitlab repo (origin)
```

如果您有本地未提交的更改，此操作将返回错误。要在保留这些更改的同时进行更新，可以使用“暂存”：

```
git stash #backup and hide changes
git pull --rebase upstream master
git push
git stash pop #restore backed up changes
```

自动变基

我们建议使用“变基”，以避免每次提交/拉取/推送循环后日志变得混乱。设置自动变基会比较方便，这样就不必每次都在上述命令中添加：

```
git config --global branch.autosetuprebase always
```

现在，你的文件 `~/.gitconfig` 应包含：

```
[branch]
  autosetuprebase = always
```

也请检查本地主干文件夹中的 `.git/config` 文件（变基 = true）：

```
[remote "origin"]
  url = git@gitlab.com:yade-dev/trunk.git
  fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
  remote = origin
  merge = refs/heads/master
  rebase = true
```

拉取变基后的分支

如果其他人在 GitLab 服务器上对你本地正在使用的分支进行了变基，command`git pull` 可能会提示分支已分叉，并拒绝执行操作。这种情况下，可以使用以下 command：

```
git pull --rebase upstream branchName
```

这将使你的本地分支历史记录与 GitLab 服务器上的分支历史记录保持一致。

如果你担心弄乱本地分支，可以随时使用以下 command 创建该分支的副本：

```
git branch backupCopyName
```

如果你忘记创建备份副本，但又想返回，那么仍然可以创建一个副本，并使用以下 command 返回：

```
git reset --merge ORIG_HEAD
```

`ORIG_HEAD` 会在执行可能有危险的操作（合并、变基等）之前备份 `HEAD` 的位置。

推荐阅读一篇关于解决冲突的教程。

Not例如： 如果你不知道如何解决 Git 问题，可以试试Git 版“选择自己的冒险”。

3.2.3 合并请求

yade-dev 的成员

如果你已经测试过更改并准备好将其合并到 yade-dev 的 master 分支，就需要通过 gitlab.com 界面创建一个“合并请求”（MR）（参见代码仓库网页顶部的“+”按钮）。将源分支和目标分支分别设为 yade-dev/trunk/newlyCreatedBranch 和 yade-dev/trunk/master。MR 会触发流水线，其中包括编译、运行回归测试和生成文档（新构建的文档可通过设置->页面访问，也可以在 `doc_18_04` 流水线中的构建作业右侧面板的“作业产物”中点击“浏览”按钮，然后导航到路径 `Artifacts/install/share/doc`）。如果整个流水线成功，便可以将合并请求合并到 master 分支。

注意： 如果要将其他 yade-dev 分支合并请求到 yade-dev 的主进程，流水线将使用附属于 yade-dev 的组运行器（组运行器由3SR、UMS Gricad 和格但斯克理工大学慷慨提供）。

新开发者

欢迎！刚开始时，在自己的 GitLab 个人资料中使用 YADE 的本地派生版本会非常方便。当你确信所做的更改已准备好合并到 YADE 官方发行版中时，请按以下方式创建合并请求（MR）：

1. 请确保您的工作位于单独的分支上，而不是 `master` 分支。您可以使用命令 `git checkout -b myNewFeature` 将您的分支“复制”到另一个分支。请确保与主进程分支相比，改动量不大。如果代码改进较多，最好将其拆分成几个较小的合并请求。这样我们就能更快地检查并合并。

2. 使用类似以下的命令，将您的分支推送到 GitLab 个人资料中的代码仓库：

```
git push --set-upstream origin myNewFeature
```

3. 你应该会看到类似这样的内容：

```
remote:remote: To create a merge request for myNewFeature, visit:
remote:   https://gitlab.com/myProfileName/trunk/-/merge_requests/new?merge_request
→%5Bsource_branch%5D=myNewFeature_remote:
```

4. 访问上面提到的链接时，你需要选择“更改分支”，并确保选中了正确的目标分支。通常应该是 `yade-dev/trunk:master`，因为这是官方的 YADE 代码仓库。

5. 填写标题和描述，然后点击页面底部的“创建合并请求”。

6. 审核合并请求后，我们可以点击它，在持续集成（CI）流水线中运行。出于安全原因，此流水线无法自动启动。流水线检查通过后，该请求将被合并。

或者，也可以通过以下方式根据你的提交创建补丁：

```
git format-patch origin  #create patch file in current folder)
```

并将其作为附件发送到开发者邮件列表（yade-dev@lists.launchpad.net）。无论采用哪种方式，在审核你的更改后，都会将其添加到主干。

当 拉取请求经过审核并获批后，你的更改就会合并到主干 k. 所有人 都可以通过 `git fetch` 获取这些更改。

3.2.4 推送指南

以下是向 `yade-dev/trunk` 推送的一般指南。

1. 在计算机上全局设置自动变基（只需设置一次，见上文），或至少在当前本地分支上设置。上游不接受未变基的拉取请求。这样做是为了保持历史记录线性，并避免产生合并提交。

2. 检查差异，确保不会提交垃圾代码（通常是散落在各处的“cout<<”），可在终端中使用：

```
git diff file1
```

或者使用你偏好的差异比较工具，例如 KDiff3：

```
git difftool -t kdiff3 file1
```

或者，也可以使用任意 Git 图形用户界面：gitg、git-cola.....

3. 有选择地提交：

```
git commit file1 file2 file3 -m "message" # is good
git commit -a -m "message"                # is bad. It is the best way to commit␣
→things that should not be commited
```

4. 确保使用最新版本，启动时运行：

```
git pull --rebase upstream master
```

5. 确保编译成功且回归测试通过：尝试 `yade --test` 和 `yade --check`。

非常感谢您对 Yade 的支持！

第4章

理论背景与扩展

4.1 离散元法公式

离散元法公式已在前面的第 2.1 章中介绍。离散元法公式作为全部离散元法计算的共同基础。

4.2 使用 Yade 和 OpenFOAM 进行 CFD-DEM 耦合模拟

FoamCoupling 引擎提供了一个框架，可使用开源有限体积求解器 OpenFOAM进行欧拉-拉格朗日流体-颗粒模拟。耦合依赖于 消息传递接口库（MPI）；由于 OpenFOAM 是并行求解器，求解器之间的通信也通过 MPI 消息实现。必须在编译时通过 ENABLE_MPI 标志启用

```
cmake -DCMAKE_INSTALL_PREFIX=/path/to/install /path/to/source -DENABLE_MPI=1
```

Yade 将颗粒信息（颗粒位置、速度等）发送到所有 OpenFOAM 进程。每个 OpenFOAM 进程都会在局部网格中搜索该颗粒；如果找到该颗粒，则使用颗粒位置处的流体速度（有两种可用的插值方法）和颗粒速度计算流体动力阻力和力矩。流体动力被发送到 Yade 进程，并添加到力容器中。颗粒流体动力的负值（插值回流体单元中心）被设为纳维-斯托克斯方程中的源项。 OpenFOAM 求解器 也必须安装，以便在 Yade 和 OpenFOAM 之间建立 MPI 连接。有关耦合方法的技术细节，请参见 [Kunhappan 2017]和 [Kunhappan2018]。

4.2.1 背景

在含颗粒多相流的标准欧拉-拉格朗日建模中，颗粒被视为点质量。本耦合中实现了两种方法：

1. 点力耦合
2. 基于体积分数的力耦合。

在这两种方法中，颗粒尺度的流动都无法解析，因此使用解析或经验流体动力模型来描述流体-颗粒相互作用。为了在流体网格上准确解析颗粒体积分数和流体动力，颗粒尺寸必须小于流体单元尺寸。

点力耦合（*icoFoamYade*）

在点力耦合中，假设颗粒尺寸小于最小的流体长度尺度，因此颗粒雷诺数为 $Re_p < 1.0$。颗粒雷诺数定义为惯性力与黏性力之比。对于球体，相应的长度尺度为直径，因此：

$$Re_p = \frac{\rho_f |\mathbf{U}_r| d_p}{\mu} \tag{4.1}$$

其中，在（4.1）中，ρ_f 是流体密度，$|\mathbf{U}_r|$ 是颗粒与流体之间相对速度的范数，d_p 是颗粒直径，而 μ 是流体动力黏度。除了雷诺数之外，还定义了另一个表征颗粒因质量而具有的惯性的无量纲数，称为斯托克斯数：

$$St_k = \frac{\tau_p |\mathbf{U}_f|}{d_p} \tag{4.2}$$

其中，在方程（4.2）中，τ_p 是颗粒弛豫时间，其定义为：

$$\tau_p = \frac{\rho_p d_p^2}{18\mu}$$

对于 $Re_p < 1$ 和 $St_k < 1$，作用在颗粒上的流体动力可表示为点力。该力使用斯托克斯阻力公式计算：

$$\mathbf{F}_h = 3\pi\mu d_p (U_f - U_p) \tag{4.3}$$

由（4.3）得到的力作用于颗粒上，也作用于流体侧（颗粒所在的单元中）；该流体动力被表示为体积力：

$$\mathbf{f}_h = \frac{-\mathbf{F}_h}{V_c \rho_f} \tag{4.4}$$

其中，在方程（4.4）中，V_c 是单元的体积，ρ_f 是流体密度。因此，组合系统的纳维-斯托克斯方程为：

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) = -\frac{\nabla p}{\rho} + \nabla \bar{\bar{\tau}} + \mathbf{f}_h \tag{4.5}$$

以及连续性方程：

$$\nabla \cdot \mathbf{U} = 0 \tag{4.6}$$

体积平均耦合（*pimpleFoamYade*）

在体积平均耦合中，考虑了颗粒体积分数的影响。纳维-斯托克斯方程的形式如下：

$$\frac{\partial (\varepsilon_f \mathbf{U}_f)}{\partial t} + \nabla \cdot (\varepsilon_f \mathbf{U}_f \mathbf{U}_f) = -\frac{\nabla p}{\rho} + \varepsilon_f \nabla \bar{\bar{\tau}} - K (U_f - U_p) + \mathbf{S}_u + \varepsilon_f \mathbf{g} \tag{4.7}$$

以及连续性方程：

$$\frac{\partial \varepsilon_f}{\partial t} + \nabla \cdot (\varepsilon_f \mathbf{u}_f) = 0 \tag{4.8}$$

其中，方程（4.7）和（4.8） ε_f 中，表示流体体积分数。请注意，我们并不直接求解 ε_f，而是根据局部颗粒体积分数 ε_s， $\varepsilon_f = 1 - \varepsilon_s$ 求得。 K 是颗粒阻力参数， $\mathbf{u}_f$ 和 $\mathbf{u}_p$ 分别是流体速度和颗粒速度。 $\mathbf{S}_u$ 表示显式源项，其中包含其他流体动力的影响，例如阿基米德力/环境力、附加质量力等。这些力的具体形式将在本节后续部分介绍。

欧拉量和拉格朗日量的插值与平均基于高斯包络 $G_\star$。在这种方法中，颗粒的影响会被其所在单元的相邻单元“感知”。设 $\mathbf{x}_c$ 和 $\mathbf{x}_p$ 分别为流体单元中心和颗粒位置，则高斯滤波器 $G_\star(\mathbf{x}_c - \mathbf{x}_p)$ 定义为：

$$G_\star(\mathbf{x}_c - \mathbf{x}_p) = \left(2\pi\sigma^2\right)^{\frac{3}{2}} \exp\left(-\frac{\|\mathbf{x}_c - \mathbf{x}_p\|^2}{2\sigma^2}\right) \tag{4.9}$$

其中， σ为筛选器的标准差，定义如下：

$$\sigma = \delta / \left(2\sqrt{2\ln 2}\right) \tag{4.10}$$

其中，在方程（4.10）中， δ 为截断范围（目前设为 $3\Delta x$，其中 Δx 为流体单元尺寸），并遵循以下规则：

$$G_\star(\|\mathbf{x}_c - \mathbf{x}_p\| = \delta/2) = \frac{1}{2} G_\star(\|x_c - x_p\| = 0)$$

流体单元 c 的颗粒体积分数 $\varepsilon_{s,c}$ 计算如下：

$$\varepsilon_{s,c} = \frac{\sum_{i=1}^{N_p} V_{p,i} G_{\star(i,c)}}{V_c} \tag{4.11}$$

其中，在（4.11）中， N_p 为单元 c上的颗粒体积贡献数量， $G_{\star(i,c)}$ 为根据（4.9）得到的高斯权重， $V_{p,i} G_{\star(i,c)}$ 构成单个颗粒体积贡献。 V_c 为流体单元体积， $\varepsilon_f + \varepsilon_s = 1$

使用以下表达式，将欧拉量 φ 从网格（单元格）平均并插值到颗粒位置：

$$\widetilde{\varphi} = \sum_{i=1}^{N_c} \varphi_i G_{\star(i,p)} \tag{4.12}$$

流体动力

在方程（4.7）中，项 K 是阻力参数。在当前实现中， K 基于 Schiller-Naumann 阻力定律，其表达式如下：

$$K = \frac{3}{4} C_d \frac{\rho_f}{d_p} \left|\left|\widetilde{\mathbf{u}}_f - \mathbf{u}_p\right|\right| \varepsilon_f^{-h_{exp}} \tag{4.13}$$

在方程（4.13）中，ρ_f 是流体密度，d_p 是颗粒直径，h_{exp} 定义为“受阻系数”，其值设为 $h_{exp} = 2.65$。阻力系数 C_d 适用于颗粒雷诺数不超过 $Re_p < 1000$的情况。其表达式 C_d 如下：

$$C_d = \frac{24}{Re_p}\left(1 + 0.15Re_p^{0.687}\right) \tag{4.14}$$

颗粒所受流体动力阻力的表达式为：

$$F_{drag} = V_p K(\widetilde{U}_f - U_p)$$

在流体方程中，阻力参数（$-K$）的负值根据方程（4.11分配回网格。由于阻力包含对流体速度的非线性依赖 U_f，因此在流体求解器中将此项设为隐式源项。

颗粒所受的阿基米德力/环境力计算如下：

$$F_{by} = \left(\widetilde{-\nabla p} + \widetilde{\nabla \bar{\tau}}\right) V_p \tag{4.15}$$

其中，在（4.15）中，$\widetilde{\nabla p}$ 是颗粒中心处的平均压力梯度，而$\widetilde{\nabla \bar{\tau}}$ 是颗粒位置处的平均黏性应力散度。

附加质量力：

$$F_{am} = C_m \left(\frac{D\widetilde{U_f}}{Dt} - \frac{dU_p}{dt}\right) V_p \tag{4.16}$$

其中在方程（4.16）中，$\frac{DU_f}{Dt}$是流体速度的物质导数。

因此，作用在颗粒上的净流体动力为：

$$F_{hyd} = F_{drag} + F_{by} + F_{am}$$

而在流体一侧，显式源项 $S_{u,\,c}$ 对于流体单元 c 表示为：

$$S_{u,c} = \frac{\sum_{i=1}^{N_p} -F_{hyd,i}\varepsilon_{s,c}G_{\star(i,c)}}{\rho_f V_c}$$

4.2.2 设置算例

在 Yade 中

在 Yade 一侧设置算例相当简单。用于描述 Yade 场景的 Python 脚本基于此方法。请确保在 Yade 和 OpenFOAM 中都设置了完全一致的墙面/周期性边界条件。颗粒不应离开流体域。如果颗粒“逃出”域，则会向日志文件打印/写入警告消息，并且模拟将中断。

此示例位于examples/openfoam/scriptYade.py，用于演示耦合。创建了指向 Yade 的符号链接，并在脚本中导入它。在实例化耦合引擎之前，通过调用 initMPI() 函数初始化 MPI 环境

```
initMPI()
fluidCoupling = FoamCoupling()
fluidCoupling.getRank()
```

将粒子编号列表和颗粒数量传递给耦合引擎

```
sphereIDs = [b.id for b in O.bodies if type(b.shape)==Sphere]
numparts = len(sphereIDs);

fluidCoupling.setNumParticles(numparts)
fluidCoupling.setIdList(sphereIDs)
fluidCoupling.isGaussianInterp = False
```

必须设置力／速度插值模式的类型。对于高斯包络插值，必须设置*isGaussianInterp* 标志，并且必须使用求解器*pimpleFoamYade* 。该引擎添加到时间步进器之后的 O.engines 中

```
O.engines = [
ForceResetter(), ......,
GlobalStiffnessTimeStepper,
fluidCoupling ..newton ]
```

子步进/数据交换间隔会根据时间步长之比 foamDt/yadeDt 自动设置（详情参见 *exchangeDeltaT* ）。

在 OpenFOAM 中

此 Git 代码仓库中提供了两种求解器。求解器 *icoFoamYade* 基于点力耦合方法，而求解器 *pimpleFoamYade* 基于体积平均耦合。它们分别基于现有的 icoFoam 和 pimpleFoam 求解器。可以使用任何 OpenFOAM 支持的网格；有关网格选项和网格划分的更多详情，请参见 此处。在本例中，网格使用 OpenFOAM 的 *blockMesh* 实用程序生成。算例按照常规的 OpenFOAM 方式设置，包含目录 *0*、*system* 和 *constant*

```
0/
  U                       ## velocity boundary conditions
  p                       ## pressure boundary conditions
  uSource                 ## source term bcs (usually set as calculated).

system/
  controlDict             ## simulation settings : start time,  end time,  delta T,  solution␣
→write control etc.
  blockMeshDict           ## mesh setup for using blockMesh utility : define coordinates of␣
→geometry and surfaces. (used for simple geometries -> cartesian mesh.)
  decomposeParDict        ## dictionary for setting domain decomposition, (in the present␣
→example scotch is used)
  fvSchemes               ## selection of finite volume schemes for calculations of␣
→divergence,  gradients and interpolations.
  fvSolution              ## linear solver selection,  setting of relaxation factors and␣
→tolerance criterion,

constant/
  polymesh/               ## mesh information,  generated by blockMesh or other mesh utils.
  transportProperties     ## set the fluid and particle properties. (just density of the␣
→particle)
```

注意：请始终将时间步长设为小于颗粒弛豫时间尺度，目前尚未自动计算该值！基于雷诺平均纳维–斯托克斯方程的湍流建模尚未实现。

不过，可以通过直接数值模拟（DNS）使用当前的公式进行完全解析的湍流流动模拟。目前尚不支持动态/移动网格问题。（如果你有兴趣实现任何新功能，请告诉我。）

准备模拟时，请按照以下步骤操作：

```
blockMesh          ## generate the mesh
decomposePar       ## decompose the mesh
```

可以使用任何OpenFOAM 支持的网格。目前不支持动态网格。

执行

通过以下命令执行模拟：

```
mpiexec -n 1 python3 scriptYade.py : -n NUMPROCS icoFoamYade -parallel
```

下方的视频展示了编译和执行耦合计算流体力学-离散元模拟所涉及的步骤

4.2.3 后处理

ParaView 可用于可视化 Yade 求解结果（使用 VTKRecorder）和 OpenFOAM 解。要可视化流体求解结果，请创建一个空文件 $name.foam$，在 ParaView 中打开该文件，然后在流水线下方的属性部分中，将“重构算例”更改为“分解算例”；或者也可以使用重构算例本身，但需先运行$reconstructPar$ 实用程序，不过这会耗费较多时间。

4.3 使用 Yade 和 Es- cript 进行有限元-离散元分层多尺度建模

作者：Ning Guo 和 Jidong Zhao

机构：香港科技大学

Escript 下载页面：https://launchpad.net/escript-finley

mpi4py 下载页面（可选，需要 MPI）：https://bitbucket.org/mpi4py/mpi4py

测试平台：搭载 Ubuntu 10.04、32 位的桌面电脑；搭载 Ubuntu 12.04、14.04、64 位的服务器；搭载 CentOS 6.2、6.5、64 位的簇；

4.3.1 简介

该代码基于两个开源软件包构建：用于离散元法模块的 Yade 和用于有限元法模块的 Escript。它实现了分层多尺度模型（FEMxDEM），用于模拟颗粒介质的边值问题（BVP）。有限元法用于离散问题域。有限元网格的每个高斯点都嵌入一个由离散元法模拟的代表性体积单元（RVE）堆积体，该堆积体会向有限元法返回局部材料本构响应。中等规模的问题通常涉及数百到数千个代表性体积单元，因此计算耗时很长。为此，代码通过在超级计算机上使用多进程，或在高性能计算集群上使用 mpi4py（需要 MPICH 或 Open MPI）来实现并行化。代码中的 MPI 实现仍处于试验阶段。mpipool.py 由 mpi4py 软件包的作者 Lisandro Dalcin 贡献。代码的用法请参阅示例。

4.3.2 有限元公式

注意： 本节及下一节摘自 [Guo2014]，旨在提供一些理论背景。欢迎使用有限元-离散元耦合的 Yade 用户改进以下两节内容。

在这种针对颗粒介质分层多尺度建模的 FEM/DEM 耦合框架中，首先将给定边值问题 Ω 的几何域离散为合适的有限元网格。经过有限元离散后，得到如下待求解的方程组：

$$\mathbf{K}\mathbf{u} = \mathbf{f}, \tag{4.17}$$

其中，$\mathbf{K}$ 为刚度矩阵，$\mathbf{u}$ 为有限元节点处的未知位移向量，而 $\mathbf{f}$ 为由施加的边界牵引力集总得到的节点力向量。对于典型的线弹性问题，$\mathbf{K}$可由弹性模量构造，方程（4.17）可直接求解。而对于涉及非线性的情况，例如颗粒介质，其中 $\mathbf{K}$ 取决于状态参数和加载历史，需要采用牛顿-拉夫森迭代法，并将刚度矩阵替换为切线矩阵 $\mathbf{K}_t$，该矩阵由切线算子组装而成：

$$\mathbf{K}_t = \int_\Omega \mathbf{B}^T \mathbf{D} \mathbf{B} dV, \tag{4.18}$$

其中，$\mathbf{B}$ 为变形矩阵（即形函数的梯度），$\mathbf{D}$ 为四阶切线算子张量 $\mathbf{D}$的矩阵形式。在每次牛顿-拉夫森迭代中，$\mathbf{K}_t$ 和内部应力 $\boldsymbol{\sigma}$ 都会更新，该迭代过程旨在最小化残余力 $\mathbf{R}$，以求得收敛解：

$$\mathbf{R} = \int_\Omega \mathbf{B}^T \boldsymbol{\sigma} dV - \mathbf{f}. \tag{4.19}$$

每个局部高斯积分点处的切线算子和应力张量是上述计算中的关键变量，需要在每次迭代和加载步之前进行评估。基于连续体的传统有限元法通常会对材料作出本构关系假设，并基于该本构假设推导切线矩阵和应力增量（例如，使用方程 $\mathbf{D}^{ep}$ （4.18）中的弹塑性模量 $\mathbf{K}_t$ 来组装并积分应力）。耦合有限元法/离散元法多尺度方法从每个高斯点处嵌入的离散元组装体中获取这两个物理量，从而避免了对唯象假设的需求。

4.3.3 多尺度求解过程

层级】【过程可概括为以下步骤 ：

1. 使用适当的有限元网格离散问题域，并为每个高斯点附加一个具有适当初始状态的离散元组装体。

2. 施加一个全局加载步，即通过有限元边界条件对 $\partial\Omega$ 施加。

   a) 确定每个代表性体积单元的当前切线算子。b) 使用方程（4.18）组装全局切线矩阵，并通过有限元法求解方程（ $\mathbf{u}$ ）获得位移试探解。c) 在有限元网格的每个高斯点处插值变形 $\nabla\mathbf{u}$，并将 $\nabla\mathbf{u}$ 作为离散元边界条件，对相应的代表性体积单元运行离散元模拟。d) 推导每个代表性体积单元更新后的总应力，并根据方程（4.19）计算有限元域的残差。e) 重复上述步骤 (a) 至 (d)，直至收敛，然后完成当前加载步。

3. 进入下一个加载步，并重复步骤 2。

在步骤 2(c) 中，根据有限元法求解结果插值得到用于离散元边界条件的变形 $\mathbf{u}$ 时，我们同时考虑无穷小应变 ε 和旋转 ω

$$\nabla \mathbf{u} = \underbrace{\frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^{\mathsf{T}})}_{\varepsilon} + \underbrace{\frac{1}{2}(\nabla \mathbf{u} - \nabla \mathbf{u}^{\mathsf{T}})}_{\omega} \tag{4.20}$$

T相应的代表性体积单元堆积体将根据此处规定的边界条件发生变形 .

另外，有必要对步骤 2(d) 中代表性体积单元的应力演化作几点说明。在传统有限元法中，为处理非线性材料响应，应力以增量方式更新。若假设小应变，当材料发生大变形时，增量应力–应变关系可能导致不准确的数值结果，因此需要采用适用于大变形的替代公式。这一问题在当前的分层框架中自然可以避免。在我们的框架中，每个高斯点处的离散元组装体都会记忆其过去的状态历史记录（例如压力水平、孔隙比和组构），并在每个加载步和迭代步中根据当前施加的边界条件（包括拉伸和旋转）进行求解。在每个加载步结束时，我们不采用增量应力更新方案，而是通过均质化直接根据求解后的离散元组装体推导出总真实应力（柯西应力），然后将其返回有限元求解器以进行全局求解。这样，我们无需借助其他客观应力度量来处理大变形问题。不过，我们注意到，仍然需要适当的应变度量，而且有限元网格不应发生严重畸变，否则就需要重新划分有限元域的网格。

求解过程的更详细描述见 [Guo2013], [Guo2014], [Guo2014b], [Guo2014c], [Guo2015]。

4.3.4 YADE 侧的工作

YADE 的版本至少应为 rev3682，在该版本中 Bruno 添加了 stringToScene 函数。在安装之前，我在源代码（“yade”子文件夹中）中添加了一些函数。但对于有限元-离散元耦合而言，只有一个函数（“./pkg/dem/Shop.cpp”中的“Shop::getStressAndTangent”）是必要的，该函数返回离散堆积体的应力张量和切线算子。前者使用 Love 公式进行均质化，后者则均质化为弹性模量。安装完成后，我们会得到可执行文件：yade-versionNo。随后，通过“ln yade-versionNo yadeimport.py”生成一个链接到该可执行文件的 .py 文件。该 .py 文件将作为 YADE 的包装库。之后，我们将通过“from yadeimport import *”将 YADE 的全部函数导入 Python 脚本中（见 simDEM.py 文件）。

打开一个 Python 终端。确保可以运行

```
import sys
sys.path.append('where you put yadeimport.py')
from yadeimport import *
Omega().load('your initial RVE packing, e.g. 0.yade.gz')
```

如果运行成功，你还应该能够运行

```
from simDEM import *
```

4.3.5 在 Escript 侧进行工作

没有特别的要求。但请确保这些模块可在 Python 中调用，这意味着 Escript 的主文件夹应位于 PYTHONPATH 和 LD_LIBRARY_PATH 中。这些模块在 msFEM*.py 中封装为一个类。

打开 Python 终端。确保你可以运行：

```
from esys.escript import *
from esys.escript.linearPDEs import LinearPDE
from esys.finley import Rectangle
```

（注意：当前实现使用 Escript。只要提供 Python 绑定，也可以替换为其他有限元法软件包，例如 FEniCS（http://fenicsproject.org）。但需要修改接口文件“ms- FEM*.py”。）

4.3.6 示例测试

完成步骤 1 和 2 后，便可以运行多尺度分析所需的全部脚本。初始代表性体积单元堆积体（默认名称为“0.yade.gz”）应由用户提供（例如，使用 YADE 准备固结堆积体）；问题初始化时，simDEM.py 会加载该堆积体。只要将相同的代表性体积单元堆积体分配给问题域中的所有高斯积分点，试样初始时就是均匀的。用户也可以为不同的高斯积分点指定不同的代表性体积单元，从而生成天然非均质的试样。

simDEM.py 始终是必需的，而单个试验只需要一个 msFEM*.py。例如，对于二维（三维）干燥试验，需要使用 msFEM2D.py（msFEM3D.py）；同样，对于水力-力学耦合问题（仅二维、饱和），则使用采用 u-p 格式的 msFEMup.py。默认使用多进程处理。若要尝试 MPI 并行化，请在主脚本中构造问题时设置 useMPI=True。下面列出了“example”子文件夹中的示例测试。注意：需要生成初始代表性体积单元堆积体（默认命名为 0.yade.gz）；例如，对于二维堆积体，可使用“example”子文件夹中的 prepareRVE.py 生成（三维堆积体也同理）。

1. **干燥密砂二维排水双轴压缩试验** (biaxialSmooth.py)注：试验描述和结果见 [Guo2014] 和 [Guo2014c]。

2. **干砂在平移模式下由刚性无摩擦墙支挡时的二维被动破坏** (retainingSmooth.py) 注：代表性体积单元堆积体采用滚动阻力模型（CohFrictMat）。试验描述和结果见 [Guo2015]。3. **采用 Gmsh 生成网格的二维半域基础沉降问题**(footing.py,footing.msh) 注：代表性体积单元堆积体采用滚动阻力模型（CohFrictMat）。Gmsh 生成了六节点三角形单元，每个单元包含三个高斯积分点。试验描述和结果见 [Guo2015]。

4. **使用 MPI 并行处理的干燥密砂三维排水常规三轴压缩试验** (triaxialRough.py) 注 *1*：该模拟非常耗时。在单节点上使用多进程（16 个进程，中央处理器频率为 2.0 GHz）需要 ~4.5 天。将 useMPI 开关设为 True（如示例脚本所示），并使用四个节点（80 个进程，中央处理器频率为 2.2 GHz）时，模拟耗时不到 24 小时。我们的 test 中加速比约为 4.4。注 *2*：使用 MPI 时，需要安装 mpi4py。MPI 实现可以是 MPICH 或 Open MPI。还应将文件“mpipool.py”放在主文件夹中。我们的 test 基于 openmpi-1.6.5。这项工作仍在进行中。试验描述和结果将在之后给出。

5. **采用 MPI 并行对饱和密实砂进行渗透率变化条件下的二维全局不排水双轴压缩 test** (undrained.py)注意： 这项工作仍在进行中。test 的描述和结果将在稍后呈现。

4.3.7 免责声明

本项目广泛使用并依赖上述提及的一些第三方软件包，并对其贡献表示感谢。欢迎使用和再分发代码。但本项目不提供任何保证；甚至不保证其适销性或适用于特定用途。

4.4 在Yade中模拟声发射

建议引用：

Caulk, R.（2018），《离散元方法的随机增强：非均质岩石中拉伸破裂的研究》。*Yade* 技术档案。数字对象标识符 10.5281/zenodo.1202039。下载全文

Caulk, Robert A.（2020），《使用离散元法模拟拉伸断裂过程中非均质岩石中的声发射》。开放岩土力学，第2卷，文章编号2，共19页。数字对象标识符：10.5802/ogeo.5。全文

4.4.1 摘要

本文简要介绍Yade中声发射（声发射）的模拟。Yade基于聚类应变能的声发射模型遵循以下文献提出的方法： [Hazzard2000] 和 [Hazzard2013]。关于Yade方法的验证，以及岩石非均质性对岩石拉伸破坏过程中声发射影响的考察，详见 [Caulk2018] 和 [Caulk2020]。

4.4.2 模型描述

通过假设每个断裂键（或断裂键簇）代表一个事件位置，来模拟数值声发射事件。此外，相关的系统应变能变化代表事件震级。键断裂后，对预定义空间半径（λ）内所有完整键的应变能 （E_i）求和：

$$E_i = \frac{1}{2}\left(\frac{F_n^2}{k_n} + \frac{F_s^2}{k_s}\right)$$

$$E_o = \sum_i^N E_i$$

其中 F_n、 F_s 和 k_n、 k_s 分别是破坏前相互作用的法向力和剪切力（N）以及刚度（N/m）分量。Yade 的实现采用每个断裂键周围应变能的最大变化量来估算声发射的矩震级。键一旦断裂，就会计算半径范围内的总应变能（$E_o = \sum_i^N E_i$）（半径由用户设定，为平均颗粒直径的数量 λ。 E_o 用作参考应变能，以计算后续时间步中的 $\Delta E = E - E_o$ 。最后，max(ΔE) 用于 [Scholz2003]推导的经验方程：

$$M_e = \frac{2}{3}\log \Delta E - 3.2$$

如果事件发生在其他事件的空间和时间窗口内，则会将它们聚类；这种方法与 [Hazzard2000] 和 [Hazzard2013]提出的方法类似。空间窗口就是用户定义的λ ，而时间窗口 T_{max} 的计算方式为：

$$T_{max} = int\left(\frac{D_{avg}\lambda}{max(v_{p1}, v_{p2})\Delta t}\right)$$

其中， D_{avg} 是构成失败事件的颗粒的平均直径（米），v_{p1} 而v_{p2} 是颗粒密度的 P 波速度（米/秒），Δt 是模拟的时间步（秒/时间步）。如图-*cluster*所示，聚类事件的最终位置就是聚类事件质心的平均值。这里，更新后的参考应变能通过将新断裂键周围唯一相互作用的应变能加到原始参考应变能上来计算（E_o）：

- 原始键断裂，在空间窗口 E_{orig} 内对断裂键的应变能（N_{orig}）求总和，$_o = \sum_{i=1}^{N_{orig}} E_i$

- 在原始键断裂的空间和时间窗口内检测到新的断裂键
- 通过添加唯一的键来更新参考应变 E_o （N_{new}）在新的断裂键空间窗口内$E_{new,o} = E_{orig,o} + \sum_{i=1}^{N_{new}} E_i$

此方法为计算$\Delta E = E - E_{new,o}$维持一个物理参考应变能，并且高度依赖空间窗口的尺寸。最终，聚类会增加较大事件的数量，从而使 b 值更接近典型的古腾堡-里希特曲线 [Hazzard2013]。

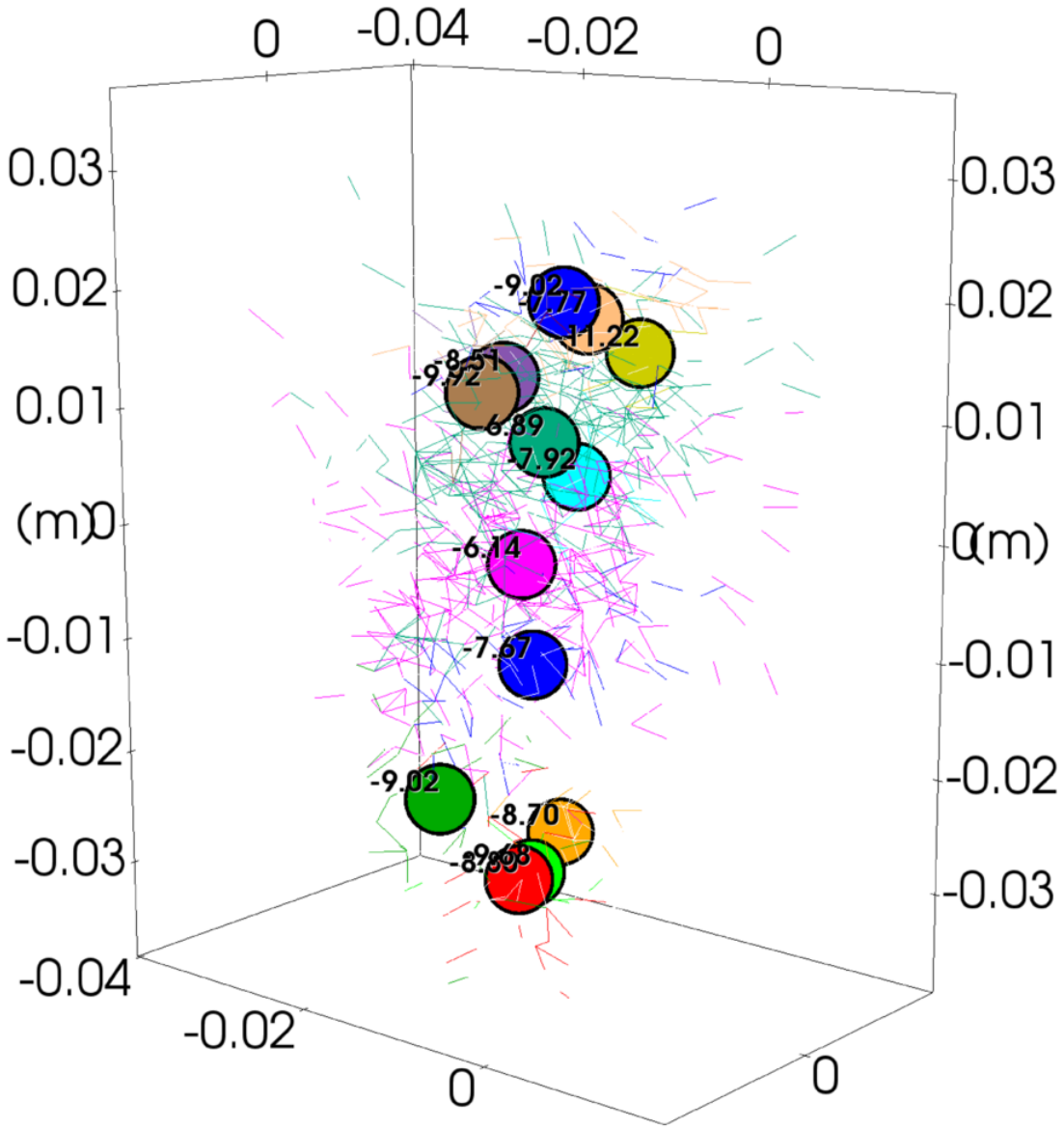


图 1：聚类后的断裂键（彩色直线）以及最终声发射事件（彩色圆）示例，并标出其事件震级。

如需详细了解底层算法，请参阅源代码。

4.4.3 在 Yade 中激活算法

Yade 的节理黏聚摩擦颗粒模型支持声发射模拟*(JCFpm)*。因此，模拟需要使用 *JCFpmMat* 、*JCFpmPhys* 和 *定律2_ScGeom__-JCFpmPhys*

你的材料赋值和引擎列表可能如下所示：

