arXiv:2601.07065v1 [stat.CO] 11 Jan 2026

FORMULACOMPILER.JL 和 MARGINS.JL：Julia 中的高效边际效应

Julia 中的边际效应

Eric Feltham哥伦比亚大学数据科学研究所
与社会学系

eric.feltham@aya.yale.edu

2026年1月13日

摘要

边际效应分析对于解释统计模型至关重要，但现有实现面临计算约束，限制了大规模分析。我们推出了两个 Julia 软件包来弥补这一不足。Margins.jl 提供简洁的双函数 API，以 2×2框架组织分析：按评估情境（总体与概况）和分析目标（效应与预测）划分。该软件包支持通过二阶差分进行交互作用分析、弹性度量、用于构建代表性概况的分类混合，以及稳健标准误。FormulaCompiler.jl 提供计算基础，将统计公式转换为零分配、类型特化的求值器，使每行计算达到 $O(p)$，且不受数据集大小影响。这两个软件包合计实现了 622 倍的平均加速比和 460 倍的内存用量减少，相较于 R 的 marginaleffects 包，在 500,000 个观测值上成功计算平均边际效应和 Delta 方法标准误，而 R 因内存耗尽无法完成计算——为 Julia 的统计生态系统提供了首个全面且高效的边际效应实现。

关键词 边际效应 •Julia • 统计计算 • 自动微分 • 公式编译 • 均值处的边际效应 • 平均边际效应

1 引言

社会科学、经济学和生物统计学领域的研究人员越来越多地使用统计模型的边际效应，从对系数的解释转向能够直接回答实质性问题的量（Williams 2012；NortonandDowd2018；McCabe 等，2022；Arel-Bundock、Greifer 和 Heiss，2024）。

Julia（Bezanson 等，2017）的统计生态系统为统计公式（StatsModels.jl）提供了框架，并提供了用于拟合广义线性模型（GLM.jl）和混合效应模型（MixedModels.jl）的软件包。MixedModels.jl 软件包实现了与 Bates 等（2015）为 R 中的 Lme4 软件包所描述的相同计算方法，独立的基准测试表明，对于某些模型设定，其加速比约为 60 倍（Markwick 2022）。然而，此前一直没有一个完善的软件包能够计算边际效应，以补充这些软件包的功能。

此外，R 中现有的实现面临计算约束，限制了大规模分析。R 的 margins 软件包（Leeper [2014] 2025）在处理大型数据集时存在性能限制，其中方差估计被认为是计算成本最高的操作。尽管 R 较新的 marginaleffects 包（Arel-Bundock、Greifer 和 Heiss，2024年）实现了显著的性能提升——在某些操作上比 margins 快达 1000 倍——但它仍受限于 $O(n)$ 内存扩展：通过德尔塔法计算标准误需要一个具有 n 行（每个观测对应一行）[1]，的雅可比矩阵，而 R 底层的 `predict()` 方法通常会将其具体化

[1] **Margins.jl** 通过将梯度逐行累积到一个 $O(p)$ 缓冲区中，而不是将完整的 $n \times p$雅可比矩阵实例化，从而避免了这一问题。

为稠密设计矩阵，通过 `model.matrix()`（Chambers 1992）。Python 的 statsmodels（Seabold 和 Perktold，2010年）面临类似约束：`get_margeff()` 会创建多个中间数组，其大小随 n 扩展，导致在大型数据集上计算平均边际效应时内存耗尽。Python 版 marginaleffects 已经存在（Arel-Bundock、Greifer 和 Heiss，2024年），但继承了相同的扩展特性。这些约束可能导致方法上的妥协：计算均值处的边际效应（MEM），而不是平均边际效应（AME）（Williams 2012）；对大型数据集进行子抽样；或省略标准误计算。

本文提出两项贡献：（1）一个面向 Julia 的全面边际效应软件包；（2）一种解决现有实现内存限制的架构方法。两个互补的软件包共同实现了这两项贡献。**Margins.jl** 为边际效应分析提供统计接口，包括交互项和分类变量对比，并通过双函数 API 将总体平均分析（`population_margins()`）与特定剖面分析（`profile_margins()`）区分开来。这种设计明确呈现了单函数接口中容易被掩盖的区别：一种是在观测到的总体分布上对效应取平均，另一种是在具有代表性的协变量剖面处评估效应。**FormulaCompiler.jl** 通过位置映射编译提供计算基础，将兼容 StatsModels.jl 的公式转换为零分配、类型特化的求值器。这些软件包与 Julia 统计生态系统（GLM.jl、MixedModels.jl、StatsModels.jl）集成，同时保留了用户熟悉的公式接口。图 1展示了这一双层架构。将 **Margins.jl** 与 R 的 marginaleffects 包进行比较的基准测试表明，其速度和内存使用均有显著改善，详情见“性能基准测试”一节。

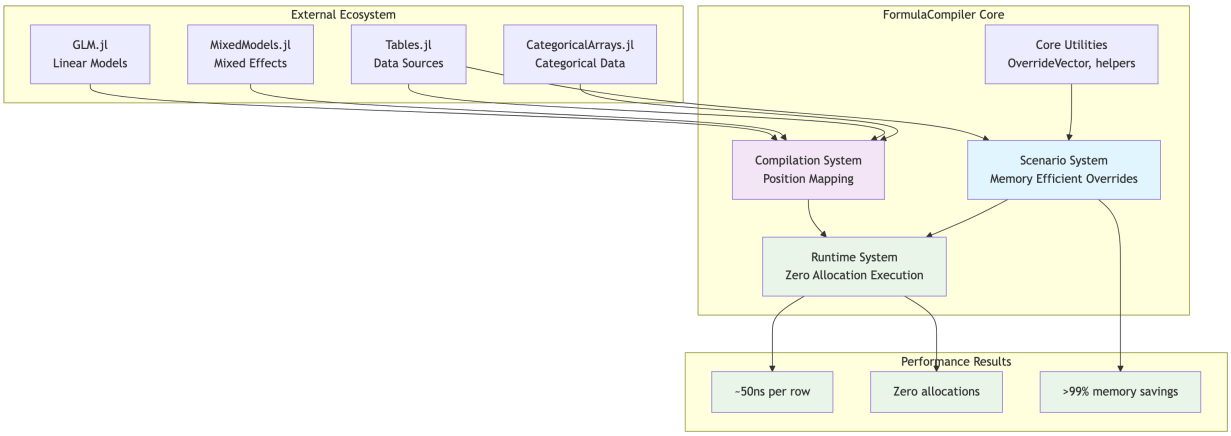


图 1：系统架构图，展示 Margins.jl 如何在 FormulaCompiler.jl 的计算引擎之上提供统计接口，并与 Julia 的统计生态系统集成，包括 GLM.jl、MixedModels.jl 和 StatsModels.jl。

本文其余部分安排如下。首先介绍边际效应分析的统计框架，该框架构成了**Margins.jl** 设计的基础。随后介绍 **Margins.jl** 应用程序接口及其 2×2 组织结构。接下来一节将详细说明**FormulaCompiler.jl**如何通过位置映射编译和零分配求值实现其性能特征。我们将展示基准测试结果，并与 R 的 marginaleffects 包进行性能比较；之后通过案例研究说明典型的分析工作流程。最后，我们将总结本文的贡献并探讨未来的研究方向。

2 相关工作

2.1 边际效应软件

边际效应的计算在各种统计环境中有着悠久的软件发展历史。Stata 的 `margins` 命令（Williams 2012）确立了方法学标准，在其集成环境中提供了全面的功能，用于计算平均边际效应（AME）、均值处的边际效应（MEM）以及调整后预测值。

在 R 中，margins 软件包（Leeper [2014] 2025）将 Stata 的功能移植给 R 用户，并确立了影响后续开发的 API 约定。emmeans 软件包（Lenth 等，2025）为析因设计和估计边际均值引入了参考网格框架。最近，marginaleffects（Arel-Bundock、Greifer 和 Heiss，2024）显著提升了性能（速度最高可比 margins 快 1000 倍）。然而，即使是这一最先进的实现，仍受 $O(n)$ 内存扩展性的限制：德尔塔法的雅可比矩阵会随数据集大小增长，而 R 的 `predict()` 方法会在内部将设计矩阵实体化。

Python 的 statsmodels（Seabold 和 Perktold，2010）通过特定于模型的`get_margeff()`方法提供边际效应计算，但其支持仅限于离散选择模型，并且并非所有评估模式都支持标准误计算。

在 Julia 中，Effects.jl（Alday 和 Kleinschmidt，2024）提供参考网格预测和估计边际均值，但仅限于基于轮廓的预测——它无法计算边际效应（导数），也不支持总体平均量。

2.2 公式系统

基于 Wilkinson 表示法（Wilkinson 和 Rogers，1973）的统计公式接口是模型设定的标准方法。R 的`model.matrix()`（Chambers 和 Hastie，1992）、Python 的 Patsy（Smith，2011）以及 Julia 的 StatsModels.jl（Kleinschmidt 等，2024）都实现了这一框架的变体。这些实现遵循一种通用模式：先解析公式设定，然后为整个数据集生成完整的设计矩阵 $O(n \times p)$。这种方法适用于模型估计，但对于涉及重复逐行求值的工作负载（包括边际效应计算），会带来显著的额外开销。

2.3 定位

Margins.jl 和 **FormulaCompiler.jl** 致力于弥合统计易用性与计算性能之间的差距。与现有的边际效应软件包不同，**Margins.jl** 明确围绕 2×2 评估情境与分析目标框架来组织分析，从而清晰说明所计算的量是什么。与现有公式系统不同，**FormulaCompiler.jl** 通过位置映射编译，避免物化设计矩阵，从而实现 $O(p)$ 逐行求值。这两个软件包相结合，使得在现有实现中难以或无法进行的大规模分析成为可能，同时保留了熟悉的接口。

3 MarginalEffects：统计框架

本节为边际效应分析奠定统计基础，介绍组织**Margins.jl**设计的 2×2 框架。

我们考虑使用 StatsModels.jl 公式表示的模型。令 p 表示对比展开后固定效应项的数量。

- 线性模型： $E[y_i] = x_i^T \beta$，其中 $x_i \in \mathbb{R}^p$
- 广义线性模型： $\mu_i = g^{-1}(\eta_i)$， $\eta_i = x_i^T \beta$
- 混合模型：我们仅针对固定效应，从 MixedModels.jl 拟合结果中提取 $x_i^T \beta$

分类预测变量会根据所选的对比编码展开。交互项、变换和布尔项遵循标准的公式语义。

边际效应表示解释变量每增加一个单位，结果变量的期望变化量。对于连续协变量，其形式化定义为 $\frac{\partial E[Y|X]}{\partial x_k}$，即条件期望关于协变量 x_k 的偏导数。对于分类变量，边际效应通过各水平之间的对比计算得出。

