统计计算

李东风

2022-08-29

目录

前言

统计学是研究如何有效地搜集、整理和分析带有随机性的数据，以及对所观察的问题作出推断或预测，直至为采取一定的决策和行动提供依据和建议的学科。统计学用于实际数据的建模和分析，需要把统计建模过程变成适当的计算机算法，并编程实现。这样，学习统计学专业的学生，不仅需要学习高等数学、概率论、数理统计、回归分析、多元统计分析、时间序列分析、生存分析等许多数学和概率统计专业课程，还需要了解统计学有关的计算机算法的理论和实现细节，这样才能了解统计建模计算中的各种问题，不至于滥用、错用统计方法，在自己研究新的统计模型以及进行数据分析建模时才能开发出正确、可靠、高效的算法和程序。

本书包含了传统的统计计算基本概念与算法，如误差分析、矩阵计算、最优化、随机数生成、随机模拟等。在统计学相关的算法中，有很多算法可以归入所谓“计算统计”范畴，这些算法主要是借助于现代计算机的强大性能进行统计推断，比如自助法、MCMC 方法、序贯重要抽样方法、置换检验等。本书对这类方法也进行了部分论述。

本书使用 R 语言和 Julia 语言作为算法的实现和描述语言。选用 R 的原因是 R 特别适用于统计计算方法的开发，统计学家普遍地使用 R 作为统计计算和统计算法的开发语言，而且 R 也很容易自学。Julia 语言则是和 R 一样易学易用的科学计算语言，同时又克服了 R 在执行迭代计算时速度慢的缺点。教材的附录给出了简短的 R 编程入门和 Julia 语言入门讲解，这一部分在授课时可以让学生自学。本书作为教材使用时，教师也可以选用其它的计算机语言，如 Python、Matlab、C++、JAVA、C、FORTRAN 等。

本书包含大量的例题和习题，其中大部分需要编程实现。读者应该多动手实现，并与标准的算法程序进行比较。R 软件和 Julia 语言中包含了许多种标准的算

法，可以用来检查读者的编程计算结果是否正确。

本书可以作为高校统计学本科专业统计计算课程的教材，也可以作为统计学以及其它专业的本科生、研究生和研究人员关于统计计算算法的参考书。在作为本科生教材使用时，由于课时的限制需要教师适当选择讲授。建议标星号 “(*)” 的部分可以略去。

书中使用了其它教材的一些例子和习题，没有一一指明出处，在此感谢原作者。

本书已经在 2016 年由高等教育出版社出版，但是统计计算是不断发展的学科，作者感到有必要随时更新书的内容，尤其是计算实例、详细的计算程序，新的重要算法，这样的更新用网页形式更为及时，也能接触到更多的读者。欢迎提出宝贵意见。

Part I

统计计算导引

Chapter 1

统计计算介绍

1.1 统计计算的范畴

统计计算是现代统计的重要组成部分。从上个世纪三、四十年代起，数理统计的理论和方法得到跨越式的发展，统计推断理论、回归分析、试验设计、方差分析、序贯分析、时间序列分析、随机过程等理论和方法在这个时期逐渐成熟。但是，直到上个世纪八十年代，统计学作为一门学科才真正得到了广泛的普及，其应用深入到了我们的学术研究和社会生活的每一个方面，只要需要分析数据的地方就需要用到统计学。这种普及很大程度上要归功于电子信息技术的高速发展。

统计计算就是统计方法和实际计算的结合。统计计算有以下两个方面的内容：

- 把统计方法变成可靠、高效的算法，并编程实现。这是经典的统计计算要解决的问题，比如计算分布函数值、分位数函数值、计算线性回归参数估计和检验、求解最大似然估计等。
- 借助于现代计算机的强大处理能力，发展新的统计方法。这是计算技术对统计学的贡献，比如用随机模拟方法求解贝叶斯模型、Bootstrap 置信区间，等等。这个方面有时被称为 “计算统计”(computational statistics)。

第二个方面的内容包括计算密集的统计方法及相应的理论工作。**随机模拟**方法是其中一个重要内容。随机模拟的基本思想是在计算机上模拟生成一个统计问

题的数据并进行大量的重复，这样相当于获得了此问题的海量的样本。如果我们的目的是评估某种建模方法，我们可以对每个样本建模，最后从所有建模结果评估这种方法的性能。可以从理论模型产生海量模拟样本后对此模型进行理论推断，如蒙特卡洛检验。可以对观测数据进行多次重复再抽样生成许多新样本，如 Bootstrap 方法。在贝叶斯统计框架下，我们可以从先验分布抽样并按照模型产生大量样本并结合观测数据计算其似然，从而获得参数后验分布的大量样本，以此进行贝叶斯推断。借助于随机模拟，我们可以试验各种各样的模型与方法，发现表现优秀的模型和方法后再进行深入研究。

另外，各行各业中数据收集越来越广泛，在迅速积累的海量数据中包含了许多以前无法触摸的现象和规律，对海量数据进行探索性分析，从海量数据中发现规律，已经成为统计学和信息科学的热门研究方法，通常称为机器学习、数据挖掘等。这也是统计计算第二个方面的重要内容。

现在已经有了许多专用的统计软件，比如 R、SAS 等，为我们平常遇到的许多问题提供了现成的解决办法，那么，我们为什么还需要学习统计计算呢？

我们遇到的具体应用问题常常是没有现成的方法可以套用的。即使有现成的统计软件可用，我们也需要理解这些软件的工作原理以避免错误使用；在遇到新问题时，需要能够修改原有代码或编写新代码，把计算工具结合在一起解决自己的数据分析问题，而不是修改自己的问题以适应现成的软件。

1.2 算法和计算机语言

算法是完成某项任务的步骤的精确的描述。比如，泡方便面的一种算法为：

1. 准备方便面、容量 500 毫升的碗、300 毫升开水;
2. 将方便面包装撕开，放入碗中；
3. 将调料包撕开，放入碗中；
4. 向碗中倒入开水；
5. 等待 5 分钟。

当然，算法主要针对在电子计算机上的计算而设计。好的算法应该满足如下要求：

- 结果正确，即要求算法的最后结果是我们问题的正确解，最好能够验证结果的正确性。
- 指令可行，即指令含义明确无歧义，指令可以执行并且在现有的计算条件下算法能在允许的时间计算结束。
- 高效，尽可能少地消耗时间和内存、外存资源。

电子计算机由 CPU、内存、大容量外存、输入/输出装置等硬件构成，但是依赖于软件完成任务。软件在执行时是一系列机器指令进行取值、存储、加法等操作。操作系统是最基本的计算机软件，用来管理内存位置、软件指令、输入输出和其他程序的运行调度。

计算机软件可以用于完成特定任务，如字处理、记账，也可以适用于较宽的范围，比如电子表格软件可以用来记账、试算、作图，而**计算机语言**则是用来编制新的软件的工具。

计算机语言根据其运行方式可以分为解释型和编译型两种，解释型语言逐句解释程序并逐句执行，编译型语言把整个程序编译为二进制代码后再执行。按照计算机语言的抽象程度区分，包括二进制形式的机器语言，仅能用在特定的硬件中；汇编语言，用与 CPU 指令对应的命令编写，主要用于底层功能；面向细节的通用语言，如 Pascal, C, Fortran, Lisp, Cobol, C++, Java 等，优点是通用性和可复用性。

更高级的计算机语言有 R、Matlab 等，提供了包括向量、矩阵等高级数据类型，代码与统计学的数学公式相似，直接支持求和、向量、矩阵等运算，易写易读，用户不用自己实现如解线性方程、求特征根这样的基础操作, 但是这样的语言一般是解释执行的，执行效率难以改进，不利于使用循环或迭代算法。本书使用 R 和 Julia 作为配套的编程语言。

R 软件是一个统计计算软件, 同时也是一种计算机编程语言，与 S 语言基本兼容。S 语言是 Rick Becker, John Chambers 等人在贝尔实验室开发的一个进行数据分析和交互作图的计算机语言，可以使用向量、矩阵、对象等数据进行编程，功能强大，程序简单。R 是 GPL 授权的自由软件，最初由新西兰 Auckland 大学的 Ross Ihaka 和 Robert Gentleman 于 1997 年发布，现在由 R 核心团队开发，全世界的用户贡献了上万个软件包，功能涵盖了经典和现代统计方法的绝大部分，是世界上许多顶尖的统计学家进行统计研究和发表算法的工具。见 R 的网站: http://www.r-project.org/。我们将讲授 R 的基本使用，在算法示

例和习题中使用 R 作为编程语言，并在讲到具体统计计算方法时提及 R 中有关的函数。

Julia 程序语言也是一种计算机编程语言，就像 C、C++、Fortran、Java、R、Python、Matlab 等程序语言一样。Julia 语言历史比较短，发布于 2012 年，是 MIT 的几位作者和全世界的参与者共同制作的。主网站在https://julialang.org/。Julia 与 R、Python 等一样是动态类型语言，程序与 R、Python 一样简单易用；同时，Julia 语言支持实时编译，使得 Julia 程序的效率基本达到和 C、Fortran 等强类型语言同样的高效率，尤其适用于数值计算，在现今的大数据应用中也是特别合适的语言，在大数据应用中排在 Python、R 之后，已经获得广泛的关注，现在用户较少只是因为历史还太短。本书将混合使用 R 语言和 Julia 语言。

本书目的主要是要求学生掌握统计计算方法、理解统计计算思想，但是这些算法的正确、高效实现也是很重要的。我们设置了足够的习题让学生通过实际编程了解算法实现中的各种问题。

1.3 优秀程序设计要旨

在进行程序设计时，应注意以下几点：

- 要对程序抱有怀疑态度。逐步验证每一个模块。
- 大的任务要分解为小的模块。对每个模块进行详细测试。象 R、Julia、Python、MATLAB 这样的更高级语言的优点是许多模块已经由系统本身提供了，如矩阵计算、函数优化等。但即使这样也应进行测试，因为系统提供的功能有可能不适用于你的特殊情况。
- 编写一系列测试问题，涵盖应用的不同情况。最开始用最简单的测试。测试也应该包括超出范围的问题，这时程序应该能正确地判别错误输入。
- 好的程序不应该是用了很多高明的技巧以至于别人很难看懂，而是越直接越好，这样只要出错误就是明显的错误。
- 要有程序文档。
- 只要对程序的每一模块都进行了详细验证和检查，就可以确信程序的正确性。

1.4 内容提要

本书包括基本的统计计算方法，如计算分布函数、分位数函数的一般方法，矩阵计算方法、最优化方法、随机数生成算法。另外还用较大篇幅讲述了随机模拟方法，包括随机模拟的基本思想、改进精度的方法、重要应用。最后，还介绍了完全由计算方法发展出来的统计方法，如 Bootstrap、EM 算法、MCMC 方法等。

第一部分包括本章，以及第 2-4 章，第 2 章讲解误差的来源和分类以及避免和减少误差的方法，这对我们了解算法的局限并在算法实现时避免产生有缺陷的算法有重要意义。第 3 章讲描述统计量计算，第 4 章讲简单的统计图形用法。

第二部分包括第 5-9 章，讲随机数产生与检验，内容有均匀随机数的产生方法和检验方法，非均匀随机数的各种生成方法，包括函数变换、舍选抽样、重要抽样等。关于随机向量和随机过程的产生方法也进行了简单介绍。

第三部分包括第 10-20 章，讲随机模拟方法。第 10-14 章用随机模拟积分作为例子，讲解随机模拟方法的基本思想，包括减小随机模拟误差的技术。第 15 章介绍了离散随机事件模拟中随机服务系统模拟问题。随机模拟方法对新统计方法的研究比较也具有广泛应用，第 16 章用一个例子演示了随机模拟在统计方法研究中的用法。第 17 章讲述了 Bootstrap 方法，这是完全利用随机模拟方法解决统计推断问题的一个具体示例。第 18 章讲置换检验，此方法基于对称性的思想。MCMC 是现代统计计算的重要工具，尤其是在贝叶斯建模中起到关键作用，第 19 章讲解了 MCMC 的部分理论基础和实际做法。序贯重要抽样方法也是现代统计计算中一类重要方法，第 20 章做了介绍。

第四部分包括第 21-26 章，针对分布函数和分位数函数等近似计算问题，介绍函数逼近的多项式方法、连分数表示，插值方法，样条函数，数值积分和数值微分。

在回归分析等线性模型、多元模型、函数数据分析的计算中广泛用到矩阵计算方法。第五部分包括第 27-34 章，介绍统计计算中常用的矩阵方法，比如矩阵三角分解、正交三角分解、特征值分解、奇异值分解，广义特征值，广义逆等。

很多统计计算问题都会归结为一个最优化问题，即求解函数的无约束或有约束的最小值（最大值）点的问题，比如最大似然估计、非线性回归等。第六部分包括第 35-39 章，首先给出最优化问题的一些理论基础，然后讨论无约束最优化

的方法，再给出约束最优化的一些算法，并讨论了统计计算中的一些特定的优化问题，如最大似然估计、非参数回归、EM 算法。

附录 A 介绍了 R 语言的基础，由于篇幅所限仅包括数据类型、程序结构、函数的基础介绍，进一步的用法还要参考其他教材和 R 用户手册。附录 B 介绍了 Julia 语言的基本使用。附录 C 介绍了自由软件的计算机代数系统 Maxima 的基本用法。

本书的每一章后面附有习题，有一些是对教材中理论的进一步延伸讨论，有一些是程序编制问题。读者可以选作这些习题，加深对教材内容的理解，并获得实际编程锻炼。

本书是在已经出版的教材的基础上逐步更新，添加新内容而逐渐形成的，内容随时会有增删。

参考：Gentle [2002], Gentle [2009], 高惠璇 [1995], Kochenderfer 。

Chapter 2

误差

2.1 误差的种类

统计计算的算法要得到正确的结果，就需要尽可能减少误差。统计问题中的误差有模型误差、实验误差和数值计算误差，在统计计算研究中主要解决的是如何减少数值计算误差的问题。

统计计算的算法通常是用来求解某种统计模型。任何用来解决实际问题的数学模型都或多或少地简化了实际问题，忽略掉一些细节，从而**模型误差**不可避免。如果模型不合适，其它误差控制得再完美，问题也不能得到解决；更糟的是，良好的计算结果会给使用者以错误的信心。比如，我们使用的回归模型要求观测是独立的，而实际数据观测有不可忽略的序列相关性，尽管我们用软件算出了很完美的结果，这个结果也是错误的。我们应当仔细选择模型，尽可能减少模型误差。

建立统计模型所需的数据来自实验、观测、抽样调查等过程，在这样的过程中会出现**实验误差**，包括随机误差、系统误差、过失误差。

随机误差是试验过程中由一系列随机因素引起的不易控制的误差，可以通过多次重复试验或改进模型设计来减小随机误差。随机误差可能来自物理量本身的波动，比如测量风速，就是在测量一个随时变化的物理量，不可避免地受到随机误差影响。随机误差可能来自不可控制的随机因素影响，比如，在用雷达测量飞机的方位和速度时，可能受到地磁、气温、地形的影响。由于测量仪器精

度的限制也会产生随机误差，比如用最小刻度是 1 度的温度计测量温度，只能把不足 1 度的值四舍五入或者估计小数点后一位数字。随机误差也可能来自特定条件下才发生的程序错误。

系统误差是多次测量持续偏高或偏低的误差，多次重复测量不能消除或减少系统误差。系统误差可能来自仪器本身的误差，比如用不锈钢直尺测量家具高度，直尺本身在温度不同时长度有细微变化。系统误差也可能来自仪器使用不当，比如用天平测量质量时天平没有配准。当发现有系统误差时，必须找出引起误差的原因并消除。

在记录实验数据时由于人的过失可以导致误差发生, 这样的误差称为**过失误差**。比如，在记录仪表（如水表、电表）的读数时看错数字，在记录数值时写错小数点位置，在上传数据时报告了过时的或错误的数据，等等。统计数据分析必须甄别并改正这样的过失误差，否则会对分析结果产生严重影响。在使用计算机软件处理数据时程序设计的质量问题也会导致误差发生。比如，当输入条件不满足模型要求而程序中未进行检查时，可能给出错误的结果。

2.2 数值误差

数值误差是用电子计算机进行数据存储和计算时产生的误差，设计算法时必须了解并尽可能避免这种误差。

设 A 为结果的精确值，a 为 A 在计算机中的近似值，则 $\Delta = a - A$ 称为**绝对误差**, 简称误差。$\delta = \frac{a-A}{A} = \frac{\Delta}{A}$ 称为**相对误差**。相对误差没有量纲，常常用百分数表示。绝对误差和相对误差常常仅考虑其绝对值。实际中如果能估计绝对误差和相对误差的取值区间或绝对值的上限，则可以确定误差大小。如果绝对误差 Δ 的绝对值上限估计为 $\tilde{\Delta}$, 则相对误差的绝对值上限可以用 $\delta = \frac{\tilde{\Delta}}{a}$ 估计，因为 A 的真实值一般是未知的。有 p 位有效数字的一个十进制近似数的相对误差控制在 5×10^{-p} 以下。

没有数值计算经验的读者往往会有一个错误认识，认为计算机得到的结果总是准确的、可信的。即使不考虑模型误差、随机误差和系统误差的影响，用计算机进行数值计算也不可避免地受到误差影响，只要我们把误差控制在可接受的范围就可以了。

2.2.1 计算机二进制

为了解数值计算误差，我们首先需要了解计算机中数值的存储方式。计算机中的数都用二进制表示，包括定点表示和浮点表示两种。

2.2.1.1 定点表示

定点数主要用于保存整数，是精确表示的，定点数的四则运算可以得到精确结果，但整数除法需要特别注意余数问题，另外定点表示的整数范围有限，比如用 32 个二进制位可以表示 $-2^{31} \sim 2^{31}-1$ 之间的整数（约 10 位有效数字），定点数的计算结果有可能溢出，即超出能表示的整数范围。

例 2.1 (Julia 中的定点整数 (*))**.** 显示 Julia 语言中整数的界限与二进制表示。

整数 1 的表示：

```
x=1
typeof(x)
## Int64
bitstring(x)
## "0000000000000000000000000000000000000000000000000000000000000001"
```

上面的 x 是 64 位定点整数，首位的 0 是符号位，表示非负，后面有 62 个 0 和 1 个 1。

```
x=Int8(1)
bitstring(x)
##"00000001"
```

上面的 x 是 8 位定点整数，首位的 0 是符号位，表示非负，后面有 6 个 0 和 1 个 1。

```
x=UInt8(1)
bitstring(x)
## "00000001"
```

上面的 x 是 8 位无符号定点整数，有 7 个 0 和 1 个 1, 没有符号位。

```
x = Int8(-1)
bitstring(x)
## "11111111"
```

上面的 x 是 8 位定点整数，首位 1 表示负号，后面有 7 个 1。

Julia 中各种整数类型的最大可表示值：

```
typemax(Int8)
## 127
```

Int8 约有 2-3 位十进制有效数字。

```
typemax(Int16)
## 32767
```

Int16 约有 4-5 位十进制有效数字。

```
typemax(Int32)
## 2147483647
```

Int32 约有 9-10 位十进制有效数字。

```
typemax(Int64)
## 9223372036854775807
```

Int64 约有 18-19 位十进制有效数字。

Int64 的加法超出存储界限时发生溢出，而且不提供错误信息：

```
typemax(Int64) + 1
## -9223372036854775808
bitstring(typemax(Int64))
## "0111111111111111111111111111111111111111111111111111111111111111"
bitstring(typemax(Int64)+1)
## "1000000000000000000000000000000000000000000000000000000000000000"
```

变成了负数。

但是 Int8, Int16, Int32 可以自动提升到更大储存的类型:

```
typemax(Int8) + 1
## 128
```

注意 Julia 中整数加法、减法、乘法结果还是整数，除法按实数除法计算，整数之间的乘法也是整数。两个整数之间进行乘方计算时结果也是整数，这是 Julia 语言的特殊规定，其它语言中一般不这样规定，R 语言中的四则运算和乘方运算基本都是按浮点数计算。

Julia 程序：

```
2^63
-9223372036854775808
```

溢出了。写成实数的乘方：

```
2.0^63
## 9.223372036854776e18
```

※※※※※

2.2.1.2 浮点表示

数值计算时主要使用浮点表示, 比如用 64 个二进制位能表示绝对值在 $10^{-308} \sim 10^{308}$ 之间的实数，有效数字大约有 16 ~ 17 位。二进制浮点数的表示包括 (S, E, F) 三个部分，即正负号、指数部分、小数部分。小数部分的位数 d 决定了精度。一般存储 $2^{-1} \le F < 1$, 这样 F 中首位一般是 1，有些计算机不保存这一位。例如，二进制数 $101.101_2 = 0.101101_2 \times 2^3$ 可以表示为 $(+, +11_2, .101101_2)$。

不同的计算机硬件可能采用不同的浮点数表示，在现代常用的 IEEE 64 位浮点数的表示中，一个浮点数 x 被表示成

$$x = \pm 2^{q-1023} \times (1.b_1 b_2 \ldots b_{52})$$

其中符号 $\pm$ 用 64 位的首位表示，随后 11 位存储指数部分，看成一个 11 位无符号定点整数 q，指数部分 E 代表 $\times 2^{q-1023}$，但是 $q=0$ 和 $q=2^{11}-1$ 有特殊解释。最后的 52 位是二进制小数部分 F，约定小数点前面总是 1 并且不保存这个 1。当 $q=0$ 时，小数部分约定小数点前面是 0，保存所有的二进制小数位，这时小数部分的有效位数比正常情况少。$q>0$ 时绝对值最小的浮点数约为 `2.2E-308`。当 $q=2^{11}-1$(所有的 11 位指数位都为 1) 时，代表某些不正常的实数，这时如果小数位都是零，用来代表 Inf(无穷大); 如果小数位有 1，用来代表 NaN(比如 0.0/0.0 的结果)。$q<2^{11}-1$ 时绝对值最大的浮点数约为 `1.80E308`。

由于 E 和 F 两部分的位数限制，二进制浮点数只有有限个，这些数的集合记为 $\mathcal{F}$，在这个集合中可以制定一个次序，(S,E,F) 的下一个数是 $(S,E,F+2^{-d})$。对于实数域 $\mathbb{R}$ 中的实数 x，当 $|x|$ 超出浮点数范围时，我们不能表示；当 $|x|$ 没有超出浮点数范围时，一般也只能用 $\mathcal{F}$ 中的数来近似表示。比如，十进制数 0.1 如果表示成二进制数，是一个无限循环小数 $0.000\dot{1}100\dot{1}_2$, 只能存储其中有限位。在近似表示时对于不能保存的位数计算机中可能用了舍入法或者截断法处理, 这样造成的误差叫做**舍入误差**。

例 2.2 (Julia 中的浮点数的二进制表示 (*))**.** 用 Julia 语言演示 64 位浮点数 0.1 的二进制存储。

```
bitstring(0.1)
## "0_01111111011_1001100110011001100110011001100110011001100110011010"
```

这是 Julia 语言中 Float64 类型的浮点数，按 IEEE 64 位浮点数格式存储。首位 0 是符号位，后面 11 位是指数位，存储 q 的无符号整数值，恰好等于 1019，表示 $2^{1019-1023}=2^{-4}$，最后 52 位是小数位，都在二进制小数点后面，而小数点前面有一个 1，所以近似表示 $1.\dot{1}00\dot{1}$，因此表示为

$$0.1 \approx +2^{-4}\times\left[1+(1\times 2^{-1}+0\times 2^{-2}+0\times 2^{-3}+1\times 2^{-4})\right.$$
$$\left.+(1\times 2^{-5}+0\times 2^{-6}+0\times 2^{-7}+1\times 2^{-8})+\cdots\right].$$

※※※※※

设实数 z 在浮点数可表示范围内，将其浮点表示记为 $\mathrm{fl}(z)$，则绝对舍入误差满足

$$|\mathrm{fl}(z)-z|\le U|z|,\forall z.$$

其中 U 称为机器单位 (machine unit)，fl 用截断近似时 $U = 2^{-(d-1)}$, fl 用四舍五入时 $U = 2^{-d}$。一个浮点数 z 用浮点表示 $\mathrm{fl}(z)$ 后误差范围如下

$$\mathrm{fl}(z) = z(1+u),\ |u| \le U.$$

和 U 类似的一个数是机器 ε_{m}，ε_{m} 是使得 $1+\varepsilon_{\mathrm{m}}$ 和 1 的表示不相同的最小正浮点数。可以认为 $U \approx \varepsilon_{\mathrm{m}}/2$。典型的双精度计算 ε_{m} 约为 10^{-16} 数量级，而单精度的相应值与双精度相差约 10^9 倍, 即单精度计算与双精度计算的精度相差约 9 位有效数字。使用双精度实数可以减少表示误差和计算误差，但是不能消除这两种误差，所以不能盲目相信使用了双精度后就总是能准确地进行数值计算。

在 R 软件中用变量`.Machine$double.eps` 保存双精度计算的机器 ε_{m} 值。在 Julia 语言中用 `eps(Float64)` 获得双精度浮点数的机器 ε_{m} 值。

2.2.2 计算误差

由于计算机存储的有限性，浮点数的四则运算不符合结合律，计算结果与计算次序有关。不同的算法可能导致不同的计算误差，应该尽可能选用计算精度高的数学公式，另外在设计算法时需要注意避免一些损失精度的做法。

例 2.3 (结合律的反例)**.** 浮点数的四则运算不符合结合律，计算结果与计算次序有关。

$1.1 + 0.1 - 1.2$ 的精确结果是 0，但是在 R 中计算，得

```
1.1 + 0.1 - 1.2
## [1] 2.220446e-16
```

而

```
1.1 - 1.2 + 0.1
## [1] 1.387779e-16
```

两个计算次序不同，结果都不等于零而且不同计算次序结果不同。Julia 计算结果与 R 的结果相同。

※※※※※

例 2.4 (累加的简化例子). 计算

$$\sum_{n=1}^{999} \frac{1}{n(n+1)}$$

可以直接累加计算，造成很大的累积误差。只要把公式变换成

$$\sum_{n=1}^{999} \frac{1}{n(n+1)} = \sum_{n=1}^{999} \left(\frac{1}{n} - \frac{1}{n+1} \right) = 1 - \frac{1}{1000} = 0.999$$

就只要计算一个除法和一个减法。

```
nmax <- 999
exact <- 1 - 1/(nmax + 1)
direct <- 0.0
for(n in 1:nmax){
  direct <- direct + 1/(n*(n+1))
}
direct
## [1] 0.999
direct - 0.999
## [1] 6.661338e-16
exact - 0.999
## [1] 0
```

在 R 中计算，后一算法与精确值 0.999 的差等于 0，前一算法与精确值 0.999 的差等于 6.66133814775094e-16。

Julia 中计算：

```
let
    nn = 999
    s = 0.0
    for n=1:nn
    s += 1.0/(n*(n+1))
    end
    @show s;
```

```
    @show s - 0.999;
end;
## s = 0.9990000000000007
## s - 0.999 = 6.661338147750939e-16
```

累加计算的结果有 10^{-16} 级别的误差。

※※※※※

在进行浮点数加减时，两个绝对值相差很大的数的加减严重损失精度。设 $|a| \gg |b|$($\gg$ 表示 “远大于”), 则 $a+b$ 会被舍入为 a。比如，计算

$$0.1234 + 0.00001234$$

如果仅允许保留 4 位有效数字，则结果为 0.1234。为避免这样的问题，应该只对大小相近的数相加减，比如，可以把小的数组合后分别相加，再加起来。

两个相近数相减损失有效数字。如:

$$0.8522 - 0.8511 = 0.0011$$

有效数字个数从 4 位减低到 2 位。统计中如下的方差计算公式：

$$\sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - \frac{1}{n}(\sum_{i=1}^{n} x_i)^2$$

就存在这样的问题。在对分布函数 $F(x)$ 计算右尾部概率 $1-F(x)$ 时，如果 x 很大则 $F(x)$ 很接近 1，计算 $1-F(x)$ 会造成结果有效数字损失。为此，统计软件中计算分布函数时常常可以直接计算右尾概率，以及概率的对数。

例 2.5. 逻辑斯蒂分布函数为 $F(x) = \frac{1}{1+e^{-x}}$, 当 x 很大时如果直接计算 $1 - F(x) = 1 - \frac{1}{1+e^{-x}}$，就会出现有效位数损失；只要改用

$$1 - F(x) = \frac{1}{1+e^{x}}$$

计算，就可以避免两个相近数相减，提高精度。

例如，用 $1 - \frac{1}{1+e^{-x}}$ 计算 $1-F(8)$ 结果为:

```r
x <- 8
y1 <- 1 - 1/(1 + exp(-x)); sprintf("%20.16E", y1)
## [1] "3.3535013046637197E-04"
```

用 $\frac{1}{1+e^x}$ 计算 $1-F(8)$ 结果为

```r
x <- 8
y2 <- 1/(1 + exp(x)); sprintf("%20.16E", y2)
## [1] "3.3535013046647811E-04"
```

第一种计算公式损失了 4 位以上的有效数字:

```r
cat(sprintf("%20.16E", y1), "\n", sprintf("%20.16E", y2), "\n", sep="")
## 3.3535013046637197E-04
## 3.3535013046647811E-04
```

用 Julia 语言进行上述计算:

```julia
x = 8
y1 = 1 - 1/(1 + exp(-x))
y2 = 1/(1 + exp(x))
[string(y1), string(y2)]
```

```
2-element Array{String,1}:
 "0.00033535013046637197"
 "0.00033535013046647811"
```

第一种计算公式损失了 4 位以上的有效数字。

※※※※※

例 2.6. 计算 $\sqrt{x^2+1}-|x|$，直接计算在 $|x|$ 很大时会有较大误差，改成 $1/(\sqrt{x^2+1}+|x|)$ 则不损失精度。

例 2.7 (数值微分中的浮点运算误差 (*))**.** 如果某函数自变量相对变动在机器精度 U 附近时函数值也有变化，则函数值的计算结果被舍入误差严重影响，不能得到有效计算结果。

对一元可微函数 $f(x)$,

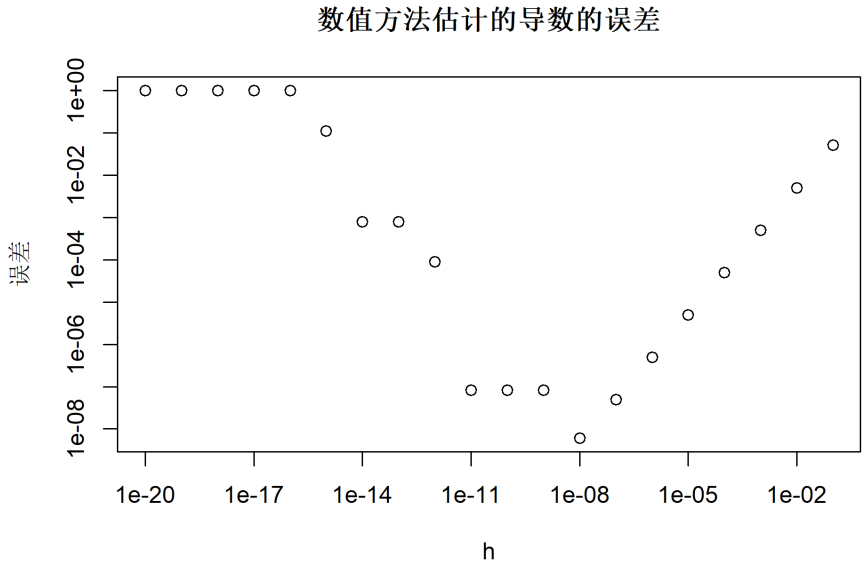
$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

当 h 很小时可以用 $\frac{f(x+h)-f(x)}{h}$ 近似计算 $f'(x)$。

取 $f(x) = e^x$, 用这种方法近似计算 $x = 0$ 处 $f'(x)$ 的值。真值等于 1。

令 $g(h) = |\frac{e^h-1}{h} - 1|$ 为近似计算的误差。用浮点数计算，分别取 $h = 10^{-1}, 10^{-2}, \ldots, 10^{-20}$。

```
hvec <- 10^seq(-1, -20, by=-1)
yp <- abs((exp(hvec) - 1) / hvec - 1.0)
plot(hvec, yp,
     xlab="h", main=" 数值方法估计的导数的误差",
     ylab=" 误差", log="xy")
abline(h=0, col="gray", lty=3)
```



```
cbind(hvec, yp)
##         hvec           yp
##  [1,] 1e-01 5.170918e-02
##  [2,] 1e-02 5.016708e-03
##  [3,] 1e-03 5.001667e-04
##  [4,] 1e-04 5.000167e-05
##  [5,] 1e-05 5.000007e-06
##  [6,] 1e-06 4.999622e-07
##  [7,] 1e-07 4.943368e-08
##  [8,] 1e-08 6.077471e-09
##  [9,] 1e-09 8.274037e-08
## [10,] 1e-10 8.274037e-08
## [11,] 1e-11 8.274037e-08
## [12,] 1e-12 8.890058e-05
## [13,] 1e-13 7.992778e-04
## [14,] 1e-14 7.992778e-04
## [15,] 1e-15 1.102230e-01
## [16,] 1e-16 1.000000e+00
## [17,] 1e-17 1.000000e+00
## [18,] 1e-18 1.000000e+00
## [19,] 1e-19 1.000000e+00
## [20,] 1e-20 1.000000e+00
```

发现 $g(0.1) = 0.052$ 有较大误差，逐步减小 h 的值，同时误差变小，$g(10^{-8}) = 6.1 \times 10^{-9}$，这时误差已经比较小。为了进一步减小误差，取更小的 h 值，发现并不能减小误差，$g(10^{-11}) = 8.3 \times 10^{-8}$ 与 $h = 10^{-8}$ 时误差相近。继续减小 h 的值，发现 $g(10^{-15}) = 0.11$，误差反而变得很大，这是因为 $\exp(10^{-15}) - 1$ 是两个十分接近的数相减，有效位数损失很多，除以 10^{-15} 又放大了误差。进一步减小 h，$g(10^{-16}) = 1$, 这是因为 $\exp(10^{-16}) - 1$ 的浮点结果等于 0。

此例的 Julia 语言版本：

```
h = 10.0 .^ (-1:-1:-20)
g(x) = abs((exp(x) - 1.0)/x - 1.0)
[h g.(h)]
```

```
20×2 Matrix{Float64}:
 0.1       0.0517092
 0.01      0.00501671
 0.001     0.000500167
 0.0001    5.00017e-5
 1.0e-5    5.00001e-6
 1.0e-6    4.99962e-7
 1.0e-7    4.94337e-8
 1.0e-8    6.07747e-9
 1.0e-9    8.27404e-8
 1.0e-10   8.27404e-8
 1.0e-11   8.27404e-8
 1.0e-12   8.89006e-5
 1.0e-13   0.000799278
 1.0e-14   0.000799278
 1.0e-15   0.110223
 1.0e-16   1.0
 1.0e-17   1.0
 1.0e-18   1.0
 1.0e-19   1.0
 1.0e-20   1.0
```

※※※※※

例 2.8. 误差的累积会放大误差。

把 0.1 累加一千万次，与真实值 `1e6` 比较。

Julia 语言编程：

```
x = 0.0
for i=1:10_000_000
    x += 0.1
end
x - 1_000_000.0
## -0.0001610246254131198
```

误差约 10^{-4}。注意双精度数 ε_m 在 10^{-16} 左右，`1e6` 的表示误差在 10^{-10} 左右，这个累加误差比表示误差大了 6 个数量级。

※※※※※

浮点数做乘法运算时，结果有效位主要取决于精度较低的数。做除法运算时，如果分母绝对值很小则会产生较大的绝对误差。

在比较两个浮点数是否相等时，因为浮点数表示和计算的不精确性，程序中不应该直接判断两个浮点数是否相等，而应该判断两者差的绝对值是否足够小。

设计计算机数值计算的算法时一定要注意浮点算法的局限。要了解能保存的最大和最小实数。例如，浮点数的范围有时不够用。超过最大值会发生**向上溢出**，绝对值小于最小正浮点数会发生**向下溢出**。向上溢出一般作为错误；向下溢出经常作为结果零。还要了解相对误差的范围，用机器精度 U 或 ε_m 表示。

例 2.9 (计算同生日概率). 设一个班有 n 个人，则至少有两人的生日的月、日重合的概率为

$$p_n = 1 - \frac{P_{365}^n}{365^n}$$

其中 $P_{365}^n = 365 \times 364 \times \cdots \times (365 - n + 1)$。

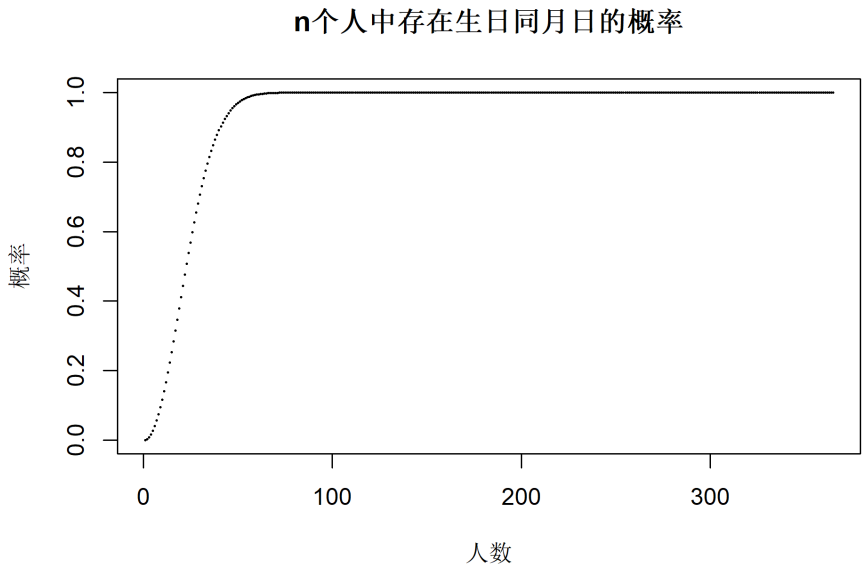
如果直接计算分式的分子和分母，则当 n 达到 121 时分母的计算溢出。

```
365^120
## [1] 2.986371e+307
365^121
## [1] Inf
```

只要改用如下计算次序就可以解决这个问题：

$$P_n = 1 - \prod_{j=0}^{n-1} \frac{365 - j}{365}$$

```r
days <- 365
nvec <- 1:days
pvec <- numeric(days)
pvec[1] <- 0
for(n in 2:days){
  pvec[n] <- 1 - prod((365 - (0:(n-1)))/365)
}
plot(nvec, pvec, main="n 个人中存在生日同月日的概率",
     pch=16, cex=0.2,
     xlab=" 人数", ylab=" 概率", ylim=c(0,1))
```



事实上，以上的对 n 的循环还可以用 `cumprod()` 函数简化为：

```r
nvec <- 1:365
pvec <- 1 - cumprod((365 - 0:364)/365)
```

Julia 语言版本：

```
365.0^120
## 2.986370829311945e307
365.0^121
## Inf
```

```
days = 365
nvec = 1:days
pvec = zeros(days)
pcum = 1
for n=1:days
    pcum *= (365 - (n-1))/365
    pvec[n] = 1 - pcum
end
pvec
```

注意 Julia 语言的循环不损失效率，不需要费尽心思改成向量化的程序风格；而 R 如果用循环则会慢一到二个数量级。

Julia 也支持“广播”（类似于向量化运算）和 `cumsum`，上述 Julia 程序也可改写为：

```
nvec = 1:365
pvec = 1 .- cumprod((365 .- (0:364)) ./ 365)
```

※※※※※

例 2.10 (softmax 函数的稳定性问题 (*))**.** 在机器学习中常用如下的 softmax 函数：

$$\text{softmax}(x_1, \dots, x_n) = \left(\frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}},\ i = 1, 2, \dots, n \right).$$

讨论其由于数值范围限制造成的误差。

当各自变量 x_i 取值范围适中时，这个函数没有什么问题。但是当某个 x_i 取值很大时，e^{x_i} 会发生向上溢出错误；当所有 x_i 都是绝对值很大的负数时，e^{x_j} 会

向下溢出变成 0 或者只有很少的有效位数，使得分母等于零或接近于零，造成除零错误或者产生很大相对误差。为此，将公式修改为

$$\mathrm{softmax}(x_1,\dots,x_n)=\left(\frac{\exp\{x_i-\max_k x_k\}}{\sum_{j=1}^{n}\exp\{x_j-\max_k x_k\}},\ i=1,2,\dots,n\right),$$

因为 $x_i-\max_k x_k\le 0$，所以不会向上溢出；因为分母中恰好有一个 $e^0=1$，所以不会出现分母绝对值很小导致大的相对误差的情形。

※※※※※

2.2.3 截断误差

除了数值计算中浮点数表示和计算带来的误差，另一类更大的误差是**截断误差**。比如，计算机中的超越函数如 e^x 通常用级数逼近，而级数只能计算到有限项，舍去部分造成误差。在确定需要计算的项数时，机器精度 U 或 ε_{m} 给出了一个精度的界限，任何使得结果相对变化小于 U 或 ε_{m} 的尾项都不需要再加进来。数值计算中方程求根、求最小值点经常使用迭代法求解，而迭代法也不能无限执行，一般预定一个精度，达到精度时计算停止，会造成误差。所以，即使是编程语言内置的函数或公认的程序包的结果也是有数值计算误差的, 不一定总能达到机器精度 U，见例2.7。

2.3 计算复杂度

例 2.11. 计算多项式

$$P_n(x)=a_0+a_1x_1+\cdots+a_nx^n$$

若直接计算，a_kx^k 需要计算 k 次乘法，总共需要计算 $1+2+\cdots+n=\frac{1}{2}n(n+1)$ 次乘法和 n 次加法, 我们称这样的算法需要 $O(n^2)$ 次浮点运算。

如果用如下递推算法（秦九韶法）：

秦九韶法
$u_0\leftarrow a_n$

秦九韶法

for $(k$ **in** $1:n)$ {

$u_k \leftarrow u_{k-1} \cdot x + a_{n-k}$

}

输出 u_n

就只需要 n 次乘法和 n 次加法, 即 $O(n)$ 次浮点运算，在提高了效率的同时也会提高计算精度。

例2.11展示了不同算法的运算次数可以有重大的差别。一个算法的计算开销主要是计算时间长短和占用的存储空间大小。计算时间取决于算法的运算次数，计算机中各种运算的种类较多，数值计算的算法主要计算**浮点运算次数** (Floating Point Operations, FLOPs)。浮点运算次数将算法中每一次四则运算计为一次浮点运算。浮点运算次数一般表示为 $O(n), O(n^2)$ 等随问题规模 n 而变化的形式，最好能精确到具体的比例系数，如 $O(\frac{4}{3}n^3)$。

按照数学分析中的记号习惯，称浮点运算次数 $T_n = O(n^k)$，$k > 0$，是指存在 $C > 0$ 使得当 n 充分大时 $T_n \leq Cn^k$。例如，例2.11的直接计算方法的浮点运算次数为 $\frac{1}{2}n(n+1)+n = \frac{1}{2}n^2 + \frac{3}{2}n$，令 $C = \frac{3}{2}$, 则 $n \geq 1$ 时 $\frac{1}{2}n(n+1)+n \leq Cn^2$，所以可以称此算法的浮点运算次数为 $O(n^2)$。若 $\lim_{n\to\infty} \frac{T_n}{Cn^k} = 1$，称 $T_n \sim Cn^k$，这时也称浮点运算次数为 $O(Cn^k)$，比如例2.11的直接计算方法的浮点运算次数为 $O(\frac{1}{2}n^2)$。事实上，若 T_n 为 n 的 k 次多项式，首项系数为 C，则 $T_n \sim Cn^k$，而且对任意小的 $\delta > 0$ 都有 n 充分大时 $T_n \leq (C+\delta)n^k$，所以将 $T_n \sim Cn^k$ 记作 $T_n = O(Cn^k)$ 是合理的。称浮点次数为 $O(n^k)$ 的算法具有 k 阶时间复杂度。

若 $\lim_{n\to\infty} \frac{T_n}{n^k} = 0$，记作 $T_n = o(n^k)$。$o(n^k)$ 一定是 $O(n^k)$。$T_n \sim Cn^k$ 可以记作 $T_n = Cn^k + o(n^k)$。

2.4 随机误差的度量

随机误差来自于观测样本或随机模拟中的随机因素，一般随着样本量增大而减小，但随机误差是随机变量，不能给出严格误差界限，只能用概率方法描述随机误差大小。

为了方便讨论当样本量 n 趋于无穷时随机误差大小的变化规律，引入如下的 O_p 和 o_p 的记号。

$O_p(\cdot)$ 是概率论中表示依概率有界的记号。如果 $\{\xi_n\}$ 和 $\{\eta_n\}$ 是两个随机变量序列，满足

$$\lim_{M\to\infty}\sup_{n\geq 1}P\left(\left|\frac{\xi_n}{\eta_n}\right|>M\right)=0,$$

则称 $\{\frac{\xi_n}{\eta_n}\}$ **依概率有界**，记为 $\frac{\xi_n}{\eta_n}=O_p(1)$ 或 $\xi_n=O_p(\eta_n)$。

如果对 $\forall\delta>0$ 都有

$$\lim_{n\to\infty}P\left(\left|\frac{\xi_n}{\eta_n}\right|>\delta\right)=0$$

称 ξ_n/η_n **依概率趋于零**，记为 $\xi_n/\eta_n=o_p(1)$ 或 $\xi_n=o_p(\eta_n)$。

O_p 和 o_p 是我们用来估计随机误差幅度的有效工具，有如下性质:

(1) $o_p(1)=O_p(1)$;
(2) 如果 $\{\eta_n\}$ 是趋于零的常数列，则 $\xi_n=O_p(\eta_n)$ 时必有 $\xi_n=o_p(1)$。
(3) $o_p(1)+o_p(1)=o_p(1)$;
(4) $o_p(1)+O_p(1)=O_p(1)$;
(5) $o_p(1)\cdot o_p(1)=o_p(1)$;
(6) $o_p(1)\cdot O_p(1)=o_p(1)$;
(7) 如果 $\{\xi_n\}$ 以概率 1 趋于零，或 $\{\xi_n\}$ 是趋于零的非随机实数列，则 $\xi_n=o_p(1)$;
(8) 单个随机变量 $\xi=O_p(1)$;
(9) 如果随机变量序列 ξ_n 依分布收敛到分布 F, 则 $\xi_n=O_p(1)$，且 $\xi_n+o_p(1)$ 仍依分布收敛到分布 F。

如果 ξ_n 满足中心极限定理

$$\frac{\xi_n-E\xi_n}{\sqrt{\mathrm{Var}(\xi_n)}}\xrightarrow{d}\mathrm{N}(0,1)$$

设 $n\mathrm{Var}(\xi_n)$ 有上界 D，则

$$\xi_n-E\xi_n=\frac{\sqrt{n\mathrm{Var}(\xi_n)}}{\sqrt{n}}\cdot O_p(1)=O_p\left(\frac{\sqrt{D}}{\sqrt{n}}\right)=o_p(1).$$

例如，设 $X_1,X_2,\ldots$ 独立同分布，有共同的方差 σ^2, $\bar{X}_n=\frac{1}{n}\sum_{i=1}^n X_i$, 则 $\bar{X}_n-EX_1=O_p(\frac{\sigma}{\sqrt{n}})$。

2.5 问题的适定性与算法稳定性

把算法粗略看成

$$\text{输出} = f(\text{输入})$$

问题的适定性可以看成是输入微小变化时输出变化的大小，如果输出连续地依赖于输入并且输入的微小变化引发的输出变化也是微小的，则问题是适定的。

为了问题的适定性，定义**条件数**如下：

$$\frac{|f(\text{输入}+\delta)-\text{输出}|}{|\text{输出}|} = \text{条件数}\cdot\frac{|\delta|}{|\text{输入}|}.$$

即条件数 (condition index) 是输出的相对变化与输入的相对变化的比值。当 $f(x)$ 为一元可微函数时，计算 $f(x)$ 的条件数可以用微分表示为

$$\kappa(f,x) = |xf'(x)/f(x)|.$$

条件数较小，比如 $\kappa < 10$ 的时候，可以设计算法给出问题的精确解。条件数很大的问题称为**病态问题** (ill-conditioned), 可能会有很大误差。条件数等于无穷或不存在的问题称为**不适定问题** (ill-posed)。

即使问题是适定的，同一个问题也可以有多种不同算法实现，不同算法结果的精度可能有很大的差别，只有设计良好的算法才能使得结果达到条件数代表的精度。设计不当的算法可能因为浮点运算的不良影响使得结果精度远低于条件数所代表的精度，这样的算法称为**不稳定**的算法。对于适定问题，应该设计稳定的算法使得浮点运算的不良影响受控，使得结果能够基本达到条件数所代表的精度。

例 2.12 (解一元二次方程的稳定性问题 (*))**.** 考虑二次方程

$$z^2 - x_1 z + x_2 = 0,\ (x_1 > 0, x_2 > 0).$$

设其中 x_2 很小，这时两个根都是正根，求其中较小根 z_2。

z_2 定义为

$$z_2 = z_2(x_1,x_2) = \frac{x_1 - \sqrt{x_1^2 - 4x_2}}{2} \tag{2.1}$$

在 x_2 变化时条件数

$$\begin{aligned} C =& \left| x_2 \frac{\partial z_2(x_1, x_2)}{\partial x_2} / z_2(x_1, x_2) \right| \\ =& x_2 \frac{1}{x_1^2 - 4x_2} \frac{2}{x_1 - \sqrt{x_1^2 - 4x_2}} = \frac{z_1}{z_1 - z_2} \end{aligned}$$

当 z_1 较大，z_2 很小时 C 接近 1，问题适定性很好。当判别式 $x_1^2 - 4x_2$ 很接近于零时两个根 z_1 和 z_2 差很小，条件数很大，这时 x_2 的一个微小变化可能引起根 z_2 的很大变化。

即使在 $C = 1$ 时不适当的算法也可能会造成结果不稳定。设判别式 $x_1^2 - 4x_2$ 不接近于零，两个根 z_1 和 z_2 差距很大，但是当 x_2 很小时公式(2.1)的分子中 x_1 和 $\sqrt{x_1^2 - 4x_2}$ 很接近，相减会造成精度损失，改用公式

$$z_2 = \frac{2}{x_1 + \sqrt{x_1^2 - 4x_2}} \tag{2.2}$$

则避免了相近数相减，提高了精度。

下面程序中 `x1>0`, `x2>0`，方程的左边看成 `z`, `x1`, `x2` 的函数：

```
lhs <- function(z, x1, x2) z^2 - x1*z + x2
```

给定 `x1`, `x2`, 按(2.1)计算 `z1`(较大根) 和 `z2`(较小根) 的函数为：

```
z1_1 <- function(x1, x2) (x1 + sqrt(x1^2 - 4*x2))/2
z2_1 <- function(x1, x2) (x1 - sqrt(x1^2 - 4*x2))/2
```

给定 `x1`, `x2`, 按(2.2)计算 `z2`(较小根) 的函数为：

```
z2_2 <- function(x1, x2) 2*x2/(x1 + sqrt(x1^2 - 4*x2))
```

给定 `x1`, `x2`, 计算条件数的函数为：

```
kapa <- function(x1, x2){
  z1_1(x1,x2)/(z1_1(x1,x2) - z2_2(x1,x2))
}
```

取正常的 x1=1，取 x2 为很小的值，这里取 $0.5\varepsilon_m$：

```
x1 <- 1.0
x2 <- 0.5*.Machine$double.eps
```

这时，第一种方法解出的 z2 为：

```
z2_1(x1,x2)
## [1] 1.110223e-16
```

第二种方法的解出的 z2 为：

```
z2_2(x1,x2)
## [1] 1.110223e-16
```

两个解的差为：

```
z2_1(x1,x2) - z2_2(x1,x2)
## [1] -2.46519e-32
```

对 x2 增加一个扰动 δ：

```
dx2 <- 0.1*.Machine$double.eps
```

这时两种方法解出的 z2 及其差为：

```
z2_1(x1, x2 + dx2)
## [1] 1.665335e-16
z2_2(x1, x2 + dx2)
## [1] 1.332268e-16
z2_1(x1, x2 + dx2) - z2_2(x1, x2 + dx2)
## [1] 3.330669e-17
```

两种方法的结果差别很大。增加扰动后的变化与条件数给出的相对误差界限比较，条件数为：

```
kapa(x1,x2)
## [1] 1
```

条件数等于 1，问题适定；但是两种方法在输入的 x2 加了扰动之后的相对变化分别为：

```
(z2_1(x1, x2 + dx2) - z2_1(x1,x2))/z2_1(x1, x2)
## [1] 0.5
(z2_2(x1, x2 + dx2) - z2_2(x1,x2))/z2_2(x1, x2)
## [1] 0.2
```

这些相对变化应该都近似等于 $C\delta/x_2 = 0.2$，但第一种方法的相对变化超出了应有的值，其原因是(2.1)的分子中 x2 很小，基本相当于 x1 - x1，是两个很相近的数相减，会严重损失精度。

此例计算的 Julia 语言版本：

```
f0(z, x1, x2) = z^2 - x1*z + x2 # 方程左边
# 较小的根，第一种公式，直接计算
z2v1(x1,x2) = (x1 - sqrt(x1^2 - 4*x2))/2.0 # 直接求较小根的公式
# 较小的根，第二种公式，避免减法损失精度
z2v2(x1,x2) = 2*x2/(x1 + sqrt(x1^2 - 4*x2)) # 改进的求较小根的公式
# 计算 z2 的条件数，在 x2 变化时 z2 的条件数
f1cond(x1,x2) = 2*x2/(x1^2 - 4*x2)/(x1 - sqrt(x1^2 - 4*x2))
f1cond(1.0, 0.5*eps(1.0))      # 取 x1=1, x2=0.5 , 求条件数，约为 1
## 1.0000000000000004
```

取 x1=1, x2=0.5 , 直接算较小根:

```
y1 = z2v1(1.0, 0.5*eps(1.0))
## 1.1102230246251565e-16
```

取 x1=1, x2=0.6 , 直接求较小根:

```
y1d = z2v1(1.0, 0.6*eps(1.0))
## 1.6653345369377348e-16
```

采用第一种公式时，x2 变化 0.1 时第一种方法求较小根的估计的条件数，大于理论条件数：

```
(y1d - y1)/(0.1*eps(1.0))
## 2.5
```

取 x1=1, x2=0.5 , 用改进方法计算较小根：

```
y2 = z2v2(1.0, 0.5*eps(1.0))
## 1.1102230246251568e-16
```

取 x1=1, x2=0.6 , 用改进方法计算较小根：

```
y2d = z2v2(1.0, 0.6*eps(1.0))
## 1.332267629550188e-16
```

x2 变化 0.1 时用改进方法得到的结果估计条件数，等于理论值：

```
(y2d - y2)/(0.1*eps(1.0))
## 0.9999999999999998
```

第二种算法得到的相对变化与条件数一致，而第一种算法的结果则有过大的变化。Julia 的计算与 R 的结果类似但不相同。

※※※※※

稳定性研究经常用**倒推法**：设精确结果应该为 $y = f(x)$, x 是输入，y 是输出。有误差的结果记作 $y^* = f^*(x)$, 设 y^* 等于某个输入 x^* 的精确结果 $f(x^*)$，则 $\|x - x^*\|$ 的大小代表了算法的稳定程度, $\|x - x^*\|$ 称为倒推误差，$\|x - x^*\|/\|x\|$ 称为倒推相对误差。如果 $\|x - x^*\|/\|x\|$ 很小则称算法是倒推稳定的。

例 2.13 (浮点数加法的倒推误差)**.** 考虑两个大小相近的浮点数加法的倒推稳定性。

令 $f(x)=x_1+x_2$, 则浮点运算结果为

$$f^*(x)=\mathrm{fl}(\mathrm{fl}(x_1)+\mathrm{fl}(x_2))$$

(这里不考虑加法运算本身的误差)，于是

$$\begin{aligned}f^*(x)=&x_1+\delta_1x_1+x_2+x_2+\delta_2x_2+\delta_3(x_1+x_2)\\=&[x_1+(\delta_1+\delta_3)x_1]+[x_2+(\delta_2+\delta_3)x_2]\\=&f\left((x_1+(\delta_1+\delta_3)x_1,\ x_2+(\delta_2+\delta_3)x_2)^T\right)\end{aligned}$$

其中 $|\delta_i|\le U$, $i=1,2,3$。倒推相对误差为

$$\frac{\|x^*-x\|}{\|x\|}=\frac{\|((\delta_1+\delta_3)x_1,\ (\delta_2+\delta_3)x_2)^T\|}{\|x\|}\le 2U=\varepsilon_{\mathrm{m}}.$$

※※※※※

习题

习题 1

利用机器 ε 的定义编程计算机器 ε 的值。

习题 2

设某正数 a 四舍五入后保留了 p 位有效数字，表示成 $a^*=(0.a_1a_2a_3\ldots a_p)_{10}\times 10^m$ $(a_1\neq 0)$。估计其绝对误差和相对误差的范围，并分析误差与有效数字位数 p 的关系。

习题 3

如下级数

$$S_n=1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+(-1)^{n+1}\frac{1}{n}$$

收敛到 $\ln 2$。用下列四种不同方法计算 $n=10^6$ 时级数值并比较精度：

1. 按正常次序相加；

2. 按相反次序相加；
3. 成对组合后相加：$(1-\frac{1}{2})+(\frac{1}{3}-\frac{1}{4})+...$；
4. 按成对组合的通分后结果相加：$\frac{1}{1\times 2}+\frac{1}{3\times 4}+...$。

习题 4

对函数 e^{-x}，可以用两种公式计算：

$$(1)\quad e^{-x}=1-x+x^2/2-x^3/3!+\cdots+(-1)^k x^k/k!+...;$$
$$(2)\quad e^{-x}=1/(1+x+x^2/2+x^3/3+\cdots+x^k/k!+...)$$

对 $x=1,2,3,4$，计算到 $k=10$ 的级数并比较两种算法的计算精度。

习题 5

有如下一组观测数据：

$$249,\ 254,\ 243,\ 268,\ 253,\ 269,\ 287,\ 241,\ 273,\ 306$$
$$303,\ 280,\ 260,\ 256,\ 278,\ 344,\ 304,\ 283,\ 310$$

用两种方法计算样本方差：

1. 公式
$$S^2=\frac{1}{n-1}\left(\sum_{i=1}^{n}x_i^2-n\bar{x}^2\right)$$
2. 公式
$$S^2=\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2$$

其中每一个中间步骤保留 6 位有效数字 (R 中用 `signif` 函数把数据四舍五入到指定有效位数)。把结果与使用高精度计算的结果比较。

习题 6

对如下 t 的表达式找出发生严重精度损失的 t 值，并变化计算方法以避免精度损失：

(1) $\dfrac{1}{t} - \dfrac{1}{t^2 + a}$;

(2) $e^{-2t^2} - e^{-8t^2}$;

(3) $\ln(e^{t+s} - e^t)$;

(4) $[(1 + e^{-t})^2 - 1]t^2 e^{-t}$.