```
	JCFmat = O.materials.append(JCFpmMat(young=young, cohesion=cohesion,
		density=density, frictionAngle=radians(finalFricDegree),
		tensileStrength=sigmaT<style 标识符='2'>, poisson=poisson<style 标识符='5'>, label='JCFmat'<style 标识符='8'>,

		jointNormalStiffness=2.5e6,jointShearStiffness=1e6,jointCohesion=1e6))

	O.engines=[
			ForceResetter(),
			InsertionSortCollider([Bo1_Box_Aabb(),Bo1_Sphere_Aabb
				,Bo1_Facet_Aabb()]),
			InteractionLoop(
				[Ig2_Sphere_Sphere_ScGeom(), Ig2_Facet_Sphere_ScGeom()],
				[Ip2_FrictMat_FrictMat_FrictPhys(),
					Ip2_JCFpmMat_JCFpmMat_JCFpmPhys( \
						xSectionWeibullScaleParameter=xSectionScale,
						xSectionWeibullShapeParameter=xSectionShape,
						weibullCutOffMin=weibullCutOffMin,
						weibullCutOffMax=weibullCutOffMax)],
				[Law2_ScGeom_JCFpmPhys_JointedCohesiveFrictionalPM(\
					recordCracks=True, recordMoments=True,
					Key=identifier, label='interactionLaw'),
				Law2_ScGeom_FrictPhys_CundallStrack()]
),

	    GlobalStiffnessTimeStepper(),
	    VTKRecorder(recorders=['jcfpm','cracks','facets','moments'] \
				Key=identifier, label='vtk'),
			NewtonIntegrator(damping=0.4)
	]
```

其中大部分内容只是像往常一样启用 JCFpm，相关的声发射命令如下：

```
Law2_ScGeom_JCFpmPhys_JointedCohesiveFrictionalPM(...  recordMoments=True ...)
VTKRecorder(... recorders=[... 'moments' ...])
```

正确激活和使用声发射算法还需要一些其他命令：

clusterMoments 用于指示 Yade 按照模型描述中的上述说明，在用户设定的空间半径内对新产生的断裂交互作用进行聚类。该值默认为 True。

momentRadiusFactor 是 λ，其定义见上述模型描述。momentRadiusFactor 会改变超出初始相互作用的颗粒半径数量，Yade 根据该数量计算应变能变化。此外，Yade 使用 λ 来寻找用于聚类的其他断裂键。该值默认为 5（ [Hazzard2013] 得出结论，该值能够为整个系统的总计应变能变化提供准确的应变能变化近似）。

永不擦除 允许将旧的相互作用存储在内存中，即使它们已不再影响模拟。为确保 Yade 声发射簇模型稳定运行，必须将此值设为 True。

4.4.4 声发射的可视化与后处理

声发射的可视化和后处理方式与 JCFpm 裂纹类似。只要 *recordMo-ments=为 True* 且 记录器=[‘力矩’] ，，模拟就会生成带时间戳的 .vtu 文件，以便进行 Paraview 后处理。在 Paraview 中，可以根据模、组成相互作用的数量和事件时间筛选 声发射。 *fig-aeexample* 展示了三点弯曲试验中采集的声发射，以及根据模和时间进行筛选的结果

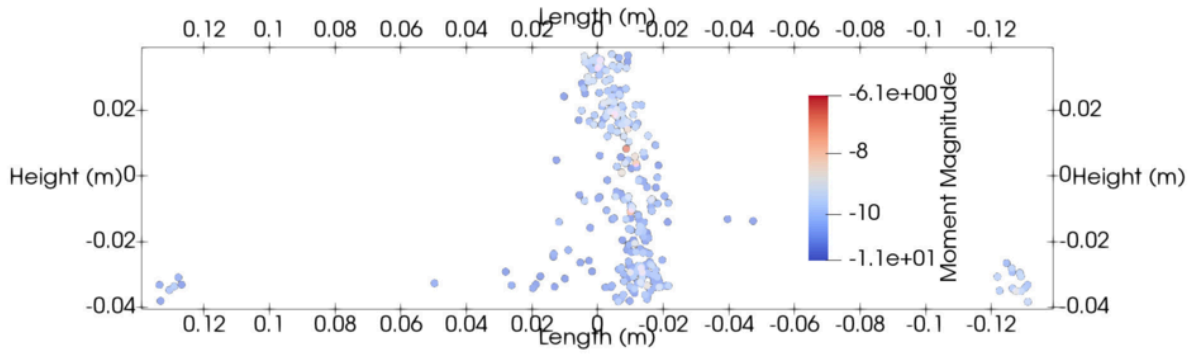


图 2：三点弯曲试验中模拟的声发射示例，并按模和时间进行筛选。

4.4.5 岩石非均质性的考虑

[Caulk2018] 和 [Caulk2020] 假设非均质岩石的行为取决于相互作用的颗粒边缘长度的分布。为支持这一假设， [Caulk2018] 和 [Caulk2020] 展示了如何利用阴极发光颗粒图像对岩石非均质性进行建模。基于所谓的颗粒边缘相互作用长度分布构建威布尔分布。在 Yade 的 *JCFpm* 中，威布尔分布用于通过修正相互作用面积来调整接触颗粒的相互作用强度 A_{int}：

$$A_{int} = \pi(\alpha_w \times \min(R_a, R_b))^2$$

其中 α_w 是威布尔修正因子，其分布如图-*weibullDist*所示。不同威布尔形状参数对应的抗拉强度分布如图-*strengthDist*所示。注意：威布尔形状因子为∞ 时，等同于未增强的 JCFpm 模型。

在 Yade 中，只需向*JCFpmPhys* 传递一个威布尔形状参数，即可轻松应用岩石非均质性：

```
Ip2_JCFpmMat_JCFpmMat_JCFpmPhys(
xSectionWeibullScaleParameter=xSectionScale,
xSectionWeibullShapeParameter=xSectionShape,
weibullCutOffMin=weibullCutOffMin,
weibullCutOffMax=weibullCutOffMax)
```

其中 截面威布尔形状参数 是所需的威布尔形状参数。尺度参数也可以用类似的方式赋值。如果要控制允许的最小修正因子，可以将其赋给 *weibullCutoffMin* 。最大修正因子也可以用类似的方式控制。

4.5 使用 YADE 一维垂向 VANS 流体求解

本说明旨在详细介绍如何在 YADE 中实际应用离散元-流体耦合。它是对以下三篇说明的补充： [Maurin2018_VANSbasis], [Maurin2018_VANSfluidResol] 以及[Maurin2018_VANSvalidations]它们分别详细介绍了流体动量平衡方程的理论基础、数值求解以及代码验证。

所有耦合和流体求解都仅依赖 HydroForceEngine 引擎，本文将详细介绍其用法。YADE 源代码中的 trunk/examples/HydroForceEngine/ 文件夹包含了使用 HydroForceEngine 实现不同用途的示例脚本。为了熟悉该引擎，建议阅读本文并测试/修改这些示例脚本。

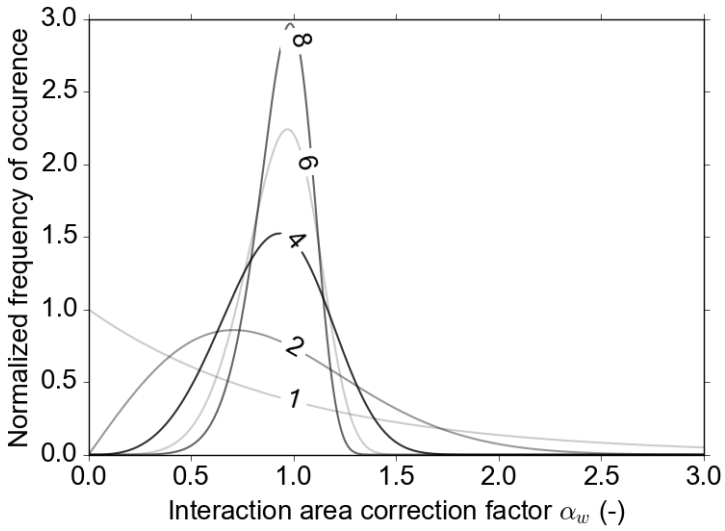


图 3：用于生成 α_w 的不同形状参数对应的威布尔分布。

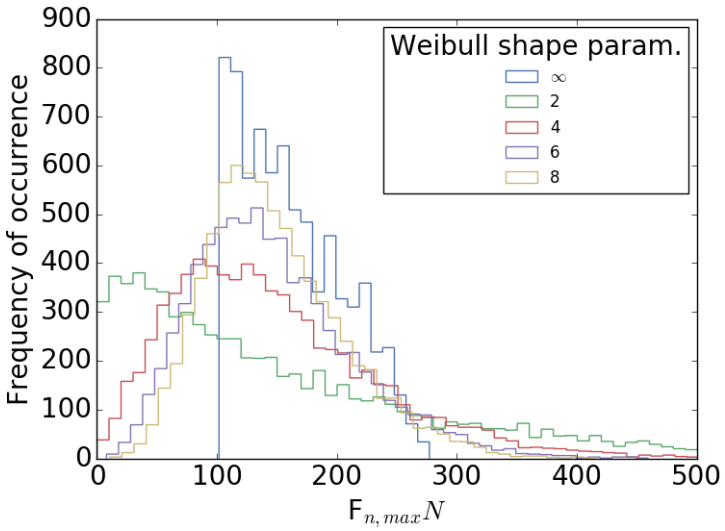


图 4：最 大离散元法颗粒黏结抗拉强度分布（对应不同的威布尔形状参数）

s.

4.5.1 YADE 中的离散元-流体耦合与流体求解

在 YADE 中，流体与离散元法的耦合通过名为 HydroForceEngine 的引擎实现，该引擎的源代码位于 trunk/pkg/common/HydroForceEngine.cpp 和 hpp 文件中。HydroForceEngine 主要有三个函数：

- 根据一维垂向流体速度剖面，对每个颗粒施加阻力和浮力（Hydro- ForceEngine::action）
- 它可以计算平均阻力、颗粒速度和固相体积分数剖面（Hydro- ForceEngine::averageProfile）
- 它可以根据给定的平均阻力、颗粒速度和固相体积分数剖面，求解第一节中详细介绍的流体速度方程（HydroForceEngine::fluidResolution）

我们可以清楚地看到这三个函数之间的联系。其思路是从离散元法中计算平均速度分布，将其作为流体求解的输入，然后根据得到的流体速度剖面，对颗粒施加相应的流体力。下文将结合 YADE 源代码 trunk/examples/HydroForceEngine/ 中的示例脚本，分别详细说明这三点，并提供精确的计算与成像结果。

4.5.2 阻力和浮力的施加（HydroForceEngine::action）

默认情况下，将 HydroForceEngine 添加到引擎列表时，它会对通过 ids 变量传入 HydroForceEngine 的标识符所对应的所有颗粒施加阻力和浮力。例如，在示例脚本 trunk/examples/HydroForceEngine/ 的引擎列表中就是这样做的：

```
O.engines = [ForceResetter(), ......
HydroForceEngine(densFluid = densFluidPY,...,ids = idApplyForce),
......NewtonIntegrator(gravity=gravityVector, label='newtonIntegr')]
```

其中， idApplyForce 对应一个颗粒 ID 列表，流体动力将作用于列表中的颗粒。下文将详细介绍作用于该 ID 列表中颗粒的浮力和阻力的表达式。

当流体处于静止状态时（HydroForceEngine.steadyFlow = False），HydroForceEngine 根据流体密度和重力加速度 g 对颗粒施加浮力，表达式如下：

$$\mathbf{f}_b^p = -\rho^f V^p \mathbf{g}.$$

同时，如果流体处于稳态湍流状态，与流体压力梯度相关的浮力在流向上没有分量（参见 [Maurin2018] 第 5 页的讨论）。将选项 HydroForceEngine.steadyFlow 设为 True 后，浮力的表达式变为：

$$\mathbf{f}_b^p = -\rho^f V^p (\mathbf{g}.\mathbf{e}_x)\mathbf{e}_x.$$

此外，HydroForceEngine 会对 ID 列表中的每个颗粒施加阻力。该阻力取决于颗粒的速度和流体速度，后者由一维流体速度剖面 HydroForceEngine.vxFluid 定义。此流体速度剖面可以根据流体模型计算得出，也可以由用户设定并保持恒定。根据这一一维垂向流体速度剖面，作用于颗粒 p 的阻力为：

$$\mathbf{f}_D^p = \frac{1}{2} C_d A \rho^f \|u_p^f \mathbf{e}_x - \mathbf{v}^p\| \left(u_p^f \mathbf{e}_x - \mathbf{v}^p\right),$$

其中，u_p^f 是颗粒 p中心处的流体速度，$\mathbf{v}^p$ 是颗粒速度，ρ^f 是流体密度，$A = \pi d^2/4$ 是球体在流动中所受作用的面积，C_d 是阻力系数，用于计及

颗粒雷诺数的影响 [Dallavalle1948] 以及由于其他颗粒存在而增加的阻力（受阻效应， [Richardson1954]:

$$C_d = \left(0.44 + \frac{24}{Re_p}\right)(1-\varphi_p)^{-\gamma} = \left(0.44 + 24\frac{\nu^f}{\|u_p^f e_x - v^p\| d}\right)(1-\varphi_p)^{-\gamma}$$

其中，φ_p 颗粒中心处的固相体积分数由 HydroForceEngine.phiPart 计算得出， γ Richardson-Zaki 指数可通过参数 HydroForceEngine.expoRZ 设置（默认值为 3.1）。

HydroForceEngine 也可以施加升力，但默认不启用此功能（HydroForceEngine.lift =False）。考虑到实际公式存在不确定性，作者不建议使用此功能（参见 [Maurin2015]和 [Schmeeckle2007]第 6 页的讨论）。

用户可以指定流体速度剖面（HydroForceEngine.vxFluid）和固相体积分数剖面（HydroForceEngine.phiPart），因此无需使用函数 average- Profile 和流体求解，即可通过 HydroForceEngine 对颗粒施加阻力和浮力。此类用法的示例可在源代码中找到：trunk/examples/HydroForceEngine/oneWayCoupling脚注{在此算例中，我们所说的单向耦合是指流体会影响颗粒，但颗粒不会反过来影响流体}。

4.5.3 固相平均 (HydroForceEngine::averageProfile)

如前所述，为了求解流体方程，需要从离散元法（DEM）中计算固相体积分数、固相速度和平均阻力剖面。为此，设置了函数 Hydro- ForceEngine.averageProfile()。该函数用于计算规则网格上的平均剖面，计算位置位于两个网格节点之间。为了与流体速度剖面的编号方式一致，平均向量的尺寸为 $ndimz+1$ 尽管默认情况下，顶部和底部边界处的物理量不会计算，而是设为零footnote{无需计算边界处的固相 DEM 物理量，因为流体求解中不考虑这些位置，参见 [Maurin2018_VANS流体求解] 的边界小节。}。textcolor{red}{你应该这样做}

固相体积分数剖面通过计算所考虑层内颗粒的体积来评估。该层由沿壁面法向的网格步长定义，但在长度和宽度方向上均覆盖整个试样。为匹配流体求解，我们仅沿壁面法向进行离散平均。同时，这在物理上也是合理的：在稳态下，问题的平均状态呈单向性，因此测得的平均物理量应仅沿垂直方向 z 变化。因此，固相体积分数通过计算所考虑层内颗粒的体积来评估$i+1/2$:

$$\varphi_{i+1/2} = \sum_{p\in[idz;(i+1)dz]} V^p_{i+1/2};$$

其中，总和是针对颗粒而言的 p 这些颗粒至少有一部分体积位于该层内 $i+1/2$，即高度介于 $i*dz$和 $(i+1)*dz$之间，并且 $V^p_{i+1/2}$是所考虑颗粒中包含在所考虑层内的体积。后者对应于球体切片两个点之间的积分，可在柱坐标中进行解析求值。遵循这一公式以及 [Jackson2000] 采用加权阶跃函数的形式体系，任何颗粒相关量 K均可按以下公式求平均：

$$\langle K\rangle^p\big|_{i+1/2} = \frac{\sum_{p\in[idz;(i+1)dz]} V^p_{i+1/2} K^p}{\sum_{p\in[idz;(i+1)dz]} V^p_{i+1/2}},$$

其中， K^p 是与颗粒 p相关的量，例如颗粒流向速度。在这种情况下，我们可以写为：

$$\langle v_x\rangle^p\big|_{i+1/2} = \frac{\sum_{p\in[idz;(i+1)dz]} V^p_{i+1/2} v^p_x}{\sum_{p\in[idz;(i+1)dz]} V^p_{i+1/2}},$$

其中， v_x^p 是颗粒 p的速度。关于流体传递给颗粒的平均流向阻力的计算，可以类似地写为：

$$\langle f_{D,x}\rangle^p|_{i+1/2} = \frac{\sum_{p\in[idz;(i+1)dz]} V^p_{i+1/2} f^p_{D,x}}{\sum_{p\in[idz;(i+1)dz]} V^p_{i+1/2}},$$

其中， $f^p_{D,x}$ 是作用在颗粒 p上的阻力。

正如下一部分将详细说明的，这些平均剖面可用于流体求解，也可用于分析，例如 [Maurin2015b] [Maurin 2018]中对推移质输运的分析。

4.5.4 流体求解\HydroForceEngine::fluidResolution

为了在流体-DEM耦合框架中使用流体求解，需要调用函数 HydroForceEngine.averageProfile()，以计算平均固相体积分数剖面、流向速度和流向阻力。后者是计算流体方程中 β 所考虑各项所必需的（详见 [Maurin2018_VANSfluidResol] ）。 β 定义如下：

$$n\left\langle f_x^f\right\rangle^p\Big|_{i+1/2} = \beta_{i+1/2}\left(\left\langle u_x\right\rangle^f\Big|_{i+1/2} - \left\langle v_x\right\rangle^p|_{i+1/2}\right)$$

因此，可以直接根据平均阻力、颗粒速度以及上一次迭代时的流体速度来计算（在流体求解中显式处理项 β ）：

$$\beta^n_{i+1/2} = \frac{n\left\langle f_x^f\right\rangle^p\Big|^{n-1}_{i+1/2}}{\left\langle u_x\right\rangle^f\Big|^{n-1}_{i+1/2} - \left\langle v_x\right\rangle^p|^{n-1}_{i+1/2}}$$

其中，固体变量以上标 $n-1$ 表示，因为这些变量已知，无需在每个时间步重新计算footnote{某种程度上， β^n 或许更适合写成 β^{n-1}}。这一项称为 taufsi，可直接在代码中计算。

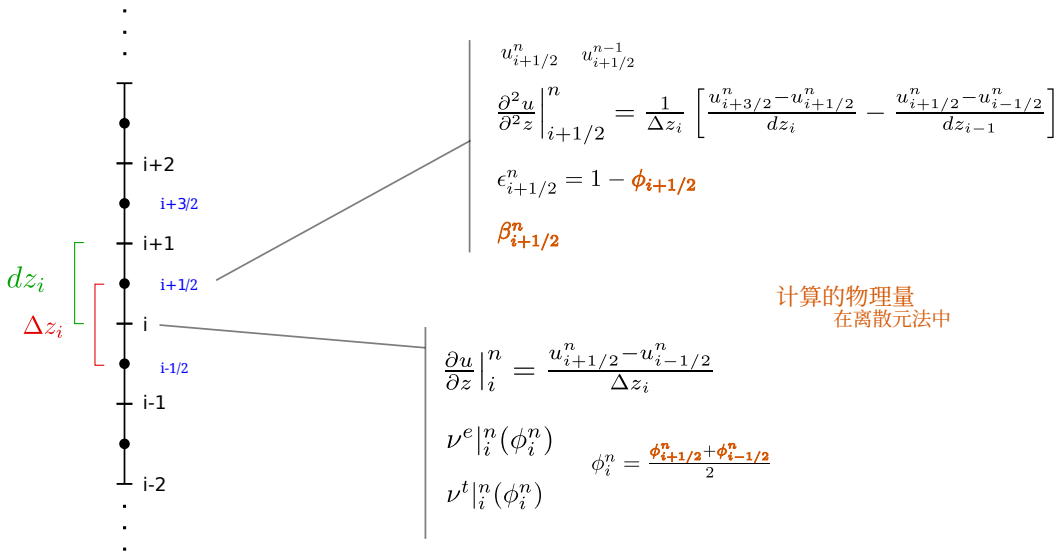


图 5：规则网格下数值流体求解及变量定义的示意图。所有定义同样适用于采用可变空间步长的网格。

求解流体分辨率所需的全部物理量——在[Maurin2018_VANSfluidResol] 中强调并在图 *fig-scheme* 中再次提及——现已明确给出。可以直接在 YADE 中使用函数 HydroForceEngine.averageProfile() 对这些量进行求值。此后，可在给定的时间 t_{resol} 内，以给定的时间步 Δt 直接调用

函数 HydroForceEngine.fluidResolution（t_{resol}，Δt）可执行 [Maurin2018_VANSfluidResol]中描述的流体求解，$N = t_{resol}/\Delta t$ 持续 Δt时间，时间步为 β，并将 $\langle v_x \rangle$ 和 φ 的垂向分布视为恒定不变。因此，不仅要注意时间步，也要注意耦合周期；耦合周期不应过长，以免速度分布发生剧烈变化，而传递的阻力未相应增大，从而导致 DEM 中出现不符合物理规律的行为。

在 YADE 源代码中的示例脚本 trunk/examples/HydroForceEngine/twoWayCoupling/sedimentTransportEx1DRANSCoupling.py 中，离散元法（DEM）与流体求解的耦合周期为 $fluidResolPeriod = 10^{-2}s$（默认值），流体时间步为 $dtFluid = 10^{-5}s$。这意味着，离散元法（DEM）会先运行 $10^{-2}s$，然后在进行流体求解时冻结；流体求解持续 $fluidResolPeriod/dtFluid = 10^3$步，每步耗时 $\Delta t = 10^{-5}$。之后，离散元法（DEM）再次运行，但采用新的流体速度剖面，持续 $10^{-2}s$，随后再次冻结，依此类推。两次流体求解之间的周期应经过测试，且不宜过长（参见 [Maurin2015b] 的附录）。
与此同时，流体求解也可以单独使用，无需与离散元法（DEM）耦合，尤其可用于在已知算例中验证流体求解。这在 YADE 源代码的示例文件夹 trunk/examples/HydroForceEngine/fluidValidation/ 中完成，其中考虑并验证了泊肃叶流和对数层算例。

4.6 势粒子与势块

本节所述算法的科学发展渊源可追溯至：[Boon2012](势块代码），[Boon2013b]（势粒子代码），以及 [Boon2015]（块体生成代码）。

4.6.1 简介

本节介绍两种用于模拟的代码：(i) 基于势粒子概念模拟 [Houlsby2009], 提出的非球形颗粒，其求解过程采用 [Boon2013] 针对三维情况提出的方法；(ii) 基于线性规划概念，使用平面线性不等式模拟多面体块 [Boon2012]。这些代码在 YADE 中定义了两个形状类，即*PotentialParticle* 和*PotentialBlock*。尽管语法上有一些相似之处，但它们在颗粒的形态特征以及用于促进接触检测的方法方面存在明显差异。

势粒子代码（本文简称*PP*）详见[Boon2013], ，其中非球形颗粒由 2nd 次多项式函数与部分球体组合而成，其边缘则采用用户定义的曲率半径进行圆化。

势块代码（本文简称*PB*）用于模拟表面为平面的多面体颗粒，其依据是[Boon2012], 的研究。在该方法中，使用光滑的内部势颗粒来计算接触法向量。此代码与块体生成算法兼容，该算法定义于[Boon2015], 中，其中势块可通过原始完整块体与不连续面的交集生成。

这两种代码彼此独立，也就是说，可以分别编译和使用其中任意一种，而无需启用另一种；两者之间不会相互作用（即无法在 PP 和 PB 之间建立接触）。启用 PB 代码会自动编译块体生成算法。

4.6.2 Potential Particles 代码（PP）

势粒子的概念由势粒子 提出并发展，作者是 [Houlsby2009]。一对势粒子之间的接触检测问题被表述为一个约束优化问题，并在二维中使用牛顿-拉夫森方法求解方程。在 [Boon2013] 中，该方法扩展到了三维，并提出了更稳健的求解方案。许多数值优化求解器通常无法处理不连续性、病态梯度（雅可比矩阵）或曲率（海森矩阵），而这些问题

在 [Boon2013], 中通过重新表述问题，并使用锥优化求解器求解方程，克服了这些障碍。先前版本使用 MOSEK（采用其学术许可证），而目前则使用 [Boon2013] 编写的内部代码来求解锥优化问题。势粒子的定义如4.21式 [Houlsby2009]所示：

$$f = (1-k)\left(\sum_{i=1}^{N}\langle a_i x + b_i y + c_i z - d_i\rangle^2 - r^2\right) + k(x^2+y^2+z^2-R^2) \tag{4.21}$$

其中 (a_i, b_i, c_i) 是相对于颗粒局部坐标系定义的 i^{th} 平面的法向量， d_i 是该平面到局部原点的距离。 $\langle\rangle$ 是 Macaulay 括号，即 $\langle x\rangle = x$ 当 $x > 0$ 时； $\langle x\rangle = 0$ 当 $x \leq 0$ 时。各平面的组装方式使其法向量均朝外。将它们的平方相加，并以距离 r扩展，该距离也与角点处的曲率半径有关。此外，还添加了一个“影子”球形颗粒； R 是球体的半径， $0 < k \leq 1$表示颗粒球形度所占的比例。图*fig-pp*展示了参数 k取不同值时若干长方体势颗粒的几何形状。

The 出于计算方面的考虑，势函数按如下形式归一化（4.22） [Houlsby2009]:

$$f = (1-k)\left(\sum_{i=1}^{N}\frac{\langle a_i x + b_i y + c_i z - d_i\rangle^2}{r^2} - 1\right) + k\left(\frac{x^2+y^2+z^2}{R^2} - 1\right) \tag{4.22}$$

此势函数取值为：

- $f = 0$：位于颗粒表面
- $f < 0$：位于颗粒内部
- $f > 0$：位于颗粒外部

为确保数值稳定性，不建议使用接近$k=0$的值。特别是，从理论角度而言，不能使用极值$k=0$ ，因为势粒子 是针对严格凸形状（曲面）构造的。

4.6.3 Potential Blocks 代码（PB）

该势块代码是在 CW Boon 的哲学博士论文期间开发的 [Boon2013b] ，并在 [Boon2012]中进行了讨论。它最初是为岩石工程应用开发的，用于模拟具有平坦表面的多边形和多面体块体。这些块体仅通过线性不等式定义。与*PotentialParticle* 形状类不同，它不考虑球形项（因此，实际上 $k=0$）。尽管k 和R 是 *PotentialBlock* 形状类的输入参数，但在计算过程中它们的作用为 NULL。特别是， R 在源代码中用作块体的特征尺寸，但它并不表示“影子颗粒”的半径；而对于势粒子，它表示的正是影子颗粒的半径。这个R 值在势块代码中用于计算线搜索的初始二分步长，以获得颗粒上的一个点；随后，该点用于计算接触过程中的重叠距离。

对于由N 个平面定义的凸粒子，其所占据的空间可通过以下不等式定义（4.23）：

$$a_i x + b_i y + c_i z \leq d_i, i = 1:N \tag{4.23}$$

其中 (a_i, b_i, c_i) 是相对于颗粒局部坐标系定义的 i^{th} 平面的单位法向量，且 d_i 是该平面到局部原点的距离。根据 [Boon2012], , 使用内光滑势粒子计算接触法线，其公式如下（4.24）：

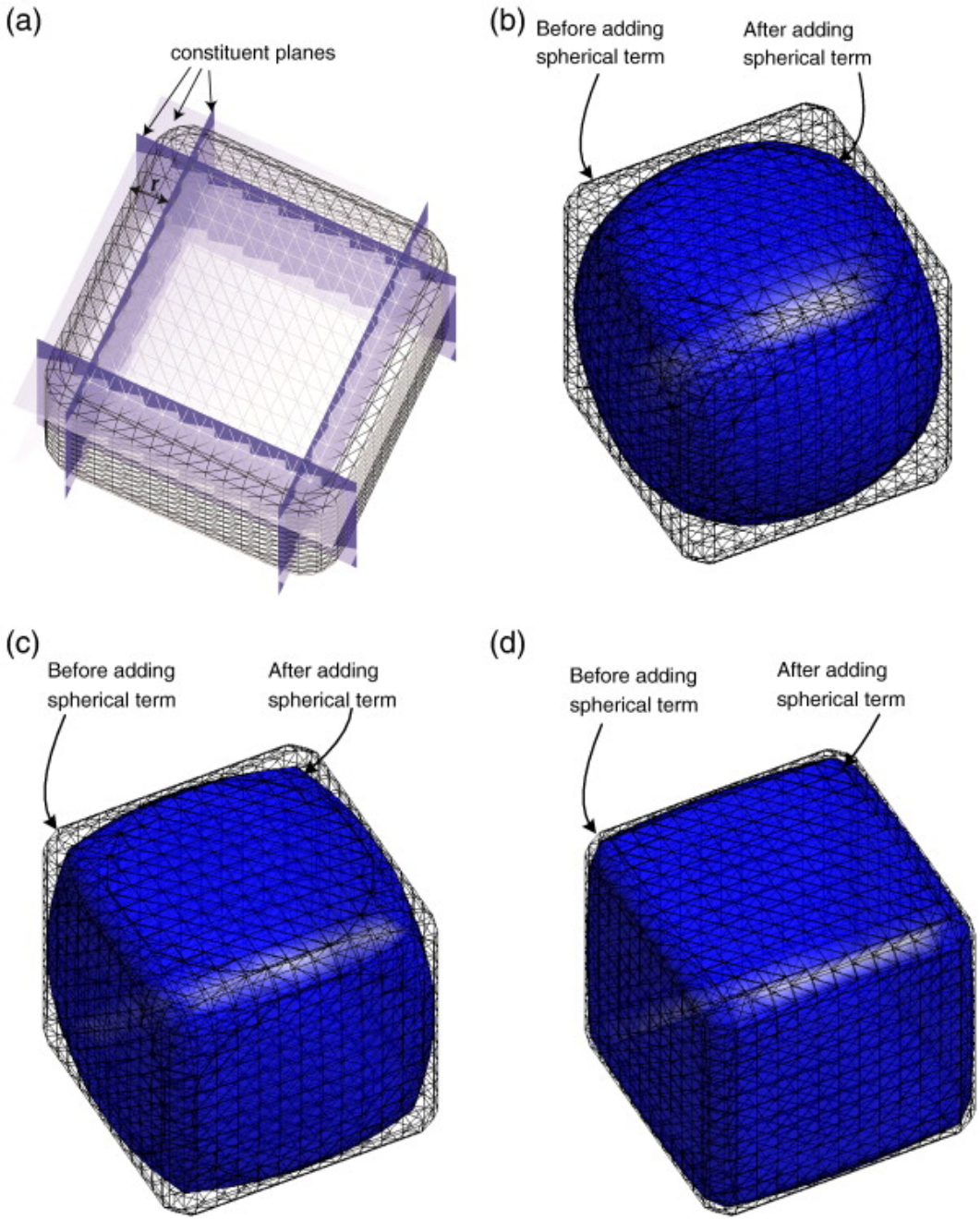


图 6：势颗粒的构造。(a) 组成平面经平方处理后，向外扩展常数 r 的距离，并添加一部分球体。包含球形项的颗粒见 (b) k=0.9、(c) k=0.7 和 (d) k=0.4（据 [Boon2013]）。

$$f = \sum_{i=1}^{N} \langle a_i x + b_i y + c_i z - d_i + r \rangle^2 \tag{4.24}$$

该势颗粒的内部由距离 r 定义，即位于实际粒子内部，且边缘以半径或曲率r舍入（另见图 *fig-pbInner*）。

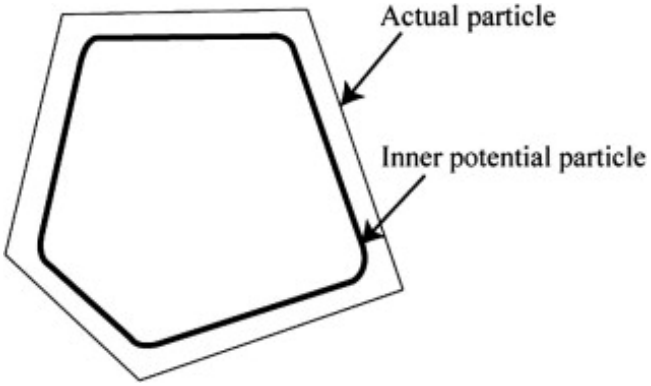


图 7：势颗粒定义在实际粒子内部。颗粒上任意一点的法向量均可通过对势颗粒求一阶导数计算得出。（改编自 [Boon2012]）。

在 YADE 中，势块 的数学表达式略有不同，因为其形状由一组平面组成，如（4.25）所示：

$$a_i x + b_i y + c_i z - d_i - r = 0, i = 1 : N \tag{4.25}$$

而用于计算接触法线的内部势颗粒 定义如（4.26）所示：

$$f = \sum_{i=1}^{N} \langle a_i x + b_i y + c_i z - d_i \rangle^2. \tag{4.26}$$

现在，势块 表面距 $(d_i + r)$ 局部粒子中心的距离为 $(d_i + r)$，而内部势颗粒距 d 局部粒子中心的距离为 d 。

值得强调的是，势块 的形状由平面组定义，而不是像势粒子 代码那样由单个隐式势函数定义。势块 代码中的内部势颗粒仅用于计算接触法线。

确定一对块之间是否相交的问题，可转化为一个标准线性规划问题：求出满足定义这两个块的所有线性不等式的可行域。接触点计算为可行域的解析中心，这是凸优化计算中内点法的一个知名概念。接触法线由定义在块内部的光滑“势颗粒”的梯度求得。重叠距离则通过沿接触法线在重叠区域内进行二分搜索来计算。

势块 的线性规划求解器最初使用的是 CPLEX，但后来更新为由 COIN-OR 开发的 CLP，因为后者可以从 Ubuntu 或 Debian 发行版下载，无需学术许可证。

4.6.4 引擎

PP 和 PB 代码使用各自的类来处理包围体、接触几何与物理，以及以 vtk 格式记录输出；同时，它们从摩擦材料类*FrictMat*中获取颗粒间摩擦角。除非另有说明，调用这些类的语法相似。

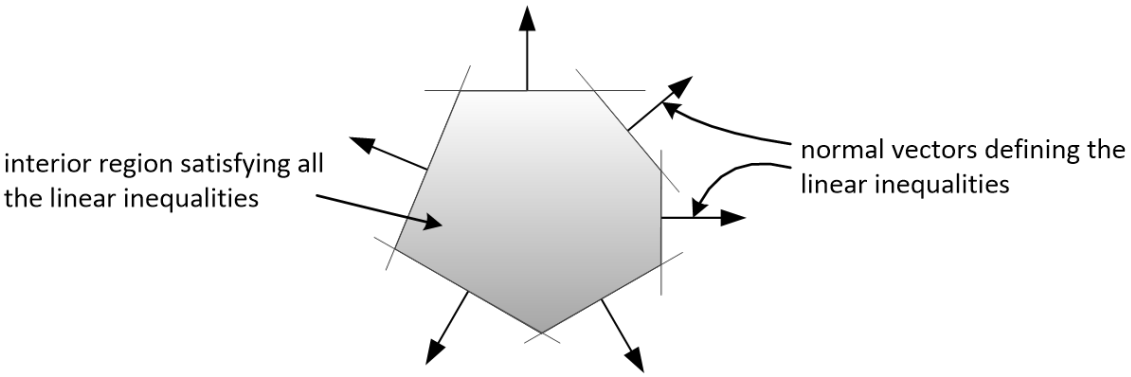


Fig. 8：一个势块。面的法向量朝外（依据 [Boon2013b]）

形状	*PotentialParticle*	*PotentialBlock*
材料	*FrictMat*	*FrictMat*
边界函子	*PotentialParticle2AABB*	*PotentialBlock2AABB*
IGeom	*ScGeom*	*ScGeom*
IGeomFunctor	*Ig2_PP_PP_ScGeom*	*Ig2_PB_PB_ScGeom*
IPhys	*KnKsPhys*	*KnKsPBPhys*
IPhysFunctor	*Ip2_FrictMat_FrictMat_KnKsPhys*	*Ip2_FrictMat_FrictMat_KnKsPBPhys*
LawFunctor	*定律2_SCG_KnKsPhys_KnKs定律*	*定律2_SCG_KnKsPBPhys_KnKsPB定律*
VTK 记录器	*PotentialParticleVTKRecorder*	*PotentialBlockVTKRecorder*

一个简单的模拟循环使用势块时如下所示：