对于链接函数为 g 的广义线性模型，在响应尺度上计算边际效应时需要用到链式法则：

$$\frac{\partial \mu}{\partial x_k} = \frac{d\mu}{d\eta} \cdot \frac{\partial \eta}{\partial x_k} = g'^{-1}(\eta) \cdot \frac{\partial (x^T \beta)}{\partial x_k}$$

其中，$(g^{-1})'(\eta) = \frac{d\mu}{d\eta}$ 是逆链接函数的导数。

调整后预测值表示在指定协变量配置下评估的条件期望 $E[Y|X]$，提供特定条件下的拟合值。

3.1 2×2 框架

边际效应分析基于两个维度，这两个维度定义了研究人员可选择估计的不同量（表1）：

表1：边际效应分析的 2×2 框架：评估情境（行）与分析目标（列）的交叉组合。剖面分析包含两种变体：在样本均值处评估（MEM/APM），以及在任意参考剖面处评估（MER/APR）。

情境\ 目标	效应	预测
总体	平均边际效应（AME）	平均调整预测（AAP）
剖面（在均值处）	平均值处边际效应（MEM）	平均值处调整预测（APM）
剖面（在参考点处）	参考点处边际效应（MER）	参考点调整预测（APR）

评估情境　　决定推断的目标是总体层面参数还是特定剖面估计：

- 总体方法计算在观测协变量经验分布上的平均边际量，得到能够反映数据中异质性的具有总体代表性的估计。
- 剖面方法在由参考网格定义的特定协变量配置处评估边际量。 `means_grid()` 辅助函数在样本均值处进行评估（MEM/APM），代表一个“典型”观测。自定义网格（见第 4.3节）通过 `cartesian_grid()` 或用户提供的 DataFrame 在任意参考剖面处进行评估（MER/APR），代表研究者指定的反事实情景。两者都使用 `profile_margins()` 但解释不同。

分析目标规定所关注的量表示效应（导数／对比）还是预测（拟合值）。

对于不含交互项的线性模型，边际效应是常数（等于系数），因此剖面方法与总体方法会得出相同结果。对于含交互项的模型，AME 等于 MEM（协变量均值处的效应），因为边际效应对其他协变量仍呈线性，但在任意剖面处的效应会有所不同。对于非线性模型（Logit、Probit 等），无论模型设定如何，这一区别都具有实质性意义：

- AME 通过对观测到的协变量组合取平均，提供具有总体代表性的效应估计
- MEM 提供特定（可能不具代表性）取值点处的效应估计

研究者越来越倾向于使用 AME 而非 MEM，原因就在于此（Arel-Bundock、Greifer 和 Heiss，2024年；Hanmer 和 Kalkan，2013年），但 AME 的统计优势伴随着计算成本：进行 $O(n)$ 次边际效应计算，而 MEM 只需 $O(1)$ 次。现有软件的性能限制历来促使研究者将 MEM 作为一种优化策略，这也说明计算约束会如何影响方法选择。

3.2 方差估计

边际效应的标准误通过德尔塔法计算。对于标量边际效应 $m(\beta)$：

$$\text{Var}(m(\hat{\beta})) \approx g^T \Sigma g$$

其中， $g = \frac{\partial m}{\partial \beta}$ 是梯度， Σ是参数协方差矩阵。

对于平均边际效应，这需要累积各观测值的梯度：

$$g_{\text{AME}} = \frac{1}{n} \sum_i g_i$$

高效计算这些量——尤其是每个观测值的梯度——对于实现大规模统计推断至关重要，也促成了后续章节所述的计算架构。

4 Margins.jl：统计接口

Margins.jl 将 2×2 统计框架转化为简洁的双函数 API，使分析选择更加明确。本节介绍应用程序接口设计、主要功能和典型工作流程。

4.1 应用程序接口设计：双函数

Margins.jl 根据评估情境将分析分为两个入口点，这不同于其他软件包通常只提供一个带有许多选项的函数（例如 R 的 Margins 或 marginaleffects ）。

```
# Population analysis: average across observed sample distribution
population_margins(model, data; type=:effects)  # AME
population_margins(model, data; type=:predictions)  # AAP

# Profile analysis: evaluate at specific covariate configurations
profile_margins(model, data, reference_grid; type=:effects)  # MEM
profile_margins(model, data, reference_grid; type=:predictions)  # APM
```

对于剖面分析， `profile_margins()` 会计算由参考网格定义的特定协变量配置下的效应或预测值。当 `type=:effects`时，该函数会在每个网格点处计算边际效应（连续变量的导数，分类变量的对比）。与对所有观测到的协变量组合的效应取平均值的总体分析不同，剖面分析会在网格中指定的确切协变量值处进行计算：如果网格包含样本均值，则得到均值处的边际效应（MEM）；对于自定义网格，则得到任意参考点处边际效应（MER）。
对于分类变量，剖面分析会在每个网格点处计算对比（例如，“B”与“A”），而不是对整个样本的对比取平均值。二元变量被视为分类变量，其效应通过切换水平时预测结果的离散变化来计算。

这种区分有几个好处：

1. 清晰度：用户可以明确选择总体分析或剖面分析 2. 合适的默认设置：剖面分析需要参考网格；总体分析则不需要 3. 文档：每个函数只记录与其相关的参数 4. 性能：实现可以针对不同情况分别进行优化

4.2 总体分析

`population_margins()` 函数会根据观测样本分布计算平均值 :

```
using Margins, GLM, DataFrames

# Fit model
model = glm(@formula(y ~ x1 + x2 + group), data, Binomial())

# Average marginal effects for all continuous variables
ame_result = population_margins(model, data; type=:effects)
```

```
EffectsResult: 4 population effects (N=5000)
──────────────────────────────────────────────────────────────────────
Variable Contrast         dy/dx   Std. Err.   [95% Conf.   Interval]
──────────────────────────────────────────────────────────────────────
x1       dy/dx         -0.00737     0.00687    -0.02084      0.0061
x2       dy/dx          0.00145     0.00685    -0.01198     0.01487
group    B - A         -0.02436     0.01686    -0.05741     0.00869
group    C - A          0.01283     0.01683    -0.02015     0.04581
──────────────────────────────────────────────────────────────────────
```

```
# AME for specific variables only
ame_x1 = population_margins(model, data; type=:effects, vars=[:x1])

# AME at counterfactual scenario values
ame_scenario = population_margins(
    model, data; type=:effects, vars=[:x1], scenarios=(x2=1.0,)
)

# AME stratified by groups
ame_grouped = population_margins(model, data; type=:effects, vars=[:x1], groups=:region)
```

`scenarios` 参数通过覆盖特定变量值，在样本分布上计算效应，从而支持反事实分析。 `groups` 参数支持子组分析，针对分组变量的每个取值分别计算平均边际效应

估计值。加权分析可通过 `weights` 参数实现，该参数会将观测权重纳入边际效应计算和方差估计。对于调查数据，权重会调整平均边际效应，使其代表目标总体而非样本（Winship 和 Radbill 1994；Lumley 2004）。加权平均边际效应的计算公式为 $\partial = \sum_i w_i \cdot \partial_i / \sum_i w_i$，其中 w_i 为观测权重。

4.3 剖面分析与参考网格

The `profile_margins()` 函数在由参考网格定义的特定协变量配置处进行求值 d:

```
# Effects at sample means
mem_result = profile_margins(model, data, means_grid(data); type=:effects)
# Effects at custom grid pointsgrid = cartesian_grid(x1 = [0.0, 1.0, 2.0],
x2 = [-1.0, 0.0, 1.0])
effects_at_grid = profile_margins(model, data, grid; type=:effects)



# Predictions across a profile sweep
pred_grid = profile_margins(model, data, grid; type=:predictions)
```

Margins.jl 提供了几种参考网格构建器：

- `means_grid()`：连续变量的样本均值
- `cartesian_grid()`：指定值的笛卡尔积
- `balanced_grid()`：分类变量的平衡设计
- `quantile_grid()`：基于分位数的网格点

```
# Example: Create a grid over two variables
grid = cartesian_grid(
income = [25000, 50000, 75000, 100000],
education = ["HS", "BA", "MA"])
# Returns 12-row DataFrame: all combinations of income × education
```

还有其他专用构建器可用（例如，`hierarchical_grid()` 用于嵌套设计）。用户也可以直接提供任意 DataFrame 作为自定义参考网格。

4.4 分类变量

Margins.jl 会通过基于对比的计算自动处理分类变量。效应以对比的形式计算，支持 `:baseline`（与参考水平比较，默认）或 `:pairwise`（所有两两比较）这两种对比类型。预测值是在分类变量的每个水平上计算的。

```
# Model with categorical predictor
model = lm(@formula(y ~ x + group), data)

# AME automatically detects and handles categorical 'group'
result = population_margins(model, data; type=:effects)
```

```
EffectsResult: 3 population effects (N=5000)
------------------------------------------------------------------------
Variable Contrast         dy/dx   Std. Err.   [95% Conf.   Interval]
------------------------------------------------------------------------
x        dy/dx          0.39729     0.00711     0.38335     0.41122
group    B - A          0.37231     0.01751       0.338     0.40662
group    C - A          0.59665     0.01731     0.56272     0.63058
------------------------------------------------------------------------
```

输出显示连续变量 `x` 及其导数（`dy/dx`），以及分类变量 `group` 相对于基准水平 A 的对比。

4.5 分类混合

对于剖面分析，**Margins.jl** 支持分类混合——对分类水平进行加权组合，以避免任意选择“代表值”。默认情况下，分类变量使用观测比例作为混合权重。例如，如果样本包含 60% 的 A组和 40% 的 B组，默认混合会将效应计算为$0.6 \times \text{effect}(A) + 0.4 \times \text{effect}(B)$。自定义权重可覆盖观测比例，用于敏感性分析或匹配外部总体分布。

```
# Instead of picking a single group level...
# Use frequency-weighted mixture of observed proportions
grid = means_grid(data; group=:mixture)
mem_mixture = profile_margins(model, data, grid; type=:effects)
```

在内部，混合物以加权水平组合的形式表示：

```
struct CategoricalMixture{T}
    levels::Vector{T}
    weights::Vector{Float64}
end
```