Chapter 3

概括统计量

3.1 总体和样本

统计的主要作用在于分析数据，发现数据中的规律。统计学是以概率论为数学基础的。我们先回顾一些基本概念。

假设一个变量的多个观测值 $x_1, x_2, \dots, x_n$ 可以看成某随机变量 X 的 n 个独立观测，称 X 为**总体**，称 $x_1, x_2, \dots, x_n$ 为总体 X 的**简单随机样本** (简称**样本**)，称不需要未知参数的值、仅从样本计算得到的量为**统计量**。

对样本 $x_1, x_2, \dots, x_n$ 从小到大排序得到 $x_{(1)} \le x_{(2)} \le \cdots x_{(n)}$, 称为样本的**次序统计量**。

为了估计 $F(x) = P(X \le x)$, 使用如下统计量

$$F_n(x) = \frac{\#(\{i : x_i \le x\})}{n} = \frac{1}{n} \sum_{i=1}^{n} I_{(-\infty, x]}(x_i)$$

其中 $\#(A)$ 表示集合 A 中元素的个数，$I_A(x)$ 是集合 A 的示性函数，当 $x \in A$ 时等于 1，当 $x \notin A$ 时等于 0。$F_n(x)$ 是样本统计量，也是 x 的函数，称 $F_n(x)$

为**经验分布函数**。如果 $x_{(1)} < x_{(2)} < \cdots < x_{(n)}$，易见

$$F_n(x) = \begin{cases} 0 & \text{当 } x < x_{(1)} \\ \frac{1}{n} & \text{当 } x_{(1)} \le x < x_{(2)} \\ \cdots & \cdots\cdots \\ \frac{i-1}{n} & \text{当 } x_{(i-1)} \le x < x_{(i)} \\ \cdots & \cdots\cdots \\ \frac{n-1}{n} & \text{当 } x_{(n-1)} \le x < x_{(n)} \\ 1 & \text{当 } x \ge x_{(n)} \end{cases}$$

可见 $F_n(x)$ 是 x 的单调递增右连续阶梯函数，跳跃点为各 $x_{(i)}$ 处，跳跃高度为 $\frac{1}{n}$。如果有 $x_{(i)}$ 有 k 个相等的观测值，则在 $x_{(i)}$ 处跳跃高度为 $\frac{k}{n}$。由强大数律易见 $\lim_{n\to\infty} F_n(x) = F(x)$, a.s. $\forall x \in (-\infty, \infty)$, 即 $F_n(x)$ 是 $F(x)$ 的强相合估计。进一步地，$F_n(x)$ 是 $F(x)$ 的一致强相合估计。

事实上，$F_n(x)$ 也是一个真正的分布函数。设随机变量 $W \sim F_n(x)$, 则 W 服从离散分布，在 $\{x_1, x_2, \ldots, x_n\}$ 内取值，如果各 x_i 互不相同则 W 服从 $\{x_1, x_2, \ldots, x_n\}$ 上的离散均匀分布，$P(W = x_i) = \frac{1}{n},\ i = 1, 2, \ldots, n$。如果 $\{x_1, x_2, \ldots, x_n\}$ 中有相同的观测值则其相应的取值概率是 $\frac{1}{n}$ 乘以重复次数。

下面给出**分位数函数**的定义。设随机变量 X 的分布函数 $F(x)$ 严格单调上升且连续，则存在反函数 $F^{-1}(p)$ $(0 < p < 1)$ 使得

$$F(F^{-1}(p)) = p,\ \forall 0 < p < 1.$$

称 $F^{-1}(p)$ 为 X 或 $F(x)$ 的分位数函数，称 $x_p = F^{-1}(p)$ 为 $F(x)$ 的 p 分位数。如果 $F(x)$ 没有反函数 (当 $F(x)$ 在某个区间等于常数时)，则只要 x_p 满足

$$P(X \le x_p) \ge p, \quad P(X \ge x_p) \ge 1 - p$$

就称 x_p 是 X(或 $F(x)$) 的 p 分位数。这样的 x_p 可能不唯一，取其中最小的一个，可得

$$x_p = \inf\{x : F(x) \ge p\}.$$

3.2 样本的描述统计量

在对数据建模分析之前，我们需要预先排除数据中的错误，了解数据的特点，考察数据是否符合模型假定的前提条件。这些前期准备工作统称为**探索性数据分析** (Exploratory Data Analysis, EDA)。

最常见的数据形式是一个变量的多次观测，或一组变量的多次观测。对单个变量，我们关心其分布情况；对多个变量，我们还关心变量之间的关系。

下面介绍用描述统计量来概括数据的方法。

随机变量主要有两种，离散型和连续型。离散型随机变量可以代表某种分类值，比如性别、行业、省份，也可以是离散取值的数值变量，比如 n 次独立重复试验的成功次数，一块电路板上的缺陷个数，等等。

对于取分类值的随机变量，EDA 的作用是考察其取值集合，计算取每个不同值的次数和比例，并纠正不一致的输入。比如，变量

```
sex <- factor(c('男', '女', '女', '男', '男生'))
```

保存了 5 个人的性别，用

```
table(sex)
## sex
##   男 男生   女
##   2    1    2
```

可以求得每个不同值的取值次数，并且可以发现输入存在不一致性。

Julia 版本：

```
sex = [" 男", " 女", " 女", " 男", " 男生"]
using StatsBase
countmap(sex)
```

```
Dict{String,Int64} with 3 entries:
  "女"  => 2
```

```
"男"  => 2
"男…  => 1
```

对于数值型的变量（包括离散取值和连续取值），我们主要关心其分布情况。如果是离散取值的，我们需要确定其可取值集合；如果是连续取值，我们需要了解其可取值的区间或集合。然后，我们可以从样本中计算描述分布特征的统计量，来描述变量分布。称这样的统计量为**描述统计量**。

作为基础，先给出估计分位数的方法。设 $x_{(1)} \le x_{(2)} \le \cdots \le x_{(n)}$ 为样本的次序统计量，用

$$\hat{x}_p = x_{([np])}$$

可以估计 x_p。其中 $[x]$ 表示对实数 x 向下取整, 即小于等于 x 的最大整数。当 x_p 唯一时，$\hat{x}_p$ 是 x_p 的强相合估计。但是，当 n 不太大时，这样的估计方法有些粗略。改进 x_p 估计的想法是，因为 x_p 是 $F(x) = p$ 的解，可以用适当方法把阶梯函数 $F_n(x)$ 变成严格单调上升连续函数，设改进后 $\tilde{F}_n(x) \approx F(x)$，再解 $\tilde{F}_n(x) = p$ 得到 x_p 的估计。例如，令

$$\tilde{F}_n(x_{(i)}) = \frac{i - \frac{1}{3}}{n + \frac{1}{3}},\ i = 1, 2, \dots, n$$

(如果有若干个 $x_{(i)}$ 取值相等，i 取最小下标) 并把相邻点用线段相连来定义 $\tilde{F}_n(x)$，求 $\tilde{F}_n(x) = p$ 得到 x_p 的估计。详见 [Hyndman and Fan, 1996]。R 语言中函数 `quantile(x)` 可以计算样本分位数，并提供了 [Hyndman and Fan, 1996] 描述的 9 种不同的计算方法。Julia 语言中 `Statistics.quantile` 函数计算样本分位数。

为了概括地描述数值型的随机变量分布，可以使用以下几类常用的数字特征：

3.2.1 位置特征量

假设 $F(x)$ 是一个分布函数，则 $\{F(x-\theta), \theta \in \mathbb{R}\}$ 构成了一族分布，其中 θ 是代表分布位置的数字特征。例如，$\mathrm{N}(\mu, \sigma^2)$ 分布中 μ 是代表分布位置的数字特征。

可以用来描述总体 X 的分布位置的数字特征包括期望值 EX，中位数 m, 众数 d。

3.2.1.1 均值

期望值 EX 是样本取值用取值概率或概率密度作为加权的加权平均。比如，在总共 10000 张售价 1 元的彩票中仅有 1 张是有 5000 元奖金的，则随机抽取一张的获利 Y 的期望为如下加权平均：

$$\begin{aligned} EX =& (5000-1)\times P(\text{中奖}) + (-1)\times P(\text{不中奖}) \\ =& (5000-1)\times \frac{1}{10000} + (-1)\times \frac{9999}{10000} = -0.5(\text{元}) \end{aligned}$$

对样本 $x_1, x_2, \dots, x_n$，用样本平均数

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

估计 EX。在 R 语言中用 `mean(x)` 求平均值。在 Julia 中用 `Statistics.mean(x)` 求平均值。

样本平均值受到极端值的影响很大。比如，某企业有 1 名经理和 100 名员工，经理月薪 100000 元，其他员工月薪 1000 元，则 101 位雇员的平均工资为

$$\bar{x} = \frac{1}{101}(100000 + 100\times 1000) \approx 1980.2$$

并不能真实反映工资的一般情况。

例 3.1 (样本均值递推算法 (*))**.** 用递推方法计算均值，并用 R 程序和 Julia 程序实现。

如果我们需要递推地计算样本平均值，记 $\bar{x}_k = \frac{1}{k}\sum_{i=1}^{k} x_i$，可以用如下算法：

样本平均值递推计算

$\bar{x}_1 \leftarrow x_1$
for (k **in** $2:n$) {
    $\bar{x}_k \leftarrow \frac{k-1}{k}\bar{x}_{k-1} + \frac{1}{k}x_k$
}

R 程序实现：

```r
mean.it <- function(x){
  xm <- x[1]
  for(k in seq(2, length(x))){
    xm <- (k-1)/k*xm + x[k]/k
  }
  xm
}
```

测试：

```r
set.seed(1)
x <- round(runif(5), 2); x
## [1] 0.27 0.37 0.57 0.91 0.20
mean(x)
## [1] 0.464
mean.it(x)
## [1] 0.464
```

Julia 函数：

```julia
function mean_iter(x::Vector)
    n = length(x)
    xm::Float64 = x[1]
    for k = 2:n
        xm = (k-1.0)/k*xm + x[k]/k
    end
    xm
end
```

测试：

```julia
using Random, Statistics
Random.seed!(101)
x = round.(rand(5), digits=2);
```

```
@show x;
## x = [0.72, 0.18, 0.31, 0.54, 0.93]
@show mean_iter(x);
## mean_iter(x) = 0.536
@show mean(x);
## mean(x) = 0.536
```

※※※※※

3.2.1.2 中位数

总体 X 的中位数 m 是满足

$$P(X \le m) \ge \frac{1}{2}, \quad P(X \ge m) \ge \frac{1}{2}$$

的数, 即 X 的 $\frac{1}{2}$ 分位数。对一组样本 $X_1, X_2, \dots, X_n$，当 n 为奇数时令 $\hat{m} = x_{(\frac{n+1}{2})}$, 即从小到大排序后中间一个的值，当 n 为偶数时令 $\hat{m} = \frac{1}{2}(x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)})$, 即从小到大排序后中间两个的平均值。称这样得到的 $\hat{m}$ 为**样本中位数**，用来估计总体中位数 m。在 R 语言中用 `median(x)` 求样本中位数。在 Julia 语言中用 `Statistics.median(x)`。

3.2.1.3 众数

总体 X 的众数 d 是 X 的分布密度的最大值点，或 X 的概率分布中概率值最大的取值点，可以用样本值中出现最多的一个数来估计。如果需要更精确的众数估计可以先估计分布密度，再设法估计密度的最大值点。对于纯数值型的变量来说，众数作为位置特征量比较粗略，实际用处不大。

例 3.2 (正态样本众数计算 (*))**.** 设 $X \sim \mathrm{N}(\mu, \sigma^2)$，则总体期望值、中位数和众数都是 μ。对模拟样本计算。

编写一个求众数的自定义函数：

```
sample.mode <- function(x){
  tf <- table(x)
  maxf <- max(tf)
  strmode <- names(tf)[tf == maxf]
  if(length(strmode) < length(x)){
    sort(as.numeric(strmode))
  } else {
    numeric(0)
  }
}
```

这个自定义函数规定如果每个值都恰好出现一次就没有样本众数，否则众数可以是一个或者多个。测试：

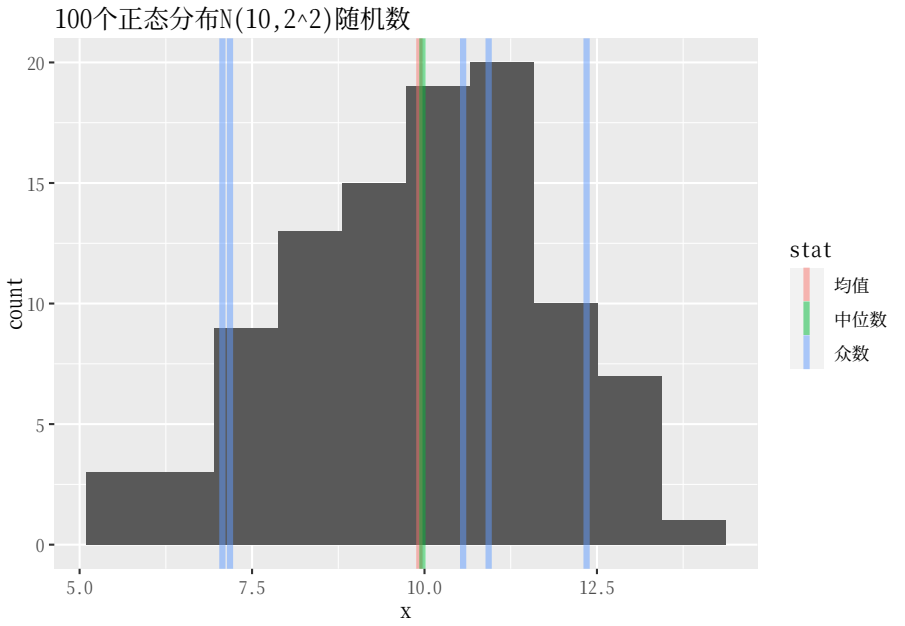
```
set.seed(101)
x <- round(rnorm(100, 10, 2), 2)
mean(x)
## [1] 9.9255
median(x)
## [1] 9.97
sample.mode(x)
## [1]  7.07  7.18 10.56 10.93 12.35
```

有多个众数。

作直方图并显示样本均值、中位数、众数的位置：

```
demo.location <- function(x, title=""){
  d1 <- rbind(
    tibble(x=mean(x), stat=" 均值"),
    tibble(x=median(x), stat=" 中位数"),
    tibble(x=sample.mode(x), stat=" 众数"))
  ggplot(data=tibble(x=x), mapping=aes(x=x)) +
    geom_histogram(bins=10) +
```

```
    geom_vline(data=d1, mapping=aes(
      xintercept=x, col=stat),
      size=1.5, alpha=0.5) +
    labs(title=title)
}
demo.location(x, title="100 个正态分布 N(10,2^2) 随机数")
```



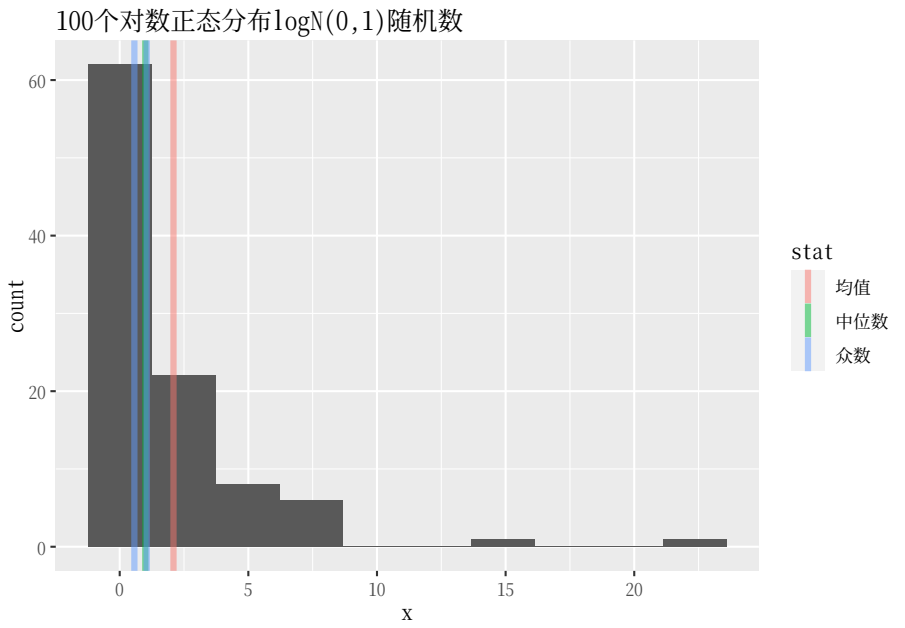
例 3.3 (对数正态样本众数计算 (*))**.** 设 $\ln X \sim \mathrm{N}(\mu, \sigma^2)$，即 X 服从对数正态分布。对模拟样本计算。

取 $\ln X$ 为标准正态分布。

```
set.seed(102)
x <- round(exp(rnorm(100)), 2)
mean(x)
## [1] 2.0901
median(x)
## [1] 0.995
```

```
sample.mode(x)
## [1] 0.57 1.05
```

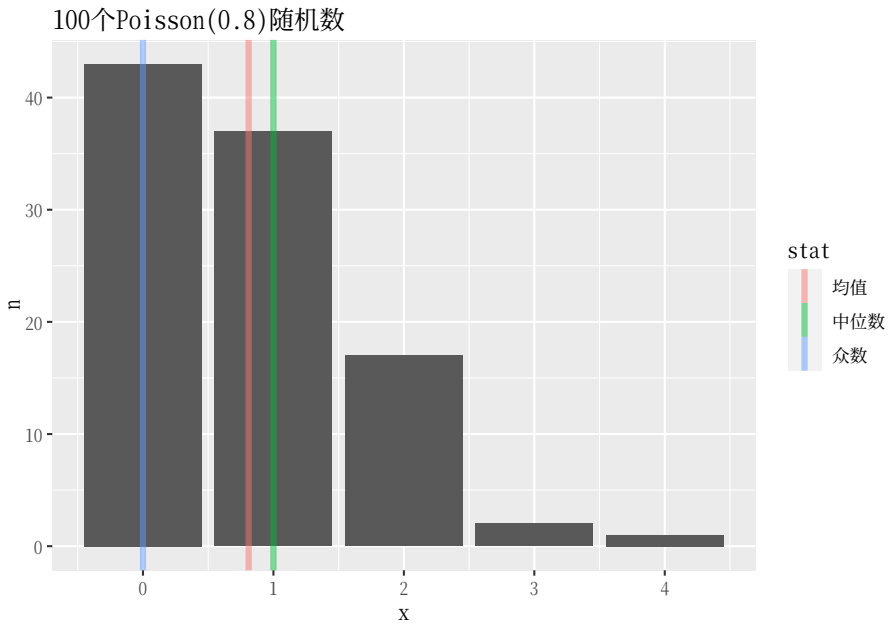
```
demo.location(x, title="100 个对数正态分布 logN(0,1) 随机数")
```



例 3.4 (泊松样本众数计算 (*))**.** Poisson 模拟样本的计算。

```
set.seed(103)
x <- rpois(100, lambda=0.8)
mean(x)
## [1] 0.81
median(x)
## [1] 1
sample.mode(x)
## [1] 0
table(x)
## x
##  0  1  2  3  4
## 43 37 17  2  1
```

```
d1 <- rbind(
  tibble(x=mean(x), stat=" 均值"),
  tibble(x=median(x), stat=" 中位数"),
  tibble(x=sample.mode(x), stat=" 众数"))
d2 <- tibble(x=x) %>%
  count(x)
ggplot(data=d2, mapping=aes(
    x=x, y=n)) +
  geom_col() +
  geom_vline(data=d1, mapping=aes(
    xintercept=x, col=stat),
    size=1.5, alpha=0.5) +
  labs(title="100 个 Poisson(0.8) 随机数")
```



3.2.2 分散程度（变异性）特征量

不同分布之间的差别，除了大小差别（用位置特征代表），还包括分散程度的差别。比如，两个合唱小组的身高分别为 (单位：厘米)

A	159	160	162	160	159
B	150	160	180	160	150

则两个组的平均身高都是 160 厘米，但整齐程度就相差甚远。

可以衡量分散程度的特征量包括标准差、方差、变异系数、极差、四分位间距。

3.2.2.1 方差和标准差

总体 X 的方差为 $\mathrm{Var}(X) = E(X - EX)^2$, 标准差为 $\sigma_X = \sqrt{\mathrm{Var}(X)}$。样本 $x_1, x_2, \dots, x_n$ 的样本方差和样本标准差分别为

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$
$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

其中分母 $n-1$ 在 $x_1, x_2, \dots, x_n$ 是 Y 的独立重复抽样时保证了 S^2 是 $\mathrm{Var}(X)$ 的无偏估计。在 R 语言中用 `var(x)` 求样本方差，用 `sd(x)` 求样本标准差。在 Julia 语言中用 `Statistics.var(x)` 求样本方差，用 `Statistics.std(x)` 求样本标准差。

数据分析中主要使用标准差而不使用方差，这是因为标准差与原来数据具有相同量纲，而方差的量纲与原来数据量纲不同。上面的两个合唱小组的身高标准差分别为 1.225 厘米和 12.25 厘米。

因为方差和标准差基于数学期望，它们会受到极端值的影响。比如，在上面所举的月薪的例子中，标准差为 9851 元，其中经理的月薪值是一个极端值，去掉这个值之后标准差为零。

例 3.5 (方差递推算法 (*))**.** 给出方差递推算法，并用 R 和 Julia 实现。

如果需要递推计算方差，记 $S_k^2 = \frac{1}{k-1}\sum_{i=1}^{k}(x_i - \bar{x}_k)^2$, 可用如下递推算法：

样本方差递推算法

$\bar{x}_1 = x_1;\ S_1^2 = 0$
for $(k$ **in** $2:n)$ {
$S_k^2 = \frac{k-2}{k-1} S_{k-1}^2 + \frac{1}{k}(x_k - \bar{x}_{k-1})^2$
$\bar{x}_k = \frac{k-1}{k}\bar{x}_{k-1} + \frac{1}{k}x_k$
}

(其中 S_1^2 仅作为初值使用)

R 函数：

```
var.it <- function(x){
  n <- length(x)
  xm <- x[1]
  ss <- 0
  for(k in 2:n){
    ss <- (k-2)/(k-1)*ss + (x[k] - xm)^2 / k
    xm <- (k-1)/k*xm + x[k]/k
  }
  ss
}
```

测试：

```
set.seed(1)
x <- round(runif(5), 2); x
## [1] 0.27 0.37 0.57 0.91 0.20
var(x)
## [1] 0.08158
var.it(x)
## [1] 0.08158
```

Julia 版本：

```
function var_iter(x::Vector)
    n = length(x)
    xm::Float64 = x[1]
    ss = 0.0
    for k=2:n
        ss = (k-2)/(k-1)*ss + (x[k] - xm)^2 / k
        xm = (k-1)/k*xm + x[k]/k
    end
    ss
end
```

测试：

```
x = [0.09, 0.83, 0.63, 0.16, 1.0]
var(x)
## 0.16267
var_iter(x)
## 0.16266999999999998
```

※※※※※

3.2.2.2 变异系数

标准差是有量纲的分散程度特征，用**变异系数**

$$\mathrm{CV} = \frac{S}{\bar{x}} \times 100\%$$

可以表示变量相对于其平均值的变化情况，是一个无量纲的量。比如，100 克的标准差，如果是来自平均 500 克一袋的奶粉，变异系数达到 20%，说明此种奶粉的装袋质量很差；如果同样是 100 克的标准差但是来自平均 50 千克一袋的面粉，则变异系数为 0.2%，说明此种面粉的装袋质量很好。

3.2.2.3 极差和四分位间距

样本的**极差**定义为样本的最大值减去样本的最小值，也能比较直观地反映样本值分散程度。

设 $\hat{x}_{1/4}$ 和 $\hat{x}_{3/4}$ 是从样本 $x_1, x_2, \ldots, x_n$ 中估计的分位数 $x_{1/4}$ 和 $x_{3/4}$，称 $\hat{x}_{3/4} - \hat{x}_{1/4}$ 为**四分位间距** (Inter-Quartile Range, IQR，又称为四分位差、内距)。四分位间距是最靠近分布中心的 50% 的样本值所占的范围大小，可以反映样本分散程度，而且不受到极端值影响。

受前面传统的中位数估计方法的启发，我们可以用如下方法计算 1/4 和 3/4 分位数。把样本从小到大排序后，如果 n 为奇数，则用 $x_{(1)}, x_{(2)}, \ldots, x_{(\frac{n+1}{2})}$ 的中位数作为 $x_{1/4}$ 的估计，用 $x_{(\frac{n+1}{2})}, x_{(\frac{n+1}{2}+1)}, \ldots, x_{(n)}$ 的中位数作为 $x_{3/4}$ 的估计。如果 n 为偶数，则用 $x_{(1)}, x_{(2)}, \ldots, x_{(\frac{n}{2})}$ 的中位数作为 $x_{1/4}$ 的估计，用 $x_{(\frac{n}{2}+1)}, x_{(\frac{n}{2}+2)}, \ldots, x_{(n)}$ 的中位数作为 $x_{3/4}$ 的估计。

3.2.3 与分布形状有关的特征量

包括**偏度**w 和**峰度**k。总体 X 的偏度和（超额）峰度分别定义为

$$w = \frac{E(X-EX)^3}{(\mathrm{Var}(X))^{3/2}}, \qquad k = \frac{E(X-EX)^4}{(\mathrm{Var}(X))^2} - 3$$

从样本 $x_1, x_2, \ldots, x_n$ 计算的偏度和峰度分别定义为 (见 [SAS Institute Inc., 2010] 328–329)

$$\hat{w} = \frac{n}{(n-1)(n-2)} \sum_{i=1}^{n} \left(\frac{x_i - \overline{x}}{S} \right)^3$$

$$\hat{k} = \frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=1}^{n} \left(\frac{x_i - \overline{x}}{S} \right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

其中 $\bar{x}$ 和 S 分别为样本平均值和样本标准差。

例 3.6 (样本偏度和样本峰度的程序 (*))**.** 用 R 和 Julia 计算样本偏度和样本峰度。

R 函数：

```
skewness <- function(x){
  n <- length(x)
  (n /(n-1) * n/(n-2)) *
    mean((x - mean(x))^3) / (var(x))^1.5
}
```

```
kurtosis <- function(x){
  n <- length(x)
  ((n+1)/(n-1) * n/(n-2) * n/(n-3)) *
    mean((x - mean(x))^4) / (var(x))^2 -
    3*(n-1)^2/(n-2)/(n-3)
}
```

测试：

```
set.seed(1)
x <- round(rnorm(10, 10, 2), 2); x
## [1]  8.75 10.37  8.33 13.19 10.66  8.36 10.97 11.48 11.15  9.39
skewness(x)
## [1]  0.3516643
kurtosis(x)
## [1] -0.3150973
```

Julia 函数：

```
function skewness(x)
    n = length(x)
    s = 0.0
    xm = mean(x)
    xsd = std(x)
    for k=1:n
        s += (x[k] - xm)^3
    end
    n/(n-1)*n/(n-2)/n*s/std(x)^3
end
function kurtosis(x)
    n = length(x)
    s = 0.0
    xm = mean(x)
    xsd = std(x)
```

```
    for k=1:n
        s += (x[k] - xm)^4
    end
    (n+1)/(n-1)*n/(n-2)/(n-3)*s/std(x)^4 - 3*(n-1)/(n-2)*(n-1)/(n-3)
end
```

测试：

```
x = [8.75, 10.37, 8.33, 13.19, 10.66, 8.36, 10.97, 11.48, 11.15, 9.39]
skewness(x)
## 0.35166432816653354
kurtosis(x)
## -0.3150972802809626
```

※※※※※

偏度 w 反映了分布的对称性。以连续型分布为例，设 X 有分布密度 $p(x)$，若 $p(x)$ 关于 EX 对称，则 $w=0$。如果 $p(x)$ 左右两个尾部不对称，右尾长而左尾短，则 $w>0$，称这样的分布为**右偏**分布。反之，如果 $p(x)$ 的左尾长而右尾短，则 $w<0$, 称这样的分布为**左偏**分布。

峰度 k 反映了分布的两个尾部的衰减速度情况。如果 $p(x)$ 在 $y\to\pm\infty$ 衰减速度较慢（比如 $p(x)=O(\frac{1}{x^p}),\ p>1$），则称 $p(x)$ 是**重尾 (厚尾) 分布**, 这时 $k>0$。对于正态分布 $\mathrm{N}(\mu,\sigma^2)$，$k\equiv 0$。重尾分布的样本会有比较多的极端值，即与分布中心距离很远的值。

3.2.4 相关系数

对于多个随机变量，可以用**相关系数**来简单描述两两之间的关系。随机变量 X 和 Y 的相关系数定义为

$$\rho=\rho(X,Y)=\frac{E[(X-EX)(Y-EY)]}{\sqrt{\mathrm{Var}(X)\cdot\mathrm{Var}(Y)}}$$

这是一个取值于 $[-1,1]$ 的数，当 $|\rho|$ 接近于 1 时代表 X 和 Y 有很强的线性相关关系。对于 X 和 Y 的一组样本 $(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)$，相关系数估

计为

$$R = \frac{\sum_{i=1}^{n}[(x_i - \bar{x})(y_i - \bar{y})]}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \cdot \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

也满足 $-1 \le R \le 1$。

注意，相关系数只考虑了两个变量之间的线性相关性。例如，$X \sim \mathrm{N}(0,1)$，$Y = X^2$ 是 X 的函数，但是 X 和 Y 的相关系数等于零。实际数据中，样本相关系数绝对值很小不表明两个变量相互独立；样本相关系数绝对值较大，则两个变量之间很可能是不独立的，但不能仅靠相关系数确认两个变量之间有线性相关性：非线性的相关也可能导致相关系数绝对值较大。

如果我们有 p 个随机变量 $(X_1, X_2, \dots, X_p)^T$(T 表示矩阵转置)，看成随机向量 X，X 的期望为 $EX = (EX_1, EX_2, \dots, EX_p)^T$，协方差阵为 $\mathrm{Var}(X) = E[(X - EX)(X - EX)^T]$, $\mathrm{Var}(X)$ 的第 k 行第 j 列元素为

$$\mathrm{Cov}(X_k, X_j) = E[(X_k - EX_k)(X_j - EX_j)],\ k, j = 1, 2, \dots, p.$$

X 的相关阵 R 是 X 的各分量的相关系数组成的矩阵，主对角线元素等于 1，第 k 行第 j 列元素为 $\rho(X_k, X_j)$。

对 X 进行 n 次独立观测得到样本 $(x_{i1}, x_{i2}, \dots, x_{ip})$, $i = 1, 2, \dots, n$, 用

$$\bar{x}_{\cdot j} = \frac{1}{n}\sum_{i=1}^{n} x_{ij},\ j = 1, 2, \dots, p$$

估计 EX，用

$$\gamma_{kj} = \frac{1}{n-1}\sum_{i=1}^{n}(x_{ik} - \bar{x}_{\cdot k})(x_{ij} - \bar{x}_{\cdot j})$$

估计 $\mathrm{Cov}(X_k, X_j)$, 矩阵 $(\gamma_{kj})_{k,j=1,2,\dots,p}$ 叫做样本协方差阵，用来估计 X 的协方差阵。X 的样本相关系数组成的矩阵叫做样本相关系数阵。

在 R 中，如果 `x` 是一个 $n \times p$ 的矩阵或者数据框，每行看作 p 元随机向量的独立观测，则 `var(x)` 计算其样本协方差阵，`cor(x)` 计算其样本相关系数阵。

在 Julia 中用 `Statistics.cov(X)` 计算样本协方差阵,用 `Statistics.cor(X)` 计算样本相关系数矩阵，用 `Statistics.cor(x, y)` 计算样本相关系数。

习题

习题 1

编写 R 函数 `skewness(x)`, 输入向量 `x` 的值后，按如下公式

$$\phi = \frac{n}{(n-1)(n-2)} \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{s} \right)^3$$

计算样本偏度，其中 n 为 `x` 的元素个数，$\bar{x}$ 和 s 分别为 `x` 中元素的样本平均值和样本标准差。

习题 2

为了求样本分位数，修改样本经验分布函数为

$$\tilde{F}_n(x_{(i)}) = \frac{i - \frac{1}{3}}{n + \frac{1}{3}},\ i = 1, 2, \ldots, n$$

并把相邻点用线段相连来定义 $\tilde{F}_n(x)$，求 $\tilde{F}_n(x) = p$ 得到 x_p 的估计。编写求样本分位数的程序程序，并用 R 的 `quantile()` 函数验证。

习题 3

证明第3.2.1和3.2.2中的均值和方差的递推算法。

习题 4

把均值和方差的递推算法推广到随机向量的均值和协方差阵计算问题。

Chapter 4

统计图形

图形比描述统计量更能直观、全面地反映变量分布和变量之间的关系。直方图、密度估计图、盒形图、茎叶图、QQ 图可以用来查看单个变量分布情况；散点图可以用来查看两个变量的变化关系; 三维曲面图、等值线图、数字图像可以反映自变量 x, y 与因变量 z 的关系。

R 软件的统计图形详见http://www.math.pku.edu.cn/teachers/lidf/docs/Rbook/html/_Rbook/index.html的第 28——30 章，Julia 语言的作图详见http://www.math.pku.edu.cn/teachers/lidf/docs/Julia/JuliaStat2.html, http://www.math.pku.edu.cn/teachers/lidf/docs/Julia/JuliaPlotGadfly.html, http://www.math.pku.edu.cn/teachers/lidf/docs/Julia/JuliaPlotPlots.html。

4.1 直方图

用描述统计量可以把连续型分布的最重要的特征概括表示，但是，为了了解分布密度的形状，应该对分布密度 $p(y)$ 进行估计。比如，分布密度有两个峰的现象就是上面叙述的描述统计量无法发现的。

直方图 (histogram) 是最简单的估计分布密度的方法。

设随机变量 $Y \sim f(y)$, 分布密度 $f(y)$ 连续, $y_1, y_2, \dots, y_n$ 为 Y 的简单随机样本。取分点 $t_0 < t_1 < \cdots < t_m$，把 Y 的取值范围分为 m 个小区间: $\cup_{k=1}^{m}(t_{k-1}, t_k]$;

令

$$\begin{aligned} p_k =& P\{t_{k-1} < Y \le t_k\} \\ =& \int_{t_{k-1}}^{t_k} f(y)dy = f(\xi_k)(t_k - t_{k-1}),\ k = 1, 2, \dots, m, \end{aligned}$$

其中 $\xi_k \in (t_{k-1}, t_k]$。设样本值 $y_1, y_2, \dots, y_n$ 落入第 k 个小区间 $(t_{k-1}, t_k]$ 的个数为 u_k(称为**频数**), 用 u_k/n 估计 p_k, 称 u_k/n 为样本落入第 k 个小区间的频率或**百分比**。当 $n \to \infty$ 时，取分点 $\{t_k\}$ 使小区间长度趋于零，即

$$d = \max\{t_k - t_{k-1} : k = 1, 2, \dots, m\} \to 0,$$

如果 Y 的取值范围没有下界, 则要求 $t_0 \to -\infty$; 如果 Y 的取值范围没有上界, 则要求 $t_m \to +\infty$。这时有 $u_k/n \to p_k$, $k = 1, 2, \dots, m$, a.s., 于是

$$\left| \frac{u_k}{n(t_k - t_{k-1})} - f(y) \right| \to 0,\ \text{a.s.},\ y \in (t_{k-1}, t_k],$$

即 $f(x)$ 可以用 x 所在的小区间上的频率除以小区间长度来估计。令

$$f_n(y) = \begin{cases} \sum_{k=1}^{m} \frac{u_k/n}{t_k - t_{k-1}} I_{(t_{k-1}, t_k]}(y), & \text{当 } t_0 < y \le t_m \\ 0, & \text{当 } y \le t_0 \text{ 或 } y > t_m, \end{cases}$$

则 $n \to \infty$ 时 $f_n(y) \to f(y)$, a.s.。另外，$f_n(y) \ge 0, y \in (-\infty, \infty)$, $\int_{-\infty}^{\infty} f_n(y)dy = 1$, 满足密度函数的一般要求。

根据上述讨论，我们可以用如下的等距概率直方图来估计分布密度 $f(y)$。确定区间端点 (a, b) 使得样本值 $y_1, y_2, \dots, y_n$ 都落入 (a, b) 内，确定分组数 m，令组距 $h = \frac{b-a}{m}$, 分点

$$t_k = a + kh,\ k = 0, 1, \dots, m,$$

计算样本值落入第 k 个小区间的个数 u_k 和频率 u_k/n。以 $(t_{k-1}, t_k]$ 为底，$(u_k/n)/(t_i - t_{i-1}) = (u_k/n)/h$ 为高画长方形, 这 m 个长方形的顶部为密度估计值。

另一种等距直方图用小区间对应的频数 u_k 作为这些长方形的高度，这样的等距直方图不是密度函数的估计，但是与等距概率直方图只差常数倍 nh。

那么，如何确定分组数 m 呢？这个问题没有公认的做法。样本量 n 大时组数应当大，数据取值范围小时组数应当小，数据有效位少使得不同值少时组数应当小，一般应使得每一小区间不为空，观测数达到 5 以上更好。实际取 m 和分点时，还需要使得分点的小数位数较少。

按照 AMISE(平方误差积分渐近期望) 最小原则，可取区间长度 h 为

$$h = 3.49\sigma n^{-1/3},$$

其中总体标准差 σ 可用样本标准差代替，或者用 IQR 来估计 σ，得

$$h = 2 \cdot \mathrm{IQR} \cdot n^{-1/3}.$$

按绝对误差一致最小准则，可取

$$h = 1.66S \left(\frac{\ln n}{n}\right)^{1/3}$$

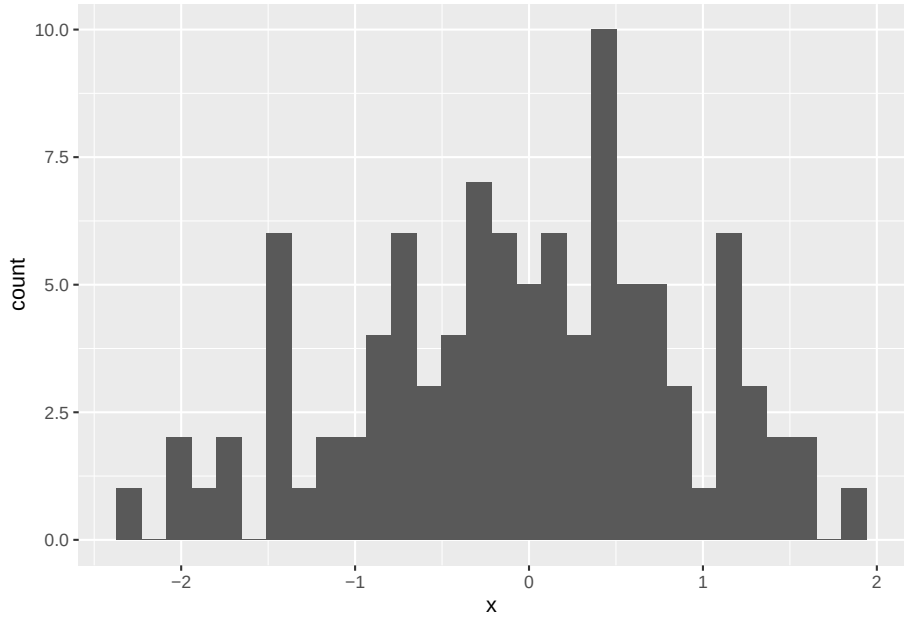
其中 S 是样本标准差。

关于直方图分组数 m 选取的讨论参见 [Gentle, 2002]§9.2。

在 R 软件中，用 `hist(x, freq=FALSE)` 作估计分布密度的等距概率直方图，分组按默认规则分组 (见 Sturges(1926))；或者用 `MASS::truehist()` 函数。但是，ggplot2 包的作图功能更灵活、强大、美观，下面尽可能使用 ggplot2 包作图。

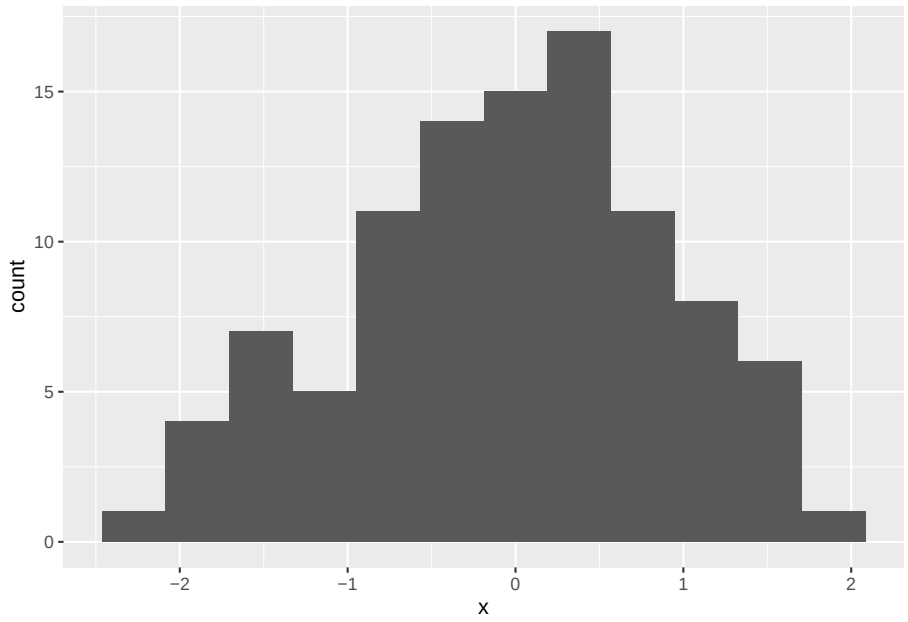
下面的程序首先生成了 100 个标准正态分布随机数（可以理解为来自标准正态分布的样本量为 100 的一组简单随机样本）, 然后作了等距概率直方图:

```
set.seed(101); x <- rnorm(100)
d <- tibble(x = x)
ggplot(data=d, mapping=aes(x=x)) +
  geom_histogram()
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



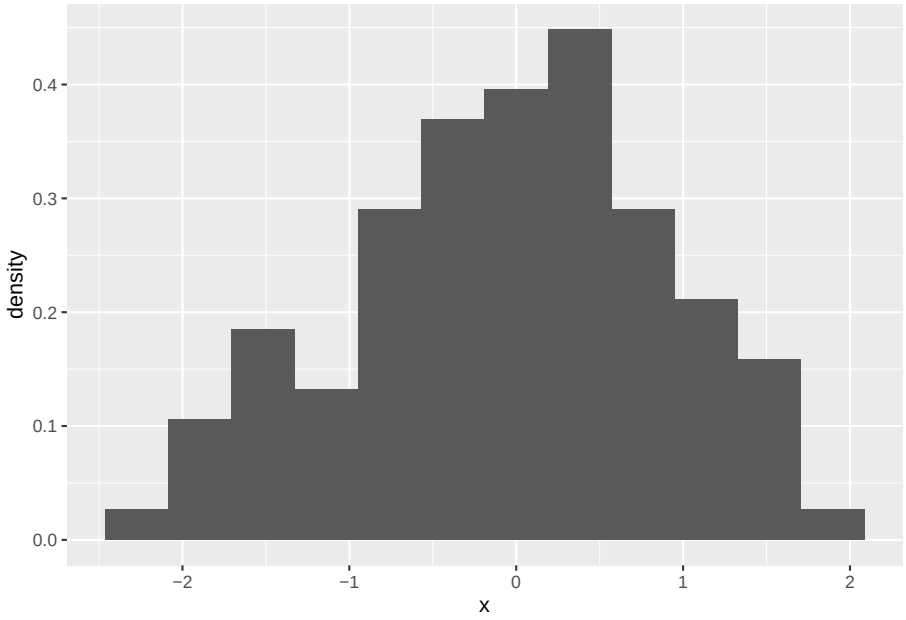
可以用 `bins` 指定分组数：