```
O.engines=[ForceResetter(), InsertionSortCollider([PotentialBlock2AABB()], verletDist=0.01),
InteractionLoop([Ig2_PB_PB_ScGeom(twoDimension=True, unitWidth2D=1.0, calContactArea=True)],
[Ip2_FrictMat_FrictMat_KnKsPBPhys(kn_i=1e8, ks_i=1e7, Knormal=1e8, Kshear=1e7,␣
→viscousDamping=0.2)], [Law2_SCG_KnKsPBPhys_KnKsPBLaw(label='law', neverErase=False,␣
→allowViscousAttraction=False)]),
NewtonIntegrator(damping=0.2, exactAsphericalRot=True, gravity=[0,0,-9.81]),
PotentialBlockVTKRecorder(fileName='./vtk/file_prefix', iterPeriod=1000,␣→twoDimension=True,
 sampleX=30, sampleY=30, sampleZ=30, maxDimension=0.2, label='vtkRecorder')]
```

应注意 二维 参数，它用于定义接触应按二维还是三维方式处理。

4.6.5 接触定律

在两种代码中，法向力的计算方式如下：

$$\mathbf{F_n} = \mathrm{Knormal} \cdot A_c \cdot u_n \cdot \mathbf{n} \tag{4.27}$$

其中 Knormal 为法向刚度系数 [kN/m^3]； A_c 为接触面积 [m^2]， u_n 为重叠距离。因此，每个接触的法向刚度 [kN/m] 为 $k_n = \mathrm{Knormal} \cdot A_c$，其中 A_c 会在每个时间步长中更新。

剪切力采用类似的逻辑逐步计算。接触发生滑移前，剪切力向量的增量计算如下：

$$\Delta \mathbf{F}_s = -K_{shear} \cdot A_c \cdot \Delta \mathbf{u}_s \tag{4.28}$$

其中 K_{shear} 剪切刚度系数 $[kN/m^3]$ 和 Δu_s 当前的相对剪切位移 .

接触面积

接触面积通过启发式算法计算：沿接触剪切方向搜索重叠体积表面上的点。具体而言，它是沿剪切方向、经过接触点的重叠体积二维切片的面积。如果 `twoDimension=True`，则 *contactArea* 参数计算如下：

```
if(twoDimension) { phys->contactArea = phys->jointLength*unitWidth2D;}
```

unitWidth2D 参数由用户定义（通常等于 1.0），表示二维模拟中的面外宽度。如果 *calContactArea* =True，则计算 *contactArea* 和 *jointLength* 参数。反之，这些参数均视为等于 1.0，接触定律退化为具有恒定刚度的线性定律。对于 *contactArea*，取一个最小值，以表示重叠体积实际为一个点的情况。

重叠距离

重叠距离 u_n 通过沿接触法线方向应用区间二分搜索算法计算，搜索从接触点开始，找到重叠区域表面上两个相对的点。该距离存储在参数 *penetrationDepth*中，表示这两个相对点之间的距离。

4.6.6 PP 和 PB 的形状定义

势粒子 和势块 代码的一大优势在于，只需使用颗粒各个面的方程，即可解决颗粒形状的几何定义和接触检测问题。通过这种方式，使用单一数据结构即可建立接触，无需存储顶点信息或顶点之间的连接关系。这一特性使其计算开销较低，同时所有接触都采用相同的处理方式（无需区分面-面、面-边、面-顶点、边-边、边-顶点或顶点-顶点接触）。因此，颗粒的几何体在形状类中通过各面的法向量值以及各面到局部原点的距离来定义。

例如 ，要定义一个带圆角的长方体（6 个面），边长为 D，其中心位于其本地 质心处，并 沿主轴对齐，使用 势颗粒 代码，我们设置：

```
r=D/10.
k=0.3
R=D/2.
b=Body()
b.shape=PotentialParticle( r=r, k=k, R=R,
                           a=[   1.0,    -1.0,     0.0,     0.0,     0.0,     0.0],
                           b=[   0.0,     0.0,     1.0,    -1.0,     0.0,     0.0],
                           c=[   0.0,     0.0,     0.0,     0.0,     1.0,    -1.0],
                           d=[D/2.-r,  D/2.-r,  D/2.-r,  D/2.-r,  D/2.-r,  D/2.-r], ...)
```

向量参数 a, b, c, d 的第一个元素表示第一个平面的法向量及其到局部原点的距离，第二个元素表示第二个平面，以此类推。

使用 势粒子代码时，这并不是一个完美的立方体，因为颗粒几何形状是通过式（4.22）中的势函数定义的。需要注意的是，在该势函数中，这些平面按平方和的方式组合，颗粒棱边以曲率半径 r 圆滑处理，然后通过添加半径为 R的“阴影”球形颗粒，使颗粒面弯曲，添加比例由参数 k定义。向量参数 r 的每个元素都减去一个值 d，以补偿势粒子因 r而膨胀。

上述参数 a_i、b_i、c_i、d_i 对应于式（4.25）中使用的平面：

$$
\begin{aligned}
1.0x + 0.0y + 0.0z = D/2 &\Leftrightarrow +x = D/2\\
-1.0x + 0.0y + 0.0z = D/2 &\Leftrightarrow -x = D/2\\
0.0x + 1.0y + 0.0z = D/2 &\Leftrightarrow +y = D/2\\
0.0x - 1.0y + 0.0z = D/2 &\Leftrightarrow -y = D/2\\
0.0x + 0.0y + 1.0z = D/2 &\Leftrightarrow +z = D/2\\
0.0x + 0.0y - 1.0z = D/2 &\Leftrightarrow -z = D/2
\end{aligned}
$$

要使用 势块 代码将边长为D的立方体建模，我们定义：

```
r=D/10.
R=D/2.*sqrt(3)
b=Body()
b.shape=PotentialBlock( r=r, R=R,
                        a=[   1.0,    -1.0,     0.0,     0.0,     0.0,     0.0],
                        b=[   0.0,     0.0,     1.0,    -1.0,     0.0,     0.0],
                        c=[   0.0,     0.0,     0.0,     0.0,     1.0,    -1.0],
                        d=[D/2.-r,  D/2.-r,  D/2.-r,  D/2.-r,  D/2.-r,  D/2.-r], ...)
```

使用 势块 代码时，该颗粒的几何形状（即其占据的空间）具有尖锐的边和扁平的面，由给定的平面定义；而在计算接触法线时，则使用边缘经过圆角处理的内部势颗粒，其形式如式（4.26）所示，并完全位于实际粒子内部。存储在向量参数 d中的平面到局部原点的距离会减去 r，以根据式（4.25）获得精确的边长 D。值r 必须足够小，以使 $d_{min} - r > 0$；同时也应足够大，以便正确计算接触点处内部势颗粒的梯度。建议值为 $r \approx 0.5 * d_{min}$。

为确保数值稳定性，建议将每个平面的法向量归一化，使得 $a_i{}^2 + b_i{}^2 + c_i{}^2 = 1$。颗粒面的数量不受限制，这一特性使得多种凸颗粒形状的建模成为可能。

实际上，颗粒的几何体通常以顶点及其连接关系的形式给出（例如，以表面网格的形式，如 .stl 文件）。在这种情况下，用户可以计算每个面的法向量，从而得到系数 a_i、b_i、c_i；再利用每个面上的一个顶点，计算系数 d_i。目前正在开发一个 Python 例程，让用户无需额外操作即可完成这些计算。

4.6.7 PP 和 PB 的体定义

使用 *PotentialParticle* 或 *PotentialBlock* 形状类定义刚体时，必须使用 `_commonBodySetup` 函数进行组装，该函数位于文件 py/utils.py中。例如，要定义一个*PotentialParticle*：

```
O.materials.append(FrictMat(young=-1,poisson=-1,frictionAngle=radians(0.0),density=2650,label=
→'frictionless'))

b=Body()
b.shape=PotentialParticle(...)
b.aspherical=True # To be used in conjunction with exactAsphericalRot=True in the␣
→NewtonIntegrator
```

（续见下页）

（接上页）

```
# V: Volume
# I11, I22, I33: Principal inertias
utils._commonBodySetup(b,V,Vector3(I11,I22,I33), material='frictionless', pos=(0,0,0),␣
→fixed=False)b.state.pos=Vector3(xPos,yPos,zPos)
b.state.ori=Quaternion((random.random(),random.random(),random.random()),random.random())
b.shape.volume=V;O.bodies.append(b)
```

必须先定义*PotentialParticle*，使其局部坐标轴与主轴重合，此时惯性张量为对角形式。更具体地说，必须旋转定义平面法向量的平面系数（a_i, b_i, c_i），这样当颗粒的方向为零时，*PotentialParticle*的方向就与其主轴一致。

需要注意的是，这里提到的主惯性值 `I11, I22, I33` 除以了所考虑材料的密度，因为它们会在 `_commonBodySetup` 函数中乘以密度。颗粒的质量也在同一函数中计算，因此我们无需手动设置 `b.mass=V*density`。

对于势粒子，必须如上所示手动计算体积和惯性，并将其赋给刚体。对于势块，体积和惯性张量可自动计算。用户无需将颗粒定义到其主轴上，因为源代码会自动处理；如果未为参数*R*指定值，则将其计算为最远顶点间距离的一半。

例如，定义一个*PotentialBlock*:

```
O.materials.append(FrictMat(young=-1,poisson=-1,frictionAngle=radians(0.0),density=2650,label=
→'frictionless'))b=Body()
b.shape=PotentialBlock(R=0.0, ...) #here we set R=0.0 to trigger automatic calculation of R
b.aspherical=True # To be used in conjunction with exactAsphericalRot=True
utils._commonBodySetup(b,b.shape.volume,b.shape.inertia, material='frictionless',␣
→pos=Vector3(xPos,yPos,zPos),  fixed=False)
b.state.ori=b.shape.orientation # this will rotate the particle to its initial random system.␣
→If b.state.ori=Quaternion.Identity,  the PB is oriented to its principal axes
O.bodies.append(b)
```

4.6.8 边界粒子

PP代码和PB代码支持定义边界 粒子，这些粒子只与非边界粒子相互作用。这些粒子有多种用途，例如模拟作用于颗粒样本的加载板；不同应用还可能衍生出其他用途。在这两种代码中，都可以通过形状类中的布尔参数*isBoundary* 将颗粒设为边界颗粒。

在PP代码中，所有颗粒都使用*Knormal* 和*Kshear*作为相同的法向和剪切接触刚度，由*Ip2_FrictMat_FrictMat_KnKsPhys* 函子定义。

PB代码支持为边界与非边界 颗粒之间，或非边界 与非边界 颗粒之间的相互作用定义不同的接触刚度值。当 `isBoundary=False` 时，会将相关的*PotentialBlock* 设定为与其他非边界颗粒相互作用，并分别采用法向刚度系数*Knormal* 和*Kshear*。当 `isBoundary=True` 时，会将相关的*PotentialBlock* 设定为与非边界颗粒相互作用，并分别采用法向刚度系数kn_i 和ks_i。

4.6.9 可视化

可通过 qt 环境（OpenGL）对 *PotentialParticle* 和 *PotentialBlock* 形状类进行可视化。此外，还可以使用 *export.VTKExporter.exportPotentialBlocks* 函数以及 *PotentialParticleVTKRecorder* 和 *PotentialBlockVTKRecorder* 引擎，以 vtk 格式导出分析中的几何信息和交互信息（可在 ParaView 中可视化）。请注意，目前 *PotentialBlockVTKRecorder* 记录的是颗粒的舍入近似形状，而非具有尖锐角点和边的实际颗粒。

在 qt 环境中，使用行进立方体算法对 *PotentialParticle* 形状类进行可视化，显示精度可由用户设定。此项由以下参数控制：

```
# Potential Particles
Gl1_PotentialParticle.sizeX=20
Gl1_PotentialParticle.sizeY=20
Gl1_PotentialParticle.sizeZ=20
```

对于 vtk 格式的输出，也可以使用 *PotentialParticleVTKRecorder* 或 *Potential-BlockVTKRecorder*进行类似设置，语法如下：

```
# Potential Particles
PotentialParticleVTKRecorder(sampleX=30, sampleY=30, sampleZ=30, maxDimension=20)

# Potential Blocks
PotentialBlockVTKRecorder(sampleX=30, sampleY=30, sampleZ=30, maxDimension=20)
```

参数 sizeX,Y,Z（用于 OpenGL 可视化）和 sampleX,Y,Z（用于 vtk 格式输出）表示将颗粒的 Aabb 划分为网格的分区数量，该网格用于根据全局坐标轴 X、Y、Z 绘制颗粒几何体。值越大，颗粒形状的显示就越精确，但会降低 qt 中的可视化速度和 .vtk 文件的写入速度，并增加 .vtk 文件的大小。对于 vtk 格式输出，用户还可以设置参数*maxDimension*；如果选定的 sampleX,Y,Z 值过小，该参数将覆盖这些值，如下所述：

$$\text{if } |xmax-xmin|/sampleX > maxDimension \Rightarrow sampleX = |xmax-xmin|/maxDimension$$
$$\text{if } |ymax-ymin|/sampleY > maxDimension \Rightarrow sampleY = |ymax-ymin|/maxDimension$$
$$\text{if } |zmax-zmin|/sampleZ > maxDimension \Rightarrow sampleZ = |zmax-zmin|/maxDimension$$

PotentialParticleVTKRecorder 和 *PotentialBlockVTKRecorder* 还支持可选地记录颗粒的速度（线速度和角速度）、交互信息（接触点和力）、颜色及标识符，方法如下：

```
# Potential Particles
PotentialParticleVTKRecorder(..., REC_VELOCITY=True, REC_INTERACTION=True, REC_COLORS=True,␣
→REC_ID=True)

# Potential Blocks
PotentialBlockVTKRecorder(..., REC_VELOCITY=True, REC_INTERACTION=True, REC_COLORS=True, REC_
→ID=True)
```

力链 和其他可视化输出默认可在 qt 中使用，也可以使用 v
tk 格式提取 the classic *VTKRecorder* orthe *export.VTKExporter* 类。

布尔参数 *twoDimension* 用于指定在 VTK 输出中将颗粒渲染为二维还是三维：

```
# Potential Particles
PotentialParticleVTKRecorder(..., twoDimension=False)

# Potential Blocks
PotentialBlockVTKRecorder(..., twoDimension=False)
```

此参数不应与 *Ip2_FrictMat_FrictMat_KnKsPBPhys.twoDimension*参数混淆；该参数用于定义如何计算接触力，具体说明见 引擎截面。

4.6.10 轴对齐包围盒

PP 和 PB 代码使用各自的 BoundFunctors，分别称为 *PotentialParticle2AABB* 和 *Potential-Block2AABB*，来定义每个颗粒的轴对齐包围盒。在这两种绑定函子中，都有一个布尔参数 *AabbMinMax*，允许用户选择近似的立方体 Aabb 或更精确的 Aabb。

具体而言，如果 `AabbMinMax=False`，则采用尺寸为 `1.0*R` 的立方体 Aabb。PP 和 PB 代码均实现了此功能，尽管 势块 没有球形项。在这种情况下，半径 R 用作参考长度，表示立方体 Aabb 对角线的一半。通常不建议使用这种近似的立方体 Aabb，因为它可能增加空接触的数量，从而延长进行近似接触检测所需的时间；同时，该方法依赖于半径 R，启用此选项时，该半径的取值应能包住整个颗粒。

如果使用 `AabbMinMax=True`，则可以定义更准确的 Aabb。目前，*PotentialParticle* 的初始 Aabb 必须由用户在颗粒局部坐标系中，针对颗粒的初始方向手动定义。为此，用户必须在形状类中手动指定 Aabb 的两个极值点：*minAabbRotated*、*maxAabbRotated*。另一方面，如果布尔参数*AabbMinMax* `=True`，则会根据颗粒的顶点自动计算并更新*PotentialBlock*的 Aabb。

如可视化小节所述，Aabb 的尺寸在实现颗粒渲染的代码中用作绘制空间（适用于 qt 环境中的 PP代码），也用于创建 vtk 格式的输出文件（适用于两种代码）。这是通过使用两个辅助参数实现的：*minAabb* 和*maxAabb*。对于势块代码，如果未给这些参数赋值，则会在*PotentialBlockVTKRecorder* 中使用颗粒的 Aabb 自动配置绘制空间。为了让现有代码能够在 qt 和 vtk 输出中将颗粒正确渲染为闭合表面，我们需要定义一个略大于实际空间的绘制空间。此处，绘制空间由颗粒的 Aabb 表示。因此，minAabb、maxAabb 与 minAabbRotated、maxAabbRotated 之所以有所区别，是因为需要满足以下两个条件：1. 用于主要接触检测的 Aabb 必须尽可能紧凑，以尽量减少空接触的数量；2. 用作渲染空间的 Aabb 必须略大一些，以确保正确渲染。如果用于可视化的 Aabb 某个维度小于实际尺寸，则颗粒该侧的面会被渲染为空心，只显示边；此功能可用于透视边界，例如 examples/PotentialParticles/cubePPscaled.py示例中的 vtk 输出所示。

回顾一下，在势粒子 代码中，*minAabbRotated* 和*maxAabbRotated* 参数定义了用于辅助主要接触检测的初始 Aabb，而*minAabb* 和*maxAabb* 参数则用于在 qt 和 VTK 输出中对颗粒进行可视化。在势块 代码中，用于辅助主要接触检测的 Aabb 会根据粒子顶点自动计算，这些顶点也用于在 qt 中进行可视化；而参数*minAabb* 和*maxAabb*用于在 VTK 输出中进行可视化，也可以不赋值，以触发*PotentialBlockVTKRecorder*中颗粒绘制空间的自动配置。

下面给出了两个简短示例，演示这些功能的语法。

对于势粒子代码：

```
b=Body()
b.shape=PotentialParticle(AabbMinMax=True,
                          minAabbRotated=Vector3(xmin,ymin,zmin),
                          maxAabbRotated=Vector3(xmax,ymax,zmax),
                          minAabb=Vector3(xmin,ymin,zmin),
                          maxAabb=Vector3(xmax,ymax,zmax),   ...)
```

用于势块 代码：

```
b=Body()
b.shape=PotentialBlock(AabbMinMax=True,
                       minAabb=Vector3(xmin,ymin,zmin),
                       maxAabb=Vector3(xmax,ymax,zmax),  ...)
```

4.6.11 块生成算法

该势块代码兼容于块体生成算法，该算法发表于 [Boon2015]，可根据颗粒与不连续面的交集对颗粒进行切分。该算法最初为研究岩体而开发。此代码以预处理器的形式硬编码于 YADE 中。使用同一数据结构来定义颗粒形状和不连续面，可以在合理的计算成本下生成大量颗粒。该方法采用顺序细分概念，并结合线性规划框架。通过引入更多约束，可以对非贯通节理进行建模。

用于演示此代码用法的示例位于examples/PotentialBlocks/WedgeYADE.py此脚本中使用的不连续面以 CSV 格式包含在 exam-ples/PotentialBlocks/joints/jointC.csv 中。

关于如何使用此代码的文档目前正在编写中。

4.6.12 示例

示例可在以下文件夹中找到：examples/PotentialParticles 和 examples/PotentialBlocks/，其中展示了代码的语法。

4.6.13 免责声明

这些代码是为学术目的开发的。由于原开发人员的博士论文历时多年，期间进行了大量试验，也出现了许多错误，因此有些变量已不再使用。鉴于这段代码依赖于 YADE 生态系统中的许多组件，建议用户谨慎使用。

4.6.14 引用

为致谢我们的科学贡献，请引用以下内容：

势块

- Boon CW（2013年）《节理岩体的离散元建模：算法及其验证》。哲学博士论文，牛津大学
- Boon CW、Houlsby GT、Utili S（2012年）离散元法中凸多边形和多面体颗粒之间的一种新接触检测算法。《计算机与岩土工程》，44：73-82

势粒子

- Houlsby GT（2009年）势颗粒：一种在离散元法中对非圆形颗粒建模的方法。《计算机与岩土工程》，36(6)：953-959
- Boon CW、Houlsby GT、Utili S（2013年）一种用于三维非球形颗粒的新接触检测算法。《粉体技术》，离散元法特刊，248：94-102

块体生成

- Boon CW，Houlsby GT，Utili S（2015年）一种基于线性规划的新岩石切片方法。《计算机与岩土工程》，65：12-29

第 5 章

性能增强

5.1 使用图形处理器加速 Yade 的 FlowEngine

（注：感谢 Robert Caulk 编写并分享本指南）

5.1.1 摘要

本文档介绍了如何为 Yade 的孔隙有限体积法（PFV）方案添加 Suite Sparse 的 GPU 加速，具体示例见 [Caulk2019]。本指南面向具有中级到高级水平的 Yade 用户，因此假定读者了解如何修改和编译 Yade 的源文件。读者将了解到本指南介绍了系统要求、必要先决条件的安装，以及修改版 Yade 的安装。最后，本文档展示了通过加速不同尺寸模型的因式分解所能实现的性能提升。

5.1.2 硬件、软件和模型要求

- **硬件：** – 支持 CUDA 的图形处理器，配备 >3 GB 内存（要求 64 MB）

- **软件：**

  – NVIDIA CUDA 工具包– SuiteSparse（CHOLMOD v2.0.0+）– Metis （随 SuiteSparse 提供） – CuBlas– OpenBLAS– LAPACK

- **模型：** – 流体耦合（孔隙有限体积法，即 Yade 的“FlowEngine”）– >10k 个颗粒，但要获得显著的速度提升，可能需要 >30k– 需要频繁重新划分网格

5.1.3 安装 CUDA

以下 CUDA 安装说明是对这些说明的精简版本 .

```
lspci | grep -i nvidia #Check your graphics card
# Install kernel headers and development packages
sudo apt-get install linux-headers-$(uname -r)
#Install repository meta-data (see **Note below):
sudo dpkg -i cuda-repo-<distro>_<version>_<architecture>.deb
sudo apt-get update  #update the Apt repository cache
sudo apt-get install cuda #install CUDA
# Add the CUDA library to your path
export PATH=/usr/local/cuda/bin${PATH:+:${PATH}}
export LD_LIBRARY_PATH=/usr/local/cuda/lib64\ ${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

注意：请使用此工具确定你的 `<distro>_<version>_<architecture>` 值。

重新启动计算机。

通过导航到 `/usr/local/cuda/samples` 并执行 `make` 命令，验证 CUDA 安装。现在，你可以导航到 `/usr/local/cuda/samples/1_Utilities/deviceQuery/` 并执行 `./deviceQuery` 。验证 `Result = PASS`。

5.1.4 安装 OpenBLAS 和 LAPACK

执行以下命令：

```
sudo apt-get install libopenblas-dev liblapack-dev
```

5.1.5 安装 SuiteSparse

下载 SuiteSparse 软件包 并将文件解压到 `/usr/local/`。运行 `make config` 并验证 `CUDART_LIB` 以及 `CUBLAS_LIB` 指向已安装的 CUDA 库。典型路径如下 `CUDART_LIB=/usr/local/cuda-x.y/lib64` 以及 `CUBLAS_LIB=/usr/local/cuda-x.y/lib64`。如果路径为空，可能需要导航到 `CUDA_PATH` 中的 `/usr/local/SuiteSparse/SuiteSparse_config/SuiteSparse_config.mk` 并手动修改，使其指向 CUDA 安装目录。返回 SuiteSparse 主文件夹并执行 `make`。现在，SuiteSparse 已在计算机上编译并安装完成。

通过导航到 `SuiteSparse/CHOLMOD/Demo` 并执行 `shgpu.sh`来测试 CHOLMOD 的图形处理器功能。注意：需要从 此处 下载 nd6k.mtx，并将其放入主目录。

5.1.6 编译 Yade

按照 此处所述说明，运行 `cmake` 并使用 `-DCHOLMOD_GPU=ON` 和 `-DSUITESPARSEPATH=/usr/local/SuiteSparse` （或其他自定义路径）。检查输出，确认 CHOLMOD（及 AMD 等依赖项）、SuiteSparse、CuBlas 和 Metis 的路径均被识别为安装这些软件包时创建的路径。以下是需要检查的输出示例：

```
-- Found Cholmod in /usr/local/SuiteSparse/lib/libcholmod.so
-- Found OpenBlas in /usr/lib/libopenblas.so
-- Found Metis in /usr/local/SuiteSparse/lib/libmetis.so
-- Found CuBlas in /usr/local/cuda-x.y/libcublas.so
-- Found Lapack in /usr/lib/liblapack.so
```

如果其中任何软件包有多个版本，系统可能会找到错误的版本。在这种情况下，你需要卸载旧库（例如 `sudo apt-get remove libcholmod`

如果另一个库是通过 apt-get 安装的），或者编辑 `cMake/Find_____.cmake` 中的路径。如果你将某个版本的 Cuda 安装在与 `/usr/local` 不同的位置，则需要在编译前编辑 `cMake/FindCublas.cmake`，以反映这些更改。

Metis 与 SuiteSparse 一起编译，因此 Metis 库和 Metis 包含目录应链接到 `usr/local/SuiteSparse/` 中的文件。准备就绪后，使用 `make -jX install` 完成安装。请注意，添加 `CHOLMOD_GPU` 会改变 `useSolver=4`，使其使用图形处理器而非中央处理器。若希望在中央处理器上使用 `useSolver=4` 且避免意外副作用（可能发生内存泄漏），建议使用 `CHOLMOD_GPU=OFF` 重新编译。当然，`useSolver=3` 应始终能在中央处理器上运行。

5.1.7 控制图形处理器

在 Yade 中，可以通过设置 `flow.useSolver=4`` 来启用图形处理器加速求解器。有几个环境变量可以控制允许使用的内存、允许的图形处理器矩阵大小等。CHOLMOD 用户指南中对此有详细说明，可在 `SuiteSparse/CHOLMOD/Doc` 中找到。至少，用户需要执行 `export CHOLMOD_USE_GPU=1` 来设置环境变量。如果希望使用 `multithread=True` 功能，还建议使用 `export CHOLMOD_-GPU_MEM_BYTES=3000000000` 指定可用图形处理器内存的一半（对于 6GB 显卡）。如果配置了多图形处理器，可以执行 `export CUDA_VISIBLE_DEVICES=1` 告诉 Yade 使用其中一个（或两个图形处理器，适用于 SuiteSparse-4.6.0-beta）；其中 1 表示要使用的图形处理器。

5.1.8 性能提升

[Catalano2012]展示了离散元法+PFV 耦合的性能，并指出了其优缺点。离散元法+PFV 耦合的一大优点是三角剖分成本、体积计算成本和力计算成本具有渐近特性（[Catalano2012], 图 5.4）。换言之，当颗粒数量超过 ~200k 后，额外增加的计算成本可以忽略不计。离散元法+PFV 耦合的主要缺点是，随着线性方程组规模不断增大，因式分解和求解的计算成本会呈指数级增长（[Catalano2012], 图 5.7）。如图 *fig-cpuvsgpu*所示，使用 Tesla K20 图形处理器，在 210 万个自由度和 35.6 万个颗粒的情况下，因式分解所需时间最多可减少 75%。

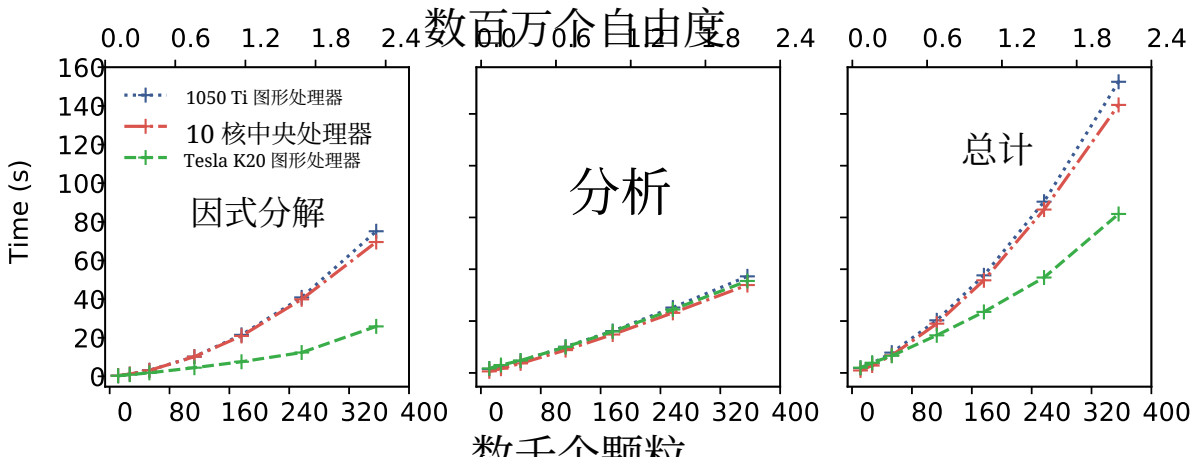


图 1：使用 10 核中央处理器、1050Ti 图形处理器和 Tesla K20 图形处理器对不同尺寸的矩阵进行因式分解和分析所需的时间 [Caulk2019]。

注意：Tesla K20 5GB 中央处理器 + 10-核心 Xeon E5 2.8 GHz 中央处理器

5.2 MPI 并行化

模块 *mpy* 通过域分解（分布式内存）实现并行化，使用由 OpenMPI 实现的消息传递接口（MPI）。它旨在通过在大量计算节点上运行 Yade 的独立实例来充分利用这些节点。共享内存和分布式内存方法彼此兼容，也就是说，可以同时使用两者运行混合任务；在某些情况下，这很可能是最佳求解方案。

大多数（最初是全部）对 OpenMPI 库的调用都在 Python 中通过 mpi4py 完成。不过，为了提高效率，一些关键通信会通过 C 语言的 Python 封装器++ 函数触发，其中会生成、发送/接收并处理消息。

该模块的开发始于 2018年。在一次高性能计算黑客马拉松期间，该模块获得了贡献。一项扩展支持与 *OpenFoam* 并行耦合。

注意： 另请参阅 *mpy* 模块的参考文档。

注意： 免责声明：尽管 *yade.mpy* 模块提供了 *mpirun* 函数，它看起来像是 *O.run()* 的简单替代品，但使用 mpy 设置模拟可能看似简单，实则不然。目前预计，一般而言，在任意脚本中简单地将“run”替换为“mpirun”不会提升速度，甚至可能彻底失败（未来或许会有所改进）。要理解其中原因并解决这些问题，掌握 MPI 工作原理的基础知识肯定会有所帮助（尤其是 mpi4py）。

5.2.1 概念

子域：域分解后分配给一个 MPI 进程的一组（子集）刚体，无论其是否具有空间连贯性。Yade 中对应的类是 *Subdomain*，这是一个 形状 实例，带有用于 MPI 通信的辅助函数。从某种意义上说，*Subdomain* 之于已订阅刚体，正如 *Clump*（另一种 形状）之于团簇成员。

秩：用于标识子域的索引，取值范围为 0 到 N-1（其中 N 为 MPI 进程数）。刚体所属子域的秩可通过 *Body.subdomain* 获取。并行执行期间，每个子域对应一个 Yade 实例和一个特定场景。场景的秩由 *Scene.subdomain* 给出。

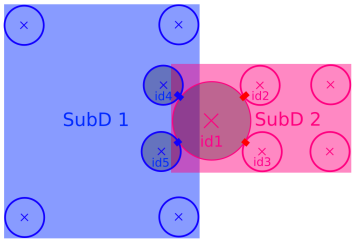
主进程：指秩为 $rank$ $=0$ 的子域。此子域的行为与其他子域不同。通常，主进程负责处理边界条件，并控制整个模拟的状态转换与终止。与标准子域不同，它可能不包含大量原始刚体（即，不包括墙或盒子等用于包围场景的对象）。交互式执行时，主进程是响应 Python 提示符的进程。

划分与合并：将主进程的场景切分为一组更小、分布式的场景称为“划分”。通过“合并”可撤销划分，将所有刚体以及（可选地）所有相互作用发送回主线程。先划分、运行，再合并，场景应与完全未使用 MPI 时相同（即与在单线程中执行相同迭代次数后的场景相同）。因此，此后正常调用 O.run() 应照常运行。

交集：子域中被其他子域包围盒相交的刚体子集（参见*fig-subdomains*）。*intersection(i,j)* 指当前（i）子域拥有且与子域j 相交的刚体（通过$O._sceneObj.subD.intersections[j]$获取）；*mirrorIntersection(i,j)* 指由j 拥有且与当前域相交的刚体（通过$O._sceneObj.subD.mirrorIntersections[j]$获取）。刚体按*Body.id*列出。根据定义，*intersection(i,j)=mirrorIntersection(j,i)*。

交集和镜像交集会作为并行碰撞检测的一部分自动更新。它们定义了需要进行通信的刚体状态。交集中的刚体需要被

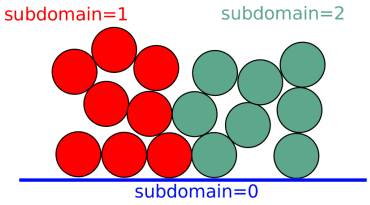
发送 到其他子域（实际上，每次迭代仅发送更新后的位置和速度），而镜像交集中的刚体需要从其他子域接收。



两个重叠的子域及其交集。在这种情况下，我们有*SubD1.intersections[SubD2.subdomain]=[id4,id5]* 和 *SubD1.mirrorIntersections[SubD2.subdomain]=[id1]*，其中 *SubD1* 和 *SubD2* 是 Subdomain 的实例。

5.2.2 操作说明

为了演示已实现并行算法的主要内部步骤，我们来看示例脚本examples/mpi/testMPI_2D.py。以三个 MPI 进程运行此脚本（交互模式或被动模式），会生成如*fig-scene-mpi*所示的场景。随后它会执行 *mpirun*，从而触发下文所述的步骤。

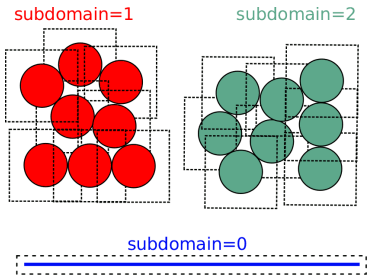


在此场景中，我们有三个 MPI 进程（三个子域），原始刚体划分在子域/进程秩 1 和 2 之间。拥有子域 =0 的主进程持有边界/墙类型刚体。刚体可以手动分配，也可以通过域分解算法自动分配。本文档后续章节将介绍域分解算法的详细信息。

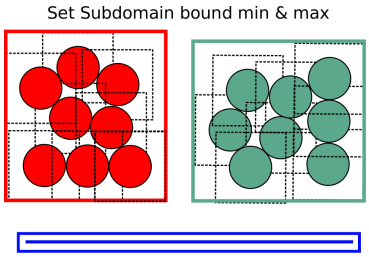
场景拆分:

在 MPI 执行开始时调用的函数 *mpy.splitScene*会将特定引擎静默添加到场景中，以处理接下来发生的事情。这种侵入性操作甚至会更改某些既有引擎的设置，尤其是 *InsertionSortCollider*，使其能够适应 MPI。 *InsertionSortCollider.verletDist* 是影响模拟效率的重要因子。其原因将在后续步骤中变得显而易见。

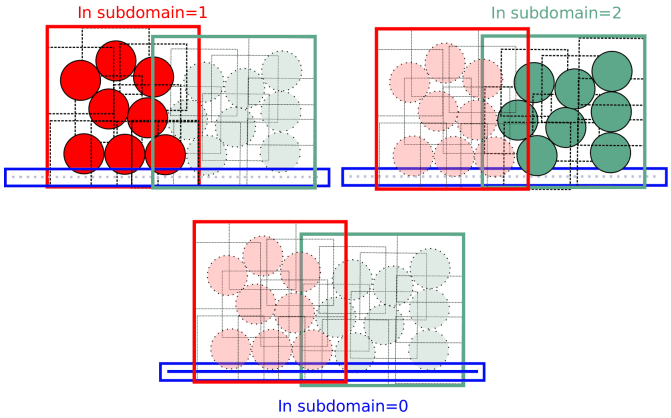
边界分发： 在下一步中， *Body.bound* 会与扩展后的 *Aabb*一起分发，如图 *fig-regularbounds* 所示（虚线）。请注意，子域的 *Subdomain Aabb*是通过取所属刚体的最小值和最大值获得的，参见图 *fig-sub DBounds*，其中子域的 *Aabb*以实色线条表示。此时，其他子域的最小值和最大值尚未知。