这样一来，基于配置的 marginaleffects 就可以根据样本频数对各分类水平进行加权，从而比选择单一参考水平提供更具代表性的估计。

4.6 结果类型与输出

这两个函数都会返回结构化结果类型，用于组织输出，以便进行下游分析和展示：

```
struct EffectsResult <: MarginsResult
    estimates::Vector{Float64}
    standard_errors::Vector{Float64}
    variables::Vector{String}
    terms::Vector{String}
    profile_values::Union{Nothing, NamedTuple}
    group_values::Union{Nothing, NamedTuple}
    gradients::Matrix{Float64}
    metadata::Dict{Symbol, Any}
end

struct PredictionsResult <: MarginsResult
    estimates::Vector{Float64}
    standard_errors::Vector{Float64}
    profile_values::Union{Nothing, NamedTuple}
    group_values::Union{Nothing, NamedTuple}
    gradients::Matrix{Float64}
    metadata::Dict{Symbol, Any}
end

struct ContrastResult
    contrast::Float64
    se::Float64
    t_stat::Float64
    p_value::Float64
    ci_lower::Float64
    ci_upper::Float64
    estimate1::Float64
    estimate2::Float64
    gradient::Vector{Float64}
    row1::Union{Int, Nothing}
    row2::Union{Int, Nothing}
    metadata::Dict{Symbol, Any}
end
```

`EffectsResult` 包含边际效应（AME、MEM 或 MER）及 Delta 方法标准误，而`PredictionsResult` 包含调整后预测值（AAP、APM 或 APR）。两者都会存储梯度矩阵，以便后续进行对比计算； `profile_values`/`group_values` 则记录评估情境。比较两个估计值时（例如，在不同协变量值处的预测值，或不同组间的效应）， `contrast()` 函数会返回 `ContrastResult`，提供有关差异的完整统计推断。这些类型可以自然地转换为 DataFrame：

```
result = population_margins(model, data; type=:effects)

# Access individual components
result.estimates         # Point estimates
result.standard_errors   # Standard errors
result.variables         # Variable names

# Access as DataFrame
df = DataFrame(result)
```

```
4x10 DataFrame
 Row | type    variable  contrast  estimate     se          t_stat     p_value   ci_lower    ci_upper
     | String  String    String    Float64      Float64     Float64    Float64   Float64     Float64
-----+-----------------------------------------------------------------------------------------------
   1 | AME     x1        dy/dx     -0.00737132  0.0068739   -1.07236   0.283557  -0.0208439  0.00610128
   2 | AME     x2        dy/dx      0.00144605  0.00685147   0.211057  0.832843  -0.0119826  0.0148747
   3 | AME     group     B - A     -0.0243615   0.0168627   -1.4447    0.148542  -0.0574117  0.00868869
   4 | AME     group     C - A      0.0128309   0.0168256    0.762581  0.445713  -0.0201467  0.0458084
```

该 DataFrame 包含 `type`(平均边际效应/MEM)、 `variable`、 `contrast`、点估计、标准误、检验统计量、p值、置信区间和样本量等列。

该 `cont rast()` 函数用于计算估计值之间的差异，并采用适当的 delta 方法进行推断：

```
# Get profile effects with gradient information
result = profile_margins(model, data,
cartesian_grid(education=["HS", "College", "Grad"]);type=:effects,
 vars=[:x1])df = DataFrame(result; include_gradients=true)


# Contrast by row indices
cr = contrast(df, 2, 1, vcov(model))  # Row 2 vs Row 1

# Contrast by column specifications (NamedTuple)
cr = contrast(df,
(variable=:x1, education="College"),
(variable=:x1, education="HS"), vcov(model))
```

4.7 弹性

弹性以百分比形式而非绝对单位表示边际效应，便于比较尺度不同的变量。 Y 相对于 X 的弹性衡量的是 X 变化百分之一时 Y 的百分比变化： $\varepsilon = \frac{\partial Y}{\partial X} \cdot \frac{X}{Y}$ 。半弹性（`eyex` 和 `dyex`）则提供混合度量，其中只有一方以百分比形式表示（Stock 和 Watson（2011））。

Margins.jl 通过 `measure`参数支持弹性变体：

```
# Standard elasticity: (dy/dx) * (x/y)
population_margins(model, data; type=:effects, measure=:elasticity)

# Semi-elasticity dyex: (dy/dx) * x
population_margins(model, data; type=:effects, measure=:dyex)
```

```julia
# Semi-elasticity eydx: (dy/dx) / y
population_margins(model, data; type=:effects, measure=:eydx)
```

4.8 交互效应与二阶差分

当模型包含交互项时，研究者通常希望了解一个变量的边际效应是否取决于另一个变量的水平（Ai 和 Norton，2003年；Brambor、Clark 和 Golder，2006年）。**二阶差分**用于量化这种依赖关系：调节变量处于不同水平时，边际效应之间的差异。对于连续型调节变量，这一概念还可扩展为边际效应相对于调节变量的导数。

在线性模型中，二阶差分直接等于交互系数。然而，在非线性模型（Logit、Probit、泊松）中，交互系数定义在链接函数尺度上，并不代表预测结果尺度上的交互效应（McCabe 等，2022）。要正确解释，必须根据边际效应计算二阶差分：

$$\text{Second Difference} = \frac{\partial E[Y|X, Z_{\text{high}}]}{\partial X} - \frac{\partial E[Y|X, Z_{\text{low}}]}{\partial X}$$

Margins.jl 提供 `second_differences()` 函数系列，用于计算这些量，并采用适当的 delta 法标准误。可采用两种计算方法：（1）离散对比：比较预先计算的平均边际效应在不同调节变量水平上的差异；（2）局部导数：通过有限差分计算 ∂平均边际效应$/\partial Z$，默认在 Z 的样本均值处进行评估。

连续型 × 分类变量：连续变量的效应是否因组别而异?

```julia
# Effect of income on purchase probability by education level
model = glm(@formula(purchase ~ income * education + age), data, Binomial())

# Compute AMEs across education levels
ames = population_margins(model, data;
scenarios=(education=["hs", "college", "grad"],), type=:effects)


# Test whether income effect differs by education
sd = second_differences(ames, :income, :education, vcov(model))
```

连续型 × 连续型：一个连续变量的效应如何随另一个连续变量而变化 ?

```julia
# Effect of price on demand varies with household income
model = lm(@formula(quantity ~ price * income + controls), data)

# Local derivative: how does price effect change with income?
sd = second_differences_at(model, data, :price, :income, vcov(model);
    at=[30000, 50000, 100000])  # Evaluate at specific income levels
```

此计算使用对称有限差分，而非自动微分；有关步长选择和数值稳定性的详细信息，请参阅附录 A。

分类变量 × 分类变量：一个分类变量的效应是否取决于另一个分类变量?

```julia
# Employment effect on wellbeing varies by region
model = lm(@formula(wellbeing ~ employed * region + age), data)

# Compute AMEs across regionsames = population_margins(model, data;
scenarios=(region=["north", "south", "east", "west"],),
type=:effects)


# All pairwise comparisons of employment effect across regions
sd = second_differences_all_contrasts(ames, :employed, :region, vcov(model))
```

二阶差分的标准误通过 delta 方法计算，并适当考虑底层边际效应估计值之间的协方差：

$$\mathrm{SE}(\Delta) = \sqrt{(g_2 - g_1)^T \Sigma (g_2 - g_1)}$$

其中， g_1 和 g_2 分别是各边际效应相对于模型参数的梯度向量，而 Σ 是参数协方差矩阵。

4.9 与 Julia 生态系统的集成

Margins.jl 与 Julia 统计生态系统的集成始于公式层。StatsModels.jl（Kleinschmidt 等，2024）提供了两个软件包赖以构建的公式接口。**FormulaCompiler.jl**将 `FormulaTerm` 对象分解为原子操作，可处理所有标准项类型，包括连续变量、带有对比矩阵的分类项、函数应用（例如 `log()`、 `sqrt()`）以及多重交互项。Tables.jl（Quinn，2024）接口支持零分配求值架构：数据通过 `Tables.columntable()` 转换为列式 `NamedTuple` 格式，从而实现类型稳定的列访问，其中每次加载操作都会在编译时解析为直接的属性访问。

在模型拟合方面，**Margins.jl** 通过标准 StatsAPI 方法提取必要组件。对于 GLM.jl（Bates 等，2024）模型，系数通过 `coef()` 获取，参数协方差通过 `vcov()` 获取，连接函数通过 `Link(model)` 获取。支持所有标准 GLM 连接函数——恒等连接、对数连接、Logit、Probit、互补对数-对数链接、Cauchit 连接、逆连接和平方根连接——并且均实现了一阶和二阶导数，以便进行 Delta 方法方差计算。对于 MixedModels.jl（Bates、Alday 和 Kokandakar，2025），**FormulaCompiler.jl**通过剥离随机效应项来提取固定效应公式，从而能够为 `LinearMixedModel` 和 `GeneralizedLinearMixedModel` 对象计算边际效应。如未来工作部分所述，目前仅针对固定效应计算边际效应。

分类变量通过与 CategoricalArrays.jl 集成而得到特殊处理（Bouchet-Valat、Kami´nski 和 Quinn，2024年）。系统会通过其 `CategoricalArray` 类型自动检测变量，并直接从 StatsModels.jl 的对比编码规范中获取对比矩阵，支持所有标准方案，包括虚拟变量编码、效应编码和 Helmert 编码。对于基于剖面的分析，**FormulaCompiler.jl**提供`CategoricalMixture` 类型，用于表示分类水平的加权组合，从而能够得到代表性估计，避免任意选择参考水平。布尔变量也以类似方式处理，基于混合的评估中会根据样本频数计算 `true` 的概率。

数据管道利用 Julia 的接口抽象来提升灵活性。输入数据可以采用任何兼容 Tables.jl 的格式——DataFrames.jl（Bouchet-Valat 和 Kami´nski，2023年）、 `NamedTuple`或自定义表类型——并会自动转换为高效评估所需的列式格式。输出也遵循相同模式：结果类型实现 Tables.jl 接口（`Tables.istable`、 `Tables.rows`、 `Tables.schema`），因此可直接转换为 DataFrames，并可配置输出格式。这种双向的 Tables.jl 集成使 **Margins.jl** 的结果能够自然地流入下游分析管道。
一些额外的软件包扩展了核心功能。StandardizedPredictors.jl（Kleinschmidt 和 Alday，2024年）支持自动反变换：使用 Z 分数化预测变量拟合模型时，**FormulaCompiler.jl** 会识别`ZScoredTerm` 对象，并以原始变量尺度报告边际效应。未来版本计划支持其他标准化类型（例如， `CenteredTerm`）。CovarianceMatrices.jl（Ragusa，2024年）通过 `vcov` 参数提供异方差稳健（HC0–HC3）和聚类稳健（CR0/CR1）标准误（参见“稳健标准误”一节）。导数计算所依赖的自动微分使用 ForwardDiff.jl（Revels、Lubin 和 Papamarkou，2016年）；对偶数在构造时预先转换，以消除每次访问时的类型转换开销——这是实现零分配求值的关键因素。