```
ggplot(data=d, mapping=aes(x=x)) +
  geom_histogram(bins=12)
```



每组条形高度默认使用该组的观测个数，可以用 `y=..density..` 指定使用密度估计：

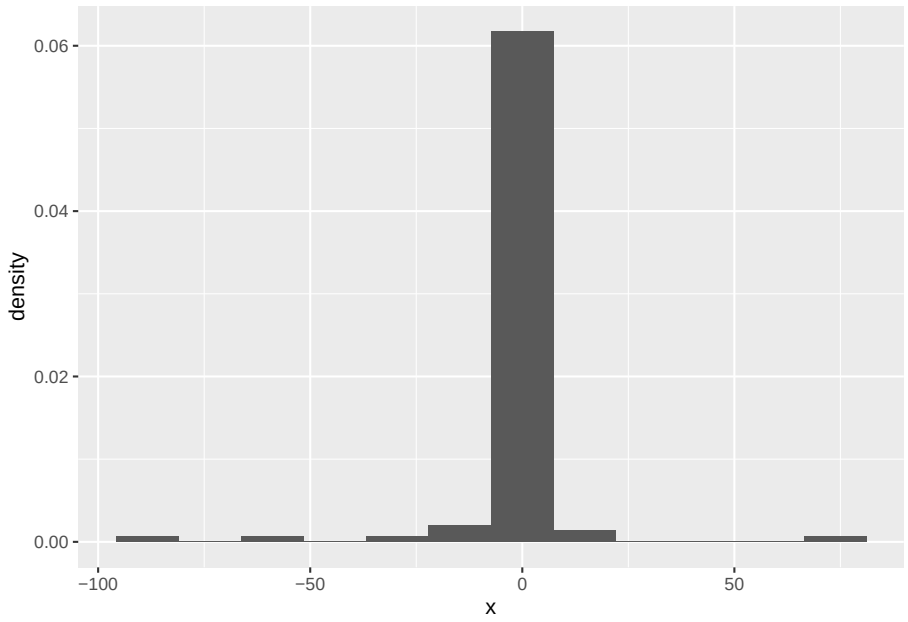
```
ggplot(data=d, mapping=aes(x=x, y=..density..)) +
  geom_histogram(bins=12)
```



对于正态分布密度这样的非重尾的密度，等距概率直方图可以比较好地反映分布密度形状。但是，如果分布在两侧或一侧有重尾，这时直方图的部分小区间可能只有很少点甚至没有点，部分小区间则集中了过多的点，等距概率直方图就不能很好地反映分布密度形状。

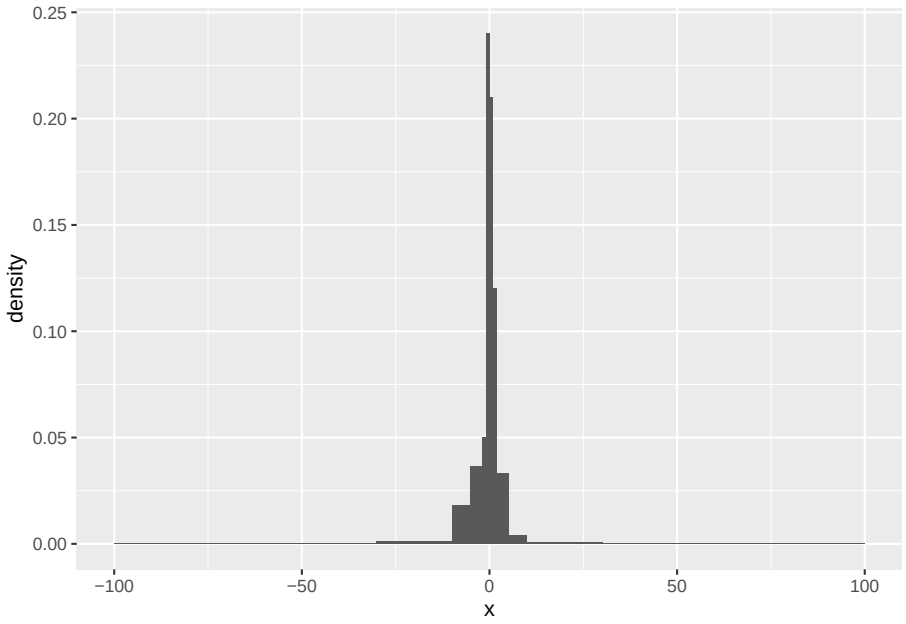
如下生成了 100 个柯西分布随机数，然后对这组样本做了等距概率直方图，可以看出效果较差:

```
set.seed(103); x <- rcauchy(100)
d <- tibble(x = x)
ggplot(data=d, mapping=aes(x=x, y=..density..)) +
  geom_histogram(bins=12)
```



这种情况下可以把频数少的小区间合并，把频数过大的小区间拆分，产生不等距的概率直方图。比如，上面的柯西分布随机数可以用如下程序得到较好的直方图：

```
ggplot(data=d, mapping=aes(x=x, y=..density..)) +
  geom_histogram(breaks=c(
    -100, -50, -30,  -10, -5, -2, -1, 0,
    1, 2, 5, 10, 30, 100))
```



4.2 核密度估计

在等距概率直方图作法中，估计 y_0 处的密度 $p(y_0)$ 时其所在区间 $(t_{k-1}, t_k]$ 的所有 $p(y)$ 都估计成同一个值，这些分点 $\{t_k\}$ 是预先取好的，与要估计 $p(y)$ 的自变量 y 的位置无关。我们可以针对每一个 y，以 y 为中心、$\frac{h}{2}$ 为半径做小区间 $\left[y-\frac{h}{2}, y+\frac{h}{2}\right]$，用

$$\tilde{p}(y) = \frac{\#\left(\left\{y_i : y_i \in \left[y-\frac{h}{2}, y+\frac{h}{2}\right]\right\}\right)/n}{h}$$

来估计 $p(y)$，其中 $\#(A)$ 表示集合 A 的元素个数。上式可以写成

$$\begin{aligned}\tilde{p}(y) =& \frac{1}{h}\frac{1}{n}\sum_{i=1}^{n} I_{\left[y-\frac{h}{2}, y+\frac{h}{2}\right]}(y_i) \\ =& \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h} I_{\left[-\frac{1}{2},\frac{1}{2}\right]}\left(\frac{y-y_i}{h}\right),\ y \in (-\infty, \infty).\end{aligned}$$

这个分布密度估计叫做 Rosenblatt 直方图估计。如果把上面的区间改为左开右闭区间 $\left(y-\frac{h}{2}, y+\frac{h}{2}\right]$, $\tilde{p}(y)$ 与经验分布函数 $F_n(y)$ 恰好满足如下关系：

$$\tilde{p}(y) = \frac{F_n\left(y+\frac{h}{2}\right) - F_n\left(y-\frac{h}{2}\right)}{h},\ y \in (-\infty, \infty).$$

即 $\tilde{p}(y)$ 是对经验分布函数用差商近似估计 $F(x)$ 导数的结果。

设函数 $K_1(x) = I_{[-\frac{1}{2},\frac{1}{2}]}(x)$, Rosenblatt 直方图估计可以写成

$$\tilde{p}(y) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K_1\left(\frac{y-y_i}{h}\right),$$

这样形式的密度估计叫做**核密度估计**，$K_1(x)$ 叫做核函数。

一般地，核函数应该满足如下要求:

1. $K(x) \geq 0$;
2. $\int_{-\infty}^{\infty} K(x)dx = 1$;
3. $K(x)$ 是偶函数;
4. $\int_{-\infty}^{\infty} x^2K(x)dx < +\infty$。

一般核密度估计为

$$\hat{p}(y) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K\left(\frac{y-y_i}{h}\right)$$

$K(x)$ 的条件 (1) 和 (2) 使得 $\hat{p}(y)$ 满足 $\hat{p}(y) \geq 0, y \in (-\infty,\infty)$, $\int_{-\infty}^{\infty}\hat{p}(y)dy = 1$。当 h 很小时，密度估计很不光滑，在每个 y_i 处有一个尖锐的峰而没有观测值的地方密度估计为零，估计偏差小而方差大。当 h 较大时，估计比较光滑，估计偏差大而方差小。渐近地取 $h = O(n^{-1/5})$, 核密度估计的均方误差为 $O(n^{-4/5})$, 优于直方图估计，参见 [Gentle, 2002] §9.3。

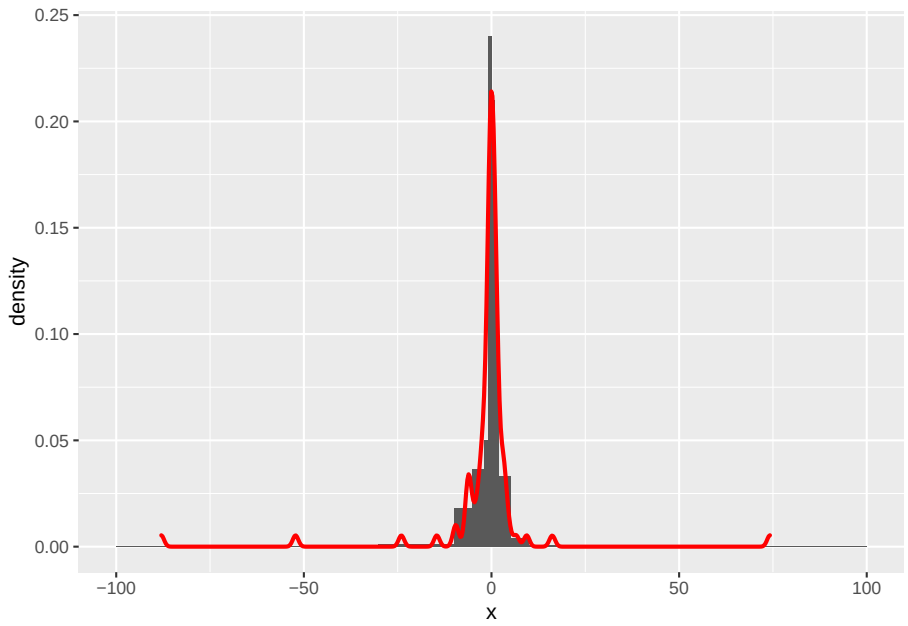
注意，核密度估计可以很容易地推广到随机向量联合分布密度的估计。

其它核函数还有：

$$\begin{aligned}
K_2(x) =& (1-|x|)I_{[-1,1]}(x)(\text{三角形核})\\
K_3(x) =& \frac{3}{4}(1-x^2)I_{[-1,1]}(x)(\text{二次曲线核})\\
K_4(x) =& \frac{15}{16}(1-x^2)^2I_{[-1,1]}(x)(\text{双二次核})\\
K_5(x) =& \frac{70}{81}(1-|x|^3)^3I_{[-1,1]}(x)(\text{双三次核})\\
K_6(x) =& \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}(\text{高斯核})
\end{aligned}$$

R 软件中函数 `density(x)` 进行核密度估计。ggplot2 包用 `geom_density()` 画核密度估计曲线。上面例子中柯西分布随机数的直方图可以用如下程序增加核密度估计曲线：

```
ggplot(data=d, mapping=aes(x=x, y=..density..)) +
  geom_histogram(breaks=c(
    -100, -50, -30,  -10, -5, -2, -1, 0,
    1, 2, 5, 10, 30, 100)) +
  geom_density(color="red", size=1)
```

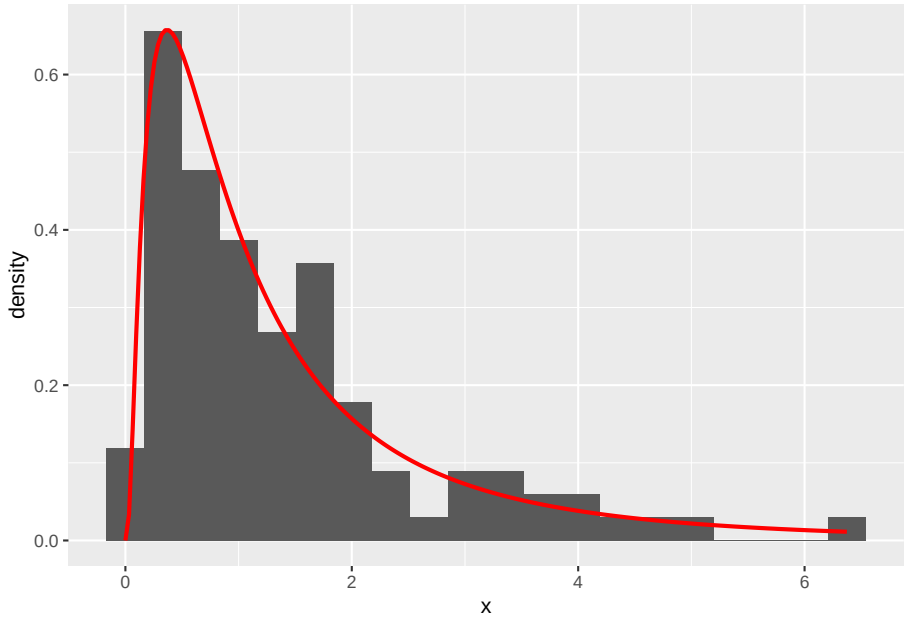


4.3 盒形图

对于单峰的分布密度，**盒形图** (boxplot) 可以概括地表现其分布情况。

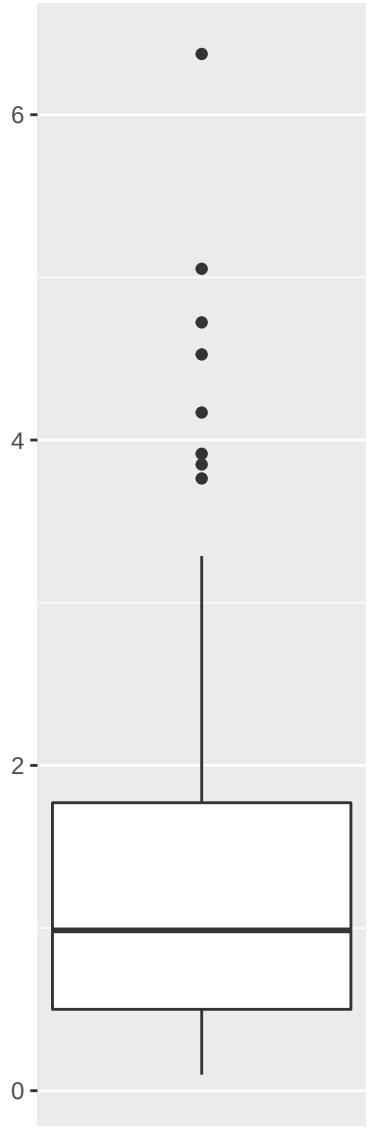
100 个对数正态分布随机数的直方图，直方图上叠加了估计的对数正态密度曲线：

```
set.seed(101)
x <- exp(rnorm(100))
d <- tibble(x=x)
d1 <- tibble(x = seq(0, max(x), length=200)) %>%
  mutate(y=dlnorm(x), mean(log(x)), sd(log(x)))
ggplot(data=d, mapping=aes(x=x, y=..density..)) +
  geom_histogram(bins=20) +
  geom_line(data=d1, mapping=aes(x=x, y=y), col="red", size=1)
```



盒形图：

```
ggplot(data=d, mapping=aes(x=1, y=x)) +
  geom_boxplot() +
  scale_x_continuous(breaks=NULL) +
  labs(x=NULL, y=NULL)
```



对数正态分布是右偏的单峰分布。盒形图中间有条粗线，位于分布的中位数上；盒子的下边缘和上边缘分别位于分布的 1/4 分位数和 3/4 分位数处，所以盒形的范围内包括了样本值分布从小到大排列中间的 1/2 的样本值。在盒子两端的外侧分别向下和向上延伸出两条线，叫做**触须线**，触须线长度小于等于 1.5 倍 IQR(即盒子高度) 并最长只能延伸到最小值或最大值的地方。如果有样本值超出了触须线的范围，就用散点符号画出，这些样本值距离分布中心较远，称为

离群值 (outliers)。如果有较多、较远的离群值，这个样本分布就是重尾的。

所以，我们能够从盒形图上看到中位数、1/4 分位数、3/4 分位数、最小值、最大值的位置，并能发现离群值，判断分布左端和分布右端是否有重尾。另外，比较中位数上方和中位数下方的长度可以发现分布是对称的、左偏的，还是右偏的。上图中的盒形图是右偏的。

在 R 软件中，用 `boxplot(x)` 作盒形图。在 ggplot2 中用 `geom_boxplot()` 作盒形图，但是需要有 `x` 变量，可以定义 `x` 变量为常数，`y` 变量为要作图的变量。盒形图更常用于并排作多个，比较不同类别的观测，或者比较若干个变量。

4.4 茎叶图

茎叶图是一种低精度的图形，用来检查数据输入和分布形状。在 R 软件中用 `stem(x)` 作茎叶图。

如下程序输入了若干个人的身高（单位: 厘米）并作了茎叶图，

```
y <- c(145, 150, 155, 156, 159, 160, 166, 170, 190)
stem(y)
```

结果如下

```
The decimal point is 1 digit(s) to the right of the |

14 | 5
15 | 0569
16 | 06
17 | 0
18 |
19 | 0
```

图形分为竖线左边的 “茎” 和右边的 “叶” 两部分。茎部分构成了纵坐标轴，小数点位置在竖线的地方，但是根据数据范围的不同可能需要调整，比如上图的说明部分约定了小数点位置是向竖线右方移动一位（即乘以 10）。叶部分的每

个数字是一片叶子，代表了一个样本点，叶子是样本值最后一个有效数字（或四舍五入后的值）。比如，茎 “15” 右边有 5 片叶子，分别代表样本值 150、155、156、159。

把这个图逆时针旋转 90 度来看，可以看成是一个直方图，某个茎右侧的叶子越多，分布密度在这个值附近越大。

图中的 190 明显远离了其它值。由此可见，从茎叶图可以比较容易地发现离群点。

4.5 正态 QQ 图和正态概率图

在许多统计模型中都要假设模型中的随机变量服从正态分布。对随机变量 Y，设 $y_1, y_2, \ldots, y_n$ 是 Y 的简单随机样本，如何判断 Y 是否服从正态分布？

可以使用假设检验的方法，零假设为 Y 服从正态分布，对立假设为 Y 不服从正态分布。这样的检验有 Shapiro-Wilk 检验等。

另外，我们也可以从分布图形来直观地判断。直方图与核密度估计图可以直接查看分布密度形状是否与正态分布相近。

正态分布随机数的正态 QQ 图:

```
set.seed(101)
x <- rnorm(100)
d <- tibble(x=x)
ggplot(data=d, mapping=aes(sample=x)) +
  stat_qq() +
  geom_qq_line(color="red")
```

对数正态分布随机数的正态 QQ 图:

```
set.seed(101)
x <- rlnorm(100)
d <- tibble(x=x)
ggplot(data=d, mapping=aes(sample=x)) +
```

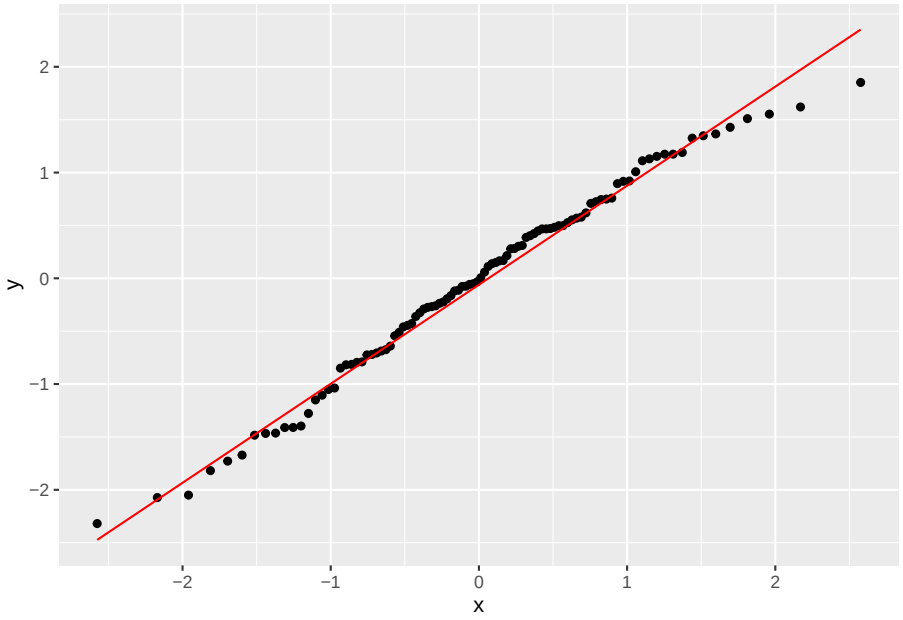


图 4.1: 正态样本的正态 QQ 图

```
  stat_qq() +
  geom_qq_line(color="red")
```

这样的形状表示右偏分布。

指数分布随机数相反数的正态 QQ 图:

```
set.seed(101)
x <- -rexp(100)
d <- tibble(x=x)
ggplot(data=d, mapping=aes(sample=x)) +
  stat_qq() +
  geom_qq_line(color="red")
```

这样的形状表示左偏分布。

t(2) 分布随机数的正态 QQ 图:

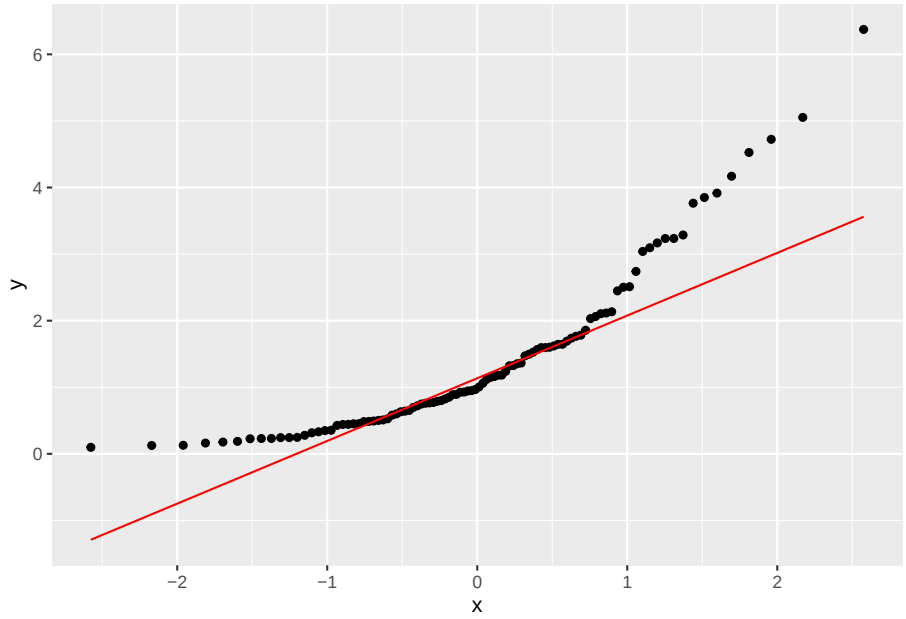


图 4.2: 对数正态样本的正态 QQ 图

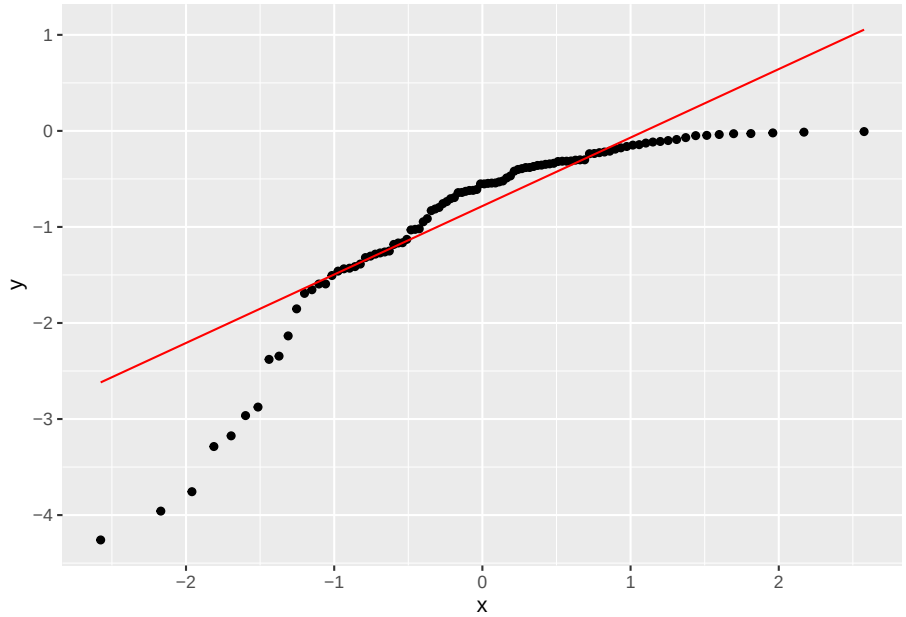


图 4.3: 指数分布相反数样本的正态 QQ 图

```
set.seed(101)
x <- rt(100, 2)
d <- tibble(x=x)
ggplot(data=d, mapping=aes(sample=x)) +
  stat_qq() +
  geom_qq_line(color="red")
```

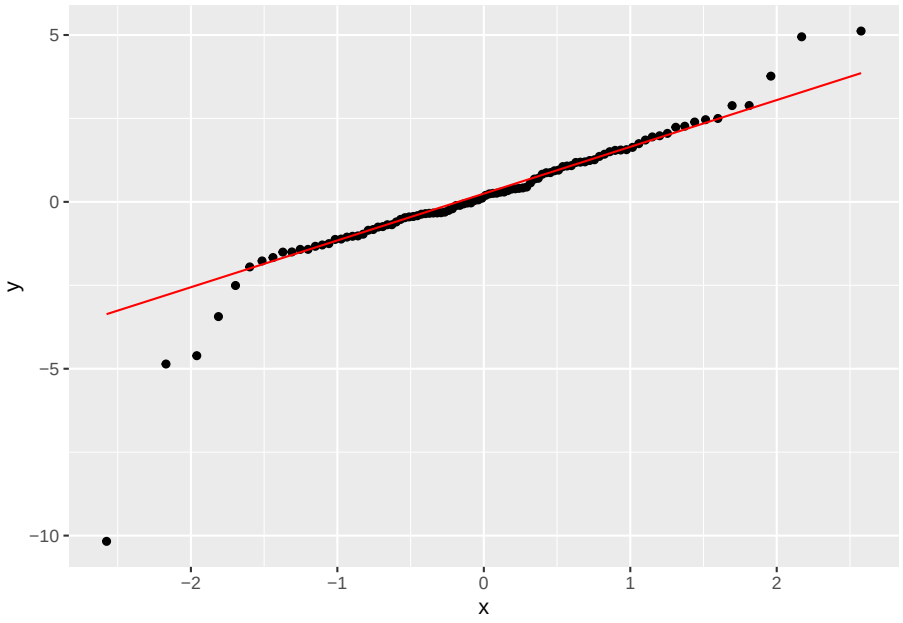


图 4.4: t 分布样本的正态 QQ 图

这样的形状表示分布两侧有厚尾现象。

上面四个图形叫做**正态 QQ 图**，对应的样本分别来自为正态分布、对数正态分布、指数分布的相反数、t(2) 分布的随机数。

正态 QQ 图包括一组散点和一条直线。设样本次序统计量为 $y_{(1)}, y_{(2)}, \cdots, y_{(n)}$，则 $y_{(i)}$ 可以作为是总体的 i/n 分位数的估计。设标准正态分布的分布函数为 $\Phi(\cdot)$，令 $x_i = \Phi^{-1}(i/n)$，则 $(x_i, y_{(i)})$ 分别是标准正态分布 i/n 分位数和样本的 i/n 分位数，以 $(x_i, y_{(i)}), i = 1, 2, \ldots, n$ 为坐标在直角坐标系中作图就得到了正态 QQ 图中的散点。通过代表 1/4 分位数和 3/4 分位数的两个点作直线，

就是正态 QQ 图中的那些直线。

如果样本来自正态 $N(\mu,\sigma^2)$, 记 $\mathrm{N}(\mu,\sigma^2)$ 的分布函数为 $F(x)$，易见 $F(x)=\Phi\left(\frac{x-\mu}{\sigma}\right)$。因为 $y_{(i)}$ 是样本的 i/n 分位数，应该近似等于 $F(x)$ 的 i/n 分位数 $F^{-1}(i/n)$，所以

$$\begin{aligned} y_{(i)} \approx & F^{-1}(i/n) \\ F(y_{(i)}) \approx & \frac{i}{n} \\ \Phi\left(\frac{y_{(i)}-\mu}{\sigma}\right) \approx & \frac{i}{n} \\ \frac{y_{(i)}-\mu}{\sigma} \approx & \Phi^{-1}(i/n)=x_i \\ y_{(i)} \approx & \mu+\sigma x_i \end{aligned}$$

可见这时正态 QQ 图中的散点应该落在以 μ 为截距、以 σ 为斜率的直线附近。图4.1就是这样的情况。

上述讨论中我们取 $y_{(i)}$ 作为总体的 i/n 分位数的估计，这样，最小值 $y_{(1)}$ 代表 $1/n$ 分位数而最大值 $y_{(n)}$ 代表 100% 分位数，两侧不对称。实际作图时，$y_{(i)}$ 对应的概率 p 可能会进行一些修正；比如，SAS 软件中的正态 QQ 图把 $y_{(i)}$ 作为 $(i-\frac{3}{8})/(n+\frac{1}{4})$ 分位数的估计。

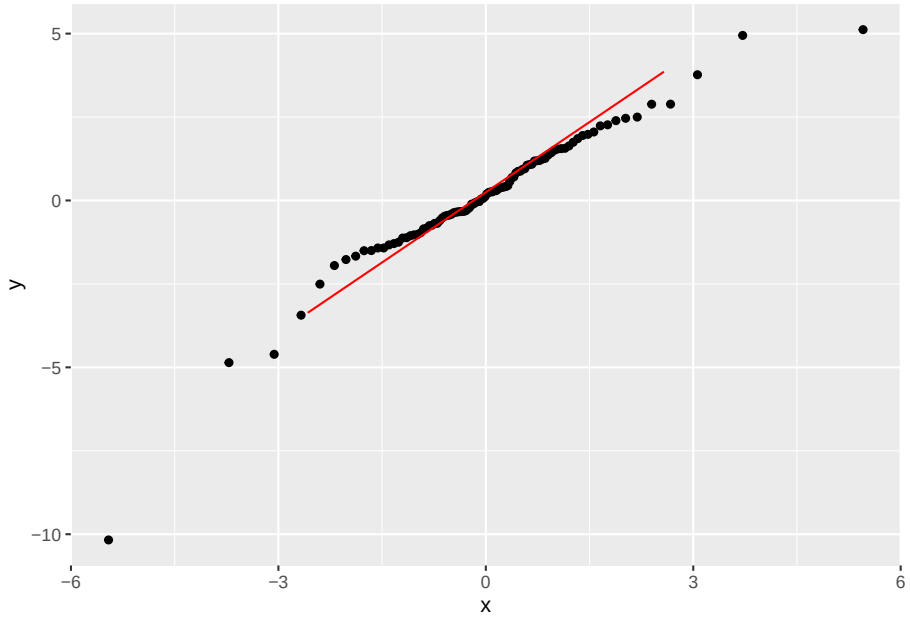
如果样本来自于右偏的分布，正态 QQ 图就会呈现出下凸形状（见图4.2）。这是因为，图中直线上端代表了正态分布情况下在 3/4 分位数之后应有的走势，图中散点的上端高于代表正态分布的直线，说明实际数据的分布密度的右端比正态分布密度要长；此图的左端的散点高于代表正态分布的直线，说明实际数据的分布密度的左端比正态分布密度要短。这正是右偏分布的典型特征。

类似地，图4.3是典型的左偏分布的形状，而图4.4则是分布基本对称但两端的尾部都比正态分布长，从而是对称重尾分布的典型形状。

正态 QQ 图的一个变种是正态概率图，这时散点坐标和直线都不变，但是横坐标轴的刻度不是 QQ 图中 x_i 的值，而是标为 $\Phi(x_i)$ 的值，即 (x_i,y_i) 所对应的概率值。

还可以针对用其它理论分布的分位数为 x 轴作 QQ 图，例如，针对 $\mathrm{t}(2)$ 分布的 QQ 图：

```
params <- as.list(MASS::fitdistr(d$x, "t")$estimate)
## Warning in log(s): NaNs produced
ggplot(data=d, mapping=aes(sample=x)) +
  stat_qq(distribution=qt, dparams=params["df"]) +
  geom_qq_line(color="red")
```



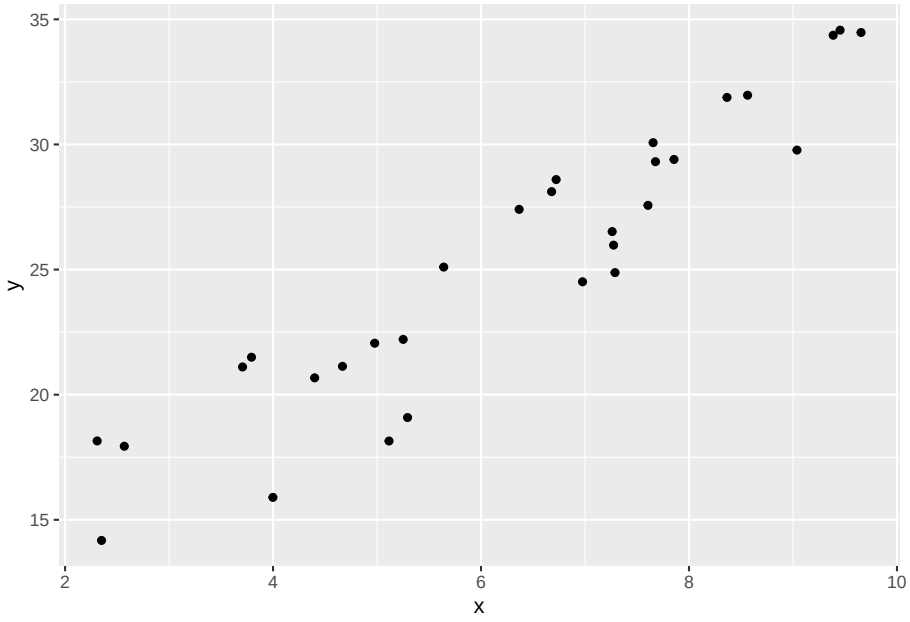
4.6 散点图和曲线图

设有两个变量 X 和 Y 以及它们的 n 组观测值 $(x_i, y_i), i = 1, 2, \ldots,$。以 (x_i, y_i) 为坐标画点，把这 n 组观测值画在平面直角坐标系中，叫做**散点图**。

散点图能够比较直观地表现两个变量之间的关系，两个变量的取值范围，取值的聚集、分组，离群点等情况。

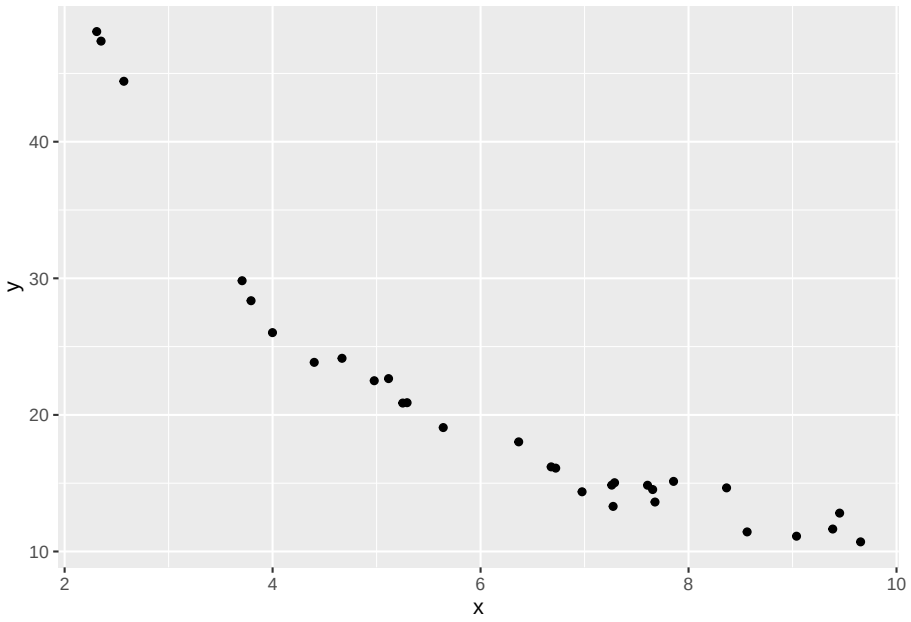
```
set.seed(101); n <- 30
d <- tibble(x = runif(n, 2, 10)) %>%
  mutate(y = 10 + 2.5*x + 2*rnorm(n))
```

```
ggplot(data=d, mapping=aes(x=x, y=y)) +
  geom_point()
```



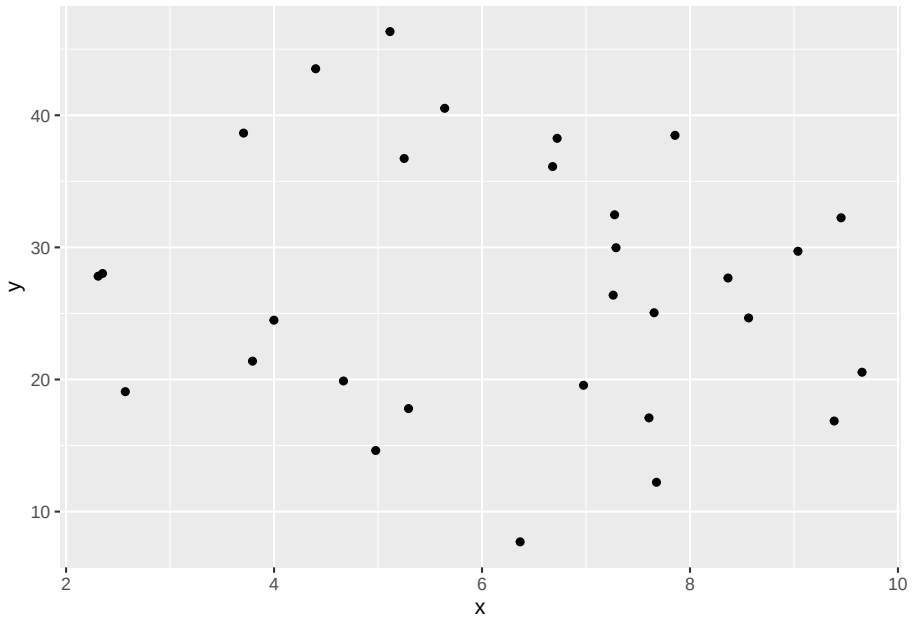
上图中的 X 和 Y 有线性相关关系，且 X 增加 Y 也倾向于增加。

```
d2 <- d %>%
  mutate(y = 110 / x + 1*rnorm(n))
ggplot(data=d2, mapping=aes(x=x, y=y)) +
  geom_point()
```



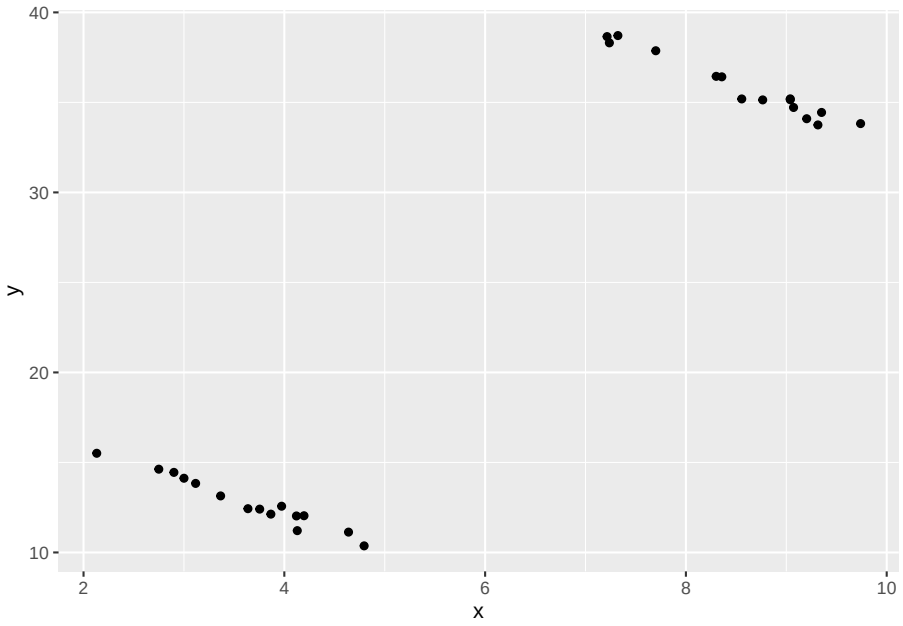
上图中的 X 和 Y 有非线性相关关系，X 增加时 Y 倾向于减少。

```
d3 <- d %>%
  mutate(y = rnorm(n, 25, 10))
ggplot(data=d3, mapping=aes(x=x, y=y)) +
  geom_point()
```



上图中的 X 和 Y 没有明显的相关，也没有明显的聚集模式。

```
set.seed(101); n <- 30
d4a <- tibble(x = runif(n/2, 2, 5)) %>%
  mutate(y = 20 - 2*x + 0.3*rnorm(n/2))
d4b <- tibble(x = runif(n/2, 7, 10)) %>%
  mutate(y = 53 - 2*x + 0.3*rnorm(n/2))
d4 <- rbind(d4a, d4b)
ggplot(data=d4, mapping=aes(x=x, y=y)) +
  geom_point()
```

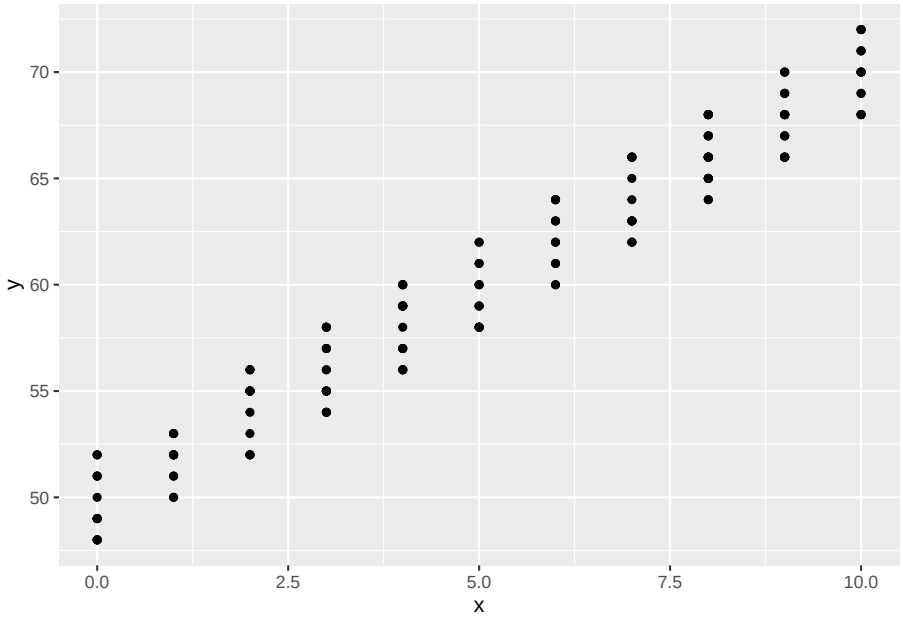


上图中的 X 和 Y 的观测值明显地分成两组，且在每一组内部 X 和 Y 的值是负相关的。注意，如果对第四组数据不画图而直接作线性回归，结果也会有显著的线性相关，但是 X 和 Y 会被误认为有正相关。

在 R 软件中用 `plot(x, y)` 画散点图。ggplot2 用 `geom_point()` 画散点图。

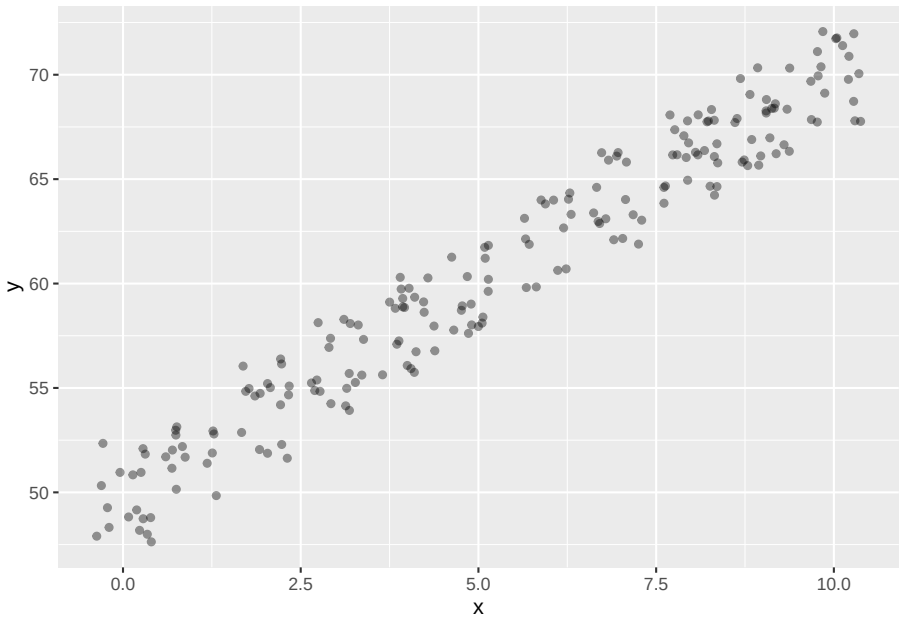
如果 x 或者 y 变量取离散值，则散点图中的点可能有重叠，如：

```
set.seed(101); n <- 200
d5 <- tibble(x = sample(0:10, size=n, replace=TRUE)) %>%
  mutate(y = 50 + 2*x + sample((-2):2, size=n, replace=TRUE))
ggplot(data=d5, mapping=aes(x=x, y=y)) +
  geom_point()
```



上图是 200 对观测值的散点图，但是因为图中的点是在水平方向和竖直方向对齐的，很多点重合在一起，使我们不容易发现数据的规律。R 软件的 `jitter(x, amount)` 函数可以对输入的 `x` 加一个 ±`amount` 幅度内的扰动，对扰动后的 `x` 和 `y` 作散点图就可以避免过多的点重合。ggplot2 的 `geom_jitter()` 可以将每个点进行小的扰动，避免重叠。设置一定的透明度也可以表现出局部点的疏密程度。

```
ggplot(data=d5, mapping=aes(x=x, y=y)) +
  geom_jitter(alpha = 0.4)
```



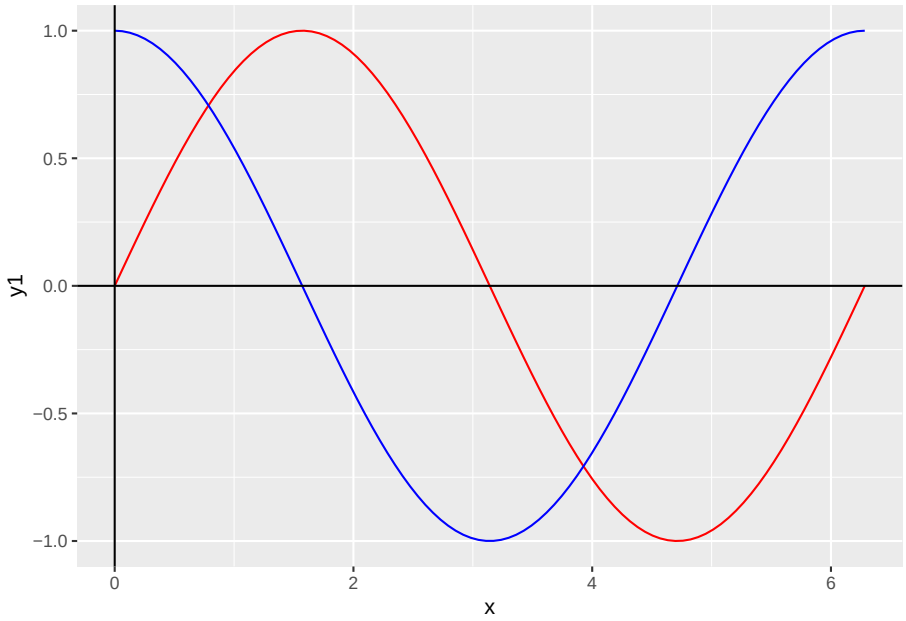
有些数据是随时间变化的，而数据中有时已经有时间变量，也可能没有明显的时间变量。在没有时间变量时，有时输入数据的次序号可以作为时间变量。为了考察随时间变化的量（称为时间序列），可以把以此变量为纵坐标、以时间为横坐标的点用线相连，称这样的图为**曲线图**或时间序列图，实际上，这样的图是折线图，曲线图是一种习惯称呼。

在 R 中，用 `plot(x, y, type=l)` 作曲线图。用 `lines(x, y)` 在已有图形上添加线，用 `points(x, y)` 在已有图形上添加点，用 `abline(h=...)` 在已有图形上添加横线，用 `abline(v=...)` 在已有图形上添加竖线。用 `rug` 函数可以在把点的横坐标值或纵坐标值用短线标在坐标区域边缘。ggplot2 的 `geom_line()` 作折线图（曲线图）。

如下 R 程序作了正弦曲线和余弦曲线图：

```
d6 <- tibble(x = seq(0, 2*pi, length=100)) %>%
  mutate(y1 = sin(x), y2=cos(x))
ggplot(data = d6, mapping=aes(x=x)) +
  geom_line(mapping=aes(y=y1), col="red") +
  geom_line(mapping=aes(y=y2), col="blue") +
  geom_hline(yintercept=0) +
```

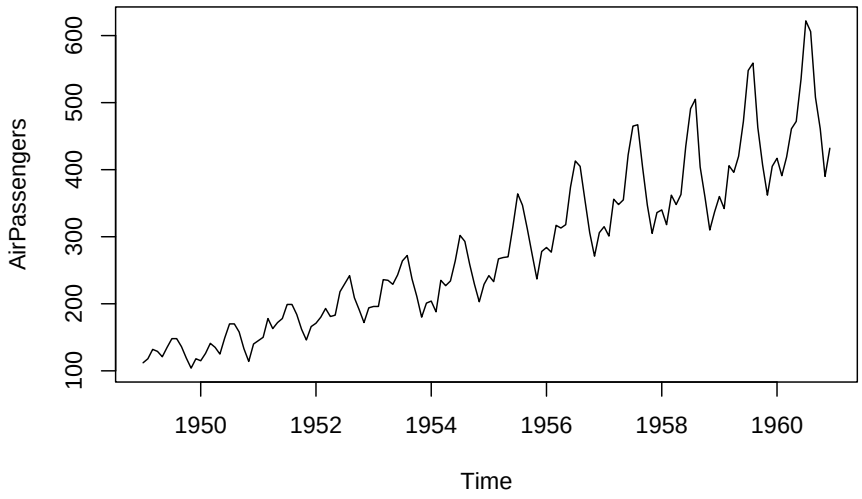
```
  geom_vline(xintercept=0)
```



R 函数 `matplot` 可以在同一坐标系中做多条曲线或多组散点图。

R 中有专门的时间序列数据类型，这种数据类型是向量的变种，定义了序列的开始时间和采样频率（比如每月一次的数据的采样频率为 12），所以序列中每个数值都有对应的时间。用 `ts` 函数定义时间序列。如果 `y` 是时间序列，`plot(y)` 可以直接作时间序列曲线图。下图是美国泛美航空公司 1949—1960 年每月国际航班订票数的时间序列图：