域边界更新： 常规刚体和本地子域的边界分派完成后，通过函数获取其他子域的边界信息

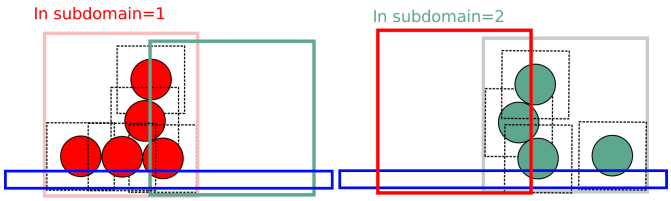


mpy.updateDomainBounds。在此集合通信中，每个子域都会向其他子域广播其 *Aabb.min*和 *Aabb.max* 。图 *fig-subdomain-bounds* 展示了一个示意图，其中每个子域都已接收到其他子域的 *Aabb.min* 和 *Aabb.max* 。



并行碰撞检测:

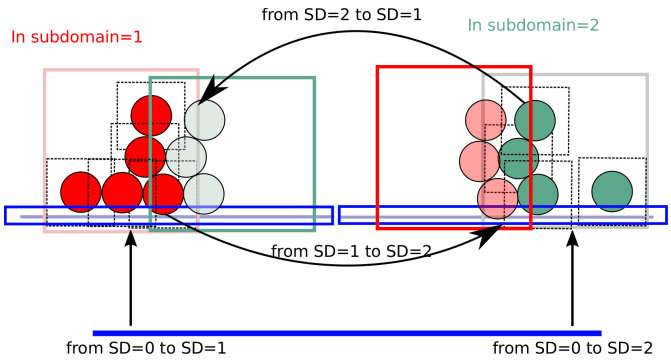
- 获得其他子域的 *Aabb.min* 和 *Aabb.max* 后，使用碰撞检测算法确定与远程子域存在交集的刚体。随后使用已识别刚体的标识符构建 *Subdomain.intersections* 列表。



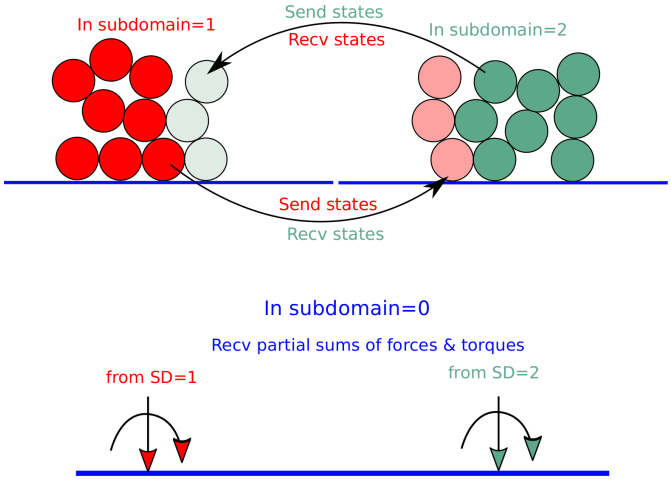
- 下一步是获取与当前子域相交的远程刚体的标识符（*Subdomain.mirrorIntersections*）。每个子域都会将本地刚体交集列表发送到相应的远程子域，同时接收来自其他子域的相交刚体标识符列表。如果当前子域的 *BodyContainer* 中不存在远程刚体，该子域便会向相应子域 请求 这些远程刚体。图 *fig-mirrorIntersections* 展示了这一操作的示意图，其中子域 =1 从子域接收三个刚体 =2，并从子域接收一个刚体=0。子域=2 从子域接收三个刚体=1。子域=0 只发送自己的刚体，不会 从工作进程子域接收刚体。此操作为与其他子域之间传输刚体状态做好了准备。

更新状态:

一旦确定了子域、相关的相交刚体以及远程刚体，这些刚体的状态便会在每个时间步长通过相互作用子域之间的点对点通信进行发送和接收。如果与主子域（子域=0）发生相互作用，则只发送给定子域施加在主进程刚体上的总力和力矩。图 *fig-sendRecvStates*



展示了一个示意图，其中子域=1 与子域=2之间的远程刚体状态通过通信进行交换。子域=0 从子域=1 和子域=2接收力和力矩。



5.2.3 MPI 初始化与通信

mpy 模块试图保留 Yade 最重要的特性之一：交互式访问场景（或此处的多个场景）中的对象，具体说明如下。交互式执行不会使用 OpenMPI 的*mpiexec* command，而是在 Yade 启动后由 mpy 模块生成一个工作进程池。在生产环境中，可以使用被动作业；此时，*mpiexec* 会在调用 Yade 前执行。

注意： 本页中的大多数示例都使用 4 个 MPI 进程。原则上，即使可用核心数少于 4 个，仍然可以运行这些示例（这称为过度订阅，不过也可能因操作系统和 MPI 实现而失败）。过度订阅不会带来性能提升，但对于实验很有用（例如，在单核计算机上测试本页中的示例）。

交互模式

交互模式主要用于在 MPI 执行后检查模拟，以便进行调试。这里介绍的函数（尤其是 *sendCommand*）在一般情况下也很有用，可用于完成更高级的任务，例如控制模拟不同阶段之间的转换，以及收集和处理结果。

从 Python 提示符显式初始化

如下所示，可以使用 mpy.initialize() 创建一组 Yade 实例。请注意，接下来的命令序列应在启动 Yade 后直接输入到 Python 提示符中，

如果将其粘贴到由 Yade 执行的脚本中，结果将不会完全相同（请参阅下一节“自动初始化”）：

```
@suppress
Yade [1]: from yade.utils import *

@suppress
Yade [1]: O.engines=yade.utils.defaultEngines

Yade [2]: wallId=O.bodies.append(box(center=(0,0,0),extents=(2,0,1),fixed=True))

Yade [3]: for x in range(-1,2):
   ...:    O.bodies.append(sphere((x,0.5,0),0.5))
   ...:

Yade [5]: from yade import mpy as mp

@suppress
Yade [5]: mp.COLOR_OUTPUT=False

@doctest
Yade [6]: mp.initialize(4)
Master: I will spawn  3  workers
->  [6]: (0, 4)
```

执行 mp.initialize(np) 后，Yade 的父实例将担任主进程（秩为 =0）。它是唯一一个会执行直接在提示符中键入的命令的实例。其他实例（秩从 =1 到 =np-1）处于空闲状态，并等待主进程发送命令。可以使用 *mpy.sendCommand()* 向其他实例发送命令；默认情况下，该命令会返回结果或结果列表。下面我们使用该命令来验证生成的工作进程指向不同的（仍为空的）场景：

```
Yade [8]: len(O.bodies)
 ->  [8]: 4

Yade [10]: mp.sendCommand(executors="all",command="len(O.bodies)",wait=True) #check content
->  [10]: [4, 0, 0, 0]

Yade [9]: mp.sendCommand(executors="all",command="str(O)") # check scene pointers
->  [9]: ['<yade.wrapper.Omega object at 0x7f9c0a399490>',
'<yade.wrapper.Omega object at 0x7f9231213490>',
'<yade.wrapper.Omega object at 0x7f20086a1490>',
'<yade.wrapper.Omega object at 0x7f622b47f490>']
```

发送命令可以通过调用底层的 mpi4py（参见 mpi4py 文档）来管理全部类型的消息传递。请注意，sendCommand 默认会阻塞。若不设置“wait=False”，下一个示例将会挂起，因为主进程和工作进程都会等待对方发送消息。

```
Yade [3]: mp.sendCommand(executors=1,command="message=comm.recv(source=0); print('received',
→message)",wait=False)

Yade [4]: mp.comm.send("hello",dest=1)
received hello
```

每个可序列化的 Python 对象（即几乎所有 Yade 对象）都可以通过这种方式传输。下文请注意使用 *mpy.mprint*（通过数量和字体颜色标识工作进程）。还要注意，通过 *sendCommand* 传递的命令是在 mpy 模块的上下文中执行的，因此*comm*、*mprint*、*rank* 以及该模块的全部对象，都可以在不使用 *mp.* 前缀的情况下访问。

```
Yade [3]: mp.sendCommand(executors=1,command="O.bodies.append(comm.recv(source=0))",
→wait=False) # leaves the worker idle waiting for an argument to append()
```

（下页续）

（接上一页）

从 Python 脚本显式初始化

虽然 sendCommand() 函数对高级操作很有用，但它也有局限性。Python 语言的一些基本功能无法使用，例如，函数定义和循环就无法正常工作——事实上，任何无法写在单行中的代码片段都不行。实际上，mpy 模块提供了一种通过脚本进行初始化的机制，可以在脚本中声明函数和变量。

每当 Yade 启动时带有一个脚本作为参数，系统就会记住该脚本的名称；如果调用 mpy.initialize()（由脚本自身调用，或在提示符下交互式调用），所有 Yade 实例都会使用同一个脚本进行初始化。这样一来，分发函数定义和仿真参数就变得轻而易举（也可以分发场景构造，如下所示）。

MPI 通常就是这种行为：所有进程执行同一个程序。使用“mpiexec -np N yade ...”时也是如此。

如果将上面的首批命令粘贴到用于启动 Yade 的脚本中，所有工作进程都会像主进程一样插入相同的刚体（而在交互式执行时，只有主进程会插入刚体）。以下是该脚本：

```
# script 'test1.py'
wallId=O.bodies.append(box(center=(0,0,0),extents=(2,0,1),fixed=True))
for x in range(-1,2):
        O.bodies.append(sphere((x,0.5,0),0.5))
from yade import mpy as mp
mp.initialize(4)
print( mp.sendCommand(executors="all",command="str(O)",wait=True) )
print( mp.sendCommand(executors="all",command="len(O.bodies)",wait=True) )
```

输出如下：

```
yade test1.py.....Running script test1.pyMaster: will spawn  3  workersNoneNoneNoneNoneNone
None['<yade.wrapper.Omega object at 0x7feb979403a0>', '<yade.wrapper.Omega object at␣
→0x7f5b61ae9440>',  '<yade.wrapper.Omega object at 0x7fdd466b8440>',
 '<yade.wrapper.Omega␣→object at 0x7f8dc7b73440>'][4, 4, 4, 4]
```

这是因为全部实例都会在初始化()相位执行脚本。由于脚本中有两处*print(mp.sendCommand(...))*，所以会打印 2x3 次“None”；工作进程也会尝试执行这段代码，但是

对于 m *sendCommand* 默认会返回，因此 无。

这个结果在逻辑上说得通，但如果我们要把一次模拟拆分成若干部分，这就不是我们想要的结果。解决方法（所有 MPI 程序的典型做法）是在条件语句中使用进程的 秩 。这样，脚本的不同部分便可根据各工作进程的秩，由各自以不同方式执行。例如，为了得到与之前相同的结果，可以按如下方式修改脚本：

```
# script 'test2.py'
from yade import mpy as mp
mp.initialize(4)
if mp.rank==0: # only master
        wallId=O.bodies.append(box(center=(0,0,0),extents=(2,0,1),fixed=True))
        for x in range(-1,2):
        O.bodies.append(sphere((x,0.5,0),0.5))

        print( mp.sendCommand(executors="all",command="str(O)",wait=True) )
        print( mp.sendCommand(executors="all",command="len(O.bodies)",wait=True) )
        print( mp.sendCommand(executors="all",command="str(O)",wait=True) )
```

结果如下：

```
Running script test2.pyMaster: will spawn  3  workers
['<yade.wrapper.Omega object at 0x7f21a8c8d3a0>', '<yade.wrapper.Omega object at␣
→0x7f3142e43440>', '<yade.wrapper.Omega object at 0x7fb699b1a440>', '<yade.wrapper.Omega␣
→object at 0x7f1e4231e440>'][4, 0, 0, 0]
```

我们也可以使用 秩 将来自不同空间区域的刚体分配给不同的工作进程，如示例 examples/mpi/helloMPI.py中所示，其中位置取决于秩：

```
# rank is accessed without "mp." prefix as it is interpreted in mpy module's scope
mp.sendCommand(executors=[1,2],command= "ids=O.bodies.append([sphere((xx,1.5+rank,0),0.5) for␣
 →xx in range(-1,2)])")
```

请注意，调用 *mp.initialize(N)* 相对于其他命令的位置不会影响工作进程的执行（对它们而言，initialize() 只会返回），因此程序逻辑不应依赖于此。工作进程会在相同的 MPI 上下文中从头到尾执行脚本，而该上下文在执行第一行时就已设置。这可能会导致违反直觉的行为，下面是一个脚本：

```
# testInit.py
# script.py
O.bodies.append([Body() for i in range(100)])

from yade import mpy as mp
mp.mprint("before initialize: rank ", mp.rank,"/", mp.numThreads,"; ",len(O.bodies)," bodies")
mp.initialize(2)
mp.mprint("after initialize: rank ", mp.rank,"/", mp.numThreads,"; ",len(O.bodies)," bodies")
```

以及输出：

```
Running script testInit.py
Master: before initialize: rank  0 / 1 ;  100  bodies
Master: will spawn  1  workers
Master: after initialize: rank  0 / 2 ;  100  bodies
Worker1: before initialize: rank  1 / 2 ;  100  bodies
Worker1: after initialize: rank  1 / 2 ;  100  bodies
```

mpirun（自动初始化）

要根据前面介绍的命令有效地运行分布式离散元模拟，会非常繁琐。因此，mpy 模块提供了 *mpy.mpirun* 函数，可自动完成大部分步骤，如简介中所述。主要包括：根据分配给刚体的秩将场景划分为子域，并随着时间积分的推进处理子域之间的碰撞（其中包括大幅更改引擎列表，以完成所有这些工作）。

如有需要，首次运行 mpirun 时会调用 initialize() 函数，因此用户端可以省略此调用。迭代结束时，子域会根据参数*withMerge*合并到主进程上的集中式场景中。

下面是一个具体示例：将地板分配给主进程，并将多个球体组分配给子域：

```
import os
from yade import mpy as mp

NSTEPS=5000 #turn it >0 to see time iterations, else only initilization
numThreads = 4 # number of threads to be spawned, (in interactive mode).

#materials
young = 5e6
compFricDegree = 0.0
O.materials.append(FrictMat(young=young, poisson=0.5, frictionAngle = radians(compFricDegree),␣
→density= 2600, label='sphereMat'))
O.materials.append(FrictMat(young=young*100, poisson = 0.5, frictionAngle = compFricDegree,␣
→density =2600, label='wallMat'))


#add spheres

mn, mx=Vector3(0,0,0), Vector3(90,180,90)
pred = pack.inAlignedBox(mn,mx)
O.bodies.append(pack.regularHexa(pred,radius=2.80,gap=0, material='sphereMat'))

#walls (floor)

wallIds=aabbWalls([Vector3(-360,-1,-360),Vector3(360,360,360)],thickness=10.0, material=
→'wallMat')
O.bodies.append(wallIds)

#engines
O.engines=[
        ForceResetter(),
        InsertionSortCollider([
                Bo1_Sphere_Aabb(),
                Bo1_Box_Aabb()], label = 'collider'), # always add labels.
        InteractionLoop(
                [Ig2_Sphere_Sphere_ScGeom(),Ig2_Box_Sphere_ScGeom()],
                [Ip2_FrictMat_FrictMat_FrictPhys()],
                [Law2_ScGeom_FrictPhys_CundallStrack()],
                label="interactionLoop"
        ),
        GlobalStiffnessTimeStepper(timestepSafetyCoefficient=0.3, timeStepUpdateInterval=100, ␣
→parallelMode=True, label = 'timeStepper'),
        NewtonIntegrator(damping=0.1,gravity = (0, -0.1, 0), label='newton'),
        VTKRecorder(fileName='spheres/3d-vtk-', recorders=['spheres', 'intr', 'boxes'], ␣
→parallelMode=True, iterPeriod=500), #use .pvtu to open spheres, .pvtp for ints, and .vtu for␣
→boxes.
]
```

（下页继续）

（接上一页）
```
#set a custom verletDist for efficiency
collider.verletDist = 1.5
```

随后执行该脚本：

```
yade script.py
```

F若要继续运行更多时间步，则必须在 Yade 提示符下执行 mp.mpirun command：

```
Yade [0]: mp.mpirun(100,4,withMerge=False) #run for 100 steps and no scene merge.

Yade [1]: mp.sendCommand([1,2],"kineticEnergy()",True) # get kineticEnergy from workers 1 and
→2.

Yade [2]: mp.mpirun(1,4,withMerge=True) # run for 1 step and merge scene into master. Repeat
→multiple time to watch evolution in QGL view
```

非交互式执行

除了在启动 Yade 后生成 MPI 进程外，还可以使用 OpenMPI 中经典的“mpiexec”来运行 Yade。重要的是，这可能是高性能计算作业提交系统唯一允许使用的方法。使用 mpiexec 时，没有交互式 shell，或者交互式 shell 无法正常工作（通常在生产环境中这没问题）。作业需要自行运行（或“*mpirun*”）并终止。

上文所述的函数 初始化 和 *mpirun* 能够透明地处理交互式和被动执行，用户脚本在这两种情况下的行为应该相同。之所以说“应该”，是因为启动时后台发生的情况并不完全相同，有时可能会显现出来（请告知我们）。

如果脚本调用 *mpy.mpirun* 并指定时间步数，模拟（例如参见 exam-ples/mpi/vtkRecorderExample.py）将通过以下命令执行：

```
mpiexec -np NUMSUBD+1 yade vtkRecorderExample.py
```

其中 *NUMSUBD* 对应所需的子域数量。

注意： 请记住，主进程在不处理普通子域时仍计数为 1，因此进程数量始终为$NUMSUBD$ $+1$。

5.2.4 划分

将初始场景划分为子域，以及在颗粒运动后更新子域，是影响效率的两个关键问题。分解方式可以由用户指定（见下方第一节），但 mpy 模块也为这两者提供了算法。

注意： mpy 模块对如何定义子域没有任何要求，也不必使用此处介绍的辅助函数。即使将球体云中的刚体随机分配到若干子域中（使子域彼此重叠，并且都覆盖整个场景），也能正常工作。只是这样做并非最优，因为与合理的空间划分相比，子域之间的相互作用数量会增加。

自行划分

要强制进行分解，只需为 *Body.subdomain* 赋予一个值，该值对应于它应归属的进程秩。这既可以在之后再进行划分的集中式场景中完成，也可以在每个子域中分别插入相应的刚体子集（参见 场景构建）

在示例脚本 examples/mpi/testMPI_2D.py 中，球体按如下方式生成（此处为集中式构建，很容易改为分布式构建）。对于每个可用的工作进程，都会生成一组位于不同空间位置的球体。每组中的球体都会被分配一个子域秩（以及用于可视化的颜色），以便 mpirun() 运行后由相应的工作进程接管：

```
for sd in range(0,numThreads-1):col = next(colorScale)ids=[]
for i in range(N):#(numThreads-1) x N x M spheres, one thread is for master and will␣
→keep only the wall, others handle spheresfor j in range(M):
id = O.bodies.append(sphere((sd*N+i+j/30.,j,0),0.500,color=col)) #a␣
→small shift in x-positions of the rows to break symmetryids.append(id)
for id in ids: O.bodies[id].subdomain = sd+1
```

不知道如何划分？交给 mpirun 吧

初始划分

如果*DO-MAIN_DECOMPOSITION* =为 True，mpirun 将自行决定如何将刚体分配到多个子域。在这种情况下，顺序脚本与其 MPI 版本之间的差异仅限于导入 mpy，并在启用 *DO-MAIN_DECOMPOSITION* 标志后调用 mpirun。

刚体自动划分到子域所依据的算法是 Berger 提出的正交递归二分算法 [Berger1987], 和 [Fleissner2007]。该分区通过在多个层级上对空间进行二分来实现，每个层级中最长的轴被选作二分轴。层级数量确定为 $\mathrm{int}(\log_2(\mathrm{N}_w))$ 其中 N_w 表示工作进程子域的数量。该分解过程的示意图见 *fig-bisectionAlgo*，其中有 4 个工作进程子域。在初始阶段（层级 $= 0$），我们假设子域=1 包含刚体位置（以及刚体）信息。首先确定最长轴，并以此作为二分轴／平面。沿二分轴对包含刚体位置的列表进行排序，然后确定排序后列表的中位数。位置（二分坐标）小于中位数的刚体以当前子域的颜色标记（SD=1），另一半则以 SD $= 2$的颜色标记。每个层级的子域着色按以下规则确定：

```
if (subdomain <  1<<level) : this subdomain gets the bodies with position lower than␣
→the median.
if ((subdomain >  1<<level) and (subdomain <  1<<(level+1) ) ) : this subdomain gets␣
→the bodies with position greater than median, from subdomain - (1<<level)
```

此过程将持续进行，直到达到层数。

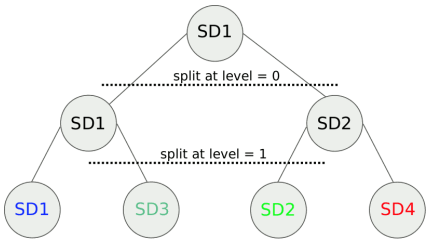
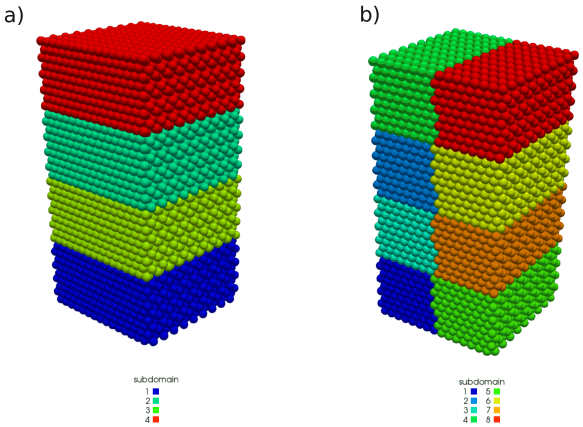


图 *fig-domainDecompose* 展示了使用 ORB 算法得到的分区结果：（a）4 个子域，（b）8 个子域。当前实现也支持工作进程子域数量为奇数的情况。



当前实现可在 py/bisectionDecomposition.py中找到，并行版本可在此处找到。

注意： 导入py/bisectionDecomposition.py会触发对*mpi4py* 的导入，最终还会导入 MPI 库及相关环境变量。 *mpy* 模块在导入时可能会更改某些 MPI 设置，因此最好在某些 `mpy.initialize()`之后再导入*bisectionDecomposition* 。

更新分区（负载均衡）

随着刚体移动，每个子域都可能发生整体变形，刚体也可能扩散，从而导致多个子域之间出现不理想的重叠。这种子域重叠会降低 MPI 工作进程之间通信的效率，因此我们希望通过一种算法尽可能保持子域紧凑。该算法会动态地将刚体重新分配到新的子域，以尽量减少 MPI 通信。该算法利用 *InsertionSortCollider* 高效地重新分配刚体，并与碰撞检测同步进行；当 $mpy.REALLOCATE_FREQUENCY > 0$。

该算法并非 集中式，因此能够保持可扩展性。此外，它只在具有交集的 MPI 工作进程之间进行点对点通信。重新分配依赖于用于本地决策的筛选器。目前有一个可用的筛选器，名为*mpy.medianFilter*。也可以改用自定义筛选器。

中值筛选器的刚体重新分配准则涉及查找两个子域之间中值平面的位置，使得将刚体划分到该平面的“+”侧和“-”侧之后，每个子域中的刚体总数仍保持不变。由此产生的视频中展示了这种划分类型。尽管在视频的某些时刻，中值平面似乎旋转得相当快，但实际上每次重新分配之间都会进行五次碰撞检测，也就是说，需要经过数千次时间迭代，才能有效地将划分从一种颜色旋转到另一种颜色。这种渐进式旋转很有益处，因为否则初始划分会形成扁平圆盘。

注意： 这里的负载均衡并非指让每个核心承担相同的工作量。事实上，只要为每个工作进程分配相同数量的刚体，按定义就已经实现了这种均衡（而且子域归根结底是刚体列表，并非空间中的特定区域）。这里的目标是减少整体通信时间。

集中式与分布式场景构建

对于集中式场景构建方法，主进程会创建场景中的所有刚体，并为它们分配子域。作为 mpy 初始化的一部分，某些引擎会被修改或插入，随后场景会被广播给工作进程。每个工作进程都会接收整个场景，通过 *Body.subdomain* 识别分配给自己的刚体（如果工作进程的 `rank==b.subdomain` 保留了这些刚体），并删除其余刚体。前面的示例采用了这种场景构建方式，这是迄今为止最简单的方法。与构建用于非 MPI 执行的场景相比，它实际上没有区别，只需调用 *mp.mpirun* 而不是*O.run*即可。

不过，对于大量刚体和进程，集中式场景构建与分发可能会耗费大量时间。由于内存使用量呈平方级增长，这种方式也可能受内存限制：假设 32 核节点上每个线程有 N 个刚体，那么集中式构建意味着在某个时刻（拆分期间），内存中会同时存在整个场景的 32 份副本，即在单个节点上有 32^2N 个刚体。对于大规模并行应用，应优先采用分布式构建。

在分布式模式下，每个工作进程都会实例化自己的刚体，并将其插入本地 *BodyCon-tainer*。必须注意正确分配刚体标识符，因为初始时不应有两个不同的工作进程拥有相同的索引。使用指定标识符向 *BodyContainer* 插入刚体，可通过 *BodyCon-tainer.insertAtId* 完成。设置 `DISTRIBUTED_INSERT` 标志即可启用分布式模式，用户负责设置子域并对刚体进行分区。分布式插入示例参见 examples/mpi/parallelBodyInsert3D.py。

相关代码片段如下，其中通过跳过循环中的所有步骤（只保留秩正确的那个步骤）来进行过滤（请注意，所有工作进程都会运行同一个循环，但它们各自的秩都不同）：

```
#add spheres
subdNo=0
import itertools
_id = 0 #will be used to count total number of bodies regardless of subdomain attribute, so␣
→that same ids are not reused for different bodies
for x,y,z in itertools.product(range(int(Nx)),range(int(Ny)),range(int(Nz))):
        subdNo+=1
        if mp.rank!=subdNo: continue
        ids=[]
        for i in range(L):#(numThreads-1) x N x M x L spheres, one thread is for master and␣
→will keep only the wall, others handle spheres
                for j in range(M):
                        for k in range(N):
                                dxOndy = 1/5.; dzOndy=1/15.  # shifts in x/y-positions to make␣
→columns inclines
                                px= x*L+i+j*dxOndy; pz= z*N+k+j*dzOndy; py = (y*M+j)*(1 -
→dxOndy**2 -dzOndy**2)**0.5 #so they are always nearly touching initialy
```

（续见下页）

（接上页）

```
                                id = O.bodies.insertAtId(sphere((px,py,pz),0.500),_
→id+(N*M*L*(subdNo-1)))
                                _id+=1
                                ids.append(id)
        for id in ids: O.bodies[id].subdomain = subdNo

if mp.rank==0: #the wall belongs to master
        WALL_ID=O.bodies.insertAtId(box(center=(Nx*L/2,-0.5,Nz*N/2),extents=(2*Nx*L,0,2*Nz*N),
→fixed=True),(N*M*L*(numThreads-1)))
```

二分算法可用于定义初始划分，也适用于分布式情形，因为它以点数据集作为输入。只要所有工作进程使用相同的数据集（例如，使用随机数生成器生成相同的序列），它们最终就会得到相同的分区，并可据此实例化各自的刚体。

5.2.5 合并

可选参数可以实现上例所示的“合并”功能，该参数可用于 *mpirun*，也可以使用独立函数 *mpy.mergeScene*。

如果在 mpirun 中将 withMerge=设为 True，主场景中的刚体便会更新，以反映其分布式副本的演化。mpirun 在完成所需迭代次数后会执行一次此操作。此合并操作还可以包含相互作用的更新。*mpy.mergeScene* 则会在当前迭代中执行相同操作。合并是一项开销很大的任务，需要传输大量消息，因此应审慎执行，并保持适当的频率。对于大规模并行场景，合并甚至可能成为主要瓶颈。尽管如此，它仍可用于通过三维视图进行调试，或执行各种后处理任务。MERGE_W_INTERACTIONS 用于执行完整合并，即包含工作进程子域内及子域之间的相互作用；否则，仅使用刚体的位置和状态。包含相互作用的合并应生成常规的 Yade 场景，可在非 MPI 模式下继续进行时间步进，或（更实用地）用于某些后处理。通常，正确进行时间积分并不需要合并操作。

5.2.6 提示与预期问题

引擎中的 MPI 支持

对于 MPI 算例，必须启用 *parallelMode* 标志，以便 *GlobalStiffnessTimeStepper* 和 *VTKRecorder* 运行。目前，只有这两个引擎已升级为支持 MPI。

至于其他方面，请阅读下一节并谨慎操作。如果你想为其他引擎实现 MPI 支持，那就太好了；可以参考现有的两个示例。请告诉我们！

归约（部分和）

像动能这样的物理量，不能仅通过对每个子域返回的 *kineticEnergy()* 求和来获取整个场景的值。这是因为每个子域还可能包含来自相交子域的刚体图像，这些图像的速度、质量或其他待求和的量也可能被计入每个工作进程的返回值。尽管 Yade 中一些最常用的函数可能会逐步获得 MPI 支持，以筛除来自远程域的刚体，但目前这还不是标准做法。因此，可能需要针对具体情况实现部分和，并在用户脚本中进行适当的筛选。

这只是一个例子，说明如果在复杂脚本中直接将 *run* 替换为 *mpirun*，可能会导致许多问题。

其他

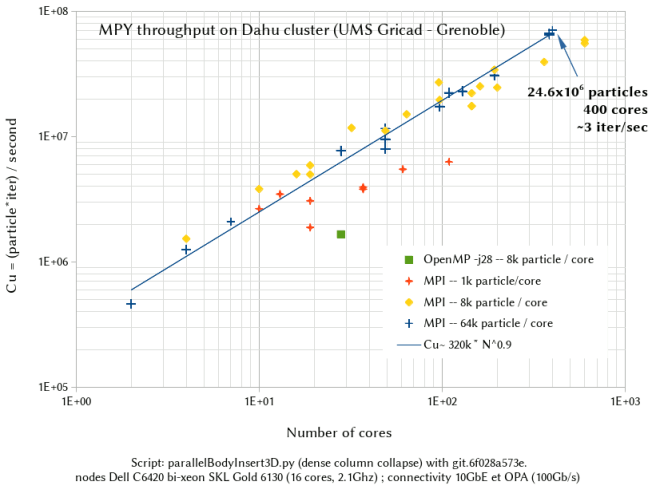
- sendCommand() 的延迟被硬编码为 0.001 秒，以免所有核心在等待 command 时始终处于 100% 忙碌状态（可能只给操作系统留下少量资源）。如果在复杂算法中以高频率调用 sendCommand()，缩短这段休眠时间可能会有所帮助。

5.2.7 控制变量

•详细输出_输出内容：列出每个操作/步（例如*mpy.splitScene*、*mpy.parallelCollide* 等）的详细信息会打印到控制台，便于调试

- 累积_力：控制主进程所拥有刚体上的力求和。
- 擦除_远程_主进程：擦除主子域中的远程刚体，还是将其保留为无界对象？这对快速合并很有用。
- OPTIMIZE_COM, USE_CPP_MPI：使用子域 类中的优化通信函数和 MPI 函数
- YADE_TIMING：报告计时统计信息，输出通信、碰撞检测及其他操作所用的时间。
- DISTRIBUTED_INSERT：由各子域创建并插入刚体，用于分布式场景构建。
- DOMAIN_DECOMPOSITION：如果为真，则使用二分分解算法将刚体分配给工作进程/子域。
- MINIMAL_INTERSECTIONS：缩减位置/速度通信的数据量（在碰撞阶段结束时，可以从交集中排除那些除刚体<->子域之外没有相互作用的刚体）。
- REALLOCATE_FREQUENCY：如果为 >0，则会在子域之间迁移刚体，以实现高效负载均衡。如果为 $=1$，则每次触发碰撞器时都会进行重新分配；否则，每经过 N 次碰撞检测后进行一次。
- REALLOCATE_MINIMAL：重新分配之前会先最小化交集，从而减少重新分配的刚体数量
- USE_CPP_REALLOC：使用优化的 C++ 函数执行刚体重新分配
- 流体_耦合：与 OpenFOAM 耦合的标志。

5.2.8 基准测试



注释：

- 从每个核心 1k 个颗粒增加到 8k 个颗粒，性能有明显提升。显然，1k 太少了，大部分时间都花在通信上。
- 从每核心 8k 增至 64k 时，单核心吞吐量大致相同，性能也与线性增长相差不远。数据已排除随机噪声的影响；总体而言，我无法确定这些非线性分别来自代码还是硬件。

- 结论是，如果每核心没有至少 8k 个球体（复杂形状可能更少），MPI 就不适合你。这与 Dion Weatherley（DEM8+beer）估算的 10k 一致。
- 看起来 OpenMP 表现很差，但请注意，该基准测试脚本针对 MPI 做了大量调优。它采用了很大的 verletDist，并在虚拟相互作用上耗费了更多时间，以尽量减少全局更新。
- 我认为，针对 OpenMP 进行调优后，对于不足一百万个颗粒的情况，-j26（或 2xMPIx -j13）可能达到甚至超过 26 个 MPI 线程的性能。考虑到额外的难度，MPI 更适用于超过一百万个颗粒，或使用多个计算节点的情况。

- 名义上的单核吞吐量并不出色。运行高效脚本时，我的笔记本电脑可接近 1e6Cu，而在 Dahu 上每个核心只能达到 0.3e6Cu。这并不是 MPI 的问题；即使只使用单个核心，我的笔记本电脑的表现也会优于 Dahu。

5.3 在亚马逊 EC2 上使用云计算运行 YADE

（注：感谢 Robert Caulk 编写并分享本指南）

5.3.1 摘要

本指南旨在帮助 YADE 用户将模拟迁移到亚马逊云科技（AWS）EC2。使用可扩展的云计算运行 YADE，最显著的两项优势是降低前期成本和提高性能。从启动实例、安装 YADE，到在球体云上运行 YADE 仿真，整个过程可在 5 分钟内完成。EC2 实例运行后，您可以像在本地工作站上提交作业一样提交 YADE 脚本。

5.3.2 启动 EC2 实例

首先登录 亚马逊 EC2控制台。您需要使用现有或新建的 Amazon 账户。登录后，点击 AWS 主页左上角的“服务”，即可找到 EC2 控制台。首先点击 `launch an instance` 蓝色按钮（图 *fig-console*）。选择亚马逊机器映像（AMI）： `Ubuntu Server 16.04 LTS` （图 *fig-ubuntu*）。

接下来，选择实例类型。建议查看 各实例的规格，以便为 YADE 仿真选择合适的计算能力。本文档不会详细介绍如何选择尺寸，但您可以查阅许多 YADE 专用性能报告，以帮助您做出决定。不过，实例类型是一项重要的选择。对于大多数 YADE 模拟， `Compute Optimized` 实例都是必需的，因为它们可提供高性能处理器和有保障的计算能力。C3.2xlarge（图 *fig-type*）相当于配备 25 MB 缓存的 8 核 2.8 GHz Xeon E5，很可能是进行中大型 YADE 模拟的最佳选择。

启动前，系统会要求你 `select an existing key pair or create a new key pair`。请创建一个新文件，下载它，并将其放在一个你知道路径的文件夹中。通过在终端中导航到同一目录并输入以下命令来修改文件权限：

```
chmod 400 KeyPair.pem
```

