arXiv:2302.05705v1 [stat.ME] 11 Feb 2023

一种实用高效的固定枢轴选择算法及其可扩展的 MATLAB 套件

Ivano Azzini、Domenico Perrotta 和 Francesca Torti

欧盟委员会联合研究中心（JRC）

2023年2月14日

摘要

许多统计问题和应用都需要反复计算顺序统计量，例如中位数，但大多数统计与编程环境的主发行版中并未提供线性选择算法。本文介绍一种算法，它在形式上等价于 quickselect，同时保持枢轴位置固定。与目前已知的最佳解决方案相比，这使实现更加简单，也更实用。此外，该算法还支持“预言式”枢轴位置选项，可大幅缩短某些统计应用的收敛时间。我们将算法扩展到加权百分位数（例如加权中位数），可用于关联不同精度测量的数据、图像滤波、medcouple 等描述性统计，以及提升算法中多个预测变量的组合。我们提供了适用于 MATLAB、C语言和 R 的新函数，并将其打包成一个内容广泛的 MATLAB 工具箱，涵盖稳健统计方法，其中许多方法如今都可以通过高效的（加权）选择进行优化。

关键词— quickselect、顺序统计量、加权百分位数、medcouple、稳健方法、MATLAB、C语言、R

1 引言

众所周知， $selection\ problem$ 是从包含 $n \geq k$ 个元素的未排序数组中找出 $k\text{-}th\ order\ statistic$ 的过程；假设这些元素互不相同[1]，且所有排列的可能性相同。因此，它定义在全序集 U 上，如下所示：

输入：一个数组 $A = (a_1 \ldots a_n)$ 和一个整数 k，满足 $1 \leq k \leq n$ 且 $a_i \in U$，对每个 $i = 1$，.....，n。

输出：$a_{(k)}$，即 A中第 k小的元素。

索引表示法（k）表示数组 A 中 k 元素的 $rank$，即顺序统计量列表 $R_k(A) = \sum_{i=1}^{n} I_{[0,\infty)}(a_k - a_i)$ 中的索引，其中 I 为示性函数。秩（k）可通过（k）$= \lceil \frac{p_k}{100} \cdot n \rceil$与统计百分位数 p_k相关联，其中上取整函数 $\lceil \cdot \rceil$ 给出大于˜或等于其自变量的最小整数。 A的 $median$ （记为 A）是 n 为奇数时选择问题的一个特例，即：若对于整数 m 有 $n = 2m + 1$，则

[1]这一常见假设简化了形式化表达和推导。实际的解决方案也适用于存在重复值的情况。

$\tilde{A} = a_{(m+1)} = a_{(\frac{n+1}{2})}$。否则，当 $n = 2m$ 时， $\tilde{A}$ 是两个中间顺序统计量 $a_{(m)} = a_{(\frac{n}{2})}$ 和 $a_{(m+1)} = a_{(\frac{n}{2}+1)}$ 的算术平均值。

选择算法的时间复杂度 $T(n)$ 通过统计 A 中元素之间的比较次数和交换操作次数来衡量。最优秀的分区方法——例如著名的 quickselect [Hoare 的 Find, 31, 32] ——平均而言具有线性时间复杂度，但在最坏情况下 $T(n)$ 会变为二次时间。理论上在最坏情况下也能保持线性时间复杂度的解决方案，在实现上却面临诸多问题；而且在实际应用中，其 CPU 时间甚至比基于排序的朴素方法 [8] 更长——尽管人们普遍承认，应用需要更多“实用有效的算法” [53, p. 347]。本文探讨了一种易于实现且满足这一功能需求的算法，因此将其命名为 simpleselect。

这些简化得益于一种迭代式固定枢轴位置策略，该策略围绕位置 k 进行原地数组交换（第 2 节）。它的效率与 quickselect 相当（第 3 节），而通常通过对初始数组进行随机化，可以避免出现二次时间运行的情况；随机化只需付出少量代价，即 $\mathcal{O}(n)$ 次交换操作（我们使用基于 [21, pp. 26-27] 的结果的反向洗牌 [37, pp. 124-125], ）。因此，quickselect 和 simpleselect 是等价的“拉斯维加斯”算法：二者都会产生相同的正确输出；但前者在每个执行步骤中随机化枢轴位置，后者则只对初始数组随机化一次。

固定枢轴及由此带来的简化，促成了两项实用扩展。第一项是引入一个“预言机”，用于提示在 A 中何处可以找到顺序统计量的值。第4节通过两个知名的稳健多元估计量展示了它的优势。第二项是扩展到加权百分位数，相关内容见第5节。

为了提高可移植性和易用性，我们已在 MATLAB、C语言和 R 中实现了 simpleselect 及其加权版本，因为这些语言的主发行版均未提供相应的替代方案。它们的文件名分别为 quickselectFS 和 quickselectFSw，以强调其与 Hoare 算法的等价性。我们通过 Python 示例说明如何调用 C 函数（附录 A.3）。为了便于在不同环境和一般模拟设置下评估这些函数，我们还新增了一个 MATLAB 函数，用于重现 R 软件通过梅森旋转算法生成的随机数（附录 A.4）。

显然，只有将这些新函数与其他依赖重复计算顺序统计量和加权百分位数的一般方法结合使用，才能体现其实际价值。因此，我们将它们打包到 FSDA [47, 48], 中。FSDA 是一个内容广泛的 MATLAB 库，专为稳健统计而设计，并通过 GitHub 开放协作贡献。出于同样的原因，我们还为该软件包增添了高效的新函数，用于计算 quickselect 类过程中的比较次数的理论分布 [5] （第 A.2 节），以及 medcouple [13] （第 5.3 节）；medcouple 是一种稳健偏度估计量，会大量调用加权中位数。

我们在配备 16 GB 随机存取存储器的英特尔 2.9 GHz 四核 i7 中央处理器上运行了模拟。我们使用 MATLAB R2021b 版本开发，但在早得多的版本中结果也一致。

2 Simpleselect

2.1 概念

如果数组 A 已完全有序，或至少前 k 个位置有序，那么显然 $a_{(k)} = a_k$。但是，如果我们只关心元素 $a_{(k)}$，那么对数组进行（部分）排序就超出了所需。Knuth [37, pp. 207–219] 对历史渊源以及为寻找 $\mathcal{O}(n)$ 时间内解决方案所付出的努力作了精辟的梳理；这些方案依赖于对快速排序 [30, 18] 的改造，采用了包含以下步骤的 *divide and conquer* 分区方法：

1. 用于选择元素 $a_s \in A$ 的位置 s 的判据，该元素称为 *pivot*。

2. 一种重新排列 A 中其他元素的过程，使位置 1 到 $s-1$ 上的元素小于位置 $s+1$ 到 n 上的元素。

根据枢轴与目标顺序统计量的相对位置，将 (1) 和 (2) 递归或迭代地应用于 A 的两个部分之一：

- 如果 $s > k$ 新目标是左侧部分中的第 k 个元素；
- 如果 $s < k$ 新目标是右侧部分中的第 $(k-s)$ 个元素；
- 如果位置 s 上的元素就是所需的顺序统计量 $a_{(k)}$。

在 Hoare 的 Find 中，枢轴是随机选取的。更复杂的基于分区的算法，例如中位数的中位数 [10] 和 introselect [44], ，在概念上与之相同，但在如何以审慎（有时颇为晦涩）的准则选择枢轴方面有所不同，以实现线性最坏情况。

下文将展示 simpleselect 策略的实际优势：该策略将枢轴固定在位置 k ，并反复交换其值与周围数值（右侧/左侧部分中的数值），直至枢轴获得正确的值；而不是移动枢轴位置（随机移动或采用其他策略），直到枢轴到达目标位置 k。这种简化的迭代策略，其性能仍可与已知的最佳解决方案相媲美。摒弃递归还能确保其可扩展至大型数组，因为这样可以避免大多数计算环境中常见的栈空间管理问题。我们发现了一个类似的策略，但仅限于中位数计算，见 https://rosettacode.org/wiki/Rosetta_Code。据我们所知，其性质尚未得到研究。

2.2 原地置换

查找 $a_{(k)}$ 需要对 A 的元素进行置换（$\sigma(1)\ldots,\sigma(n)$），使得 $a_{\sigma(1)}, a_{\sigma(2)}, \ldots,$ $a_{\sigma(k-1)} \leq a_{\sigma(k)}$，或者等价地， $a_{\sigma(k)} < a_{\sigma(k+1)}, a_{\sigma(k+2)}, \ldots, a_{\sigma(n)}$。这可以通过一系列由元素 a_k确定的交换来实现。用 A^t 表示步骤 $t \in \mathcal{N}$时数组的状态，用 a_k^t 表示其位于位置 k的元素，并且

$$A_{L_k}^t = \{a_i^t \in A^t \mid a_i^t \leq a_k^t \quad \wedge \quad i < k\}$$
$$A_{R_k}^t = \{a_i^t \in A^t \mid a_i^t > a_k^t \quad \wedge \quad i > k\}$$

用集合表示步骤 t 时满足所需排序要求的 a_k^t 的左侧和右侧元素。当 $\left|A_{L_k}^t\right| = k-1$ 或等价地 $\left|A_{R_k}^t\right| = n-k+1$时，对于某个 t，即可找到顺序统计量 $a_{(k)}$。在步骤 t，集合 $A_{L_k}^t$ 至多可通过 $k-1$ 次比较构建。请注意，鉴于该问题具有对称性：（i）我们可以不检查 $A_{R(k)}^t$；（ii）求解 $k > \lceil \frac{n}{2} \rceil$ 与求解 $k < \lceil \frac{n}{2} \rceil$并无区别；（iii）查找 $k = \lceil \frac{n}{2} \rceil$ 是要求最高的情形。现在，我们可以用算法形式表述选择问题：

输入： $A =$（$a_1 \ldots a_n$）和 k当$\left|A_{L_k}^t\right| \neq k-1$ 执行时，选择元素 a_k 作为枢轴 a_k^t；
通过依赖于与 a_k^t进行比较的一系列交换，构建集合$A_{L_k}^t$；评估$A_{L_k}^t$的基数；结束
输出： a_k