```
plot(AirPassengers)
```

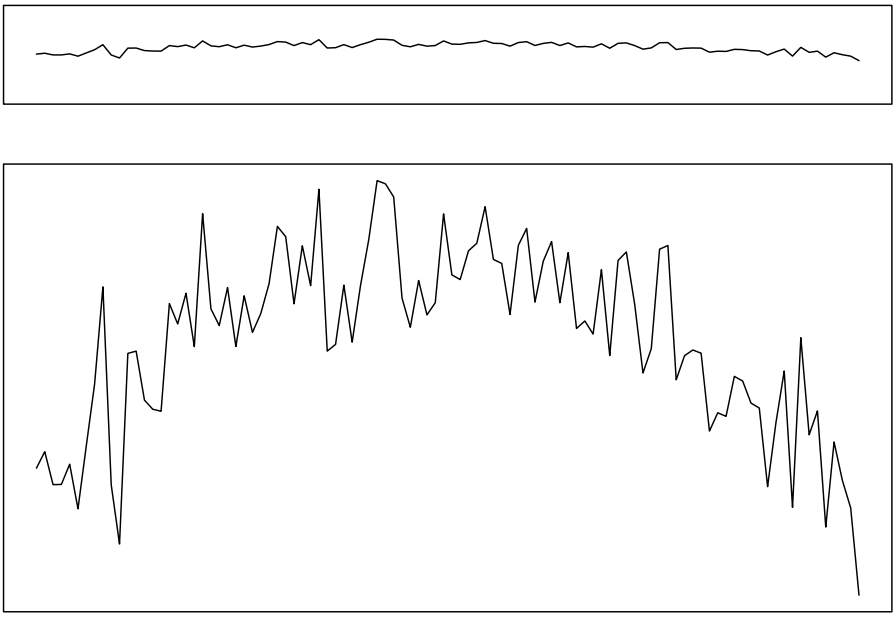


对于时间序列，还可以以 $(y_i, y_{i+1}), i = 1, 2, \ldots, n-1$ 为坐标画散点图以查看自相关情况。时间序列还有一些专门的图形，如自相关函数图、偏相关函数图、谱密度估计图等。

有时随时间变化的是变量间的关系，这就更困难，可以做一系列随时间变化的图形。

散点图、曲线图等图形的坐标轴刻度一般是自动确定的，大致取为该维数据的最小值到最大值的范围。如果需要比较多幅图，要注意坐标范围的问题，必要时可以使多幅图的坐标范围统一化。ggplot2 可以按一个或者几个分类变量将观测分类，每一类作一幅小图，小图的坐标范围自动统一化。

曲线图的宽高比不仅影响视觉效果，不合适的宽高比可能误导我们对变化规律的认识。下面有两幅时间序列图，实际是同一时间序列，但是下面的图清楚地表现了一个有先升后降趋势的序列，而上面的图表面上看是一个在某一水平线上下波动的序列：

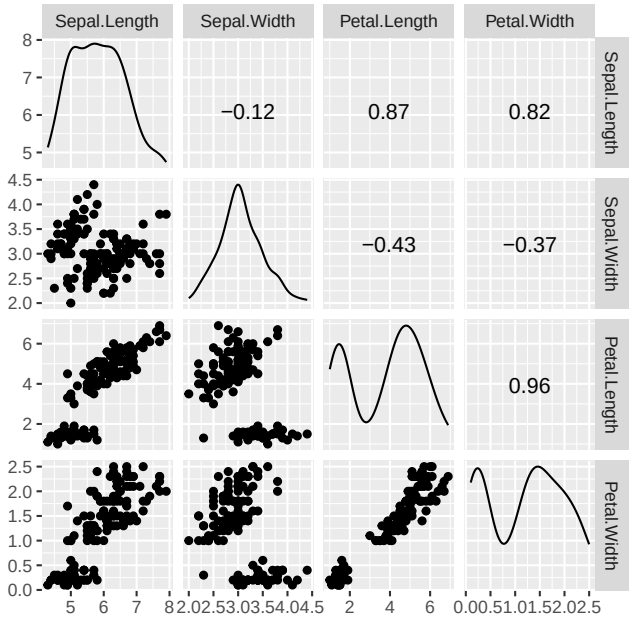


4.7 三维图

如果有多个数值型变量，可以两两作散点图，构成散点图矩阵。在 R 中可以用 `pairs` 函数作散点图矩阵。ggplot2 和 GGally 包提供了散点图矩阵功能。

下图是三种不同的鸢尾花的 150 个样品的花瓣、花萼长、宽的数据的散点图矩阵:

```
library(GGally, quietly=TRUE, warn.conflicts=FALSE)
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
ggscatmat(data=iris, columns=1:4)
```



对于三个区间变量 X, Y, Z 之间的关系，可以用各种三维图形直观地查看。

为了查看二元函数 $z = f(x, y)$ 的图像，可以作曲面图、等值线图和栅格图。R 软件的 `surface(x, y, z)` 作曲面图，其中向量 `x` 和向量 `y` 的值构成 Oxy 平面上的一张网格，`z` 为一个矩阵，第 i 行第 j 列元素为 $f(x_i, y_j)$ 的值。R 函数 `contour(x, y, z)` 作等值线图，这也是反映曲面 $f(x, y)$ 形状的图形，想法来自于地图中的等高线。R 函数 `image(x, y, z)` 作栅格图，每个 (x_i, y_j) 处用不同的颜色或灰度代表不同的 $z = f(x_i, y_j)$ 的值。

ggplot2 的 `geom_contour()` 作等值线图，`geom_raster()` 作栅格图。

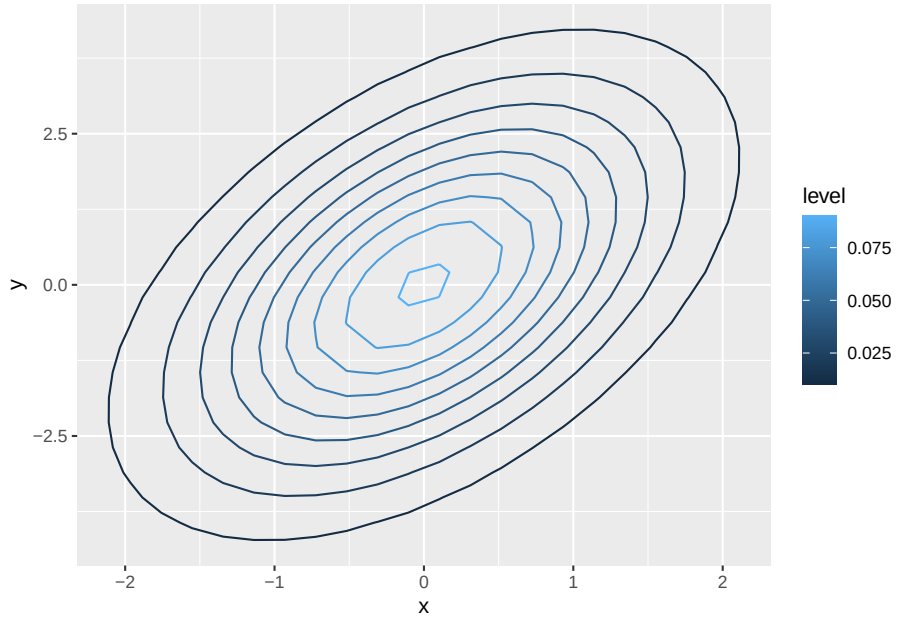
设随机向量 (X, Y) 服从联合正态分布，$EX = EY = 0$, $\mathrm{Var}(X) = \sigma_1^2$, $\mathrm{Var}(Y) = \sigma_2^2$, $\rho(X, Y) = R$。则 (X, Y) 的联合密度为

$$f(x, y) = \frac{1}{2\pi\sqrt{\sigma_1^2\sigma_2^2(1-R^2)}} \exp\left\{-\frac{1}{2}\frac{\sigma_2^2x^2 + \sigma_1^2y^2 - 2R\sigma_1\sigma_2xy}{\sigma_1^2\sigma_2^2(1-R^2)}\right\}$$

如下的 R 程序作 $f(x, y)$ 的等值线图：

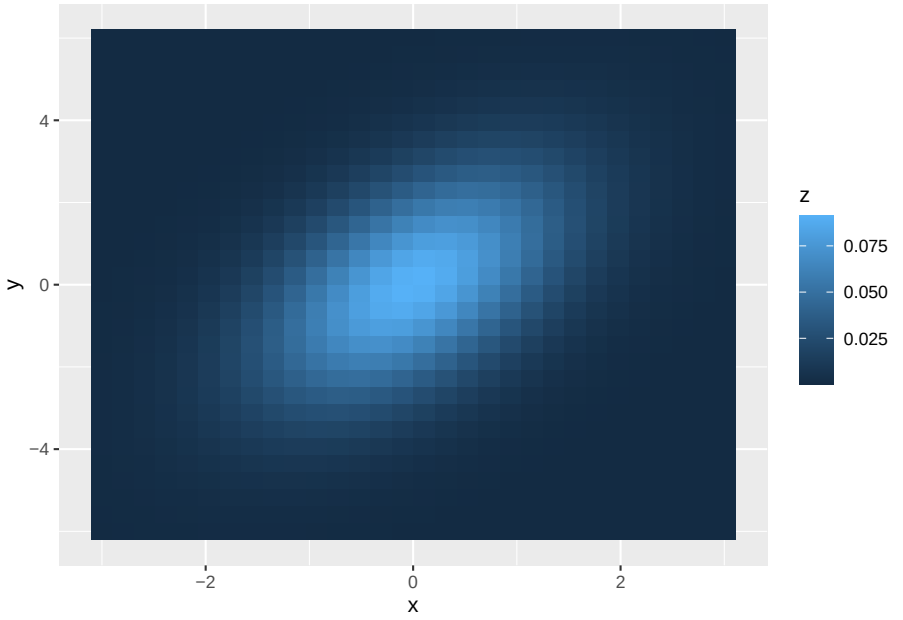
```
ng <- 30
s1 <- 1; s2 <- 2
R <- 0.5
```

```
d <- s1^2 * s2^2 * (1 - R^2)
f <- function(x, y) 1 / (2*pi*sqrt(d)) *
     exp(-0.5 * (s2^2 * x^2 + s1^2 * y^2 - 2*R*s1*s2*x*y)/d)
d7 <- expand_grid(
  x = seq(-3, 3, length=ng)*s1,
  y = seq(-3, 3, length=ng)*s2) %>%
  mutate(z = f(x, y))
ggplot(data = d7, mapping = aes(x=x, y=y, z=z, color=..level..)) +
  geom_contour()
```



栅格图：

```
ggplot(data = d7, mapping = aes(x=x, y=y, fill=z)) +
  geom_raster()
```



类似于散点图，三维散点图可以反映三个变量之间的关系。在 R 中，可以用 `lattice` 包的 `cloud` 函数作三维散点图。

如果变量 X 和 Y 是数值型的，变量 Z 是分类的，则可以画 (X, Y) 散点图，但每个点根据 Z 的不同值使用不同符号（圈、三角、加号、星号、点等）或不同颜色，这样可以用散点图表达出三维信息。如果变量 Z 是数值型的，可以用散点图中绘点符号的大小来表示 Z 的大小。下图是三种鸢尾花各 50 朵的花瓣长、宽数据的散点图，用了不同符号和颜色来代表不同种类。R 函数 `plot`、`lines`、`points` 的 `pch` 参数可以用来规定绘点符号，`col` 参数可以用来规定绘点颜色，`cex` 参数可以用来规定绘点符号的大小。ggplot2 支持 color, fill, shape 等维度。

```
ggplot(data = iris, mapping = aes(
  x = Petal.Length, y = Petal.Width,
  color = Species, shape = Species)) +
  geom_point(size=2, alpha=0.5)
```

rgl 扩展包提供了绘制动态三维散点图的功能。程序如

```
library(rgl)
with(iris, plot3d(
  Sepal.Length, Sepal.Width, Petal.Length,
  type='s', col=as.numeric(factor(Species))))
```

rgl 可以绘制二元正态分布动态曲面图：

```
library(rgl)
s1 <- 1; s2 <- 2
R <- 0.5
xlim <- c(-3, 3)*s1
ylim <- c(-3, 3)*s2
d <- s1^2 * s2^2 * (1 - R^2)
f <- function(x, y) 1 / (2*pi*sqrt(d)) *
     exp(-0.5 * (s2^2 * x^2 + s1^2 * y^2 - 2*R*s1*s2*x*y)/d)
f <- Vectorize(f)
persp3d(f, xlim=xlim, ylim=ylim, n=30)
```

习题

习题 1

分别对样本量 $n = 10, 20, 30, 50, 100$, 模拟生成标准正态分布、对数正态分布、柯西分布样本，并用 `hist()` 函数做直方图。提示：在 R 命令行运行

```
?Distribution
```

可以获得各种与分布有关的函数说明。

习题 2

用 R 程序绘制 `iris` 数据框中三种花的花瓣长、宽和花萼长、宽的散点图矩阵，用不同符号和颜色代表三个不同品种。

Part II

随机数生成

Chapter 5

均匀分布随机数生成

设随机变量 X 有分布函数 $F(x)$，$\{X_i, i = 1, 2, ...\}$ 独立同分布 $F(x)$，则 $\{X_i, i = 1, 2, ...\}$ 的一次观测的数值 $\{x_i, i = 1, 2, ...\}$ 叫做分布 $F(x)$ 的随机数序列，简称**随机数**。随机数是统计中一个重要的计算工具 "随机模拟" 的基本构成元素。

我们可以用物理方法得到一组真实的随机数，比如，反复抛掷硬币、骰子、正二十面体骰子，抽签、摇号，等等。这些方法得到的随机数质量好，但是数量不能满足随机模拟的需要。

另一种办法是预先生成大量的真实随机数存储起来，进行随机模拟时读取存储的随机数。这种方法的速度较低，已经被取代了。

现在进行随机模拟的主流方法是使用计算机实时地产生随机数，严格来说是伪随机数。**伪随机数**是用计算机算法产生的序列 $\{x_i, i = 1, 2, ...\}$, 其表现与真实的 $F(x)$ 的独立同分布序列很难区分开来。因为计算机算法的结果是固定的，所以伪随机数不是真正的随机数；但是，好的伪随机数序列与真实随机数序列表现相同，很难区分，现在的计算机模拟都是使用伪随机数，我们把伪随机数也叫做随机数。

需要某种分布的随机数时，一般先生成均匀分布随机数，然后由均匀分布随机数再转换得到其它分布的随机数。产生均匀分布随机数的算法叫做**均匀分布随机数发生器**。

计算机中伪随机数序列是迭代生成的，即 $x_n = g(x_{n-1}, x_{n-2}, \dots, x_{n-p}), n = 1, 2, \dots, p$ 是正整数，g 是一个非线性函数。如何找到这样的 g，使得产生的序列呈现出随机性？首先要使结果有随机性。比如，把一个大的整数平方后取中间的位数，然后递推进行，叫做平方取中法。这种方法均匀性差而且很快变成零，所以已经不再使用。还可以把序列的前两项相乘然后取中间的数字，这种方法也有类似缺点。

现在常用的均匀分布随机数发生器有线性同余法、反馈位寄存器法以及随机数发生器的组合。均匀分布随机数发生器生成的是 $\{0, 1, \dots, M\}$ 或 $\{1, 2, \dots, M\}$ 上离散取值的离散均匀分布，然后除以 M 或 $M+1$ 变成 $[0, 1]$ 内的值当作连续型均匀分布随机数，实际上是只取了有限个值。因为取值个数有限，根据算法 $x_n = g(x_{n-1}, x_{n-2}, \dots, x_{n-p})$ 可知序列一定在某个时间发生重复，使得序列发生重复的间隔 T 叫做随机数发生器的**周期**。好的随机数发生器可以保证 M 很大而且周期很长。

现代的统计计算编程语言如 R、Python、Julia、MATLAB 等都已经包含了性能很好的随机数发生器，这里只是提供关于随机数发生器的一般知识，一般的读者应该没有必要开发自己的随机数发生器，这一部分用作教材时可以仅作介绍。

5.1 线性同余发生器 (LCG)

5.1.1 同余

定义 5.1 (同余)**.** 设 i, j 为整数，M 为正整数，若 $j - i$ 为 M 的倍数，则称 i 与 j 关于模 M **同余**，记为 $i \equiv j \pmod{M}$。否则称 i 与 j 关于 M 不同余。

例 5.1.

$$\begin{aligned} 11 &\equiv 1 \pmod{10}, \quad 1 \equiv 11 \pmod{10}, \\ -9 &\equiv 1 \pmod{10}. \end{aligned}$$

同余有如下性质：

(1) 对称性：$i \equiv j \pmod{M} \iff j \equiv i \pmod{M}$。
(2) 传递性：若 $i \equiv j \pmod{M}$, $j \equiv k \pmod{M}$, 则 $i \equiv k \pmod{M}$。

(3) 若 $i_1 \equiv j_1 \pmod M$, $i_2 \equiv j_2 \pmod M$, 则

$$i_1 \pm i_2 \equiv j_1 \pm j_2 \pmod M, \quad i_1 i_2 \equiv j_1 j_2 \pmod M.$$

(4) 若 $ik \equiv jk \pmod M$(k 为正整数), 则 $i \equiv j \pmod{\frac{M}{\gcd(M,k)}}$, 其中 $\gcd(M,k)$ 表示 M 和 k 的最大公约数。

证明: (1)、(2) 和 (3) 的加减运算用同余定义很容易证明。来证明 (3) 中乘法的结果。由 $i_1 \equiv j_1 \pmod M$ 和 $i_2 \equiv j_2 \pmod M$ 的定义可知存在整数 k_1 和 k_2 使得 $j_1 - i_1 = k_1 M$, $j_2 - i_2 = k_2 M$, 于是

$$\begin{aligned} j_1 j_2 - i_1 i_2 =& j_1 j_2 - j_1 i_2 + j_1 i_2 - i_1 i_2 \\ =& j_1(j_2 - i_2) + i_2(j_1 - i_1) = j_1 \cdot k_2 M + i_2 \cdot k_1 M \\ =& (j_1 k_2 + i_2 k_1) M \end{aligned}$$

即有 $i_1 i_2 \equiv j_1 j_2 \pmod M$。

再来证明 (4)。设 $ik - jk = nM$, n 为整数，则 $i - j = \frac{nM}{k}$ 为整数。设 $M = M_1 \cdot \gcd(M,k)$, $k = k_1 \cdot \gcd(M,k)$, 则 $i - j = \frac{nM_1}{k_1}$ 且 M_1 和 k_1 互素，于是 n 被 k_1 整除，所以 $i \equiv j \pmod{M_1}$，即 $i \equiv j \pmod{\frac{M}{\gcd(M,k)}}$。

设 M 是正整数，A 为非负整数，A 除以 M 的余数为 $A \pmod M = A - [A/M] \times M$, 其中 $[\cdot]$ 表示取整。显然 $0 \le (A \pmod M) \le M - 1$，$A$ 和 $A \pmod M$ 关于模 M 同余。在 R 中用 `x %% y` 表示 x 除以 y 的余数。

5.1.2 线性同余发生器

线性同余随机数发生器是利用求余运算的随机数发生器。其递推公式为

$$x_n = (a x_{n-1} + c) \pmod M, \ n = 1, 2, ...$$

这里等式右边的 $(a x_{n-1} + c) \pmod M$ 表示 $a x_{n-1} + c$ 除以 M 的余数，正整数 M 为除数，正整数 a 为乘数，非负整数 c 为增量，取某个非负整数 x_0 为初值可以向前递推，递推只需要序列中前一项，得到的序列 $\{x_n\}$ 为非负整数，$0 \le x_n < M$。最后，令 $R_n = x_n / M$，则 $R_n \in [0,1)$，把 $\{R_n\}$ 作为均匀分布随机数序列。这样的算法的基本思想是因为很大的整数前面的位数是重要的有效位数而后面若干位有一定随机性。如果取 $c = 0$，线性同余发生器称为**乘同余发生器**, 如果取 $c > 0$，线性同余发生器称为**混合同余发生器**。

因为线性同余法的递推算法仅依赖于前一项，序列元素取值只有 M 个可能取值，所以产生的序列 $x_0, x_1, x_2, ...$ 一定会重复。若存在正整数 n 和 m 使得 $x_n = x_m (m < n)$，则必有 $x_{n+k} = x_{m+k}, k = 0, 1, 2, ...$, 即 $x_n, x_{n+1}, x_{n+2}, ...$ 重复了 $x_m, x_{m+1}, x_{m+2}, ...$，称这样的 $n-m$ 的最小值 T 为此随机数发生器在初值 x_0 下的周期。由序列取值的有限性可见 $T \leq M$。

例 5.2. 考虑线性同余发生器

$$x_n = 7x_{n-1} + 7 \pmod{10}.$$

取初值 $x_0 = 7$, 数列为: $(7, 6, 9, 0, 7, 6, 9, 0, 7, ...)$, 周期为 $T = 4 < M = 10$。

例 5.3. 考虑线性同余发生器

$$x_n = 5x_{n-1} + 1 \pmod{10}$$

取初值 $x_0 = 1$。数列为: $(1, 6, 1, 6, 1, ...)$, 显然周期 $T = 2$。

例 5.4. 考虑线性同余发生器

$$x_n = 5x_{n-1} + 1 \pmod{8}$$

取初值 $x_0 = 1$。数列为 $(1, 6, 7, 4, 5, 2, 3, 0, 1, 6, 7, ...)$, $T = 8 = M$ 达最大周期。

从例子发现 $T \leq M$ 且有的线性同余发生器可以达到最大周期 M，称为满周期。满周期时，初值 x_0 取为 $0 \sim M-1$ 间的任意数都是一样的，$X_M = x_0$，序列从 X_M 开始重复。如果发生器从某个初值不是满周期的，那么它从任何初值出发都不是满周期的，不同初值有可能得到不同序列。比如随机数 $x_n = 7x_{n-1} + 7 \pmod{10}$, 从不同初值出发的序列可能为如 $7, 6, 9, 0, 7, \cdots$; $1, 4, 5, 2, 1, \cdots$; $3, 8, 3$。

不同的 M, a, c 选取方法得到不同的周期，适当选取 M, a, c 才能使得产生的随机数序列和真正的 U[0, 1] 随机数表现接近。

5.1.3 混合同余发生器 (*)

线性同余发生器中 $c > 0$ 时称为混合同余发生器。下列定理给出了混合同余发生器达到满周期的一个充分条件。

定理 5.1. *当下列三个条件都满足时，混合同余发生器可以达到满周期:*

(1) c 与 M 互素;
(2) 对 M 的任一个素因子 P，$a-1$ 被 P 整除;
(3) 如果 4 是 M 的因子，则 $a-1$ 被 4 整除。

常取 $M=2^L$，L 为计算机中整数的尾数字长。这时根据定理5.1的建议可取 $a=4\alpha+1$，$c=2\beta+1$，α 和 β 为任意正整数，x_0 为任意非负整数，这样的混合同余发生器是满周期的，周期为 2^L。

好的均匀分布随机数发生器应该周期足够长，统计性质符合均匀分布，序列之间独立性好。把同余法生成的数列看成随机变量序列 $\{X_n\}$，在满周期时，可认为 X_n 是从 $0\sim M-1$ 中随机等可能选取的，即

$$P\{X_n=i\}=1/M,\ i=0,1,\ldots,M-1$$

这时

$$EX_n=\sum_{i=0}^{M-1} i\cdot\frac{1}{M}=(M-1)/2,$$

$$\mathrm{Var}(X_n)=EX_n^2-(EX_n)^2=\sum_{i=0}^{M-1} i^2\frac{1}{M}-\frac{(M-1)^2}{4}=\frac{1}{12}(M^2-1)$$

于是当 M 很大时

$$ER_n=\frac{1}{2}-\frac{1}{2M}\approx\frac{1}{2};$$

$$\mathrm{Var}(R_n)=\frac{1}{12}-\frac{1}{12M^2}\approx\frac{1}{12}$$

可见 M 充分大时从一、二阶矩看生成的数列很接近于均匀分布。

随机数序列还需要有很好的随机性。数列的排列不应该有规律，序列中的两项不应该有相关性。

因为序列由确定性公式生成，所以不可能真正独立。至少我们要求是序列自相关性弱。对于满周期的混合同余发生器可以得序列中前后两项自相关系数的近似公式

$$\rho(1)\approx\frac{1}{a}-\frac{6c}{aM}(1-\frac{c}{M})$$

所以应该选 a 值大 (但 $a<M$)。

例 5.5. Kobayashi 提出了如下的满周期 2^{31} 的混合同余发生器

$$x_n = (314159269x_{n-1} + 453806245) \pmod{2^{31}}$$

其周期较长，统计性质比较好。

Julia 实现：

```
function rng_KS_fixed_seed(n=1, seed=0)
  y = Vector{Float64}(n)
  x::Int64 = seed
  for i in 1:n
    x = (314159269*x + 453806245) % 2147483648
    y[i] = x / 2147483648
  end
  y
end
## 测试:
y = rng_KS_fixed_seed(100)
```

上一版本需要用户输入种子，如果需要多次调用，就需要同时输出当前种子值。利用“闭包”（closure）可以记住函数的当前状态，下一版本的实现可以自动记住当前种子：

```
function rng_KS(seed=0)
  local the_seed::Int64 = seed

  function rng(n=1)
    y = Vector{Float64}(n)
    x::Int64 = the_seed
    for i in 1:n
      x = (314159269*x + 453806245) % 2147483648
      y[i] = x / 2147483648
    end
```

```
    the_seed = x
    y
  end

  rng
end
```

测试：

```
myrng = rng_KS()
myrng(2)
## 2-element Array{Float64,1}:
##  0.21132
##  0.0102247
```

```
myrng(2)
## 2-element Array{Float64,1}:
##  0.188331
##  0.665933
```

可以看出每次调用产生不同的序列。

5.1.4 乘同余法 (*)

线性同余发生器中 $c = 0$ 时的生成方法称为乘同余法，或称积式发生器。这时的递推公式为

$$x_n = ax_{n-1} \pmod{M},\ n = 1, 2, ...$$

问题是如何选 M, a 达到大周期且统计性质好。显然各 x_n 都不能等于 0；如果某个 $x_k = 0$, 则 $x_{k+1} = x_{k+2} = \cdots = 0$。乘同余法有可能进入 0 就不再周期变化，称为退化情况，这时周期为 1。所以乘同余法能够达到的最大周期是 $M - 1$，每个 x_n 都只在 $\{1, 2, ..., M - 1\}$ 中取值。

定义 5.2 (阶数)**.** 设正整数 a 与正整数 M 互素，称满足 $a^V \equiv 1 \pmod{M}$ 的最小正整数 V 为 a 对模 M 的阶数 (或次数)，简称为 a 的阶数。

我们来证明阶数存在。考虑乘同余发生器 $x_n = ax_{n-1}$, $n = 1, 2, ...$, 取 $x_0 = 1$。由同余的传递性可知 $x_n \equiv a^n x_0 \pmod{M}$ 即 $x_n \equiv a^n \pmod{M}$，而 $0 \leq x_n \leq M-1$ 所以 $x_n = a^n \pmod{M}$。因为 a 与 M 互素所以 $x_n \neq 0$，序列 $\{x_n\}$ 只在 $1, \ldots, M-1$ 中取值，必存在 $0 \leq i < j < i+M$ 使得 $x_j = x_i$，其中 $x_i \equiv a^i \pmod{M}$，$x_j \equiv a^j \pmod{M}$，于是 $a^i \equiv a^j \pmod{M}$, 由同余的性质 (4) 有 $1 \equiv a^{j-i} \pmod{\frac{M}{\gcd(a^i, M)}}$, 其中 $\gcd(a^i, M) = 1$ 所以 $a^{j-i} \equiv 1 \pmod{M}$。取 $V = j - i$ 则 V 为正整数且 $a^V \equiv 1 \pmod{M}$。

Lemma 5.1. *设 a 与 M 互素，初值 x_0 与 M 互素，则乘同余法的周期为 a 对模 M 的阶数 V。*

证明：由同余的传递性可知 $x_V \equiv a^V x_0 \pmod{M}$, 而 $a^V \equiv 1 \pmod{M}$ 所以

$$a^V x_0 \equiv x_0 \pmod{M}$$

于是

$$x_V \equiv x_0 \pmod{M}.$$

这时必有 $x_V = x_0$，所以乘同余法的周期 $T \leq V$。

如果 $T < V$，则存在 $0 \leq i < j < i+V$ 使得 $x_j = x_i$，因 $x_j \equiv a^j x_0 \pmod{M}$, $x_i \equiv a^i x_0 \pmod{M}$, 由同余的传递性可知 $a^j x_0 \equiv a^i x_0 \pmod{M}$，由同余的性质 (4) 有

$$a^j \equiv a^i \pmod{\frac{M}{\gcd(M, x_0)}}$$

而 $\gcd(M, x_0) = 1$ 所以 $a^j \equiv a^i \pmod{M}$, 由同余的性质 (4) 可知

$$a^{j-i} \equiv 1 \pmod{\frac{M}{\gcd(a^i, M)}}$$

其中 $\gcd(a^i, M) = 1$ 所以 $a^{j-i} \equiv 1 \pmod{M}$，$0 < j - i < V$，与 V 是满足 $a^V \equiv 1 \pmod{M}$ 的最小正整数矛盾。证毕。

※※※※※

对乘同余法，当 M 与 a 互素且 M 与 x_0 互素时，$x_n = a^n x_0 \pmod{M}$，易见 x_n 与 M 互素，序列不会退化为 0.

例 5.6. 考虑如下乘同余发生器

$$\begin{aligned} x_n &= (8\alpha + 5)x_{n-1} \pmod{2^L} \\ x_0 &= 4b + 1 \end{aligned}$$

(其中 α, b 为非负整数)。再考虑如下的混合同余发生器

$$\begin{aligned} x_n^* =&(8\alpha+5)x_{n-1}^* + (2\alpha+1) \pmod{2^{L-2}} \\ x_0^* =&b \end{aligned}$$

则 $x_n = 4x_n^* + 1, n = 0, 1, 2, ...$。

解:

用归纳法。当 $n = 0$ 时

$$x_0^* = b,\ x_0 = 4b + 1$$

所以 $x_0 = 4x_0^* + 1$ 成立。设 $x_n = 4x_n^* + 1$ 成立，由

$$x_n = (8\alpha+5)x_n \pmod{2^L},$$

其中

$$\begin{aligned} (8\alpha+5)x_n =&(8\alpha+5)(4x_n^*+1) \\ =&4[(8\alpha+5)x_n^* + (2\alpha+1)] + 1 \end{aligned}$$

于是

$$\begin{aligned} (8\alpha+5)x_n \pmod{M} =&4\{[(8\alpha+5)x_n^* + (2\alpha+1)] \pmod{2^{L-2}}\} + 1 \\ =&4x_{n+1}^* + 1 \end{aligned}$$

得证。

这个例子中，由定理5.1可知 $\{x_n^*\}$ 是满周期的, $\{x_n\}$ 与 $\{x_n^*\}$ 一一对应，所以 $\{x_n\}$ 周期为 2^{L-2}。这里的乘同余发生器与一个混合同余发生器等价。

※※※※※

对乘同余法得到的随机序列 $\{X_n\}$，当 M 充分大时可以类似得到

$$ER_n \approx \frac{1}{2}, \quad \mathrm{Var}(R_n) \approx \frac{1}{12}.$$

为了使得前后两项的相关系数较小，应取 a 较大且使得 a 的二进制表示排列无规律。

5.1.5 素数模乘同余法 (*)

若取 M 为小于 2^L 的最大素数，选取适当的 a 可以达到 $T = M - 1$ 周期，这样的发生器叫**素数模乘同余发生器**。

定义 5.3 (素元)**.** 设正整数 a 和素数 M 互素，若 a 对模 M 的阶数 V 满足 $V = M - 1$，则称 a 为 M 的素元 (或原根)。

乘同余发生器中取 a 为 M 的素元可以达到最大周期 $M - 1$。

例 5.7. $M = 7$ 是素数，取 $a = 3$ 与 M 互素。考虑素数模乘同余发生器

$$x_n = a^n \pmod{M},\ n = 1, 2, \ldots,\ x_0 = 1,$$

序列为 $1, 3, 2, 6, 4, 5, 1, 3, \ldots$。周期达到 $M - 1 = 6$，所以 $a = 3$ 是 $M = 7$ 的素元。

可以验证 $a = 5$ 也是 $M = 7$ 的素元，序列为 $1, 5, 4, 6, 2, 3, 1, 5, \ldots$。$1, 2, 4, 6$ 不是 $M = 7$ 的素元。此例说明素元可以不唯一。

在选取素数模乘同余发生器的参数 M 和 a 的时候，可以取 M 为小于 2^L 的最大素数 (L 为计算机整数表示的尾数位数)，取 a 为 M 的素元，这样保证周期 $T = M - 1$。a 应尽可能大，a 的二进制表示尽可能无规律。这样在一个周期内可以得到 $1, 2, \ldots, M - 1$ 的一个排列。初值 x_0 可以取 $1, 2, \ldots, M - 1$ 中任一个。

例 5.8. Lewis-Goodman-Miller(1969) 的素数模乘同余发生器为

$$x_n = a x_{n-1} \pmod{2^{31} - 1},\ n = 1, 2, \ldots,\ x_0 \text{ is arbitrary positive integer}$$

a 取如下四个值之一: $(7^5 = 16807, 397204094, 764261123, 630360016)$。

其中 $a = 16807$ 的版本有许多应用，但是其高维表现很差，现在应该避免使用。

Julia 实现，实现一个用户指定种子的版本：

```
function rng_LGM_fixed_seed(n=1, seed=11111, form=4)
  mults = [16807, 397204094, 764261123, 630360016]
  a = mults[form]
```

```
  y = Vector{Float64}(n)
  x::Int64 = seed
  for i in 1:n
    x = (a*x) % 2147483647
    y[i] = x / 2147483647
  end
  y
end
## 测试:
rng_LGM_fixed_seed(5, 112233)
## 5-element Array{Float64,1}:
##  0.230227
##  0.556961
##  0.770818
##  0.692613
## 0.289898
```

注意，在设计线性同余法程序时，如果使用程序语言中的整数型或无符号整数型来存储 x_n，则在计算 ax_{n-1} 的乘法时可能发生溢出。对 $M = 2^L$ 情形，如果程序语言支持溢出而不作为错误，则溢出恰好完成了求除以 2^L 的余数的运算。对于 $M < 2^L$ 的情形也可以巧妙设计以利用溢出。对于较小的 a 和 M(如 a 和 M 都小于 $2^{L/2}$ 时) 不会溢出。如果使用双精度实数型来保存 x_n 则不需要考虑溢出问题。

例 5.9. 设计素数模乘同余发生器算法, 要求不产生溢出。

设 a 为正整数，M 为素数，令 $b = [M/a]$, $c = M \pmod a$，则 $M = ab + c$, 设 $a < b$。乘同余递推中的除法满足

$$\frac{ax_{n-1}}{M} = \frac{ax_{n-1}}{ab+c} \le \frac{x_{n-1}}{b}$$

且

$$\begin{aligned}\frac{x_{n-1}}{b} - \frac{ax_{n-1}}{ab+c} &< \frac{ax_{n-1}}{ab} - \frac{ax_{n-1}}{ab+a} = \frac{x_{n-1}}{b(b+1)} \\ < \frac{M}{b(b+1)} &= \frac{ab+c}{b(b+1)} < \frac{ab+a}{b(b+1)} = \frac{a}{b} < 1\end{aligned}$$

即

$$\frac{ax_{n-1}}{M} \leq \frac{x_{n-1}}{b} < \frac{ax_{n-1}}{M} + 1$$

记 $k_0 = [\frac{ax_{n-1}}{M}]$, $k_1 = [\frac{x_{n-1}}{b}]$, 则 $k_0 \leq k_1 \leq k_0 + 1$, 计算 k_0 不用计算乘法所以不会溢出，$k_1 = k_0$ 或 $k_0 + 1$。于是

$$\begin{aligned} x_n =& ax_{n-1} - k_0 M = ax_{n-1} - k_1 M + (k_1 - k_0)M \\ =& ax_{n-1} - k_1(ab + c) + (k_1 - k_0)M \\ =& a(x_{n-1} - k_1 b) - k_1 c + (k_1 - k_0)M \\ =& a(x_{n-1} \pmod b) - k_1 c + (k_1 - k_0)M \end{aligned}$$

其中 $k_1 - k_0$ 等于 0 或 1，因为 $x_n > 0$，所以如果 $a(x_{n-1} \pmod b) - k_1 c < 0$ 则 $k_1 - k_0 = 1$。

避免溢出的素数模发生器算法

设置 M, a, b, c, x_0 的值
for(n **in** $1 : N$)
    $k_1 \leftarrow \text{floor}(x_{n-1}/b)$
    $x_n \leftarrow a(x_{n-1} - k_1 b) - k_1 c$
    if $(x_n < 0)$ $x_n \leftarrow x_n + M$
}

对例5.8的素数模发生器，取 $a = 16807$ 可以按上述算法计算。

上述的算法使用了伪代码来描述，伪代码的控制结构仿照 R 语言的控制结构。

一些常见的素数模乘同余发生器参数：

- $m = 2^{31} - 1$，$a = 16807$(Lewis, Goodman, Miller), 39373(L'Ecuyer), 742938285, 950706376, 1226874159(Fishman and Moore);
- $m = 2147483399$, $a = 40692$(L'Ecuyer);
- $m = 2147483563$, $a = 40014$(L'Ecuyer)。

5.2 FSR 发生器 (*)

线性同余法的周期不可能超过 2^L(L 为整数型尾数的位数)，
而且作为多维随机数相关性大，分布不均匀。基于 Tausworthe(1965) 文章的 FSR 方法是一种全新的做法，对这些方面有改善。

FSR(反馈位移寄存器法) 按照某种递推法则生成一系列二进制数 $\alpha_1, \alpha_2, \dots, \alpha_k, \dots$，其中 α_k 由前面的若干个 $\{\alpha_i\}$ 线性组合并求除以 2 的余数产生:

$$\alpha_k = (c_p\alpha_{k-p} + c_{p-1}\alpha_{k-p+1} + \cdots + c_1\alpha_{k-1}) \pmod 2,\ k = 1, 2, \dots \quad (5.1)$$

线性组合系数 $\{c_i\}$ 只取 0, 1，这样的递推可以利用程序语言中的整数二进制运算快速实现。给定初值 $(\alpha_{-p+1}, \alpha_{-p+2}, \dots, \alpha_0)$ 向前递推，得到 $\{\alpha_k, k = 1, 2, \dots\}$ 序列后依次截取长度为 L 的二进制位组合成整数 x_n，$R_n = x_n/2^L$。不同的组合系数和初值选择可以得到不同的随机数发生器, 巧妙设计可以得到很长的周期，作为多维均匀随机数性质较好。

FSR 算法中系数 $(c_1, c_2, \dots, c_p)$ 如果仅有两个为 1，比如 $c_p = c_{p-q} = 1 (1 < q < p)$, 则算法变成

$$\begin{aligned}\alpha_k =& (\alpha_{k-p} + \alpha_{k-p+q}) \pmod 2\\ =& \begin{cases} 0 & 若 \alpha_{k-p} = \alpha_{k-p+q} \\ 1 & 若 \alpha_{k-p} \neq \alpha_{k-p+q} \end{cases}\end{aligned}$$

用 $\oplus$ 表示二进制异或运算，则

$$\alpha_k = \alpha_{k-p} \oplus \alpha_{k-p+q},\ k = 1, 2, \dots .$$

比如，取 $p = 98, q = 27$。

设计 FSR 计算机程序时，直接对包含 M 个二进制位的非负整数 $\{x_n\}$ 的数列用异或递推更方便。递推公式为

$$\begin{aligned}x_n =& x_{n-p} \oplus x_{n-p+q}, \quad (1 < q < p),\ n = 1, 2, \dots\\ R_n =& x_n/2^M\end{aligned}$$

这需要由 p 个 M 位二进制非负整数作为种子 (初值)。这种算法只需要异或运算，不受计算机字长限制，适当选取 p, q 后周期可以达到 $2^p - 1$，作为多维随机数的性质可以很好，需要预先研究得到种子表而不能随便取初值。

5.3 组合发生器法 (*)

随机数设计中比较困难的是独立性和多维的分布。可以考虑把两个或若干个发生器组合利用, 可以比单个发生器由更长的周期和更好的随机性。

例 5.10. Wichmann 和 Hill(1982)(见 Wichmann and Hill [1982]) 设计了如下的线性组合发生器。

利用三个 16 位运算的素数模乘同余发生器:

$$\begin{aligned} U_n &= 171U_{n-1} \pmod{30269} \\ V_n &= 172V_{n-1} \pmod{30307} \\ W_n &= 170W_{n-1} \pmod{30323} \end{aligned}$$

作线性组合并求余:

$$R_n = (U_n/30269 + V_n/30307 + W_n/30323) \pmod 1$$

这个组合发生器的周期约有 7×10^{12} 长, 超过 $2^{32} \approx 4 \times 10^9$。

Julia 实现 (需要用户输入种子):

```
function rng_WH_fixed_seed(n, seed1=11, seed2=13, seed3=17)
  y = Vector{Float64}(n)
  u::Int64 = seed1
  v::Int64 = seed2
  w::Int64 = seed3
  for i in 1:n
    u = (171*u) % 30269
    v = (172*v) % 30307
    w = (170*w) % 30323
    y[i] = (u/30269 + v/30307 + w/30323) % 1.0
  end
  y
end
## 测试:
```

```
rng_WH_fixed_seed(5)
## 5-element Array{Float64,1}:
##  0.231228
##  0.518513
##  0.153344
##  0.502289
##  0.875749
```

下面是重复调用时自动记忆上一次结束时种子的随机数发生器制造机：

```
function make_rng_WH(seed=[1011,2013,3017])
  local the_seed = seed

  function rng(n)
    y = Vector{Float64}(n)
    u::Int64 = the_seed[1]
    v::Int64 = the_seed[2]
    w::Int64 = the_seed[3]
    for i in 1:n
      u = (171*u) % 30269
      v = (172*v) % 30307
      w = (170*w) % 30323
      y[i] = (u/30269 + v/30307 + w/30323) % 1.0
    end

    the_seed = [u, v, w]
    y
  end

  rng
end
```

例 5.11. MacLaren 和 Marsaglia(1965) 设计了组合同余法，组合两个同余发生器，一个用来“搅乱”次序。

设有两个同余发生器 A 和 B。用 A 产生 m 个随机数 (如 $m = 128$)，存放在数组 $T = (t_1, t_2, \dots, t_m)$. 需要产生 x_n 时，从 B 中生成一个随机下标 $j \in \{1, 2, \dots, m\}$，取 $x_n = t_j$，但从 A 再生成一个新随机数 y 代替 T 中的 t_j，如此重复。

这样组合可以增强随机性，加大周期 (可超过 2^L)。也可以只使用一个发生器，用 x_{n-1} 来选择下标。

在 R 软件中，用 `runif(n)` 产生 n 个 U(0,1) 均匀分布的随机数。R 软件提供了若干种随机数发生器，可以用 `RNGkind` 函数切换。在使用随机数进行随机模拟研究时，如果希望模拟研究的结果可重复，就需要在模拟开始时设置固定的随机数种子。虽然不同的随机数发生器种子的形式有所不同，在 R 中，总是可以用函数 `set.seed(seed)` 来设置种子，其中 `seed` 是一个序号，实际的种子由这个序号决定。

5.4 随机数的检验 (*)

文献中已经有许多随机数生成算法，统计软件中也已经包含了许多公认的可靠的随机数发生器。但是，我们要解决一个自己的模拟问题时，还是要反复确认所用的随机数在自己的问题中是好的，没有明显缺陷的。比如，一个经典的称为 RANDU 的随机数发生器用在一维积分时效果很好，但连续三个一组作为三维均匀分布则很不均匀。

为了验证随机数的效果，最好是找一个和自己的问题很接近但是有已知答案的问题，用随机模拟计算并考察其精度。

对均匀分布随机数发生器产生的序列 $\{R_n, n = 1, 2, \dots, N\}$ 可以进行各种各样的检验以确认其均匀性与独立性。下面列举一些检验的想法：

- 对随机数列 $\{R_n, n = 1, 2, \dots, N\}$ 计算

$$\bar{R} = \frac{1}{N}\sum_{n=1}^{N} R_n,\ \overline{R^2} = \frac{1}{N}\sum_{n=1}^{N} R_n^2,\ S^2 = \frac{1}{N}\sum_{n=1}^{N}(R_n - \frac{1}{2})^2$$

  则 $N \to \infty$ 时 $\bar{R}$、$\overline{R^2}$ 和 S^2 均渐近服从正态分布，可以用 Z 检验法检验这三个统计量与理论期望值的偏离程度。

- 把 $[0,1]$ 等分成 k 段，用拟合优度卡方检验法检验 $\{R_n, n=1,2,\dots,N\}$ 落在每一段的取值概率是否近似为 $1/k$。
- 用 Kolmogorov-Smirnov 检验法进行拟合优度检验，看 $\{R_n, n=1,2,\dots,N\}$ 是否与 U[0,1] 分布相符。
- 把 $\{R_n, n=1,2,\dots,N\}$ 每 d 个组合在一起成为 $\mathbb{R}^d$ 向量，把超立方体 $[0,1]^d$ 每一维均分为 k 份，得到 k^d 个子集，用卡方检验法检验组合得到的 $\mathbb{R}^d$ 向量落在每个子集的概率是否近似为 k^{-d}。
- 把 $\{R_n, n=1,2,\dots,N\}$ 看作时间序列样本，计算其样本自相关函数列 $\{\hat{\rho}_j, j=1,2,\dots\}$，在 $\{R_n, n=1,2,\dots\}$ 独立同分布情况下 $\{\sqrt{N}\hat{\rho}_j, j=1,2,\dots,N\}$ 应该渐近服从独立的标准正态分布，可以据此进行白噪声检验。
- 把 $\{R_n\}$ 离散化为 $y_n=\text{floor}(kR_n), n=1,2,\dots,N$, 令 $\xi_n=y_n$, $\eta_n=y_{n+b}$(b 为正整数), $n=1,2,\dots,N-b$, 用列联表检验法检验 ξ_n 和 η_n 的独立性。
- 游程检验。把序列 $\{R_n, n=1,2,\dots,N\}$ 按顺序切分为不同段，每一段中的数值都是递增的，这样的一段叫做一个**上升游程**，如 (0.855), $(0.108, 0.226)$, $(0.032, 0.123)$, $(0.055, 0.545, 0.642, 0.870)$, ...。在相邻的两个上升游程中前一个游程的最后一个值大于后一个游程的第一个值。在 $\{R_n, n=1,2,\dots,\}$ 独立同分布条件下游程长度的理论分布可以得知，然后可以比较实际游程长度与独立同分布条件下期望长度。
- 扑克检验。把 $\{R_n\}$ 离散化为 $y_n=\text{floor}(8R_n), n=1,2,\dots,N$, 然后每连续的 8 个作为一组，计数每组内不同数字的个数（$1\sim 8$ 个）。在 $\{R_n\}$ 独立同均匀分布条件下每组内不同数字个数的理论分布概率可以计算出来，然后用卡方检验法检验实际观测与理论概率是否相符。
- 配套检验。$\{R_n\}$ 离散化为 $y_n=\text{floor}(kR_n)$(k 为正整数), $n=1,2,\dots,N$, 然后顺序抽取 $\{y_n, n=1,2,\dots,N\}$ 直到 $\{y_n\}$ 的可能取值 $\{0,1,\dots,k-1\}$ 都出现过为止，记录需要抽取的 $\{y_n\}$ 的个数 L，反复抽取并记录配齐数字需要抽取的值的个数 $l_j, j=1,2,\dots$。在 $\{R_n\}$ 独立同 U(0,1) 分布条件下这样的 L 分布可以得到，可以计算 $\{l_j\}$ 的平均值并用渐近正态分布检验观测均值与理论均值的差异大小，或直接用卡方检验法比较 $\{l_j\}$ 的样本频数与理论期望值。

- 正负连检验。令 $y_n = R_n - \frac{1}{2}$，把连续的 $\{y_n\}$ 的正值分为一段，把连续的 $\{y_n\}$ 的负值分为一段，每段叫做一个 “连”，连长 L 的分布概率为 $P(L = k) = 2^{-k}, k = 1, 2, ...$，可以用卡方检验法检验 L 的分布；总连数 T 满足 $ET = \frac{n+1}{2}$, $\mathrm{Var}(T) = \frac{n-1}{4}$, 可以用 Z 检验法检验 T 的值与理论期望的差距。

- 升降连检验。计算 $y_n = R_n - R_{n-1}$, $n = 2, 3, ..., N$，把连续的正值的 $\{y_n\}$ 分为一段叫做一个上升连，把连续的负值的 $\{y_n\}$ 分为一段叫做一个下降连，可以用卡方检验法比较连的长度与 $\{R_n\}$ 独立同分布假设下的理论分布，或用 Z 检验法比较总连数与理论期望值的差距。

习题

习题 1

对线性同余发生器

$$x_n = (ax_{n-1} + c) \pmod{M},\ n = 1, 2, ...$$

证明若从初值 x_0 出发周期等于 M，则以 $0 \sim M - 1$ 中任何一个整数作为初值时周期都等于 M；若从某初值 x_0 出发周期小于 M，则以 $0 \sim M - 1$ 中任何一个整数作为初值时周期都小于 M。

习题 2

编写 R 程序，对某个均匀分布随机数发生器产生的序列做均匀性的卡方检验。

习题 3

设随机变量 X 密度函数 $p(x)$ 和分布函数 $F(x)$ 已知，编写 R 程序，对 X 的某个随机数发生器产生的序列做拟合优度卡方检验。

习题 4

设随机变量 X 的分布密度为

$$p(x) = \frac{1}{2} + x,\ 0 \le x \le 1,$$

分别给出用逆变换法、舍选法、复合法生成 X 随机数的算法，并比较三种方法的效率。

Chapter 6

非均匀随机数生成

6.1 逆变换法

定理 6.1. 设 X 为连续型随机变量，取值于区间 (a,b)(可包括 $\pm\infty$ 和端点)，X 的密度在 (a,b) 上取正值，X 的分布函数为 $F(x)$，$U \sim U(0,1)$，则 $Y = F^{-1}(U) \sim F(\cdot)$。

定理 6.2. 设 X 为离散型随机变量，取值于集合 $\{a_1, a_2, ...\}(a_1 < a_2 < ...)$，$F(x)$ 为 X 的分布函数，$U \sim U(0,1)$，根据 U 的值定义随机变量 Y 为

$$Y = a_i \text{ 当且仅当 } F(a_{i-1}) < U \leq F(a_i),\ i = 1, 2, ...$$

(定义 $F(a_0) = 0$) 则 $Y \sim F(y)$。

6.2 用逆变换法生成离散型随机数

例 6.1 (离散均匀分布)**.** 如果随机变量 X 在 $\{1, 2, ..., m\}$ 中取值且 $P(X = i) = \frac{1}{m}, i = 1, 2, ..., m$，则称 X 服从**离散均匀分布**。

用均匀分布生成：

```
rng.discrete.uni <- function(n=1, m=2){
  ceiling(m*runif(n))
}
```

其中 n 是要生成的个数，m 是离散均匀分布的取值个数。

尝试用它生成 60 次骰子点数：

```
rng.discrete.uni(60, m=6)
##  [1] 3 1 5 4 2 2 4 3 4 4 6 5 5 6 3 4 5 2 3 1 5 6 2 4 6 5 1 3 3 4 3 2 2 1 4 6 2
## [39] 1 6 3 3 5 2 3 2 1 6 5 1 1 3 5 5 3 5 5 4 5 4
```

实际上, R 中已经有这样的函数。sample.int(m, size=n, replace=TRUE) 可以从 $\{1, 2, \dots, m\}$ 中随机有放回地抽取 n 个。如

```
sample.int(6, size=60, replace=TRUE)
##  [1] 1 5 1 4 3 2 1 5 3 2 4 2 1 5 4 6 5 6 4 6 1 4 5 4 2 2 1 4 6 6 6 2 1 2 5 6 2
## [39] 2 5 3 3 1 2 6 4 1 3 4 2 2 4 4 3 4 2 6 1 1 1
```

例 6.2 (一般有限离散分布 (*))**.** 设 X 为一个仅在 $\{a_1, a_2, \dots, a_m\}$ 中取值的离散随机变量，$P(X = a_i) = p_i, i = 1, 2, \dots, m$。为了由 $U \sim \mathrm{U}[0, 1]$ 生成 X 的随机数，可以利用定理 2.2.2，即当且仅当 $F(a_{i-1}) < U \leq F(a_i)$ 时取 $X = a_i$。

第一种算法的 Julia 程序版本：

```
function rnddis_v1(x, prob, n)
    Fvs = cumsum(prob)
    m = length(prob)
    y = similar(x, (n,))
    for k=1:n
        U = rand()
        for i=1:m
            if U <= Fvs[i]
                y[k] = x[i]
                break
```

```
            end
        end
    end

    return y
end
```

以 $P(X=1)=0.1$, $P(X=2)=0.3$, $P(X=3)=0.6$ 为例。测试 1000 个：

```
using Random, StatsBase
Random.seed!(101)
y = rnddis_v1(1:3, [0.1, 0.3, 0.6], 1000)
counts(y)
## counts(y) = [91, 319, 590]
```

上面的算法依次判断 U 是否小于等于 $F(a_1)$，是否小于等于 $F(a_2)$，...，是否小于等于 $F(a_m)=1$。一旦条件成立就不再继续判断，所以排在前面的判断成立概率越大，平均需要的判断次数越少。只要重排 X 的取值次序使得取值概率大的排在前面就可以改进效率。但是要注意重排过程也需要耗费时间。

改进后的 Julia 程序:

```
function rnddis_v2(x, prob, n)
    m = length(x)
    ind = sortperm(prob, rev=true)
    xs = copy(x)[ind]
    Fvs = cumsum(prob[ind])
    y = similar(x, (n,))
    for k=1:n
        U = rand()
        for i=1:m
            if U <= Fvs[i]
                y[k] = xs[i]
                break
```

```
            end
        end
    end

    return y
end
```

经过测试，第二个版本一般效率不如第一个版本，仅在有许多小概率类别时，第二个版本才有轻微的优势，如：

```
using BenchmarkTools
prob = [fill(0.01, 50); 0.5]
@btime y = rnddis_v1(1:length(prob), prob, 100_000);
## 2.440 ms (4 allocations: 781.81 KiB)
@btime y = rnddis_v2(1:length(prob), prob, 100_000);
## 1.422 ms (8 allocations: 783.28 KiB)
```

事实上，R 函数 `sample()` 已经可以对有限离散进行有放回抽样，做法是

```
sample(x, size=n, replace=TRUE, prob=prob)
```

如

```
set.seed(101)
x <- sample(1:3, size=1000, replace=TRUE,
    prob=c(0.1, 0.3, 0.6))
prop.table(table(x))
## x
##     1     2     3
## 0.098 0.299 0.603
```

例 6.3 (无放回抽样 (*)). 生成 $(1, 2, \dots, n)$ 的一个随机排列。

第一种想法是从 $A = (1, 2, \dots, n)$ 中随机抽取一个，设为 i_1，令 $B = (i_1)$；然后把 i_1 从 A 中剔除，从剩下的 $n-1$ 个元素中继续随机抽取一个，设为 i_2，令 $B = (i_1, i_2)$，如此重复直至 A 中的元素都被抽取到 B 中。

这种做法需要两个向量 A 和 B 来存储。为了减少存储的使用，A 中仅剩 k 个元素时，向量 A 的后面 $n-k$ 个位置可以用来存放已经抽取出来的元素。从 A 的前 k 个中随机抽取一个后，可以把这个元素放在 A_k 的位置并把原来 A_k 填在抽取产生的空位。程序如下：

```
function randperm(n)
    x = collect(1:n)
    for k=n:-1:2
        i = ceil(Int, k * rand())
        if i   k
            x[k], x[i] = x[i], x[k]
        end
    end
    return x
end
using Random
Random.seed!(101)
@show randperm(5);
## randperm(5) = [3, 2, 5, 1, 4]
```

如果仅需要从 $\{1,2,\ldots,n\}$ 中随机无放回第抽取 r 个，只要修改上面程序中的循环使其只执行 r 次即可。程序为：

```
function randperm(n, size=n)
    x = collect(1:n)
    for k=n:-1:(n-size+2)
        i = ceil(Int, k * rand())
        if i   k
            x[k], x[i] = x[i], x[k]
        end
    end
    return x[(n-size+1):n]
end
```

比如，从 $1:10$ 中随机无放回地抽取 5 个：

```
using Random
Random.seed!(101)
@show randperm(10, 5);
## randperm(10, 5) = [6, 4, 3, 2, 8]
```

随机无放回抽取 r 个可以假设 $r \leq \frac{n}{2}$, 否则只要抽取余集就可以了。

R 已有进行随机无放回抽取的函数。`sample.int(n)` 就可以实现对 $\{1, 2, \ldots, n\}$ 的一个随机排列。如

```
set.seed(101)
sample.int(10)
##  [1]  9 10  6  7  1  2  3  8  5  4
```

用 `sample.int(n, size)` 可以从 $\{1, 2, \ldots, n\}$ 随机无放回地抽取 `size` 个，如

```
sample.int(10, size=5)
## [1] 6 3 8 9 1
```

例 6.4 (几何分布随机数). 设随机变量 X 表示在成功概率为 $p(0 < p < 1)$ 的独立重复试验中首次成功所需的试验次数，则 X 的概率分布为

$$P(X = k) = pq^{k-1}, k = 1, 2, \ldots, \ (q = 1 - p)$$

称 X 服从几何分布, 记为 $X \sim \text{Geom}(p)$。

设 $U \sim \text{U}(0, 1)$，注意到

$$\begin{aligned} F(k) =& P(X \leq k) = P(\text{在前}k\text{次试验中至少一次成功}) \\ =& 1 - P(\text{前}k\text{次试验都失败}) \\ =& 1 - q^k, \ k = 1, 2, \ldots \end{aligned}$$

利用定理6.2, 生成 X 的方法为当且仅当 $1 - q^{k-1} < U \leq 1 - q^k$ 时取 $X = k$, $k = 1, 2, \ldots$。此条件等价于

$$q^k \leq 1 - U < q^{k-1}$$

取

$$\begin{aligned} X &= \min\{k : q^k \le 1-U\} \\ &= \min\{k : k\log(q) \le \log(1-U)\} \\ &= \min\{k : k \ge \frac{\log(1-U)}{\log(q)}\} \\ &= \text{ceil}\left(\frac{\log(1-U)}{\log(q)}\right) \end{aligned}$$

在没有指定底数时，$\log(\cdot)$ 默认使用自然对数。注意到 $1-U$ 也是服从 U(0,1) 分布的，所以只要取

$$X = \text{ceil}\left(\frac{\ln(U)}{\ln(q)}\right).$$

则 X 服从几何分布。

```
## 输入
##   - n: 需要输出的个数
##   - p: 成功概率
rng.geom <- function(n=1, p=0.5){
  ceiling(log(runif(n)) / log(1-p))
}
```

测试：

```
set.seed(101)
x <- rng.geom(10000, p=0.1)
mean(x) # EX=1/p
## [1] 9.9389
```

事实上，`rgeom(size, prob)` 生成 `size` 个成功概率为 `prob` 的几何分布随机数。

例 6.5 (用几何分布生成独立试验序列 (*))**.** 产生 $X_1, X_2, \dots, X_N$ iid b(1,p)，即

$$P(X_i = 1) = p = 1 - P(X_i = 0),\ i = 1, 2, \dots, N$$

设 $(U_1, U_2, \dots, U_n)$ iid U(0,1)，则当 $U_i \le p$ 时取 $X_i = 1$，当 $U_i > p$ 时取 $X_i = 0$ 可以构造独立试验序列 $(X_1, X_2, \dots, X_N)$。

当 p 比较小的时候利用几何分布可以更快地构造独立试验序列。

想法是生成首次成功时间，则在成功之前的试验都是失败，然后再生成下次成功时间，两次成功之间的试验为失败，一直到凑够 N 次试验为止。Julia 程序：

```
function randb(n=1, p=0.5)
    x = fill(0, n)
    k = 0
    while k <= n
        # 几何分布随机数:
        T = ceil(Int, log(rand())/log(1-p))
        if(k+T <= n)
            x[k+T] = 1
        end
        k += T
    end

    return x
end
```

测试：

```
Random.seed!(101)
x = randb(1000_000, 0.01)
@show counts(x);
## counts(x) = [989958, 10042]
```

例 6.6 (二项分布随机数 (*))**.** 设 X 为 n 次成功概率为 p 的独立重复试验的成功次数，则

$$p_k = P(X = k) = \binom{n}{k} p^k (1-p)^{n-k},\ k = 0, 1, \dots, n\ (0 < p < 1)$$

称 X 服从二项分布，记为 $X \sim \mathrm{B}(n, p)$。$\binom{n}{k} = \frac{n!}{k!(n-k)!}$ 是从 n 个中取出 k 个的不同组合的个数。

为了生成二项分布随机数，可以生成 n 个两点分布随机数（长度为 n 的独立试验序列），其和为一个二项分布随机数，依次从独立试验序列中每 n 个求和就可以得到二项分布随机数序列。这样的算法效率较差。

设法用定理6.2构造二项分布随机数。二项分布取值概率有如下递推式:

$$p_{k+1} = \frac{n-k}{k+1}\frac{p}{1-p}p_k,\ k = 0, 1, \dots, n-1$$

所以在利用定理6.2构造二项分布随机数时，可以递推计算

$$F(k+1) = F(k) + p_{k+1} = F(k) + \frac{n-k}{k+1}\frac{p}{1-p}p_k,\ k = 0, 1, \dots, n-1$$

初步的 Julia 程序如下：

```
## 输入:
##     - n: 需要输出的个数（注意不是二项分布参数!）
##     - size: 二项分布参数中的试验次数
function randbinom_v1(n, size=1, prob=0.5)
    x = fill(0, n)
    for i = 1:n
        U = rand()
        k = 0
        cc = prob/(1-prob)
        a = (1-prob)^size
        F = a
        while U > F
            a *= cc*(size-k)/(k+1)
            F += a
            k += 1
        end
        x[i] = k
    end

    x
end
```

测试：

```
Random.seed!(101)
x = randbinom_v1(10000, 10, 0.3)
@show mean(x);
## mean(x) = 3.0108
```

上面的程序每生成一个随机数都要计算许多个分布函数值。如果需要生成的随机数很多，而二项分布的试验次数参数不大，应该预先将这些分布函数值存储在一个向量中。改进版本如：

```
## 输入:
##     - n: 需要输出的个数（注意不是二项分布参数!）
##     - size: 二项分布参数中的试验次数
function randbinom_v2(n, size=1, prob=0.5)
    x = fill(0, n)
    Fvs = zeros(size+1)
    k = 0
    cc = prob/(1-prob)
    a = (1-prob)^size
    Fvs[0+1] = a
    for k = 0:(size-1)
        a *= cc*(size-k)/(k+1)
        Fvs[(k+1)+1] = Fvs[k+1] + a
    end

    Uvs = rand(n)
    for i = 1:n
        U = Uvs[i]
        k = 0
        while U > Fvs[k+1]
            k += 1
        end
        x[i] = k
```

```
    end

    return x
end
```

当 $p > \frac{1}{2}$ 时，可以用上述算法生成 $Y \sim \mathrm{B}(n, 1-p)$，令 $X = n - Y$ 则 $X \sim \mathrm{B}(n, p)$，可以减少判断次数。

上面的算法先判断 X 是否应该取 0，如不是再判断 X 是否应该取 1，……，判断次数为 X 的值加 1，所以平均判断次数为 $EX + 1 = np + 1$。当 np 较大时，应该先判断取值概率较大的那些值。$P(X = k)$ 在 np 附近达到最大值。令 $K = \mathrm{floor}(np)$，应先判断要生成的随机变量 X 是小于等于 K 还是大于 K。先计算出 $F(K)$。当 $U \le F(K)$ 时，X 取值小于等于 K，这时可依次判断 X 是否应取 $K, K-1, \ldots, 0$，在这个过程中可以反向递推计算各个 $F(k)$ 的值。当 $U > F(K)$ 时，X 取值大于 K，这时可依次判断 X 是否应取 $K+1, K+2, \ldots, n$，在这个过程中可以递推计算各个 $F(k)$ 的值。

改进后需要的判断次数约为 $|X - np| + 1$，因为 n 较大时二项分布近似正态分布 $\mathrm{N}(np, np(1-p))$，所以平均判断次数约为

$$\begin{aligned} & 1 + E|X - np| \\ =& 1 + \sqrt{np(1-p)} E\left|\frac{X - np}{\sqrt{np(1-p)}}\right| \\ \approx& 1 + \sqrt{np(1-p)} E|Z| \quad (\text{其中} Z \sim \mathrm{N}(0,1)) \\ =& 1 + 0.798\sqrt{p(1-p)} \cdot \sqrt{n}. \end{aligned}$$

当 $n \to \infty$ 时判断次数比原来的 $O(n)$ 降低到了 $O(\sqrt{n})$。

例 6.7 (泊松分布随机数 (*))**.** 设离散型随机变量 X 分布为

$$p_k = P(X = k) = e^{-\lambda}\frac{\lambda^k}{k!},\ k = 0, 1, 2, \ldots (\lambda > 0)$$

称 X 服从泊松分布，记为 $X \sim \mathrm{Poisson}(\lambda)$。

易见

$$p_{k+1} = \frac{\lambda}{k+1} p_k,\ k = 0, 1, 2, \ldots$$

所以在利用定理6.2构造泊松随机数时，可以递推计算

$$F(k+1) = F(k) + p_{k+1} = F(k) + \frac{\lambda}{k+1} p_k,\ k = 0, 1, 2, ...$$

Julia 程序如下：

```
## 输入:
##     - n: 需要输出的个数（注意不是二项分布参数!）
##     - lambda: 泊松分布的速率参数
function randpois_v1(n,  =1)
    x = fill(0, n)
    for i = 1:n
        U = rand()
        p = exp(- )
        F = p
        k = 0
        while U > F
            p *=  /(k+1)
            F += p
            k += 1
        end
        x[i] = k
    end

    x
end
```

测试：

```
Random.seed!(101)
x = randpois_v1(10000, 3)
@show mean(x);
## mean(x) = 3.0057
```

这样的算法先判断是否令 X 取 0，不成立则再判断是否令 X 取 1，如此重复，判断的次数为 X 的值加 1，所以平均判断次数需要 $EX+1 = \lambda+1$ 次，当 λ 较

小时这种方法效率不错；如果 λ 比较大，这时 $p(k) = e^{-\lambda}\frac{\lambda^k}{k!}$ 在最接近于 λ 的两个整数之一上最大，应该先判断这两个整数及其邻近的点。令 $K = \text{floor}(\lambda)$, 先递推计算出 $F(K)$ 和 $F(K+1)$，然后判断 $F(K) < U \le F(K+1)$ 是否成立，如果成立就令 $X = K+1$; 否则，如果 $U \le F(K)$ 则依次判断是否应取 X 为 $K, K-1, \ldots, 0$, 在这个过程中可以反向递推计算各 $F(k)$；如果 $U > F(K+1)$ 则依次判断是否应取 X 为 $K+2, K+3, \ldots$，这个过程中仍然用递推计算各 $F(k)$。

这样的改进的算法需要的判断次数约为 $|X-\lambda|+1$，所以平均判断次数为 $E|X-\lambda|+1$。因为 λ 较大时泊松分布渐近正态分布 $\text{N}(\lambda, \lambda)$, 所以平均判断次数约为

$$\begin{aligned} 1 + E|X-\lambda| =& 1 + \sqrt{\lambda} E\left|\frac{X-\lambda}{\sqrt{\lambda}}\right| \\ \approx& 1 + \sqrt{\lambda} E|Z| \quad (\text{where } Z \sim \text{N}(0,1)) \\ =& 1 + 0.798\sqrt{\lambda} \end{aligned}$$

当 λ 很大时平均判断次数由 $O(\lambda)$ 降低到了 $O(\sqrt{\lambda})$。

6.3 用逆变换法生成连续型随机数

连续型分布的随机数都可以用逆变换法生成：$X = F^{-1}(U)$, $U \sim \text{U}(0,1)$。对于反函数 F^{-1} 容易计算的情形，逆变换法是最方便的。

例 6.8 (三角形分布)**.** 用逆变换法生成各种三角形分布的随机数。

Beta(2,1) 分布的分布密度和分布函数分别为

$$p(x) = 2x,\ x \in [0,1] \quad F(x) = x^2,\ x \in [0,1].$$

密度函数为三角形。若 $U \sim \text{U}(0,1)$，则 $X = \sqrt{U}$ 为 Beta(2,1) 随机数。

Beta(1,2) 分布的分布密度和分布函数分别为

$$p(x) = 2(1-x),\ x \in [0,1] \quad F(x) = 1-(1-x)^2,\ x \in [0,1].$$

密度函数为三角形。若 $U \sim \text{U}(0,1)$，则 $X = 1 - \sqrt{1-U}$ 为 Beta(1,2) 随机数, 因为 U 与 $1-U$ 同分布所以 $X = 1 - \sqrt{U}$ 也是 Beta(1,2) 随机数。

对于 $0 < m < 1$，三角形分布 Tri(0,1,m) 的密度函数为

$$p(x) = \begin{cases} \frac{2}{m}x, & 0 < x \le m \\ \frac{2}{1-m}(1-x), & m < x < 1. \end{cases}$$

分布函数为

$$F(x) = \begin{cases} \frac{x^2}{m}, & 0 < x \le m \\ 1 - \frac{(1-x)^2}{1-m}, & m < x < 1. \end{cases}$$

反函数为

$$F^{-1}(u) = \begin{cases} \sqrt{mu}, & 0 < u \le m \\ 1 - \sqrt{(1-m)(1-u)}, & m < u < 1. \end{cases}$$

所以若 $U \sim \mathrm{U}(0,1)$，则 $X = F^{-1}(U)$ 服从三角形分布 Tri(0,1,m)。

标准右三角形分布 Beta(2,1) 的随机数的 R 程序：

```
rng.rtri <- function(n){
  sqrt(runif(n))
}
```

标准左三角形分布 Beta(1,2) 的随机数的 R 函数：

```
rng.ltri <- function(n){
  1 - sqrt(runif(n))
}
```

标准三角形分布 $\mathrm{Tri}(0,1,m)$ 的随机数：

```
rng.tri <- function(n, m){
  u <- runif(n)
  ifelse(u <= m, sqrt(m*u), 1 - sqrt((1-m)*(1-u)))
}
```

用直方图测试：

```r
pdftri <- function(x, m) {
  ifelse(x <= m, 2/m*x, 2/(1-m)*(1-x))
}
set.seed(101)
x = rng.tri(10000, 0.6)
hist(x, breaks=15, prob=TRUE, ylim=c(0,2))
curve(pdftri(x, 0.6), 0, 1, add=TRUE, col="red")
```

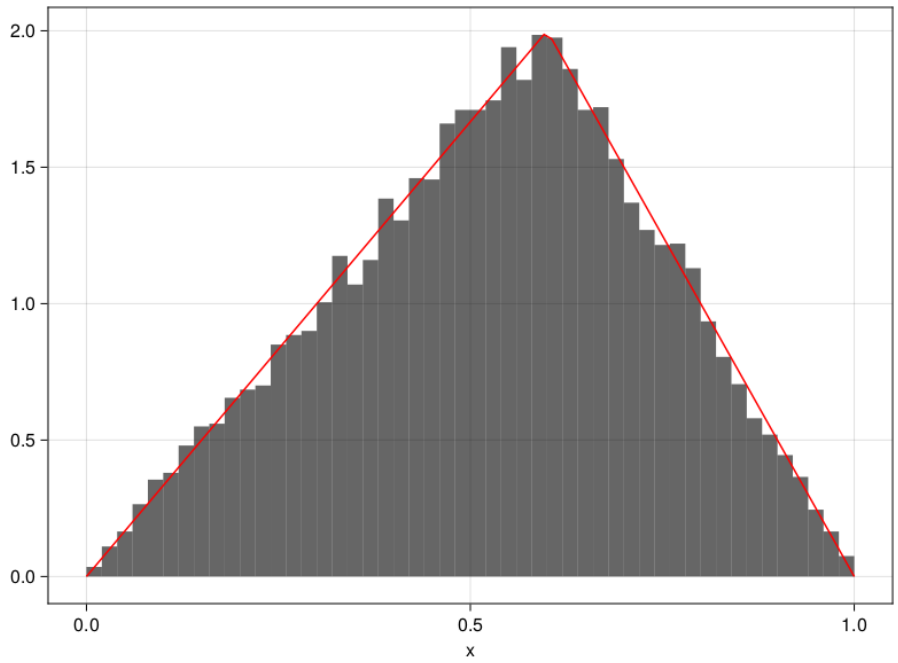
用 Julia 编程：

```julia
function randtri(n, m=0.5)
    Finv(u) = u <= m ? sqrt(m*u) : 1 - sqrt((1-m)*(1-u))
    Finv.(rand(n))
end
```

用直方图和密度曲线测试：

```julia
function hist_with_pdf(
    x, pdf, x1=minimum(x), x2=maximum(x))
    xs = range(x1, x2, length=100)
    ys = pdf.(xs)
    plt = data((; x=x)) * mapping(:x) *
        histogram(bins=50, normalization=:pdf)
    plt2 = data((; x=xs, y=ys)) * mapping(:x, :y) *
        visual(Lines, color=:red)
    draw(plt+plt2)
end

using Random, Distributions
using CairoMakie, AlgebraOfGraphics
CairoMakie.activate!()
m = 0.6
x = randtri(10000, m)
```

```
f = x -> x <= m ? 2/m*x : 2/(1-m)*(1-x)
dr = hist_with_pdf(x, f, 0.0, 1.0)
```



例 6.9 (指数分布随机数)**.** 生成指数分布的随机数。

服从指数分布 $\mathrm{Exp}(\lambda)$ $(\lambda > 0)$ 的随机变量 X 的分布密度和分布函数分别为

$$\begin{aligned} p(x) =& \lambda e^{-\lambda x},\ x > 0 \\ F(x) =& 1 - e^{-\lambda x},\ x > 0. \end{aligned}$$

反函数为

$$F^{-1}(u) = -\lambda^{-1} \log(1-u).$$

所以 $U \sim \mathrm{U}(0,1)$ 时 $X = -\lambda^{-1}\log(1-U)$ 服从 $\mathrm{Exp}(\lambda)$。因为 $1-U$ 与 U 同分布，所以取 $X = -\lambda^{-1}\log U$ 也服从 $\mathrm{Exp}(\lambda)$。

当 $\lambda = 1$ 时 $\mathrm{Exp}(1)$ 叫做标准指数分布，$-\log U$ 服从标准指数分布，标准指数分布乘以 λ^{-1} 即为 $\mathrm{Exp}(\lambda)$ 分布。

指数分布随机数发生器 R 函数：

```
rng.exp <- function(n, rate=1){
  -log(runif(n))/rate
}
```

R 已经提供了 `rexp(n, rate=1)` 函数。使用 Julia 语言编程：

```
function randexp(n,  =1)
    -log.(rand(n)) ./
end
```

例 6.10 (利用指数分布随机数产生泊松随机数 (*))**.** 利用指数分布和泊松过程的关系生成泊松随机数。

$N(t)$ 为到时刻 t 为止到来的事件个数 $(t \geq 0)$，如果两个事件到来之间的间隔都服从独立的 Exp(λ) 分布则 $N(t)$ 为泊松过程，且 $N = N(1)$ 服从参数为 λ 的泊松分布。若 $U_1, U_2, \ldots$ 是独立的 U(0,1) 随机变量列，$X_1, X_2, \ldots$ 是独立的 Exp(λ) 随机变量列，则

$$N = \max\{n : \sum_{i=1}^{n} X_i \leq 1\}.$$

X_i 可以用 $-\lambda^{-1} \ln U_i$ 生成，所以

$$\begin{aligned} N &= \max\{n : -\lambda^{-1} \sum_{i=1}^{n} \ln U_i \leq 1\} \\ &= \max\{n : U_1 U_2 \ldots U_n \geq e^{-\lambda}\}. \end{aligned}$$

可以这样生成泊松随机数：相继生成均匀随机数，直至其连乘积小于 $e^{-\lambda}$, 取 n 为使用的均匀随机数个数减 1，即

$$N = \min\{n : U_1 U_2 \ldots U_n < e^{-\lambda}\} - 1.$$

Julia 程序：

```
function randpois_v2(n,  =1)
    x = fill(0, n)
    a = exp(- )
```

```
    for i = 1:n
        p = 1
        k = 0
        while true
            k += 1
            p *= rand()
            if p < a
                break
            end
        end
        x[i] = k-1
    end
    x
end
```

测试:

```
Random.seed!(101)
x = randpois_v2(10000, 3)
@show mean(x);
## mean(x) = 3.0007
```

6.4 利用变换生成随机数

定理 6.3. 设随机变量 X 有密度 $p(x)$，$Y = g(X)$ 是 X 的函数，函数 $g(\cdot)$ 有反函数 $x = g^{-1}(y) = h(y)$，$h(y)$ 有一阶连续导数，则 Y 有密度

$$f(y) = p(h(y)) \cdot |h'(y)|.$$

定理 6.4. 设随机向量 (X, Y) 具有联合密度函数 $p(x, y)$，令

$$\begin{cases} u = g_1(x, y), \\ v = g_2(x, y), \end{cases}$$

设 $(u,v)=(g_1(x,y),g_2(x,y))$ 的反变换存在唯一，记为

$$\begin{cases} x=h_1(u,v), \\ y=h_2(u,v), \end{cases}$$

设 h_1,h_2 的一阶偏导数存在，反变换的 *Jacobi* 行列式

$$J=\begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{vmatrix} \neq 0,$$

则随机向量 (U,V) 的联合密度为

$$f(u,v)=p(h_1(u,v),h_2(u,v))\cdot|J|.$$

有些分布之间有已知的关系，可以利用这样的关系产生随机数。如:

- 二项分布 B(n,p) 是 n 个伯努利分布 b(1,p) 之和，可以产生 n 个伯努利分布 b(1,p) 随机数求和得到二项分布随机数。
- 如果 U 服从标准均匀分布 U(0,1), 则 $X=a+(b-a)U$ 服从 U(a,b)。
- 如果 Z 服从标准正态分布，则可以用 $X=\mu+\sigma Z$ 服从 N(μ,σ^2) 分布。
- 如果 Z 服从标准指数分布 Exp(1), 则 $X=\lambda^{-1}Z$ 服从指数分布 Exp(λ)。
- 如果 Z 服从标准伽马分布 Gamma(α,1), 则 $X=\lambda^{-1}Z$ 服从伽马分布 Gamma(α, λ)。
- n 个独立的 Exp(λ) 指数分布随机变量之和服从伽马分布 Gamma(n,λ)。设 $U_1,U_2,\ldots,U_n$ 为独立 U(0,1) 随机变量，令

  $$\begin{aligned} X=&\sum_{i=1}^{n}(-\lambda^{-1})\ln U_i \\ =&-\lambda^{-1}\ln(U_1U_2\ldots U_n) \end{aligned}$$

  则 X 服从 Gamma(n,λ) 分布。
- 若 $Z_1,Z_2,\ldots,Z_n$ 独立同标准正态分布,则 $X=Z_1^2+Z_2^2+\cdots+Z_n^2\sim\chi^2(n)$ 分布。
- 如果 $X\sim$ Gamma$(\frac{n}{2},1)$, 则 $2X\sim\chi^2(n)$。

- 若 $Y \sim \mathrm{N}(0,1)$, $Z \sim \chi^2(n)$ 且 Y 与 Z 相互独立，则 $X = Y/\sqrt{Z}$ 服从 $\mathrm{t}(n)$ 分布。
- 若 $Y \sim \chi^2(n_1)$, $Z \sim \chi^2(n_2)$, Y 与 Z 相互独立，则 $X = \frac{Y/n_1}{Z/n_2} \sim \mathrm{F}(n_1, n_2)$。

下面的定理给出了一种生成标准正态随机数的算法。

定理 6.5. *设 U_1, U_2 独立且都服从 $U(0,1)$,*

$$X = \sqrt{-2\ln U_1}\cos(2\pi U_2),$$
$$Y = \sqrt{-2\ln U_1}\sin(2\pi U_2).$$

则 X, Y 独立且都服从标准正态分布。叫做 Box-Muller 变换。

例 6.11 (用 Box-Muller 变换生成正态随机数)**.** 只要生成两个独立的 U(0,1) 随机变量 U_1 和 U_2，按照上述定理就可以生成两个独立的标准正态分布随机数。

R 程序实现：

```
rng.normbm <- function(n, mu, sigma){
  n2 <- ceiling(n/2); nn <- 2*n2
  amp <- sqrt(-2*log(runif(n2)))
  ang <- 2*pi*runif(n2)
  mu + sigma*c(amp*cos(ang), amp*sin(ang))[1:n]
}
```

6.5 舍选法

设随机变量 Z 的分布函数很难求反函数 $F^{-1}(u)$。如果 Z 取值于有限区间 $[a,b]$ 且 $p(x)$ 有上界 M:

$$p(x) \le M, \ \forall x \in [a,b]$$

则可以用如下算法生成 Z 的随机数：

舍选法 I

until$(Y \le p(X))\{$
　　从 U(0,1) 抽取 U_1, U_2
　　取 $X \leftarrow a + (b-a) * U_1,\ Y \leftarrow M * U_2$
$\}$
输出 $Z \leftarrow X$

这种方法叫做舍选法 I。每次循环生成的 (X, Y) 实际上构成了矩形 $[a, b] \times [0, M]$ 上的二维均匀分布，循环退出条件为 $Y \le p(X)$, 循环退出时 (X, Y) 的值落入了 $p(x)$ 曲线下方，这时 (X, Y) 服从曲边梯形 $\{(x, y) : a \le x \le b, 0 \le y \le p(x)\}$ 上的二维均匀分布，所以循环退出时 $Z = X$ 在 x 附近取值的概率是与 $p(x)$ 成正比的。

定理 6.6. *舍选法 I 产生的 Z 密度为 $p(x)$，算法所需的迭代次数是均值为 $M(b-a)$ 的几何分布随机变量。*

舍选法 I 的优点是只要随机变量在有限区间取值且密度函数有界就可以使用这种方法。但是，舍选法 I 有许多缺点：首先，X 的取值范围必须有界；其次，X 的密度 $p(x)$ 必须有界；第三，当算法中 X 取到 $p(x)$ 值很小的位置时，被拒绝的概率很大, X 被接受的概率实际是曲线 $p(x)$ 下的面积和整个矩形面积之比，当 $p(x)$ 高低变化很大时这个比例很低，算法效率很差。

为了解决第一个和第二个问题，我们抽取 X 时不再从 U(a, b) 抽取，而可以从一个一般密度 $g(x)$ 抽取，称这个密度为 “试投密度”，要求 $g(x)$ 的随机数容易生成。为了解决第三个问题，我们应该力图使得 $g(x)$ 形状与 $p(x)$ 相像，即 $p(x)$ 大时相应 $g(x)$ 也大，$p(x)$ 小时相应 $g(x)$ 也小，这样目标密度小的地方尝试抽取 X 也少。在很多实际问题中，目标密度 (要生成随机数的密度)$p(x)$ 可能本身是未知的，只知道 $\tilde{p}(x) = ap(x)$, 其中 a 为未知常数。要求存在常数 c 使得

$$\frac{\tilde{p}(x)}{g(x)} \le c,\ \forall x$$

这样，如果 $\tilde{p}(x)$ 在接近 $\pm\infty$ 处有定义，需要满足 $\tilde{p}(x) = O(g(x))(x \to \pm\infty)$。选取试投密度 $g(\cdot)$ 时上述常数 c 越小越好。

这时算法改为如下的舍选法 II:

舍选法 II

until $\left(Y \le \frac{\tilde{p}(X)}{(cg(X))}\right)\{$
　　从 $g(x)$ 抽取 X
　　从 U(0,1) 抽取 Y
$\}$
输出 $Z \leftarrow X$

定理 6.7. *舍选法 II 产生的 Z 密度为 $p(x)$，算法所需的迭代次数是均值为 $\frac{c}{a}$ 的几何分布随机变量。*

从定理可以看出，为了提高舍选法 II 的效率，应该取试投密度 $g(x)$ 使得 $g(x)$ 与 $p(x)$ 形状越接近越好, 常数 c 越小越好。

例 6.12 (用舍选法生成 Beta(2,4) 随机数)**.** 用舍选法产生 Beta(2,4) 的随机数，密度为

$$p(x) = 20x(1-x)^3,\ 0 < x < 1.$$

可以用舍选法 I，这时

$$M = \max_{0\le x\le 1} p(x) = p(\frac{1}{4}) = \frac{135}{64},$$

平均迭代次数为 $\frac{135}{64} \approx 2.1$。

程序：

```
function randbeta24(n)
    xs = zeros(n)
    local x, y
    for i = 1:n
        while true
            x = rand()
            y = 135/64*rand()
            y <= 20*x*(1-x)^3 && break
        end
        xs[i] = x
```

```
    end
    return xs
end
```

事实上 R 已经提供了 `rbeta(n, shape1, shape2)` 函数。

例 6.13 (用舍选法生成 Gamma(3/2,1) 随机数). 生成 Gamma$(\frac{3}{2},1)$ 的随机数 Z，密度为

$$p(x) = \frac{1}{\Gamma(\frac{3}{2})} x^{\frac{1}{2}} e^{-x},\ x > 0.$$

其中 $\Gamma(\frac{3}{2}) = \frac{\sqrt{\pi}}{2}$。

用舍选法 II。$p(x)$ 形状在尾部与指数分布相近，且 $EZ = \frac{3}{2}$，所以使用期望为 $\frac{3}{2}$ 的指数分布作为试投密度 $g(x) = \frac{2}{3} e^{-\frac{2}{3}x}$, 求常数 c 使得 $\sup(p(x)/g(x)) \le c$。用微分法求得 $p(x)/g(x)$ 最大值点 $\frac{3}{2}$，最大值为 $c = \frac{3\sqrt{3}}{\sqrt{2\pi e}}$, 这时

$$\frac{p(x)}{cg(x)} = \sqrt{\frac{2e}{3}} x^{\frac{1}{2}} e^{-\frac{1}{3}x},$$

平均迭代次数为 $c = \frac{3\sqrt{3}}{\sqrt{2\pi e}} \approx 1.257$.

Julia 程序:

```
function randgamma1(n)
    local a, x, y, xs
    xs = zeros(n)
    a = sqrt(2*exp(1)/3)
    for i=1:n
        while true
            x = -1.5*log(rand())
            y = rand()
            y <= a * sqrt(x) * exp(-1/3*x) && break
        end
        xs[i] = x
    end
    return xs
end
```

例 6.14 (用舍选法生成 Gamma 随机数 (*))**.** 讨论生成一般的 Gamma 分布随机数的问题。

设 Z 服从伽马分布 Gamma(α,λ)，密度为

$$p(x)=\frac{\lambda^\alpha}{\Gamma(\alpha)}x^{\alpha-1}e^{-\lambda x},\ x>0\ (\alpha>0,\lambda>0).$$

Z 的期望值为 α/λ。注意到如果 W 服从 Gamma$(\alpha,1)$ 分布则 W/λ 服从 Gamma(α,λ)，所以只需要考虑 $\lambda=1$ 的情形。这时密度为

$$p(x)=\frac{1}{\Gamma(\alpha)}x^{\alpha-1}e^{-x},\ x>0\ (\alpha>0).$$

为生成 Gamma$(\alpha,1)$ 的随机数，尝试使用指数分布 Exp(t) 作为试投密度 $g(x)$，如何取 t 呢？

令

$$h(x)=\frac{p(x)}{g(x)}=\frac{1}{t\Gamma(\alpha)}x^{\alpha-1}e^{-(1-t)x},$$

t 需要使得 $h(x)$ 有上界且上界最小。当 $0<\alpha<1$ 时 $\lim_{x\to0}h(x)=+\infty$ 所以 $0<\alpha<1$ 时不能使用指数分布作为试投密度。

当 $\alpha=1$ 时伽马分布就是指数分布。

所以首先考虑 $\alpha>1$ 时 t 的取法。用微分法求得 $h(x)$ 的最大值点。事实上，当 $t\geq1$ 时 $\lim_{x\to+\infty}h(x)=\infty$，所以只有 $0<t<1$ 时 $h(x)$ 才有界，令

$$u(x)=x^{a-1}e^{-bx},\ a>1,b>0,x>0$$

则

$$u'(x)=x^{a-2}e^{-bx}((a-1)-bx)$$

解 $u'(x)=0$ 得 $x_0=(a-1)/b$ 且易见 x_0 是 $u(x)$ 的最大值点。于是 $h(x)$ 的最大值为 $x_0=\frac{\alpha-1}{1-t}$。令 $c(t)$ 为 $h(x_0)$ 随 $t\in(0,1)$ 变化的函数，则

$$c(t)=h(x_0)=\frac{1}{\Gamma(\alpha)}(\alpha-1)^{\alpha-1}e^{-(\alpha-1)}\frac{1}{t(1-t)^{\alpha-1}},$$

为了求最小的 $c(t)$ 只要求 $t(1-t)^{\alpha-1}$ 的最大值点，用微分法易得 $t=\frac{1}{\alpha}=\frac{1}{EX}$ 时 $c(t)$ 最小，所以试投密度的期望与 $p(x)$ 期望相同。$c(t)$ 的最大值为

$$c_0=c(\frac{1}{\alpha})=\frac{\alpha^\alpha e^{-(\alpha-1)}}{\Gamma(\alpha)}.$$

当 $t=\frac{1}{\alpha}$ 时

$$h(x)=\frac{\alpha\lambda^{\alpha-1}}{\Gamma(\alpha)}x^{\alpha-1}\exp\left\{-\frac{\alpha-1}{\alpha}x\right\},x>0,\alpha>1.$$

所以 $t=\frac{1}{\alpha}$ 时

$$h_1(x)=\frac{p(x)}{c_0g(x)}=\frac{e^{\alpha-1}}{\alpha^{\alpha-1}}x^{\alpha-1}e^{-\frac{\alpha-1}{\alpha}x},x>0,\alpha>1$$

$h_1(x)$ 的最大值为 1。

$\alpha>1$ 时 Gamma(α,λ) 分布的随机数发生器 Julia 程序：

```
function randgammg(n,  =2.0,  =1.0)
    local xs, x, y, t
    function h(x)
        exp( -1) /  ^( -1) *
        x^( -1) * exp(-( -1)/  *x)
    end

    xs = zeros(n)
    t = 1 /
    for i=1:n
        while true
            x = -log(rand()) / t
            y = rand()
            y <= h(x) && break
        end
        xs[i] = x
    end

    return xs ./
end
```

当伽马分布的参数 $\alpha\in(0,1)$, $\lambda=1$ 时，可以找到如下分为 $0<x\leq 1$ 和

$x > 1$ 两段的控制函数:

$$M(x) = \begin{cases} \frac{1}{\Gamma(\alpha)} x^{\alpha-1}, & 0 < x \le 1, \\ \frac{1}{\Gamma(\alpha)} e^{-x}, & x > 1 \end{cases}$$

使得 $p(x) \le M(x)$，然后把 $M(x)$ 归一化为一个密度就可以作为试投密度。积分得

$$c_1 = \int_0^\infty M(x)\,dx = \frac{1}{\Gamma(\alpha)} \left(\frac{1}{\alpha} + e^{-1} \right)$$

令

$$g(x) = \frac{1}{c_1} M(x) = \frac{1}{\frac{1}{\alpha} + e^{-1}} \cdot \begin{cases} x^{\alpha-1}, & 0 < x \le 1, \\ e^{-x}, & x > 1 \end{cases}$$

设 $G(x)$ 为对应的分布函数，则

$$G(x) = \begin{cases} \frac{1}{1+\alpha e^{-1}} x^\alpha, & 0 < x \le 1, \\ \frac{1}{1+\alpha e^{-1}} + \frac{\alpha}{1+\alpha e^{-1}} (e^{-1} - e^{-x}), & x > 1 \end{cases}$$

求反函数 $G^{-1}(p)$。$G(1) = \frac{1}{1+\alpha e^{-1}}$，反函数也分 $p \le \frac{1}{1+\alpha e^{-1}}$ 和 $p > \frac{1}{1+\alpha e^{-1}}$ 两段。解得

$$G^{-1}(p) = \begin{cases} [(1+\alpha e^{-1})p]^{\frac{1}{\alpha}}, & 0 < p \le \frac{1}{1+\alpha e^{-1}}, \\ -\ln\left(e^{-1} - \frac{1}{\alpha}\left(p(1+\alpha e^{-1}) - 1\right)\right), & \frac{1}{1+\alpha e^{-1}} < p < 1 \end{cases}$$

对 $0 < \alpha < 1$，令

$$h_2(x) = \frac{p(x)}{g(x)} = \begin{cases} \frac{\frac{1}{\alpha}+e^{-1}}{\Gamma(\alpha)} e^{-x}, & 0 < x \le 1, \\ \frac{\frac{1}{\alpha}+e^{-1}}{\Gamma(\alpha)} x^{\alpha-1}, & x > 1 \end{cases}$$

这个分段函数的两段的最大值都分别在区间的左端点达到，所以

$$c_3 = \max_{x>0} h_2(x) = \frac{\frac{1}{\alpha} + e^{-1}}{\Gamma(\alpha)}$$

令

$$h_3(x) = \frac{p(x)}{c_3 g(x)} = \begin{cases} e^{-x}, & 0 < x \le 1, \\ x^{\alpha-1}, & x > 1 \end{cases}$$

则 $h_3(x)$ 最大值为 1。

$0 < \alpha < 1$ 时 Gamma(α, λ) 分布的随机数发生器 Julia 程序：

```
function randgammas(n,  =0.5,  =1.0)
    local xs, x, y, t
    h(x) = x < 1 ? exp(-x) : x^( -1)
    function Ginv(p)
        if p < 1/(1+ *exp(-1))
            ((1 +  *exp(-1))*p)^(1/ )
        else
            -log(exp(-1) - 1/ *(p*(1 +  *exp(-1)) - 1))
        end
    end

    xs = zeros(n)
    t = 1 /
    for i=1:n
        while true
            x = Ginv(rand())
            y = rand()
            y <= h(x) && break
        end
        xs[i] = x
    end

    return xs ./
end
```

例 6.15 (用舍选法生成正态随机数 (*))**.** 如果 $X \sim \mathrm{N}(0,1)$ 则 $Y = \mu + \sigma X \sim \mathrm{N}(\mu, \sigma^2)$，所以只要产生 N(0,1) 随机数 Z。用舍选法 II，标准正态分布密度在尾部正比于 $e^{-\frac{1}{2}x^2}$，用标准指数分布密度来作为试投密度 (可以像例6.14那样证明在指数分布中参数为 1 的指数分布是最优的)。

因为指数分布是非负的，所以先产生 $|Z|$ 的随机数，易见 $|Z|$ 的密度函数为

$$p(x) = \sqrt{\frac{2}{\pi}} e^{-\frac{1}{2}x^2},\ x > 0$$

取 $g(x)=e^{-x}, x>0$,

$$h(x)=\frac{p(x)}{g(x)}=\sqrt{\frac{2}{\pi}}e^{x-\frac{1}{2}x^2},\ x>0$$

为使 $h(x)$ 最大只要使 $x-\frac{1}{2}x^2$ 最大，所以 $h(x)$ 最大值点为 $x_0=1$，取

$$c=h(1)=\sqrt{\frac{2e}{\pi}},\quad \frac{p(x)}{cg(x)}=e^{-\frac{1}{2}(x-1)^2},$$

产生 $|Z|$ 随机数后只需要以各自 $\frac{1}{2}$ 的概率加上正负号。算法迭代次数为 $c=\sqrt{\frac{2e}{\pi}}\approx 1.32$。

例 6.16 (用舍选法和 Box-Muller 变换生成正态随机数 (*))**.** 在 Box-Muller 公式中令 $R=\sqrt{-2\ln U_1}$, $\theta=2\pi U_2$，则 R^2 与 θ 独立，$R^2\sim \mathrm{Exp}(\frac{1}{2})$, $\theta\sim \mathrm{U}(0,2\pi)$, R 是 (X,Y) 的极径，θ 是 (X,Y) 的极角，是一个均匀随机角度。我们只要能生成极角服从 $\mathrm{U}(0,2\pi)$ 的随机点的直角坐标就可以获得 $\cos\theta$ 和 $\sin\theta$ 的值而不需要计算正弦和余弦。

设随机向量 (V_1,V_2) 服从单位圆 $C=\{(x,y):x^2+y^2\le 1\}$ 上的均匀分布 $\mathrm{U}(C)$，则 (V_1,V_2) 的极角 θ 服从 $\mathrm{U}(0,2\pi)$。(V_1,V_2) 可以用舍选法产生。得到这样的 (V_1,V_2) 后只要令

$$X=\sqrt{-2\ln U_1}\frac{V_1}{\sqrt{V_1^2+V_2^2}},\ Y=\sqrt{-2\ln U_1}\frac{V_1}{\sqrt{V_2^2+V_2^2}}$$

则 X,Y 服从独立的标准正态分布。

另外还可以证明，若 (V_1,V_2) 服从单位圆上的均匀分布 $\mathrm{U}(C)$, 则 $S=V_1^2+V_2^2$ 服从 U(0,1) 且与 (V_1,V_2) 的极角 θ 独立。这样，U_1 可以用 S 代替。

Julia 程序：

```
function randnormdsbm(n,  =0.0,  =1.0)
    local n2, nn, xs, v1, v2, s, amp
    n2 = ceil(Int, n/2)
    nn = 2*n2
    xs = zeros(nn)
    for i=1:n2
        while true
```

```
            v1 = 2*rand()-1
            v2 = 2*rand()-1
            s = v1^2 + v2^2
            s <= 1 && break
        end
        amp = sqrt(-2*log(s)/s)
        xs[i] = amp*v1
        xs[n2+i] = amp*v2
    end

    if n<nn
        pop!(xs)
    end
    return   .+   .* xs
end
```

例 6.17 (条件伽马分布随机数). 生成取值大于 5 的 Gamma(2,1) 的随机数 Z。密度为

$$p(x) = \frac{xe^{-x}}{\int_5^\infty ue^{-u}du} = \frac{xe^{-x}}{6e^{-5}},\ x > 5$$

一个显然的算法为:

```
until(X > 5){
  从Gamma(2,1) 抽取 X
}
输出 X
```

这样的算法舍去小于 5 的试投值比较多，接受概率只有约 0.04，效率很低。

仍使用舍选法 II，试投密度使用大于 5 的指数分布。因为 Gamma(2,1) 期望为 $2/1 = 2$，使用大于 5 的 Exp(1/2)，密度为

$$g(x) = \frac{e^{-\frac{1}{2}x}}{\int_5^\infty e^{-\frac{1}{2}u}du} = \frac{1}{2}e^{\frac{5}{2}}e^{-\frac{1}{2}x},\ x > 5$$

比值

$$h(x)=\frac{p(x)}{g(x)}=\frac{1}{3}e^{\frac{5}{2}}xe^{-\frac{1}{2}x},\ x>5$$

是单调减函数，所以 $h(x)\le c=h(5)=\frac{5}{3}$,

$$\frac{p(x)}{cg(x)}=\frac{1}{5}e^{\frac{5}{2}}xe^{-\frac{1}{2}x}.$$

如何生成 $g(x)$ 的随机数呢？注意指数分布的随机变量 ξ 有如下的无记忆性：$P(\xi>a+b|\xi>a)=P(\xi>b)$，所以在 $\xi>a$ 条件下的指数分布，可以看成是从 a 出发的一个指数分布，即在 $\xi>a$ 条件下的指数分布与 $\xi+a$ 同分布。所以 $g(x)$ 的随机数可以用指数分布随机数加 5 实现。Julia 程序如下：

```
function randgammatr(n)
    local xs, x, y
    h(x) = 1/5*exp(2.5)*x*exp(-0.5*x)
    xs = zeros(n)
    for i=1:n
        while true
            x = 5 - 2*log(rand())
            y = rand()
            y <= h(x) && break
        end
        xs[i] = x
    end

    return xs
end
```

6.6 复合法

设离散型随机变量 I 的概率分布为

$$P(I=i)=\alpha_i,\ i=1,2,\dots,m$$

若随机变量 $Z_1, Z_2, \ldots, Z_m$ 都是离散型随机变量，定义随机变量 X 的值为

$$X = \begin{cases} Z_1, & \text{当} I = 1, \\ Z_2, & \text{当} I = 2, \\ \ldots & \ldots\ldots \\ Z_m, & \text{当} I = m. \end{cases}$$

则 X 是离散型随机变量，且

$$P(X = x) = \sum_{i=1}^{m} P(X = x | I = i) P(I = i) = \sum_{i=1}^{m} \alpha_i P(Z_i = x).$$

如果 X 的分布可以这样定义，则可以先生成 I 的样本，再根据 I 的值生成 Z_I 的样本，结果就是 X 的样本。

如果上面的 $Z_1, Z_2, \ldots, Z_m$ 都是连续型随机变量，密度函数分别为 $p_1(z), p_2(z), \ldots, p_m(z)$，则 X 的分布函数为

$$\begin{aligned} F(x) =& P(X \le x) = \sum_{i=1}^{m} P(X \le x | I = i) P(I = i) \\ =& \sum_{i=1}^{m} \alpha_i P(Z_i \le x). \end{aligned}$$

于是 X 有分布密度

$$p(x) = F'(x) = \sum_{i=1}^{m} \alpha_i p_i(x).$$

如果 X 的分布密度有这样的形式，可以先生成 I，然后根据 I 的值生成 Z_I 作为 X。

复合法需要比较多的随机数，在其它方法很难实现或效率较低时复合法有它的优势。

例 6.18. 设离散型随机变量 X 的分布为

$$P(X = k) = \begin{cases} 0.05, & j = 1, 2, 3, 4, 5 \\ 0.15, & j = 6, 7, 8, 9, 10 \end{cases}$$

虽然可以当作普通的取有限个值的离散型随机变量来生成，但是 X 的分布明显可以看成如下的混合分布：随机变量 I 分布为

$$P(V = 1) = 0.25,\ P(V = 2) = 0.75$$

随机变量 Z_1 服从 $1,2,3,4,5$ 上的离散均匀分布，随机变量 Z_2 服从 $6,7,8,9,10$ 上的离散均匀分布，X 的分布为 $I=1$ 时取 Z_1, $I=2$ 时取 Z_2。所以，生成 X 的随机数的 R 程序如下：

```
function randmix(n)
    local xs, u1, u2
    xs = zeros(n)
    for i=1:n
        u1, u2 = rand(2)
        if u1 < 0.75
            xs[i] = ceil(Int, 5*u2) + 5
        else
            xs[i] = ceil(Int, 5*u2)
        end
    end

    return xs
end
```

例 6.19 (梯形分布 (*))**.** 对 $0<a<1$，考虑梯形密度

$$p_1(x)=a+2(1-a)x,\ 0<x<1,$$

或

$$p_2(x)=2-a-2(1-a)x,\ 0<x<1$$

因为分布函数为

$$F_1(x)=ax+(1-a)x^2,\ 0<x<1$$

或

$$F_2(x)=(2-a)x-(1-a)x^2,\ 0<x<1$$

分布函数反函数为

$$F_1^{-1}(u)=\frac{-a+\sqrt{a^2+4(1-a)u}}{2(1-a)},\ 0<u<1$$

或

$$F_2^{-1}(u)=\frac{2-a-\sqrt{(2-a)^2-4(1-a)u}}{2(1-a)},\ 0<u<1$$

可以用逆变换法获得 $p_1(x)$ 和 $p_2(x)$ 的随机数。另外，注意 $X \sim p_1(x)$ 当且仅当 $1-X \sim p_2(x)$，为了生成 $p_2(x)$ 的随机数，也可以生成 $X \sim p_1(x)$ 然后用 $1-X$ 作为 $p_2(x)$ 的随机数。

下面给出复合法生成 $p_1(x)$ 随机数的做法。$p_1(x)$ 下的区域可以分解为一个面积等于 a 的矩形和一个面积等于 $1-a$ 的三角形，令

$$\begin{aligned} g_1(x) =& 1,\ 0<x<1, \\ g_2(x) =& 2x,\ 0<x<1, \\ p_1(x) =& a \cdot g_1(x) + \ (1-a) \cdot g_2(x),\ 0<x<1 \end{aligned}$$

可见 $p_1(x)$ 是由一个均匀分布和一个三角形分布复合而成。于是，生成梯形 $p_1(x)$ 的随机数的 R 程序为