现在实例已启动，你需要通过 SSH 连接到它。在 Unix 系统上，只需输入以下命令即可：

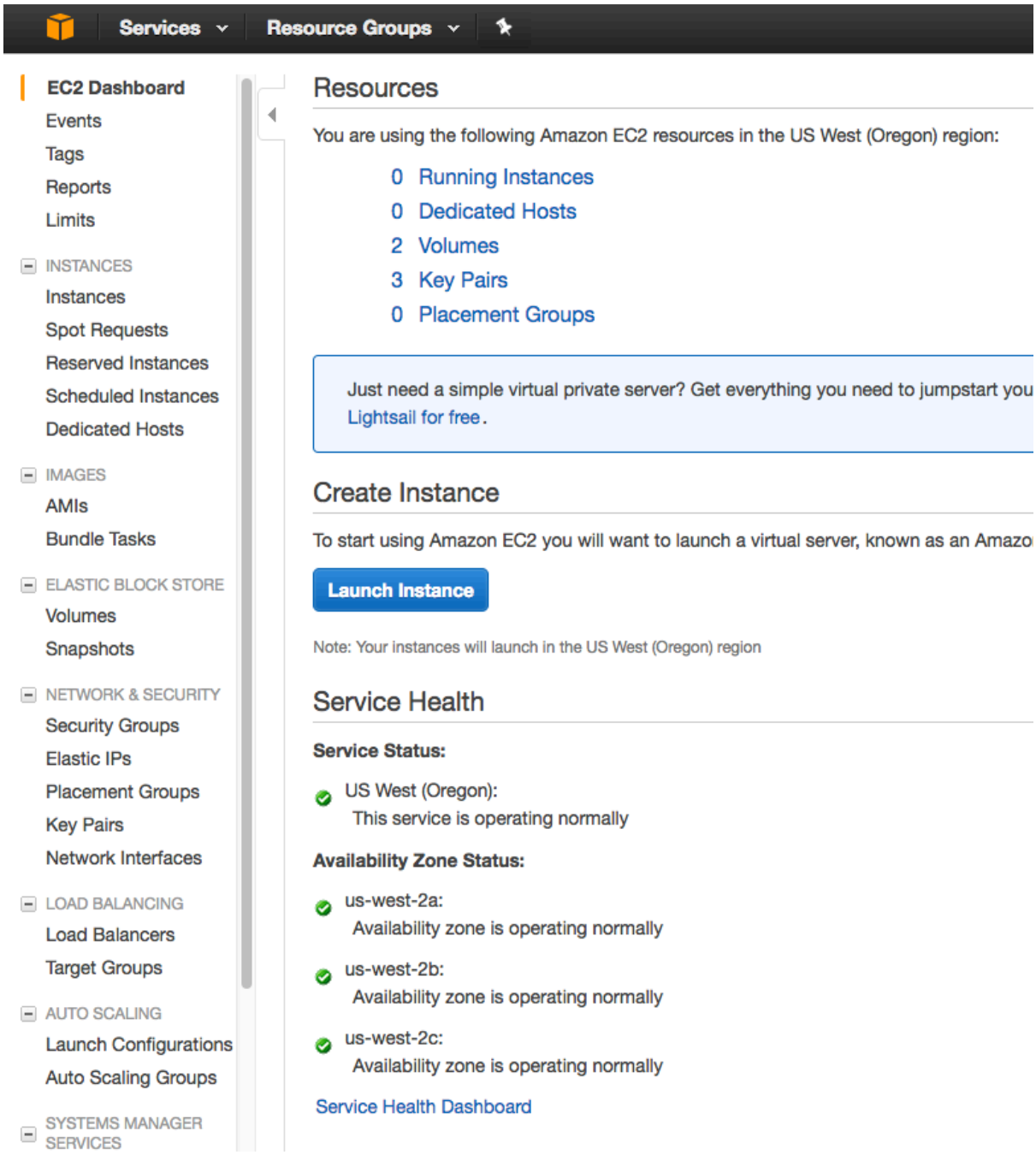
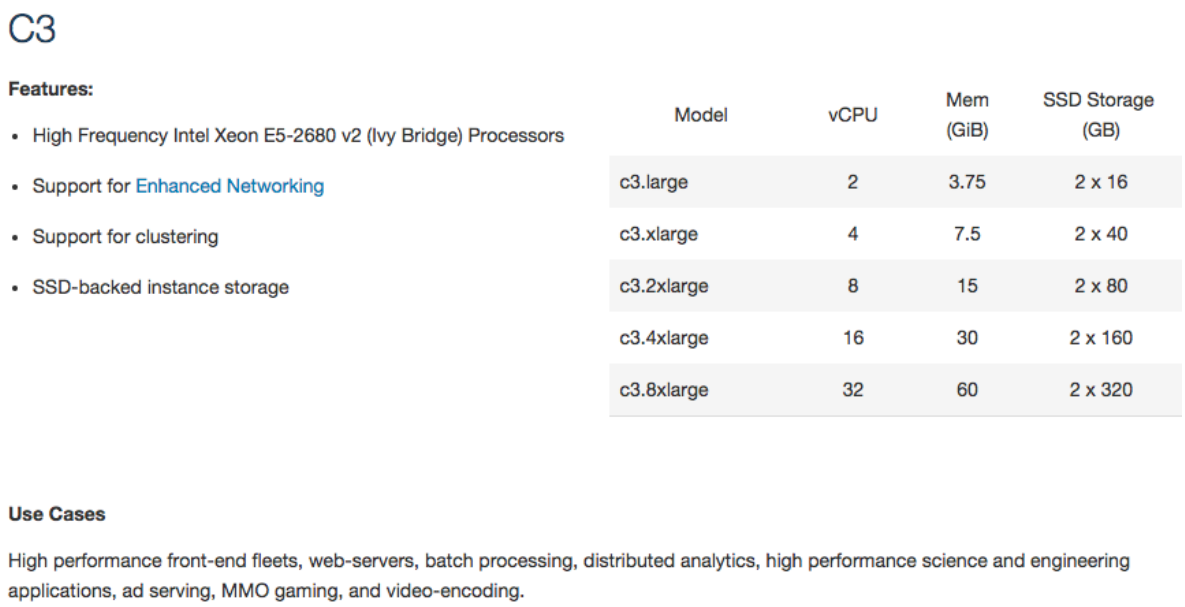


图 2：亚马逊云科技（AWS）控制台



图 3：选择 Ubuntu Server 16.04 LTS AMI



C3

Features:

- High Frequency Intel Xeon E5-2680 v2 (Ivy Bridge) Processors
- Support for Enhanced Networking
- Support for clustering
- SSD-backed instance storage

Model	vCPU	Mem (GiB)	SSD Storage (GB)
c3.large	2	3.75	2 x 16
c3.xlarge	4	7.5	2 x 40
c3.2xlarge	8	15	2 x 80
c3.4xlarge	16	30	2 x 160
c3.8xlarge	32	60	2 x 320

Use Cases

High performance front-end fleets, web-servers, batch processing, distributed analytics, high performance science and engineering applications, ad serving, MMO gaming, and video-encoding.

图 4：计算优化型（C3）实例层级

```
ssh -i path/to/KeyPair.pem ubuntu@ec2-XX-XXX-XX-XX.us-west-2.compute.amazon.com
```

进入终端。还可以使用 PuTTY，或使用 AWS 网站上的基于 Java 的终端等其他选项。您可以前往 `Instances` AWS 控制台左侧的菜单，找到所需信息。右键单击实例（如图*fig-connect* 所示），然后单击“连接”。

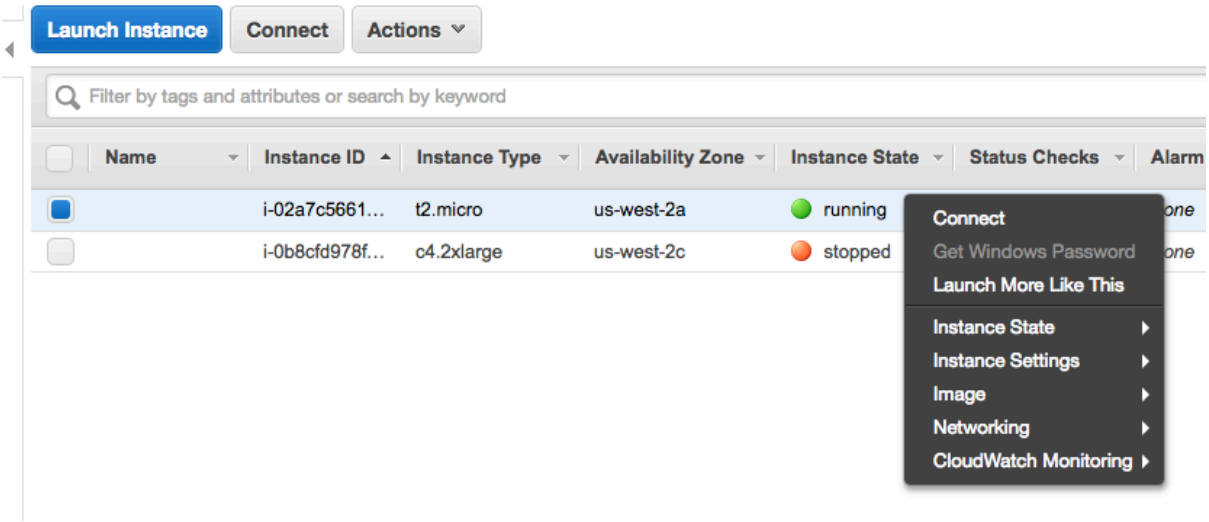


图 5：连接到实例

您将看到公共 DNS，其显示效果应类似于图*fig-dns*。

5.3.3 安装 YADE 和管理文件

通过 SSH 连接到实例后，您需要安装 YADE。请运行以下命令来安装 yadedaily、Python 以及其他一些实用工具：

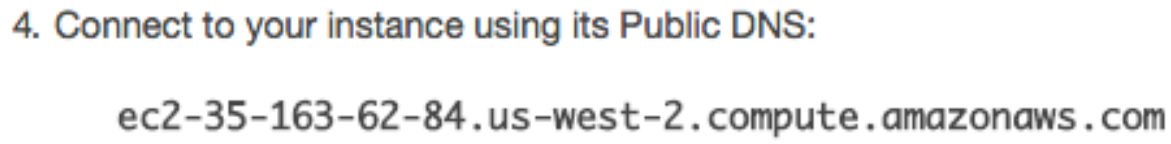


图 6：公共 DNS

```
#install yadedaily
sudo bash -c 'echo "deb http://www.yade-dem.org/packages/ xenial/" >> /etc/apt/sources.list'
wget -O - http://www.yade-dem.org/packages/yadedev_pub.gpg | sudo apt-key add -
sudo apt-get update
sudo apt-get install -y yadedaily

# install python
sudo apt-get -y install python
sudo apt-get -y install python-pip python-dev build-essential

# install htop
sudo apt-get -y install htop
```

请注意， `..packages/ xenial/` 应与 Ubuntu 发行版相匹配。16.04 LTS 对应 Xenial；但如果您选择启动 Ubuntu 14.04，则需要将 'xenial' 改为 'trusty'。
最后，您需要上传必要的 YADE 文件。如果您有一个包含模拟内容且名为 `yadeSimulation` 的文件夹，可以通过执行以下 command 上传该文件夹及其内容：

```
scp -r -i path/to/KeyYADEbox.pem path/to/yadeSimulation ubuntu@ec2-XX-XXX-XX-XX.us-west-2.
→compute.amazonaws.com:~/yadeSimulation
```

您现 在应该可以通过切换到相应的目录并输入以下内容来运行模拟 ：

```
yadedaily nameOfSimulation.py
```

如需取回输出文件（下文中名为"out"的文件夹）以进行后处理，请使用与上传该文件夹时相同的命令，但要对调远程和本地文件目标：

```
scp -r -i path/to/KeyYADEbox.pem ubuntu@ec2-XX-XXX-XX-XX.us-west-2.compute.amazonaws.com:~/
→yadeSimulation/out/ path/to/yadeSimulation/out
```

5.3.4 在终端中绘图

云计算面临的主要问题之一是缺乏图形反馈。借助 gnuplot 出色的"哑终端"功能，可以轻松地通过图形检查模拟的 status 命令。只需进入保存模拟输出的子文件夹并输入以下命令，即可轻松绘制任意数据：

```
gnuplot
set terminal dumb
plot ``data.txt" using 1:2 with lines
```

其中，"1:2"指的是 data.txt 中你想要相互绘图的列。输出应类似于：

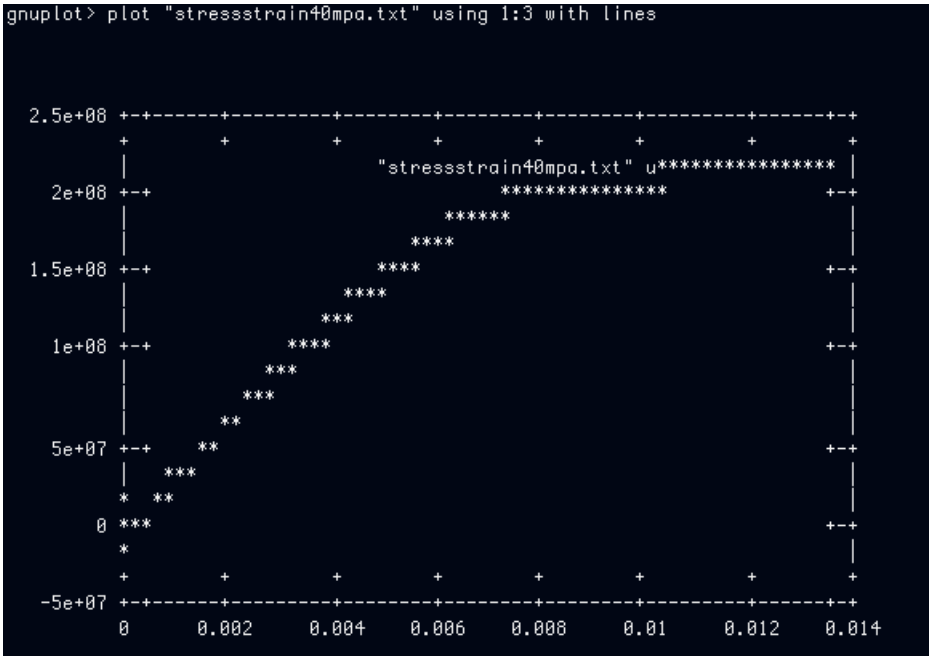


图 7：Gnuplot 输出

5.3.5 注释

- Amazon AWS 允许你停止实例，并在之后重新启动它，同时保留相同的文件和软件包安装。如果你希望创建多个包含相同安装内容和文件目录的实例，可以为默认图像创建快照，然后用它创建各种体积，并将这些体积挂载到新实例上。所有这些操作都可以通过 EC2 控制台轻松地以图形方式完成。
- 您可以使用竞价型实例。这是一种特殊类型的实例，允许您竞价使用闲置服务器。其价格大幅折扣，任何希望运行数百小时模拟的 YADE 用户都值得考虑。
- 对于大多数模拟，如果每次模拟使用超过 8 个核心，计算效率会降低。建议使用 yadedaily-batch 合理分配核心，以确保每次模拟始终分配 8 个核心，并让处理器保持 100% 运行。
- 创建一个 tmux 会话，以避免与服务器断开连接时 YADE 模拟终止。

```
tmux  # starts a new session
tmux attach -t 0  # attach session 0
tmux kill -t 0  # kill session
## cntrl - b - d to move back to home
## cntrl - b - [ to navigate within the session
```

5.4 高精度计算

Yade 支撑体高精度和任意精度 `Real` 类型的计算。所有测试和检查均已通过，但目前这项支持仍被视为实验性功能。后端库为Boost Multiprecision以及相应的Boost 数学工具包。

已支持的类型如下：

type	bits	小数位数[1]	备注
`float`	32	6	硬件加速（无用，仅用于测试用途）
`double`	64	15	硬件加速
`long double`	80	18	硬件加速
`boost float128`	128	33	根据处理器类型，它可能由硬件加速——实现，由 Boost 封装
`boost mpfr`	`Nbit`	`Nbit/(log(2)/log(10))`	使用外部 mpfr 库，由 Boost 封装
`boost cpp_bin_float`	`Nbit`	`Nbit/(log(2)/log(10))`	仅使用 Boost，但速度较慢

最后两种类型是任意精度的，其位数 `Nbit` 或小数位数在编译时通过参数指定。

注意：详情请参阅文件Real.hpp 。所有 `Real` 类型均通过了实数类型概念testBoost 概念库中的测试。对 Eigen 和CGAL 的支持通过数值特征实现。

5.4.1 安装

预编译的 Yade 软件包默认使用 `double` 类型。要使用高精度类型，必须按照常规的 安装说明从源代码编译并安装 Yade。另请注意以下事项：

1. 必须安装以下软件包： `python3-mpmath libmpfr-devlibmpfrc++-dev libmpc-dev` （ `mpfr` 和 `mpc` 相关软件包仅在使用`boost::multiprecision::mpfr` 类型时才需要）。这些软件包已列入默认依赖项中。

2. 需要使用版本 9.2.1 或更高版本的 g++ 编译器。请注意，在现有的 Linux 安装上仅升级编译器（旧版 Linux 安装中尚未提供针对不同 GCC 版本的软件包）比较困难，因此不建议这样做。更简单的解决方案是升级整个 Linux 安装。

3. 调用 CMake 时指定：

   1. 位数，如 `REAL_PRECISION_BITS=……`；2. 或所需的小数位数，如 `REAL_DECIMAL_PLACES=……`，但不能同时指定两者

   3. 要使用 MPFR，请指定 `ENABLE_MPFR=ON` （默认值为 `OFF` ）。

   任意精度（`mpfr` 或 `cpp_bin_float`）类型仅在要求超过 128 位或超过 39 位小数时使用。在这种情况下，如果 `ENABLE_MPFR=OFF` 则会使用速度较慢的`cpp_bin_float` 类型。39 位与 33 位小数之间的差异，源于指数占用了 15 位。注意：一种具有 62 位小数的快速四双精度实现（Debian 软件包 `libqd-dev`）正在开发中，由 Boost Multiprecision 团队负责。

5.4.2 支持的模块

在编译期间，可以通过向 CMake 传递一个 `ENABLE_*` 命令行参数来启用或禁用多个 Yade 模块。下表列出了目前支持高精度的模块

[1] 此表中的小数位数，是指由二进制表示完全确定的小数位数。要完整重建二进制表示，还需要少量额外的十进制数字。下面的 Python 示例展示了这一点： `for a in range(16): print(1./pow(2.,a))`，它表明每个二进制数字都会在十进制表示末尾产生“额外的” …25 ，但这些十进制数字并非由二进制表示完全确定，因为例如 …37 不可能由此得到。表示 …37需要更多二进制位，但 …25 是由最后一个可用位产生的。

（标记为“可能”的模块未经测试） :

ENABLE * 模块名称_	HP 支持	CMake 默认设置	注释
ENABLE_GUI	yes	ON	原生支持[2]
ENABLE_CGAL	yes	ON	原生支持[2]
ENABLE_VTK	yes	ON	已支持[3]
ENABLE_OPENMP	部分	ON	部分支持[4]
ENABLE_MPI	可能	OFF	未测试[5]
ENABLE_GTS	yes	ON	已支持[6]
ENABLE_GL2PS	yes	ON	已支持[6]
ENABLE_LINSOLV	no	OFF	不支持[7]
ENABLE_PARTIALSAT	no	OFF	不支持[7]
ENABLE_PFVFLOW	no	OFF	不支持[7]
ENABLE_TWOPHASEFLOW	no	OFF	不支持[7]
ENABLE_THERMAL	no	OFF	不支持[7]
ENABLE_LBMFLOW	yes	ON	已支持[6]
ENABLE_SPH	可能	OFF	未测试[8]
ENABLE_LIQMIGRATION	可能	OFF	未测试[8]
ENABLE_MASK_ARBITRARY	可能	OFF	未测试[8]
ENABLE_PROFILING	可能	OFF	未测试[8]
ENABLE_POTENTIAL_BLOCKS	no	OFF	不支持[9]
ENABLE_POTENTIAL_PARTICLES	yes	ON	已支持[10]
ENABLE_DEFORM	可能	OFF	未测试[8]
ENABLE_OAR	可能	OFF	未测试[8]
ENABLE_FEMLIKE	yes	ON	已支持[6]
ENABLE_ASAN	yes	OFF	已支持[6]
ENABLE_MPFR	yes	OFF	原生支持[2]
ENABLE_LS_DEM	no	ON	不支持[11]

在高精度 cmake 阶段，不支持的模块会自动禁用 .

5.4.3 双精度、四精度及更高精度

有时，如果将 C 中计算的关键部分++ 以更高精度执行，就能更好地保证其在默认精度下产生正确结果。一个简单的例子是求解线性方程组（本质上是求逆矩阵），其中某些系数非常接近零。缓解此类问题的另一个例子是Kahan 求和算法。

如果满足要求，Yade 支持以如下方式使用更高精度乘数： RealHP<1> 是上述描述的 Real 类型，而每个更大的数字都是 Real 精度的乘数。 RealHP<2>是 RealHP<1>的双精度， RealHP<4> 是四倍精度，依此类推。其通用公式为

[2] 此功能原生支持，这意味着已为 Eigen 和 CGAL 编写了专用数值特征，并添加了图形用户界面和 Python 支持。[3] VTK 通过兼容层支持，该层会将所有数值转换为 double 类型。见下文。[4] OpenMPArrayAccumulator 实验性支持 long double 和 float128。对于类型 mpfr 和 cpp_-bin_float ，

将使用单线程版累加器。文件 lib/base/openmp-accu.hpp 仍需进一步测试。如有疑问，请使用 ENABLE_OPENMP=OFF编译 Yade。其他所有情况下，OpenMP 多线程都应能正常工作。[5] MPI 支持尚未经过测试，网络数据发送也尚未经过测试。[6] 该模块已通过测试， yade --test 和 yade --check 均已通过，且大多数示例也能正常运行。不过，尚未针对所有可能的用例进行广泛测试。[7] 不支持；该代码使用外部 CHOLMOD 库，而该库仅支持 double 类型。要使其正常工作，应使用原生 Eigen 线性方程求解器。[8] 此功能默认 OFF ，尚未对该功能的支持进行测试。[9] 势块使用外部线性规划库 coinor，而该库仅支持 double 类型。要使其正常工作，必须使用 Eigen 实现线性规划例程，或者 coinor 库开始使用 C++ 模板，或者使用类似 LAPACK 库的转换器/封装器。[10] 该模块默认启用， yade --test 和 yade --check 均已通过，且大多数示例也能正常运行。不过，计算是在较低 double 精度下进行的。已为 LAPACK 库实现封装器/转换器层。要使其达到完整精度，应使用 Eigen 重新实现这些例程。[11] 未来可能改进。请参见其中的注释。

小数位数的数量在RealHP.hpp 文件中实现，使用的小数位数只是 Real 精度小数位数的 N 倍；当原生类型不可用时，就会使用它。可用的原生精度类型系列列于RealHPLadder 类型列表中。

所有列于 MathEigenTypes.hpp 中的类型都遵循相同的命名模式： Vector3rHP<1> 是常规Vector3r，而 Vector3rHP<N> 对于任何已支持的 N，都使用精度乘数 N。随后，可以使用 Eigen 算法，通过 MatrixXrHP<N> 以更高的 N 求解线性方程组，从而获得更高精度的结果。关键截面已完成后，再继续使用默认 Real 精度进行计算。CGAL 类型也使用相同的命名约定，例如 CGAL_AABB_treeHP<N>，它们在文件AliasCGAL.hpp中声明。

在全面转向 C++20 标准之前，有一项小限制：实际支持的精度乘数由 RealHPConfig.hpp 文件中的以下两个定义决定：

1. #define YADE_EIGENCGAL_HP (1)(2)(3)(4)(8)(10)(20) ——此处列出的乘数可用于 CGAL 和 Eigen 中的 C++，针对 RealHP<N>。它们不会显著增加编译时间，但仍必须在此列出。将代码迁移到 C++20 后，此定义将被移除，所有乘数都将通过单个模板约束得到支持。这一不便源于 CGAL 和 Eigen 库都只为特定类型提供模板特化，而不支持通用类型族。因此，此定义用于声明所需的模板特化。

提示： 使用默认设置编译 Yade 时，默认可用的最高精度 N=（20）对应 300 位小数，无需更改 REAL_DECIMAL_PLACES=…… CMake 编译选项。

---

2. #define YADE_MINIEIGEN_HP (1)(2) - 此处列出的精度乘数会导出到 Python 中，代价较高：每增加一个，编译时间就会延长 1 分钟。只有在调试时，增加更多乘数才可能有用。此处默认列出 double RealHP<2> 类型，以便从 Python 中探索更高精度类型。另请注意， mpmath 支撑体 一次只支持一种精度 。对不同 mpmath 精度的变量的支持并不好，不过 mpmath 作者承诺今后会改进这一点。幸运的是，这对 Yade 用户来说并不是什么大问题，因为这里的总体目标是在 C++ 代码的关键截面中进行更精确的计算，而不是在 Python 中。每次发生转换时 调整 mpmath 精度，可以部分缓解这一问题 。因此 C++ python 需要牢记，变量 mpmath.mp.dps 始终反映最近一次转换所使用的精度，即使该转换发生在图形用户界面中（而非正在运行的脚本中）也是如此。现有 mpmath 变量不会被截断到较低精度；超出的位数只会被忽略，直到再次提高 mpmath.mp.dps 精度。不过，赋值时可能会发生截断。

有时，我们需要对不同精度的 C++ 类型进行直观的向上转换，例如将 RealHP<1> 加到 RealHP<2> 类型上。文件 UpconversionOfBasicOperator-sHP.hpp 正是用于此目的。默认情况下不会包含此头文件，因为大多数情况下，将这两种不同类型相加（从效率角度来看）可能是个错误，编译器也会检测到并报错。包含此头文件后，就可以执行此操作，并且该操作的结果类型始终是所用两种类型中精度较高的类型。此文件只能包含在.cpp 文件中。如果将其包含在任何 .hpp 文件中，就可能导致 C++ 类型安全问题，并产生意想不到的后果。以下 test 例程展示了该头文件的用法。

警告： 尝试在 YADE_MINIEIGEN_HP 中使用未注册的 N，为 Vector3rHP<N> 类型在YADE_CLASS_BASE_DOC_ATTRS_* 宏中将其导出到 Python 将不起作用。只有YADE_MINIEIGEN_HP 中列出的这些 N 才能正常工作。不过，在代码关键部分的 C++ 计算中，使用来自 YADE_EIGENCGAL_-HP的这些值是安全的（也是预期的），无需将它们导出到 Python。

5.4.4 兼容性

Python

要声明具有 `Real` 和 `RealHP<N>` 精度的 Python 变量，请使用函数 *math.Real(...)*、*math.Real1(...)*、*math.Real2(...)*。支持的精度列于 `YADE_MINIEIGEN_HP`，但请注意 *mpmath* 转换限制。

Python 原生支持 使用 包实现高精度类型 `mpmath` 。使用已支持模块的旧版 Yade 脚本，只需切换到 `yade.minieigenHP`，即可立即转换为高精度模式。为此，需要将以下代码行：

```
from minieigen import *
```

必须替换为：

```
from yade.minieigenHP import *
```

分别 `import minieigen` 必须替换为 `import yade.minieigenHP as minieigen`；旧名称 `as minieigen` 仅用于向后兼容。使用非 `double` 类型作为 `Real`时，将使用高精度（二进制兼容）版本的 minieigen。

`RealHP<N>` 更高精度的向量和矩阵可在 Python 中通过 `.HPn` 模块作用域访问。例如：

```
import yade.minieigenHP as mne
mne.HP2.Vector3(1,2,3) # produces Vector3 using RealHP<2> precision
mne.Vector3(1,2,3)     # without using HPn module scope it defaults to RealHP<1>
```

相应的数学函数，例如：

```
import yade.math as mth
mth.HP2.sqrt(2) # produces square root of 2 using RealHP<2> precision
mth.sqrt(2)     # without using HPn module scope it defaults to RealHP<1>
```

也已支持，并通过调用相应的 C++ 函数来实现，通常比 `mpmath`函数更快。

警告： 仍有部分 Python 代码可能尚未迁移到高精度，因此可能无法与 `mpmath` 模块正常配合使用。详情请参阅 调试章节 。

C++

引入高精度之前，我们假设 `Real` 实际上是 POD `double` 类型。可以在 `memset(…)`上使用 `memcpy(…)` 以及类似的函数 `double`。这种做法并不妥当，甚至还使用了一些编译器 `#pragma` 命令来屏蔽编译警告。为了使`Real` 能够与其他类型配合使用，必须移除此假设。文件 `memcpy(…)` 中仍有一个 openmp-accu.hpp，需要将其移除。今后的开发应避免使用此类原始内存访问函数。

其余所有 `double` 均已替换为 `Real` ，任何在代码中尝试使用 `double` 类型的行为都会在 GitLab CI 流水线中失败。
所有高精度类型的数学函数都通过文件 MathFunctions.hpp进行封装；这些内联重定向会指向 Yade 当前编译所用类型的相应函数。如果使用 `std::` ，代码将无法通过流水线检查。所有接受 `Real` 参数的函数现在都应调用 `yade::math::` 命名空间中的这些函数。仅接受 `Real` 参数的函数可以省略 `math::` 说明符，改用参数依赖查找。示例：

1. 对 `std::min(a,b)` 的调用会被替换为 `math::min(a,b)`，因为 `a` 或 `b` 可能是 `int` （非 `Real`），因此必须使用 `math::` 。

2. 对 `std::sqrt(a)` 的调用可以替换为 `sqrt(a)` 或 `math::sqrt(a)`，这得益于 参数依赖查找，因为 `a` 始终是 `Real`。

如果需要新增数学函数，则必须在以下位置添加：

1.lib/high-precision/MathFunctions.hpp 或 lib/high-precision/MathComplexFunctions.hpp 或 lib/high-precision/MathSpecialFunctions.hpp，具体取决于函数类型。2. py/high-precision/_math.cpp，详见 *math* 模块 。3. py/tests/testMath.py4. py/tests/testMathHelper.py

新函数的测试应添加到 py/tests/testMath.py中的以下某个函数中：`oneArgMathCheck(…):`，`twoArgMathCheck(…):`，`threeArgMathCheck(…):`。还应补充 `self.defaultTolerances` 中的预期误差容差近似值表。为了更可靠地确定容差，建议暂时增加 测试循环中的测试数量。为了确定当前已实现函数的容差，循环中使用了 `range(1000000)`。

注意： 在 `C++` 函数调用中传递参数时，建议使用 `const Real&`，而不是像 `Real`那样复制参数。原因如下：在非高精度的常规情况下，`double` 类型和引用都占 8 个字节。但 `float128` 占 16 个字节，而其引用仍只有 8 个字节。因此，对于常规精度，两者没有差异。对于所有更高精度的类型，将 `const Real&` 用作函数参数更有利。此外，对于 `const Vector3r&` 参数，即使不使用高精度，速度提升也更明显。

字符串转换

在 `pyt hon` 侧，建议使用 *math.Real(...) math.Real1(...)*，或 *math.toHP1(...)* 来声明 repython并使用 *math.radiansHP1(...)* 将角度转换为弧度，采用完整的 *Pi* 精度。

另一方面，建议使用 `C++` yade::math::toString(...) 和 yade::math::fromStringReal(...) 转换函数，而不是 `boost::lexical_cast<std::string>(…)`。`toString` 及其高精度版本 toStringHP 函数（位于文件 RealIO.hpp中）可确保转换时保持完整精度。需要注意的是，`std::to_string` 不能保证这一点，而 `boost::lexical_cast` 也不能保证这一点。

对于更高精度类型，可以在运行时通过修改 `C++` `python` 在`RealHP<N>` 字符串转换过程中的精度，来进行控制；具体方法是修改 *math.RealHPConfig.extraStringDigits10* 静态参数。每个十进制数字需要 $\log_{10}(2) \approx 3.3219$ 位。`std::numeric_limits<Real>::digits10` 提供的信息说明，有多少个十进制数字可由二进制表示完全确定，也就是说，这些数字绝对准确。不过，要转换回二进制，还需要更多十进制数字，因为每一位使用 $\log_2(10) \approx 0.3010299$ 个十进制数字，而来自 `std::numeric_limits<Real>::digits10` 的最后一位并不足够。通常，*extraStringDigits10*设为 3 或更大，就足以确保数值往返转换始终正常工作。不过，如果只想从 Python 中提取结果，而不再将结果输入以继续计算，那么建议将*extraStringDigits10* 设为较小的值，例如 0 或 1，以免产生精度更高的错觉，实际上并非如此：这些额外的十进制数字在十进制意义上并不准确。它们只是为了实现正常的数值往返转换。另请参阅与 Boost 开发者进行的 简短讨论。更多详情请参阅文件 RealHPConfig.cpp。

注意：参数 `extraStringDigits10` 不会影响 `double` 转换，因为boost::python 会针对这种特定类型使用内部转换器。如果有需要，未来可能会进行更改。例如，使用类似于 ThinRealWrapper 的类。

需要注意的是，通过 `RealHP<2>` 从`RealHP<1>` 字符串表示方式`RealHP<1>` 创建更高阶类型（例如 `RealHP<1>`）存在歧义。请看下面的示例：

```
import yade.math as mthmth.HP1.getDecomposedReal(1.23)['bits']
Out[2]: '10011101011100001010001111010111000010100011110101110'
mth.HP2.getDecomposedReal('1.23')['bits']  # passing the same arg in decimal format to HP2␣
→produces nonzero bits after the first 53 bits of HP1Out[3]:
→'10011101011100001010001111010111000010100011110101110000101000111101011100001010001111010111000010100011110101
_mth.HP2.getDecomposedReal(mth.HP1.toHP2(1.23))['bits'] # it is possible to use yade.math.HPn.
→toHPm(…) conversion, which preserves binary representationOut[4]:
→'10011101011100001010001111010111000010100011110101110000000000000000000000000000000000000000000000000000000000000
→
```

这两种 RealHP<2> 二进制表示方式中哪一种更理想，取决于所需的内容：

1. 1.23 这个十进制数的最佳二进制近似值。
2. 表示 将该数的 53 个二进制位扩展到更高精度，以继续计算 s on t**同一个**先前以较低精度表示的数。

要实现 1.，只需将参数 '1.23' 作为字符串传递。要实现 2.，请使用 *math.HPn.toHPm(...)* 或*math.Realn(...)* 转换，它通过单次 静态_转换<RealHP<m>>(...)来保持二进制保真度。类似问题也在 mpmath 和Boost 文档中进行了讨论。

toHPn 与 Realn 的区别如下：这些函数 HPn.toHPm 会创建一个 $m \times n$ 矩阵，将 RealHP<n> 转换为 RealHP<m>。当 $n < m$ 时，额外的位会被置为零（上文的算例 2；根据具体需求，也可以说“发生了精度损失”）。函数 *math.Real(...)*、*math.Real1(...)*、*math.Real2(...)*是该矩阵对角线元素的别名（上文的算例 1；根据具体需求，也可以说使用这些函数时“不会发生转换损失”）。

提示： 所有 RealHP<N> 类型高于 double 的函数参数也可以接受十进制字符串。这样可以保留高于 Python 默认浮点精度的精度。

警告： 相反，所有类型为 double 的函数参数都不能接受十进制字符串。为了解决这个问题，可以使用接受字符串参数的 toHPn(…) 转换器。

提示： 为了更轻松地调试此问题，函数 *math.toHP1(...)* 会 引发 RuntimeError，前提是参数是 Python 浮点数（而不是十进制字符串）。

警告： 我再怎么强调这个问题也不为过，请尝试在不同于 yade --check （或 yade ./checkGravityRungeKuttaCashKarp54.py）的精度下运行 double，在将这一行改为 g = -9.81之后。在这个（特定且简单的）例子中， getCurrentPos() 函数在 Python 端失败，因为低精度 g 与高精度 t相乘。

复数类型

同样支持复数。所有标准 C++ 函数都可在lib/high-precision/MathComplexFunctions.hpp中使用，并且也会导出到 Python 中的py/high-precision/_math.cpp。这里有一个 CMake 编译选项 ENABLE_COMPLEX_MP，可启用使用更好的复数类型，来源为

boost::multiprecision 用于表示 ComplexHP<N> 类型系列的库： complex128、 mpc_complex、 cpp_complex 和 complex_adaptor。只要可能，默认就会启用它：对于 Boost 版本>= 1.71。对于较旧的 Boost， ComplexHP<N> 类型则由 std::complex<RealHP<N>> 表示，该类型在某些数学函数中会产生更大的数值误差。

使用ENABLE_COMPLEX_MP=ON（默认）时，前面提到的lib/high-precision/UpconversionOfBasicOperatorsHP.hpp对复数类型不起作用，这是一个已报告的Boost 库问题。

使用 MPFR 类型时，必须安装 libmpc-dev 软件包（如上所述） .