对于模型 fi 型，边际效应会自动以原始变量尺度报告 e:

```
model = lm(@formula(y ~ x + z), data, contrasts=Dict(:x => ZScore()))
population_margins(model, data)  # Effects per unit of raw x,  not per SD
```

4.10 稳健标准误

Margins.jl 通过与 Covariance- Matrices.jl 集成，支持异方差稳健标准误和簇稳健标准误（Zeileis，2004年、2006年）。 `vcov` 参数接受任何协方差估计量，因此无需修改边际效应计算即可进行稳健推断。

异方差稳健标准误:

```
using CovarianceMatrices

# HC1 (most common)
population_margins(model, data; vcov=HC1())

# HC3 (more conservative)
population_margins(model, data; vcov=HC3())
```

簇稳健标准误：

```
# Single-level clustering by firm
population_margins(model, data; vcov=CR0(data.firm_id))
```

该实现使用德尔塔法，将稳健协方差估计通过边际效应变换进行传递。对于边际效应 $\theta = g(\beta)$ 及其梯度 $\nabla_\beta g$，稳健标准误为：

$$\text{SE}_{\text{robust}}(\hat{\theta}) = \sqrt{\nabla_\beta g(\hat{\beta})^T \, \hat{\Sigma}_{\text{robust}} \, \nabla_\beta g(\hat{\beta})}$$

ˆ其中，Σ_{robust} 是夹心估计量（HC0-HC3）或簇稳健估计量（CR0/CR1）。这种方法确保边际效应的稳健推断继承与底层模型系数的稳健推断相同的理论性质。

5 FormulaCompiler.jl：计算基础

Margins.jl 的性能特征源自**FormulaCompiler.jl**；后者通过位置映射编译策略，将统计公式转换为零分配求值器。“零分配”指逐行求值：构造过程会为缓冲区产生一次性 $O(p)$ 分配，但后续求值不会产生随数据集大小 n 扩展的额外分配——这是避免现有实现所受内存限制的关键。本节介绍支持高效边际效应计算的关键架构元素。

5.1 位置映射编译

传统公式系统会在进行任何计算之前构建完整的 $O(n \times p)$ 设计矩阵。对于边际效应，这会形成瓶颈：计算平均边际效应需要在每个观测值上求值公式，而 Delta 方法标准误则需要在所有 n 行上累积梯度。在大规模情况下，物化完整矩阵或$n \times p$雅可比矩阵可能会难以承受。

FormulaCompiler.jl 可将兼容 StatsModels.jl 的公式编译为完全类型化的求值器结构体，其中每个操作的输出位置都在编译时确定。编译过程首先将公式的每个元素——变量引用、变换和交互项——分解为原子操作。随后为每个项 j 分配一个固定的输出位置 $\text{Position}_j \in \{1, \ldots, p\}$，从而建立公式结构到内存布局的静态映射。

FormulaCompiler.jl 直接基于 StatsModels.jl 的 Wilkinson 表示法公式系统构建。该软件包从拟合模型的模式（其中存储了公式和对比编码规范）中提取 `FormulaTerm`，并遍历项树，包括 `InteractionTerm`、`FunctionTerm`、`CategoricalTerm`及其他项类型，以生成操作序列。分类变量的对比矩阵取自 StatsModels.jl 的对比编码规范，从而确保与原始模型拟合保持一致。采用这种复用策略，意味着任何 GLM.jl 或 MixedModels.jl 接受的公式，都能与 **FormulaCompiler.jl** 一样正常工作。

编译后的公式表示为一个完全类型化的结构体，其中所有结构信息都编码在类型参数中：

```
struct UnifiedCompiled{T, OpsTuple, ScratchSize, OutputSize}
    ops::OpsTuple
    scratch::Vector{T}
end
```

类型参数编码了完整的求值策略：`T` 是元素类型（通常为 `Float64`，或用于自动微分的 `Dual`），`OpsTuple` 包含类型化操作序列，而 `ScratchSize` 和 `OutputSize`是指定缓冲区维度的编译时常量。

Op操作被编码为 `AbstractOp` 的子类型，位置则嵌入在类型参数中 :

```
struct LoadOp{Column, OutPos} <: AbstractOp end
struct BinaryOp{Func, InPos1, InPos2, OutPos} <: AbstractOp end
struct ContrastOp{Column, OutPositions}
    contrast_matrix::Matrix{Float64}
end
```

例如， `LoadOp{:x, 3}` 将列 `:x` 加载到暂存位置 3，并将位置 1 和 2 相乘，将结果存储在位置 4。这种编码将所有索引决策嵌入类型参数中，使 Julia 编译器能够针对每种独特的公式结构特化求值代码。运行时，求值器只需执行固定的操作序列，无需分支或动态查找。

对于较长的操作序列，Julia 的元组特化启发式可能会失效，导致性能下降。 **Formu-laCompiler.jl** 采用混合分派策略：对于包含 ≤ 10 个操作的公式，使用递归；而对于较大的公式，则使用`@generated` 函数。[2] `@generated` 指的是 Julia 元编程在编译时代码生成方面的能力（Bezanson 等，2017）：它会在编译时检查操作类型，生成特化代码，将整个操作序列展开为单个函数体，不含循环或递归。无论公式复杂度如何，这都能确保零内存分配——包含 100+个项的公式，其逐行性能与简单公式相同。
预先分配一个固定大小的暂存向量，并在多次求值之间重复使用，同时通过 `@inline` 注解和边界检查消除确保每次操作的开销为零。一次编译、多次求值模式非常适合边际效应计算，因为必须对每个观测值求值同一个公式。图2 展示了从公式输入到已编译求值器的这一流水线。

```
# Example: formula to compiled evaluator
model = lm(@formula(y ~ x1 + x2 * group), data)
compiled = compile_formula(model, data)  # Stage 4 output
output = Vector{Float64}(undef, width(compiled))
compiled(output, data, 1)  # Zero-allocation row evaluation
```

5.2 零分配求值

给定已编译的求值器和列式数据接口（Tables.jl；Quinn（2024年）），行求值可原地进行。公式只编译一次，随后即可反复调用生成的求值器，且不会产生堆内存分配：

```
compiled = compile_formula(model, data)
row = Vector{Float64}(undef, length(compiled))

for i in 1:nrow(data)
    compiled(row, data, i)
end
```

预热后，每次迭代分配的内存为零字节。核心求值仅通过预分配缓冲区需要 $O(p)$内存，预热后不会产生堆内存分配；情景分析则仅通过反事实向量类型增加 $O(1)$的开销，这些类型向求值器呈现修改后的值，同时底层数据保持不变。

这与将 $O(n \times p)$ 设计矩阵具体化的传统实现有根本区别：**Formula-Compiler.jl** 从不构造完整矩阵，而是按需对每一行求值。图3展示了行求值期间的操作序列，并突出了类型稳定分派和直接内存写入。

```
# Zero-allocation evaluation loop
compiled = compile_formula(model, data)
row = Vector{Float64}(undef, width(compiled))

for i in 1:nrow(data)compiled(row, data, i)  # Updates row in-place,

 zero allocationsend
```

[2] `@generated` 函数是 Julia 的一项特性：编译器会根据参数的类型，在编译时计算函数体本身。这样，编译器便能针对每种独特的公式结构生成专门化且展开的代码。

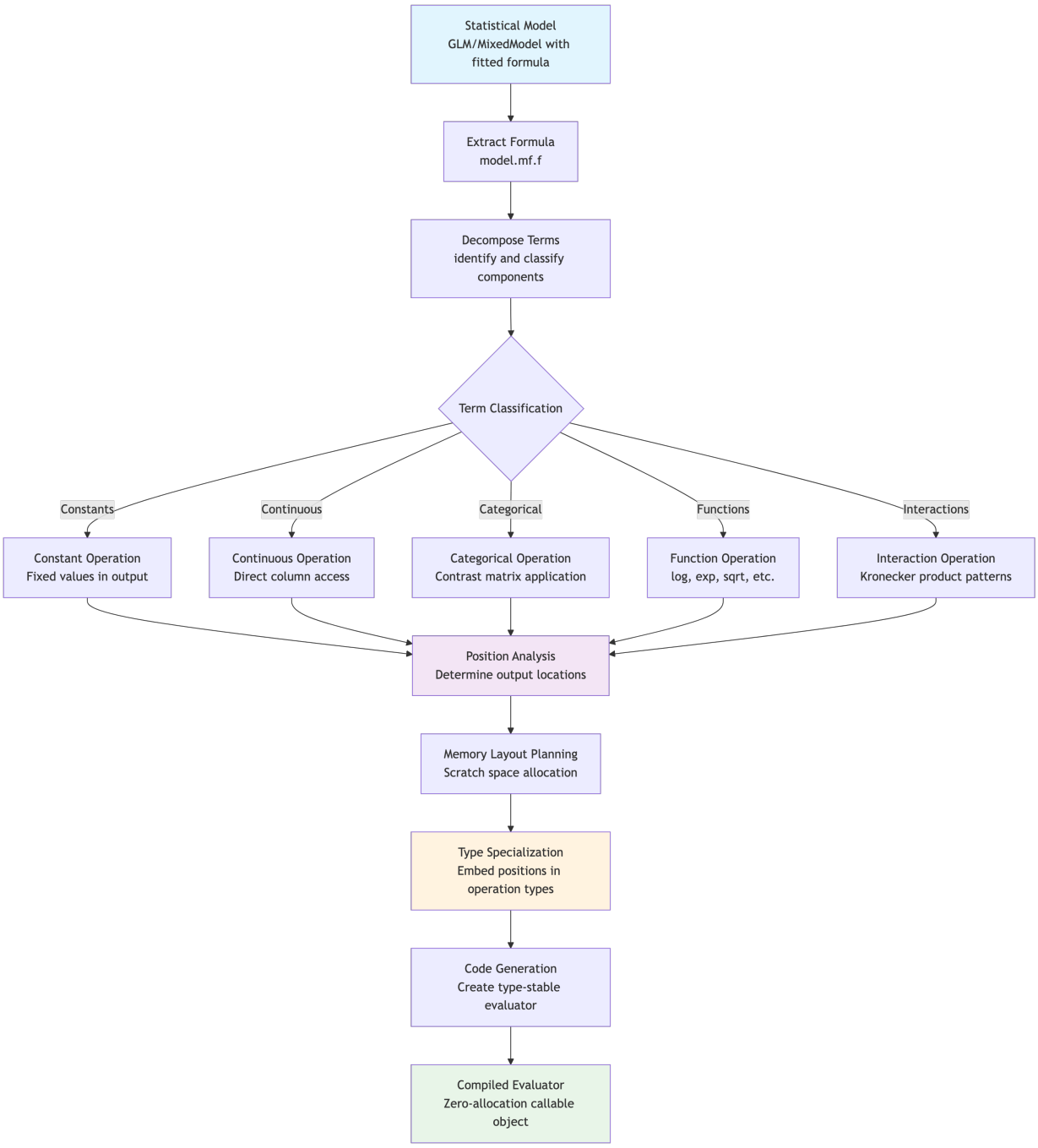


图 2：编译流水线：FormulaCompiler.jl 将 StatsModels.jl 公式经过四个阶段进行转换——(1) 将项分解为原子操作，(2) 将位置映射到固定的输出索引，(3) 将操作编码为 Julia 的类型参数，以及 (4) 通过缓冲区预分配生成代码——最终生成零分配求值器。