```
# 梯形分布密度随机数，用复合法
function randlad(n, a=0.5)
    local xs
    xs = zeros(n)
    for i=1:n
        if rand() < a
            xs[i] = rand()
        else
            xs[i] = sqrt(rand())
        end
    end

    return xs
end
```

习题

习题 1

写程序生成 n 个如下离散型分布随机数: $P(X=1)=1/3$, $P(X=2)=2/3$。

(1) 生成 $n = 100$ 个这样的随机数，计算 X 取 1 的百分比；
(2) 生成 $n = 1000$ 个这样的随机数，计算 X 取 1 的百分比；
(3) 生成 $n = 10000$ 个这样的随机数，计算 X 取 1 的百分比；
(4) 用中心极限定理推导 n 个这样的随机数取 1 的百分比 f_n 的渐近分布，并用此渐近分布检验上面的三组样本是否与 $P(X = 1) = 1/3$ 相符。

习题 2

设随机变量 X 服从离散分布 $P(X = k) = p_k, k = 1, 2, \ldots, m$。编写 R 程序，输入 $\{p_k\}$, 输出 n 个该离散分布的随机数。

习题 3

洗好一副编号分别为 $1, 2, \ldots, 54$ 的纸牌，依次抽取出来，若第 i 次抽取到编号 i 的纸牌则称为成功抽取。编写程序估计成功抽取个数 T 的期望和方差，推导理论公式并与模拟结果进行比较。

习题 4

做投掷两枚骰子的试验，连续试验直到点数之和的所有可能值 2,3,…,12 都出现一次，所需的试验次数记为 T，用随机模拟方法估计 T 的期望和方差。

习题 5

编写例6.6中原始的和改进的生成二项分布随机数的算法并用 R 程序实现，比较两种算法得到的序列是否相同。

习题 6

编写例6.7中原始的和改进的生成泊松分布随机数的算法并用 R 程序实现，比较两种算法得到的序列是否相同。

习题 7

设随机变量 X 表示成功概率为 p 的独立重复试验中第 r 次成功所需要的试验次数，称 X 服从负二项分布。

(1) 利用负二项分布与几何分布的关系构造 X 的随机数。
(2) 直接利用负二项分布的概率分布构造 X 的随机数。

习题 8

用变换法生成如下分布的随机数:

(1) Beta$(\frac{1}{n}, 1)$ 分布，密度为

$$p(x) = \frac{1}{n} x^{\frac{1}{n}-1}, x \in [0, 1]$$

(2) Beta$(n, 1)$ 分布，密度为

$$p(x) = nx^{n-1}, x \in [0, 1]$$

(3) 密度为

$$p(x) = \frac{2}{\pi\sqrt{1-x^2}}, x \in [0, 1]$$

(4) 柯西分布，密度为

$$p(x) = \frac{1}{\pi(1+x^2)}, x \in (-\infty, \infty)$$

(5) 密度为

$$p(x) = \cos(x), x \in [0, \frac{\pi}{2}]$$

(6) 威布尔分布，密度为

$$p(x) = \frac{\alpha}{\eta} x^{\alpha-1} e^{-\frac{x^\alpha}{\eta}},\ x > 0\ (\alpha > 0, \eta > 0)$$

习题 9

设 X 为标准指数分布 Exp(1) 随机变量，模拟在 $X < 0.05$ 条件下 X 的分布，密度为

$$p(x) = \frac{e^{-x}}{1 - e^{-0.05}},\ 0 < x < 0.05$$

生成 1000 个这样的随机数，并用它们估计 $E(X|X<0.05)$，推导 $E(X|X<0.05)$ 的精确值并与模拟结果比较。

习题 10

设随机变量 X 分布为

$$P(X=k)=\frac{e^{-\lambda}\frac{\lambda^k}{k!}}{\sum_{i=0}^{m}e^{-\lambda}\frac{\lambda^i}{i!}},\ k=0,1,...,m$$

给出模拟此分布的两种方法。

习题 11

设 $X\sim \mathrm{B}(n,p)$, k 为满足 $0\le k\le n$ 的给定的整数，随机变量 Y 的分布函数为 $P(Y\le y)=P(X\le y|X\ge k)$。记 $\alpha=P(X\ge k)$。分别用逆变换法和舍选法生成 Y 的随机数。当 α 取值大的时候还是取值小的时候舍选法不可取？

习题 12

给出生成如下密度

$$p(x)=xe^{-x},\ x>0$$

的随机数的两种方法并比较其效率。

习题 13

设随机变量分别以概率 0.06, 0.06, 0.06, 0.06, 0.06, 0.15, 0.13, 0.14, 0.15, 0.13 取值 $1,2,...,10$，应用复合方法给出产生此分布随机数的算法。

习题 14

设随机变量 X 的分布为

$$P(X=k)=\frac{1}{2^{k+1}}+\frac{2^{k-1}}{3^k},\ k=1,2,...$$

给出模拟此随机变量的算法。

习题 15

设随机变量 X 的分布密度为

$$p(x) = \frac{1}{2}e^{-|x|},\ -\infty < x < \infty$$

称 X 服从双指数分布或 Laplace 分布。分别用逆变换法和复合法生成 X 的随机数。

习题 16

给出生成密度函数为

$$p(x) = \frac{1}{0.000336}x(1-x)^3, 0.8 < x < 1$$

的随机数的有效算法。

习题 17

设 X 密度为

$$p_R(x) = \frac{2(x-a)}{(b-a)^2},\ x \in (a,b)$$

此密度单调上升，是以 $[a,b]$ 为底的三角形，叫做 $\mathrm{RT}(a,b)$ 分布；若 X 的密度为

$$p_L(x) = \frac{2(b-x)}{(b-a)^2},\ x \in (a,b)$$

此密度单调下降，也是以 $[a,b]$ 为底的三角形，叫做 $\mathrm{LT}(a,b)$ 分布。试证明：

(1) 若 $X \sim \mathrm{RT}(0,1)$，则 $Y = a + (b-a)X \sim \mathrm{RT}(a,b)$; 若 $X \sim \mathrm{LT}(0,1)$，则 $Y = a + (b-a)X \sim \mathrm{LT}(a,b)$。
(2) $X \sim \mathrm{RT}(0,1)$ 当且仅当 $1 - X \sim \mathrm{LT}(0,1)$。
(3) 若 U_1, U_2 独立且分别服从 U(0,1) 分布，则 $X = \max(U_1, U_2) \sim \mathrm{RT}(0,1)$, $Y = \min(U_1, U_2) \sim \mathrm{LT}(0,1)$。

习题 18

设 $\alpha \sim \mathrm{U}(0, 2\pi)$, $R \sim \mathrm{Exp}(\frac{1}{2})$ 与 α 独立，令

$$\begin{cases} X = \sqrt{R}\cos\alpha \\ Y = \sqrt{R}\sin\alpha \end{cases}$$

证明 X, Y 相互独立且都服从 N(0,1) 分布。

习题 19

设 $U_1, U_2, \ldots, U_k, V_1, V_2, \ldots, V_{k-1}$ 独立同 U(0,1) 分布。令 $T = -\log(U_1 U_2 \ldots U_k)$, 设 $V_1, V_2, \ldots, V_{k-1}$ 从小到大排列结果为 $V_{(1)} \le V_{(2)} \le \ldots V_{(k-1)}$, 记 $V_{(0)} = 0$, $V_{(k)} = 1$, 令

$$X_i = T(V_{(i)} - V_{(i-1)}),\ i = 1, 2, \ldots, k,$$

证明 $X_1, X_2, \ldots, X_k$ 独立同标准指数分布 Exp(1)。

习题 20

设随机向量 (X, Y) 服从单位圆 $C = \{(x, y) : x^2 + y^2 \le 1\}$ 上的均匀分布，$R^2 = X^2 + Y^2$，θ 为 (X, Y) 的极角，则 R^2 与 θ 独立，$R^2 \sim \mathrm{U}(0,1)$, $\theta \sim \mathrm{U}(0, 2\pi)$。

习题 21

设随机变量 X 分布函数为 $G(x)$，密度函数为 $g(x)$。对 $a < b$，令

$$F(x) = \frac{G(x) - G(a)}{G(b) - G(a)}, a \le x \le b$$

(1) $F(x)$ 是一个分布函数，其对应的分布是 X 在什么条件下的条件分布？
(2) 证明可以用如下方法生成 $F(x)$ 的随机数: 反复生成 $X \sim G(x)$，直到 $X \in [a, b]$，输出 X 的值为 $F(x)$ 的随机数。

习题 22

设随机变量 X 的概率分布为 $p_k = P(X = k), k = 1, 2, ..., \sum_{k=1}^{\infty} p_k = 1$。记

$$\lambda_n = P(X = n|X > n - 1) = \frac{p_n}{1 - \sum_{k=1}^{n-1} p_k},\ n = 1, 2, ...$$

(1) 证明 $p_1 = \lambda_1$, $p_n = (1 - \lambda_1)(1 - \lambda_2) \cdots (1 - \lambda_{n-1})\lambda_n$ $(n = 2, 3, ...)$。如果把 X 看成某种产品的寿命，则 λ_n 表示此产品在其寿命大于 $n - 1$ 的条件下，寿命为 n 的概率。$\{\lambda_n, n \geq 1\}$ 叫做**离散机会比**。一个生成 X 的随机数的方法是连续生成均匀随机数直到第 n 个小于 λ_n，叫做离散机会比方法，算法如下:

$k \leftarrow 0$
until$(U < \lambda_k)\{$
    Generate $U \sim \mathrm{U}(0, 1)$
    $k \leftarrow k + 1$
$\}$
$X \leftarrow k$

(2) 证明上述算法产生的随机数符合 X 的分布。

(3) 假设 X 是一个参数为 p 的几何分布随机变量，求 $\lambda_n, n = 1, 2, ...$。说明此时上述算法是用来做什么的，它的有效性为什么是明显的？

习题 23

假设 X 为取值于 $\{1, 2, ...\}$ 的随机变量，$\lambda_n, n = 1, 2, ...$ 为其离散机会比，满足 $0 \leq \lambda_n \leq \lambda, n = 1, 2, ...$。用如下算法产生 X 的随机数:

$S \leftarrow 0$

until$(U_2 \leq \lambda_S/\lambda)\{$

Generate $U_1 \sim \mathrm{U}(0,1)$, let $Y \leftarrow \mathrm{ceil}\left(\dfrac{\log(U_1)}{\log(1-\lambda)}\right)$

$S \leftarrow S + Y$

Generate $U_2 \sim \mathrm{U}(0,1)$

$\}$

$X \leftarrow S$

(1) 算法中的 Y 服从什么分布？

(2) 证明这样得到的 X 是离散机会比为 $\{\lambda_n\}$ 的离散型随机变量。

Chapter 7

随机向量和随机过程的随机数

7.1 条件分布法

产生随机向量的一种方法是条件分布法。设 $X=(X_1,X_2,\dots,X_r)$ 的分布密度或分布概率 $p(x_1,x_2,\dots,x_r)$ 可以分解为

$$p(x_1,x_2,\dots,x_r)=p(x_1)p(x_2|x_1)p(x_3|x_1,x_2)\cdots p(x_r|x_1,x_2,\dots,x_{r-1})$$

则可以先生成 X_1，由已知的 X_1 的值从条件分布 $p(x_2|x_1)$ 产生 X_2，再从已知的 X_1,X_2 的值从条件分布 $p(x_3|x_1,x_2)$ 产生 X_3，如此重复直到产生 X_r。

7.1.1 多项分布随机数

进行了 n 次独立重复试验 $Y_1,Y_2,\dots,Y_n$，每次的试验结果 Y_i 在 $1,2,\dots,r$ 中取值，$P(Y_i=j)=p_j,j=1,2,\dots,r;i=1,2,\dots,n$。令 X_j 为这 n 次试验中结果 j 的个数 $(j=1,2,\dots,r)$，称 $X=(X_1,X_2,\dots,X_r)$ 服从多项分布，其联合概率函数为

$$P(X_j=x_j,j=1,2,\dots,r)=\frac{n!}{x_1!x_2!\dots x_r!}p_1^{x_1}p_2^{x_2}\dots p_r^{x_r}.$$

(其中 $\sum_{j=1}^{r}x_j=n$)。

要生成 X 的随机数，考虑不同结果数 r 与试验次数 n 的比较。当 r 和 n 相比很大时，每个结果的出现次数都不多，而且许多结果可能根本不出现，这样，可

以模拟产生 $Y_i, i = 1, 2, \dots, n$ 然后从 $\{Y_i\}$ 中计数得到 $(X_1, X_2, \dots, X_r)$。当 r 与 n 相比较小时，每个结果出现次数都比较多，可以使用条件分布逐个地产生 $X_1, X_2, \dots, X_r$。

X_1 表示 n 次重复试验中结果 1 的出现次数，以结果 1 作为成功，其它结果作为失败，显然 $X_1 \sim \mathrm{B}(n, p_1)$。产生 X_1 的值 x_1 后，n 次试验剩余的 $n - x_1$ 次试验结果只有 $2, 3, \dots, r$ 可取，于是在 $X_1 = x_1$ 条件下结果 2 的出现概率是

$$P(Y_i = 2 | Y_i \neq 1) = \frac{P(Y_i = 2)}{\sum_{k=2}^{r} P(Y_i = k)} = \frac{p_2}{1 - p_1}$$

于是剩下的 $n - x_1$ 次试验中结果 2 的出现次数服从 $\mathrm{B}(n - x_1, \frac{p_2}{1-p_1})$ 分布，从这个条件分布产生 $X_2 = x_2$。类似地，剩下的 $n - x_1 - x_2$ 次试验中结果 3 的出现次数服从 $\mathbf{B}(n - x_1 - x_2, \frac{p_3}{1-p_1-p_2})$ 分布，从这个条件分布中抽取 $X_3 = x_3$，如此重复可以产生多项分布 X 的随机数 $(x_1, x_2, \dots, x_r)$。

Julia 实现：

```
## 输入
##   - n: 需要输出的随机数个数
##   - m: 模型中的独立重复试验次数
##   - prob: 每次试验，各个结果出现的概率
## 输出: n 行 r 列矩阵，r 是不同结果个数
using Random, Distributions
function randmultnom(n, m, prob)
    local r, pc, xmat, m1
    r = length(prob)
    pc = [0.0; cumsum(prob)]
    xmat = zeros(Int, n, r) # 每一行是一个抽样向量
    for i=1:n
        m1 = m
        for k=1:(r-1)
            xmat[i, k] = rand(Binomial(m1, prob[k]/(1 - pc[k])))
            m1 -= xmat[i, k]
        end # for k
        xmat[i,r] = m1
    end # for i
```

```
    return xmat
end
```

测试：

```
using StatsBase
Random.seed!(101)
prob = [0.1, 0.3, 0.6]
x = randmultnom(100_000, 5, prob)
# (1,2,2) 的概率：
p122 = 30*prod(prob .^ [1,2,2]); p122
## 0.0972
mean((x[:,1] .== 1) .& (x[:,2] .== 2) .& (x[:,3] .== 2) )
## 0.09767
# (1,1,3) 的概率：
p113 = 20*prod(prob .^ [1,1,3]); p113
# 0.1296
mean((x[:,1] .== 1) .& (x[:,2] .== 1) .& (x[:,3] .== 3) )
# 0.12974
```

7.2 多元正态分布模拟

设随机向量 $X = (X_1, X_2, \dots, X_p)^T$ 服从多元正态分布 $\mathrm{N}(\mu, \Sigma)$，联合密度函数为

$$p(x) = (2\pi)^{-\frac{p}{2}} |\Sigma|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right\},\ x \in R^p$$

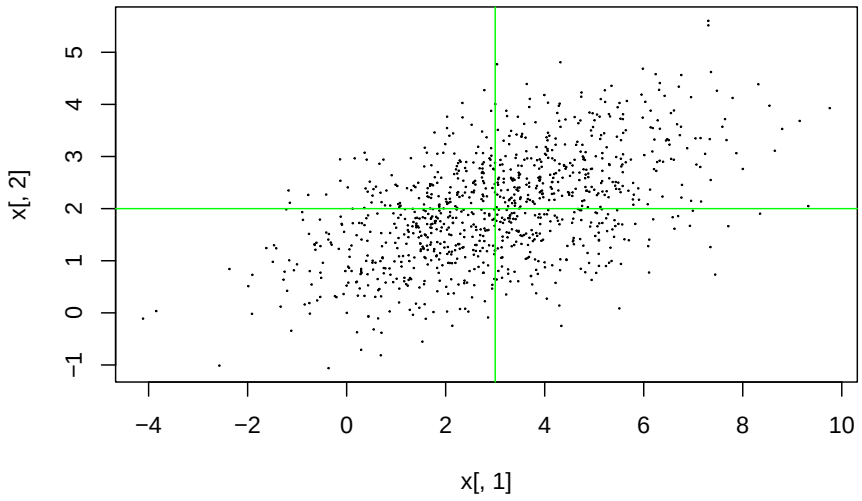
正定矩阵 Σ 有 Cholesky 分解 $\Sigma = CC^T$，其中 C 为下三角矩阵。设 $Z = (Z_1, Z_2, \dots, Z_p)^T$ 服从 p 元标准正态分布 $\mathrm{N}(0, I)$(I 表示单位阵)，则 $X = \mu + CZ$ 服从 $\mathrm{N}(\mu, \Sigma)$ 分布。

随机数发生器的程序：

```
## 生成多元正态分布随机数，
## n 是需要的个数，
## mu 是均值向量，
## Sigma 是协方差阵，需要是对称正定阵。
rng.mnorm <- function(n, mu, Sigma){
  m <- length(mu)
  M <- chol(Sigma) # Sigma = M' M
  y <- matrix(rnorm(n*m), n, m) %*% M
  for(j in seq(along=mu)){
    y[,j] <- y[,j] + mu[j]
  }
  y
}
```

测试：

```
x <- rng.mnorm(1000, c(3,2), rbind(c(4, 1), c(1, 1)))
plot(x[,1], x[,2], type="p", cex=0.1)
abline(v=3, h=2, col="green")
```




```
var(x)
##           [,1]     [,2]
## [1,] 4.151643 1.125089
## [2,] 1.125089 1.070800
```

Julia 版本：

```
using LinearAlgebra
function randmnorm(n,  , Σ)
    local m, L, y
    m = length( )
    L = cholesky(Σ).L
    y = randn(n, m) * L
    y = y .+  '

    return y
end
```

测试程序：

```
using Random
using CairoMakie, AlgebraOfGraphics
CairoMakie.activate!()
x = randmnorm(1000, [3, 2], [4 1; 1 1])
plt = data((; x=x[:,1], y=x[:,2])) * mapping(:x, :y) * visual(Scatter)
draw(plt)
cov(x)
```

结果略。

7.3 泊松过程模拟

参数为 λ 的泊松过程 $N(t), t \geq 0$ 是取整数值的随机过程，$N(t)$ 表示到时刻 t 为止到来的事件个数，两次事件到来的时间间隔相互独立，都服从指数分布 $\text{Exp}(\lambda)$。

所以，如果要生成泊松过程前 n 个事件到来的时间，只要生成 n 个独立的 $\text{Exp}(\lambda)$ 随机数 $X_1, X_2, \ldots, X_n$，则 $S_k = \sum_{i=1}^{k} X_i, k = 1, 2, \ldots, n$ 为各个事件到来的时间。

如果要生成泊松过程在时刻 T 之前的状态，只要知道发生在 T 之前的所有事件到来时间就可以了。

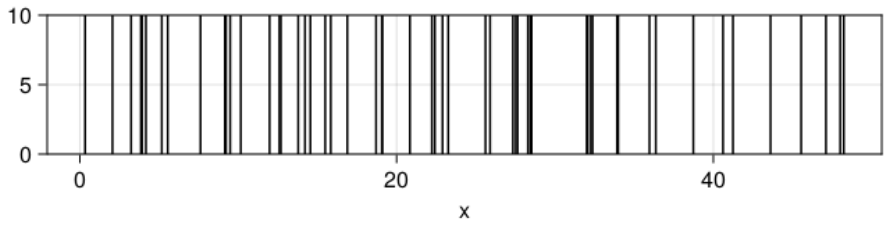
Julia 程序：

```
# 泊松过程模拟
function randpproc(T=100.0,  =1.0)
    local xs, S, nev, X
    xs = Float64[]
    S = 0.0 # 当前时刻
    nev = 0 # 已发生事件个数
    while true
        X = -log(rand()) /
        S += X
        if S <= T
```

```
            nev += 1
            push!(xs, S)
        else
            break
        end
    end # while true
    return xs
end
```

测试：

```
using Random, Distributions
using CairoMakie, AlgebraOfGraphics
CairoMakie.activate!()
Random.seed!(101)
x = randpproc(50, 1)
plt = data((; x=x)) * mapping(:x) * visual(VLines)
draw(plt)
## 估计速率
mean(x[2:end] .- x[begin:end-1])
## 0.8870887289991723
```



生成泊松过程在时刻 T 之前的状态的另外一种方法是先生成 $N(T) \sim \text{Poisson}(\lambda)$，设 $N(T)$ 的值为 n，再生成 n 个独立的 U(0,1) 随机变量 $U_1, U_2, \ldots, U_n$，从小到大排序为 $U_{(1)} \leq U_{(2)} \leq \ldots U_{(n)}$，则 $(TU_{(1)}, TU_{(2)}, \ldots, TU_{(n)})$ 为时刻 T 之前的所有事件到来时间。

为了生成强度函数为 $\lambda(t), t \geq 0$ 的非齐次泊松过程到时刻 T 为止的状态，如

果 $\lambda(t)$ 满足

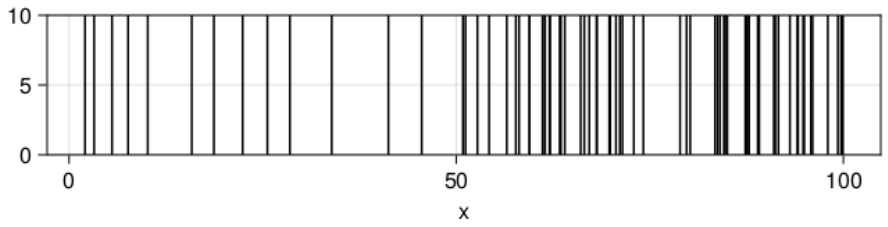
$$\lambda(t) \le M,\ \forall t \ge 0,$$

则可以按照生成参数为 M 的齐次泊松过程的方法去生成各个事件到来时刻，但是以 $\lambda(t)/M$ 的概率实际记录各个时刻。Julia 程序如下:

```
function randnhpproc(T,   = x -> 1.0, M=1.0)
    local xs
    xs = randpproc(T, M)
    xs = [t for t in xs if rand() <  (t) / M]
    return xs
end
```

测试。假设前 50 分钟速率为 0.2, 后 50 分钟速率为 1。

```
fl(t) = t <= 50 ? 0.2 : 1.0
Random.seed!(101)
x = randnhpproc(100.0, fl, 1.0)
plt = data((; x=x)) * mapping(:x) * visual(VLines)
dr = draw(plt; axis = (; width=600, height=100))
```



7.4 布朗运动模拟

时间区间 $[0,T]$ 上的一维的标准布朗运动为随机过程 $\{W(t), 0 \le t \le T\}$，满足:

- $W(0) = 0$;

- 以概率 1 地，$W(t)$ 作为 t 的函数，（每条轨道）在 $[0,T]$ 连续；
- 独立增量；
- $W(t)-W(s)\sim \mathrm{N}(0,t-s)$, 对任意 $0\le s<t\le T$。

$W(t)$ 分布为

$$W(t)\sim \mathrm{N}(0,t),\ 0<t\le T.$$

对标准布朗运动 $W(t)$，令

$$X(t)=\mu t+\sigma W(t), 0\le t\le T,$$

称 $X(t)$ 为有漂移 μ 和扩散系数 σ^2 的布朗运动，简记为 $\mathrm{BM}(\mu,\sigma^2)$。其中 μ，σ 为常数，$\sigma>0$。$X(t)$ 仍是高斯过程，从 0 出发，有独立增量，轨道连续，边缘分布为

$$X(t)\sim \mathrm{N}(\mu t,\sigma^2 t).$$

也可以构造 $X(0)=x$ 的解，只要给每个 $X(t)$ 加 x 即可。

可以定义漂移和扩散系数时变的布朗运动，存在时变的函数 $\mu(t)$ 和 $\sigma(t)$，使得增量 $X(t)-X(s)(0\le s<t\le T)$ 服从正态分布，期望为

$$E[X(t)-X(s)]=\int_s^t \mu(u)\,du,$$

方差为

$$\mathrm{Var}[X(t)-X(s)]=\int_s^t \sigma^2(u)\,du.$$

设 $0<t_1<t_2<\cdots<t_n\le T$，要生成 $(W(t_1),\dots,W(t_n))$ 或者 $(X(t_1),\dots,X(t_n))$ 的抽样，最简单的方法是利用布朗运动的独立增量性和正态性。设 $Z_1,\dots,Z_n$ 是独立同标准正态分布的随机变量（随机数），$t_0=0$, $W(0)=0$, $X(0)=0$。

令

$$W(t_{i+1})=W(t_i)+\sqrt{t_{i+1}-t_i}Z_{i+1},\ i=0,1,\dots,n-1. \tag{7.1}$$

对 $X(t)\sim \mathrm{BM}(\mu,\sigma^2)$，令

$$X(t_{i+1})=X(t_i)+\mu\cdot(t_{i+1}-t_i)+\sigma\sqrt{t_{i+1}-t_i}Z_{i+1},\ i=0,1,\dots,n-1. \tag{7.2}$$

对于漂移和扩散系数时变的 $X(t)$，令

$$X(t_{i+1}) = X(t_i) + \int_{t_i}^{t_{i+1}} \mu(u)\,du + \left(\int_{t_i}^{t_{i+1}} \sigma^2(u)\,du\right)^{1/2} Z_{i+1},\ i = 0, 1, \dots, n-1. \tag{7.3}$$

只要输入的 $Z_1, \dots, Z_n$ 的联合分布正确，则输出的 $(W(t_1), \dots, W(t_n))$ 和 $(X(t_1), \dots, X(t_n))$ 的联合分布也是正确的，没有离散化误差。

这只输出了 $t_1, t_2, \dots, t_n$ 这有限个点上的轨道的值，中间的值未知，如果用通常的线性插值等方法近似，就会产生离散化误差，分布也不准确。

在漂移和扩散系数时变时，如果将(7.3)中的积分替换成如下的欧拉 (Euler) 近似：

$$X(t_{i+1}) = X(t_i) + \mu(t_i)\cdot(t_{i+1} - t_i) + \sigma(t_i)\sqrt{t_{i+1} - t_i}\,Z_{i+1},\ i = 0, 1, \dots, n-1,$$

则输出的 $(X(t_1), \dots, X(t_n))$ 的联合分布也是近似的，带有离散化误差 (discretization error)，而且这种误差是累积的，时间越往后误差越大。

7.5 平稳时间序列模拟 (*)

平稳时间序列中的 ARMA 模型可以递推生成。

对 AR(p) 模型

$$X_t = a_1 X_{t-1} + a_2 X_{t-2} + \cdots + a_p X_{t-p} + \varepsilon_t,\ t \in \mathbb{Z}$$

($\mathbb{Z}$ 表示所有整数的集合), 其中 $\{\varepsilon_t\}$ 为白噪声 WN(0,σ^2), 即方差都为 σ^2、彼此不相关的随机变量序列，$\{\varepsilon_t\}$ 可以用 N(0,σ^2) 分布的独立序列来模拟，也可以用其它有二阶矩的分布。因为 AR(p) 模型具有所谓 “稳定性”，所以我们从任意初值出发按照递推 N_0 步 (N_0 是一个较大正整数) 后，再继续递推 N 步后得到的 N 个 X_t 就可以作为上述 AR(p) 模型的一次实现。根据模型稳定性的好坏，N_0 可取为 $50 \sim 1000$ 之间。

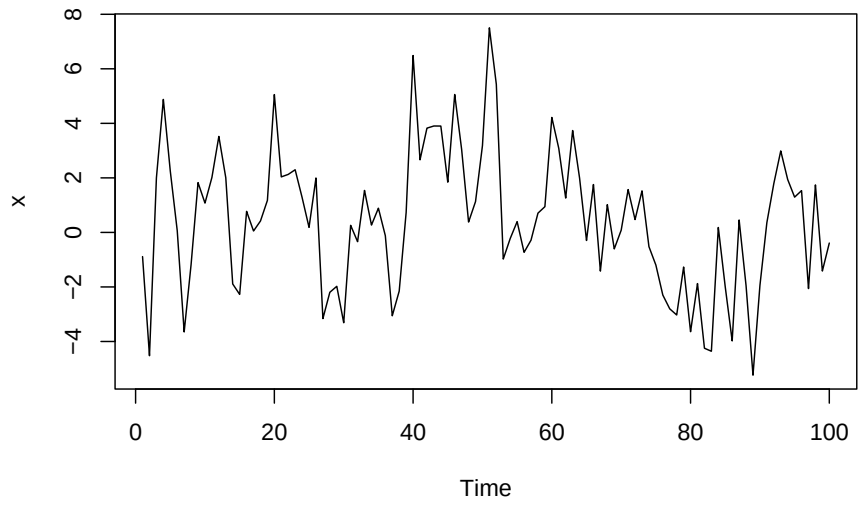
下面的 R 程序直接使用递推，比较简单。

```
## 输入:
##   - n: 需要输出的序列长度
##   - a: 自回归系数 $(a_1, a_2, \dots, a_p)$, 是递推方程右侧的系数
##   - sigma: 新息项的标准差
##   - n0: 需要舍弃的初始部分
ar.gen.v1 <- function(n, a, sigma=1.0, n0=1000){
  n2 <- n0 + n
  p <- length(a)
  arev <- rev(a)
  x2 <- numeric(n2)
  eps <- rnorm(n2, 0, sigma)
  for(tt in seq(p+1, n2)){
    x2[tt] <- eps[tt] + sum(x2[(tt-p):(tt-1)] * arev)
  }
  x <- x2[(n0+1):n2]
  x <- ts(x)
  attr(x, "model") <- "AR"
  attr(x, "a") <- a
  attr(x, "sigma") <- sigma

  x
}
```

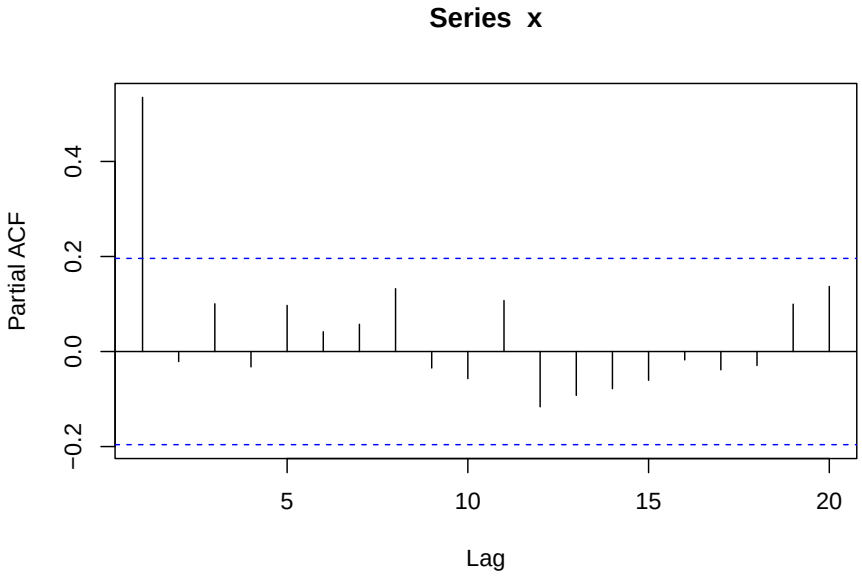
测试，用如下 AR(1) 模型：

$$X_t = 0.5 * X_{t-1} + \varepsilon_t,\ \varepsilon_t \sim \mathrm{N}(0, 2^2)$$

```
x <- ar.gen.v1(100, c(0.5), sigma=2)
plot(x)
```




```
pacf(x)
```

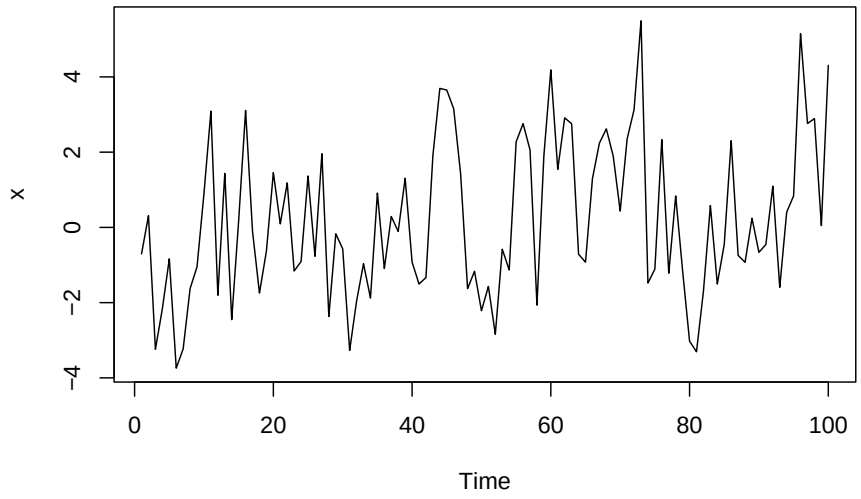


因为 R 的循环很慢，所以改用 `stats::filter()` 函数写成如下版本：

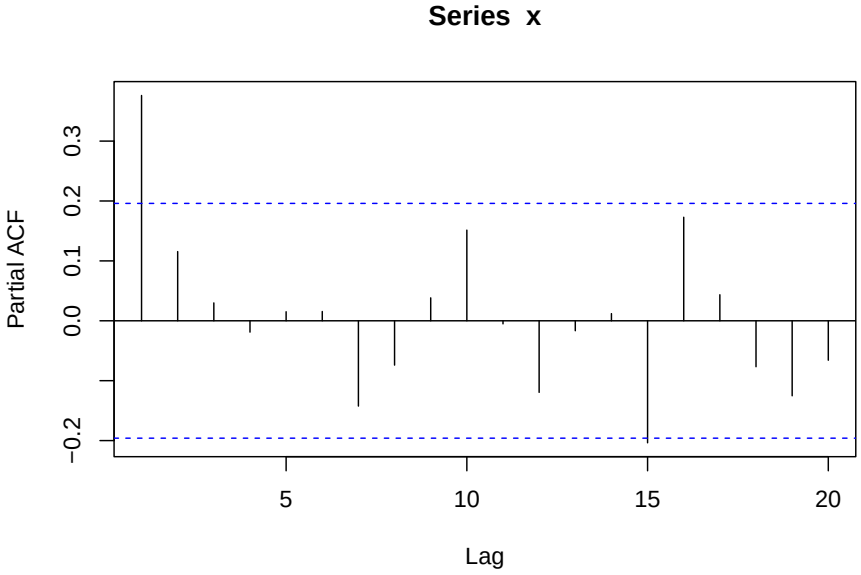
```
ar.gen <- function(n, a, sigma=1.0,
                   n0=1000,
                   x0=numeric(length(a))){
  n2 <- n0 + n
  p <- length(a)
  eps <- rnorm(n2, 0, sigma)
  x2 <- stats::filter(
    eps, a, method="recursive", sides=1, init=x0)
  x <- x2[(n0+1):n2]
  x <- ts(x)
  attr(x, "model") <- "AR"
  attr(x, "a") <- a
  attr(x, "sigma") <- sigma

  x
}
```

测试：

```
x <- ar.gen(100, c(0.5), sigma=2)
plot(x)
```


x
4
2
0
−2
−4
0
20
40
60
80
100
Time

```
pacf(x)
```


Series x
Partial ACF
0.3
0.2
0.1
0.0
−0.2
5
10
15
20
Lag

对于 MA(q) 模型

$$X_t = \varepsilon_t + b_1\varepsilon_{t-1} + \cdots + b_q\varepsilon_{t-q},\ t \in \mathbb{Z}$$

生成 $\varepsilon_{1-q}, \varepsilon_{2-q}, \ldots, \varepsilon_0, \varepsilon_1, \ldots, \varepsilon_N$ 后可以直接对 $t = 1, 2, \ldots, N$ 按公式计算得到 $\{X_t, t = 1, 2, \ldots, N\}$。

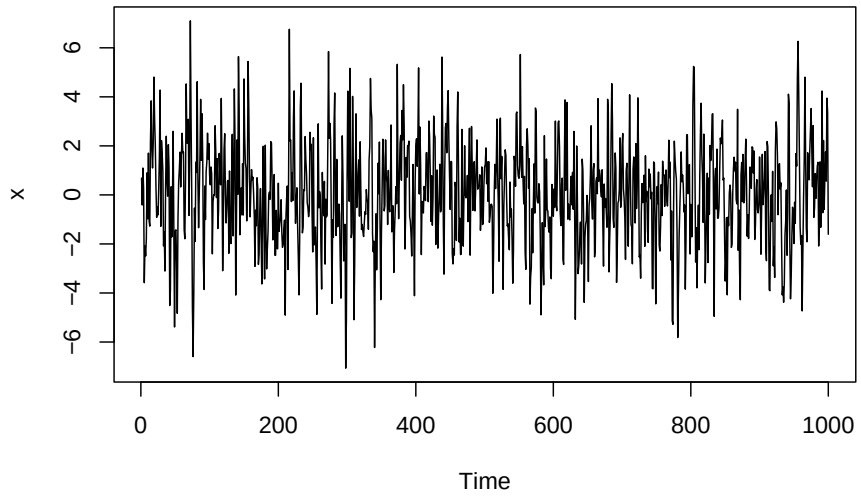
模拟 MA(q) 的 R 程序：

```
ma.gen <- function(n, a, sigma=1.0){
  q <- length(a)
  n2 <- n+q
  eps <- rnorm(n2, 0, sigma)
  x2 <- stats::filter(
    eps, c(1,a), method="convolution", sides=1)
  x <- x2[(q+1):n2]
  x <- ts(x)
  attr(x, "model") <- "MA"
  attr(x, "b") <- a
  attr(x, "sigma") <- sigma
  x
}
```

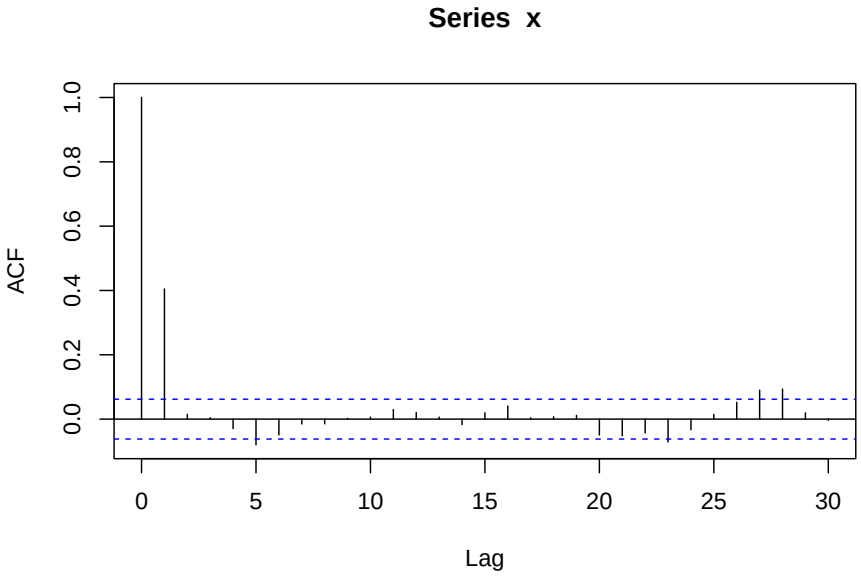
测试，模型为

$$X_t = \varepsilon_t + 0.5\varepsilon_{t-1},\ \varepsilon_t \sim \mathrm{N}(0, 2^2)$$

```
x <- ma.gen(1000, c(0.5), sigma=2)
plot(x)
```


x
6
4
2
0
−2
−4
−6
0
200
400
600
800
1000
Time

```
acf(x)
```


Series x
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
Lag

对于 ARMA(p,q) 模型

$$X_t = a_1 X_{t-1} + a_2 X_{t-2} + \cdots + a_p X_{t-p} + \varepsilon_t + b_1 \varepsilon_{t-1} + \cdots + b_q \varepsilon_{t-q},\ t \in \mathbb{Z}$$

也可以从初值零出发递推生成 $N_0 + N$ 个然后只取最后的 N 个作为 ARMA(p,q) 模型的一次实现。

模拟程序如下:

```
arma.gen <- function(
  n, a, b, sigma=1.0,
  n0=1000, x0=numeric(length(a))){
  n2 <- n0 + n
  p <- length(a)
  ## first generate n0+n MA(q) series
  eps <- ma.gen(n2, b, sigma=sigma)
  x2 <- stats::filter(
    eps, a, method="recursive", sides=1, init=x0)
  x <- x2[(n0+1):n2]
  x <- ts(x)
  attr(x, "model") <- "ARMA"
  attr(x, "a") <- a
  attr(x, "b") <- b
  attr(x, "sigma") <- sigma
  x
}
```

R 函数 `filter` 可以进行递推和卷积计算。R 函数 `arima.sim` 可以进行 ARMA 模型和 ARIMA 模型模拟。

习题

习题 1

设坛子中有 n 个不同颜色的球，第 i 中颜色的球有 n_i 个 $(i = 1, 2, \ldots, r)$。从坛子中随机无放回地抽取 m 个球，设随机变量 X_i 表示取出的第 i 种颜色球的

个数，设计高效的算法模拟 $(X_1, X_2, \ldots, X_r)$ 的值。

习题 2

利用稀松法编写模拟到时刻 $T = 10$ 为止的强度为

$$\lambda(t) = 3 + \frac{4}{t+1}$$

的非齐次泊松过程的 R 程序; 设法改进这个算法的效率。

习题 3

用 R 的 `filter()` 函数编写 AR(p) 模型的模拟程序。

习题 4

用 R 的 `filter()` 函数编写 MA(q) 模型的模拟程序。

Chapter 8

R 中的随机数函数 (*)

8.1 R 语言中与分布有关的函数

R 语言作为一个面向统计计算、数据分析、作图的语言，提供了丰富的与概率分布有关的函数。对一个概率分布 xxx，函数 `dxxx()` 计算分布密度函数或者概率质量函数，函数 `pxxx()` 计算分布函数，函数 `qxxx()` 计算分位数函数，函数 `rxxx()` 生成该分布的若干个随机数。

在 R 命令行用

```
?Distributions
```

查看 stats 包给出的各种与分布有关的函数。

下面以正态分布为例介绍这些函数的用法。设正态分布密度为 $f(x)$, 分布函数为 $F(x)$，分位数函数为 $F^{-1}(x)$。密度函数 $f(x)$ 用法为：

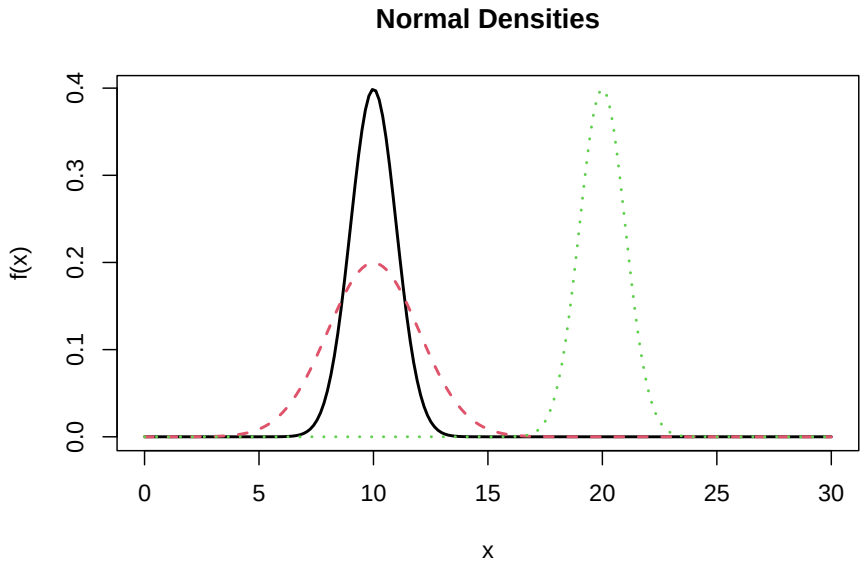
```
dnorm(x, mean=0, sd=1, log=FALSE)
```

计算标量或者向量 `x` 处的正态分布密度函数值，可以用参数 `mean` 指定分布均值参数，用参数 `sd` 指定分布的标准差参数，如果加选项 `log=FALSE` 可以返回函数值的自然对数值，这在计算对数似然函数时有用。参数缺省为 `mu=0`, `sd=1`。调用时自变量 `x` 经常是向量，结果返回各个元素处的密度函数值；参数通常是

标量，但是也允许取为向量，这时一般与自变量 x 的向量一一对应，但是也可以长度不相同，短的循环使用。

例如，计算三个不同的正态密度，并作图：

```
muv <- c(10, 10, 20)
sdv <- c(1, 2, 1)
np <- 200
x <- seq(0, 30, length.out=np)
ym <- matrix(0, np, 3)
for(j in 1:3){
  ym[, j] <- dnorm(x, muv[j], sdv[j])
}
matplot(x, ym, type="l", lwd=2, xlab="x", ylab="f(x)", main="Normal Densities")
```



正态分布函数（$F(x)$）用法为：

```
pnorm(q, mean=0, sd=1, lower.tail=TRUE, log.p=FALSE)
```

其中 q 是要计算分布函数的自变量位置，可输入向量，mean 和 sd 是参数，参

数也允许取为向量。加选项 `lower.tail=FALSE` 可以计算 $1-F(x)$，加选项 `log.p=TRUE` 可以输出结果的自然对数值。

正态分位数函数用法为：

```
qnorm(p, mean=0, sd=1, lower.tail=TRUE, log.p=FALSE)
```

其中 `p` 是 $(0,1)$ 内取值的标量或者元素在 $(0,1)$ 内取值的向量，`mean` 和 `sd` 是参数。加选项 `lower.tail=FALSE` 则求的是 $F^{-1}(1-p)$。加选项 `log.p=TRUE`，则求 $F^{-1}(e^p)$。

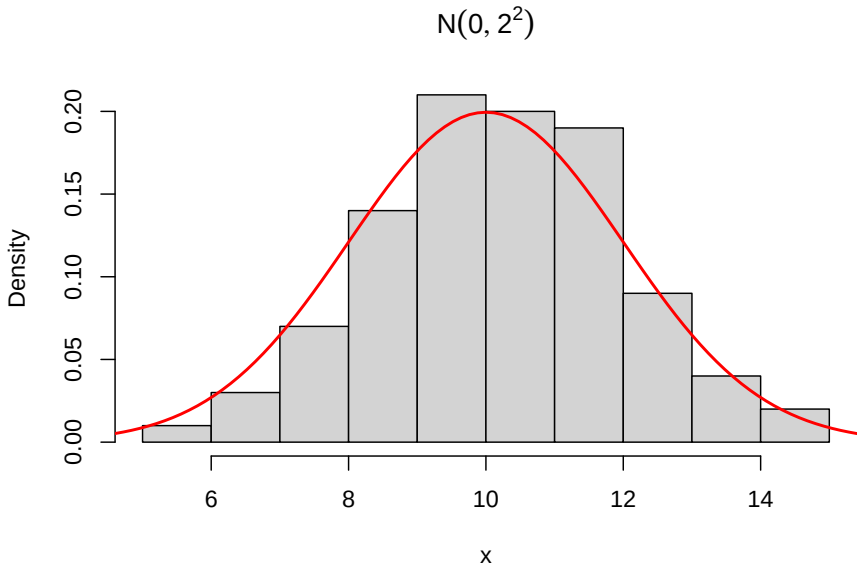
正态随机数函数用法为:

```
rnorm(n, mean=0, sd=1)
```

其中 `n` 是需要输出的随机数个数，`mean` 和 `sd` 是分布参数。这个函数中没有指定种子的选项，要指定种子和选择不同的基础随机数发生器需要使用特殊的函数来改变。用 `set.seed(seed)` 指定一个种子的编号，其中 `seed` 是一个整数值。

下面的例子生成了 100 个 $\mathrm{N}(10,2^2)$ 的随机数，将其舍入到小数点后两位精度（实际数据中并不存在真正的无限位数的实数），作直方图，并画理论的密度曲线：

```
set.seed(1)
x <- rnorm(100, 10, 2)
hist(x, freq=FALSE, main=expression(N(0,2^2)))
curve(dnorm(x, 10, 2), 4, 16, lwd=2, col="red", add=TRUE)
```



8.2 分布列表

R 的 stats 包是默认安装的基础包，其中已经实现多种分布的有关函数，函数类型见上面以正态为例的说明。分布包括：

- beta: 贝塔分布
- binom: 二项分布，包括两点分布
- cauchy: 柯西分布
- chisq: 卡方分布
- exp: 指数分布
- f: F 分布
- gamma: 伽马分布
- geom: 几何分布
- hyper: 超几何分布
- lnorm: 对数正态分布
- multinom: 多项分布
- norm: 正态分布

- pois: 泊松分布
- t: t 分布
- unif: 均匀分布
- weibull: 威布尔分布

对每个上面的函数名 xxx，都有对应的 `dxxx()`, `pxxx()`, `qxxx()`, `rxxx()` 函数。

其它的一些 R 扩展包中提供了更多的分布，参见 R 的 CRAN 网站的 Task View 中 Distributions 部分。

8.3 R 随机数发生器

我们知道，非均匀的随机数发生器实际是通过均匀随机数发生器构造出来的。R 中内建了多种不同均匀随机数发生器算法，在 R 命令行用如下命令查看关于 R 的随机数发生器的说明：

```
?RNG
```

在调用 R 的产生随机数的函数如 `runif()`, `rnorm()` 时，随机数发生器的当前状态是随之更新的，这样，连续两次 `runif(5)` 的调用结果是不同的，如：

```
set.seed(1)
round(runif(5), 2)
```

```
## [1] 0.27 0.37 0.57 0.91 0.20
```

```
round(runif(5), 2)
```

```
## [1] 0.90 0.94 0.66 0.63 0.06
```

当前选中的随机数发生器的当前状态保存在一个整数型向量类型的全局变量`.Random.seed` 中。直接保存这个变量的值，也可以用来保存自己的长时间

模拟的中间状态；将保存的值重新赋给这个变量就可以恢复随机数发生当时的状态，但是不要直接修改这个全局变量的内容。

Random.seed 需要自己保存和恢复，适用于较长时间的模拟保存中间状态，以免后续的模拟失败后还需要从头开始。对于从一开始进行的模拟，为了使得结果是可重复的，用 set.seed(seed) 设置一个随机数种子，即模拟的起始点，其中 seed 是一个整数，本身并不是真正的算法中的种子，可以看成是种子的一种编号。set.seed() 只保证结果可重复，选取不同种子时不能保证结果一定是不同的，但是不同种子给出的随机数序列一般是不同的，相同的情况及其少见。

set.seed() 还可以有一个选项 kind，取为一个表示不同发生器算法的字符串，当前的默认发生器算法是"Mersenne-Twister"。

随机数算法的选择有：

- "Wichmann-Hill"
- "Marsaglia-Multicarry"
- "Super-Duper"
- "Mersenne-Twister"
- "Knuth-TAOCP-2002"
- "Knuth-TAOCP"
- "L'Ecuyer-CMRG"
- "user-supplied" 由用户提供自己的随机数发生器

因为正态随机数常用，所以正态随机数发生器可以在 set.seed() 中用参数 normal.kind 指定，可取值有：

- "Kinderman-Ramage"
- "Ahrens-Dieter"
- "Box-Muller"
- "Inversion" (逆变换法，这是默认选择)

函数 RNGkind(kind=NULL, normal.kind=NULL) 用来选择发生器算法而不指定具体种子。调用 RNGkind() 可以返回当前使用的随机数发生器和正态分布随机数算法的名称。

8.4 R 中使用的若干个随机数发生器的算法说明

8.4.1 Wichmann-Hill 发生器

Wichmann 和 Hill(见 Wichmann and Hill [1982]) 设计了如下的线性组合发生器。

利用三个 16 位运算的素数模乘同余发生器:

$$\begin{aligned} U_n &= 171U_{n-1} \pmod{30269} \\ V_n &= 172V_{n-1} \pmod{30307} \\ W_n &= 170W_{n-1} \pmod{30323} \end{aligned}$$

作线性组合并求余:

$$R_n = (U_n/30269 + V_n/30307 + W_n/30323) \pmod 1$$

这个组合发生器的周期约有 7×10^{12} 长, 超过 $2^{32} \approx 4 \times 10^9$。

R 中使用 Wichmann-Hill 发生器时，全局变量.Random.seed 第一个元素是对应着当前随机数发生器的一个编码，后续的三个元素是当前的 U, V, W 的值，因为是余数，所以分别在 $1-30268, 1-30306, 1-30322$ 之间取值。

下面是 R 3.4.1 的源程序包中 RNG.c 文件中 Wichmann-Hill 发生器的部分 C 代码：

```
I1 = I1 * 171 % 30269;
I2 = I2 * 172 % 30307;
I3 = I3 * 170 % 30323;
value = I1 / 30269.0 + I2 / 30307.0 + I3 / 30323.0;
return fixup(value - (int) value);/* in [0,1) */
```

变量 I1, I2, I3 是 C 语言的 32 位有符号整数类型，分别代表一个素数模乘同余发生器，最后组合时取三个发生器变换到 $(0, 1)$ 中的和的小数部分，fixup() 函数用来确保不会有 0 和 1 值被返回。

8.4.2 Marsaglia-Multicarry

这是 George Marsaglia 提出的一种 “multiply-with-carry” 发生器，周期长达 $2^{60} \approx 1.2 \times 10^{18}$，种子由两个整数组成。

下面是 R 3.4.1 的源程序包中 RNG.c 文件中 Marsaglia-Multicarry 发生器的部分 C 代码：

```
/* 0177777(octal) == 65535(decimal)*/
I1= 36969*(I1 & 0177777) + (I1>>16);
I2= 18000*(I2 & 0177777) + (I2>>16);
return fixup(((I1 << 16)^(I2 & 0177777)) * i2_32m1); /* in [0,1) */
```

可以看出这也是一个混合发生器，`I1` 和 `I2` 是两个组件发生器，变量 `I1`, `I2` 是 C 语言的 32 位有符号整数类型，`I1 & 0177777` 取 `I1` 的后 16 位，`I1 >> 16` 取 `I1` 的前 16 位（看成无符号整数），迭代相当于 `I1` 的后 16 位二进制表示的无符号整数乘以 36969，加上 `I1` 的前 16 为二进制表示的无符号整数的值。`I2` 的迭代类似。最后将两个发生器组合起来，组合时仅取上面迭代后的 `I1` 的 32 位二进制中的后 16 位的取反，和迭代后的 `I2` 的后 16 位的取反，组成一个 32 位二进制整数，除以 $2^{32} - 1$。

8.4.3 Super-Duper

这是 Marsaglia 在 1970 年代提出的著名算法，周期约为 4.6×10^{18}，种子是两个整数，第二个数是奇数。R 中使用了 Reeds, J., Hubert, S. 和 Abrahams, M. 在 1982-1984 年的实现。

下面是 R 3.4.1 的源程序包中 RNG.c 文件中 Super-Duper 发生器的部分 C 代码：

```
/* This is Reeds et al (1984) implementation;
 * modified using __unsigned__  seeds instead of signed ones
 */
I1 ^= ((I1 >> 15) & 0377777); /* Tausworthe */
I1 ^= I1 << 17;
```

```
I2 *= 69069;        /* Congruential */
return fixup((I1^I2) * i2_32m1); /* in [0,1) */
```

也用了两个随机数发生器的混合，混合时，使用两个 32 位二进制整数 I1 和 I2 的按位异或运算混合在一起。关于 I1 的迭代，是将 I1 的 32 位二进制数右移 15 位，仅剩下原来高位的 17 位，再将这样得到的 32 个二进制整数与这个整数左移 17 位的结果按位异或，得到 I1 的值。I2 则是一个素数模乘同余发生器，每次乘以 69069，依靠 32 位整数的自动溢出求余。最后输出时将 I1 和 I2 两个 32 位二进制整数作按位异或，除以 $2^{32}-1$ 后输出。

8.4.4 Mersenne-Twister

见 Matsumoto and Nishimura [1998] 。是一个扭曲的 GFSR，周期有 $2^{19937}-1$，在 623 维是均匀的，种子是 624 个 32 位整数与一个当前位置。

下面是 R 3.4.1 的源程序包中 RNG.c 文件 Mersenne-Twister 中发生器的部分 C 代码：

```
/* ================== Mersenne Twister ========================== */
/* From http://www.math.keio.ac.jp/~matumoto/emt.html */

/* A C-program for MT19937: Real number version([0,1)-interval)
   (1999/10/28)
     genrand() generates one pseudorandom real number (double)
   which is uniformly distributed on [0,1)-interval, for each
   call. sgenrand(seed) sets initial values to the working area
   of 624 words. Before genrand(), sgenrand(seed) must be
   called once. (seed is any 32-bit integer.)
   Integer generator is obtained by modifying two lines.
     Coded by Takuji Nishimura, considering the suggestions by
   Topher Cooper and Marc Rieffel in July-Aug. 1997.