Eigen 和 CGAL

Eigen 和 CGAL 库原生支持高精度。

- Eigen 所需的所有声明都在以下文件中提供：EigenNumTraits.hpp 和MathEigen-Types.hpp
- CGAL 所需的所有声明均由以下文件提供：CgalNumTraits.hpp 和 AliasCGAL.hpp

VTK

由于 VTK 仅用于记录结果，以便日后在 ParaView等其他软件中查看，因此目前似乎没有必要记录所有小数位数。因此， C++中的所有记录命令都会将 Real 类型转换为 double 类型，并使用 static_cast<double> 命令。这是通过以下类实现的： vtkPointsReal、 vtkTransformReal 和 vtkDoubleArrayFromReal ，它们位于文件VTKCompatibility.hpp中。未来 VTK 可能会支持非 double 类型。如果有此需要，可以在那里更新接口。

LAPACK

Lapack 是一个仅支撑 double 类型的外部库。由于它不支持模板化，因此无法与 Real 类型搭配使用。当前的解决方案是，在调用线性方程求解器（及其他函数）时，将参数降精度转换为 double ，然后再将其转换回 Real。这种临时解决方案舍弃了高精度的所有优势，因此未来计划用 Eigen 或其他支持任意浮点类型的模板化库替换 Lapack。

5.4.5 调试

高精度实现仍处于实验阶段。使用过程中可能会出现一些错误，并非所有错误都能通过检查和测试发现。以下示例或许能有所启发：

1. 尝试使用指向 Vector3r 成员的 const 引用——这类问题会导致运行时发生段错误。2. 一部分 Python 代码与 mpmath 不兼容——检查和测试尚未覆盖 Python 代码中的全部行，因此预计还会出现更多类似错误。解决方法是将不兼容的 Python 函数放入py/high-precision/math.py中。然后将对原函数的调用替换为对该函数的调用 yade.math，例如 numpy.linspace(…) 替换为 yade.math.linspace(…)。

调试时， long double 类型具有最大的灵活性，因为专门的文件ThinRealWrap-per.hpp、ThinComplexWrapper.hpp 就是为此编写的。它们使用boost::operators，并采用偏序域实现。请注意，它们不提供运算符++。

在这两个文件中引入了几个 #defines ，以帮助调试更难处理的问题：

1. YADE IGNORE IEEE INFINITY NAN - 可用于检测使用 NaN 或 Inf _ _ _ _时的所有情况。此外，建议在使用 -Ofast 标志编译 Yade 时启用此定义，不启用 -fno-associative-math -fno-finite-math-only -fsigned-zeros

2. YADE_WRAPPER_THROW_ON_NAN_INF_REAL, YADE_WRAPPER_THROW_ON_NAN_INF_COMPLEX - 当计算因未知原因完全出错时，可用于调试。

另请参阅地址消毒器章节，因为它在许多情况下对调试非常有用。

提示：如果崩溃发生在宏内部，例如 YADE_CLASS_BASE_DOC_ATTRS_CTOR_PY，了解问题在宏中的具体位置会很有帮助。为此，可以使用 g++ 预处理器移除宏，然后编译不含宏的预处理后代码。使用以下命令的某种变体调用预处理器：

```
g++ -E -P core/Body.hpp -I ./ -I /usr/include/eigen3 -I /usr/include/python3.7m > /tmp/Body.hpp
```

可以使用 clang-format 让这个文件更易读：

```
./scripts/clang-formatter.sh /tmp/Body.hpp
```

请注意，此类文件往往很大，而且 clang-format 运行缓慢。因此，有时只使用文件的最后一部分会更有用，因为宏已在其中完成后处理。然后在原始文件中替换相应的宏，再继续调试。不过这一次，就能发现问题发生在宏的哪个位置。

注意：在提出有关高精度的问题时，建议在问题标题开头加上[RealHP]。

第6章

文献

6.1 Yade 技术档案

6.1.1 关于

Yade 技术档案（YTA）旨在通过阐明 Yade 开源代码背后的理论、解释算法实现并提供实用教程，提高 Yade 相关出版物的可重复性。为此，YTA 消除了读者与计算类期刊文章之间普遍存在的不透明性，增强并提高了现有 Yade 期刊论文的可见度，促进学术合作，并拓展开放获取学术界。

6.1.2 贡献

YTA 征集各种 Yade 相关材料，包括但不限于：

- 代码包的理论描述
- 代码包的用户指南和教程
- 演示文稿
- 课程材料
- 期刊文章的补充材料

6.1.3 联系方式

如需贡献，请联系 rob.caulk@gmail.com。关于单篇出版物的问题，请联系文档描述中附带的电子邮件地址。有关代码的一般问题，请访问 我们的问答论坛。

6.1.4 档案

Chareyre, Bruno；Caulk, Robert；Chèvremont, William；Guntz, Thomas；Kneib, François；Kunhappen, Deepak；Pourroy, Jean（2019年），用于颗粒系统动力学的 MPI 分布式计算。 *Yade* 技术档案。下载全文， 观看视频摘要， 阅读海报摘要

Pirnia, Pouyan；Duhaime Francois；Ethier Yannic；Dubé, Jean-Sébastien（2019年），COMSOL-Yade 接口（ICY）使用指南。*Yade* 技术档案。下载全文，发送电子邮件 seyed-pouyan.pirnia.1@ens.etsmtl.ca ， 下载帮助文件

Maurin, Raphael（2018年），YADE 一维垂向 VANS 流体求解：数值求解细节。*Yade* 技术档案。下载全文，发送电子邮件 raphael.maurin@imft.fr，参阅教程：使用 *YADE* 一维垂向 *VANS* 流体求解

Maurin, Raphael（2018年），YADE 一维垂向 VANS 流体求解：理论基础。*Yade* 技术档案。下载全文，发送电子邮件 raphael.maurin@imft.fr，请参阅教程：使用 *YADE* 一维垂向 *VANS* 流体求解

Maurin, Raphael（2018年），YADE 一维垂向 VANS 流体求解：验证。*Yade* 技术档案。下载全文，发送电子邮件 raphael.maurin@imft.fr，请参阅教程：使用 *YADE* 一维垂向 *VANS* 流体求解

Caulk, Robert（2018年），用于研究非均质岩石拉伸破裂的离散元方法随机增强。*Yade* 技术档案。数字对象标识符 10.5281/zenodo.1202039。下载全文，发送电子邮件 rob.caulk@gmail.com ，请参阅教程：在*Yade*中模拟声发射

6.2 Yade相关出版物

本页面列出了关于 Yade 本身或使用 Yade 完成的出版物。

第一节提供了我们恳请您在出版物中引用 Yade 时使用的参考文献，详情请见 “致谢Yade”一节。

随着使用 Yade 的出版物数量不断增加，已难以将其全部列出，因此近些年的收录仅为部分。您可以帮助我们：如果您发表了与 Yade 相关的出版物，或知道相关出版物，请随时将其添加到此列表中。如果您无法直接访问源代码，请将参考文献（以 BibTeX 条目形式）发送给 Yade 开发者。如果有可免费获取的 PDF 文件，请添加可直接下载全文的网址。只要符合法律规定，Yade 的网络服务器很乐意托管此类 PDF。

注意： 此文件由以下文件生成： doc/yade-articles.bib、 doc/yade-conferences.bib、doc/yade-theses.bib、doc/yade-tech-archive.bib和 doc/citing_yade.bib。

6.2.1 引用 Yade

对应的 BibTeX 条目此处。另请参阅“致谢Yade”。

6.2.2 期刊文章 6.2.3 会议资料和书籍章节
6.2.4 硕士和博士学位论文 6.2.5 Yade 技术档案

6.3 引用

Yade 文档中引用的全部外部文章。

注意： 此文件由doc/references.bib生成。

第7章

索引和表格

- 通用索引
- 模块索引
- 搜索

参考文献

[yade:doc3]V. Smilauer 等（2021），Yade 文档第3版。Yade 项目。数字对象标识符 10.5281/zenodo.5705394 (http://yade-dem.org/doc/)[Aboul2017] Aboul Hosn, R., Sibille, L., Benahmed, N., Chareyre, B.（2017），采用球形颗粒和颗粒间滚动摩擦对松散土进行离散数值建模。《颗粒物质》（19）。数字对象标识符 10.1007/s10035-016-0687-0[Albaba2015] Albaba, A, Lambert, S, Nicot, F, Chareyre, B（2015），利用离散元建模研究颗粒流作用于刚性墙的荷载与组构之间的关系。《颗粒物质》（17），第 603–616 页。数字对象标识符 10.1007/s10035-015-0579-8[Angelidakis2021] Angelidakis, Vasileios, Nadimi, Sadegh, Utili, Stefano（2021），颗粒工程形状分析器（SHAPE）：基于成像数据无缝表征并简化颗粒形态。《计算机物理通讯》，第 107983 页。[Bance2014] Bance, S., Fischbacher, J., Schrefl, T., Zins, I., Rieger, G., Cassignol, C.（2014），形状各向异性永磁体的微磁学。《磁学与磁性材料杂志》（363），第 121–124 页。[Barbosa2020] Barbosa, Luis Alfredo Pires（2020），对整体土壤的团聚体结构建模，以量化土壤耕作工具形成苗床时的破碎特性和能量需求。《生物系统工程》（197），第 203–215 页。[Barbosa2020b] Barbosa, Luis Alfredo Pires, Keller, Thomas, de Oliveira Ferraz, Antonio Carlos（2020），团聚体破裂的尺寸效应：利用易碎性与分形维数之间的关系为离散元模型设定参数。《粉体技术》（375），第 327–336 页。[Benniou2020] Benniou, H., Accary, A., Malecot, Y., Briffaut, M., Daudeville, L.（2020），高应力水平下混凝土的离散元建模：饱和度的影响。《计算颗粒力学》。数字对象标识符 10.1007/s40571-020-00318-5[Bonilla2015] Bonilla-Sierra, V., Scholtès, L., Donzé, F.V., Elmouttie, M.K.（2015），利用摄影测量数据和离散裂隙网络–离散元建模分析岩质边坡稳定性。《岩土工程学》，第 1–15 页。数字对象标识符 10.1007/s11440-015-0374-z[Boon2012] Boon, C.W., Houlsby, G.T., Utili, S.（2012），一种用于离散元法中凸多边形颗粒与多面体颗粒间接触检测的新算法。《计算机与岩土工程》（44），第 73–82 页。数字对象标识符 10.1016/j.compgeo.2012.03.012[Boon2013] Boon, C.W., Houlsby, G.T., Utili, S.（2013），一种用于三维非球形颗粒的新型接触检测算法。《粉体技术》（248），第 94–102 页。数字对象标识符 10.1016/j.powtec.2012.12.040[Boon2014] Boon, C.W., Houlsby, G.T., Utili, S.（2014），利用二维和三维离散元法分析重新认识1963年瓦伊昂滑坡。《岩土工程学》（64），第 800–816 页。数字对象标识符 10.1680/geot.14.P.041[Boon2015]Boon, C.W., Houlsby, G.T., Utili, S.（2015），一种基于线性规划的新型岩石切片方法。《计算机与岩土工程》（65），第 12–29 页。数字对象标识符 10.1016/j.compgeo.2014.11.007

[Boon2015b] Boon, C.W., Houlsby, G.T., Utili, S.（2015），通过 DEM 设计节理岩体隧道支护。《岩石力学与岩石工程》（48），第 603–632 页。数字对象标识符 10.1007/s00603-014-0579-8[Bourrier2013]Bourrier, F., Kneib, F., Chareyre, B., Fourcaud, T.（2013），利用植物根系加固颗粒土的离散建模。《生态工程》。数字对象标识符 10.1016/j.ecoleng.2013.05.002[Bourrier2015] Bourrier, F., Lambert, S., Baroth, J.（2015），一种用于设计落石防护栏的基于可靠性的方法。《岩石力学与岩石工程》（48），第 247–259 页。[Yuan2017] C. Yuan, B. Chareyre（2017），一种用于可变形颗粒介质水力-力学耦合的孔隙尺度方法。《计算力学与工程方法》。数字对象标识符 10.1016/j.cma.2017.02.024[Yuan2016] C. Yuan, B. Chareyre, F. Darve（2016），颗粒材料排水的孔隙尺度模拟：有限尺寸效应与代表性基本体积。《水资源进展》（95），第 109–124 页。[Catalano2014a] Catalano, E., Chareyre, B., Barthélémy, E.（2014），流体-颗粒相互作用及其涌现的孔隙力学效应的孔隙尺度建模。《国际岩土力学数值与分析方法杂志》（38），第 51–71 页。数字对象标识符 10.1002/nag.2198 (http://arxiv.org/pdf/1304.4895.pdf)[Caulk2020] Caulk, Robert A.（2020），使用离散元法对非均质岩石拉伸断裂过程中的声发射进行建模。《开放岩土力学》（2）。数字对象标识符 10.5802/ogeo.5[Chalak2017] Chalak, C., Chareyre, B., Nikooee, E., Darve, F.（2017），部分饱和介质：从离散元模拟到热力学解释。《欧洲环境与土木工程杂志》（21），第 798–820 页。数字对象标识符 10.1080/19648189.2016.1164087[Chapelle2021] Chapelle, David, Maynadier, Anne, Bebon, Ludovic, Thi’ebaud, Fr’ed’eric（2021），氢储存：不同技术、挑战与利害关系。聚焦 TiFe 氢化物。载于《可再生氢及其他可持续能源载体进展》，施普林格，[Chareyre2012a] Chareyre, B., Cortis, A., Catalano, E., Barthélemy, E.（2012），密集球体堆积中黏性流动及诱导力的孔隙尺度建模。《多孔介质中的传输》（92），第 473–493 页。数字对象标识符 10.1007/s11242-011-9915-6[Chassagne2020] Chassagne, Rémi, Frey, Philippe, Maurin, Raphaël, Chauchat, Julien（2020），推移质输运过程中双分散混合物的运动性。《物理评论：流体》（5），第 114307 页。[Chen2007] Chen, F., Drumm, E. C., Guiochon, G.（2007），利用离散元法预测/验证一维颗粒运动。《国际岩土力学杂志》，美国土木工程师学会（7），第 344–352 页。数字对象标识符 10.1061/(ASCE)1532-3641(2007)7:5(344)[Chen2011a] Chen, F., Drumm, E., Guiochon G.（2011），离散元法与有限体积法耦合求解两个经典土力学问题。《计算机与岩土工程》。数字对象标识符 10.1016/j.compgeo.2011.03.009[Chen2014] Chen, J., Huang, B., Shu, X., Hu, C.（2014），使用开源代码对沥青混合料实验室压实进行离散元模拟。土木工程材料杂志。[Chen2012] Chen, Jingsong, Huang, Baoshan, Chen, Feng, Shu, Xiang（2012），离散元法在 Superpave 旋转压实中的应用。《道路材料与路面设计》（13），第 480–500 页。数字对象标识符 10.1080/14680629.2012.694160[Cheng2016] Cheng, H., Yamamoto, H., Thoeni, K.（2016），利用离散元法研究土工袋中的应力状态和组构各向异性。《计算机与岩土工程》（76），第 170–183 页。数字对象标识符 10.1016/j.compgeo.2016.03.006

[Chevremont2020] Chèvremont, William、Bodiguel, Hugues、Chareyre, Bruno（2020），**用于悬浮液数值模拟的润滑接触模型**。《粉体技术》（372），第600–610页。

[Chevremont2019] Chèvremont, William、Chareyre, Bruno、Bodiguel, Hugues（2019），**摩擦悬浮液流变学的定量研究：大范围黏性数中摩擦系数的影响**。《物理评论：流体》（4），第064302页。

[Coulibaly2020] Coulibaly, Jibril B、Shah, Manan、Loria, Alessandro F Rotta（2020），**热循环效应对颗粒材料结构和物理属性的影响**。《颗粒物质》（22），第1–19页。[Cuomo2016] Cuomo, S.、Chareyre, B.、d'Arista, P.、Sala, M.D.、Cascini, L.（2016），**通过离散元方法对非饱和土雨滴溅蚀进行微观力学建模**。*CATENA*（147），第146–152页。[Dang2010a] Dang, H. K.、Meguid, M. A.（2010），**具有预定义属性的离散元试样生成算法**。《国际岩土力学杂志》（10），第85–91页。数字对象标识符 10.1061/(ASCE)GM.1943-5622.0000028

[Dang2010b]Dang, H. K., Meguid, M. A.（2010年），**评估显式动力松弛技术在分析非线性岩土工程问题中的性能**。《计算机与岩土工程》（37），第125–131页。数字对象标识符 10.1016/j.compgeo.2009.08.004

[DePue2019b]De Pue, Jan, Cornelis, Wim M（2019），农业交通荷载下应力传递的离散元模拟：第1部分——与连续体模型的比较及参数研究。《土壤与耕作研究》（195），第104408页。数字对象标识符 https://doi.org/10.1016/j.still.2019.104408[DePue2020a]De Pue, Jan, Lamand'e, Mathieu, Cornelis, Wim M（2020），农业交通荷载下应力传递的离散元模拟：第2部分——轮胎—土壤界面处的剪切应力。《土壤与耕作研究》（203），第104660页。数字对象标识符 https://doi.org/10.1016/j.still.2020.104660[DePue2020b] De Pue, Jan, Lamand'e, Mathieu, Schjonning, Per, Cornelis, Wim M（2020），农业交通荷载下应力传递的离散元模拟：第3部分——现场试验评估。《土壤与耕作研究》（200），第104606页。数字对象标识符 https://doi.org/10.1016/j.still.2020.104606[Deng2021] Deng, Na, Wautier, Antoine, Thiery, Yannick, Yin, Zhen-Yu, Hicher, Pierre-Yves, Nicot, Franccois（2021），颗粒材料中临界状态的吸引力。《固体力学与物理学杂志》（149），第104300页。[Donze2008] Donzé, F.V.（2008），黏聚性摩擦地质材料中的冲击。《欧洲环境与土木工程期刊》（12），第967–985页。[Donze2021] Donzé, Frédéric-Victor, Klinger, Yann, Bonilla-Sierra, Viviana, Duriez, Jérôme, Jiao, Liqing, Scholtès, Luc（2021），基于地球、火星和冰卫星走滑断层段评估脆性地壳厚度。《构造地球物理学》（805），第228779页。[Dyck2015] Dyck, N.J., Straatman, A.G.（2015），球形空隙相多孔介质微观结构数字化生成的新方法。《国际传热与传质杂志》（81），第470–477页。[Effeindzourou2016] Effeindzourou, A., Chareyre, B., Thoeni, K., Giacomini, A., Kneib, F.（2016），在离散元法的一般框架下对可变形结构进行建模。《土工织物与土工膜》（44），第143–156页。数字对象标识符 10.1016/j.geotexmem.2015.07.015[Epifancev2013] Epifancev, K., Nikulin, A., Kovshov, S., Mozer, S., Brigadnov, I.（2013），基于螺旋机械三维分析和YADE代码的泥炭团块成形过程建模。《美国机械工程杂志》（1），第73–75页。数字对象标识符 10.12691/ajme-1-3-3

[Epifantsev2012] Epifantsev, K., Mikhailov, A., Gladky, A.（2012），块状泥炭的生产、挤出、模具口模的入口段和定径段形状、离散元法 [RUS]。《矿业信息与分析公报》（科学技术期刊），第212–219页。[Favier2012] Favier, L., Daudon, D., Donzé, F.V.（2012），采用三维离散元模型研究超临界黏聚颗粒流对刚性障碍物的冲击。《寒区科学与技术》（85），第232–241页。[Favier2009a] Favier, L., Daudon, D., Donzé, F.V., Mazars, J.（2009），利用离散元法预测颗粒流的阻力系数。《统计力学杂志：理论与实验》（2009），第P06012页。[Feng2021] Feng, Shaochuan, Kamat, Amar M, Sabooni, Soheil, Pei, Yutao（2021），激光粉末床熔融悬垂区域表面粗糙度成因的实验与数值研究。《虚拟与实体原型》，第1–19页。[Houlsby2009]G.T. Houlsby（2009），势粒子：一种在离散元法中对非圆形颗粒建模的方法。《计算机与岩土工程》（36），第953–959页。数字对象标识符 10.1016/j.compgeo.2009.03.001[Gladky2017] Gladkyy, Anton, Kuna, Meinhard（2017），使用组合莫尔–库仑–威布尔破坏准则对多面体颗粒开裂进行离散元模拟。《颗粒物质》（19），第41页。数字对象标识符 10.1007/s10035-017-0731-8[Gladky2014] Gladkyy, Anton, Schwarze, Rüdiger（2014），用于离散元法的不同毛细桥模型比较。《颗粒物质》，第1–10页。数字对象标识符 10.1007/s10035-014-0527-z[Gougucs2020] Göğüş, Özge Dinç（2020），受拉硬岩损伤演化的三维离散分析。《阿拉伯地球科学杂志》（13），第1–11页。[Grabowski2021] Grabowski, A, Nitka, M, Tejchman, J（2021），使用离散元法对筒仓中准静态受限颗粒流剪切局部化进行微观建模。《计算机与岩土工程》（134），第104108页。[Grabowski2020] Grabowski, Aleksander, Nitka, Michal（2020），采用剪切局部化早期检测的基本岩土工程试验三维离散元模拟。《岩土工程与力学研究》。[Grujicic2013] Grujicic, M, Snipes, JS, Ramaswami, S, Yavari, R（2013），针对简易爆炸装置（IED）袭击下建筑物结构倒塌与生存能力的离散元建模与分析。《材料科学与应用进展》（2），第9–24页。[Guo2014] Guo, Ning, Zhao, Jidong（2014），用于颗粒介质分层多尺度建模的有限元/离散元耦合方法。《国际工程数值方法期刊》（99），第789–818页。数字对象标识符 10.1002/nme.4702[Gusenbauer2012] Gusenbauer, M., Kovacs, A., Reichel, F., Exl, L., Bance, S., Özelt, H., Schrefl, T.（2012），用于生物医学应用的自组织磁珠。《物理学B：凝聚态》（324），第977–982页。[Gusenbauer2014] Gusenbauer, M., Nguyen, H., Reichel, F., Exl, L., Bance, S., Fischbacher, J., Özelt, H., Kovacs, A., Brandl, M., Schrefl, T.（2014），用于生物医学应用的磁珠引导自组装。《物理学B：凝聚态》（435），第21–24页。[Hadda2015] Hadda, N., Nicot, F., Wan, R., Darve, F.（2015），颗粒材料破坏过程中的微观结构自组织。《力学报告》。[Hadda2013] Hadda, Nejib, Nicot, François, Bourrier, Franck, Sibille, Luc, Radjai, Farhang, Darve, Félix（2013），颗粒介质中二阶功的微观力学分析。《颗粒物质》（15），第221–235页。数字对象标识符 10.1007/s10035-013-0402-3[Harthong2012b]Harthong, B., Jerier, J.-F., Richefeu, V., Chareyre, B., Doremus, P., Imbault, D., Donzé, F.V.（2012），弹塑性球体堆积体中的接触碰撞，

应用于粉末压实。《国际机械科学杂志》(61)，第32–43页。[Harthong2009]Harthong, B., J erier, J.F., Doremus, P., Imbault, D., Donzé, F.V.（2009），采用离散元法对颗粒材料进行高密度压实建模。《国际固体与结构杂志》(46)，第3357–3364页。数字对象标识符 10.1016/j.ijsolstr.2009.05.008[Hartong2012a] Harthong, B., Scholtès, L., Donzé, F.-V.（2012），采用耦合离散元–离散裂隙网络模型表征岩体强度。《国际地球物理学杂志》(191)，第467–480页。数字对象标识符 10.1111/j.1365-246X.2012.05642.x[Hasan2016] Hasan, Alsidqi, Karrech, Ali, Chareyre, Bruno（2016），采用离散元法评估虚拟未胶结矿山充填体中的力分布。《国际岩土力学杂志》，第06016042页。[Hassan15] Hassan, Nadine Ali, Nguyen, Ngoc Son, Marot, Didier, Bendahmane, Fateh（2021），筛除粗粒对粗粒土力学行为的影响。《国际岩土工程与地质工程杂志》(15)，第64–74页。[He2021] He, Hantao, Zheng, Junxing, Li, Zhaochao（2021），利用物理引擎加速直剪试验模拟。《计算颗粒力学》(8)，第471–492页。[Heider2021] Heider, Yousef, Suh, Hyoung Suk, Sun, WaiChing（2021），一种通过自主设计/改进神经网络实现的离线多尺度非饱和孔隙力学模型。《国际岩土力学数值与分析方法杂志》。[Hilton2013] Hilton, J. E., Tordesillas, A.（2013），低弗劳德数下颗粒床中球形侵入体受到的阻力。《物理评论E》(88)，第062203页。数字对象标识符 10.1103/Phys- RevE.88.062203[Tejada2016] I. G. Tejada, L. Sibille, B. Chareyre（2016），堵塞对颗粒在均质颗粒集合体中输运的作用。《欧洲物理快报》(115)，第54005页。[Duriez2011] J. Duriez, F. Darve, F.V. Donzé（2011），一种基于离散建模的充填岩石节理本构关系。《国际岩石力学与采矿科学杂志》(48)，第458–468页。数字对象标识符 10.1016/j.ijrmms.2010.09.008[Duriez 2013] J. Duriez, F. Darve, F.V. Donzé（2013），应用于岩石节理建模的增量非线性塑性。《国际岩土力学数值与分析方法杂志》(37)，第453–477页。数字对象标识符 10.1002/nag.1105[Duriez2016] J. Duriez, L. Scholtès, F.V. Donzé（2016），不同缺陷性质下翼裂纹扩展的细观力学。《工程断裂力学》(153)，第378–398页。数字对象标识符 10.1016/j.engfracmech.2015.12.034[Duriez2017c] J. Duriez, M. Eghbalian, R. Wan, F. Darve（2017），界面存在时三相颗粒介质中应力的细观力学本质。《固体力学与物理学杂志》(99)，第495–511页。数字对象标识符 10.1016/j.jmps.2016.10.011[Duriez2016b] J. Duriez, R. Wan（2016），通过均匀化和离散元方法研究含界面的湿颗粒介质中的应力。《工程力学杂志》(142)。数字对象标识符 10.1061/(ASCE)EM.1943-7889.0001163[Duriez2017] J. Duriez, R. Wan（2017），湿颗粒土离散元建模中的细节问题。《岩土工程学》(67)，第365–370页。数字对象标识符 10.1680/jgeot.15.P.113[Duriez2017b] J. Duriez, R. Wan（2017），接触角对湿颗粒土的力学影响。《岩土工程学报》(12)，第67–83页。数字对象标识符 10.1007/s11440-016-0500-6[Duriez2018b] J. Duriez, R. Wan（2018），一种用于描述湿颗粒材料应力-应变行为的细观力学 μ UNSAT有效应力表达式。《能源与环境岩土力学》(15)，第10–18页。数字对象标识符 10.1016/j.gete.2017.12.003

[Duriez2018] J. Duriez, R. Wan, M. Pouragha Mehdi, F. Darve，重新探讨湿颗粒土中有效应力存在性的细观力学基础。《国际岩土力学数值与分析方法杂志》(42)，第959–978页。数字对象标识符 10.1002/nag.2774[Duriez2021a]J. Duriez, S. Bonelli（2021），水平集-离散元法（LS-DEM）相对于离散元法的精度和计算成本。《计算机与岩土工程》(134)，第104033页。数字对象标识符 10.1016/j.compgeo.2021.104033[Duriez2021b] Jérôme Duriez, Cédric Galusinski（2021），一种用于精细颗粒形状数值微观岩土力学分析的 YADE 水平集-离散元法。《计算机与地球科学》(157)，第104936页。数字对象标识符 10.1016/j.cageo.2021.104936[Kozicki2012] J. Kozicki, J. Tejchman, Z. Mróz（2012），颗粒粗糙度对采用离散元法进行准静态均质三轴压缩过程中的强度、体积变化、弹性能和耗散能的影响。《颗粒物质》(14)，第457–468页。数字对象标识符 10.1007/s10035-012-0352-1[Zhao2015] J. Zhao, N. Guo（2015），颗粒土中各向异性与应变局部化之间的相互作用：多尺度分析。《岩土工程学》。（审稿中）[DePue2019] Jan De Pue, Gemmina Di Emidio, R. Daniel Verastegui Flores, Adam Bezuijen, Wim M. Cornelis（2019），校准离散元材料参数，以模拟非饱和土在单轴压缩过程中的应力-应变行为。《土壤与耕作研究》(194)，第104303页。数字对象标识符 https://doi.org/10.1016/j.still.2019.104303[Elias2014] Jan Eliáš（2014），采用可破碎多面体颗粒模拟铁路道砟。《粉体技术》(264)，第458–465页。数字对象标识符 10.1016/j.powtec.2014.05.052[Jerier2010b] Jerier, J.-F., Hathong, B., Richefeu, V., Chareyre, B., Imbault, D., Donzé, F.-V., Doremus, P.（2010），采用离散元法研究冷粉末压实。《粉体技术》（印刷中）。数字对象标识符 10.1016/j.powtec.2010.08.056[Jerier2009] Jerier, J.-F., Imbault, D.and Donzé, F.V., Doremus, P.（2009），一种基于四面体网格生成致密多分散球体堆积的几何算法。《颗粒物质》(11)。数字对象标识符 10.1007/s10035-008-0116-0[Jerier2010a] Jerier, J.-F., Richefeu, V., Imbault, D., Donzé, F.V.（2010），为大规模模拟堆积球形离散元。《计算力学与工程方法》。数字对象标识符 10.1016/j.cma.2010.01.016[Thoeni2014]K. Thoeni, A. Giacomini, C. Lambert, S.W. Sloan, J.P. Carter（2014），一种采用柔性防护网系统进行落石分析的三维离散元建模方法。《国际岩石力学与采矿科学杂志》(68)，第107–119页。数字对象标识符 10.1016/j.ijrmms.2014.02.008[Thoeni2013] K. Thoeni, C. Lambert, A. Giacomini, S.W. Sloan（2013），采用随机畸变接触模型对六边形线网进行离散建模。《计算机与岩土工程》(49)，第158–169页。数字对象标识符 10.1016/j.compgeo.2012.10.014[Kozicki2008] Kozicki, J., Donzé, F.V.（2008），一种用于采用离散建模方法进行数值模拟的新型开源软件。《计算力学与工程方法》(197)，第4429–4443页。数字对象标识符 10.1016/j.cma.2008.05.023[Kozicki2009] Kozicki, J., Donzé, F.V.（2009），YADE-OPEN DEM：一种采用离散元法模拟颗粒材料的开源软件。《工程计算》(26)，第786–805页。数字对象标识符 10.1108/02644400910985170[Kozicki2006a] Kozicki, J., Tejchman, J.（2006），脆性材料断裂的二维格构模型。《水利工程与环境力学档案》(53)，第71–88页。[Kozicki2007a] Kozicki, J., Tejchman, J.（2007），采用二维格构模型研究团聚体结构对混凝土断裂过程的影响。《力学档案》(59)，第365–384页。[Kozicki2016] Kozicki, J., Tejchman, J.（2016），采用离散元法研究平面应变压缩过程中二维颗粒涡旋和反涡旋结构。《颗粒物质》(18)。数字对象标识符 10.1007/s10035-016-0627-z

[Kozicki2017] Kozicki, J., Tejchman, J.（2017年），基于离散元法和亥姆霍兹-霍奇向量场分解研究被动土压力条件下二维砂试样中的准静态涡旋结构。《颗粒物质》(19)，第31页。数字对象标识符 10.1007/s10035-017-0714-9[Kozicki2018] Kozicki, J., Tejchman, J.（2018年），基于离散元法与亥姆霍兹-霍奇分解的结合研究三维颗粒试样中涡旋结构与剪切局部化的关系。《颗粒物质》(20)，第48页。数字对象标识符 10.1007/s10035-018-0815-0[Kozicki2014] Kozicki, J., Tejchman, Jacek, Mühlhaus, Hans-Bernd（2014年），采用离散元法对砂土三轴压缩试验进行离散模拟。《国际岩土力学数值与分析方法杂志》(38)，第1923–1952页。数字对象标识符 10.1002/nag.2285[Kozicki2019] Krzaczek, M., Nitka, M., Kozicki, J., Tejchman, J.（2019年），采用完全耦合离散元法/计算流体力学方法，对细观尺度岩体水力压裂进行模拟。《岩土工程学报》。数字对象标识符 10.1007/s11440-019-00799-6[Krzaczek2021] Krzaczek, Marek, Nitka, Michal, Tejchman, J（2021年），采用完全耦合离散元法/计算流体力学方法研究宏观孔隙中的气体含量对岩石水力压裂的影响。《国际岩土力学数值与分析方法杂志》(45)，第234–264页。[Kunhappan2017] Kunhappan, D, Harthong, B, Chareyre, B, Balarac, G, Dumont, PJJ（2017年），惯性流中高长径比柔性纤维的数值建模。《流体物理学》。数字对象标识符 10.3390/min7040056[Widulinski2011]Ł. Widuliński, J. Kozicki, J. Tejchman, D. Leśniewska（2011年），挡土墙土压力问题中砂土剪切带模式形成的离散模拟。《国际固体与结构杂志》(48)，第1191–1209页。数字对象标识符 10.1016/j.ijsolstr.2011.01.005[Lapcevic2017] Lapčević, Veljko, Torbica, Slavko（2017年），崩落岩体摩擦和破碎变化对分段崩落法中重力流形成影响的数值研究。《矿物》(7)。数字对象标识符 10.3390/min7040056[Li2021] Li, Guang-yao, Zhan, Liang-tong, Hu, Zheng, Chen, Yun-min（2021年），颗粒级配和几何体对颗粒土孔隙特征及水分特征曲线的影响：结合离散元法与孔隙网络模型的研究。《颗粒物质》(23)，第1–16页。[Lomine2013] Lominé, F., Scholtès, L., Sibille, L., Poullain, P.（2013年），采用耦合格子玻尔兹曼/离散元方法对颗粒介质中的流固相互作用进行建模：管涌侵蚀应用。《国际岩土力学数值与分析方法杂志》(37)，第577–596页。数字对象标识符 10.1002/nag.1109[Niedostatkiewicz2013] M. Niedostatkiewicz, J. Kozicki, J. Tejchman, H.B. Mühlhaus（2013年），颗粒材料直剪试验的离散建模结果与有限元结果的比较。《颗粒物质》(15)。数字对象标识符 10.1007/s10035-013-0423-y[Nitka2015b] M. Nitka, J. Tejchman, J. Kozicki, D. Leśniewska（2015年），被动土压力条件下颗粒剪切带内细观结构事件的离散元分析。《颗粒物质》(17)。数字对象标识符 10.1007/s10035-015-0558-0[Mahmoodlu2016] Mahmoodlu, MG, Raoof, A, Sweijen, T, van Genuchten, M TH（2016年），砂土压实与混合对孔隙结构及非饱和土水力属性的影响。《包气带》(15)。数字对象标识符 10.2136/vzj2015.10.0136[Mariani20213]Mariani, Marco, Beltrami, Ruben, Brusa, Paolo, Galassi, Carmen, Ardito, Raffaele, Lecis, Nora（2021年），采用粘结剂喷射技术对细氧化铝粉末进行三维打印。《欧洲陶瓷学会杂志》。[Gusenbauer2018] Markus Gusenbauer, Thomas Schrefl（2018年），微流控通道中磁性颗粒的模拟。《磁学与磁性材料杂志》(446)，第185–191页。数字对象标识符 10.1016/j.jmmm.2017.09.031

[Haustein2017]Martin Haustein, Anton Gladkyy, Rüdiger Schwarze（2017年），YADE 中可变形颗粒的离散元建模。SoftwareX (6)，第118–123页。数字对象标识符 https://doi.org/10.1016/j.softx.2017.05.001 ()[Marzougui2015] Marzougui, Donia, Chareyre, Bruno, Chauchat, Julien（2015年），稠密流体-颗粒混合物中剪切应力的微观起源。《颗粒物质》，第1–13页。数字对象标识符 10.1007/s10035-015-0560-6[Maurin2015b] Maurin, R., Chauchat, J., Chareyre, B., Frey, P.（2015年），用于推移质输运的最简流体-离散元耦合模型。《流体物理学》(27)，第113302页。数字对象标识符 10.1063/1.4935703[Maurin2018] Maurin, R., Chauchat, J., Frey, P（2018年），重新审视湍流推移质输运中的坡度影响：对垂向流动结构和输运速率标度关系的影响。《流体力学杂志》(839)，第135–156页。数字对象标识符 10.1017/jfm.2017.903[Maurin2016] Maurin, R., Chauchat, J., Frey, P.（2016年），湍流推移质输运中的密集颗粒流流变学。《流体力学杂志》(804)，第490–512页。数字对象标识符 10.1017/jfm.2016.520[Mede2018] Mede, T., Chambon, G., Hagenmuller, P., Nicot, F.（2018年），一种基于中轴的不规则颗粒形状表征方法，用于离散元模拟。《颗粒物质》(20)，第16页。数字对象标识符 10.1007/s10035-017-0785-7[Montella2016] Montellà, E. P., Toraldo, M., Chareyre, B., Sibille, L.（2016年），颗粒材料中的局部流化：理论与数值研究。《物理评论E》5 (94)，第052905页。数字对象标识符 10.1103/PhysRevE.94.052905[Montella2020] Montellà, Eduard Puig, Yuan, Chao, Chareyre, Bruno, Gens, Antonio（2020年），一种基于孔隙网络方法和格子玻尔兹曼方法的部分饱和介质混合多尺度模型。水资源进展 (144)，第103709页。[Montella2020b] Montellà, EP, Chareyre, B, Salager, S, Gens, A（2020年），静水条件下多组分格子玻尔兹曼方法的基准算例。MethodsX (7)，第101090页。[Guo2015] N. Guo, J. Zhao（2015年），经典岩土力学问题的多尺度见解。《国际岩土力学数值与分析方法杂志》。（审稿中）[Nguyen2021a]Nguyen, Ngoc-Son, Taha, Habib, Marot, Didier（2021年），一种基于德劳内三角剖分的颗粒材料孔隙网络表征新方法。《岩土工程学报》，第1–19页。[Nguyen2021b] Nguyen, Hien Nho Gia, Scholtès, Luc, Guglielmi, Yves, Donzé, Frédéric Victor, Ouraga, Zady, Souley, Mountaka（2021年），流体增压激活的剪切颗粒层的细观力学。《地球物理研究快报》(48)，第e2021GL093222页。数字对象标识符 10.1029/2021GL093222 (e2021GL093222 2021GL093222)[Nicot2011] Nicot, F., Hadda, N., Bourrier, F., Sibille, L., Darve, F.（2011年），颗粒介质中的破坏机制：离散元分析。《颗粒物质》(13)，第255–260页。数字对象标识符 10.1007/s10035-010-0242-3[Nicot2013a] Nicot, F., Hadda, N., Darve, F.（2013年），采用多尺度方法对颗粒材料进行二阶功分析。《国际岩土力学数值与分析方法杂志》。数字对象标识符 10.1002/nag.2175[Nicot2013b] Nicot, F., Hadda, N., Guessasma, M., Fortin, J., Millet, O.（2013年），关于颗粒介质中应力张量的定义。《国际固体与结构杂志》。数字对象标识符 10.1016/j.ijsolstr.2013.04.001[Nicot2012] Nicot, F., Sibille, L., Darve, F.（2012年），率无关颗粒材料中的破坏：向动态状态分岔。《国际塑性杂志》(29)，第136–154页。数字对象标识符 10.1016/j.ijplas.2011.08.002