算法的关键部分可以从代码清单（1）中识别出来，该清单由函数 quickselectFS.m 提炼而成。代码使用三个变量来控制 $A_{L_k}^t$ 和 $A_{R_k}^t$的推进：其中一个是位置，另外两个是“哨兵”左侧和右侧，使得在每个迭代步骤 t中，左侧 $\leq$ 位置，右侧 $>$ 位置。请注意，位置的每次变化都伴随着一次交换操作。还要注意，鉴于该问题的对称性，我们可以只关注数组的左侧；这意味着

循环在以下条件下终止：左侧 $=$ 位置 $+1 = k+1$。最后需要说明的是，第 13 行的 `for` 循环仅运行到 右侧-1：这是因为 A(右侧) 在第 10 行接收了枢轴元素，因此当 $i =$ 右侧时，第 14 行的 `if` 语句永远不会为真。

3 比较次数统计

本节将说明，simpleselect 的性能与 quickselect 相当，因此适用于第4节所讨论的应用。放弃递归后，便无法使用递推方程推导理论时间界限。因此，我们采用一种简单的计数方法。

3.1 最坏情况

我们开始计算最坏情况下的比较次数 c（n），这种情况发生在每个循环中 t 集合 $A^t_{L_k}$ 恰好包含 $t-1$ 个元素时。当我们在按递增顺序排列的元素中查找最大值（$k = n$）时会出现这种情况，但最后一个元素包含最小值（$a_1 < a_2 < \ldots < a_{n-1}$ 和 $a_n = a_{(1)}$）。在这种情况下（以及 $k = 1$的对称情况），变量位置在`for` 循环内部从未被修改。图1由一个用于展示

清单 1：从 quickselectFS.m 提炼出的代码。也提供 R、C语言和 C-MEX 文件版本。

```
1  函数 [kE] = quickselectFS(A,k)
2
3  % 使用 SimpleSelect 查找第 k 个顺序统计量
4  left          = 1;
5  右侧          = numel(A);
6  位置
7  = -1;
8  当 (位置~=k)
9      %% 第一部分：选择位置 $k$ 处的元素作为枢轴
10     枢轴          = A(k);                          %|
11     A(k)          = A(right);                      %| Swap
12     A(right)  =枢轴;                               %|
13     位置  = 左侧;                                  %     c2
14     %% 第二部分：构建集合 $A  {L k}^{t}$
15     对于 i=左侧:右侧-1
16         如果(A(i)<枢轴)
17             缓冲区          = A(i);                %|
18             A(i)            = A(位置);             %| 交换
19             A(位置) = 缓冲区;                      %|
20             位置            = 位置+1;              %     c1
21         end
22     end
23     A(右侧)         = A(位置);
24     A(位置) = 枢轴;
25     %% 第三部分：评估 $A  {L k}^{t}$ 的基数
26     如果 (位置 < k)
27         left  = 位置 + 1;                          %     c2
28     else
29         右侧  = 位置 - 1;
30     end
31 end
```

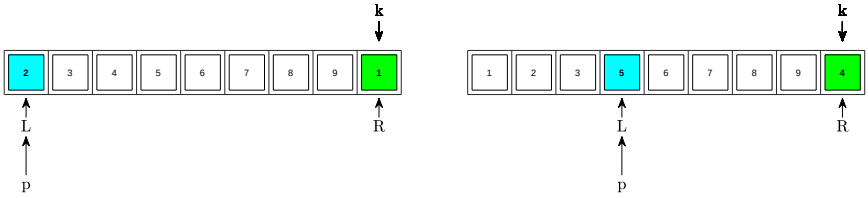


图1：simpleselect最坏情况下。左图为第 次 迭代：$|A^1_{L_9}| = 0$。右图为第 次 迭代：$|A^4_{L_9}| = 3$。若要重现，请使用`quickselectFS demo(A,k)`，其中 $A = [2,3,4,5,6,7,8,9,1]$ 和 $k = 9$。

simpleselect的函数生成，展示了这种不利排列的数组在第一次和第四次`while` 迭代时的状态。

由于比较发生在代码清单 (1) 的第 6、14 和 24 行，我们将计数分解指定为 $c(n) = c_6(n) + c_{14}(n) + c_{24}(n)$。然后用 (t,l,r,p) 表示步骤 t 时变量左侧、右侧和位置的状态。采用此记法，我们有：

$$\begin{aligned}
(t,l,r,p) = (1,1,n,1) \quad &\Rightarrow \quad c(n) = 1 + (n-0) + 1 \\
(t,l,r,p) = (2,2,n,2) \quad &\Rightarrow \quad c(n) = 1 + (n-1) + 1 \\
&\vdots \\
(t,l,r,p) = (n,n,n,n) \quad &\Rightarrow \quad c(n) = 1 + (n-(n-1)) + 1
\end{aligned}$$

因此，比较次数的最坏情况为二次时间：

$$c(n) = 2n + n^2 - \sum_{i=0}^{n-1} i = 2n + n^2 - \frac{(n-1)n}{2} = \frac{n^2+5n}{2}. \tag{1}$$

请注意， $c_6(n)$ 和 $c_{24}(n)$ 仅涉及 *index comparisons* 短整数，这

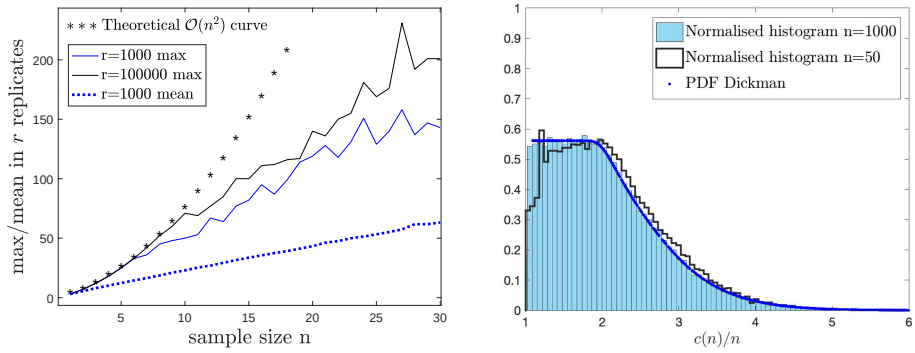


图2：simpleselect。查找均匀生成的样本中最大值的比较次数的经验值 $c(n)$。左图：在 r 次重复试验中，max($c(n)$) 和 mean($c(n)$)；均值情形的线性拟合为 $\hat{c}(n) = 2\,n$。右图： $n = 1000$（柱状样式）和 $n = 50$（阶梯图样式）的 $c(n)/n$ 经验分布，并叠加 Dickman分布（平移 1），该分布使用参数 $\beta = 1$ 通过 FSDA函数 vervaatxdf 获得。

通常比对数组内容执行 $data\ comparison\ c_{14}(n)$ 的开销小得多。如果忽略它们，方程 (1) 可简化为 $c(n) = (n^2 + n)/2$。

3.2 平均情况

通过考虑平均情况，我们可以分析实践中如何实现线性复杂度，而简化后的代码几乎使这一点变得不言自明。在首次执行`while` 语句时，第 6 行进行一次比较以检查退出条件，随后在第 24 行进行一次比较，并且另一次比较 n 在第 13–20 行的 `for` 循环内部执行。对 位置 设置的退出条件逐渐逼近：

c1：第 18 行，每当 $a_i <$枢轴时，位置便向前移动一步；c2：第 25 行，左侧跳到位置之后的单元格，这也使得
位置向前迈进一步，在第 11 行被设置为左侧。 o

如果 A 是从同一分布中提取的元素的随机样本，我们预计（$n-1$）/2 在初始扫描中，位置 将增加 A （见附录A.1）。同样，变量 左侧 在第 25 行和第 11 行的增加——变量 右侧 的增加也同样如此——会进一步缩短与 位置之间的距离，并且最多执行一次。随后，在第二次执行 `while` 语句时，预计 `for` 循环将运行约（少于） $n/2$ 个数组单元格（实际上为（$n-1$）$/2-1=$（$n-3$）/2）。后续步骤也依此类推：

$$\begin{array}{lll}
(t,l,r,p) = (1,1,n,1) & \Rightarrow & c(n) = 1+n+1 \\
(t,l,r,p) = (2,\cdot,n,\cdot) & \Rightarrow & c(n) = 1+n/2+1 \\
(t,l,r,p) = (3,\cdot,n,\cdot) & \Rightarrow & c(n) = 1+n/4+1 \\
 & \vdots & \\
(t,l,r,p) = (\log_2 n, n, n-1, n) & \Rightarrow & c(n) = 1+n/2^{(log_2 n-1)}+1
\end{array}$$

因此，总数为：

$$\begin{aligned}
c(n) &= 2\log_2 n + n\sum_{t=1}^{\log_2 n}\frac{1}{2^{t-1}} && = 2\log_2 n + n\sum_{i=0}^{\log_2 n-1}\frac{1}{2^i} \\
&= 2\log_2 n + n + n\sum_{i=1}^{\log_2 n-1}\frac{1}{2^i} && = 2\log_2 n + n + n(1-\frac{1}{2^{\log_2 n-1}}) \qquad (2) \\
&= 2\log_2 n + n + n(1-\frac{2}{2^{\log_2 n}}) && = 2\log_2 n + n + n(1-\frac{2}{n}) \\
&= 2\log_2 n + 2n - 2 && < 2n+18 \qquad \text{for } n < 1000 \qquad (3)
\end{aligned}$$

其中，式（2）中几何级数前 $\log_2 n - 1$ 项的部分和通过标准代数步骤计算得出；最终的上界（3）将对数项简化为一个常数，该常数随 n 的增长相较于 $2n$ 几乎没有增加。请注意，这一结论与图2的实证结果一致，后者确实预示着一个 $2n$ 项；同时，它也与基于递归划分方法递推关系的已知渐近结果一致 [，如文献 40，定理 1] 所示。

3.3 比较次数及其经验分布

图2左侧面板用符号“$*$”显示了方程（1）的变化过程，以及 simpleselect 在 r 次重复试验中，从在1到（为避免重复） $10000\,n$之间均匀抽取的 n 个整数集合中查找最大值所需的实际比较次数。对于较小的样本量，最大值的两条曲线遵循方程（1）的二次曲线形状，因为 r 次重复试验足以达到最坏情况（按上述顺序提取 n 个值）。随后它们