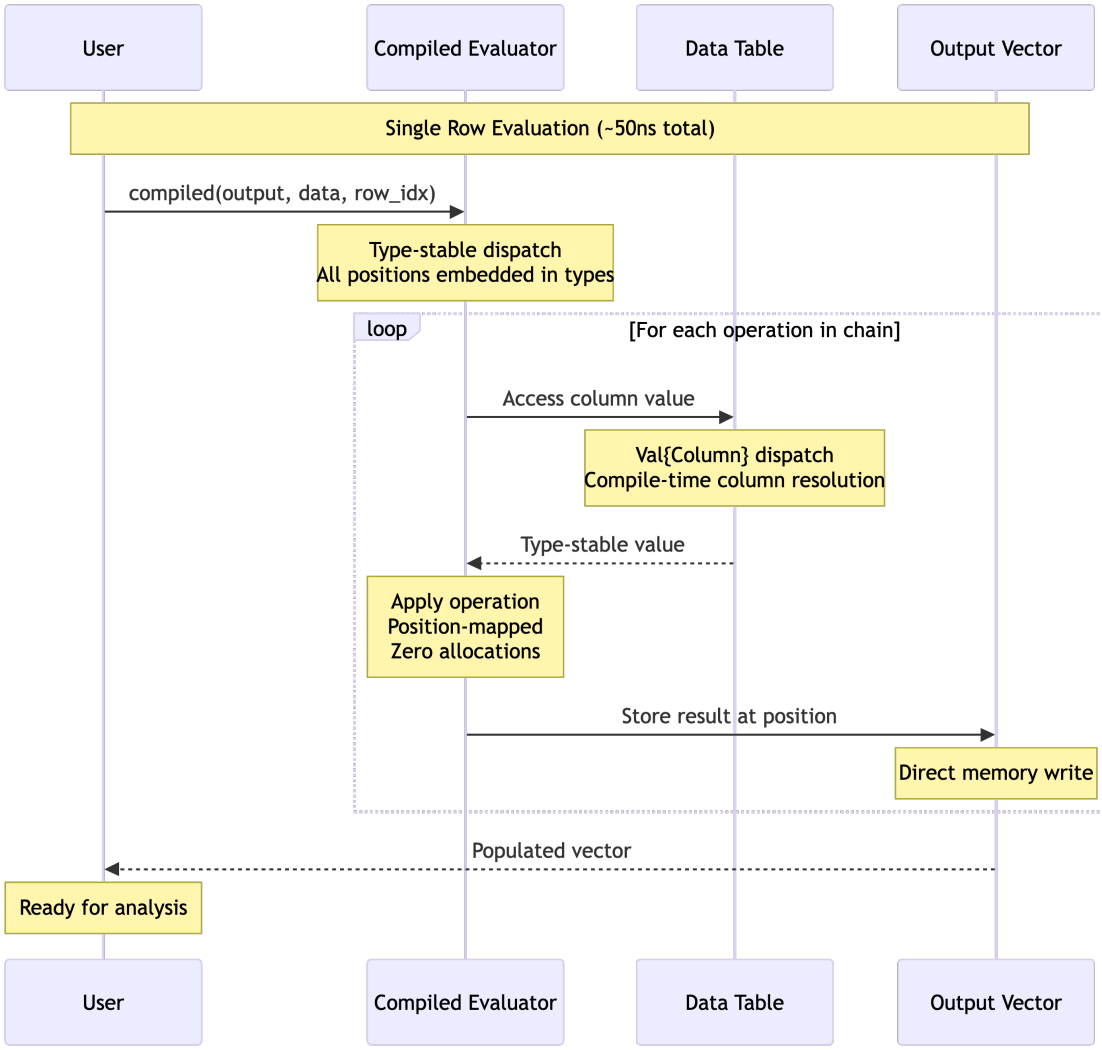


图 3：零分配行求值时序图：类型稳定分派在编译时解析列访问，从而实现直接内存写入，无需中间分配。

5.3 自动微分

边际效应需要计算模型行相对于输入变量的导数。**FormulaCompiler.jl** 通过一种手动编排的自动微分（AD）系统实现这一功能，该系统使用 ForwardDiff.jl（Revels、Lubin 和 Papamarkou，2016）的对偶数；对偶数同时携带数值及其导数，并通过链式法则实现精确的导数计算。

将自动微分应用于统计公式时，一个实际挑战是类型转换开销。标准 AD 方法会在每次访问时将 `Float64` data 转换为对偶数，因此产生的内存分配会随公式复杂度增加。**FormulaCompiler.jl** 采用预转换策略，通过两个阶段消除这一开销：

1. **构造时预转换**：在引擎构造期间，数据列会从`Float64` 转换为 `Dual{T,Float64,k}`，供 k 微分变量使用。这项成本仅发生一次，并在任何逐行计算之前完成，因此能够摊销。2. **同类型求值**：在导数计算期间，求值链始终保持对偶类型，不会发生运行时转换。变量以恒等偏导数进行播种，编译后的公式对对偶类型数据执行求值，并直接从结果中提取偏导数。

不同于使用 ForwardDiff.jl 的通用 `jacobian!` 函数（它为了灵活性而带来额外的分配开销），**FormulaCompiler.jl** 采用直接播种、原位缓冲区更新、单遍求值和基于循环的提取。这样既保证了 ForwardDiff 的数学正确性，又通过定制编排实现了零分配。

结果是：传统自动微分会产生 O（访问 ×转换）运行时分配，而此方法无论公式复杂度如何，都能实现零运行时分配。系统会针对所关注的变量构造导数求值器，并原地计算边际效应：

```
de = build_derivative_evaluator(compiled, data; vars=[:x1, :x2])

g = Vector{Float64}(undef, 2)
marginal_effects_mu!(g, de, coef(model), row_idx; link=LogitLink())
```

导数计算使用专用求值器，并在整个过程中保持类型稳定性。 `ADEvaluator` 在构造时预先将数据列转换为对偶数，从而实现零分配的雅可比矩阵计算。 `FDEvaluator`提供有限差分导数，作为自动微分的替代方案。两种求值器类型都持有根据公式输出宽度预先分配的缓冲区，确保除基础公式求值之外，导数计算不会产生任何逐行分配。

FormulaCompiler.jl 支持所有标准链接函数的边际效应计算。需要使用逆链接函数的导数（$d\mu/d\eta$），通过链式法则将线性预测值上的效应转换到响应尺度。支持的链接函数包括恒等连接、对数连接、Logit、Probit、互补对数-对数链接（cloglog）、Cauchit 连接、逆连接、平方根链接和逆平方链接。对于二元结局，Logit 和 Probit 链接最为常见；对于计数数据（泊松分布、负二项分布），通常使用对数连接；线性模型则使用恒等连接。当效应在响应尺度上计算时，系统会自动计算二阶导数，以获得 Delta 方法标准误。

5.4 启用 Margins.jl

FormulaCompiler.jl 提供了一小组基础功能，供**Margins.jl**组合使用，以实现完整的统计接口： `compile_formula()` 用于一次性公式编译， `continuous_variables()` 用于自动检测变量类型， `marginal_effects_eta!()` 和 `marginal_effects_mu!()` 分别用于在线性预测值和响应尺度上逐行计算效应， `accumulate_ame_gradient!()` 用于高效累积各观测值的梯度， `delta_method_se()` 用于根据梯度和参数协方差矩阵高效计算标准误。

Margins.jl 使用标记类型在编译时专门化计算：

```
abstract type MarginsUsage end
struct PopulationUsage <: MarginsUsage end
struct ProfileUsage <: MarginsUsage end

abstract type DerivativeSupport end
struct HasDerivatives <: DerivativeSupport
    evaluator::AbstractDerivativeEvaluator
end
struct NoDerivatives <: DerivativeSupport end
```

这些标记类型对应两个应用程序接口入口： `PopulationUsage` 用于 `population_margins()` ， `ProfileUsage`用于 `profile_margins()`。它们会传递到主计算引擎：

```
struct MarginsEngine{L<:GLM.Link, U<:MarginsUsage, D<:DerivativeSupport, C}
    compiled::C
    link::L
    # ... pre-allocated buffers
end
```

类型参数使 Julia 的多重分派能够选择经过优化的代码路径： `PopulationUsage` 会触发内存占用极低的逐行流式算法，而 `ProfileUsage` 则针对评估多个协变量剖面进行了优化。 `compiled` 字段包含一个来自 **FormulaCompiler.jl** 的 `UnifiedCompiled` 求值器，在统计接口与计算基础之间搭起桥梁。这种设计使 **Margins.jl** 能够通过高效遍历观测值来计算总体平均量，同时避免现有实现所受的内存开销限制。

5.5 情景系统

对于反事实分析，**FormulaCompiler.jl** 提供了一种情景系统，无需复制数据即可覆盖变量值，从而将每个情景的内存开销降至 $O(1)$。这一功能类似于 Stata 的 `margins` 命令及其 `at()` 选项（Williams 2012）。例如，要计算在所有观测值中 `x1 = 1.0` 这一反事实情景下的效应：

```
scenario_data = create_scenario(data; x1=1.0)

compiled(row, scenario_data, i)
```

求值过程与原始数据完全相同，仅有 $O(1)$ 的开销。反事实向量类型会向求值器提供修改后的值，同时保持底层数据不变，从而以极低的内存开销进行反事实求值。用于计算导数的单行扰动采用类型特化向量：

```
# Numeric override
NumericCounterfactualVector(data.x, 1.0)  # x = 1.0 for all

# Categorical override
CategoricalCounterfactualVector(data.group, "B")  # group = "B" for all

# Boolean override
BoolCounterfactualVector(data.treated, true)  # treated = true for all
```

这些向量会提供修改后的值，并在其他情况下委托使用基础数据，从而无需复制底层数据集即可在反事实值上计算导数。

6 性能基准测试

本节介绍性能比较，以展示该架构方法在实际应用中的效果。我们首先将 **Margins.jl** 与 R 中先进的 marginaleffects 包进行比较，随后介绍微基准测试，以验证**FormulaCompiler.jl**的零分配保证。表 6 和表 7 提供了基准测试公式和数据集大小的完整规格说明。