   Copyright (C) 1997, 1999 Makoto Matsumoto and Takuji Nishimura.
```

```
   When you use this, send an email to: matumoto@math.keio.ac.jp
   with an appropriate reference to your work.

   REFERENCE
   M. Matsumoto and T. Nishimura,
   "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform
   Pseudo-Random Number Generator",
   ACM Transactions on Modeling and Computer Simulation,
   Vol. 8, No. 1, January 1998, pp 3--30.
*/

/* Period parameters */
#define N 624
#define M 397
#define MATRIX_A 0x9908b0df   /* constant vector a */
#define UPPER_MASK 0x80000000 /* most significant w-r bits */
#define LOWER_MASK 0x7fffffff /* least significant r bits */

/* Tempering parameters */
#define TEMPERING_MASK_B 0x9d2c5680
#define TEMPERING_MASK_C 0xefc60000
#define TEMPERING_SHIFT_U(y)  (y >> 11)
#define TEMPERING_SHIFT_S(y)  (y << 7)
#define TEMPERING_SHIFT_T(y)  (y << 15)
#define TEMPERING_SHIFT_L(y)  (y >> 18)

static Int32 *mt = dummy+1; /* the array for the state vector  */
static int mti=N+1; /* mti==N+1 means mt[N] is not initialized */

/* Initializing the array with a seed */
static void
MT_sgenrand(Int32 seed)
{
```

```
    int i;

    for (i = 0; i < N; i++) {
    mt[i] = seed & 0xffff0000;
    seed = 69069 * seed + 1;
    mt[i] |= (seed & 0xffff0000) >> 16;
    seed = 69069 * seed + 1;
    }
    mti = N;
}

/* Initialization by "sgenrand()" is an example. Theoretically,
   there are 2^19937-1 possible states as an intial state.
   Essential bits in "seed_array[]" is following 19937 bits:
    (seed_array[0]&UPPER_MASK), seed_array[1], ..., seed_array[N-1].
   (seed_array[0]&LOWER_MASK) is discarded.
   Theoretically,
    (seed_array[0]&UPPER_MASK), seed_array[1], ..., seed_array[N-1]
   can take any values except all zeros.                             */

static double MT_genrand(void)
{
    Int32 y;
    static Int32 mag01[2]={0x0, MATRIX_A};
    /* mag01[x] = x * MATRIX_A  for x=0,1 */

    mti = dummy[0];

    if (mti >= N) { /* generate N words at one time */
    int kk;

    if (mti == N+1)   /* if sgenrand() has not been called, */
        MT_sgenrand(4357); /* a default initial seed is used   */
```

```
    for (kk = 0; kk < N - M; kk++) {
        y = (mt[kk] & UPPER_MASK) | (mt[kk+1] & LOWER_MASK);
        mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1];
    }
    for (; kk < N - 1; kk++) {
        y = (mt[kk] & UPPER_MASK) | (mt[kk+1] & LOWER_MASK);
        mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1];
    }
    y = (mt[N-1] & UPPER_MASK) | (mt[0] & LOWER_MASK);
    mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1];

    mti = 0;
    }

    y = mt[mti++];
    y ^= TEMPERING_SHIFT_U(y);
    y ^= TEMPERING_SHIFT_S(y) & TEMPERING_MASK_B;
    y ^= TEMPERING_SHIFT_T(y) & TEMPERING_MASK_C;
    y ^= TEMPERING_SHIFT_L(y);
    dummy[0] = mti;

    return ( (double)y * 2.3283064365386963e-10 ); /* reals: [0,1)-interval */
}
```

8.4.5 L'Ecuyer-CMRG 与并行计算

是 L'Ecuyer et al. 在 2002 提出的，参见 L' Ecuyer [1999] 。是两个发生器的混合：

$$
\begin{aligned}
x_n &= 1403580 x_{n-2} - 810728 x_{n-3} \pmod{2^{32} - 209} \\
y_n &= 527612 y_{n-1} - 1370589 y_{n-3} \pmod{2^{32} - 22853} \\
z_n &= (x_n - y_n) \pmod{4294967087} \\
u_n &= z_n / 4294967088
\end{aligned}
$$

并检查 $u_n = 0$ 时以接近 0 的正值代替。

这个随机数发生器的优点是算法相对简单，周期长达 2^{191}，并可以很容易地切分成长度为 2^{127} 的不同段，每一段的种子容易确定，这样，在并行计算时，每个工人节点可以从不同段的种子出发进行模拟。程序如：

```
RNGkind("L'Ecuyer-CMRG")
set.seed(1)
## start M workers
s <- .Random.seed
for (i in 1:M) {
  s <- nextRNGStream(s)
  # send s to worker i as .Random.seed
}
```

参见 R 的 parallel 包的说明文档。

Chapter 9

Julia 语言中的随机数发生器与分布类介绍 (*)

9.1 与分布有关的函数

Julia 语言的 Base 部分提供了 `rand` 函数和 `randn` 函数。`rand(n)` 返回 n 个标准均匀分布 U(0,1) 随机数，类型是 Float64 的一维数组，无自变量的 `rand()` 调用返回一个 U(0,1) 随机数。`randn(n)` 和 `randn()` 产生标准正态分布的随机数。

加载`Distributions`包后可以生成其它分布的随机数，为 Base 的 `rand` 函数增加了方法，还提供了各种与分布的有关函数。

产生随机数的函数调用格式为`rand(distribution, n)`或 `rand(distribution)`，其中 `distribution` 是一个分布，用函数调用表示，比如标准正态分布用 `Normal()` 表示，$N(10, 2^2)$ 用 `Normal(10,2)` 表示。用 `Random.seed!(k)` 设定一个随机数种子序号，这样可以在模拟时获得可重复的结果。比如，生成 100 个 Possion(2) 分布随机数:

```
using Random, Distributions
Random.seed!(101)
y = rand(Poisson(2), 100)
```

结果是 Int64 型的一维数组。作其直方图：

```
using CairoMakie
CairoMakie.activate!()
using AlgebraOfGraphics
plt = data((; y = y)) * mapping(:y) * histogram()
draw(plt, axis=(; title="Random sample from Poisson(2)"))
```

Distributions 包不仅包含了 R 软件中提供的密度函数、对数密度函数、分布函数、分位数函数、随机数函数等与分布有关的函数，还可以输出每个参数分布的各种矩的理论值，计算矩母函数和特征函数，给定观测样本后计算参数的最大似然估计，用共轭先验计算后验分布，对参数作最大后验密度估计等。分布类型包括一元随机变量 (Univariate)、随机向量 (Multivariate)、随机矩阵 (Matrixvariate)，又可分为离散型 (Discrete) 和连续型 (Continuous)。分布类型的大类名称包括：

- DiscreteUnivariateDistribution
- ContinuousUnivariateDistribution
- DiscreteMultivariateDistribution
- ContinuousMultivariateDistribution
- DiscreteMatrixDistribution
- ContinuousMatrixDistribution

Distributions 包中定义了各种概率分布。例如，`Binomial(n, p)` 表示二项分布，`Normal( , )` 表示正态分布分布，`Multinomial(n, p)` 表示多项分布，`Wishart(nu, S)` 表示 Wishart 分布。

可以从一元分布生成截断分布（truncated），如 `Truncated(Normal(mu, sigma), l, u)`。

用 `params(distribution)` 返回一个分布的参数，如

```
params(Binomial(10, 0.2))
```

`rand(distribution, n)` 生成 `n` 个指定分布的随机数，对一元分布，结果是一维数组，连续型分布的元素类型是 Float64, 离散型分布的元素类型是 Int。

对多元分布，结果是矩阵，**每列**为一个抽样。对随机矩阵分布，结果是元素为矩阵的数组。

有一些函数用来返回分布的特征量，对一元分布，有：

- `maximum()` 分布的定义域上界
- `minimum()` 分布的定义域下界
- `mean()` 期望值
- `var()` 方差
- `std()` 标准差,
- `median()` 中位数
- `modes()` 众数（可有多个）
- `mode()` 第一个众数
- `skewness()` 偏度
- `kurtosis()` 峰度，正态为 0
- `entropy()` 分布的熵

用 `mgf(distribution, t)` 计算矩母函数在 `t` 处的值,用 `cf(distribution, t)` 计算特征函数在 `t` 处的值。

用 `pdf(distribution, x)` 计算分布密度（概率质量）函数在 `x` 处的值，如果 `x` 是数组，结果返回数组；如果 `x` 是数组且函数写成 `pdf!(distribution, x)`，则结果保存在 `x` 中返回。`logpdf()` 计算其密度函数的自然对数值。

用 `cdf(distribution, x)` 计算分布函数在 `x` 处的值，`logcdf` 计算其对数值，`ccdf` 计算 1 减分布函数值，`logccdf` 计算其对数值。

用 `quantile(distribution, p)` 计算分位数函数在 `p` 处的值。

类似于 `pdf`，这些函数一般支持向量化，输入 `x` 为向量时返回向量，写成以叹号结尾的函数名时，输入为向量则结果保存在输入向量的存储中返回。可这样向量化的函数包括 `pdf`, `logpdf`, `cdf`, `logcdf`, `ccdf`, `logccdf`, `quantile`, `cquantile`, `invlogcdf`, `invlogccdf`。

比如，计算 $\mathrm{N}(10, 2^2)$ 的 0.5 和 0.975 分位数：

```
quantile(Normal(10, 2), [0.5, 0.975])
##  2-element Array{Float64,1}:
##   10.0
##   13.9199

10 + 2*quantile(Normal(), [0.5, 0.975])
##  2-element Array{Float64,1}:
##   10.0
##   13.9199
```

比如，计算标准正态分布密度并作曲线图：

```
x = range(-3.0, 3.0; length=100)
y = pdf.(Normal(), x)
plt = data((; x=x, y=y)) *
    mapping(:x, :y) * visual(Lines)
draw(plt)
```

用 `rand(distrbution)` 生成指定分布的一个抽样，用 `rand(distrbution, n)` 生成指定分布的 `n` 个抽样，用 `rand!(distribution, x)` 生成指定分布的多个抽样保存在数组 `x` 中返回。

用 `fit(distribution, x)` 作最大似然估计，比如模拟生成 $\mathrm{N}(10, 2^2)$ 样本，然后做最大似然估计：

```
Random.seed!(101)
x=rand(Normal(10, 2), 30)
fit(Normal, x)
## Distributions.Normal{Float64}(
##    =10.349862945450266,  =2.4124946578030015)
```

9.2 支持的分布

这里列举出 Julia 的 Distributions 包支持的分布。

9.2.1 连续型一元分布

- Arcsine(a,b)
- Beta(,)
- BetaPrime(,)
- Biweight(,)
- Chi( )
- Chisq( )
- Cosine(,)
- Epanechnikov(,)
- Erlang(,)
- Exponential( )
- FDist(1, 2)
- Frechet(,)
- Gamma(,)
- GeneralizedExtremeValue(, ,)
- GeneralizedPareto(, ,)
- Gumbel(,)
- InverseGamma(,)
- InverseGaussian(,)
- Kolmogorov()
- KSDist(n)
- KSOneSided(n)
- Laplace(,)
- Levy(,)
- Logistic(,)
- LogNormal(,)
- NoncentralBeta(, ,)
- NoncentralChisq(,)
- NoncentralF(1, 2,)
- NoncentralT(,)
- Normal(,)
- NormalCanon(,)
- NormalInverseGaussian(, , ,)

- Pareto(,)
- Rayleigh( )
- SymTriangularDist(,)
- TDist( )
- TriangularDist(a,b,c)
- Triweight(,)
- Uniform(a,b)
- VonMises(,)
- Weibull(,)

9.2.2 离散型一元分布

- Bernoulli(p)
- BetaBinomial(n, ,)
- Binomial(n,p)
- Categorical(p)
- DiscreteUniform(a,b)
- Geometric(p)
- Hypergeometric(s, f, n)
- NegativeBinomial(r,p)
- Poisson( )
- PoissonBinomial(p)
- Skellam(1, 2)

9.2.3 截断分布

设 distribution 为一个一元分布，用

```
Truncated(distribution, l, u)
```

可以返回一个截断分布，其中 l 为下界，u 为上界。

特别地，TruncatedNormal(mu, sigma, l, u) 表示截断正态分布。

9.2.4 多元分布

- `Multinomial`
- `MvNormal`
- `MvNormalCanon`
- `MvLogNormal`
- `Dirichlet`

9.2.5 随机矩阵分布

- `Wishart(nu, S)`
- `InverseWishart(nu, P)`

9.2.6 混合分布

通过 `MixtureModel()` 函数构造。

Part III

随机模拟

Chapter 10

随机模拟介绍

在用数学模型，包括概率统计模型处理实际应用中的问题时，我们希望建立的模型能够尽可能地符合实际情况。但是，实际情况是错综复杂的，如果一味地要求模型与实际完全相符，会导致模型过于复杂，以至于不能进行严格理论分析，结果导致模型不能使用。所以，实际建模时会忽略许多细节，增加一些可能很难验证的理论假设，使得模型比较简单，可以用数学理论进行分析研究。

这样，简化的模型就可以与实际情况有较大的差距，即使我们对模型进行了完美的理论分析，也不能保证分析结果是可信的。这一困难可以用随机模拟的方法解决。

模拟是指把某一现实的或抽象的系统的某种特征或部分状态，用另一系统（称为模拟模型）来代替或模拟。为了解决某问题，把它变成一个概率模型的求解问题，然后产生符合模型的大量随机数，对产生的随机数进行分析从而求解问题，这种方法叫做**随机模拟**方法，又称为蒙特卡洛 (Monte Carlo) 方法。

例如，一个交通路口需要找到一种最优的控制红绿灯信号的办法，使得通过路口的汽车耽搁的平均时间最短，而行人等候过路的时间不超过某一给定的心理极限值。十字路口的信号共有四个方向，每个方向又分直行、左转、右转。因为汽车和行人的到来是随机的，我们要用随机过程来描述四个方向的汽车到来和路口的行人到来过程。理论建模分析很难解决这个最优化问题。但是，我们可以采集汽车和行人到来的频率，用随机模拟方法模拟汽车和行人到来的过程，并模拟各种控制方案，记录不同方案造成的等待时间，通过统计比较找出最优

的控制方案。

随机模拟中的随机性可能来自模型本身的随机变量，比如上面描述的汽车和行人到来，也可能是把非随机的问题转换为概率模型的特征量估计问题从而用随机模拟方法解决。

例 10.1 (用随机模拟估计圆周率)**.** 为了计算圆周率 π 的近似值可以使用随机模拟方法。

如果向正方形 $D = \{(x, y) : x \in [-1, 1], y \in [-1, 1]$ 内随机等可能投点，落入单位圆 $C = \{(x, y) : x^2 + y^2 \leq 1\}$ 的概率为面积之比 $p = \frac{\pi}{4}$。如果独立重复地投了 N 个点，落入 C 中的点的个数 ξ 的平均值为 $E\xi = pN$，由概率的频率解释，

$$\frac{\xi}{N} \approx \frac{\pi}{4}, \quad \pi \approx \hat{\pi} = \frac{4\xi}{N}$$

可以这样给出 π 的近似值。

R 程序实现:

```
sim.pi <- function(N=10000){
  x <- runif(N, -1, 1)
  y <- runif(N, -1, 1)
  p <- mean((x^2 + y^2) <= 1)
  pi.hat <- 4*p
  pi.hat
}
```

测试:

```
set.seed(101)
sim.pi(1000000)
## [1] 3.145352
sim.pi(1000000)
## [1] 3.139572
```

上述模拟的 Julia 语言版本：

```
using Random, Distributions
function sim_pi(N=10000)
    x = rand(Uniform(-1.0,1.0), N)
    y = rand(Uniform(-1.0,1.0), N)
    p = mean((x .^ 2 + y .^ 2) .<= 1)
    pihat = 4*p
    pihat
end

## 测试:
Random.seed!(101)
sim_pi(1_000_000)
## 3.140552
sim_pi(1_000_000)
## 3.14062
```

※※※※※

随机模拟方法会引入所谓**随机模拟误差**：上例中估计的 $\hat{\pi}$ 实际是随机的，如果再次独立重复投 N 个点，得到的 $\hat{\pi}$ 和上一次结果会有不同。这是随机模拟方法的特点，即结果具有随机性。因为结果的随机性导致的误差叫做随机模拟误差。

使用随机模拟方法，我们必须了解随机模拟误差的大小，这样我们才能够设计合适的重复试验次数来控制随机模拟误差。比如，这个例子中 ξ 服从 $\mathrm{B}(N, \frac{\pi}{4})$ 分布，有

$$\mathrm{Var}(\hat{\pi}) = \frac{\pi(4-\pi)}{N},$$

由中心极限定理，$\hat{\pi}$ 近似服从 $\mathrm{N}(\pi, \frac{\pi(4-\pi)}{N})$ 分布，所以随机模拟误差的幅度大约在 $\pm 2\sqrt{\frac{\pi(4-\pi)}{N}}$ (随机模拟误差 95% 以上落入此区间)。

例 10.2. 用上面的例子计算随机模拟误差，并计算 95% 的误差界限绝对值。

```
NN <- 10^(2:6)
xmat <- matrix(0, length(NN), 3)
xmat[,1] <- NN
colnames(xmat) <- c("N", " 随机模拟误差", "95% 误差界")
```

```
for(i in seq(nrow(xmat))){
  xmat[i," 随机模拟误差"] <- sim.pi(NN[i]) - pi
  xmat[i, "95% 误差界"] <- 2*sqrt(pi*(4-pi)/NN[i])
}
xmat
##           N 随机模拟误差    95% 误差界
## [1,] 1e+02  0.258407346 0.328436674
## [2,] 1e+03 -0.013592654 0.103860796
## [3,] 1e+04  0.009207346 0.032843667
## [4,] 1e+05 -0.005552654 0.010386080
## [5,] 1e+06  0.001411346 0.003284367
```

Julia 语言版本：

```
Random.seed!(101)
NN = 10 .^ (2:6)
xmat = zeros(length(NN), 3)
xmat[:,1] = NN
for i=eachindex(NN)
  xmat[i,2] = sim_pi(NN[i]) - pi
  xmat[i,3] = 2*sqrt(pi*(4-pi)/NN[i])
end
xmat
## 结果略去。
```

※※※※※

这样的随机误差幅度也是随机模拟误差的典型情况，样本量增大 100 倍，误差界才减小到原来的 $\frac{1}{10}$。

随机模拟方法虽然避免了复杂的理论分析，但是其结果具有随机性，精度很难提高：为了增加一位小数点的精度，即误差减小到原来的 $\frac{1}{10}$，重复试验次数需要增加到原来的 100 倍。随机模拟方法有如下特点:

- 应用面广，适应性强。只要问题能够清楚地描述出来，就可以编写模拟程序产生大量数据，通过分析模拟数据解决问题。

- 算法简单, 容易改变条件, 但计算量大。
- 结果具有随机性，精度较低 (一般为 $O(\frac{1}{\sqrt{N}})$ 级)。
- 模拟结果的收敛服从概率论规律。
- 对维数增加不敏感。在计算定积分时，如果使用传统数值算法，维数增加会造成计算时间指数增加，但是如果使用随机模拟方法计算，则维数增加常常仅仅使计算时间随维数 d 增加而线性地增加。

随机模拟用在科学研究中，常常作为探索性试验来使用。假设科学家有了一个新的模型或技术的想法，但是不知道它的效果怎样所以还没有对其进行深入的理论分析，就可以用随机模拟方法大量地重复生成模拟数据，根据多次重复的总体效果来判断这种模型或技术的性能。如果模拟获得了好的结果，再进行深入理论分析对模型进行完善；如果模拟发现了这个模型的缺点，可以进行有针对性的修改，或者考虑转而其它解决办法。

随机模拟在科学研究中的另一种作用是说明新的模型或技术的有效性。在公开发表的统计学论文中，已经有一半以上的文章包括随机模拟结果（也叫数值结果），用来辅助说明自己提出的模型或方法的有效性。有时因为对模型或方法很难进行彻底的理论分析，仅仅使用大量的随机模拟结果来说明模型或方法的有效性。当然，因为模型都是有可变参数的，随机模拟只能针对某些参数组合给出结果，所以，一般认为仅有模拟结果而没有理论分析结果的研究论文是不全面的。第16章给出一个用随机模拟说明统计技术优良性的例子。

除了以上应用，随机模拟还是许多新的统计方法的主要工具，例如，bootstrap 置信区间和 bootstrap 偏差修正，置换检验，MCMC。利用大量计算机计算（包括随机模拟）来进行统计推断的统计学分支叫做“计算统计”（computational statistics），在本章后面各节将介绍随机模拟的一些应用和技巧。

Chapter 11

随机模拟积分

某些非随机的问题也可以通过概率模型引入随机变量，化为求随机模型的未知参数的问题。例10.1中用随机投点法估计 π 就是这样的一个例子。

随机模拟解决非随机问题的典型代表是计算定积分。通过对随机模拟定积分的讨论，可以展示随机模拟中大部分的问题和技巧。随机模拟积分也称为蒙特卡洛 (Monte Carlo) 积分 (简称 MC 积分)。实际上，统计中最常见的计算问题就是积分和最优化问题。

11.1 随机投点法

设函数 $h(x)$ 在有限区间 $[a,b]$ 上定义且有界，不妨设 $0 \le h(x) \le M$。要计算 $I = \int_a^b h(x)dx$，相当于计算曲线下的区域 $D = \{(x,y) : 0 \le y \le h(x), x \in C = [a,b]\}$ 的面积。为此在 $G = [a,b] \times (0,M)$ 上均匀抽样 N 次，得随机点 $Z_1, Z_2 \ ..., Z_N$，$Z_i = (X_i, Y_i), i = 1,2,...,N$。令

$$\xi_i = \begin{cases} 1, & Z_i \in D \\ 0, & \text{其它}, \end{cases}, \quad i = 1,...,N.$$

则 $\{\xi_i\}$ 是独立重复试验结果, $\{\xi_i\}$ 独立同 b(1,p) 分布，

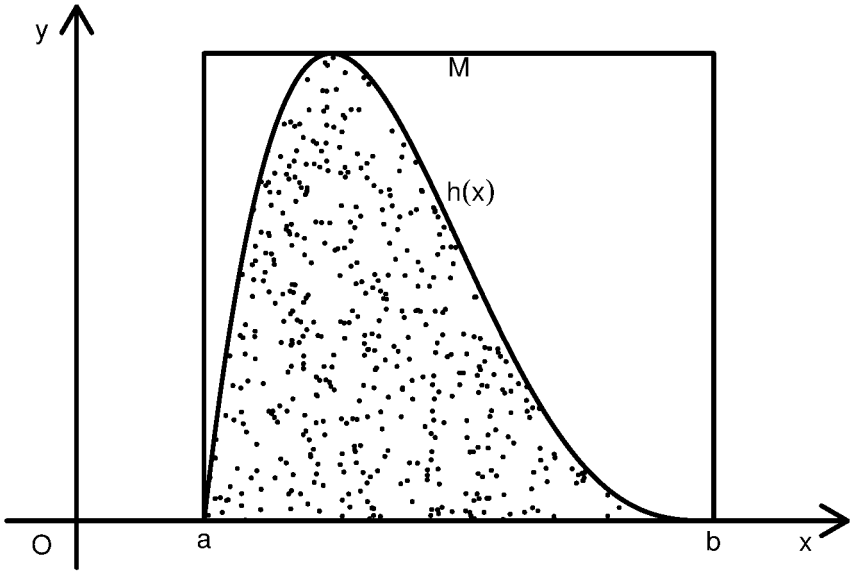
$$p = P(Z_i \in D) = V(D)/V(G) = I/[M(b-a)]. \tag{11.1}$$

其中 $V(\cdot)$ 表示区域面积。

从模拟产生的随机样本 $Z_1, Z_2\ ..., Z_N$，可以用这 N 个点中落入曲线下方区域 D 的百分比 $\hat{p}$ 来估计 (11.1)中的概率 p，然后由 $I = pM(b-a)$ 得到定积分 I 的近似值

$$\hat{I} = \hat{p}M(b-a). \tag{11.2}$$

这种方法叫做**随机投点法**。这样计算的定积分有随机性，误差中包含了随机模拟误差。



由强大数律可知

$$\begin{aligned}\hat{p} =& \frac{\sum \xi_i}{N} \to p, \text{ a.s. } (N \to \infty),\\ \hat{I} =& \hat{p}M(b-a) \to pM(b-a) = I, \text{ a.s. } (N \to \infty).\end{aligned}$$

即 $N \to \infty$ 时精度可以无限地提高（当然，在计算机中要受到数值精度的限制）。

那么，提高精度需要多大的代价呢？由中心极限定理可知

$$\sqrt{N}(\hat{p}-p)/\sqrt{p(1-p)} \xrightarrow{\mathrm{d}} \mathrm{N}(0,1), \ (N \to \infty),$$

从而

$$\sqrt{N}(\hat{I}-I)=M(b-a)(\hat{p}-p)\xrightarrow{\mathrm{d}}\mathrm{N}(0,[M(b-a)]^2p(1-p)) \tag{11.3}$$

当 N 很大时 $\hat{I}$ 近似服从 $\mathrm{N}(I,[M(b-a)]^2p(1-p)/N)$ 分布，称此近似分布的方差 $[M(b-a)]^2p(1-p)/N$ 为 $\hat{I}$ 的**渐近方差**。计算渐近方差可以用 $\hat{p}$ 代替 p 估计为 $[M(b-a)]^2\hat{p}(1-\hat{p})/N$。式(11.3)说明 $\hat{I}$ 的误差为 $O_p(\frac{1}{\sqrt{N}})$, 这样，计算 $\hat{I}$ 的精度每增加一位小数，计算量需要增加 100 倍。随机模拟积分一般都服从这样的规律。

11.2 平均值法

为了计算 $I=\int_a^b h(x)\,dx$，上面的随机投点法用了类似于舍选法的做法，在非随机问题中引入随机性时用了二维均匀分布和两点分布，靠求两点分布概率来估计积分 I。随机投点法容易理解，但是效率较低，另一种效率更高的方法是利用期望值的估计。取 $U\sim\mathrm{U}(a,b)$，则

$$\begin{aligned}E[h(U)]=&\int_a^b h(u)\frac{1}{b-a}du=\frac{I}{b-a},\\ I=&(b-a)\cdot Eh(U).\end{aligned}$$

若取 $\{U_i,i=1,\dots,N\}$ 独立同 $\mathrm{U}(a,b)$ 分布, 则 $Y_i=h(U_i),i=1,2,\dots,N$ 是 iid 随机变量列, 由强大数律，

$$\bar{Y}=\frac{1}{N}\sum_{i=1}^N h(U_i)\to Eh(U)=\frac{I}{b-a},\quad \text{a.s. } (N\to\infty)$$

于是

$$\tilde{I}=\frac{b-a}{N}\sum_{i=1}^N h(U_i) \tag{11.4}$$

是 I 的强相合估计。称这样计算定积分 I 的方法为**平均值法**。

由中心极限定理有

$$\sqrt{N}(\tilde{I}-I)\xrightarrow{\mathrm{d}}\mathrm{N}(0,(b-a)^2\mathrm{Var}(h(U))).$$

其中

$$\mathrm{Var}[h(U)] = \int_a^b \big[h(u) - Eh(U)\big]^2 \frac{1}{b-a} du \tag{11.5}$$

仅与 h 有关，仍有 $\tilde{I} - I = O_p(\frac{1}{\sqrt{N}})$, 但是(11.4)的渐近方差小于(11.2)的渐近方差 (见11.4)。$\mathrm{Var}[h(U)]$ 可以用模拟样本 $\{Y_i = h(U_i)\}$ 估计为

$$\mathrm{Var}(h(U)) \approx \frac{1}{N}\sum_{i=1}^{N}(Y_i - \bar{Y})^2.$$

如果定积分区间是无穷区间，比如 $\int_0^\infty h(x)\,dx$，为了使用均匀分布随机数以及平均值法计算积分可以做积分变换，使积分区间变成有限区间。例如，作变换 $t = 1/(x+1)$, 则

$$\int_0^\infty h(x)\,dx = \int_0^1 h(\frac{1}{t} - 1)\frac{1}{t^2}\,dt.$$

从平均值法看出，定积分问题 $\int_a^b h(x)\,dx$ 等价于求 $Eh(U)$，其中 $U \sim \mathrm{U}(a,b)$。所以这一节讨论的方法也是用来求随机变量函数期望的随机模拟方法。对一般随机变量 X, 其取值范围不必局限于有限区间，为了求 X 的函数 $h(X)$ 的期望 $I = Eh(X)$，对 X 的随机数 $X_i, i = 1, 2, \ldots, N$, 令 $Y_i = h(X_i)$, 也可以用平均值法 $\hat{I} = \frac{1}{N}\sum_{i=1}^{N} h(X_i)$ 来估计 $Eh(X)$, $\hat{I}$ 是 $Eh(X)$ 的无偏估计和强相合估计，若 $\mathrm{Var}(h(X))$ 存在，则 $\hat{I}$ 的方差为 $\frac{1}{N}\mathrm{Var}(h(X))$, $\hat{I}$ 有渐近正态分布 $\mathrm{N}(Eh(X), \frac{1}{N}\mathrm{Var}(h(X)))$。设 $\{Y_i\}$ 的样本方差为 S_N^2，可以用 S_N^2/N 估计 $\hat{I}$ 的方差, 用 $\hat{I} \pm 2S_N/\sqrt{N}$ 作为 I 的近似 95% 置信区间。

例 11.1. 设 $X \sim \mathrm{N}(0,1)$，求 $I = E|X|^{\frac{3}{2}}$。

作变量替换积分可得

$$\begin{aligned} I =& 2\int_0^\infty x^{\frac{3}{2}} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}\,dx \quad (\text{令} t = \frac{1}{2}x^2) \\ =& \frac{2^{\frac{3}{4}}\Gamma(\frac{5}{4})}{\sqrt{\pi}} \approx 0.86004 \end{aligned}$$

如果用平均值法估计 I, 取抽样样本量 $N = 10000$, 产生标准正态随机数

$X_i, i = 1, 2, \ldots, n$, 令 $Y_i = |X_i|^{\frac{3}{2}}$, 令

$$\hat{I} = \bar{Y} = \frac{1}{N}\sum_{i=1}^{N} |X_i|^{\frac{3}{2}},$$

$$S_N^2 = \frac{1}{N}\sum_{i=1}^{N}(|X_i|^{\frac{3}{2}} - \hat{I})^2,$$

一次模拟的结果得到 $\hat{I} = 0.8514$, $S_N = 0.0.9138$, 因为 I 有精确值所以可以计算误差为 $0.8514 - 0.86004 = -0.0086$。

模拟的 R 程序:

```
set.seed(101)
I.true <- 2^(3/4)*gamma(5/4)/sqrt(pi); I.true
## 0.86004
N <- 10000
yvec <- abs(rnorm(N))^1.5
I.hat <- mean(yvec); I.hat
## [1] 0.8513514
sd.hat <- sd(yvec); sd.hat
## [1] 0.9137678
## 误差:
I.hat - I.true
## -0.008688552
## 估计量的标准误差二倍:
2 * sd.hat / sqrt(N)
## 0.01827536
```

Julia 语言的模拟程序:

```
using Random, Distributions
using SpecialFunctions
Random.seed!(101)
Itrue = 2^(3/4)*gamma(5/4)/sqrt(pi); Itrue
## 0.8600399873245196
```

```
N = 10000
yvec = abs.(rand(Normal(), N)) .^ 1.5;
Ihat = mean(yvec); Ihat
## 0.943961419517864
sdhat = std(yvec); sdhat
## 0.943961419517864
## 误差:
Ihat - Itrue
## 0.017180238570l1053
## 估计量的标准误差二倍:
2* sdhat / sqrt(N)
## 0.018879228390357283
```

※※※※※

实际上，一维积分用数值方法均匀布点计算一般更有效。比如，令 $x_i = a + \frac{b-a}{N} i$, $i = 0, 1, \ldots, N$, 估计 I 为 (复合梯形求积公式)

$$I_0 = \frac{b-a}{N} \left[\frac{1}{2} h(a) + \sum_{i=1}^{N-1} h(x_i) + \frac{1}{2} h(b) \right] \tag{11.6}$$

则当 $h(x)$ 在 $[a, b]$ 上二阶连续可微时误差 $I_0 - I$ 仅为 $O(\frac{1}{N^2})$ 阶，比随机模拟方法得到的精度要高得多，而且当 $h(x)$ 有更好的光滑性时还可以用更精确的求积公式得到更高精度。所以，被积函数比较光滑的一元定积分问题一般不需要用随机模拟来计算。在 R 中可以用 `integrate(f, lower, upper)` 计算定积分。如例11.1中求 $E|X|^{3/2}$ 涉及到的积分可以用 R 函数计算为：

```
I.int <- integrate(function(x) x^1.5*dnorm(x), 0, Inf); I.int
## 0.43002 with absolute error < 1e-06
2*I.int$value
## [1] 0.86004
```

Julia 的 QuadGK 包的 `quadgk()` 函数用自适应高斯-康拉德数值积分方法计算一元函数积分，如

```
import QuadGK
Itrue, err = QuadGK.quadgk(x -> x^1.5/sqrt(2*pi)*exp(-0.5*x^2), 0, Inf)
## (0.43001999365557575, 4.014291709725467e-9)
```

`quadgk()` 返回包括积分值和误差估计界限的二元组。

有表达式的定积分问题，有时可以利用符号计算软件求得精确结果，附录C介绍了符号计算软件 Maxima。对例11.1的期望问题，在 wxMaxima 中键入

```
2*integrate(x^(3/2)*1/sqrt(2*%pi)*exp(-1/2*x^2), x, 0, inf);
```

然后按 Shift+Enter 组合键进行计算，显示结果为

$$\frac{\Gamma\left(\frac{1}{4}\right)}{2^{5/4}\sqrt{\pi}}$$

※※※※※

11.3 随机模拟估计误差的评估与样本量计算

许多简单的随机模拟问题都可以看成是估计 $\theta = EY$ 的问题，设 $\mathrm{Var}(Y) = \sigma^2$，生成 Y 的独立同分布样本 $Y_i, i = 1, 2, \ldots, N$，用

$$\hat{\theta} = \bar{Y}_N = \frac{1}{N}\sum_{i=1}^{N} Y_i \tag{11.7}$$

估计 θ。令

$$S_N^2 = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \bar{Y}_N)^2.$$

当 $N \to \infty$ 时 $\hat{\theta} \to \theta$ a.s.，$S_N^2 \to \sigma^2$ a.s., 且 $\hat{\theta}$ 近似服从如下正态分布：

$$\mathrm{N}\left(\theta, \frac{\sigma^2}{N}\right), \tag{11.8}$$

其中的 σ^2 可以用 S_N^2 近似，即

$$\frac{\hat{\theta} - \theta}{S_N/\sqrt{N}} \xrightarrow{\mathrm{d}} \mathrm{N}(0, 1),$$

所以 N 充分大时 $\hat{\theta}$ 近似服从正态分布

$$\mathrm{N}\left(\theta, \frac{S_N^2}{N}\right),$$

因为估计 $\hat{\theta}$ 有随机误差，这里给出估计误差的一些度量。

定义估计的根均方误差为

$$\mathrm{RMSE} = \sqrt{E(\hat{\theta}-\theta)^2}. \tag{11.9}$$

在以上用独立同分布样本均值估计期望值 $\theta = EY$ 的做法中，因为 $E\hat{\theta} = \theta$，所以

$$\mathrm{RMSE} = \sqrt{\mathrm{Var}(\hat{\theta})} = \sqrt{\mathrm{Var}(\bar{Y}_n)} = \frac{\sigma}{\sqrt{N}}, \tag{11.10}$$

所以根均方误差可以用 $\frac{S_N}{\sqrt{N}}$ 估计。

定义估计的平均绝对误差为

$$\mathrm{MAE} = E|\hat{\theta}-\theta|. \tag{11.11}$$

当 N 充分大时由 $\hat{\theta}$ 的近似正态分布式(11.8)可知

$$\begin{aligned}\mathrm{MAE} =& E|\hat{\theta}-\theta| = \frac{\sigma}{\sqrt{N}} E\left|\frac{\hat{\theta}-\theta}{\sigma/\sqrt{N}}\right| \\ \approx& \sqrt{\frac{2}{\pi}}\frac{\sigma}{\sqrt{N}} \approx 0.7979\frac{\sigma}{\sqrt{N}}.\end{aligned} \tag{11.12}$$

所以平均绝对误差可以用 $0.80\frac{S_N}{\sqrt{N}} = 0.80\mathrm{RMSE}$ 估计。

定义估计的平均相对误差为

$$\mathrm{MRE} = E\left|\frac{\hat{\theta}-\theta}{\theta}\right| = \mathrm{MAE}/\theta \tag{11.13}$$

所以平均相对误差可以用 $0.80\frac{S_N}{\sqrt{N}\hat{\theta}} = 0.80\frac{\mathrm{RMSE}}{\hat{\theta}}$ 估计。平均相对误差为 0.005 相当于估计结果有两位有效数字，平均相对误差为 0.0005 相当于估计结果有三位有效数字。

由 $\hat{\theta}$ 的近似正态分布式(11.8)，估计 $\hat{\theta}$ 的绝对误差 $|\hat{\theta}-\theta|$ 的近似 95% 上界为 $2\frac{S_N}{\sqrt{N}}$:

$$P\left(|\hat{\theta}-\theta| \le 2\frac{S_N}{\sqrt{N}}\right) \approx 95\%,$$

所以也可以用 $2\frac{S_N}{\sqrt{N}} = 2\text{RMSE}$ 作为绝对误差大小的一个度量，用 $2\frac{S_N}{\sqrt{N}\bar{\theta}}$ 作为相对误差大小的一个度量。

为了计算样本量 N 需要取多大才能控制估计的根均方误差小于 σ_0, 可以先取较小的 N_0 抽取 N_0 个样本值计算出 $S_{N_0}^2$，用 $S_{N_0}^2$ 估计总体方差 σ^2，然后求需要的 N 的大小:

$$\frac{S_{N_0}}{\sqrt{N}} < \sigma_0, \quad N > \frac{S_{N_0}^2}{\sigma_0^2}.$$

用 $\hat{\theta} = \bar{Y}_N$ 估计 $\theta = EY$ 时，可以利用近似正态分布式(11.8)计算 θ 的近似 95% 置信区间：

$$\hat{\theta} \pm 2\frac{S_N}{\sqrt{N}}. \tag{11.14}$$

在更复杂的随机模拟问题中，参数 θ 不是用独立同分布样本的均值来估计，设样本量为 N 时参数估计为 $\hat{\theta}$ 并设 $\hat{\theta}$ 是 θ 的相合估计。这时 $\hat{\theta}$ 的方差 $\text{Var}(\hat{\theta})$ 没有简单的公式，$\hat{\theta}$ 也没有简单易得的渐近分布。为了评估 $\hat{\theta}$ 的随机误差大小，可以独立地执行 B 次模拟，每次得到一个估计量 $\hat{\theta}^{(j)}$, $j = 1, 2, \ldots, B$。令 $\bar{\theta} = \frac{1}{B}\sum_{j=1}^{B}\hat{\theta}^{(j)}$。估计根均方误差 RMSE 为

$$\widehat{\text{RMSE}} = \sqrt{\frac{1}{B}\sum_{j=1}^{B}(\hat{\theta}^{(j)} - \theta)^2}, \tag{11.15}$$

但其中的 θ 未知，仅能在已知 θ 真值的验证问题中采用此计算公式。如果已知 $\hat{\theta}$ 是无偏或渐近无偏估计，则可估计根均方误差为

$$\widehat{\text{RMSE}} = \sqrt{\frac{1}{B}\sum_{j=1}^{B}(\hat{\theta}^{(j)} - \bar{\theta})^2}. \tag{11.16}$$

类似地，若 θ 已知，估计平均绝对误差为

$$\widehat{\text{MAE}} = \frac{1}{B}\sum_{j=1}^{B}\left|\hat{\theta}^{(j)} - \theta\right|, \tag{11.17}$$

估计平均相对误差为

$$\widehat{\text{MRE}} = \frac{1}{\theta}\frac{1}{B}\sum_{j=1}^{B}\left|\hat{\theta}^{(j)} - \theta\right|, \tag{11.18}$$

在 $\hat{\theta}$ 无偏或渐近无偏时可以将式(11.17)和(11.18)中的 θ 用 $\bar{\theta}$ 代替。

重复 B 次模拟试验有时需要耗费太多时间，以至于无法做到。这时，可以参考后面讲到的 bootstrap 估计的想法来评估估计误差。

例 11.2. 对例11.1用平均值法做的估计进行误差分析。

取 $N = 10000$ 个样本点的一次模拟的结果得到 $\hat{I} = 0.8754$, $S_N = 0.9409$, 根均方误差的估计为 $\text{RMSE} = \frac{S_N}{\sqrt{N}} = 0.0094$, 平均绝对误差的估计为 $\text{MAE} = 0.80\text{RMSE} = 0.0075$, 平均相对误差的估计为 $\text{MRE} = \text{MAE}/\hat{I} = 0.0087$, 说明结果只有约两位有效数字精度。为了达到四位小数的精度，需要平均相对误差控制在 0.00005 左右, 需要解 $0.8\frac{S_N}{\sqrt{N}\hat{I}} = 0.00005$, 代入得 N 需要约 3 亿次, 可见随机模拟方法提高精度的困难程度。

以上结果的计算程序：

```
set.seed(1)
I.true <- 2^(3/4)*gamma(5/4)/sqrt(pi); I.true
## [1] 0.86004
N <- 10000
yvec <- abs(rnorm(N))^1.5
I.hat <- mean(yvec); I.hat
## [1] 0.8753857
sd.hat <- sd(yvec); sd.hat
## [1] 0.9409092
RMSE <- sd.hat/sqrt(N); RMSE
## [1] 0.009409092
MAE <- 0.80*RMSE; MAE
## [1] 0.007527274
MRE <- MAE/I.hat; MRE
## [1] 0.008598809
N.more <- (0.8*sd.hat/I.hat/0.00005)^2; N.more
## [1] 295758038
```

为了得到更可靠的 RMSE、MAE 和 MRE 的估计，重复模拟 $B = 100$ 次，得到 100 个 $\hat{I}$ 的值，从而估计 RMSE 为 0.0082, MAE 为 0.0070, MRE 为 0.0081。从一次模拟得到的误差估计与重复 100 次得到的误差估计相差不大。

计算程序为：

```
set.seed(2)
I.true <- 2^(3/4)*gamma(5/4)/sqrt(pi)
N <- 10000
B <- 100
xvec <- numeric(B) # 保存 B 个估计值
for(j in 1:B){
  xvec[j] <- mean(abs(rnorm(N))^1.5)
}
bar.I <- mean(xvec)
RMSE2 <- sqrt( mean( (xvec - bar.I)^2  )); RMSE2
## [1] 0.00823417
MAE2 <- mean( abs(xvec - bar.I) ); MAE2
## [1] 0.006991147
MRE2 <- MAE2 / bar.I; MRE2
## [1] 0.00813205
```

※※※※※

上面的误差评估仅考虑了精度与计算量之间的关系，在相同精度下，如果重复模拟次数 N 较少的算法每次模拟耗时很多，最终可能 N 较小的算法反而花费更多的时间。所以我们考虑在总耗时约束（也可以推广到计算资源约束）下不同算法的选择问题。设模拟计算总的时间限制为 T，(11.7)算法计算每个 Y_i 需要的时间长度为 t，所以在时间约束下可以采用的重复次数 $N = \text{floor}(T/t) \approx T/t$。这样，因为

$$\frac{\hat{\theta} - \theta}{\sigma/\sqrt{N}} \xrightarrow{\text{d}} \text{N}(0,1),\ (N \to \infty)$$

有

$$\frac{\hat{\theta} - \theta}{\sigma/\sqrt{T/t}} \xrightarrow{\text{d}} \text{N}(0,1),\ (T \to \infty)$$

从而

$$\sqrt{T}(\hat{\theta} - \theta) \xrightarrow{\text{d}} \text{N}\left(0, \sigma^2 t\right),\ (T \to \infty)$$

从而 $\hat{\theta}$ 近似服从

$$\text{N}\left(\theta, \frac{\sigma^2 t}{T}\right).$$

如果两个算法分别的单次模拟耗时为 t_1, t_2，分别的单次模拟方差为 σ_1^2, σ_2^2, 则为了比较在相同总耗时约束下两个算法的计算精度，只要比较 $\sigma_1^2 t_1$ 和 $\sigma_2^2 t_2$ 的值，此值较小的算法在相同的总耗时下可以获得更高精度，在相同的精度下可以花费更少的计算时间完成模拟。参见 [Glasserman, 2004]。

11.4 高维定积分

上面的两种计算一元函数定积分的方法可以很容易地推广到多元函数定积分，或称高维定积分。设 d 元函数 $h(x_1, x_2, \dots, x_d)$ 定义于超矩形

$$C = \{(x_1, x_2, \dots, x_d) : a_i \le x_i \le b_i, i = 1, 2, \dots, d\}$$

且

$$0 \le h(x_1, \dots, x_d) \le M, \ \forall x \in C.$$

令

$$\begin{aligned} D =& \{(x_1, x_2, \dots, x_d, y) : (x_1, x_2, \dots, x_d) \in C, \ 0 \le y \le h(x_1, x_2, \dots, x_d)\}, \\ G =& \{(x_1, x_2, \dots, x_d, y) : (x_1, x_2, \dots, x_d) \in C, \ 0 \le y \le M\} \end{aligned}$$

为计算 d 维定积分

$$I = \int_{a_d}^{b_d} \cdots \int_{a_2}^{b_2} \int_{a_1}^{b_1} h(x_1, x_2, \dots, x_d)\, dx_1 dx_2 \cdots dx_d, \tag{11.19}$$

产生服从 $d+1$ 维空间中的超矩形 G 内的均匀分布的独立抽样 $Z_1, Z_2, \dots, Z_N$, 令

$$\xi_i = \begin{cases} 1, & Z_i \in D \\ 0, & Z_i \in G - D \end{cases}, \quad i = 1, 2, \dots, N$$

则 ξ_i iid b(1,p),

$$p = P(Z_i \in D) = \frac{V(D)}{V(G)} = \frac{I}{MV(C)} = \frac{I}{M \prod_{j=1}^{d} (b_j - a_j)}$$

其中 $V(\cdot)$ 表示区域体积。令 $\hat{p}$ 为 N 个随机点中落入 D 的百分比，则

$$\hat{p} = \frac{\sum \xi_i}{N} \to p, \ \text{a.s.}(N \to \infty),$$

用

$$\hat{I}_1 = \hat{p}V(G) = \hat{p} \cdot M\, V(C) = \hat{p} \cdot M \prod_{j=1}^{d}(b_j - a_j) \tag{11.20}$$

估计积分 I，则 $\hat{I}_1$ 是 I 的无偏估计和强相合估计。称用式(11.20)计算高维定积分 I 的方法为随机投点法。

由中心极限定理知

$$\sqrt{N}(\hat{p} - p)/\sqrt{p(1-p)} \xrightarrow{d} \mathrm{N}(0,1),$$

$$\sqrt{N}(\hat{I}_1 - I) \xrightarrow{d} \mathrm{N}\left(0, \left(M \prod_{j=1}^{n}(b_j - a_j)\right)^2 p(1-p)\right),$$

$\hat{I}_1$ 的渐近方差为

$$\frac{\left(M \prod_{j=1}^{n}(b_j - a_j)\right)^2 p(1-p)}{N} \tag{11.21}$$

所以 $\hat{I}_1$ 的随机误差仍为 $O_p\left(\frac{1}{\sqrt{N}}\right)$, $N \to \infty$ 时的误差阶不受维数 d 的影响，这是随机模拟方法与其它数值计算方法相比一个重大优势。

在计算高维积分(11.19)时，仍可以通过估计 $Eh(U)$ 来获得, 其中 U 服从 R^d 中的超矩形 C 上的均匀分布。设 $U_i \sim$ iid $\mathrm{U}(C)$，$i = 1, 2, \ldots, N$, 则 $h(U_i), i = 1, 2, \ldots, N$ 是 iid 随机变量列,

$$Eh(U_i) = \int_C h(u) \frac{1}{\prod_{j=1}^{d}(b_j - a_j)} du = \frac{I}{\prod_{j=1}^{d}(b_j - a_j)},$$

估计 I 为

$$\hat{I}_2 = \prod_{j=1}^{d}(b_j - a_j) \cdot \frac{1}{N} \sum_{i=1}^{N} h(U_i), \tag{11.22}$$

称用式(11.22)计算高维定积分 I 的方法为平均值法。由强大数律

$$\hat{I}_2 \to \prod_{j=1}^{d}(b_j - a_j) Eh(U) = I, \text{ a.s.}(N \to \infty),$$

$\hat{I}_2$ 的渐近方差为

$$\frac{(V(C))^2 \mathrm{Var}(h(U))}{N} = \frac{\left(\prod_{j=1}^{d}(b_j - a_j)\right)^2 \mathrm{Var}(h(U))}{N}. \tag{11.23}$$

$N \to \infty$ 时的误差阶也不受维数 d 的影响。

我们来比较随机投点法(11.20)与平均值法(11.22)的精度，只要比较其渐近方差。对 $I = \int_C h(x)dx$，设 $\hat{I}_1$ 为随机投点法的估计，$\hat{I}_2$ 为平均值法的估计。因设 $0 \le h(x) \le M$，不妨设 $0 \le h(x) \le 1$, 取 $h(x)$ 的上界 $M = 1$。

令 $X_i \sim$ iid U(C), $\eta_i = h(X_i)$, $Y_i \sim$ iid U(0,1) 与 $\{X_i\}$ 独立，

$$\xi_i = \begin{cases} 1, & \text{当} Y_i \le h(X_i) \\ 0, & \text{当} Y_i > h(X_i) \end{cases} \quad i = 1, 2, \dots, N,$$

这时有

$$\hat{I}_1 = V(C)\frac{1}{N}\sum_{i=1}^{N}\xi_i, \qquad \hat{I}_2 = V(C)\frac{1}{N}\sum_{i=1}^{N}\eta_i$$

$$\mathrm{Var}(\hat{I}_1) = \frac{1}{N}V^2(C)\cdot\frac{I}{V(C)}\left(1 - \frac{I}{V(C)}\right)$$

$$\mathrm{Var}(\hat{I}_2) = \frac{1}{N}V^2(C)\cdot\left(\frac{1}{V(C)}\int_C h^2(x)dx - \left(\frac{I}{V(C)}\right)^2\right)$$

$$\mathrm{Var}(\hat{I}_1) - \mathrm{Var}(\hat{I}_2) = \frac{V(C)}{N}\int_C \left\{h(x) - h^2(x)\right\} dx \ge 0$$

可见平均值法精度更高。事实上，随机投点法多用了随机数 Y_i，必然会增加抽样随机误差。

在计算高维积分(11.19)时，如果用网格法作数值积分，把超矩形 $C = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d]$ 的每个边分成 n 个格子，就需要 $N = n^d$ 个格子点，如果用每个小超矩形的中心作为代表点，可以达到 $O(n^{-2})$ 精度，即 $O(N^{-2/d})$，当维数增加时为了提高一倍精度需要 $2^{d/2}$ 倍的代表点。比如 $d = 8$，精度只有 $O(N^{-1/4})$。高维的问题当维数增加时计算量会迅猛增加，以至于无法计算，这个问题称为**维数诅咒**。如果用 Monte Carlo 积分，则精度为 $O_p(N^{-1/2})$，与 d 关系不大。所以 Monte Carlo 方法在高维积分中有重要应用。为了提高积分计算精度，需要减小 $O_{\mathrm{p}}(N^{-1/2})$ 中的常数项，即减小 $\hat{I}$ 的渐近方差。

例 11.3. 考虑

$$f(x_1, x_2, \dots, x_d) = x_1^2 x_2^2 \cdots x_d^2,\ 0 \le x_1 \le 1, 0 \le x_2 \le 1, \dots, 0 \le x_d \le 1$$

的积分

$$I = \int_0^1 \cdots \int_0^1 \int_0^1 x_1^2 x_2^2 \cdots x_d^2 \, dx_1 dx_2 \cdots dx_d$$

当然，这个积分是有精确解 $I=(1/3)^d$ 的，对估计 I 的结果我们可以直接计算误差。以 $d=8$ 为例比较网格点法、随机投点法、平均值法的精度，这时真值 $I=(1/3)^8\approx 1.5241587\times 10^{-4}$。

用 n 网格点法，$N=n^d$，公式为

$$I_0=\frac{1}{N}\sum_{i_1=1}^{n}\sum_{i_2=1}^{n}\cdots\sum_{i_d=1}^{n}f(\frac{2i_1-1}{2n},\frac{2i_2-1}{2n},\cdots,\frac{2i_d-1}{2n})$$

误差绝对值为 $e_0=|I_0-I|$。如果取 $n=5,d=8$，需要计算 $N=390625$ 个点的函数值，计算量相当大。

计算程序：