[**Nitka2015a**] **Nitka, M., Tejchman, J.（2015年），使用离散元法对混凝土在单轴压缩和拉伸下的行为进行建模。《颗粒物质》，第 1–20 页。**[**Ostanin**2020] **Ostanin, Igor A, Oganov, Artem R, Magnanimo, Vanessa（2020），弹性球在简单立方和体心立方排列中的坍塌模式。《物理评论 E》（102），第 032901 页。**[**Jasik**2018] **P. Jasik, J. Kozicki, T. Kilich, J.E. Sienkiewicz, N. E. Henriksen（2018年），KLi 中 $2^1\Pi$ 态的电子结构和转振动预解离。《物理化学化学物理》（20），第 18663–18670 页。数字对象标识符 10.1039/c8cp02551g**[**Papachristos**2017] **Papachristos, E, Scholtès, L, Donzé, F.V, Chareyre, B（2017年），通过水力-力学模拟对水力驱动裂缝进行强度和体积特征分析。《国际岩石力学与采矿科学杂志》（93），第 163–178 页。**[**Pekmezi**2020] **Pekmezi, Gerald, Littlefield, David, Chareyre, Bruno（2020），介观尺度颗粒聚集体弹性模量的统计分布。《国际冲击工程杂志》（139），第 103481 页。**[**Pol**2021] **Pol, Antonio, Gabrieli, Fabio, Brezzi, Lorenzo（2021年），锚固式挂网系统冲切行为的离散元分析：从实验室表征到理想化原位条件。《岩土工程学报》，第 1–21 页。**[**Pouragha**2021] **Pouragha, Mehdi, Kruyt, Niels P, Wan, Richard（2021年），通过应力探测分析研究颗粒材料的非共轴塑性流动。《国际固体与结构杂志》。**

[Pirnia2018]Pouyan Pirnia、François Duhaime、Yannic Ethier、Jean-Sébastien Dubé（2018年），**ICY：COMSOL Multiphysics 与离散元代码 YADE 之间用于多孔介质建模的接口。**《计算机与地球科学》。数字对象标识符https://doi.org/10.1016/j.cageo.2018.11.002

[Puckett2011] Puckett、J.G.、Lechenault、F.、Daniels、K.E.（2011年），**密实颗粒材料中体积分数波动的局部起源。**《物理评论 *E*》（83），第 041301 页。数字对象标识符10.1103/PhysRevE.83.041301

[Aboul2018]R. Aboul Hosn、L. Sibille、N. Benahmed、B. Chareyre（2018年），**一种涉及部分流固耦合的离散数值模型，用于描述土体中的管涌效应。**《计算机与岩土工程》（95），第 30–39 页。数字对象标识符https://doi.org/10.1016/j.compgeo.2017.11.006

[Wan2015] R. Wan, J. Duriez, F. Darve（2015），三相颗粒材料中界面应力的张量描述。《能源与环境岩土力学》（4），第73–87页。数字对象标识符 10.1016/j.gete.2015.11.004[Redaelli2021] Redaelli, Irene, di Prisco, Claudio（2021），关于干燥颗粒试样的离散元数值试验：应变率在演化／非稳态条件下的作用。《颗粒物质》（23），第1–34页。[Caulk2019]Robert A. Caulk, Emanuele Catalano, Bruno Chareyre（2019），通过矩阵分解复用、并行任务管理和GPU计算加速Yade的孔隙力学耦合。《计算机物理通讯》，第106991页。数字对象标识符 https://doi.org/10.1016/j.cpc.2019.106991[Caulk2020b]Robert Caulk, Luc Sholtès, Marek Krzaczek, Bruno Chareyre（2020），颗粒系统的孔隙尺度热–水–力学模型。《计算机方法在应用力学与工程中的应用》（372），第113292页。数字对象标识符 https://doi.org/10.1016/j.cma.2020.113292[Sayeed2011] Sayeed, M.A., Suzuki, K., Rahman, M.M., Mohamad, W.H.W., Razlan, M.A., Ahmad, Z., Thumrongvut, J., Seangatith, S., Sobhan, MA, Mofiz, SA 等（2011），三轴压缩中颗粒材料在极低至高围压下的强度和变形特性。《国际土木与环境工程期刊》IJCEE-IJENS（11）。

[Scholtes2015a]Scholtès, L., Chareyre, B., Michallet, H., Catalano, E., Marzougui, D.（2015），颗粒海床中波浪诱发孔隙压力和有效应力的建模。《连续介质力学与热力学》（27），第305–323页。数字对象标识符 http://dx.doi.org/10.1007/s00161-014-0377-2[Scholtes2009a] Scholtès, L., Chareyre, B., Nicot, F., Darve, F.（2009），具有毛细效应的颗粒材料细观力学。《国际工程科学杂志》（47），第64–75页。数字对象标识符 10.1016/j.ijengsci.2008.07.002[Scholtes2009c] Scholtès, L., Chareyre, B., Nicot, F., Darve, F.（2009），多相颗粒介质中毛细机制的离散建模。《工程与科学计算机建模》（52），第297–318页。[Scholtes2015b]Scholtès, L., Donzé, F., V.（2015），节理岩质边坡阶梯式破坏的离散元分析。《法国科学院力学学报》（343），第155–165页。数字对象标识符 http://dx.doi.org/10.1016/j.crme.2014.11.002[Scholtes2012] Scholtès, L., Donzé, F.V.（2012），使用三维离散元法对裂隙岩体渐进破坏进行建模。《国际岩石力学与采矿科学杂志》（52），第18–30页。数字对象标识符 10.1016/j.ijrmms.2012.02.009[Scholtes2013] Scholtès, L., Donzé, F.V.（2013），软岩与硬岩的离散元模型：颗粒嵌锁对强度的作用。《固体力学与物理学杂志》（61），第352–369页。数字对象标识符 10.1016/j.jmps.2012.10.005[Scholtes2011] Scholtès, L., Donzé, F.V., Khanal, M.（2011），地质材料的强度尺度效应，算例：煤。《固体力学与物理学杂志》（59），第1131–1146页。数字对象标识符 10.1016/j.jmps.2011.01.009[Scholtes2009b] Scholtès, L., Hicher, P.-Y., Chareyre, B., Nicot, F., Darve, F.（2009），关于湿颗粒材料中的毛细应力张量。《国际岩土力学数值与分析方法杂志》（33），第1289–1313页。数字对象标识符 10.1002/nag.767[Scholtes2010] Scholtès, L., Hicher, P.-Y., Sibille, L.（2010），描述颗粒材料中颗粒移除所引发力学响应的多尺度方法。《力学报告》（338），第627–638页。数字对象标识符 10.1016/j.crme.2010.10.003[Shiu2008] Shiu, W., Donzé, F.V., Daudeville, L.（2008），导弹冲击期间混凝土的压实过程：离散元分析。《计算机与混凝土》（5），第329–342页。[Shiu2009] Shiu, W., Donzé, F.V., Daudeville, L.（2009），导弹冲击钢筋混凝土靶体的离散元建模。《国际计算机技术应用杂志》（34），第33–41页。[Sibille2015] Sibille, L., Hadda, N., Nicot, F., Tordesillas, A., Darve, F.（2015），颗粒塑性：离散力学的贡献。《固体力学与物理学杂志》（75），第119–139页。数字对象标识符 10.1016/j.jmps.2014.09.010[Sibille2014] Sibille, L., Lominé, F., Poullain, P., Sail, Y., Marot, D.（2014），颗粒介质中的内部侵蚀：直接数值模拟与能量解释。《水文过程》。数字对象标识符 10.1002/hyp.10351（首次在线发表于2014年10月）[Sobieski2020] Sobieski, Wojciech（2020），计算离散元生成的颗粒床中的二元曲折度。《进程》（8），第1105页。[Suhr2016a]Bettina Suhr, Klaus Six（2016），应力相关的颗粒间摩擦对直剪试验的影响。《粉体技术》（294），第211–220页。数字对象标识符 10.1016/j.powtec.2016.02.029[Suhr2016b] Suhr, Bettina, Six, Klaus（2016），摩擦现象及其对颗粒材料剪切行为的影响。《计算颗粒力学》。数字对象标识符 10.1007/s40571- 016-0119-2[Suhr2017] Suhr, Bettina, Six, Klaus（2017），不同载荷工况下铁路道砟离散元模型的参数化。《颗粒物质》（19），第64页。数字对象标识符 10.1007/s10035-017-0740-7

[Suhr2020] Bettina Suhr、Klaus Six（2020），用于离散元模拟铁路道砟的简单颗粒形状——形状描述符对堆积行为的影响。《颗粒物质》（22）。数字对象标识符 10.1007/s10035-020-1009-0[Sweijen2017b] Sweijen, T、Aslannejad, H、Hassanizadeh, SM（2017），颗粒多孔材料的毛细压力-饱和度关系：孔隙形态法与孔隙单元组装法。《水资源进展》（107），第 22–31 页。数字对象标识符 10.1016/j.advwatres.2017.06.001[Sweijen2017] Sweijen, T、Chareyre, B、Hassanizadeh, SM、Karadimitriou, NK（2017），膨胀颗粒材料的颗粒尺度建模；在超吸水聚合物中的应用。《粉体技术》（318），第 411–422 页。数字对象标识符 10.1016/j.powtec.2017.06.015[Sweijen2016] Sweijen, T、Nikooee, E、Hassanizadeh, S.M、Chareyre, B（2016），膨胀和孔隙率变化对毛细作用的影响：离散元法与孔隙单元组装法耦合。《多孔介质中的传输》（113），第 207–226 页。数字对象标识符 10.1007/s11242-016-0689-8[Sweijen2020] Sweijen, Thomas、Hassanizadeh, S Majid、Chareyre, Bruno（2020），膨胀颗粒堆积体中的非饱和流；一种颗粒尺度模型。《水资源进展》，第 103642 页。[Sweijen2018]Sweijen, Thomas、Hassanizadeh, S Majid、Chareyre, Bruno、Zhuang, Luwen（2018），颗粒多孔介质排水的动态孔隙尺度模型：孔隙单元组装法。《水资源研究》（54），第 4193–4213 页。数字对象标识符 10.1029/2017WR021769[Tian2020] Tian, Yunfu、Yang, Lijun、Zhao, Dejin、Huang, Yiming、Pan, Jiajing（2020），SS316L不锈钢选择性激光熔化的粉末床生成与单道成形的数值分析。《制造工艺杂志》（58），第 964–974 页。[Tong2012] Tong, A.-T.、Catalano, E.、Chareyre, B.（2012），孔隙尺度流动模拟：模型预测与双分散颗粒集合体实验的比较。《油气科学与技术》——IFP Energies nouvelles 评论。数字对象标识符 10.2516/ogst/2012032[Tordesillas2020] Tordesillas, Antoinette、Kahagalage, Sanath、Ras, Charl、Nitka, Michal、Tejchman, Jacek（2020），混凝土、岩石及其他颗粒复合材料中宏观裂纹位置的早期预测。《科学报告》（10），第 1–16 页。[Tran2012] Tran, V.D.H.、Meguid, M.A.、Chouinard, L.E.（2012），一种利用离散元法传播土体不确定性的算法。《岩土工程电子期刊》。[Tran2012c] Tran, V.D.H.、Meguid, M.A.、Chouinard, L.E.（2012），圆形竖井土压力分布的离散元与实验研究。《国际岩土力学杂志》。数字对象标识符 10.1061/(ASCE)GM.1943-5622.0000277[Tran2013]Tran, V.D.H.、Meguid, M.A.、Chouinard, L.E.（2013），用于拉拔荷载条件下土工格栅-土相互作用三维建模的有限-离散元框架。《土工织物与土工膜》（37），第 1–9 页。数字对象标识符 10.1016/j.geotexmem.2013.01.003[Tran2011] Tran, V.T.、Donzé, F.V.、Marin, P.（2011），高三轴荷载下混凝土的离散元模型。《水泥与混凝土复合材料》。[Tran2014] Tran, VDH、Meguid, MA、Chouinard, LE（2014），使用有限-离散元框架对土工格栅加筋土进行三维分析。《国际岩土力学杂志》。[Smilauer2006] Václav Šmilauer（2006），Yade设计的辉煌与困境。《减灾离散元小组年度报告》。[Valera2015] Valera, Roberto Rosello、Morales, Irvin Perez、Vanmaercke, Simon、Morfa, Carlos Recarey、Cortes, Lucia Arguelles、Casanas, Harold Diaz-Guzman（2015），用于生成高体积分数球体堆积的修正算法。《计算颗粒力学》，第 1–12 页。数字对象标识符 10.1007/s40571-015-0045-8

[Linden2016] van der Linden, Joost H.、Narsilio, Guillermo A.、Tordesillas, Antoinette（2016），用于分析多孔颗粒介质中复杂网络输运的机器学习框架：聚焦渗透率。《物理评论E》2（94），第 022904 页。数字对象标识符 10.1103/PhysRevE.94.022904[Wan2014] Wan, R、Khosravani, S、Pouragha, M（2014），湿颗粒土中力传递的微观力学分析。《包气带期刊》（13）。数字对象标识符 10.2136/vzj2013.06.0113[Wang2014] Wang, XiaoLiang、Li, JiaChun（2014），利用修正离散元法模拟颗粒材料的三轴响应。《中国科学：物理学 力学 天文学》（57），第 2297–2308 页。[Xiong2021] Xiong, Hao、Yin, Zhen-Yu、Zhao, Jidong、Yang, Yi（2021），研究流动方向对细颗粒流失的影响及其对间断级配颗粒土的作用。《岩土工程学报》（16），第 399–419 页。[Yuan2018]Yuan, Chao、Chareyre, Bruno、Darve, Félix（2018），理想化颗粒材料排水过程中的变形与应力。《岩土工程学报》（13）。数字对象标识符 10.1016/j.cma.2017.02.024[Yuan2019] Yuan, Chao、Moscariello, Mariagiovanna、Cuomo, Sabatino、Chareyre, Bruno（2019），部分饱和颗粒土浸润诱发坍塌的数值模拟。《颗粒物质》（21），第 64 页。[Zhang2021] Zhang, Lingran、Scholtès, Luc、Donzé, Frédéric Victor（2021），三轴压缩作用下低渗透性岩石渐进破坏过程中的渗透率演化离散元建模。《岩石力学与岩石工程》。数字对象标识符 10.1007/s00603-021-02622-9[Zhao2019] Zhao, Benzhong、MacMinn, Christopher W、Primkulov, Bauyrzhan K、Chen, Yu、Valocchi, Albert J、Zhao, Jianlin、Kang, Qinjun、Bruning, Kelsey、McClure, James E、Miller, Cass T、等（2019），多孔介质多相流孔隙尺度模型的全面比较。《美国国家科学院院刊》（116），第 13799–13806 页。[Zhao2017] Zhao, Shiwei、Zhang, Nan、Zhou, Xiaowen、Zhang, Lei（2017），颗粒形状对颗粒随机堆积组构的影响。《粉体技术》（310），第 175–186 页。[Zhao2021] Zhao, Shiwei、Zhao, Jidong（2021），SudoDEM：释放离散元法对非球形颗粒模拟的预测能力。《计算机物理通讯》（259），第 107670 页。[Zhao2021] Zhao, Yufan、Koizumi, Yuichiro、Aoyagi, Kenta、Yamanaka, Kenta、Chiba, Akihiko（2021），电子束增材制造过程中粉末床在能量吸收与热传递方面的热性质。《粉体技术》（381），第 44–54 页。[Zhong2021] Zhong, Xinran、Sun, WaiChing、Dai, Ying（2021），用于模拟颗粒混合问题的降维显式离散元求解器。《颗粒物质》（23），第 1–13 页。[Stransky2010] Jan Stránský、Milan Jirásek、Václav Šmilauer（2010），颗粒模型的宏观弹性属性。载于《2010年国际建模与仿真会议论文集》，布拉格。[Smilauer2008] Václav Šmilauer（2008），使用 Python 指挥 c++。载于 ALERT 博士生学校演讲。[Smilauer2010a] Václav Šmilauer（2010），Yade：过去、现在与未来。载于格勒诺布尔 3S-R 实验室内部研讨会。（LaTeX 源文件）[Chen2008a] Chen, F.、Drumm, E.C.、Guiochon, G.、Suzuki, K.（2008），使用耦合开源软件模拟颗粒-流体相互作用下的一维向上渗流。载于第12届国际计算机方法与岩土力学进展协会（IACMAG）国际会议论文集，印度果阿。

[*Chen2009b*] *Chen, F., Drumm, E.C., Guiochon, G.*（*2009*），级配填石塌陷坑修复的三维离散元分析：粒径对稳定概率的影响。载于交通研究委员会会议（华盛顿特区）。[*Dang2008a*] *Dang, H.K., Mohamed, M.A.*（*2008*），一种生成具有预设粒径分布的离散元模拟试样的算法。载于第*61*届加拿大岩土工程会议，埃德蒙顿，艾伯塔省。[*Dang2008b*] *Dang, H.K., Mohamed, M.A.*（*2008*），使用离散元法模拟活门问题的三维分析。载于第*61*届加拿大岩土工程会议，埃德蒙顿，艾伯塔省。[*Shiu2007a*] *W. Shiu, F.V. Donze, L. Daudeville*（*2007*），导弹冲击钢筋混凝土靶体的离散元建模。载于材料与结构计算断裂及破坏国际会议（*CFRAC 2007*），南特。[*Smilauer2007a*] *V. Šmilauer*（*2007*），离散与混合模型：在混凝土损伤中的应用。载于未发表文献。[*Kozicki2007c*] *J. Kozicki, J. Tejchman*（*2007*），使用三维格构模型模拟混凝土中的断裂过程。载于材料与结构计算断裂及破坏国际会议（*CFRAC 2007*），南特。[*Kozicki2007d*] *J. Kozicki, J. Tejchman*（*2007*），骨料密度对使用二维离散格构模型模拟混凝土断裂过程的影响。载于计算力学方法会议论文集（*CMM 2007*），罗兹-斯帕瓦。[*Kozicki2007e*] *J. Kozicki, J. Tejchman*（*2007*），使用连续方法和离散方法模拟颗粒体直剪试验。载于计算力学方法会议论文集（*CMM 2007*），罗兹-斯帕瓦。[*Kozicki2007f*] *J. Kozicki, J. Tejchman*（*2007*），使用格构模型研究混凝土拉伸断裂中的尺寸效应。载于《异质材料建模及其在建筑与生物医学工程中的应用》会议论文集（*MHM 2007*），布拉格。[*Kozicki2006b*] *J. Kozicki, J. Tejchman*（*2006*），使用格构模型对脆性材料断裂过程进行建模。载于《混凝土结构计算建模》，*EURO-C*（编者：*G. Meschke, R. de Borst, H. Mang* 和 *N. Bicanic*），*Taylor anf Francis*。[*Kozicki2006c*] *J. Kozicki, J. Tejchman*（*2006*），脆性材料的格构型断裂模型。载于第*35*届固体力学会议（*SOLMECH 2006*），克拉科夫。[*Kozicki2005b*] *J. Kozicki, J. Tejchman*（*2005*），使用离散格构模型模拟混凝土构件中的断裂。载于计算力学方法会议论文集（*CMM 2005*），琴斯托霍瓦，波兰。[*Kozicki2005c*] *J. Kozicki, J. Tejchman*（*2005*），使用离散格构模型模拟混凝土中的裂纹扩展。载于《混凝土与砌体结构的解析模型和新概念》会议论文集（*AMCM 2005*），格利维采，波兰。[*Kozicki2004a*] *J. Kozicki, J. Tejchman*（*2004*），使用离散格构模型研究混凝土断裂过程。载于*CURE* 研讨会《城市工程中的模拟》，格但斯克。[*Kozicki2003a*] *J. Kozicki, J. Tejchman*（*2003*），用于描述准脆性材料和颗粒材料行为的离散方法。载于第*16*届工程力学会议，华盛顿大学，西雅图，*CD–ROM*。[*Kozicki2003c*] *J. Kozicki, J. Tejchman*（*2003*），用于描述准脆性材料行为的格构方法。载于*CURE* 研讨会《建筑材料的有效利用》，索波特。[*Scholtes2008a*] *L. Scholtès, B. Chareyre, F. Nicot, F. Darve*（*2008*），非饱和颗粒材料中的毛细效应建模。载于第*8*届世界计算力学大会暨第*5*届欧洲应用科学与工程计算方法大会，威尼斯。[*Scholtes2009e*] *Scholtes L, Chareyre B, Darve F*（*2009*），部分饱和颗粒材料的微观力学。载于颗粒方法国际会议，*ECCOMAS-Particles*。

[Chareyre2009] Chareyre B., Scholtès L.（2009），密实颗粒材料中毛细现象的微观静力学与微观运动学。载于《POWDERS AND GRAINS 2009：第六届颗粒介质微观力学国际会议论文集》。AIP 会议论文集。DOI 10.1063/1.3180083[Gillibert2009]Gillibert L., Flin F., Rolland du Roscoat S., Chareyre B., Philip A., Lesaffre B., Meyssonier J.（2009），曲率驱动的颗粒分割：应用于X射线显微断层成像雪图像。载于IEEE计算机学会计算机视觉与模式识别会议（迈阿密，美国）。[Hicher2009] Hicher P.-Y., Scholtès L., Chareyre B., Nicot F., Darve F.（2008），关于湿颗粒材料中的毛细应力张量。载于工程力学研究所首届国际会议（EM08）（明尼阿波利斯，美国）。[Scholtes2008b] L. Scholtès, P.-Y. Hicher, F.Nicot, B. Chareyre, F. Darve（2008），关于非饱和颗粒材料中的毛细应力张量。载于EM08：工程力学研究所首届国际会议，明尼阿波利斯。[Scholtes2007a] L. Scholtès, B. Chareyre, F. Nicot, F. Darve（2007），非饱和颗粒介质的微观力学建模。载于ECCOMAS-MHM07论文集，布拉格。[Catalano2009] E. Catalano, B. Chareyre, E. Barthélémy（2009），离散模型中的流固耦合。载于Alert地质材料研讨会2009。[Catalano2010a] E. Catalano, B. Chareyre, E. Barthélémy（2010），用于地质材料流固相互作用分析的耦合模型。载于Alert地质材料研讨会2010。[Catalano2010b] E. Catalano, B. Chareyre, E. Barthélémy（2010），斯托克斯流的孔隙尺度建模。载于GdR MeGe。[Catalano2011a] E. Catalano, B. Chareyre, A. Cortis, E. Barthélémy（2011），用于地质材料的孔隙尺度水力-力学耦合模型。载于第二届基于颗粒方法国际会议——基础与应用。[Sari2011] H. Sari, B. Chareyre, E. Catalano, P. Philippe, E. Vincens（2011），使用耦合离散元-流体方法研究内部侵蚀过程。载于第二届基于颗粒方法国际会议——基础与应用。[Favier2009b] L. Favier, D. Daudon, F. Donzé, J. Mazars（2009），旨在预测雪崩压力的离散元颗粒流模型验证。载于AIP 会议论文集。DOI 10.1063/1.3180002[Thoeni2011] K. Thoeni, C. Lambert, A. Giacomini, S.W. Sloan（2011），落石防护系统的离散建模。载于《Particles 2011：基础与应用》。[Chareyre2011] B. Chareyre, E. Catalano, E. Barthélémy（2011），结合离散元法与有限体积法对水力-力学耦合进行数值模拟。载于自然多孔介质中的流动与力学国际会议：从孔隙尺度到场地尺度——Pore2Field。[Scholtes2011b] L. Scholtès, F. Donzé（2011），节理岩体中的渐进破坏机制：来自三维离散元建模的见解。载于第二届基于颗粒方法国际会议——基础与应用。[Kozicki2011] J. Kozicki, J. Tejchman（2011），使用两种不同颗粒粗糙度描述方法的离散元砂土行为数值模拟。载于第二届基于颗粒方法国际会议——基础与应用。[Stransky2011] J. Stransky, M. Jirasek（2011），使用周期性边界条件单元校准基于颗粒的模型。载于第二届基于颗粒方法国际会议——基础与应用。[Lomine2011] Lominé, F., Sibille, L., Marot, D.（2011），用于研究土壤内部侵蚀的离散元-格子玻尔兹曼耦合方法。载于第二届计算岩土力学国际研讨会（ComGeo II）论文集。

[Lomine2010a] Lominé, F., Scholtès, L., Poullain, P., Sibille, L. (2010年)，内部流体流动引起的土壤微观结构变化：采用耦合离散元/格子玻尔兹曼方法进行研究。载于《地质材料与结构进展欧洲地中海研讨会论文集》，AGS'10。[Nicot2011b] Nicot, F., Hadda, N., Bourrier, F., Sibille, L., Darve, F. (2011年)，颗粒材料塌陷机制的离散元分析。载于《第二届计算岩土力学国际研讨会论文集》（ComGeo II）。[Scholtes2011c] Scholtès, L., Hicher, P.Y., Sibille, L. (2011年)，一种描述土体内部侵蚀效应的微观力学方法。载于《岩土力学与岩土工程：从微观到宏观》论文集，IS-Shanghai 2011。[Sibille2009] Sibille, L., Scholtès, L. (2009)，内部侵蚀对土体力学行为的影响：一种离散元方法。载于《颗粒方法国际会议》论文集，Particles 2009。[Hicher2011] Hicher, P.Y, Scholtès, L., Sibille, L. (2011年)，颗粒移除对颗粒材料行为影响的多尺度建模。载于工程力学研究所论文集，EMI 2011。[Nicot2010] Nicot, F., Sibille, L., Daouadji, A., Hicher, P.Y., Darve, F. (2010年)，颗粒材料失稳的多尺度建模。载于工程力学研究所论文集，EMI 2010。[Lomine2010b] Lominé, F., Poullain, P., Sibille, L. (2010年)，采用耦合格子玻尔兹曼/离散元方法对颗粒介质中的流固相互作用进行建模：水力加载下固体颗粒脱离的应用。载于第19届流体动力学离散模拟会议，DSFD 2010。[Modenese2012] C. Modenese, S. Utili, G.T. Houlsby (2012年)，弹性黏附颗粒的离散元建模及其在月壤中的应用。载于《Earth and Space 2012：挑战性环境中的工程、科学、施工与运营》 ¬© 2012 ASCE。数字对象标识符 10.1061/9780784412190.006[Modenese2012a] Modenese, C, Utili, S, Houlsby, G T (2012年)，使用离散元法研究表面能对月壤力学性质的影响。载于《颗粒介质离散元建模》（Wu, Chuan-Yu 编），英国皇家化学会，数字对象标识符 10.1039/9781849735032-00069[Modenese2012b] Modenese, C, Utili, S, Houlsby, G T (2012年)，颗粒介质准静态条件的数值研究。载于《颗粒介质离散元建模》（Wu, Chuan-Yu 编），英国皇家化学会，数字对象标识符 10.1039/9781849735032-00187[Chareyre2012b] B. Chareyre, L. Scholtès, F. Darve (2012年)，通过离散元模拟研究若干应力张量的性质。载于 Euromech 第539次学术讨论会；《非饱和多孔介质力学：有效应力原理；从微观力学到热力学》[Chareyre2012c] B. Chareyre (2012年)，微观孔隙力学：数值模型的最新进展与展望。载于 ICACM 2012年研讨会，《结构对新兴材料属性的作用》[Michallet2012] H. Michallet, E. Catalano, C. Berni, V. Rameliarison, E. Barthélémy (2012年)，波浪与垂直墙相互作用时砂土液化的物理与数值建模。载于 ICSE6——第六届冲刷与侵蚀国际会议[Hasan2010b]A. Hasan, B. Chareyre, J. Kozicki, F. Flin, F. Darve, J. Meyssonnier (2010年)，基于微观断层成像的离散元建模，用于模拟雪的微观结构变形。载于美国地球物理联合会秋季会议摘要[Tran2012b] Tran, V.D.H, Meguid, M.A, Chouinard, L.E (2012年)，随机场理论与离散元法在岩土工程问题可靠性分析中的耦合。载于加拿大土木工程学会（CSCE）2012年度会议，埃德蒙顿[Tran2012d] Tran, V.D.H, Meguid, M.A, Chouinard, L.E (2012年)，圆形竖井土压力分布的离散元研究。载于加拿大隧道协会（TAC）2012年会议，蒙特利尔

[*Harthong*2013] *Barthélémy Harthong, Richard G Wan (2009)*，颗粒材料中的定向塑性流动与织构依赖性。载于《*Powders and Grains 2013*：第六届颗粒介质微观力学国际会议论文集》。*AIP* 会议论文集。数字对象标识符 *http://dx.doi.org/10.1063/1.4811900*[*Hilton2013b*] *J. E. Hilton, P. W. Cleary, A. Tordesillas (2013)*，颗粒床中的单元黏滑运动。载于《*Powders and Grains 2013*：第六届颗粒介质微观力学国际会议论文集》。*AIP* 会议论文集。数字对象标识符 *10.1063/1.4812063*[*Catalano2013b*] *Catalano E., Chareyre B., Barthélémy E. (2013)*，波浪作用下颗粒沉积物固-流转变的 *DEM-PFV* 分析。载于《*Powders and Grains 2013*：第六届颗粒介质微观力学国际会议论文集》。*AIP* 会议论文集。数字对象标识符 *10.1063/1.4812118*[*Guo*2013] *Ning Guo, Jidong Zhao (2013)*，用于颗粒介质跨尺度模拟的分层模型。载于《*Powders and Grains 2013*：第六届颗粒介质微观力学国际会议论文集》。*AIP* 会议论文集。数字对象标识符 *10.1063/1.4812158*[*Hadda2013b*] *Nejib Hadda, François Nicot, Luc Sibille, Farhang Radjai, Antoinette Tordesillas, Félix Darve (2013)*，颗粒材料破坏的多尺度描述。载于《*Powders and Grains 2013*：第六届颗粒介质微观力学国际会议论文集》。*AIP* 会议论文集。数字对象标识符 *10.1063/1.4811999*[*Kozicki*2013]*Jan Kozicki, Jacek Tejchman, Danuta Lesniewska (2013)*，颗粒剪切带中若干微观结构现象的研究。载于《*Powders and Grains 2013*：第六届颗粒介质微观力学国际会议论文集》。*AIP* 会议论文集。数字对象标识符 *10.1063/1.4811976*[*Maurin*2013] *R. Maurin, B. Chareyre, J. Chauchat, P. Frey (2013)*，推移质输运的离散元建模。载于 *THESIS 2013* 论文集，《地球物理流动中沉积物动力学两相建模》。[*Maurin*2015] *R. Maurin, B. Chareyre, J. Chauchat, P. Frey (2015)*，推移质输运中的颗粒流变学。载于 *CFM-2015*。[*Maurin2016b*] *R. Maurin, J. Chauchat, P. Frey (2016)*，湍流推移质输运中的密集颗粒流流变学。载于 *THESIS 2017* 论文集，《地球物理流动中沉积物动力学两相建模》。[*Frey*2017] *P. Frey, R. Maurin, L. Morchid-Alaoui, S. Gupta, J. Chauchat (2017年)*，采用流体-离散元耦合模型对推移质泥沙输运中的垂向尺寸分选进行数值试验。载于《*Powder and Grains 2017*》论文集。[*Elias*2013] *Eliáš, J. (2013)*，使用多面体单元形状对铁路道砟进行离散元模拟。载于《基于颗粒的方法 *III*：基础与应用——第三届基于颗粒的方法国际会议论文集：基础与应用》，颗粒 *2013*。[*Yuan*2014] *Chao Yuan, Bruno Chareyre, Félix Darve (2014)*，密集球体堆积中两相流排水的孔隙尺度模拟。载于第二十届水资源计算方法国际会议。[*Sweijen*2014] *Thomas Sweijen, Majid Hassanizadeh, Bruno Chareyre (2014)*，膨胀多孔介质的孔隙尺度建模；超吸水聚合物的应用。载于第二十届水资源计算方法国际会议。[*Effeindzourou*2015] *A. Effeindzourou, K. Thoeni, A. Giacomini, S.W. Sloan (2015)*，矿石堆岩石冲击的离散模型。载于《计算方法与岩土力学最新进展》。[*Effeindzourou2015a*] *Effeindzourou, A., Thoeni, K., Chareyre, B., Giacomini, A. (2015)*，一种在离散元模型中对可变形结构建模的通用方法。载于《基于颗粒的方法 *IV*：基础与应用》。

[Thoeni2015] K. Thoeni, A. Giacomini, C. Lambert, S.W. Sloan（2015），采用柔性防护网系统的落石轨迹分析。载于《面向社会与国土的工程地质学》第2卷（G. Lollino、D. Giordan、G.B. Crosta、J. Corominas、R. Azzam、J. Wasowski 和 N. Sciarra 编），施普林格，数字对象标识符 10.1007/978-3-319-09057-3_356[1stYadeWorkshop] B. Chareyre （编），第 1 届 Yade 研讨会演示文稿集（2014）。[2ndYadeWorkshop] J. Duriez（编），第2届 YADE 研讨会：多尺度耦合问题的基于离散方法的建模。摘要集（2018）。[Kozicki2005a] Kozicki, J.（2005），用于描述混凝土断裂过程的离散格构模型。载于《风险缓解离散元小组年度报告1》，法国格勒诺布尔第一大学。[Tejchman2011] Tejchman, J.（2011），采用有限元模型和离散元模型对颗粒体剪切带模式进行比较建模。《岩土材料中的分岔与劣化进展》，第255–260页。[Nitka2015c] Nitka, Michal, Tejchman, Jacek, Kozicki, Jan（2015），颗粒剪切带中微观结构现象的离散建模。载于《新千年岩土材料的分岔与劣化》。[Bonilla2014] Bonilla-Sierra, V, Donzé, FV, Scholtès, L, Elmouttie, M（2014），将摄影测量数据与离散元模型耦合，用于岩质边坡稳定性评估。载于《岩石工程与岩石力学：岩体内及岩体上的结构》。[Skarzynski2014] Ł. Skarżynski, M. Nitka, J. Tejchman（2014），混凝土梁三点弯曲双尺度模型——数值分析与实验。载于《混凝土结构计算建模》。[Nitka2014] Nitka, M, Tejchman, J（2014），采用离散元法对混凝土断裂过程中的微观结构演化进行离散建模。载于《混凝土结构计算建模》。[Sayeed2014] Sayeed, Md Abu, Sazzad, Md Mahmud, Suzuki, Kiichi（2014），考虑围压依赖性的颗粒材料力学行为。载于地质大会2012。[Nicot2015] Nicot, F, Hadda, N, Bourrier, F, Sibille, L, Tordesillas, A, Darve, F（2015），颗粒介质二阶功的微观力学分析。载于《新千年岩土材料的分岔与劣化》。[Albaba2015b] Albaba, Adel, Lambert, Stéphane, Nicot, François, Chareyre, Bruno（2015），颗粒流撞击障碍物的建模。载于《滑坡与泥石流建模的最新进展》。[Hadda2015b] Hadda, N, Bourrier, F, Sibille, L, Nicot, F, Wan, R, Darve, F（2015），对弥散与局部化破坏的微观结构簇描述。载于《从微观到宏观的岩土力学：微观与宏观岩土力学国际研讨会论文集》（IS-Cambridge 2014）。[Bourrier2015b] Bourrier, Franck, Baroth, Julien, Lambert, Stéphane（2015），基于可靠性的方法如何改进落石防护栏的设计？载于《面向社会与国土的工程地质学》第2卷。[Wan2015b] Wan, Richard, Hadda, Nejib（2015），主应力轴旋转下颗粒材料的塑性变形。载于《新千年岩土材料的分岔与劣化》。[Li2014] Li, W, Vincens, E, Reboul, N, Chareyre, B（2014），数值颗粒滤层中细颗粒的缩颈与过滤：材料组构的影响。载于《冲刷与侵蚀：第七届冲刷与侵蚀国际会议论文集》，澳大利亚珀斯，2014年12月2–4日。[Uhlmann2014] Uhlmann, Eckart, Dethlefs, Arne, Eulitz, Alexander（2014），振动光饰中的材料去除与表面形貌形成研究。载于 Procedia CIRP。