输入： $pdfName$、 $nSet$、 $kSet$、 r对于 n=nSet，执行 对于 k= kSet，执行 对于 i=1:r，执行$A_n \leftarrow n$—从分布 $pdfName$中抽取样本；EtimeCount(n、 k、 i) ← tic SimpleSelect(A_n、 k) toc；NumOpCount(n、 k、 i) ← SimpleSelectOpCount(A_n、 k)；结束 结束 结束 grpVar ← $[nSet, kSet]$；whichStat ← [“中位数”、“均值”、“最大值”、“最小值”]；输出：NumOpStats ← grpstats(NumOpCount,grpVar,whichStat)；EtimeStats ← grpstats(EtimeCount,grpVar,“中位数”)；

算法 1：用于评估 simpleselect 的执行时间和基本操作次数的伪代码。整数数组 $kSet$ 包含感兴趣的顺序统计量。根据 $pdfName$指定的分布生成随机样本，样本量由整数数组 $nSet$定义。输出中的性能指标根据 r 次重复试验，作为 $nSet$ 和/或 $kSet$ 的函数计算。

开始线性增长，其 *empirical worst-case*约为 $5n$（拟合范围为 $r = 100000$ 和$_{bn\ average}$ $2n$，最大至 1000）。类似地，第三条（点状）线表明，只需进行比较即可找到最大值。

我们还检查了 c（n）$/n$的经验分布：当 n 足够大时，它服从图2右侧面板所示的 Dickman分布；该分布是 Hoare 的 Find 的极限分布（参见 [34] 以及 [26, 27]的相关研究）。值得注意的是，即使样本量较小（$n = 50$），Dickman分布的近似效果也相当好。鉴于 Dickman分布在此背景下的核心地位，我们按照附录 A.2 中提及的研究，提供了用于计算更广泛的 Vervaat 类的概率密度函数和累积分布函数，以及生成其随机变体的函数。这些函数可用于研究不同于 Hoare 的 Find 的划分算法，这些算法可能会得到其他极限结果 [39]。

3.4 运行时间结果

我们的运行时间结果基准是内部（内置）的 sort 函数 – $SORT_{int}$–，典型的 MATLAB 用户会使用该函数，因为标准发行版没有专门用于顺序统计量的函数（prctile 和中位数使用 sort）。为评估 simpleselect 的实际性能，我们考虑实现的两种执行模式：即时编译（jit），即 MATLAB 直接分析并翻译源代码 *during* 后运行（这是 MATLAB 的标准执行模式，R 中也可用）；以及 mex 文件模式，在该模式下，我们将 simpleselect 的一个 C 代码实例编译为机器码 *before*，供后续执行。随后，将这两个实例（下文分别以 SS_{jit} 和 SS_{mex}标识）与 Numerical Recipies in C [46, 第 8.5] 节中的经典 quickselect 实现 – $NURE_{mex}$–，以及 C 语言 ++ 函数 nth element 中提供的 introselect – $NthEL_{mex}$ – 进行比较；后两者均编译为 mex 文件。后者还由 [38], 以 mex 文件形式发布，但使用了未公开的调用，这些调用未来可能发生变化并导致程序失效。

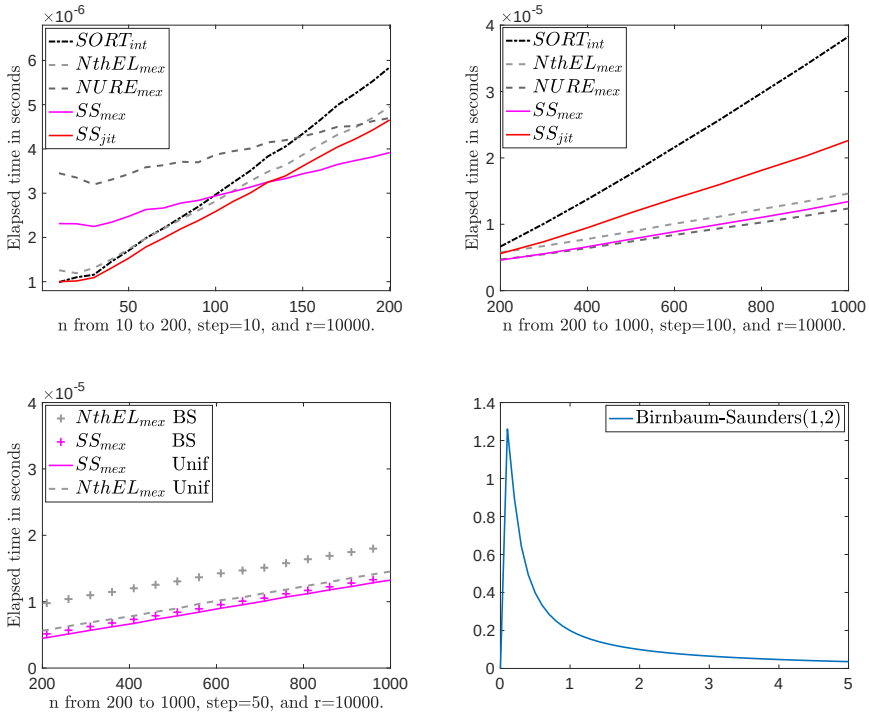


图3：查找中位数。上方图表：simpleselect (SS)、Numerical Recipies in C 中的 quickeselect ($NURE$) 、 C 语言 ++ nth element 函数中的 introselect ($NthEL$) 以及 MATLAB 内部排序 ($SORT$) 的耗时（秒）。左下图表：灰色和洋红色的 “+” 符号分别表示从右下图所示的 Birnbaum-Saunders (BS) 分布中提取 A 时 introselect 和 simpleselect 的表现；另外两条曲线使用均匀分布 (Unif)，与上方图表相同。

评估按照伪代码1进行（实际代码作为补充材料提供）。对于查找下中位数所获得的时间结果，针对取值范围在 10 到 1000 之间的n个值，在图3中报告为 10000 次重复试验的中位时间。因此，引入偏差的可能性（例如即时编译可能产生的初始延迟）被完全排除。结果表明:

1. 上方图表。simpleselect (SS_{jit}) 的纯 MATLAB实现即使在样本量较小 ($n < 100$) 时也具有竞争力，但替换 sort 函数带来的耗时下降有限。样本量增大后，其优势显著提升；若使用 MEX编译版本，优势更加明显。还需注意，其性能与 $NURE_{mex}$ 和 $NthEL_{mex}$ 相当。

2. 底部面板 simpleselect 的一项主要优势，以及其他仅依赖 A 中元素成对顺序的类似算法的优势，在于其性能不受数据分布的影响。相反，introselect 试图利用数据分区上的统计量（均值和中位数）实现最优的最坏情况性能，但可能会受到图中所示的尖峰数据分布影响。 [55] 对一种类似算法 binmedian 进行了广泛的评估练习，这说明了在实际应用中采用复杂解决方案所面临的种种困难。

4 稳健统计中的选择应用

许多稳健方法需要利用顺序统计量，在一组 p 变量观测值 $X_n = \{x_1, \ldots, x_n\}$ 中识别出无离群值子集。我们通过两个案例研究说明，为此目的采用 simpleselect 的优势：在第一种情况下，所需的顺序统计量始终固定在同一点（通常是中位数）；而在另一种情况下，它则从 $p+1$ 增至 n，从某个递进阶段开始，将大多数最大值留在数组的右侧。

最小协方差行列式（MCD）。MCD 旨在从 n p 个变量的观测值中识别出一个 h 子集，使其协方差矩阵 [52, 49] 的行列式最小。精确解需要评估 $\binom{n}{h}$ 种情况，而通常由于 $(\frac{n}{k})^k \leq \binom{n}{h} \leq (\frac{en}{k})^k$ [15, p, 计算起来十分困难。1186]。幸运的是，MCD 有一种近似解 [50]：随机选取许多初始子集，并对每个子集应用具有以下性质的不动点迭代方案：若在第 t 步，我们有一个 h 子集 H_t，其经验均值和协方差矩阵分别为 $\hat{}\mu_t$ 和 $\hat{}\,\Sigma_t$，且观测值 i 的（马氏）$\hat{}$距离为 $d_i = d(x_i, \hat{}\mu_t, \Sigma_t)$，那么由平方距离 $d_i^2 \leq d^2{}_{(h)}$ 的观测值构成的新 h 子集 H_{t+1} 满足 $|\hat{}\Sigma_{t+1}| \leq |\hat{}\Sigma_t|$。因此，每次迭代都需要计算顺序统计量 $d_{(h)}$。通常将 h 设为 $[(n+p+1)/2]$，其值约为中位数；还可以通过加权步骤将其增大，以提高估计量效率。初始随机子集 H_0 的大小为 $p+1$，这样可降低在计算初始估计值 $\hat{}\mu_0$ 和 $\hat{}\,\Sigma_0$ 时纳入离群值的可能性。循环会持续进行，直到两个协方差矩阵的行列式满足相等条件。通常只需几次迭代即可收敛，但需要抽取并迭代的子集数量可能达到数千个，适用的顺序统计量计算也同样如此。

前向搜索（FS）前向搜索（FS）[3] 通过将迭代过程与检验步骤相结合，使 h 的值适应数据。迭代生成一系列参数估计值，而检验则根据这些参数确定 h 值并检测离群值。更具体地说，迭代从对少数经过谨慎选择的观测值（例如 $m_0 = p+1$）进行的稳健拟合开始。接着，选取与基于$\hat{}\mu_0$ 和$\hat{}\,\Sigma_0$得到的稳健拟合之间平方距离最小的 $m_1 = m_0+1$ 个观测值。在第 m_1 步，再次计算参数估计值，并重复这一过程，直至纳入所有观测单位（$m = n$）。因此，迭代的每一步都需要：(i) 应用选择算法，以确定根据$\hat{}\mu_m$ 和$\hat{}\,\Sigma_m$ 计算得到的 $i = 1$，....., n 个距离中第 $d^2{}_{(m+1)}$ 小的顺序统计量；(ii) 进行 n 次比较，以识别满足$d_i^2 \leq d^2{}_{(m+1)}$的观测值 i。因此，共有 $n - m_0 + 1$ 个步骤需要计算不同（递增的）顺序统计量。