6.1 Margins.jl 与 R 的 marginaleffects

我们对 **Margins.jl** 与 R 的 marginaleffects 包（Arel-Bundock、Greifer 和 Heiss，2024）进行了基准测试，使用的是一个包含 65 个参数的真实逻辑回归模型，其中包括连续型变量和分类变量之间的交互项。两个包使用等效的统计方法计算出完全相同的量。表 2展示了标准数据集的结果，表3 展示了大规模情况下的性能。

以下代码展示了各个包如何计算平均边际效应：

Julia

```
using Margins, GLM, DataFrames

model = glm(@formula(y ~ x1 * x2 + x3 * cat1), data, Binomial())
result = population_margins(model; type=:effects)  # AME for all variables
```

R

```
library(marginaleffects)

model <- glm(y ~ x1 * x2 + x3 * cat1, data=data, family=binomial())
result <- avg_slopes(model)  # AME for all variables
```

标准数据集（5,000 个观测值）：

表2：性　　能比较：标准数据集（5,000 个观测值，65 个参数的逻辑回归）。

操作	Margins.jl（毫秒）	marginaleffects（毫秒）	加速比	Julia（MB）	R（MB）	内存比率
APM（调整后预测值）	0.70	76	109x	0.93	12.8	14x
MEM（边际效应剖面）	2.1	6413	3124x	1.9	719	383x
AAP（平均值调整后预测）	0.59	154	261x	0.46	495	1072x
AME（所有变量）	57	2974	52x	74.9	14640	196x
AME（单一变量）	1.7	299	173x	0.46	505	1092x
AME（情景）	21	294	14x	10.8	34.3	3x
平均值	—	—	**622x**	—	—	**460x**

大型数据集（500,000 个观测值）：

表3：大型数据集（500,000 个观测值）上的性能比较。R 的 marginaleffects 因内存耗尽而无法计算平均边际效应。

操作	Margins.jl	marginaleffects
AAP	66 毫秒，27 MB	13310 毫秒，48585 MB
AME（所有变量）	8.9 秒，14.2 GB	失败（“已达到 48.0 Gb 的向量内存限制”）

50 万条观测值的结果体现了根本性的扩展能力差异：**Margins.jl**成功完成了所有 65 个变量的平均边际效应计算，而 marginaleffects 则在完成计算前因内存错误而终止。 `marginaleffects` 在失败前完成的操作显示，内存比率超过 1,791 倍；平均调整后预测所需内存为 47.4 GB，而另一方仅为 27 MB（硬件详情见附录 C）。

6.2 FormulaCompiler.jl 微基准测试

上述性能特征源于 **FormulaCompiler.jl** 的零分配求值。我们通过微基准测试验证这些保证，测量逐行求值时间和内存分配（公式规范见表 6）。表 4展示了跨数据集大小的常数时间求值，而表 5 展示了随公式复杂度变化的扩展情况。

逐行求值（交互项密集公式）：

表 4：跨数据集大小的逐行求值时间，展示了在 n 中具有零内存分配的 $O(1)$ 复杂度。

n（行）	中位数（纳秒）	最小值（纳秒）	内存（字节）
10,000	13.6	12.1	0
100,000	13.6	12.1	0
1,000,000	13.7	12.1	0

随着数据集大小增加两个数量级，逐行求值时间仍保持不变，表明其复杂度为 $O(1)$ n。预热后，内存分配量恰好为零字节。

公式复杂度：

表 5： 按公式复杂度划分的逐行评估时间，显示 $O(p)$ 的扩展性及零内存分配 .

公式	中位数（纳秒）	内存（字节）
简单（3 个项）	6.8	0
复杂（交互项）	13.6	0

评估时间随公式复杂度（输出宽度 p）而变化，同时所有公式类型均保持零内存分配。图 4 展示了随着数据集大小增加两个数量级，这种恒定时间行为。

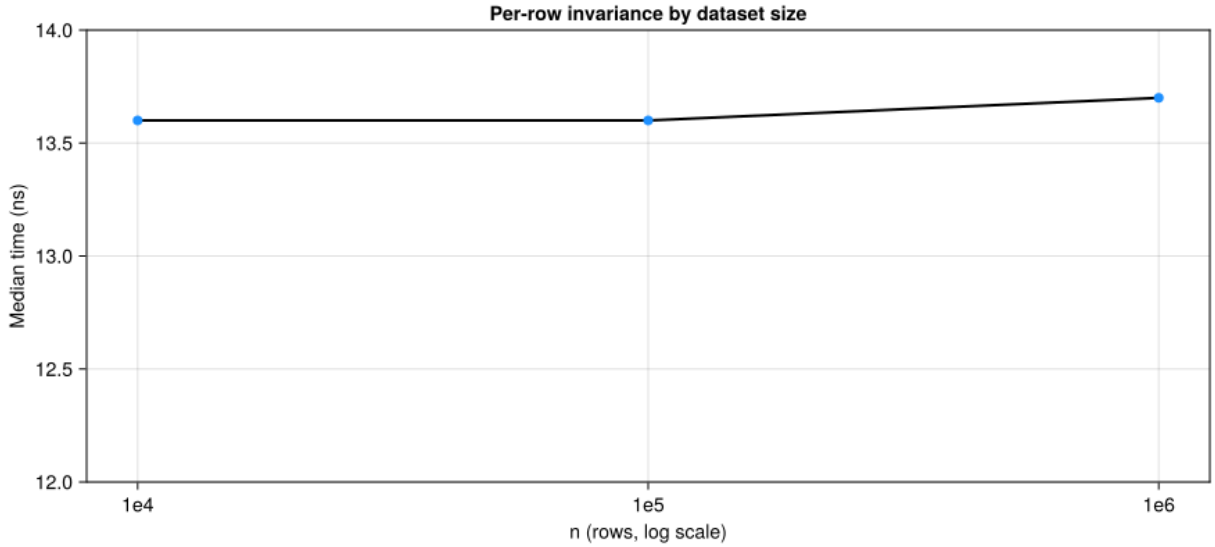


图4：当观测值从 10,000 增加到 1,000,000 时，每行评估时间保持不变，表明其关于数据集大小的复杂度为 $O(1)$，其中 n。

7 案例研究

我们通过四个案例研究展示典型的**Margins.jl** 工作流程：基于情景的反事实分析、针对分类变量和连续调节变量的交互作用检验，以及混合模型集成。

7.1 案例研究1：政策反事实

本案例展示了反事实情景分析——计算在假设的政策干预下，效应会如何变化。

模型：包含处理变量与协变量交互作用的线性模型：

```
using Margins, GLM, DataFrames

model = lm(@formula(y ~ x + z + x * treatment), data)

# Baseline: average marginal effect of x
baseline = population_margins(model, data; type=:effects, vars=[:x])

# Policy scenario: AME of x when treatment = 1 for everyone
policy = population_margins(model, data; type=:effects, vars=[:x], scenarios=(treatment=1,))
```

观测条件下的基准平均边际效应 `x`：

```
EffectsResult: 1 population effects (N=5000)
------------------------------------------------------------------------
Variable Contrast         dy/dx   Std. Err.   [95% Conf.   Interval]
------------------------------------------------------------------------
x        dy/dx          0.36917       0.007     0.35544      0.3829
------------------------------------------------------------------------
```

在所有观测值均接受处理的反事实政策情景下：

```
EffectsResult: 1 population effects (N=5000)
Scenarios: treatment
------------------------------------------------------------------------------------
Variable Contrast    at_treatment        dy/dx   Std. Err.   [95% Conf.   Interval]
------------------------------------------------------------------------------------
x        derivative  1                 0.43936     0.00999     0.41979     0.45894
------------------------------------------------------------------------------------
```

在实际观测条件下， x 的平均边际效应为 0.369。在所有人都接受处理的反事实政策情景下，平均边际效应升至 0.439。 scenarios 参数通过在所有观测值中固定特定协变量值，同时根据样本分布计算效应，从而支持反事实分析。这与 groups参数不同，后者会基于观测数据，针对分组变量的每个水平分别计算平均边际效应。对于“假设……会怎样”的问题，请使用scenarios （例如，“如果所有人都接受处理，效应会怎样？”）；对于子组分析，请使用 groups （例如，“效应因地区而有何不同？”）。

7.2 案例研究 2：含分类变量交互项的广义线性模型

本案例展示了如何计算含连续变量与分类变量交互作用的逻辑回归模型的边际效应，并通过二阶差分进行正式检验。

模型：二元结果，连续预测变量 x 与分类变量 group存在交互作用：

```
model = glm(@formula(y ~ x * group + z), data, Binomial())

# 1. Main effects (AME across all variables)
ame_result = population_margins(model, data; type=:effects)

# 2. Effect of x at each group level
ames_by_group = population_margins(model, data; type=:effects, vars=[:x],
 scenarios=(group=["A", "B", "C"],))

# 3. Test whether x effect differs by group (second differences)
sd = second_differences(ames_by_group, :x, :group, vcov(model))
```

主效应（所有变量的平均边际效应）：

```
EffectsResult: 4 population effects (N=5000)
------------------------------------------------------------------------
Variable Contrast         dy/dx   Std. Err.   [95% Conf.   Interval]
------------------------------------------------------------------------
x        dy/dx          0.01457     0.00695     0.00096     0.02819
z        dy/dx          0.01013     0.00694    -0.00346     0.02372
group    B - A          0.02114     0.01702    -0.01222     0.05451
group    C - A          0.07478     0.01697     0.04152     0.10805
------------------------------------------------------------------------
```

x 的效应因组别水平而异：

```
EffectsResult: 3 population effects (N=5000)
Scenarios: group
------------------------------------------------------------------------------------
Variable Contrast    at_group        dy/dx   Std. Err.   [95% Conf.   Interval]
```

```
------------------------------------------------------------------------------
x        derivative  A             0.02404     0.01203     0.00045     0.04762
x        derivative  B             0.03503     0.01227     0.01098     0.05908
x        derivative  C            -0.01419      0.0118    -0.03732     0.00894
------------------------------------------------------------------------------
```

二阶差分用于检验这些组特定效应是否存在显著差异：

```
 Levels    Delta(dy/dx)     SE       z       p-value
 A vs B    0.0110    0.0172   0.64     0.522
 A vs C    -0.0382    0.0168   -2.27     0.023
 B vs C    -0.0492    0.0170   -2.89     0.004
```

x 的边际效应随组水平而变化：在 A 组和 B 组为正，在 C 组为负。二阶差分检验这些差异是否具有统计显著性。在此，A 与 C、B 与 C 的对比均达到 5% 显著性水平（$p < 0.05$），表明在概率尺度上，x 的效应在 C 组与其他组之间存在显著差异。