[Nitka2014b] Nitka, M, Tejchman, J（2014），采用离散元法对混凝土断裂过程中的微观结构演化进行离散建模。载于《混凝土结构计算建模》。[Nitka2014c] Nitka, M, Tejchman, J, Kozicki, J, Lesniewska, D（2014），平均颗粒直径对颗粒剪切带中涡旋、力链和局部体积变化的影响。载于《数字人道主义者：大数据如何改变人道主义响应的面貌》。[Guo2014b] Guo, Ning, Zhao, Jidong（2015），无黏性砂土应变局部化的多尺度研究。载于《新千年岩土材料的分岔与劣化》（Chau, Kam-Tim 和 Zhao, Jidong 编），数字对象标识符 10.1007/978-3-319-13506-9_18[Toraldo2015] Toraldo, Marcella, Chareyre, Bruno, Sibille, Luc（2015），采用 DEM-PFV 耦合方法对饱和颗粒介质中的局部流化进行数值建模。载于《年度报告1》（格勒诺布尔岩土力学组编）。[Gladky2015a] Gladky, Anton, Lieberwirth, Holger, Schwarze, Ruediger（2015），造粒盘上干燥及弱润湿散料流动的离散元模拟。载于第13届美国全国计算力学大会。[Gladky2015b] Gladky, Anton, Roy, Sudeshna, Weinhart, Thomas, Luding, Stefan, Schwarze, Ruediger（2015），弱润湿颗粒材料的离散元模拟：毛细桥模型的实现。载于第四届基于颗粒方法会议（PARTICLES 2015）。[Suhr2015] Suhr, Bettina, Six, Klaus（2015），颗粒材料中的摩擦学效应及其对宏观材料行为的影响。载于第四届国际基于颗粒方法会议论文集（PARTICLES 2015）。[Darve2016] F. Darve, J. Duriez, R. Wan（2016），岩土力学中的离散元建模：近期的一些突破。载于第七届离散元法国际会议论文集。[Duriez2015]J. Duriez, R. Wan（2015），非饱和颗粒材料中的有效应力：微观力学见解。载于《科学与工程中的耦合问题VI》。[Duriez2017d] J. Duriez, R. Wan, M. Pouragha（2017），部分饱和颗粒材料：微观力学建模的见解。载于第六届Biot孔隙力学会议。[Duriez2017e] J. Duriez, R. Wan, F. Darve（2017），三相颗粒材料应力的微观结构视角。载于《从微观结构研究到多尺度建模：弥合间隙》（D. Brancherie、P. Feissel、S. Bouvier 和 A. Ibrahimbegovic 编），约翰·威立父子公司，数字对象标识符 10.1002/9781119476757.ch6[Duriez2019] J. Duriez, S. Duverger, R. Wan（2019），针对湿颗粒土的微观力学 μUNSAT[方法。载于第八届国际离散元方法会议。2020] Duriez[J. Duriez, C. Galusinski（2020），适用于任意颗粒形状颗粒材料的八叉树水平集表示。载于《2020年流体力学专题问题》论文集。数字对象标识符 10.14311/TPFM.2020.009]Sweijen2017c[Sweijen, T, Hassanizadeh, SM, Aslannejad, H, Leszczynski, S（2017），采用离散元法和多球近似对膨胀颗粒材料进行颗粒尺度建模。载于《第六届Biot孔隙力学会议》。数字对象标识符 10.1061/9780784480779.040]2016] Nikooee[Nikooee, E, Sweijen, T, Hassanizadeh, SM（2016），毛细压力、饱和度和界面面积之间关系的确定：一种孔隙单元组装方法。载于 E3S会议网络。数字对象标识符 10.1051/e3sconf/20160902002]2017] Nikooee[Nikooee, E, Habibagahi, G, Daneshian, B, Sweijen, T, Hassanizadeh, SM（2017），用于预测非饱和土持水特性的颗粒尺度模型。载于第19届国际土力学与岩土工程会议论文集。] Aboul2016bHosn, R Aboul, Sibille, L, Benahmed, N, Chareyre, B（2016），对受劣化作用的土体力学响应进行离散数值描述，受 suf-

融合。在《冲刷与侵蚀：第八届国际冲刷与侵蚀会议论文集》（英国牛津，2016年9月12日至15日）中。[Tejada2016b] Tejada, IG, Sibille, L, Chareyre, B, Vincens, E (2016)，颗粒土中颗粒迁移的数值建模。载于《冲刷与侵蚀：第八届国际冲刷与侵蚀会议论文集》（英国牛津，2016年9月12日至15日）。[Aboul2017] Hosn, Rodaina Aboul, Nguyen, Cong Doan, Sibille, Luc, Benahmed, Nadia, Chareyre, Bruno (2017)，细颗粒流失对土体力学性质影响的微观尺度分析。载于《国际地质材料分岔与劣化研讨会》。[Chareyre2017] Chareyre, Bruno, Yuan, Chao, Montella, Eduard P, Salager, Simon (2017)，颗粒-流体系统多尺度建模的探索。载于《EPJ 网络会议论文集》。[Tejada2017] Tejada, Ignacio G, Sibille, Luc, Chareyre, Bruno, Zhong, Chuheng, Marot, Didier (2017)，颗粒集合体中的颗粒输运多尺度建模。载于《EPJ 网络会议论文集》。[Montella 2017] i Montellà, Eduard Puig, Toraldo, Marcella, Chareyre, Bruno, Sibille, Luc (2017)，从连续介质解析描述到颗粒介质中局部流化的离散数值建模。载于《EPJ 网络会议论文集》。[Yuan2017b] Yuan, Chao, Chareyre, Bruno, Darve, Felix (2017)，悬摆区之外颗粒系统中的流动与变形耦合。载于《EPJ 网络会议论文集》。[Chareyre2017b] B. Chareyre , E. Nikooee , C. Chalak , C. Yuan (2017)，有效应力的微观力学见解。载于《孔隙力学 VI》。DOI 10.1061/9780784480779.051[Stransky2010b] Jan Stránský, Milan Jirásek, Václav Šmilauer (2010)，颗粒模型的宏观性质：单轴拉伸。载于《纳米与宏观力学 2010》。[Stransky2011b] Jan Stránský, Milan Jirásek (2011)，使用周期性边界条件单元对颗粒模型进行非弹性标定。载于《纳米与宏观力学 2011》。[Stransky2012a] Jan Stránský, Milan Jirásek (2012)，离散元法中等效应力张量评估综述。载于《纳米与宏观力学 2012》。[Stransky2013a] Jan Stránský, Milan Jirásek (2013)，离散元周期单元的局部化分析。载于《纳米与宏观力学 2013》。[Stransky2014a] Jan Stránský (2014)，使用离散元法和 YADE 软件生成随机 Wang 瓦片。载于《纳米与宏观力学 2014》。[Stransky2015a] Jan Stránský, Martin Doškář (2015)，基于离散元法的 Wang 瓦片铺砌与 PUC 的比较。载于《纳米与宏观力学 2015》。[Stransky2012b] Jan Stránský, Milan Jirásek (2012)，开源有限元-离散元耦合。载于《工程力学 2012》。[Stransky2013b] Jan Stránský (2013)，有限元-离散元耦合与 MuPIF 框架。载于《工程力学 2013》。[Stransky2014b] Jan Stránský (2014)，有限元法与离散元法的结合及其在铁路道砟-轨枕相互作用中的应用。载于《工程力学 2014》。[Stransky2015b] Jan Stránský, Martin Doškář (2015)，使用离散元法生成随机 Wang 瓦片。载于《工程力学 2015》。[Stransky2014c] Jan Stránský (2014)，铁路轨枕与路基相互作用建模中的 MDP 与 MKP 结合。载于《Juniorstav 2014》。[Stransky2013c] Jan Stránský (2013)，开源离散元-有限元耦合。载于《颗粒 2013》。[Caulk2019a] Caulk, R.A., Chareyre, B. (2019)，用于模拟离散元系统中耦合热-水-力过程的开放框架。载于《第八届国际离散元方法会议》。

[Caulk2019b] Caulk, R.A.„ Kozicki, J., Kunhappan, D.„ Maurin, R.„ Montellà, E.P.„ S weijen, T.„ Yuan, C.„ Chareyre, B. (2019)，Yade（无可否认地）转变为一个专注于优化、多物理场耦合和用户支持的示范项目。载于《第八届国际离散元方法会议》。[Chareyre2019] Chareyre B.„ Montellà, E.P.„ Yuan, C.„ Gens, A. (2019)，一种用于在离散元法中整合不混溶相流动的混合孔隙网络-LBM 方法。载于《第八届国际离散元方法会议》。[Kozicki2007b] J. Kozicki (2007年)，将离散模型用于描述脆性材料的断裂过程。博士论文，格但斯克工业大学。[Catalano2008a] E. Catalano (2008)，渗流对部分饱和边坡的影响——离散元法及其在开源软件 YADE 中的实现。硕士论文，格勒诺布尔第一大学。[Chen2009a] Chen, F. (2009)，开源代码在颗粒多孔介质耦合流动离散元法中的应用。博士论文，田纳西大学诺克斯维尔分校。[Duriez2009a] J. Duriez (2009)，岩体稳定性：一种力学方法。博士论文，格勒诺布尔理工学院。[Scholtes2009d] Luc Scholtès (2009)，**部分饱和颗粒介质的微观力学建模**。博士论文，格勒诺布尔国立理工学院。[Smilauer2010b] Václav Š milauer (2010)，基于离散元法的黏聚颗粒模型及其在 Yade 平台上的应用。博士论文，布拉格捷克理工大学土木工程学院与格勒诺布尔第一大学—约瑟夫·傅立叶 I-MEP2 博士生院。（LaTeX 源文件）[Smilauer2010c] Václav Šmilauer (2010)，博士论文声明。（博士论文摘要）。（LaTeX 源文件）[Jerier2009b] Jerier, J.F. (2009)，采用离散元法对延性金属粉末高密度压缩进行建模（法语）。博士论文，格勒诺布尔第一大学—约瑟夫·傅立叶。[Tran2011b] Van Tieng TRAN (2011)，极端机械荷载作用下的混凝土结构：采用离散元法对局部响应进行建模（法语）。博士论文，格勒诺布尔第一大学—约瑟夫·傅立叶。[Marzougui2011] Marzougui, D. (2011)，采用离散单元和有限体积法对推移质沉积物中的输运与变形进行水力-力学建模。硕士论文，突尼斯国立工程师学院。[Chen2011b] Chen, J. (2011)，热拌沥青混合料压实的离散元法（DEM）分析。博士论文，田纳西大学诺克斯维尔分校。[Favier2009c] Favier, L. (2009)，采用三维离散元数值方法研究颗粒流对障碍物的作用。博士论文，格勒诺布尔第一大学—约瑟夫·傅立叶。[Charlas2013] Benoit Charlas (2013)，用于储氢的金属间氢化物力学行为研究。博士论文，格勒诺布尔大学。[Catalano2012]Emanuele Catalano (2012)，双相颗粒介质的孔隙尺度水力-力学耦合模型。博士论文，格勒诺布尔大学。[Boon2013b] C.W. Boon (2013)，节理岩体的离散元建模：算法及其验证。博士论文，牛津大学。[GaeblerMedack2013] Gäbler, Nils, Medack, Jörg (2013)，溜槽中颗粒动力学的实验与模拟研究。项目报告，弗赖贝格工业大学力学与流体动力学研究所。

[GentzAverhausVilbusch2014] Gentz, Julia、Averhaus, Jan、Vilbusch, Stephan（2014），造粒圆盘上颗粒运动的数值模拟——离散元模型的设置与验证。弗赖贝格工业大学力学与流体动力学研究所项目报告。[Guo2014c] N. Guo（2014），颗粒介质剪切行为的多尺度表征。香港科技大学博士论文。[Borrmann2014] Sebastian, Borrmann（2014），离散元-计算流体力学模拟：在选定软件包中测试新的耦合方法（德文）。弗赖贝格工业大学力学与流体动力学研究所硕士论文。[Medack2014] Medack, Jörg（2014），锤式破碎机溜槽上物料分布局部变化的研究（德文）。弗赖贝格工业大学矿物加工机械研究所硕士论文。[Khosravani2014] S. Khosravani（2014），悬摆区非饱和颗粒介质的有效应力方程。卡尔加里大学土木工程系硕士论文。[Maurin2015PhD] Raphael Maurin（2015），采用欧拉-拉格朗日模型研究床载输运中的颗粒行为。格勒诺布尔-阿尔卑斯大学博士论文。[Morales2012a] D. Morales（2012），基于生产信息的离散元法采空区冒落估算。智利大学硕士论文。[Jakob2016]Christian Jakob（2016），褐煤开采排土场在地下水回升期间及之后液化行为的数值建模（德文，附英文扩展摘要）。弗赖贝格工业大学博士论文。[Stein2017] Gerhard Wolfgang Stein（2017），模块化离散元软件的设计与实现。波鸿鲁尔大学博士论文。[Stransky2018] Jan Stránský（2018），混凝土细观离散元模型及其与有限元法的结合。布拉格捷克理工大学博士论文。[Benniou2016] Hicham Benniou（2016），强冲击下水泥基材料行为的离散元建模：考虑饱和度。工程—材料、力学、能源、环境、工艺与生产博士学院（格勒诺布尔）博士论文。[DePue2019c] De Pue, Jan（2019），土壤压实相关的车辆诱发应力传递建模进展。根特大学博士论文。[Kunhappan2018] Kunhappan, Deepak（2018），惯性流动中长柔性纤维的数值建模。格勒诺布尔-阿尔卑斯大学博士论文。[Gladky2019] Gladky, Anton（2019），采用理想化材料对辊磨机料床受力进行数值研究（德文）。弗赖贝格工业大学矿物加工机械研究所博士论文。[Haustein2021a] Haustein, Martin（2021），以钢和混凝土为例研究悬浮液中的颗粒相互作用（德文）。弗赖贝格工业大学力学与流体动力学研究所博士论文。[Klichowicz2021a] Klichowicz, Michael（2021），基于定量矿物学分析的真实微观结构建模。弗赖贝格工业大学矿物加工机械与再生系统技术研究所博士论文。[Maurin2018a] Maurin, R.（2018），YADE 一维垂向 VANS 流体求解：数值求解细节。Yade 技术档案。[Maurin2018b] Maurin, R.（2018），YADE 一维垂向 VANS 流体求解：理论基础。Yade 技术档案。

[Maurin2018c] Maurin, R.（2018），YADE 一维垂向 VANS 流体求解：验证。Yade 技术档案。[Chareyre2019b] Chareyre, B.、Caulk, R.A.、Chèvremont, W.、Guntz, T.、Kneib, F.、Kunhappen, D.、Pourroy, J.（2019），颗粒系统动力学的 MPI 分布式计算。Yade 技术档案。[Pirnia2019] Pirnia, P.（2019），COMSOL-Yade 接口（ICY）使用指南。Yade 技术档案。[Caulk2018]Caulk, R.（2018），用于研究非均质岩石拉伸破裂的离散元方法随机增强。Yade 技术档案。数字对象标识符 10.5281/zenodo.1202039[Bertrand2008] D. Bertrand、F. Nicot、P. Gotteland、S. Lambert（2008），**双绞六边形网的离散元法数值建模 **。《加拿大岩土工程杂志》（45），第 1104–1117 页。[Bertrand2005] D. Bertrand、F. Nicot、P. Gotteland、S. Lambert（2005），使用离散分析对土工复合材料单元进行建模。《计算机与岩土工程》（32），第 564–577 页。[Chan2011] D. Chan、E. Klaseboer、R. Manica（2011），可变形液滴和气泡之间的液膜排液与聚并。《软物质》（7），第 2235-2264 页。[Chareyre2002a]B. Chareyre、L. Briancon、P. Villard（2002），土工织物在沟槽中的锚固能力：理论建模与实验建模。《国际土工合成材料杂志》（9），第 97–123 页。[Chareyre2005] Bruno Chareyre、Pascal Villard（2005），用于土体-夹杂物问题建模的二维动态桁架单元与离散元方法。《工程力学杂志》（131），第 689–698 页。数字对象标识符 10.1061/(ASCE)0733-9399(2005)131:7(689)[Chareyre2003] Bruno Chareyre（2003），采用离散元法对土体-土工合成材料复合结构行为进行建模——应用于边坡顶部的锚固沟。格勒诺布尔大学博士论文。[Chareyre2002b] B. Chareyre、P. Villard（2002），具有已知宏观属性的弯曲土工合成材料锚固结构的离散元建模。载于《第一届国际 PFC 研讨会论文集》，德国盖尔森基兴。[Chevremont2019] Chèvremont, William、Chareyre, Bruno、Bodiguel, Hugues（2019），摩擦悬浮液流变学的定量研究：大范围黏性数下摩擦系数的影响。《物理评论：流体》（6（4）），第 064302 页。数字对象标识符 10.1103/PhysRevFluids.4.064302[Chevremont2020] William Chèvremont、Hugues Bodiguel、Bruno Chareyre（2020），用于悬浮液数值模拟的润滑接触模型。《粉体技术》（372），第 600–610 页。数字对象标识符 https://doi.org/10.1016/j.powtec.2020.06.001[Villard2004a] P. Villard、B. Chareyre（2004），基于足尺试验和离散元建模的土工合成材料锚固沟设计方法。《加拿大岩土工程杂志》（41），第 1193–1205 页。[Lu1998] Ya Yan Lu（1998），计算对称正定矩阵的对数。《应用数值数学》（26），第 483–496 页。数字对象标识符 10.1016/S0168-9274(97)00103-7[Alonso2004] F. Alonso-Marroquin、R. Garcia-Rojo、H.J. Herrmann（2004），颗粒棘轮效应的微观力学研究。载于《土体循环行为与液化现象》。[McNamara2008] S. McNamara、R. García-Rojo、H. J. Herrmann（2008），颗粒棘轮效应的微观起源。《物理评论 E》（77）。数字对象标识符 10.1103/PhysRevE.77.031304[GarciaRojo2004] R. García-Rojo、S. McNamara、H. J. Herrmann（2004），用于颗粒棘轮效应微观力学研究的离散元方法。载于《ECCOMAS 2004 会议论文集》。

[Allen1989] M. P. Allen, D. J. Tildesley（1989），《液体的计算机模拟》。克拉伦登出版社。[DeghmReport2006] F. V. Donzé（编），《2006 年度报告》（2006）。灾害缓解离散元小组。约瑟夫·傅里叶大学，格勒诺布尔[Pournin2001] L. Pournin, Th. M. Liebling, A. Mocellin（2001），用于更好地控制密集颗粒介质数值模拟中能量耗散的分子动力学力模型。《物理评论E》（65），页码 011302。数字对象标识符 10.1103/PhysRevE.65.011302[Jung1997] Derek Jung, Kamal K. Gupta（1997），用于碰撞检测的基于八叉树的分层距离图。《机器人系统期刊》（14），页码 789–806。数字对象标识符 10.1002/(SICI)1097- 4563(199711)14:11<789::AID-ROB3>3.0.CO;2-Q[Hubbard1996] Philip M. Hubbard（1996），用球体近似多面体以实现时间关键型碰撞检测。《ACM 图形学汇刊》（15），页码 179–210。数字对象标识符 10.1145/231731.231732[Klosowski1998] James T. Klosowski, Martin Held, Joseph S. B. Mitchell, Henry Sowizral, Karel Zikan（1998），使用 k-DOP 包围体层次结构进行高效碰撞检测。《IEEE 可视化与计算机图形学汇刊》（4），页码 21–36。[Munjiza2006] A. Munjiza, E. Rougier, N. W. M. John（2006），MR 线性接触检测算法。《国际工程数值方法期刊》（66），页码 46–71。数字对象标识符 10.1002/nme.1538[Munjiza1998] A. Munjiza, K. R. F. Andrews（1998），适用于相似尺寸刚体的 NBS 接触检测算法。《国际工程数值方法期刊》（43），页码 131–149。数字对象标识符 10.1002/(SICI)1097-0207(19980915)43:1<131::AID-NME447>3.0.CO;2-S[Verlet1967] Loup Verlet（1967），关于经典流体的计算机“实验”。I. Lennard-Jones 分子的热力学性质。《物理评论》（159），页码 98。数字对象标识符 10.1103/PhysRev.159.98[Luding2008] Stefan Luding（2008），离散元法简介。载于《欧洲环境与土木工程期刊》。[Wang2009] Yucang Wang（2009），一种模拟三维键合刚体及其旋转动力学的新算法。《岩土工程学报》（4），页码 117–127。数字对象标识符 10.1007/s11440-008- 0072-1[Omelyan1999]Igor P. Omelyan（1999），一种新的旋转运动蛙跳积分器：修订的角动量方法。《分子模拟》（22）。数字对象标识符 10.1080/08927029908022097[Neto2006] Natale Neto, Luca Bellucci（2006），一种用于刚体分子动力学的新算法。《化学物理》（328），页码 259–268。数字对象标识符 10.1016/j.chemphys.2006.07.009[Johnson2008] Scott M. Johnson, John R. Williams, Benjamin K. Cook（2008），用于粒子方法的基于四元数的刚体旋转积分算法。《国际工程数值方法期刊》（74），页码 1303–1313。数字对象标识符 10.1002/nme.2210[Addetta2001] G.A. D’Addetta, F. Kun, E. Ramm, H.J. Herrmann（2001），从实体到颗粒材料——地质材料断裂与破碎过程的离散元模拟。载于《黏聚摩擦材料的连续与非连续建模》。[Pfc3dManual30] ICG（2003），PFC3D（三维颗粒流程序）《理论与背景手册》，版本 3.0。Itasca 咨询集团。[Hentz2003] Séebastien Hentz（2003），《离散元法模拟受冲击钢筋混凝土结构》。格勒诺布尔第一大学——约瑟夫·傅里叶博士论文。[Price2007] Mathew Price, Vasile Murariu, Garry Morrison（2007），真实颗粒的球团生成与轨迹比较。载于《2007 年离散元建模会议论文集》。

[Kuhl2001] E. Kuhl, G. A. D'Addetta, M. Leukart, E. Ramm（2001），黏聚摩擦材料的微平面建模与颗粒建模。载于《黏聚摩擦材料的连续与非连续建模》。数字对象标识符 10.1007/3-540-44424-6_3[Thornton1991] Colin Thornton, K. K. Yin（1991年），有黏附和无黏附弹性球体的冲击。《粉体技术》（65），页码 153–166。数字对象标识符 10.1016/0032-5910(91)80178-L[Thornton2000] Colin Thornton（2000年），颗粒介质偏应力剪切变形的数值模拟。《岩土工程学》（50），页码 43–53。数字对象标识符 10.1680/geot.2000.50.1.43[CundallStrack1979] P.A. Cundall, O.D.L. Strack（1979年），颗粒集合体的离散数值模型。《岩土工程学》（），页码 47–65。数字对象标识符 10.1680/geot.1979.29.1.47[cgal] Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Pion, Monique Teillaud, Mariette Yvinec（2002年），CGAL 中的三角剖分。《计算几何：理论与应用》（22），页码 5–19。[Satake1982] M. Satake（1982年），颗粒材料中的织构张量。载于《IUTAM 颗粒材料变形与破坏专题研讨会论文集》，代尔夫特，荷兰。[Hentz2004a]S. Hentz, F.V. Donzé, L.Daudeville（2004年），高应变率动态加载下混凝土的离散元建模。《计算机与结构》（82），页码 2509–2524。数字对象标识符 '10.1016/j.compstruc.2004.05.016< http://dx.doi.org/10.1016/j.compstruc.2004.05.016 >'_[Hentz2004b] S. Hentz, L. Daudeville, F.V. Donzé（2004年），混凝土离散元模型的识别与验证。美国土木工程师学会《工程力学杂志》（130），页码 709–719。数字对象标识符 10.1061/(ASCE)0733-9399(2004)130:6(709)[Camborde2000a] F. Camborde, C. Mariotti, F.V. Donzé（2000年），通过离散元建模研究岩石与混凝土的行为。《计算机与岩土工程》（27），页码 225–247。[Donze1999a] F.V. Donzé, S.A. Magnier, L. Daudeville, C. Mariotti, L. Davenne（1999年），用离散元法研究混凝土在高应变率压缩下的行为。《国际地球物理学杂志》（122），页码 1154–1163。数字对象标识符 10.1016/S0266-352X(00)00013-6[Magnier1998a]S.A. Magnier, F.V. Donzé（1998年），使用离散元法进行冲击数值模拟。《黏聚-摩擦材料力学》（3），页码 257–276。数字对象标识符 10.1002/(SICI)1099-1484(199807)3:3<257::AID-CFM50>3.0.CO;2-Z[Donze1995a] F.V. Donzé, S.A. Magnier（1995年），脆性行为三维数值模型的构建。《国际地球物理学杂志》（122），页码 790–802。[Donze1994a] F.V. Donzé, P. Mora, S.A. Magnier（1994年），断层与剪切带的数值模拟。《国际地球物理学杂志》（116），页码 46–52。[Hentz2005a] S. Hentz, F.V. Donzé, L.Daudeville（2005年），岩石冲击下钢筋混凝土结构的离散元建模。《意大利岩土工程杂志》（XXXIX），页码 83–94。[Donze2004a] F.V. Donzé, P. Bernasconi（2004年），使用离散元法模拟竖井掘进中的爆破模式。《电子岩土工程杂志》（9），页码 1–44。[Kettner2011] Lutz Kettner, Andreas Meyer, Afra Zomorodian（2011年），dD 同向盒的相交序列。载于《CGAL 用户与参考手册》。[Pion2011] Sylvain Pion, Monique Teillaud（2011年），三维三角剖分。载于《CGAL 用户与参考手册》。[Calvetti1997] Calvetti, F., Combe, G., Lanier, J.（1997年），二维颗粒材料的实验微观力学分析：结构演化与加载路径之间的关系。《黏聚-摩擦材料力学》（2），页码 121–163。

[Bagi2006] Katalin Bagi（2006），颗粒集合体微观结构应变张量分析。《国际固体与结构杂志》（43），第 3166–3184 页。数字对象标识符 10.1016/j.ijsolstr.2005.07.016[Weigert1999] Weigert、Tom、Ripperger、Siegfried（1999），根据半填充角计算液桥体积和整体饱和度。《颗粒与颗粒系统表征》（16），第 238–242 页。数字对象标识符 10.1002/(SICI)1521-4117(199910)16:5<238::AID- PPSC238>3.0.CO;2-E[Willett2000] Willett、Christopher D.、Adams、Michael J.、Johnson、Simon A.、Seville、Jonathan P. K.（2000），两个球形物体之间的毛细桥。《Langmuir》（16），第 9396–9405 页。数字对象标识符 10.1021/la000657y[Herminghaus2005] Herminghaus, S.（2005），湿颗粒物质的动力学。《物理学进展》（54），第 221–261 页。数字对象标识符 10.1080/00018730500167855[Rabinov2005] RABINOVICH Yakov I.、ESAYANUR Madhavan S.、MOUDGIL Brij M.（2005），固定体积液桥在两个球体之间产生的毛细力：理论与实验。《Langmuir》（21），第 10992–10997 页。（英文）[Luding2008b] Luding, Stefan（2008），黏聚性、摩擦性粉体：拉伸接触模型。《颗粒物质》（10），第 235–246 页。数字对象标识符 10.1007/s10035-008-0099-x[Zhou1999536] Y.C. Zhou、B.D. Wright、R.Y. Yang、B.H. Xu、A.B. Yu（1999），沙堆形成动力学模拟中的滚动摩擦。《物理学A辑：统计力学及其应用》（269），第 536–553 页。数字对象标识符 10.1016/S0378-4371(99)00183-1[Antypov2011]D. Antypov、J. A. Elliott（2011），关于阻尼赫兹弹簧的解析解。《欧洲物理快报》（94），第 50004 页。[Ivars2011] Diego Mas Ivars、Matthew E. Pierce、Caroline Darcel、Juan Reyes-Montes、David O. Potyondy、R. Paul Young、Peter A. Cundall（2011），**节理岩体建模的合成岩体方法**。《国际岩石力学与采矿科学杂志》（48），第 219–244 页。数字对象标识符 10.1016/j.ijrmms.2010.11.014[Potyondy2004] D.O. Potyondy、P.A. Cundall（2004），**岩石的黏结颗粒模型**。《国际岩石力学与采矿科学杂志》（41），第 1329–1364 页。数字对象标识符 10.1016/j.ijrmms.2004.09.011[Radjai2011] Radjai, F.、Dubois, F.（2011），《颗粒材料离散元建模》。约翰·威立父子出版公司。[Schwager2007] Schwager, Thomas、Pöschel, Thorsten（2007），重新审视恢复系数与线性阻尼器模型。《颗粒物质》（9），第 465–469 页。数字对象标识符 10.1007/s10035-007-0065-z[Lambert2008] Lambert, Pierre、Chau, Alexandre、Delchambre, Alain、Régnier, Stéphane（2008），两种毛细力模型的比较。《Langmuir》（24），第 3157–3163 页。[Mueller2003]Müller, Matthias、Charypar, David、Gross, Markus（2003），面向交互式应用的基于粒子的流体模拟。载于《2003 年 ACM SIGGRAPH/Eurographics 计算机动画研讨会论文集》。[Soulie2006] Soulié, F.、Cherblanc, F.、El Youssoufi, M.S.、Saix, C.（2006），液桥对多分散颗粒材料力学行为的影响。《国际岩土力学数值与分析方法杂志》（30），第 213–228 页。数字对象标识符 10.1002/nag.476[Mani2013] Mani, Roman、Kadau, Dirk、Herrmann, HansJ.（2013），剪切非饱和颗粒介质中的液体迁移。《颗粒物质》（15），第 447–454 页。数字对象标识符 10.1007/s10035-012-0387-3[Richardson1954] Richardson, J. F.„ W. N. Zaki（1954），沉降与流态化：第一部分。《化学工程师学会会刊》（32）。

[RevilBaudard2013] Revil-Baudard, T.、Chauchat, J.（2013），基于密集颗粒流流变学的片流状态两相模型。《地球物理研究杂志：海洋》（118），第 619–634 页。[RevilBaudard2015] Revil-Baudard, T.、Chauchat, J.、Hurther, D.、Barraud, P-A.（2015），基于新型声学高分辨率速度与浓度测量对片流过程的研究。《流体力学杂志》（767），第 1–30 页。数字对象标识符 10.1017/jfm.2015.23[Li1995] Li, L.、Sawamoto, M.（1995），振荡流作用下片流状态泥沙输运的多相模型。《日本海岸工程》（38），第 157–178 页。[Schmeeckle2007] Schmeeckle, Mark W.、Nelson, Jonathan M.、Shreve, Ronald L.（2007），近底湍流中静止颗粒所受的力。《地球物理研究杂志：地球表面》（112）。数字对象标识符 10.1029/2006JF000536[Wiberg1985] Wiberg, Patricia L.、Smith, J. Dungan（1985），水中跃移颗粒的理论模型。《地球物理研究杂志：海洋》（90），第 7341–7354 页。[Dallavalle1948] J. M. DallaValle（1948年），《微粒测量学：细颗粒技术》。Pitman 出版公司。[Monaghan1992] Monaghan, J.~J.（1992），平滑粒子流体动力学。《天文学与天体物理学年度评论》（30），第 543–574 页。数字对象标识符 10.1146/annurev.aa.30.090192.002551[Morris1997] Morris, Joseph P、Fox, Patrick J、Zhu, Yi（1997），使用光滑粒子流体动力学模拟低雷诺数不可压缩流。《计算物理学杂志》（136），第 214–226 页。数字对象标识符 http://dx.doi.org/10.1006/jcph.1997.5776[Lucy1977] Lucy, L.~B.（1977），检验裂变假说的数值方法。《天文学杂志》（82），第 1013–1024 页。数字对象标识符 10.1086/112164[Monaghan1985] Monaghan, J.~J.、Lattanzio, J.~C.（1985），天体物理问题的改进粒子方法。《天文学与天体物理学》（149），第 135–143 页。[yade:doc] V. Šmilauer、E. Catalano、B. Chareyre、S. Dorofeenko、J. Duriez、A. Gladky、J. Kozicki、C. Modenese、L. Scholtès、L. Sibille、J. Stránský、K. Thoeni（2010），Yade 文档。Yade 项目。（http://yade-dem.org/doc/）[yade:manual] V. Šmilauer、A. Gladky、J. Kozicki、C. Modenese、J. Stránský（2010），Yade：使用与编程。载于《Yade 文档》（V. Šmilauer 编），Yade 项目，第 1 版。（http://yade-dem.org/doc/）[yade:background] V. Šmilauer、B. Chareyre（2010），Yade 离散元法公式。载于《Yade 文档》（V. Šmilauer 编），Yade 项目，第 1 版。（http://yade-dem.org/doc/formulation.html）[yade:reference] V. Šmilauer、E. Catalano、B. Chareyre、S. Dorofeenko、J. Duriez、A. Gladky、J. Kozicki、C. Modenese、L. Scholtès、L. Sibille、J. Stránský、K. Thoeni（2010），Yade 参考文档。载于《Yade 文档》（V. Šmilauer 编），Yade 项目，第 1 版。（http://yade-dem.org/doc/）[Tonon2005] Tonon, F.（2005），用顶点坐标表示的三维四面体惯性张量显式精确公式。《数学与统计学杂志》（1），第 135–143 页。[Lourenco 1994] P B Lourenço 1994 使用界面单元分析砌体结构：理论与应用（报告编号 03-21-22-0-01，代尔夫特理工大学土木工程学院）[Hazzard2000] Hazzard, J.F、Young, R.P（2000），模拟岩石黏结颗粒模型中的声发射。《国际岩石力学与采矿科学杂志》（37），第 867–872 页。数字对象标识符 10.1016/S1365-1609(00)00017-4[Hazzard2013] Hazzard, J. F.、Damjanac, Branko（2013），对黏结颗粒模型中微震活动的进一步研究。第三届国际 FLAC/DEM 研讨会，第 1–11 页。

[Scholz2003] Scholz, C. H., Harris, R. A. (2003),《地震与断层力学》。剑桥大学出版社。[Jackson2000] R. Jackson (2000年),《流化颗粒动力学》。剑桥大学出版社。[Maurin2018_VANSbasis] R. Maurin (2018年), YADE 一维垂向 VANS 流体求解：理论基础。Yade 技术档案。[Maurin2018_VANSfluidResol] R. Maurin (2018年), YADE 一维垂向 VANS 流体求解：流体求解详情。Yade 技术档案。[Maurin2018_VANSvalidations] R. Maurin (2018年), YADE 一维垂向 VANS 流体求解：验证。Yade 技术档案。[Berger1987]Berger, M. J., Bokhari, S. H. (1987),《多处理器上非均匀问题的分区策略》。IEEE 计算机汇刊 (36)，第 570-580 页。数字对象标识符 10.1109/TC.1987.1676942[Fleissner2007] Fleissner, Florian, Eberhard, Peter (2007年),《采用分层粒子分组策略的并行负载均衡颗粒模拟》。发表于 IUTAM 多体系统接触多尺度问题研讨会。

Python 模块索引

__yade._libVersions, 401
yade._log, 403yade._math, 406yade._minieigenHP, 427yade._packObb, 490
yade._packPredicates, 487yade._packSpheres, 483yade._utils, 509

b
yade.bodiesHandling, 386

e
yade.export, 387

g
yade.geom, 392
yade.gridpfacet, 395

l
yade.libVersions, 399
yade.linterpolation, 402yade.log, 403

米
yade.math, 404
yade.minieigenHP, 427yade.mpy, 474

p
yade.pack, 479
yade.plot, 490
yade.post2d, 494

q
yade.qt, 497
yade.qt._GLViewer, 497

t
yade.timing, 499

u
yade.utils, 500

y
yade.ymport, 516