我们说明 simpleselect 在 MCD 和 FS 中的用法，重点关注其可选参数“预言机”可能带来的好处。该参数提供 j 中某个元素的索引，该元素位于 A 中，可能包含所需的第 k 个顺序统计量，或与之接近。此选项只会在开始处理前交换 $A(j)$ 和 $A(k)$；只要初始枢轴猜测没有选得太差，就能降低陷入最坏情况的概率，并提升平均情况性能。例如，在某个 FS 步骤 m 中，若变量 minMDindex 存储了构成潜在离群值组的各个单位中马氏距离最小者的索引，那么要将基本子集 bsb 的维度增加一个单位，可以使用下方第 9 行的选项，其中 $j \leftarrow$ minMDindex；也可以使用不带预言机的标准 simpleselect（第 6 行），或作为基准方法的 sort（第 3 行）：

```
1                切换bsb 更新方式为大小写
2                'sort'
```

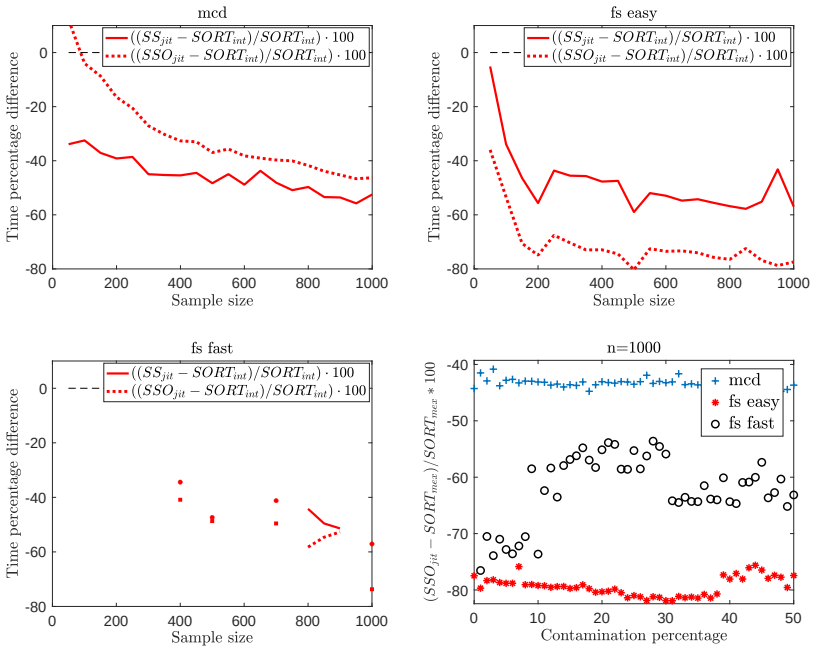


图4：在由二元正态分布生成的数据上，simpleselect 的神谕选项对 MCD 和两种前向搜索实现的影响。上方和左下方图：对于不同样本量，使用和不使用神谕选项时，相对于 SORT基线 的时间百分比减少量。右下方图：固定 $n = 1000$ 、污染水平变化时的时间减少量。

```
3                [~ , zsi] = sort(MD);bsb = zsi(1:m+1);
4          情况 'quickselectFS'k=quickselectFS(MD,m+1);
5          bsb=(MD<=k);情况 'quickselectFSoracular'k=
6          quickselectFS(MD,m+1,minMDindex);bsb=(MD
7          <=k);结束
8
9
10
11
```

在 MCD 中，迭代加权最小二乘步骤采用了类似的方法，其中位置和形状矩阵会反复更新，直至收敛。在这种情况下，我们需要找到协方差行列式最小的 h 个观测值子集，而且没有与第 9 行对应的操作；我们能做的最好尝试，是像此处第 12 行那样，从当前 h 个观测值集合之外的观测单位中随机选择一个候选项：

```
12          k =quickselectFS(MD,h+1,h+1+randi(n-h-1,1,1));
```

图4左上面板显示，随机选择神谕选项并不能改善最小协方差行列式：事实上，SSO曲线远高于SS。对于前向搜索，情况恰好相反，SSO的优势十分明显；不过，我们必须区分该算法的两种实现：一种逐步遵循原始FS公式[2, 3]，见R软件包forward和较新的FSDA函数FSMmmdeasy.m；另一种是该算法的快速（但几乎难以阅读）版本 [48]，在大多数前向搜索步骤中，它通过逻辑运算更新子集，而不使用sort或quickselectFS。快速版本——FSDA函数FSMmmd.m——将quickselectFS的应用限制在多个观测单位退出子集的步骤中，这种情况称为“交换”。

这一效果在图4左下面板中清晰可见，其中时间信息仅间歇性地出现在较大的 n 值处。不过，结果表明，即使存在交换，从而使 j的选择存在不确定性，神谕选项仍能加快该方法。

鉴于数据受到污染时交换次数会增加，我们研究了以下影响：
在传入最小协方差行列式和两种前向搜索实现的正态双变量样本中，对不断增加数量的观测单位施加固定偏移，从而形成两个独立群组
在传递给最小协方差行列式和两种前向搜索实现的正态双变量样本中。作为
不出所料，图4右下角面板显示，最小协方差行列式的时间收益
保持稳定（趋近于 -40%），因为根据定义，该算法不受交换影响。
在快速前向搜索中，污染比例较低（最高达 10%）时，这种影响最为显著
百分比，而在较高的比例（超过 35%）时则能看到收益；这是因为
我们从受污染的观测单元开始执行前向搜索，因此最初预期会落入非常
不稳定的估计，并发生大量交换。在标准前向搜索中，收益显著
（约为 -80%），并且随着污染比例接近 40% 而增加；而在这里，
污染所带来的收益更难以体现，因为大多数调用
quickselectFS 时都不依赖交换。

5 扩展至加权选择

许多统计问题都可以归结为从赋予非负权重的元素中选择一个顺序统计量。对于整数权重，这意味着在扩充后的集合中选择顺序统计量，其中每个元素按照相应的权重重复出现。我们介绍一般问题，并聚焦于 $weighted\ median$ （第 5.1 节），即 50% $weighted\ percentile$。我们将加权百分位数作为 simpleselect 算法的自然扩展，在函数 quickselctFSw 中实现（第 5.2 节）。我们还展示其在计算量很大的稳健偏度度量 medcouple 中的应用（第 5.3 节）。

5.1 加权中位数根及其应用

˜第 1 节中定义的中位数 A 也能解决以下优化问题

$$\tilde{A} = \underset{a}{\operatorname{argmin}} \sum_{i=1}^{n} |a_i - a| \tag{4}$$

直观而言，证明依赖于这样一个事实：绝对偏差之和关于 a 的导数为 $\sum_{i=1}^{n}$sgn($a_i - a$)，只有当正项的数量等于负项的数量时，该导数才为零；而这发生在 a 为中位数时。如果绝对偏差按正数 w_i加权，那么最小化问题便导出加权中位数

$$\tilde{A}_w = \underset{a}{\operatorname{argmin}} \sum_{i=1}^{n} w_i |a_i - a| \tag{5}$$

这个优化问题有着引人入胜的历史渊源：早在1760年，Boscovich和Simpson就将其表述为一条使观测值与该直线偏差之和最小的直线。拉普拉斯在其《Methode de Situation》（1818年）中给出了直线斜率的解，指出加权中位数可以求解受约束的最小绝对偏差回归；该问题可通过在（5）中将$w_i = |x_i - \overline{x}|$ 和 $a_i = （y_i - \bar{y}）/（x_i - \bar{x}）$ 替换为（x_i， y_i），即$i = 1$，...，n，这一由分别具有一个自变量和一个因变量的点构成的二维样本而得到。拉普拉斯认识到，（5）中的 $\tilde{A}_w$ 等于$a_{(k^*)}$，其中 k^* 通过考察权重$w_{(i)}$ 的顺序统计量并返回

与权重对应的最小 k，其累积权重超过总权重的50%，即：

$$k^* = \min\left\{k \;\middle|\; \sum_{i=1}^{k} w_{(i)} \geq \frac{1}{2}\sum_{i=1}^{n} w_i\right\} \tag{6}$$

这个问题通过 [19] 提出的“双中位数”（即截距的解）得以解决；一个世纪后， [6, 9]又通过单纯形算法解决了这个问题。重新审视并拓展这些思想的现代概念包括 [51] 提出的“对偶图”和“回归深度”，以及 [16, 23]提出的稳健时间序列平滑与滤波。

加权中位数和百分位数在工程学和机器学习领域也有引人关注的应用场景；在这些领域，大规模应用离不开高效算法。例如，非线性数字滤波设备采用加权百分位数进行噪声消除 [1, 56], ；医学实验则在个体估计的精度差异较大时使用加权百分位数 [11]。在机器学习中，提升程序的情形颇为有趣：这类程序旨在通过组合多个较弱的估计器来生成准确的预测（统计学在加性模型理论中研究这一情形 [24]）。 [22] 最初提出的提升方法将最终预测构建为各个模型的加权平均，但同时也表明，回归中的最优预测（AdaBoost.R2）应基于弱学习器的加权中位数。 [35, 36] 为实现一定程度的稳健性提供了采用加权中位数的其他动机； [7] 则针对 [0, 1] 给出了相关动机。近期， [4] 展示了这一方法在敏感性分析中的一项应用；在这种分析中，预测（因此也包括加权中位数）必须计算许多次。