7.3 案例研究 3：MixedModelsFixedEffects

本案例展示混合效应模型的边际效应，重点关注固定效应，同时考虑随机效应结构。该软件包与 MixedModels.jl 集成，并从模型中提取固定效应以计算边际效应。

模型：含随机截距的线性混合模型：

```
using MixedModels

model = fit(MixedModel, @formula(y ~ x + treatment + (1|group)), data)

# Fixed effects marginal effects
fe_result = population_margins(model, data; type=:effects)
DataFrame(fe_result)
```

```
EffectsResult: 2 population effects (N=10000)
------------------------------------------------------------------------------
Variable   Contrast            dy/dx   Std. Err.   [95% Conf.   Interval]
------------------------------------------------------------------------------
x          dy/dx             0.30348      0.0051     0.29348     0.31348
treatment  true - false      0.49767     0.01011     0.47785     0.51749
------------------------------------------------------------------------------
```

请注意，**Margins.jl** 仅计算固定效应的边际效应，而忽略随机效应部分。

7.4 案例研究 4：连续型 ×连续交互作用

本案例演示如何使用 `second differences at` 来检验一个连续变量 $\bar{}$ $\bar{}$ 的效应是否会随另一个连续变量而变化。我们展示两种情形：线性交互（变化率恒定）和二次交互（变化率不断变化）。

线性交互：

```
model = glm(@formula(y ~ x * z), data, Binomial())

sd = second_differences_at(model, data, :x, :z, vcov(model);
    at=quantile(data.z, [0.25, 0.5, 0.75]))
```

```
 eval_point (z)   d(AME_x)/dz      SE        z       p-value
 0.41 (25th %ile)   0.0162      0.0111    1.46      0.144
 0.77 (50th %ile)   0.0155      0.0113    1.37      0.169
 1.26 (75th %ile)   0.0142      0.0113    1.26      0.209
```

在线性交互的情况下，导数 ∂（平均边际效应）$/\partial z$ 在所有评估点上近似保持不变——无论沿着 `z` 在何处进行评估， `x` 的影响变化速率都相似。

二次交互：

```
model = glm(@formula(y ~ x * z + x * z2), data, Binomial())

sd = second_differences_at(model, data, :x, :z, vcov(model);
    at=[0.5, 1.5, 2.5])
```

```
 eval_point (z)   d(AME_x)/dz      SE        z       p-value
 0.5                0.0146      0.0365    0.40      0.689
 1.5                0.0141      0.0349    0.40      0.687
 2.5                0.0133      0.0326    0.41      0.682
```

对于二次交互，导数会随评估点而变化： `x`的效应变化速率本身取决于 `z`的取值。 `second_differences_at` 函数通过在调节变量取值范围内的多个点计算局部导数，来检测这种曲率。

8 未来工作

本研究还有若干可拓展的方法方向。首先，虽然**Margins.jl** 目前仅计算固定效应的边际效应，但更全面的处理方式应将随机效应不确定性纳入边际效应本身。这需要对随机效应分布进行积分：对于简单设计，可采用解析方法；对于更复杂的层级结构，则可通过模拟实现（Skrondal 和 Rabe-Hesketh 2009）。这将使我们能够在聚类和纵向设计中对边际效应进行推断，因为在这些设计中，随机效应反映了实质性的异质性。

此外，该软件包还可扩展对更多模型类别的支持。生存模型需要在生存概率尺度而非风险比尺度上计算边际效应，这涉及对删失分布进行积分。有序回归模型会产生类别特定边际效应——即每个结果类别的概率变化——在类别具有自然顺序时，需要谨慎解释。多项选择模型涉及跨选项效应，即改变一个协变量会影响所有选项的概率，并对无关选项独立性假设产生影响。

虽然 **Margins.jl** 支持双向交互的二阶差分，但也可以扩展该包，使其直接支持高阶交互分析。这与处理效应异质性的联系可能很有意义：边际效应随协变量分布而变化时，可以识别干预效果不同的子群体，从而与因果推断文献中的条件平均处理效应研究相衔接。最后，还可以扩展该包以支持并行计算，从而处理更复杂的模型和更大型的数据集，尤其适用于基于模拟的随机效应积分，以及针对大型数据集的异质性分析。

9 验证

这两个包都经过全面的自动化测试，以确保数值正确性和性能保证。**Margins.jl** 使用跨语言导出的相同数据集，将边际效应计算结果与 R 的 marginaleffects 包进行验证；测试容差要求点估计的差异在 0.01% 以内，标准误的差异在 0.1% 以内。测试套件涵盖 2×2 框架、分类变量对比、交互效应和弹性度量的所有组合。**FormulaCompiler.jl** 将编译后的公式输出与 StatsModels.jl 的 `modelmatrix()`参考实现进行验证，并通过 BenchmarkTools 的测量结果（Chen 和 Revels，2024年）验证零分配保证。这两个包都采用持续集成，在每次提交时自动运行测试。

10 结论

本文介绍了 **Margins.jl** 和 **FormulaCompiler.jl**，这两个用于边际效应计算的 Julia 软件包。**Margins.jl** 提供了一个统计接口，围绕 2×2 框架组织，区分总体与剖面分析，以及效应与预测。**FormulaCompiler.jl**通过基于位置映射的公式编译提供计算基础，实现零分配求值。

统计接口与计算引擎在架构上的分离，使每一层都能独立优化。用户可以使用熟悉的统计概念，而底层实现则通过类型专用编译和预分配缓冲区来处理性能方面的考量。

基准测试显示，相较于现有实现，性能有了显著提升：与 R 的 marginaleffects 包相比，平均加速比达 622 倍，内存占用减少 460 倍；此外，在对比实现因可用内存不足而无法处理的数据集大小下，该实现仍能成功完成计算。这些特性使得在更大规模下计算平均边际效应成为可能，而在这种情况下，原本可能不得不退而求其次，改为计算均值处的边际效应或省略标准误。

FormulaCompiler.jl 中的计算技术（类型专用编译、预分配缓冲区、协调式自动微分）不仅适用于边际效应，也适用于其他需要反复计算公式的工作负载。蒙特卡罗模拟、自助法推断、交叉验证、敏感性分析和功效分析都具有这种计算模式：每次计算的开销会在大量迭代中累积。

11 软件可用性

这两个软件包均已收录在 Julia 通用注册表中，可通过 `Pkg.add()`安装。此外，也可以直接从 GitHub 安装开发版本。

Margins.jl:

- 代码库：https://github.com/emfeltham/Margins.jl
- 安装：`using Pkg; Pkg.add("Margins")`
- 文档：https://emfeltham.github.io/Margins.jl/

FormulaCompiler.jl:

- 仓库：https://github.com/emfeltham/FormulaCompiler.jl
- 安装：`using Pkg; Pkg.add("FormulaCompiler")`
- 文档：https://emfeltham.github.io/FormulaCompiler.jl/

这两个软件包均采用 MIT 许可，并与 Julia 的统计生态系统（GLM.jl、MixedModels.jl、StatsModels.jl、DataFrames.jl）集成。

12 附录A：连续交互作用的有限差分法

`second_differences_at()` 函数使用对称有限差分而非自动微分，计算边际效应相对于连续调节变量的局部导数。对于在调节变量值 z 处评估的边际效应 AME(z)：

$$\frac{\partial \text{AME}}{\partial z} \approx \frac{\text{AME}(z+\delta) - \text{AME}(z-\delta)}{2\delta}$$

步长选择：默认情况下，$\delta = 0.01 \times \text{SD}(z)$，其中 SD($z$) 是该调节变量的样本标准差。这种设置在两种误差来源之间取得平衡：（1）有限差分近似产生的截断误差 $O(\delta^2)$；（2）当 δ 过小时，灾难性消去导致的浮点误差。

数值稳定性：对于绝对值较大的评估点（$|z| > 1$），步长的下限设为 $|z| \times 10^{-8}$，以确保 $z \pm \delta$ 在数值上彼此不同。对于接近零的小值，则将绝对 δ 的最小值设为 10^{-9}，以避免退化情况。

标准误：德尔塔法通过有限差分传播参数不确定性：

$$\text{SE}\left(\frac{\partial \text{AME}}{\partial z}\right) = \frac{\sqrt{(g_+ - g_-)^T \Sigma (g_+ - g_-)}}{2\delta}$$

其中，g_+ 和 g_- 分别是 AME($z+\delta$) 和 AME($z-\delta$) 的梯度向量，Σ 是参数协方差矩阵。

13 附录B：基准测试规范

基准测试公式和数据集大小代表了复杂的应用分析，需要针对包含交互项、非线性变换和分类变量对比的复杂广义线性模型/广义线性混合模型规范，在包含$n > 50,000$ 个观测值的数据集上计算边际效应。

13.1 FormulaCompiler 微基准测试

这些基准测试使用 包含 5 个变量的合成数据： `y`（连续型）、 `x` （连续型）、 `z`（正连续型）、 `group`（分类变量 A/B/C）， `b` （布尔型）。

来源： `scripts/benchmarks.jl`。

表 6：FormulaCompiler 微基准测试规格。

基准测试	n	公式	模型
核心行评估	50,000	`y ~ x * group + log1p(z)`	OLS
分配比较	10,000	`y ~ x * group + log1p(z)`	OLS
情景开销	50,000	`y ~ x * group + log1p(z)`	OLS
导数	20,000	`y ~ x * group + log1p(z)`	OLS
边际效应（η/μ）	20,000	`y ~ x * group + log1p(z)`	OLS
德尔塔法标准误	5,000	`y ~ x * group + log1p(z)`	OLS
混合模型	10,000	`y ~ x + (1 | group)`	LMM
简单公式	30,000	`y ~ x + z + group`	OLS
复杂公式	30,000	`y ~ x * group + log1p(z) + x & z + x & log1p(z)`	OLS
尺度不变性（n）	10K–1M	`y ~ x * group + log1p(z) + x & z + x & log1p(z)`	OLS
混合物	20,000	`y ~ x * group + log1p(z)`	OLS
总体平均边际效应	50,000	`y ~ x * group + log1p(z)`	OLS
剖面边际效应	10K / 1M	`y ~ x * group + log1p(z)`	OLS
分类平均边际效应（组）	20,000	`y ~ x + group`	OLS
分类变量平均边际效应（布尔值）	20,000	`y ~ x + b`	OLS
参数缩放	10,000	`y ~ x1 + ... + xp` ($p = \{5, 10, 20, 50\}$)	OLS