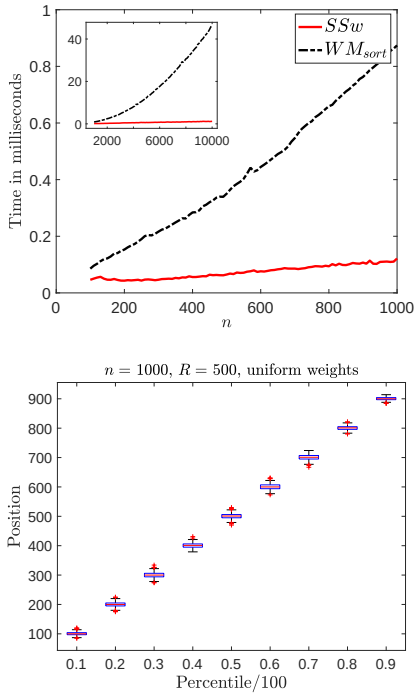
5.2 加权简易选择

方程 6 的值 k^* 由函数 quickselctFSw（列表 2，输出变量 kstar）返回，输入参数为 $p = 0.5$，即百分位数 $100p = 50$%。更一般地，对于任意百分位数 $100p \in [0, 100]$，该函数会返回一个值k_p^* 以及权重的一个排列，使得围绕k_p^* 的权重分区之和尽可能相等，从而使下列差值尽可能小：

$$\sum_{i=1}^{k_p^*-1} w_i - \sum_{i=k_p^*+1}^{n} w_i. \tag{7}$$

图 5 右上方子图中的直方图显示，quickselctFSw 得到的目标函数值（5）与最优 $\mathcal{O}$（$n$log n）解[29], 完全相同；该解通过对权重进行排序，找出总权重中和最小且达到总权重一半的权重（图例中分别标为 SSw 和 WMsort）。同一图中的左上方子图显示，即使对于较小的 n，我们的 $\mathcal{O}$（n）解也具有优势。值得注意的是，这里同样没有编译代码便得到了这一结果。图中还显示，当 n 达到数千左右时， $\mathcal{O}$（$n$log n）解会变得非常不实用。最后，左下方子图显示，我们将方法扩展到任意加权百分位数后，结果符合预期：对于均匀分布的权重，quickselctFSw 返回的位置 k_p^* 非常接近所需的输入百分位数 $100p$。

quickselctFSw 的代码清单 2 是对 quickselctFS 的自然扩展。只需将清单 1 的核心部分嵌入一个循环中（从第 9 行开始），通过将加权中位数方法扩展为 [8]，检查和或权重分区（7）的状态。quickselctFS 的核心部分（第 11–33 行）会在数据上执行，并假设加权百分位数位于位置 $k = \lceil np \rceil$ （第 6 行）。第一次迭代会对向量 D 进行排列，使位置 (1 : $k-1$) 处的权重小于第 k 个权重。此时，我们检查数组是否满足权重平衡



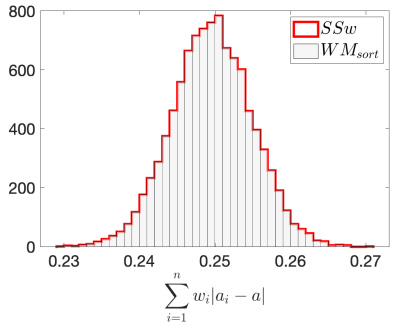


图 5：加权简易选择（SSw）与 $\mathcal{O}(n\log\ n)$ 解法 [29]（WMsort）。左上：计算加权中位数所用时间，适用于 $n \in [100, 10000]$。右上：权重和数据取自 $U(0, 1)$ 及 $n = 1000$ 时的目标函数值。左下：quickselctFSw 针对不同百分位数 $100p$ 和 $n = 1000$ 返回的 kstar 位置 k_p^* 。

（第 36、37、41 行）。如果满足，则停止迭代（第 39 行）。否则，我们再次对 $D(1:k,:)$（使用 $k+1$）或 $D(k:n,:)$（使用 $k-1$）应用 quickselctFS 的核心部分。迭代会移除或添加权重，以逐步达到最优条件。

代码的大部分耗时（60% 到 70%）都花在循环（第 16–25 行）中交换行（第 19–21 行）上。由于要交换的单元格对预计遵循 MATLAB 矩阵的列主序，我们先遍历数据元素 D(:,1)，它们在内存中连续存储，然后再遍历权重元素 D(:,2)。事实证明，这种逐单元交换方法确实比传统的整行交换指令快得多

```
buffer=D(i,:);      D(i,:)=D(position,:);      D(position,:)=buffer;
```

最后需要指出的是，一般而言，总权重中恰好达到 $100p$% 的那个位置位于两个数据值之间。因此，在第 37 行，k 的两个值都可能使 D(k,2) 满足方程 (7)。我们决定通过返回这两个值中的较小值来解决平局。对于 50% 百分位数而言，这就是 *lower weighted median*。将 quickselctFSw 与选择上加权中位数或两者之间某个插值量（例如均值）的方案进行比较时，必须考虑这一点。

5.3 快速 medcouple 的应用

加权中位数广泛用于一种快速算法中，以计算 medcouple [13], ——一种稳健偏度度量，用于调整箱线图的须线 [33]，并避免在非对称单变量数据中错误判定离群值。R 软件包 robustbase（https://cran.r-project.org/web/packages/robustbase）和 MATLAB 工具箱 LIBRA（http://wis.kuleuven.be/stat/robust.html）通过包装函数调用同一份已编译的 C 源代码，实现了快速 medcouple，以最大限度地提高速度。我们忠实地将 C 函数 mlmc.c 复现为纯 MATLAB 函数 medcouple.m，但将中位数和加权

列表 2：从 quickselectFSw.m 提炼的代码。也提供 R、C语言和 C-MEX 文件。

```
函数 [kD,          kW ,  kstar]     = quickselectFSw(D,W,p)
% 将 simpleselect 扩展到加权顺序统计量

n=length(D); 左侧 = 1; 右侧 = n; 位置=-1;
D= [D(:),W(:)];          % 数值和权重成对输入
k=ceil(n*p);             % 将枢轴索引设置为在 D(1:k,:) 中运作

BleichOverton = true;        %%% 外部循环检查 %%%
循环BleichOverton            %%% 权重条件 %%%

    while (位置~=k) % 内部循环为 quickselectFS
        枢轴          = D(k,:);                    %|
        D(k,:)        = D(右侧,:);                 %| 行交换
        D(右侧,:)     = 枢轴;                      %|
        位置          = 左侧;
        对 i=左侧:右侧
            if (D(i,1)<枢轴(1,1))
                对于 秒=1:2                        %|
                    缓冲区 = D(i,s);               %| 逐元素
                    D(i,s)    = D(位置,s);         %| 行交换
                    D(位置,s) = 缓冲区;            %|
                end                                %|
                位置 = 位置+1;
            end
        end
        D(右侧,:)=D(位置,:);
        D(位置,:)=枢轴;
        if (位置 < k)
            左侧 = 位置 + 1;
        else
            右侧 = 位置 - 1;
        end
    end

    %%% 权重检查 - 扩展 Bleich-Overton %%%
    Le=sum(D(1:k-1,2));
    若 Le-p<=0&& p-Le-D(k,2)<=0  %% 好：停止计算
        kD=D(k,1);    kW=D(k,2);    kstar=k;
        BleichOverton=false;
    else                    %% NOT OK: go back to quickselectFS
        if D(k,2)<2*(p-Le)          % 需要添加一个权重
            k=k+1; 左侧=k; 右侧=n;
        else                        % 需要移除一个权重
            k=k-1; 左侧=1; 右侧=k;
        end
    end
    位置=-1;
end
```

中位数的计算替换为调用我们编写的函数 quickselectFS 和 quickselectFSw；此外，为便于比较，还提供了由 [29] 得出的 $\mathcal{O}(n\log\ n)$ 解。为求完整，medcouple.m 还增加了选项，可计算“朴素方法” $\mathcal{O}(n^2)$ 解，以及 [13] 使用的分位数和八分位数近似值，以供比较。mlmc.c 中的一致性

	Medcouple 函数	n = 10000	n = 1000	n = 100
1	mlmc.c（完全由 C语言编译）	0.5272	0.0384	0.0114
	medcouple.m（完全由 MATLAB 解释执行），调用：			
2	- quickselectFSw（MEX）	1.4456	0.0657	0.0170
3	- quickselectFSw（即时编译）	3.5361	0.1100	0.0155
4	- 加权中位数（即时编译/sort）	14.5733	0.2823	0.0272
5	- 朴素方法（即时编译/sort）	35.1412	0.1633	0.0073

表 1：采用不同加权中位数方案计算 medcouple。使用函数 lognrnd(0,1,n,1)，从对数正态分布中抽取 100 个不同规模的随机样本，并对其计算 medcouple，所需总时间（秒）。

通过系统检查，确保 C语言与 MATLAB 中的 medcouple.m 生成的随机数一致，并采用梅森旋转算法框架。在附录 A.4 中，我们展示了实现方法，并介绍了一个在 R 中复现随机数的函数；这一点并不显而易见。

表 1 展示了各种解决方案的相对性能。我们的基准方案（第 1 行）是使用完全编译的 mlmc.c 运行，显然具有优势。不过，考虑到算法的大部分时间都用于计算加权中位数，我们预计，当 medcouple.m 使用 quickselectFSw 的 C 语言编译 MEX 版本时，性能会相当（第 2 行）。报告的结果符合预期，但请注意，通过优化 quickselectFSw 的 C 代码以提高速度，并用向量化代码替换 medcouple.m 中的循环，这种方案还可以运行得更快。最后请注意，使用标准 MATLAB 函数 quickselectFSw.m（第 3 行）会使总执行时间加倍（对于中大型 n），但仍远低于基于排序的朴素方法所需的时间（第 4 行和第 5 行）：显然有必要避免使用这些方法。

5.4 数字滤波的应用

数字滤波涵盖许多应用；这里我们以栅格图像去噪为例。最初的做法是取某个像素周围的一系列数值，并用这些数值的中位数替换该像素 [45]。这一过程会在一个窗口内重复进行，该窗口沿图像滑动。 [12] 已表明，加权中位数的效果更好，尤其适用于去除特定的结构模式。显然，在这一过程中，加权中位数会被大量使用，具体次数取决于图像分辨率。代码清单 3 显示，对于一个 3×3 权重掩膜和一张类似图6所示的 $1280 \times 1205 \times 3$ 照片（其中添加了一定比例的高斯噪声），对 quickselectFSw.m 的调用次数将为 $(row-3)\cdot(col-3)\cdot rgb = 4$，604,862。如果将第 36 行对 quickselectFSw.m 的调用替换为加权中位数，

```
36      [~ , Zclean(ind5)] = weightedMedian(A,W);
```

依赖 sort 时，执行时间增加了 4 倍以上（在我们的测试中从 16 秒增至 66 秒）。考虑到在这种情况下，加权中位数会反复在仅含 9 个值的小数组上计算，而用 quickselectFSw 替代 sort 的收益原则上非常有限，这一增幅相当显著。

6 结论

四十年前， [8] 惊讶地发现，用于计算中位数和加权中位数的已知最佳线性算法，在实际应用中所需的 CPU 时间可能比基于排序的“朴素”方法更多，即使数组包含数千个



图6：一张经过加权中位数降噪的 $1280 \times 1205 \times 3$ 像素照片，使用维纳滤波器掩模 $W = [10, 12, 9$ ； 12, 19, 12； $9, 12, 10]$。左侧面板中的细节显示了添加的噪声，这些噪声已被滤波器去除（右侧面板）。

元素。这是因为复杂算法固有的数据结构不实用，或堆、栈内存管理存在问题，导致其复杂度常数虽小却不可忽视。因此，许多统计与编程环境的主发行版本中很难找到线性选择算法，也就不足为奇了。至于加权百分位数，据我们所知，通常只提供加权中位数，而且很少有线性复杂度的解决方案。我们已经表明，即使在当今的稳健统计中，采用（加权）选择算法仍可能更为便利，而我们的简化与推广消除了固定的计算开销。

用于复现论文中结果的软件可在 https://gi thub.com/UniprJRC/FSDApapers 位于 simpleselect /ArticleReplicabilityCodes 文件夹中。我们提供核心函数 quickselectFS 和 quickselectFSw 的 MATLAB、R、C语言和 C-MEX 源代码；后者还编译为适用于 Linux、Mac OS X 和 MS-Windows 平台的二进制 MEX 文件。这些函数也包含在 FSDA 的标准发行版中，该发行版也可从 GitHub 空间 UniprJRC 下载。R 社区以及 Python 等其他开放环境的用户，可以通过 GitHub 项目 FSDA-MATLAB Coder 和 fsdaR [25] 中提供的（自动生成的）C 代码及相应的 R 包装器来运行 FSDA 工具。

图 3：使用 quickselectFSw.m 对照片进行去噪。

```
图像文件 ='boletus1280.jpg';
pnoise = 0.2;        % 添加噪声的百分比

I = imread(imfile); % 读取图像
%I = rgb2gray(I);        % 取消注释以获得黑白效果
Z = double(I);
[row , col , rgb] = size(Z);

%% 向图像添加噪声
x     = rand(size(Z));
d     = x < p噪声/2;
Z(d)  = 0;        % 设置为最小值
d     = 查找(x >= p噪声/2 & x < p噪声);
Z(d)  = 255;     % 设置为饱和值

%% 维纳滤波器的权重掩模
W = [10,12,9; 12,19,12; 9,12,10];
w = ./ sum(sum(W)); W = W(:);
w

%% 使用加权中位数去噪
%   需要调用 quickselectFSw (col-3)*(row-3)*rgb 次
Zclean = Z(:);
用于 qsw=0:1
    对于y = 2:1:(col-1)*rgb
        对于 x = 2:1:row-1
            ind1 = x-1 + (y-1 - 1).*row;
            % = sub2ind([row,col],x-1,y-1);
            ind2 = x-1 + (y     - 1).*row;
            % = sub2ind([行,col],x-1,y);
            ind3 = x-1 + (y+1- 1).*row;
            % = sub2ind([row,col],x-1,y+1);
            ind5 = x   + (y     - 1).*row;
            % = sub2ind([行,col],x,y);
            imask = [ind1; ind1+1; ind1+2; ind2; ...
                ind2+1; ind2+2; ind3; ind3+1; ind3+2];
            A = Z(imask);
            Zclean(ind5) = quickselectFSw(A,W,0.5);
        end
    end
end
Zclean = 重塑(Zclean,size(Z));
图窗;    图像显示(uint8(Zclean))
```

A 附录

A.1 simpleselect 中 `position` 的期望增量

在清单 1 中，‘`for` 循环’以 位置=左侧=1 和 右侧=n 初始化，并从外层‘`while` 循环’中接收一个枢轴值 a。随后，a 会与 A的所有元素进行比较，假定这些元素来自同一个（未知的）母分布。由于 a 取自元素 $a_k \in A$（第 8 行），因此枢轴实际上会与 $N = n - 1$ 个 $A \setminus \{a_k\}$ 元素进行比较，可能出现以下 $N + 1$ 结果：

$$a \le a_{(1)} \quad ; \quad a_{(1)} < a \le a_{(2)} \quad ; \quad \dots \quad ; \quad a_{(N-1)} < a \le a_{(N)} \quad ; \quad a > a_{(N)} \tag{8}$$

显然，当 $a_k = a_{(n)}$ 时，最后一种结果就会出现。如果我们将 A 视为由 L个独立样本组成、且每个样本都由 simpleselect 分析，那么可以采用 [28, 《第 2 讲：绘图位置》] 中提出的统计极值问题，该问题针对 N 个观测值中第 m小值的累积分布函数 $F(a_{(m)})$，表述如下：

$$P(a \le a_{(m)}) = \lim_{L\to\infty} r_L(a \le a_{(m)}) = \lim_{L\to\infty} F(a_{m:L}) = E(F(a_{(m)})) = \frac{m}{N+1} \tag{9}$$

其中，$r_L(\cdot)$ 表示基于 L 个样本计算的相对累积频数，而 $a_{m:L}$ 则表示 L 个单独的第 m 阶排序值组成的集合 $a_{(m)}$。注意，累积分布函数 (9) 是一个阶梯函数，在表示 (8) 各种结果的 $N+1$ 个区间中，每个区间都递增 $\frac{1}{N+1}$。这也意味着 $P(a = a_{(m)}) = \frac{1}{N+1}$ 独立于 m。关于 (9) 的简要证明以及相关结果的讨论，可见 [41]。

现在，考虑 A 中由满足第 18 行“`if` 语句”并使 position 增加 1 的元素构成的子集。我们可以用 $A_{lr} = \{a_i < a$ 表示该子集；$i = l$、.....、$r\}$、l和 r分别是左侧和右侧指针的索引。位置增加的预期次数就是随机变量 $X = |A_{l,r}|$的均值；下面针对 simpleselect 的第一次迭代推导该均值，其中：

$$X = |A_{1,n}| = t \quad \text{if } a = a_{(t+1)} \quad \text{for } t = 0, 1, \ldots, N$$

鉴于累积分布函数 (9) 表明 $P(X = t) = \frac{1}{N+1}$，且独立于 t，我们有：

$$E[X] = \sum_{t=0}^{N} t \cdot P(X = t) = \frac{1}{N+1}\sum_{t=1}^{N} t = \frac{N(N+1)}{2(N+1)} = \frac{N}{2} = \frac{n-1}{2} \tag{10}$$

A.2 Vervaat永恒性的实现

永恒性随机变量是一种具有以下形式的随机变量：

$$Y = W_1 + W_1 \cdot W_2 + W_1 \cdot W_2 \cdot W_3 + \ldots$$

其中，W_i 是由随机变量组成的独立同分布序列。如果每个 W_i 都具有相同的分布，记作 $W_i \sim W$，那么对于 Y，$Y \sim W(1 + Y)$，且 W 相互独立。Hoare 的 Find 的比较渐近服从一种称为 Dickman分布的特定分布，其中 $W \sim Unif([0, 1])$。遗憾的是，这种分布没有闭式表达。

Dickman分布也可以视为 Vervaat 永续随机变量的一种特殊情形，其满足：对于某个 $\beta \in (0, \infty)$，有 $W_i \sim U^{1/\beta}([0, 1])$，其中 $U \sim Unif$。换言之，Dickman分布是参数为 $\beta = 1$ 的 Vervaat 永续随机变量。永恒性随机变量的一般形式为

$$Y = \sum_{n=0}^{\infty} A_n \prod_{i}^{n} W_i$$

其中 A_n 不一定等于 1，这种分布称为 Takacs分布。我们对 Vervaat族的实现（vervaatxdf.m）（清单 4）遵循了 [5], 提出的方法。该方法以可行且优雅的方式计算概率密度和分布函数，无需进行蛮力模拟（作者提供的代码使用 Wolfram Mathematica 编写）。作为比较，我们还将 [14], 的递归模拟方法从 R 移植到了 MATLAB。该方法准确，但速度慢得多（vervaatsim.m）。为求完整，我们实现了一个函数，用上述两种方法之一模拟来自 Vervaat 分布的随机变体（vervaatrnd.m）。

清单 4：从 FSDA 工具箱提炼出的代码。遵循 Wolfram Mathematica 代码，该代码由 [5]慷慨提供。用户文档：http://rosa.unipr.it/FSDA/vervaatsim.html。

```
函数 [f, F , x] = vervaatxdf(betav,nx,pascalM)
%Vervaat 永续随机变量的概率密度函数和累积分布函数。
% betav      : 分布参数值。
% nx         : 求值点数量。
% pascalM: 预先计算的帕斯卡矩阵，用于加速.....
      模拟中的相关操作。备注：bc(a,b) =                    ...
      pascalM(a-b+1,b+1).

欧拉–马歇罗尼常数val = 0.57721566490153286060651209008240243 1;
n   = 100;
如果输入参数个数==0, betav = 1; 结束; % 默认值为 Dickman分布
如果 nargin<2, nx = 1; 结束;
如果 nargin<3, pascalM = pascal(n+1); 结束;

h   = zeros(1,n);
L   = zeros(1,n+1);
ex= 5*betav;
x   = ex.*rand(1,nx);% 来自 U 的 [0,ex]
f   = zeros(1,nx);
F   = zeros(1,nx);

对于 k=1:nx
      s=n/x(k);
      h(1) = betav * (exp(-s)       - 1)/s;
      对于 j=2:n
            h(j) = (betav   / s) *  (-1)^(j-1)      * exp(-s)    - ...
                  ((j-1)/s      * h(j-1))    ;
      end
      L(1)   = exp(betav * (- 欧拉–马歇罗尼常数 - ...
            gammainc(0,s)      - log(s) ));
      对于 i=2:n+1
            Li   = 0;
            对于 j=0:i-2
                  Li = Li  + pascalM(i-2-j+1,j+1) * h(i-1-j)              ...
                        * L(j+1);
                %Li  = Li  +bc(i-2,j)       * h(i-1-j)       * L(j+1);
            end
            L(i) = Li;
      end
      % pdf
      f(k)   = ( (-1)^n     / 阶乘(n) )          * s^(n+1)    * L(n+1);
      % 累积分布函数
      F(k) = sum((-s).^(0:n)       ./ 阶乘(0:n)          .* ...
            L((0:n)+1));
end
```

A.3 从 Python 调用 `quickselectFS.c` 和 `quickselectFSw.c`

所有统计编程环境都可以集成 C 函数，以消除性能瓶颈或计算特定算法。典型做法是在所用环境中编写一段代码（包装器），将原始数据类型映射为 C 代码中的数据类型，初始化函数参数，并获取调用编译后的 C 函数所返回的结果。代码清单 5 展示了如何从 Python 调用 quickselectFS.c。调用 quickselectFSw.c 的方法类似。