13.2 案例研究

表 7：案例研究规格。

Case	n	公式	模型
广义线性模型 Logit + 交互项	20,000	`y ~ x * group + log1p(z)`	逻辑回归
线性模型 +场景	50,000	`y ~ x + z + x * group`	OLS
混合模型固定效应	10,000	`y ~ x + treatment + (1 | group)`	LMM

13.3 R 对比基准测试

与 R 的 marginaleffects 包进行的性能比较采用了复杂逻辑回归模型，该模型反映了使用大型合成数据开展的真实世界分析，并以 Feltham、Forastiere 和 Christakis（2025）为蓝本：

数据集：5,000 个观测值（标准基准）/ 500,000 个观测值（大型基准），包含 ~40 个变量（表 8）。

表 8：R 比较基准数据集中的变量类别。

类别	变量
网络结构	`dists_a_inv`, `dists_p_inv`, `are_related_dists_a_inv`
人口统计特征	`age_p`, `wealth_d1_4_p`, `schoolyears_p`, `man_p`
同质性指标	`hhi_religion`, `hhi_indigenous`, `age_h`, `wealth_d1_4_h`
村庄特征	`population`, `coffee_cultivation`, `market`
分类变量	`socio4`, `relation`, `religion_c_p`

模型：包含嵌套交互项的 65 参数逻辑回归（`Bernoulli, LogitLink`）：

```
@formula(response ~
    socio4 + dists_p_inv + dists_a_inv + are_related_dists_a_inv +
    socio4 & (dists_p_inv + are_related_dists_a_inv + dists_a_inv) +
    (age_p + wealth_d1_4_p + schoolyears_p + man_p + same_building +
     population + hhi_religion + hhi_indigenous + coffee_cultivation +
     market) & (1 + socio4 + are_related_dists_a_inv) +
    relation + religion_c_p + relation & socio4 +
    religion_c_p & are_related_dists_a_inv +
    degree_a_mean + degree_h + age_a_mean + schoolyears_a_mean + wealth_d1_4_a_mean +
    age_h & age_h_nb_1_socio +
    schoolyears_h & schoolyears_h_nb_1_socio +
    wealth_d1_4_h & wealth_d1_4_h_nb_1_socio +
    (degree_a_mean + degree_h + age_a_mean + schoolyears_a_mean +
     wealth_d1_4_a_mean) & socio4 +
    hhi_religion & are_related_dists_a_inv +
    hhi_indigenous & are_related_dists_a_inv)
```

此规格包括二元 × 连续交互作用、包含三重交互的个体层面变量、分类变量交互以及关系层面的同质性指标。

来源：`Margins.jl/test/r_compare/performance_benchmark.jl`。

14 附录C：基准测试环境

所有基准测试均在以下系统上执行：

- 硬件：Apple M4 Pro（14核：10个性能核心、4个能效核心），48 GB 内存
- 操作系统：macOS 15.6.1
- Julia：1.12.0
- R：4.5.1

Julia 计时：使用 BenchmarkTools.jl（Chen 和 Revels 2016），并进行足够次数的预热迭代以确保即时（JIT）编译；报告的数值为各样本中的最小耗时。还使用 BenchmarkTools.jl（Chen 和 Revels 2016）测量内存分配。

R 计时：使用 `microbenchmark` 软件包，每项操作迭代 5 次（大型数据集为 3 次）；报告的数值为耗时中位数。使用 `profmem` 软件包测量内存，记录每项操作分配的总字节数。

两项基准测试均使用相同的数据集导出，以确保可比性。所有边际效应均在响应尺度上计算，并采用 Delta 方法标准误。表 6 和表 7 详细列出了基准测试公式和数据集大小；完整的基准测试脚本见可复现性材料。

参考文献

Ai, Chunrong, 和 Edward C. Norton。2003年。《Logit和Probit模型中的交互项》。*Economics Letters* 80 (1)：123–29。https://doi.org/10.1016/S0165-1765(03)00032-6。Alday, Phillip, 和 Dave Kleinschmidt。2024年。《Effects.jl：计算统计效应的基本构件》。Arel-Bundock, Vincent、Noah Greifer 和 Andrew Heiss。2024年。《如何使用 R 和 Python 的 marginal-effects 解读统计模型》。*Journal of Statistical Software*111（11月）：1–32。https://doi.org/10.18637/jss.v111.i09。Bates, Douglas、Phillip M. Alday 和 Ajinkya H. Kokandakar。2025。《通过分块Cholesky分解评估混合模型对数似然》。arXiv。https://doi.org/10.48550/arXiv.2505.11674。Bates, Douglas、Martin Mächler、Ben Bolker 和 Steve Walker。2015。《使用 Lme4 拟合线性混合效应模型》。*Journal ofStatisticalSoftware*67（10月）：1–48。https://doi.org/10.18637/jss.v067.i01。Bates, Douglas、Andreas Noack、Simon Kornblith 和 Milan Bouchet-Valat。2024。《Glm.jl：Julia 中的广义线性模型》。Bezanson, Jeff、Alan Edelman、Stefan Karpinski 和 Viral B Shah。2017。《Julia：数值计算的新方法》。*SIAMReview* 59 (1)：65–98。https://doi.org/10.1137/141000671。

Bouchet-Valat, Milan, 和 Bogumił Kami´nski。2023年。《DataFrames.jl：Julia 中灵活而快速的表格数据》。Journal of Statistical Software 107（9月）：1–32。https://doi.org/10.18637/jss.v107.i04。Bouchet-Valat, Milan、Bogumił Kami´nski 和 Jacob Quinn。2024年。《Categoricalarrays.jl：用于分类数据的数组》。Brambor, Thomas、William Roberts Clark 和 Matt Golder。2006年。《理解交互模型：改进实证分析》。《政治分析》14 (1)：63–82。https://doi.org/10.1093/pan/mpi014。Chambers, John M.。1992年。《模型的数据》。载于《S中的统计模型》。Routledge。Chambers, John M. 和 Trevor J. Hastie 编。1992年。《S中的统计模型》。加利福尼亚州太平洋格罗夫：Wadsworth & Brooks/Cole。Chen, Jiahao 和 Jarrett Revels。2016年。《噪声环境中的稳健基准测试》。arXiv。https://doi.org/10.485 50/arXiv.1608.04295。———。2024年。《Benchmarktools.jl：Julia 语言的基准测试框架》。Feltham, Eric、Laura Forastiere 和 Nicholas A. Christakis。2025。《孤立村庄中社会网络的认知表征》。《自然·人类行为》，6月。https://doi.org/10.1038/s41562-025-02221-6。Hanmer, Michael J. 和 Kerem Ozan Kalkan。2013年。《曲线背后：阐明从受限因变量模型计算预测概率和边际效应的最佳方法》。《美国政治科学杂志》57 (1)：263–77。https://www.jstor.org/stable/23496556。Kleinschmidt, Dave 和 Phillip Alday。2024年。《Standardizedpredictors.jl：Statsmodels.jl 的标准化回归预测量》。Kleinschmidt, Dave、John Myles White、Milan Bouchet-Valat 和 Douglas Bates。2024年。《Statsmodels.jl：在 Julia 中指定、拟合和评估统计模型》。Leeper, Thomas J.（2014）2025年。《Margins》。https://github.com/leeper/margins。Lenth, Russell V.、Julia Piaskowski、Balazs Banfai、Ben Bolker、Paul Buerkner、Iago Giné-Vázquez、Maxime Hervé 等。2025年。《Emmeans：估计边际均值，又称最小二乘均值》。https://cran.r-project.org/web/packages/emmeans/index.html。Lumley, Thomas。2004年。《复杂抽样样本分析》。Journal of Statistical Software 9：1–19。Markwick, Dean。2022年。《拟合混合效应模型——Python、Julia 还是 R？》。Dean Markwick。2022年1月6日。https://dm13450.github.io/2022/01/06/Mixed-Models-Benchmarking.html。McCabe, Connor J.、Max A. Halvorson、Kevin M. King、Xiaolin Cao 和 Dale S. Kim。2022年。《在广义线性模型中解读非线性概率和计数的交互效应》。《多变量行为研究》57 (2–3)：243–63。https://doi.org/10.1080/00273171.2020.1868966。Mersmann, Olaf、Claudia Beleites、Rainer Hurling、Ari Friedman 和 Joshua M. Ulrich。2024年。《Microbenchmark：精确的计时函数》。Norton, Edward C. 和 Bryan E. Dowd。2018年。《对数优势比与 Logit 模型的解读》。《卫生服务研究》53 (2)：859–78。https://doi.org/10.1111/1475-6773.12712。Quinn, Jacob。2024年。《Tables.jl：Julia 中表格的接口》。Ragusa, Giuseppe。2024年。《Covariancematrices.jl：异方差和自相关一致协方差矩阵估计》。Revels, Jarrett、Miles Lubin 和 Theodore Papamarkou。2016年。《Julia 中的前向模式自动微分》。arXiv。https://doi.org/10.48550/arXiv.1607.07892。Seabold, Skipper 和 Josef Perktold。2010年。《statsmodels：使用 Python 进行计量经济学和统计建模》。SciPy 7 (1)：92–96。https://pdfs.semanticscholar.org/3a27/6417e5350e29cb6bf04ea5a4785601d5a215.pdf。Skrondal, Anders 和 Sophia Rabe-Hesketh。2009年。《多层次广义线性模型中的预测》。《皇家统计学会杂志A辑：社会统计学》172 (3)：659–87。https://doi.org/10.1111/j.1467- 985X.2009.00587.x。Smith, Nathaniel。2011年。《Patsy：使用符号公式在 Python 中描述统计模型》。https://github.c om/pydata/patsy。Stock, James H. 和 Mark W. Watson。2011年。《计量经济学导论》。Addison-Wesley。Wilkinson, G. N. 和 C. E. Rogers。1973年。《用于方差分析的析因模型符号描述》。《皇家统计学会杂志C辑：应用统计》22 (3)：392–99。https://doi.org/10.2307/2346786。Williams, Richard。2012年。《使用 margins 命令估计和解读调整后预测值与边际效应》。《Stata期刊》12 (2)：308–31。https://doi.org/10.1177/1536867X1201200209。Winship, Christopher 和 Larry Radbill。1994年。《抽样权重与回归分析》。《社会学方法与研究》23 (2)：230–57。https://doi.org/10.1177/0049124194023002004。Zeileis, Achim。2004年。《使用 HC 和 HAC 协方差矩阵估计量进行计量经济学计算》。Journal of Statistical Software 11（11月）：1–17。https://doi.org/10.18637/jss.v011.i10。———。2006年。《三明治估计量的面向对象计算》。Journal of Statistical Software 16（8月）：1–16。https://doi.org/10.18637/jss.v016.i09。