清单 5：从 Python 调用 quickselectFS.c 的包装器。

```
# 此 Python 脚本展示了一种简便的调用方式
# 从 Python 调用 quickselectFS.c，并传入一个 NumPy 数组
# 作为第一个参数。
#
# .c 源文件应按如下方式编译：
#
# gcc -fPIC -shared -o simpleselect.so simpleselect.c
#
# C 兼容数据类型由以下内容构建：
# 外部 Python 函数库 ctypes。

导入 ctypes
从 numpy.ctypeslib导入 ndpointer
导入numpy as np

库 = ctypes.cdll.LoadLibrary("simpleselect.so")

quickselectFS = lib.quickselectFS

quickselectFS.restype = ctypes.c 双精度浮点数
quickselectFS.argtypes = [ndpointer(ctypes.c 双精度浮点数, ...
    标志="C 连续"),
                          ctypes.c int,
                          ctypes.c int]

A = np.arange(-5.0,  6.0, 1, 数据类型 = 浮点数)
打印(A)
np.random.shuffle(A)

# quickselectFS 采用原地操作，因此我们必须创建.....
    向量 A 的副本，如果我们希望保留.....
    原始 A。

辅助 = A.copy()

k = 7
第 k 个顺序统计量 = quickselectFS(aux,  长度(辅助), k)

打印(第 k 个顺序统计量)

删除 辅助
```

A.4 在 MATLAB、C语言和 R 中生成相同的随机数

原则上，只需采用相同的生成方法，就能在不同语言中生成相同的随机数。梅森旋转算法是一个著名的例子，由 [43], 提出，可用于多种语言，相关资料见 http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/emt.ht

代码清单 6 展示了如何在 C语言中使用给定的种子创建一个包含 D 个均匀随机数的数组 $[0, 1)$，并对其应用 quickselectFS。要在 MATLAB 中使用相同的种子重现这些随机数非常简单，因为 MATLAB 自 2005 年（R14SP3）起便内置支持梅森旋转算法 mt19937ar。只需运行：

```
RandStream.setGlobalStream(RandStream.create('mt19937ar','种子',
896));D = rand(n,1);
```

或者等价地

```
myseed = 896;  rng(myseed , 'twister');
```

```
D = rand(n,1);
```

请注意，以下指令会生成相同的 1 到 N 之间的 n 整数

```
D = ceil(rand(n,1)*N);                    D = randi(N,n,1);
```

也就是说，我们可以轻松地从均匀分布的浮点数得到均匀分布的整数。使用 Python 时，方法类似，因为它以梅森旋转算法作为核心生成器，并使用相同的底层 C 库。

相比之下，要在 R 中复现相同的随机数要困难得多，因为 R 中的播种算法并未完全遵循 mt19937ar，而是采用了不同的初始化和输出变换。据我们所知，这使得无法将 R 的种子值直接映射到 MATLAB/C 的种子值。R 软件包 randtoolbox 提供了一个选项，可用于生成遵循 mt19937ar 的随机数：

```
set.generator("梅森旋转算法", 初始化="init2002", ...分辨率=53, 种子=myseed)
```

如果按照手册，在 R（V 4.0.5）中使用 myseed=12345运行，我们会得到

```
> runif(5)
[1] 0.1839188 0.2045603 0.5955447 0.6531771 0.2987037
```

遗憾的是，手册中给出的序列不同；实际上，我们在 MATLAB 中通过以下方式得到了该序列：

```
>>rand('twister', myseed); rand(1, 5)结果 =0.9296
0.3164    0.1839    0.20460.5677
```

因此，randtoolbox 的行为似乎并不稳定。

清单 6：使用梅森旋转算法生成相同的随机数：C语言端。

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>
#include "simpleselect.h"          // quickselectFS/quickselectFSw
#include "mersenne.h"              // mt19937ar , 梅森旋转算法
                                   // 改进的初始化
int main() {
    int n, k;
    printf("\n 请输入样本量 n:                    ");
    scanf("%d",&n);
    printf("\n 请输入顺序统计量 k (<=n):                    ");
    scanf("%d",&k);
    双精度浮点数 D [n];

    32位无符号整数 t 种子 = 896;          // 设置种子
    初始化 genrand(seed);              // 使用种子初始化梅森旋转算法 [n]
    对于 (inti=0; i<n;  ++i) {         // 在 [0,1) 中生成随机数
      D [i] = genrand res53();         // 具有 53位分辨率
    }

    double wE =quickselectFS(D,n,k-1); // 应用 quickselectFS
    printf("\n 结果: %f\n",     wE);
    返回 0;
}
```

列表 7：使用梅森旋转算法生成相同的随机数：MATLAB 端。mtR.m 使用的子函数，用于根据给定种子初始化 R 状态。它们模拟了梅森旋转算法的 C 代码。

```
函数 状态 = initMTR(seed)
% 根据种子创建初始的 R 梅森旋转算法状态。
% 参见 RNG.c 中的 RNG Init()。
n = double(seed);% 以防输入的是 int32
状态 = 零(626,1,'32 位整数');
用于i = 1:50,n = int32lcg(n);结束
对于i = 2:626
    n = int32lcg(n);
    state(i) = n;
end
state(1) = 403;
state(2) = 624;
end

函数n = int32lcg(n)
% 模拟 R 初始化梅森旋转算法时使用的旧版 glibc 线性同余发生器，
% 使用 32 位整数运算（MATLAB 的 32 位整数会饱和，C语言中的则会回绕）。
% 参见 RNG.Init() （位于 RNG.c 中）。
n = 69069 * n + 1;
如果 n >= 2^31 || n < -2^31
    n = mod(n+2^31,2^32) - 2^31;
end
end
```

因此，我们在 FSDA 中集成了一个新函数 mtR.m，它可以生成与基础 R 使用 mt19937ar 所产生的相同均匀分布或正态分布随机数。由于无法将 R 的种子映射为 MATLAB 的种子，mtR.m 的起始输入是一个包含 R 用于生成随机数的随机数生成器状态的 626 元素 int32 向量；我们可以通过执行以下命令获取该向量：

```
RNGkind("梅森旋转算法") # 设置“梅森旋转算法” “反演”设置.seed(myseed) R 状态
```

当然，与其在 R 中生成一个含 626 个元素的向量，再将其传递给 MATLAB 中的 mtR.m，不如直接在 MATLAB 内部构建与有效的 R 种子相对应的 R 状态向量，这样更方便。幸运的是，按照梅森旋转算法的 C 代码即可实现这一点；我们已在 mtR.m 中以列表 7 的形式介绍了该代码。

此时，mtR.m 可以通过删除代表随机数生成算法的第一个代码、以标准 MATLAB 格式重新排列状态，并将向量从 R 使用的有符号整数转换为 MATLAB 使用的无符号对应值，从而处理生成的 R 状态；简言之：

```
Rstate = initMTR(myseed);MATLABstate = [Rstate(3:end);
Rstate(2)];MATLABstate = typecast(32 位整数(MATLABstate),'32位无符号
整数');
```

现在，我们可以获取 MATLAB 当前的全局随机数流，并将其状态设为转换后供 R 使用的形式（如果方便，也可以直接从 R 接收）：

```
Mstream = RandStream.getGlobalStream();
```

```
Mstream.State = MATLABstate;
```

最后一个需要记住的诀窍是，使用逆变换来计算正态随机变量（即将标准正态逆累积分布函数应用于均匀随机变量）。这取代了 MATLAB 的默认方法，即 ziggurat算法 [42]。

mtR.m函数包含许多 MATLAB/R 示例，可用于涉及这两种语言的仿真实验。

参考文献

[1] J. Astola 和 P. Kuosmanen。《 *Fundamentals of Nonlinear Digital Filtering*》。CRC出版社，第1版，1997年。doi: 10.1201/9781003067832。[2] A. C. Atkinson 和 M. Riani。《 *Robust Diagnostic Regression Analysis*》。施普林格出版社，纽约，2000年。doi: 10.1007/978-1-4612-1160-0。[3] A. C. Atkinson、M. Riani 和 A. Cerioli。《 *Exploring Multivariate Data with the Forward Search* 2004 10 1007 》。施普林格出版社，纽约，。doi: 。/ 978-0-387-21840-3。[4] I. Azzini、T. Mara 和 R. Rosati。用于回归问题的全新提升算法（bOOstd）及其在敏感性分析中的应用。第十届模型输出敏感性分析国际会议，2022年3月。[5] Lucio Barabesi 和 Luca Pratelli。关于 tak´acs 分布的性质。《*Statistics & Probability Letters*》，148:66–73，2019年。ISSN 0167-7152。doi: 10.1016/ j.spl.2019.01.005。[6] I. Barrodale 和 F. D. K. Roberts。离散线性逼近的改进算法。《 *SIAM Journal on Numerical Analysis*》，10(5):839–848，1973年。[7] Alberto Bertoni、Paola Campadelli 和 M. Parodi。用于回归的提升算法。载于 *Proceedings of the 7th International Conference on Artificial Neural Networks* 97, 343 348, 1997 ，

ICANN ' 页码 – 柏林，海德堡，。施普林格出版社。ISBN 3540636315。doi: 10.1007/bfb0020178。[8] Chaya Bleich 和 Michael L. Overton。加权中位数问题的线性时间算法。技术报告75，纽约大学库朗数学科学研究所，1983年4月。[9] Peter Bloomfield 和 William Steiger。最小绝对偏差曲线拟合。《 *SIAM Journal on Scientific and Statistical Computing*》，1(2):290–301，1980年。doi: 10.1137/0901019。[10] Manuel Blum、Robert W. Floyd、Vaughan Pratt、Ronald L. Rivest 和 Robert E. Tarjan。选择问题的时间界。《 *Journal of Computer and System Sciences*》，7 (4):448 – 461，1973年。ISSN 0022-0000。doi: 10.1016/s0022-0000(73)80033-9。[11] Jack Bowden、George Davey、Philip Haycock 和 Stephen Burgess。使用加权中位数估计量，在存在一些无效工具变量时对孟德尔随机化进行一致估计。《 *Genetic Epidemiology*》，40(4):304–314，2016年。doi: 10.1002/gepi.21965。[12] D.R.K. Brownrigg。加权中值滤波器。《 *Communications of the ACM*》，(27):807–818，1984年。doi: 10.1145/358198.358222。[13] G Brys、M Hubert 和 A Struyf。一种稳健偏度度量。*Journal of* 《*Computational and Graphical Statistics* 13 4 996 1017, 2004 10 1198 》，

( ): – 。doi: 。/ 106186004X12632。[14] Kirkwood Cloud 和 Mark Huber。Vervaat 永恒性的快速完美模拟。《*Journal of Complexity*》，42:19–30，2017年。doi: 10.1016/j.jco.2017.03.005。[15] Thomas H. Cormen、Charles E. Leiserson、Ronald L. Rivest 和 Clifford Stein。《*Introduction to Algorithms*》， *Third Edition*。麻省理工学院出版社，第3版，2009年。

[16] P.L Davies、R. Fried 和 U. Gather。用于在线监测数据的稳健信号提取。 *Journal of Statistical Planning and Inference*，122(1):65 – 78，2004。doi: 10.1016/j.jspi.2003.06.012。[17] Dorit Dor 和 Uri Zwick。选择中位数。载于 *Proceedings of the SixthAnnual ACM-SIAM Symposium on Discrete Algorithms*，SODA '95，第 28–37 页，美国费城，1995 年。工业与应用数学学会。ISBN 0-89871-349-8。doi: 10.1137/s0097539795288611。[18] R. Geoff Dromey。选择问题的一种算法。 *Software: Practice and Experience* 16 11 981 986, 1986 10 1002 4380161103 , ( ): –。 doi: . /spe. .[19] F.Y. Edgeworth。XXII. 关于一种约简涉及多个量的观测值的新方法。 *The London Edinburgh and Dublin Philosophical Magazine* , , , ( ): –。 *and Journal of Science* 25 154 184 191, 1888 [20] W. R. Farebrother。概率论与统计学史研究 XLII：关于 Boscovich 与 Simpson 在 1760 年 6 月交往的更多细节。 *Biometrika*，77 2 397 400，06 1990 10 1093 77 2 397 ( ): –。 doi: . /biomet/ . . .[21] Sir Fisher, Ronald Aylmer 和 Frank Yates。 *Statistical tables for biological agri-* , . 伦敦: *cultural and medical research* 3 1948 Oliver and Boyd，第三版，. doi: 10.2307/1905265。[22] Y. Freund 和 R.E. Schapire。在线学习的决策理论推广及其在提升中的应用。 *Journal of Computer and System Sciences*，55 1 119 0022 0000 10 1006 1997 139，1997 1504 ( ): –。ISSN -。doi: . /jcss. . .[23] Roland Fried、Jochen Einbeck 和 Ursula Gather。加权重复中位数平滑与滤波。 *Journal of the American Statistical Association*，102 (480):1300–1308，2007。ISSN 01621459。doi: 10.1198/016214507000001166。[24] Jerome Friedman、Trevor Hastie 和 Robert Tibshirani。加性逻辑回归：提升的统计学视角（附讨论及作者回应）。 *The Annals of Statistics*，28(2):337–407，2000。doi: 10.1214/aos/ 1016218223。[25] FSDA。用于 MATLAB 的灵活统计与数据分析工具箱，并扩展支持 R 和 SAS。GitHub: https://github.com/UniprJRC/FSDA；MATLAB 中央文件交换: https://www.mathworks.com/matlabcentral/fileexchange/72999-fsda；文档: http://rosa.unipr.it/FSDA/guide.html，2005-2021。[26] R. Giuliano、Z.S. Szewczak 和 M.J.G Weber。Dickman分布的几乎必然局部极限定理。 *Periodica Mathematica Hungarica* 76 155 197, , : –。 2018 10 1007 10998 017 0193 0 doi: . /s - - -。[27] Larry Goldstein。关于快速选择算法运行时间的 Dickman 近似的非渐近分布界。 *Electronic Journal of Probability* 23 1 13, 2018 10 1214 18 227 , : –。 doi: . / -EJP 。[28] E.J. Gumbel。 *Statistical Theory of Extreme Values and Some Practical Applications: A Series of Lectures* 。应用数学丛书。美国政府印刷局，1954 年。[29]Sven Haase。加权中位数。MATLAB 中央文件交换，2022年12月。URL https://www.mathworks.com/matlabcentral/fileexchange/23077-weighted-me

点。

[30] C. A. R. Hoare。算法 64：快速排序。 *Communications of the ACM*，4(7): 321，1961。doi: 10.1145/366622.366644。[31] C. A. R. Hoare。算法 65 *Communications of the ACM* 4 7 321 322, : 查 1961 10 1145 366622 366647 找。,( ): –。doi: . / . .[32] C. A. R. Hoare。程序证明：查找。 *Communications of the ACM*，14(1): 39–45，1971年1月。doi: 10.1007/978-1-4612-6315-9 10。[33] M. Hubert 和 E. Vandervieren。偏态分布的调整箱线图。*Computational Statistics & Data Analysis*，52(12):5186–5201，2008。ISSN 0167- 9473。doi: 10.1016/j.csda.2007.11.008。[34] H.K. Hwang 和 T.H. Tsai。quickselect 与狄克曼函数。 *Combinatorics Probability and Computing* 11 4 353 371, 2002 10 1017 , , ( ): –。doi: . / S0963548302005138。[35] Bal´azs K´egl。通过提升稳健回归中位数。载于 Bernhard Sch¨olkopf 和 Manfred K. Warmuth 编，*Learning Theory and Kernel Machines*，第 258–272 页，柏林、海德堡，2003 年。Springer Berlin Heidelberg。ISBN 978-3-540- 45167-9。doi: 10.1007/978-3-540-45167-9 20。[36] Bal´azs K´egl。中位数提升的泛化误差与算法收敛。载于 *Advances in Neural Information Processing Systems 17*，第 657–664 页，加拿大温哥华，2004 年 12 月 13–18 日。[37] Donald E. Knuth。 *Seminumerical Algorithms* 2 *The Art of Computer* , 第 *Programming* 1981 卷, 《》。Addison-Wesley，马萨诸塞州雷丁，第二版，。[38] P. Li。第 n 个元素。MATLAB 中央文件交换，2013年11月。https://www.mathworks.com/matlabcentral/fileexchange/29453-nth_element。[39] Hosam M. Mahmoud。quickselect 中交换操作的分布分析。 *Theoretical Computer Science* 411 16 1763 1769, 2010 0304 3975 10 1016 , ( ): –。ISSN -。doi: . /j.tcs. 2010.01.029。[40] Hosam M. Mahmoud、Reza Modarres 和 Robert T. Smythe。quickselect 分析：一种用于顺序统计量的算法。 *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications* 29 4 255 276, 1995 , ( ): –。doi: 10.1051/ita/1995290402551。[41] Lasse Makkonen。为绘图位置争议画上句号。 *Communications in Statistics: Theory and Methods* 37 3 460 467, 2008 , ( ): –。doi: 10.1080/03610920701653094。[42] George Marsaglia 和 Wai Wan Tsang。生成随机变量的 Ziggurat 方法。 *Journal of Statistical Software*，5(8):1–7，2000。doi: 10.18637/ jss.v005.i08。[43] Makoto Matsumoto 和 Takuji Nishimura。梅森旋转算法：一个 623 维均匀分布的均匀伪随机数生成器。 *ACM Trans. Model. Comput. Simul.* 8 1 3 30, 1998 1049 3301 , ( ): – jan。ISSN -。doi: 10.1145/272991.272995。[44] David R. Musser。内省排序与选择算法。*Software: Practice and Experience* 27 8 983 993, 1997 , ( ): – August。[45] W.K. Pratt。 *Digital Image Processing*。John Wiley and Sons，1978 年。

[46] Press、Teukolsky、Vetterling 和 Flannery。 *Numerical recipes in C*。剑桥大学出版社，第二版，1992年。ISBN 0-521-43108-5。[47] Marco Riani、Domenico Perrotta 和 Francesca Torti。FSDA：用于稳健分析和交互式数据探索的 MATLAB 工具箱。 *Chemometrics and Intelligent Laboratory Systems* 116 17 32, 2012 10 1016 ,

(Supplement C)：–。doi：。/j.chemolab. 2012.03.017。[48] Marco Riani、Domenico Perrotta 和 Andrea Cerioli。针对超大型数据集的前向搜索。 *Journal of Statistical Software Code Snippets* 67 1 1 20, , , 2015 10 18637 067 01 ()：–。doi：。/jss.v .c。[49] Peter Rousseeuw。具有高崩溃点的多元估计。载于 *Mathematical Statistics and Applications Vol. B*，第 283–297 页，1985年1月。doi：10.1007/978-94-009-5438-0 20。[50] Peter Rousseeuw 和 Katrien Driessen。最小协方差行列式估计量的快速算法。*Technometrics*，41：212–223，1999年8月。doi：10.1080/00401706.1999.10485670。[51] Peter Rousseeuw 和 Mia Hubert。回归深度。 *Journal of the American Statistical Association* 94 446 388 402, 1999 01621459 10 1002 , ()：–。

ISSN。doi：。/ 0471667196.ess0719。[52] Peter J. Rousseeuw。最小中位数平方回归。 *Journal of the American Statistical Association* 79 388 871 880, 1984 10 1080 01621459 1984 , ()：–。

doi：。/ . . 10477105。[53] Robert Sedgewick 和 Kevin Wayne。 *Algorithms*。Addison-Wesley Professional，第四版，2011年。ISBN 978-0321573513。doi：10.1007/978-1-4842-3829-5 7。[54] S. M. Stigler。概率论与统计学史研究 XL。Boscovich、Simpson 与一篇关于拟合线性关系的1760年手稿注释。 *Biometrika*，71 (3)：615–620，1984年。doi：10.1093/biomet/71.3.615。[55] Ryan J. Tibshirāni。通过连续分箱快速计算中位数，2008年6月。未发表手稿；可作为 arXiv 预印本 arXiv:0806.3301 获取。[56] Lin Yin、Ruikang Yang、M. Gabbouj 和 Y. Neuvo。加权中值滤波器：教程。 *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 43 3 157 192, 1996 10 1109 82 486465 , ()：–。

doi：。/ . .