```
d <- 8        # 维数
N <- 5^d      # 计算函数点数
B <- 100000   # 重复次数
I <- (1/3)^d # 积分的真值
```

积分真值：

```
I
## [1] 0.0001524158
```

网格法:

```
n <- N^{1/d}
I0 <- mean((seq(n)/n - 0.5/n)^2)^d; I0
## [1] 0.0001406409
```

网格法误差和相对误差:

```
I0 - I
## [1] -1.177493e-05
(I0-I)/I
## [1] -0.07725531
```

相对误差约 8%，误差较大。

用随机投点法求 I，先在 $(0,1)^d \times (0,1)$ 均匀抽样 $(\xi_i^{(1)}, \xi_i^{(2)}, \ldots, \xi_i^{(d)}, y_i), i = 1, \ldots, N$，令 $\hat{I}_1$ 为 $y_i \le f(\xi_i^{(1)}, \xi_i^{(2)}, \ldots, \xi_i^{(d)})$ 成立的百分比。一次估计的程序和结果为

```
set.seed(1)
x <- runif(N*d)
y <- runif(N)
fx <- apply(matrix(x, ncol=d)^2, 1, prod)
I1 <- mean(y <= fx); I1
## [1] 0.0001664
sd1 <- sqrt(I1*(1-I1)); sd1
## 0.01289854
```

估计 RMSE、MAE 和 MRE 来度量误差大小：

```
RMSE1 <- sd1/sqrt(N); RMSE1
## [1] 2.063766e-05
MAE1 <- 0.8*RMSE1; MAE1
## [1] 1.651013e-05
MRE1 <- MAE1/I1; MRE1
## [1] 0.09921953
```

估计的相对误差约为 10 个百分点，误差较大。为了确认这样的误差估计，将随机投点法重复模拟 $B = 100$ 次，得到 100 个 I 的估计，程序为：

```
set.seed(1)
B <- 100
Ihat <- numeric(B)
for(ib in 1:B){
  x <- runif(N*d)
  y <- runif(N)

  fx <- apply(matrix(x, ncol=d)^2, 1, prod)
```

```
  Ihat[ib] <- mean(y <= fx)
}
```

用 $B = 100$ 次重复模拟估计平均相对误差：

```
e1 <- mean(abs(Ihat - mean(Ihat)))/mean(Ihat); e1
## [1] 0.1049589
```

这确认了从一次模拟给出的平均相对误差估计，两者基本相同。

用平均值方法，公式为

$$\hat{I}_2 = \frac{1}{N}\sum_{i=1}^{N} f(\xi_i^{(1)}, \xi_i^{(2)}, \cdots, \xi_i^{(d)})$$

其中 $\xi_i^{(j)}, i = 1, \ldots, N, j = 1, \ldots, d$ 独立同 U(0,1) 分布。一次计算的程序为:

```
set.seed(1)
x <- runif(N*d)
fx <- apply(matrix(x, ncol=d)^2, 1, prod)
I2 <- mean(fx); I2
## [1] 0.0001513103
sd2 <- sd(fx); sd2
## [1] 0.001638286
```

平均值法的各种误差度量为：

```
RMSE2 <- sd2/sqrt(N); RMSE2
## [1] 2.621258e-06
MAE2 <- 0.8*RMSE2; MAE2
## [1] 2.097006e-06
MRE2 <- MAE2/I2; MRE2
## [1] 0.01385898
```

相对误差约为 1.3 个百分点。

类似地，重复 $B = 100$ 次，估计 MRE：

```
set.seed(1)
B <- 100
Ihat2 <- numeric(B)
for(ib in 1:B){
  x <- runif(N*d)
  y <- runif(N)

  fx <- apply(matrix(x, ncol=d)^2, 1, prod)
  Ihat2[ib] <- mean(fx)
}
```

用 $B = 100$ 次重复模拟估计平均相对误差：

```
e2 <- mean(abs(Ihat2 - mean(Ihat2) ))/mean(Ihat2); e2
## [1] 0.01381025
```

相对误差约为 1.4 个百分点，与从一次模拟估算的平均相对误差相同。

随机模拟程序是消耗计算资源特别严重的计算任务之一，设计更高效的算法、程序进行优化、考虑并行计算，都是随机模拟问题的实现需要考虑的。对随机模拟问题，高效的算法主要指估计的方差更小，这样误差小，达到一定精度需要的计算量就小。

最后，三种方法计算量相同（都计算 $N = 5^8$ 次函数值）的情况下，平均相对误差分别为：

```
round(c(" 网格法"=abs(I0-I)/I, " 随机投点法"=e1, " 平均值法"=e2), 3)
##      网格法 随机投点法    平均值法
##       0.077      0.105      0.014
```

随机投点法的误差与网格法相近；平均值法的误差只有随机投点法的 13%。

※※※※※

习题

习题 1

设 $\{U_i, i = 1, 2, ...\}$ 为独立同 U(0,1) 分布的随机变量序列。令

$$K = \min\left\{k : \sum_{i=1}^{k} U_i > 1\right\}.$$

(1) 证明 $EK = e$。
(2) 生成 K 的 N 个独立抽样，用平均值 $\bar{K}$ 估计 e。
(3) 估计 $\bar{K}$ 的标准差，给出 e 的近似 95% 置信区间。

习题 2

设 $\{U_i, i = 1, 2, ...\}$ 为独立同 U(0,1) 分布的随机变量序列。令 M 为序列中第一个比前一个值小的元素的序号，即

$$M = \min\{m : U_1 \le U_2 \le \cdots \le U_{m-1}, U_{m-1} > U_m,\ m \ge 2\}$$

(1) 证明 $P(M > n) = \frac{1}{n!}$, $n \ge 2$。
(2) 用概率论中的恒等式 $EM = \sum_{n=0}^{\infty} P(M > n)$ 证明 $EM = e$。
(3) 生成 M 的 N 个独立抽样，用平均值 $\bar{M}$ 估计 e。
(4) 估计 $\bar{M}$ 的标准差，给出 e 的近似 95% 置信区间。

Chapter 12

重要抽样法

$I = \int_C h(x)\,dx$ 中积分区域 C 可能是任意形状的, 也可能无界，$h(x)$ 在 C 内各处的取值大小差异可能很大，使得直接用平均值法估计 I 时，很多样本点处于 $|h(x)|$ 接近于零的地方，造成浪费，另外使得 $\hat{I}_2$ 的渐近方差 (见式(11.23)) 中的 $\mathrm{Var}(h(U))$ 很大 ($U \sim \mathrm{U}(C)$)。为此，考虑用非均匀抽样: $|h(x)|$ 大的地方密集投点，$|h(x)|$ 小的地方稀疏投点。这样可以有效利用样本。设 $g(x), x \in C$ 是一个密度，其形状与 $|h(x)|$ 相近，且当 $g(x) = 0$ 时 $h(x) = 0$，当 $\|x\| \to \infty$ 时 $h(x) = o(g(x))$。称 $g(x)$ 为**试投密度**或重要抽样密度。

设 X_i iid $\sim g(x)$, $i = 1, 2, \ldots, N$。令

$$\eta_i = \frac{h(X_i)}{g(X_i)},\ i = 1, 2, \ldots, N,$$

则

$$E\eta_1 = \int_C \frac{h(x)}{g(x)} g(x)\,dx = \int_C h(x)\,dx = I,$$

因此可以用 $\{\eta_i, i = 1, 2, \ldots, N\}$ 的样本平均值来估计 I，即

$$\hat{I}_3 = \bar{\eta} = \frac{1}{N}\sum_{i=1}^{N} \frac{h(X_i)}{g(X_i)}. \tag{12.1}$$

$\hat{I}_3$ 是 I 的无偏估计和强相合估计。$\hat{I}_3$ 的渐近方差为

$$\mathrm{Var}(\hat{I}_3) = \mathrm{Var}\left(\frac{h(X)}{g(X)}\right)\frac{1}{N} = O(\frac{1}{N}), \tag{12.2}$$

当 $g(x)$ 和 $|h(x)|$ 形状很接近时 $\frac{|h(x)|}{g(x)}$ 近似为常数，方差 $\mathrm{Var}(\hat{I}_3)$ 很小, 这时 $\hat{I}_3$ 的随机误差可以很小。用(12.1)式估计 I 与舍选法 II 有类似的想法，这种方法叫做**重要抽样法** (importance sampling)，是随机模拟的重要方法。

为什么当 $g(x)$ 和 $|h(x)|$ 形状很接近时 $\mathrm{Var}(\hat{I}_3)$ 很小？事实上，

$$\begin{aligned}\mathrm{Var}\left(\frac{h(X)}{g(X)}\right) =& E\left(\frac{h^2(X)}{g^2(X)}\right) - \left(E\frac{h(X)}{g(X)}\right)^2 = E\left(\frac{h^2(X)}{g^2(X)}\right) - \left(\int_C h(x)\,dx\right)^2 \\ \geq& \left(E\frac{|h(X)|}{g(X)}\right)^2 - I^2 \quad \text{(Jensen 不等式)} \qquad (12.3)\\ =& \left(\int_C |h(x)|\,dx\right)^2 - I^2, \qquad (12.4)\end{aligned}$$

当且仅当 $\frac{|h(X)|}{g(X)}$ 为常数时(12.3)的不等式中的等号成立，即 $g(x) = \frac{1}{\int_C |h(t)|\,dt}|h(x)|, x \in C$ 时 $\mathrm{Var}(\hat{I}_3)$ 达到最小。

上述求积分的问题也可以表述为求随机变量函数的期望问题。设 $Y \sim f(y)$，要求 Y 的函数 $h(Y)$ 的期望值 $Eh(Y) = \int h(y)f(y)\,dy$, 可以抽取 Y 的随机数 $Y_i, i = 1, 2, \ldots, N$, 然后用平均值法 $\frac{1}{N}\sum_{i=1}^N h(Y_i)$ 估计 $Eh(Y)$。但是，如果很难生成 Y 的随机数，或者 Y 的分布集中于 $h(y)$ 接近于零的位置以至于积分效率很低，可以找一个试投密度 $g(x)$，从 $g(x)$ 产生随机数 $X_i, i = 1, 2, \ldots, N$, 用如下重要抽样法

$$\hat{I}_{3.1} = \frac{1}{N}\sum_{i=1}^N h(X_i)\frac{f(X_i)}{g(X_i)} \qquad (12.5)$$

估计 $Eh(Y)$, 易见 $E\hat{I}_{3.1} = Eh(Y)$。称 $W_i = \frac{f(X_i)}{g(X_i)}$ 为**重要性权重**, $Eh(Y)$ 可以用重要性权重估计为

$$\hat{I}_{3.1} = \frac{1}{N}\sum_{i=1}^N W_i h(X_i). \qquad (12.6)$$

选取试投密度 $g(x)$ 时，要求当 $h(x)f(x) \neq 0$ 时 $g(x) \neq 0$, 当 x 趋于无穷时 $h(x)f(x) = o(g(x))$（$g(x)$ 相对厚尾）, 一般还要求 $g(x)$ 的形状与 $|h(x)|f(x)$ 形状接近。如果 $g(x)$ 的相对厚尾性难以确定，可以使用如下保险的试投密度

$$\tilde{g}(x) = \rho g(x) + (1-\rho)r(x), \qquad (12.7)$$

其中 ρ 接近于 1，$r(x)$ 是柯西分布或 Pareto 分布这样的重尾分布。要生成 N 个 $\tilde{g}(x)$ 的随机数，只要生成 $N\rho$ 个 $g(x)$ 的随机数和 $N(1-\rho)$ 个 $r(x)$ 的随机数。

选取不适当的试投密度会把绝大多数样本点投到了对计算积分不重要的位置，使得样本点中只有极少数点是真正有作用的。在多维问题中合适的试投密度尤其难找，经常需要反复试验。

例 12.1. 用 MC 积分法计算 $I = \int_0^1 e^x\,dx = e - 1 \approx 1.718$。对被积函数 $h(x) = e^x$ 做泰勒展开得

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots$$

取

$$g(x) = c(1 + x) = \frac{2}{3}(1 + x)$$

要产生 $g(x)$ 的随机数可以用逆变换法，密度 $g(x)$ 的分布函数 $G(x)$ 的反函数为

$$G^{-1}(y) = \sqrt{1 + 3y} - 1,\ 0 < y < 1$$

因此，取 U_i iid U(0,1), 令 $X_i = \sqrt{1 + 3U_i} - 1$, $i = 1, 2, \ldots, N$, 则重要抽样法的积分公式为

$$\hat{I}_3 = \frac{1}{N}\sum_{i=1}^{N} \frac{e^{X_i}}{\frac{2}{3}(1 + X_i)}$$

渐近方差为

$$\mathrm{Var}(\hat{I}_3) = \frac{1}{N}\left(\frac{3}{2}\int_0^1 \frac{e^{2x}}{1 + x}dx - I^2\right) \approx 0.02691/N.$$

积分真值：

```
I.true <- exp(1)-1; I.true
## [1] 1.718282
```

重要抽样法的程序实现：

```
set.seed(1)
N <- 10000
Ginv <- function(y) sqrt(1 + 3*y) - 1
X <- Ginv(runif(N))
eta <- exp(X) / (2/3*(1 + X))
I3 <- mean( eta )
var3 <- var(eta)
```

一次模拟的估计值和绝对误差、相对误差：

```
I3
## [1] 1.71873
abs(I3 - I.true)
## [1] 0.0004481696
(I3 - I.true)/I.true
## [1] 0.0002608243
```

相对误差很小，说明有效位数在三位以上。

用一次模拟的结果估计平均相对误差为：

```
MRE3 <- 0.8*sqrt(var3)/sqrt(N)/I3; MRE3
## [1] 0.0007703735
```

说明估计精度大约有三位有效数字。

从一次模拟中对 $N \cdot \mathrm{Var}(\hat{I}_3)$ 进行估计：

```
var3
## [1] 0.0273929
```

理论值是 0.02691。

如果用平均值法，估计公式为

$$\hat{I}_2 = \frac{1}{N}\sum_{i=1}^{N} e^{U_i},$$

渐近方差为

$$\mathrm{Var}(\hat{I}_2) = \frac{1}{N}\left(\int_0^1 e^{2x}\,dx - I^2\right) \approx 0.2420/N \approx 9.0 \times \mathrm{Var}(\hat{I}_3)$$

是重要抽样法方差的 9 倍。因为随机模拟误差的方差通常是 $\mathrm{Var}(Y_i)/N$，其中 Y_i 是重复 N 次抽样中的一次的估计，所以方差的倍数可以换算成计算量的倍数。在这个问题中，为了达到相同的精度，重要抽样法只需要平均值法的 1/9 的样本量。

平均值法的 R 程序：

```
set.seed(1)
N <- 10000
eta <- exp(runif(N))
I2 <- mean( eta )
var2 <- var(eta)
```

一次模拟的估计值和绝对误差、相对误差：

```
I2
## [1] 1.719776
abs(I2 - I.true)
## [1] 0.001493823
(I2 - I.true)/I.true
## [1] 0.0008693701
```

相对误差很小，说明有效位数约有三位。

用一次模拟的结果估计平均相对误差为：

```
MRE2 <- 0.8*sqrt(var2)/sqrt(N)/I2; MRE2
## [1] 0.002310961
```

说明估计精度大约有二位有效数字。

从一次模拟中对 $\mathrm{Var}(\hat{I}_2) * N$ 进行估计：

```
var2
## [1] 0.246802
```

理论值是 0.2420。

如果用随机投点法，$h(x) = e^x \le e (0 < x < 1)$, 取上界 $M = e$, 向 $[0,1]\times[0,M]$ 随机投点，落到 $f(x)$ 下方的概率为

$$p = I/(M(b-a)) = (e-1)/e,$$

设投 N 点落到 $h(x)$ 下方的频率为 $\hat{p}$, 用随机投点法估计 I 的公式为

$$\hat{I}_1 = \hat{p} \cdot M(b-a) = e\hat{p},$$

渐近方差为

$$\mathrm{Var}(\hat{I}_1) = e^2 p(1-p)/N = (e-1)/N \approx 1.718/N \approx 7.1 \times \mathrm{Var}(\hat{I}_2) \approx 64.8 \times \mathrm{Var}(\hat{I}_3)$$

即重要抽样法只需要随机投点法的 1/65 的样本量。可见选择合适的抽样算法对减少计算量、提高精度是十分重要的。

随机投点法的 R 程序:

```
set.seed(1)
N <- 10000
phat <- mean(runif(N) <= exp(runif(N)-1))
I1 <- phat*exp(1)
var1 <- exp(2)*phat*(1-phat)
```

一次模拟的估计值和绝对误差、相对误差：

```
I1
## [1] 1.719857
abs(I1 - I.true)
## [1] 0.001575084
(I1 - I.true)/I.true
## [1] 0.0009166624
```

相对误差很小，说明有效位数将近三位。

用一次模拟的结果估计平均相对误差为：

```
MRE1 <- 0.8*sqrt(var1)/sqrt(N)/I1; MRE1
## [1] 0.006095391
```

说明估计精度大约有二位有效数字。

从一次模拟中对 $N \cdot \mathrm{Var}(\hat{I}_1)$ 进行估计：

```
var1
## [1] 1.717148
```

理论值是 1.718。

※※※※※

例 12.2 (一个二元函数定积分 (*))**.** 设二元函数 $f(x,y)$ 定义如下

$$f(x,y) = \exp\{-45(x+0.4)^2 - 60(y-0.5)^2\} + 0.5\exp\{-90(x-0.5)^2 - 45(y+0.1)^4\}$$

求如下二重定积分

$$I = \int_{-1}^{1}\int_{-1}^{1} f(x,y)\,dx\,dy$$

$f(x,y)$ 有两个分别以 $(-0.4, 0.5)$ 和 $(0.5, -0.1)$ 为中心的峰，对积分有贡献的区域主要集中在 $(-0.4, 0.5)$ 和 $(0.5, -0.1)$ 附近，在其他地方函数值很小，对积分贡献很小。

$f(x,y)$ 写成 R 函数：

```
f <- function(x, y){
  exp(-45*(x+0.4)^2 - 60*(y-0.5)^2) +
    0.5*exp(-90*(x-0.5)^2 - 45*(y+0.1)^4)
}
```

用平均值法，取点数 $N = 10000$。

```
N <- 10000
set.seed(1)
X <- runif(N, -1, 1)
Y <- runif(N, -1, 1)
eta <- f(X, Y)
I2 <- 4*mean(eta)
sd2 <- 4*sd(eta)
```

一次模拟的估计值为：

```
I2
## [1] 0.1259771
```

估计的平均相对误差为:

```
0.8*sd2/sqrt(N)/I2
## [1] 0.02717669
```

$\hat{I}_2$ 的一个估计值为 $\hat{I}_2 = 0.126$, 从这一次模拟估计的平均相对误差为 0.03, 仅有一位有效数字精度，只能保证 $\hat{I}_2 = 0.1$ 基本可信。

平均值法的缺点是在整个正方形区域 $(-1,1)\times(-1,1)$ 上均匀布点，而被积函数 $f(x,y)$ 仅在两个峰附近有较大正值，其它区域基本是零。用重要抽样法改进。取试投密度为

$$\begin{aligned} g(x,y) \propto & \tilde{g}(x,y) \\ = & \exp\{-45(x+0.4)^2-60(y-0.5)^2\}+0.5\exp\{-90(x-0.5)^2-10(y+0.1)^2\}, \\ & -\infty<x<\infty, -\infty<y<\infty, \end{aligned}$$

这样抽取到 $[-1,1]\times[-1,1]$ 范围外的点对积分没有贡献，因为构成 $g(x,y)$ 的两个密度都很集中，所以效率损失不大。需要求使得 $\tilde{g}(x,y)$ 化为密度的比例常数。记 $\mathrm{N}(\mu,\sigma^2)$ 的分布密度为 $f(x;\mu,\sigma^2)$，对 $\tilde{g}(x,y)$ 积分得

$$\begin{aligned} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\tilde{g}(x,y) = & \sqrt{2\pi/90}\int_{-\infty}^{\infty} f(x;-0.4,90^{-1})dx\cdot\sqrt{2\pi/120}\int_{-\infty}^{\infty} f(y;0.5,120^{-1})dy \\ & +0.5\sqrt{2\pi/180}\int_{-\infty}^{\infty} f(x;0.5,180^{-1})dx\cdot\sqrt{2\pi/20}\int_{-\infty}^{\infty} f(y;-0.1,20^{-1})dy \\ = & \sqrt{2\pi/90}\sqrt{2\pi/120}+0.5\sqrt{2\pi/180}\sqrt{2\pi/20} \\ \approx & 0.1128199 \end{aligned}$$

```
c1 <- sqrt(2*pi/90*2*pi/120) + 0.5*sqrt(2*pi/180*2*pi/20); c1
## [1] 0.1128199
pmix <- sqrt(2*pi/90*2*pi/120)/c1; pmix
## [1] 0.5358984
0.5*sqrt(2*pi/180*2*pi/20)/c1
## [1] 0.4641016
```

于是令

$$\begin{aligned} g(x,y) =& \tilde{g}(x,y)/0.1128199 \\ =& 0.5358984 f(x;-0.4,90^{-1}) f(y;0.5,120^{-1}) \\ &+ 0.4641016 f(x;0.5,180^{-1}) f(y;-0.1,20^{-1}), \\ &-\infty < x < \infty, -\infty < y < \infty, \end{aligned}$$

用复合抽样法对 $g(x,y)$ 抽样，然后用重要抽样法得到估计值 $\hat{I}_3$。注意用重要抽样法计算时需要 X_i 独立同分布，但是其次序并不重要，当 N 很大时，可以固定取混合分布中两个分布的样本个数严格等于混合比例，而不需要从两个分布中随机抽取。

```
N <- 10000
g <- function(x, y){
   0.5358984*dnorm(x,-0.4,sqrt(1/90))*dnorm(y,0.5,sqrt(1/120)) +
   0.4641016*dnorm(x,0.5,sqrt(1/180))*dnorm(y,-0.1,sqrt(1/20))
}
set.seed(1)
N1 <- round(N*pmix)
N2 <- N-N1
X <- c(rnorm(N1, -0.4, sqrt(1/90)), rnorm(N2, 0.5, sqrt(1/180)))
Y <- c(rnorm(N1, 0.5, sqrt(1/120)), rnorm(N2, -0.1, sqrt(1/20)))
eta <- ifelse(-1 < X & X < 1 & -1 < Y & Y < 1, f(X, Y)/g(X,Y), 0)
I3 <- mean(eta)
sd3 <- sd(eta)
```

一次模拟的估计值为：

```
I3
## [1] 0.1253334
```

估计的平均相对误差为:

```
0.8*sd3/sqrt(N)/I3
## [1] 0.001697251
```

$N = 10000$ 时一次模拟得到的 $\hat{I}_3 = 0.1253$, 从这一次模拟估计的平均相对误差为 0.002, 估计精度有两位有效数字以上，估计结果可报告为 0.13。

重要抽样法与平均值法的方差相比：

```
(sd2/sd3)^2
## [1] 259.0305
```

两种方法的方差差距有 200 倍以上，说明重要抽样法节省了 200 倍的计算量。

※※※※※

12.1 标准化重要抽样法

有时 $Y \sim f(\cdot)$ 不仅很难直接抽样，而且 $f(\cdot)$ 本身未知，只能确定到差一个常数倍的 $\tilde{f}(x) = cf(x)$, 常数 c 未知，为了求常数 c 需要计算 $c = \int \tilde{f}(y)\,dy$，计算 c 一般很困难。这时，定义重要性权重为 $W_i = \frac{\tilde{f}(X_i)}{g(X_i)}$, 公式(12.6)可以改成

$$\hat{I}_4 = \frac{\sum_{i=1}^N W_i h(X_i)}{\sum_{i=1}^N W_i} \tag{12.8}$$

这称为**标准化重要抽样法**。对(12.8)的分子和分母都除以 cN 后分子 a.s. 收敛到 $Eh(Y)$, 分母 a.s. 收敛到 1, 所以(12.8)是 $Eh(Y)$ 的强相合估计，但不是无偏的。标准化重要抽样估计往往比无偏估计 $\hat{I}_{3.1}$ 有更小的均方误差。关于标准化重要抽样法的渐近方差的讨论参见 [Liu, 2001] §2.5.3。

如果需要对多个不同的函数 $h(\cdot)$ 计算 $Eh(Y)$, 则选取试抽样密度 $g(x)$ 时应使得 $g(x)$ 尽可能与 Y 的密度 $f(x)$ 形状接近，这样权重 $W_i = \frac{\tilde{f}(X_i)}{g(X_i)}$ 的分布不过于偏斜，不至于出现绝大部分权重集中于少数样本点的情形。抽样值 X_i 与权重 W_i 一起可以看作是分布 $f(\cdot)$ 的某种抽样。

定义 12.1 (适当加权抽样)**.** 随机变量序列 $\{(X_i, W_i),\, i = 1, 2, \ldots, N\}$ 称为关于密度 $f(\cdot)$ 的**适当加权抽样**, 如果对于任何平方可积函数 $h(\cdot)$ 都有

$$E[h(X_i)W_i] = cE_f[h(X)] = c\int h(x)f(x)\,dx,\ \ i = 1, 2, \ldots, N,$$

其中 c 是归一化常数。

设随机变量 (X,W) 联合密度为 $g(x,w)$，则 (X,W) 的样本为密度 $f(\cdot)$ 的适当加权抽样的充分必要条件是

$$E_g(W|x) = E_g(W)\frac{f(x)}{g(x)},\ \forall x,$$

其中 $E_g(W)$ 是关于 W 的边缘密度的期望，$E_g(W|x)$ 是在 (X,W) 的联合密度下条件期望 $E(W|X)$ 在 $X=x$ 处的值。

例 12.3 (贝塔—二项分布推断 (*))**.** 标准化的重要抽样法在贝叶斯统计推断中有重要作用。例如，设独立的观测样本 Y_j 服从如下的贝塔—二项分布：

$$f(y_j|K,\eta) = P(Y_j = y_j) \tag{12.9}$$

$$= \binom{n_j}{y_j}\frac{B(K\eta+y_j,\ K(1-\eta)+n_j-y_j)}{B(K\eta,\ K(1-\eta))},\ y_j = 0,1,\dots,n_j, \tag{12.10}$$

其中 $B(\cdot,\cdot)$ 是贝塔函数，n_j 为已知的正整数，$K>0$ 和 $0<\eta<1$ 为未知参数。贝塔—二项分布用于描述比二项分布更为分散的随机变量分布。按照贝叶斯统计的做法，假设参数 (K,η) 也是随机变量，具有所谓的 “先验分布”，假设 (K,η) 有如下的 “无信息” 先验分布密度:

$$\pi(K,\eta) \propto \frac{1}{(1+K)^2}\frac{1}{\eta(1-\eta)}, \tag{12.11}$$

则 (K,η) 有如下的 “后验密度”:

$$\begin{aligned}\tilde{p}(K,\eta|Y) \propto& \pi(K,\eta)\prod_{j=1}^{n} f(y_j|K,\eta)\\ \propto& \frac{1}{(1+K)^2}\frac{1}{\eta(1-\eta)}\prod_{j=1}^{n}\frac{B(K\eta+y_j,\ K(1-\eta)+n_j-y_j)}{B(K\eta,\ K(1-\eta))}.\end{aligned} \tag{12.12}$$

要求 $E(\log K|Y) = \int_0^\infty \log K\,\tilde{p}(K,\eta|Y)\,dK$ 的值。

如果可以从后验密度 $\tilde{p}(K,\eta|Y)$ 直接抽样，可以用平均值法估计 $E(\log K|Y)$，但从(12.12)来看很难直接抽样。为此，使用标准化的重要抽样法。为了解除 (K,η) 的取值限制，作变换 $\alpha = \log K$, $\beta = \log\frac{\eta}{1-\eta}$, 则 $\alpha,\beta \in (-\infty,\infty)$,

而(12.12)对应的 (α,β) 的后验密度为:

$$p(\alpha,\beta|Y)\propto\frac{e^{\alpha}}{(1+e^{\alpha})^2}\prod_{j=1}^{n}\frac{B(\frac{e^{\alpha}}{1+e^{-\beta}}+y_j,\ \frac{e^{\alpha}}{1+e^{\beta}}+n_j-y_j)}{B(\frac{e^{\alpha}}{1+e^{-\beta}},\ \frac{e^{\alpha}}{1+e^{\beta}})}. \tag{12.13}$$

取值无限制的随机变量试抽样密度经常使用自由度较小的 t 分布，比如 t(4) 分布，设 t(4) 分布密度函数为 $g(\cdot)$, 用独立的 t(4) 分布生成 (α,β) 的试抽样样本 $(\alpha_i,\beta_i), i=1,2,\ldots,N$，可以估计 $E(\log K|Y)$ 为

$$\hat{\alpha}=\frac{\sum_{i=1}^{N}\alpha_i\frac{p(\alpha_i,\beta_i|Y)}{g(\alpha_i)g(\beta_i)}}{\sum_{i=1}^{N}\frac{p(\alpha_i,\beta_i|Y)}{g(\alpha_i)g(\beta_i)}}.$$

其中的 $p(\alpha_i,\beta_i|Y)$ 只要用(12.13)的右侧计算，因为分子和分母的归一化常数可以消掉。

下面用 R 语言产生一组模拟的观测数据 Y，然后对这组数据估计 $E(\log K|Y)$。

先定义贝塔-二项分布的概率质量函数:

```
dbetabin <- function(y, n, K, eta){
  choose(n, y)*beta(K*eta + y, K*(1-eta) + n-y)/beta(K*eta, K*(1-eta))
}
```

其中 n 是正整数，y 是非负整数且 $0\le y\le n$，`K` 和 `eta` 是上述分布参数 K 和 η。

给定 n 个独立观测，设这组观测的 $n_1,\ldots,n_n$ 保存在向量 `narr` 中，$y_1,\ldots,y_n$ 保存在向量 `yarr` 中，于是以 (α,β) 为自变量的后验密度函数（差一个未知的常数倍），将 α 和 β 记为自变量 `a, b`，R 函数为：

```
p.post <- function(a, b){
  K <- exp(a)
  eta <- 1/(1 + exp(-b))
  val <- K / (1+K)^2
  beta0 <- beta(K*eta, K*(1-eta))
  for(j in seq(n)){
    val <- val * beta(K*eta + yarr[j], K*(1-eta) + narr[j] - yarr[j]) / beta0
  }
```

```
  val[beta0==0] <- 0
  val
}
```

注意其中的 narr 和 yarr 是还没有赋值的全局变量。在 R 的函数定义中，允许使用尚未定义的变量，只要调用该函数时变量已经在该函数定义的环境中赋值就可以。

下面用模拟方法生成观测样本，然后认为观测样本已知。

```
set.seed(1)
K.true <- 10
eta.true <- 0.2
narr <- c(20, 30, 25, 30, 40, 20, 50, 30, 20, 20)
n <- length(narr)
yarr <- numeric(n)
for(j in seq(n)){
  parr <- dbetabin(seq(0, narr[j]), narr[j], K.true, eta.true)
  yarr[j] <- sample(seq(0, narr[j]), size=1, prob=parr, replace=TRUE)
}
rbind(narr, yarr)
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## narr   20   30   25   30   40   20   50   30   20    20
## yarr    3    2    6   12    6    8   22    8    5     2
```

将这些模拟生成的数据固定下来。

```
narr <- c(20, 30, 25, 30, 40, 20, 50, 30, 20, 20)
yarr <- c( 3,  2,  6, 12,  6,  8, 22,  8,  5,  2)
```

下面用标准化的重要抽样法估计 $E(\log K|Y) = E(\alpha|Y)$。取 α 和 β 的试抽样分布为 t(4)。

```
set.seed(1)
N <- 10000
df.t <- 4
alpha.samp <- rt(N, df.t)
beta.samp <- rt(N, df.t)
wei <- p.post(alpha.samp, beta.samp) / dt(alpha.samp, df.t) / dt(beta.samp, df.t)
est <- sum(alpha.samp*wei) / sum(wei)
```

$\log(K)$ 真值为：

```
log(K.true)
## [1] 2.302585
```

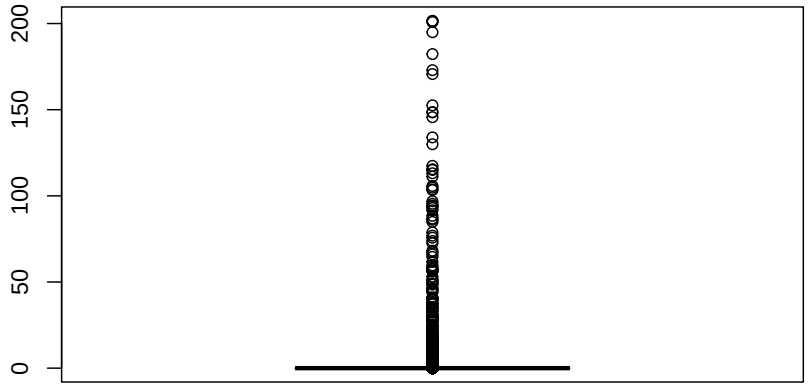
用 10000 个抽样的重要抽样法给出的 $\log(K)$ 的后验估计为：

```
est
## [1] 2.324758
```

重要性权重最好取值比较均匀，否则重要性权重很小的抽样点基本不起作用，效率较低。上述模拟的归一化重要性权重的分布情况：

```
wei <- wei/sum(wei)*N
summary(wei)
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##   0.00000   0.00000   0.00005   1.00000   0.00470 201.56573
```

```
boxplot(wei)
```



可见绝大多数抽样点都贡献很小。这也是求解比较复杂的问题时重要抽样法应用的普遍现象。

舍选控制是解决这样问题的一种技术，下面尝试采用这一技术。先看权重的分为点：

```
quantile(wei, c(0.5, 0.75, 0.8, 0.85, 0.9, 0.95, 0.99))
##          50%          75%          80%          85%          90%          95%
## 5.187513e-05 4.696631e-03 1.314482e-02 4.262636e-02 1.663021e-01 1.244243e+00
##          99%
## 2.447960e+01
```

取舍选控制阈值 c 为 90% 分位数。

```
wei.cut <- unname(quantile(wei, 0.90))
```

对 N 个权重，超过阈值的都保留；小于阈值的，以概率 W_i/c 保留，c 是阈值，其它的丢弃，这样减少了样本量。原来的舍选控制法是预先选好阈值 c，对每个 $i = 1, 2, \ldots, N$ 的每个，从 $g(x)$ 中抽取后 X_i 后，都进行舍选控制；如果第 i

个被丢弃，就重新从 $g()$ 中抽取，直到被接受。在 R 的向量化做法中，可以多抽取一些，然后丢弃的就不再重复抽取。

舍选控制：

```
sele <- wei < wei.cut; n.sele <- sum(sele)
i.over <- seq(N)[!sele] # 超过阈值的下标，保留
i.under <- seq(N)[sele] # 低于阈值的下标，按概率保留
i.accept <- i.under[runif(n.sele) < wei[sele] / wei.cut] # 选出按概率保留部分
i.keep <- c(i.over, i.accept)
N1 <- length(i.keep); N1
## [1] 1371
```

原来的 10000 个抽样仅保留了 1000 多个。权重修改为：

```
wei.new <- pmax(wei[i.keep], wei.cut)
summary(wei.new)
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##   0.1663   0.1663   0.5231   7.2917   3.2660 201.5657
```

在新的权重下保留的抽样是关于后验密度适当加权的。新的估计为:

```
est.new <- sum(alpha.samp[i.keep]*wei.new)/sum(wei.new)
est.new
## [1] 2.325295
```

※※※※※

12.2 带有舍选控制的重要抽样法 (*)

在重要抽样法和标准化重要抽样法的实际应用中，好的试抽样分布很难获得，所以权重 $\{W_i = f(X_i)/g(X_i)\}$ 经常会差别很大，使得抽样样本主要集中在少数几个权重最大的样本点上。为此，可以舍弃权重太小的样本点，重新抽样替换

这样的样本点, 这种方法称为**带有舍选控制的重要抽样法**。需要预先选定权重的一个阈值 c，并计算

$$p_c = \int \min\left\{1, \frac{f(x)}{cg(x)}\right\} g(x)\,dx = \int \min\left\{g(x), \frac{f(x)}{c}\right\} dx.$$

在产生每个抽样点 X_i 时，计算权重 W_i，当权重 $W_i \geq c$ 时接受此抽样点，但权重改为 $W_i^* = p_c W_i$；当 $W_i < c$ 时仅以 W_i/c 的概率接受此抽样点，并调整权重为 $p_c c$，如果没有被接受则重新从 $g(\cdot)$ 中抽取。

算法如下：

带有舍选控制的重要抽样法

---

for(i **in** $1:N$) {
  repeat {
    从 $g(\cdot)$ 抽取 ξ_i
    计算 $W_i \leftarrow f(\xi_i)/g(\xi_i)$
    if($W_i \geq c$) {
      令 $X_i \leftarrow \xi_i$, $W_i^* \leftarrow p_c W_i$
      break
    } **else** {
      从 U(0,1) 抽取 U
      if($U < W_i/c$) {
        令 $X_i \leftarrow \xi_i$, $W_i^* \leftarrow p_c c$
        break
      }
    }
  } # repeat
} # for
输出 (X_i^*, W_i^*), $i = 1, 2, \ldots, N$

在实际应用带有舍选控制的重要抽样法时，一般先选略小的抽样量 N_0 进行原始的重要抽样，分析权重 W_i 的分布，据此选择权重 c，比如可以选 $\{W_i\}$ 的某个分位数。舍选控制算法中的 p_c 是归一化常数，如果使用标准化重要抽样法，

p_c 可以省略，否则，p_c 可以从原始的重要抽样法结果中估计为

$$\widehat{p}_c = \frac{1}{N_0}\sum_{i=1}^{N_0} \min\left\{1,\ \frac{W_i}{c}\right\}. \tag{12.14}$$

带有舍选控制的重要抽样法得到的 $\{(X_i, W_i^*),\ i = 1, \ldots, N\}$ 是关于 $f(\cdot)$ 适当加权的, 被接受的 X_i^* 的分布密度不再是试投密度 $g(x)$，而是改成了

$$g^*(x) = \frac{1}{p_c}\min\left\{g(x), \frac{f(x)}{c}\right\}. \tag{12.15}$$

习题

习题 1

设 $X \sim \mathrm{N}(0,1)$, $h(x) = \exp(-\frac{1}{2}(x-3)^2) + \exp(-\frac{1}{2}(x-6)^2)$。令 $I = Eh(X)$。

(1) 推导 I 的精确表达式并计算结果。
(2) 用 $N = 1000$ 次函数计算的平均值法估计 I 并估计误差大小。
(3) 设计适当的重要抽样方法取 $N = 1000$ 估计 I 并估计误差大小。

习题 2

设 $X \sim \mathrm{N}(0,1)$, 则 $\theta = P(X > 4.5) = 3.398 \times 10^{-6}$。

(1) 如果直接生成 N 个 X 的随机数，用 $X_i > 4.5$ 的比例估计 $P(X > 4.5)$，平均多少个样本点中才能有一个样本点满足 $X_i > 4.5$?
(2) 取 V 为指数分布 $\mathrm{Exp}(1)$, 令 $W = V + 4.5$，用 W 的样本进行重要抽样估计 θ, 取样本点个数 $N = 1000$，求估计值并估计误差大小。

习题 3

在例12.3中，设 $n = 10$, 已知 (n_j, y_j) 的值为:

n_j	20	30	25	30	40	20	50	30	20	20
y_j	6	1	1	0	5	4	1	8	4	7

编写 R 程序估计 $E(\log K | y_1, y_2, \ldots, y_n)$。

Chapter 13

分层抽样法

用平均值法计算 $\int_C h(x)\,dx$，若 $h(x)$ 在 C 内取值变化范围大则估计方差较大。重要抽样法选取了与 $f(x)$ 形状相似但是容易抽样的密度 $g(x)$ 作为试投密度，大大提高了精度，但是这样的 $g(x)$ 有时难以找到。

如果把 C 上的积分分解为若干个子集上的积分，使得 $h(x)$ 在每个子集上变化不大，分别计算各个子集上的积分再求和，可以提高估计精度。这种方法叫做**分层抽样法**。这也是抽样调查中的重要技术。

例 13.1. 对函数

$$h(x) = \begin{cases} 1 + \frac{x}{10}, & 0 \le x \le 0.5 \\ -1 + \frac{x}{10}, & 0.5 < x \le 1 \end{cases}$$

求定积分

$$I = \int_0^1 h(x)\,dx,$$

可以得 I 的精确值为 $I = 0.05$。我们用平均值法和分层抽样法来估计 I 并比较精度。

在 $(0,1)$ 区间随机抽取 N 点用平均值法得 $\hat{I}_2$，其渐近方差为

$$\mathrm{Var}(\hat{I}_2) = \frac{\mathrm{Var}(h(U))}{N} = \frac{143}{150N} \approx \frac{0.9533}{N}.$$

模拟的 R 程序:

```
h <- function(x) ifelse(x <= 0.5, 1 + 0.1*x, -1 + 0.1*x)
I.true <- 0.05
set.seed(1)
N <- 10000
eta <- h(runif(N))
I2 <- mean(eta)
sd2 <- sd(eta)
```

取 $N = 10000$，一次平均值法得到的估计和相对误差为

```
I2
## [1] 0.0594168
(I2 - I.true)/I.true
## [1] 0.1883359
```

相对误差为 0.2，仅有一位有效数字。估计值可报告为 0.06。

估计的 $N\mathrm{Var}(\hat{I}_2)$ 为

```
sd2^2
## [1] 0.9504117
```

理论值为 0.9533。

把 I 拆分为 $[0, 0.5]$ 和 $[0.5, 1]$ 上的积分，即

$$I = a + b = \int_0^{0.5} h(x)\,dx + \int_{0.5}^{1} h(x)\,dx,$$

对 a 和 b 分别用平均值法，得

$$\begin{aligned}
\hat{a} =& \frac{0.5}{N/2}\sum_{i=1}^{N/2} h(0.5U_i) = \frac{0.5}{N/2}\sum_{i=1}^{N/2}(1 + 0.05U_i),\\
\hat{b} =& \frac{0.5}{N/2}\sum_{i=(N/2)+1}^{N} h(0.5 + 0.5U_i) = \frac{0.5}{N/2}\sum_{i=(N/2)+1}^{N}(-1 + 0.05 + 0.05U_i),\\
\hat{I}_5 =& \hat{a} + \hat{b},
\end{aligned}$$

则分层抽样法结果 $\hat{I}_5$ 的渐近方差为

$$\begin{aligned}\mathrm{Var}(\hat{I}_5) =& \mathrm{Var}(\hat{a}+\hat{b}) = \mathrm{Var}(\hat{a}) + \mathrm{Var}(\hat{b}) \\ =& 0.25\frac{\mathrm{Var}(1+0.05U)}{N/2} + 0.25\frac{\mathrm{Var}(-0.95+0.05U)}{N/2} = \frac{1/4800}{N},\end{aligned}$$

分层后的估计方差远小于不分层的结果, 可以节省样本量约 4500 倍。

分层抽样法的一次模拟计算的 R 程序如下：

```
N <- 10000
set.seed(1)
eta1 <- h(runif(N/2, 0, 0.5))
a <- 0.5*mean(eta1)
eta2 <- h(runif(N/2, 0.5, 1))
b <- 0.5*mean(eta2)
I5 <- a+b
```

一次模拟的估计为：

```
I5
## [1] 0.0500084
```

$N\mathrm{Var}(\hat{I}_5)$ 的估计为

```
Nvar <- 0.5*var(eta1) + 0.5*var(eta2); Nvar
## [1] 0.0002117651
```

理论值是 0.002083。

※※※※※

一般地，设积分 $I = \int_C h(x)\,dx$ 可以分解为 m 个不交的子集 C_j 上的积分, 即

$$I = \int_C h(x)\,dx = \int_{C_1} h(x)\,dx + \int_{C_2} h(x)\,dx + \cdots + \int_{C_m} h(x)\,dx$$

在 C_j 投 n_j 个随机点 $X_{ji} \sim \mathrm{U}(C_j)$, $i = 1, \ldots, n_j$, 则 I 的 m 个部分可以分别用平均值法估计，由此得 I 的分层估计为

$$\hat{I}_5 = \sum_{j=1}^{m} \frac{V(C_j)}{n_j} \sum_{i=1}^{n_j} h(X_{ji})$$

记 $\sigma_j^2 = \mathrm{Var}(h(X_{j1}))$, 划分子集时应使每一子集内 $h(\cdot)$ 变化不大，即 σ_j^2 较小。这时

$$\mathrm{Var}(\hat{I}_5) = \sum_{j=1}^{m} \frac{V^2(C_j)\sigma_j^2}{n_j}$$

若 σ_j^2 可估计，应取 n_j 使

$$n_j \propto V(C_j)\sigma_j, \tag{13.1}$$

即

$$n_j = N\frac{V(C_j)\sigma_j}{\sum_{k=1}^{m} V(C_k)\sigma_k},\ j = 1, 2, \ldots, m$$

这样取的样本量 $(n_1, n_2, \ldots, n_m)$ 在所有满足 $n_1 + n_2 + \cdots + n_m = N$ 的取法中使得渐近方差最小。

在分层抽样法中，划分了子集后，每一子集上的积分也可用重要抽样法计算。

分层抽样法也可以用在求随机变量函数期望的问题中。设 X 为随机变量，要求 X 的函数 $h(X)$ 的数学期望 $\theta = Eh(X)$。假设存在离散型随机变量 Y，$p_j = P(Y = y_j), j = 1, 2, \ldots, m$, 在 $Y = y_j$ 条件下可以从 X 的条件分布抽样，则

$$E[h(X)] = E\{E[h(X)|Y]\} = \sum_{j=1}^{m} E[h(X)|Y = y_j]p_j, \tag{13.2}$$

如果在 $Y = y_j$ 条件下生成 X 的 $N_j = Np_j$ 个抽样值，设为 $X_i^{(j)}, i = 1, 2, \ldots, N_j$, 则可以用 $\frac{1}{N_j}\sum_{i=1}^{N_j} h(X_i^{(j)})$ 估计 $E[h(X)|Y = y_j]$，估计 θ 为

$$\hat{\theta} = \sum_{j=1}^{m} \frac{1}{N_j} \sum_{i=1}^{N_j} h(X_i^{(j)})p_j = \frac{1}{N} \sum_{j=1}^{m} \sum_{i=1}^{Np_j} h(X_i^{(j)}), \tag{13.3}$$

这是 θ 的无偏和强相合估计，且估计方差

$$\begin{aligned}
\mathrm{Var}(\hat{\theta}) =& \frac{1}{N^2} \sum_{j=1}^{m} Np_j \mathrm{Var}[h(X)|Y = y_j] \\
=& \frac{1}{N} \sum_{j=1}^{m} \mathrm{Var}[h(X)|Y = y_j]p_j \qquad (13.4) \\
=& \frac{1}{N} E\{\mathrm{Var}[h(X)|Y]\} \leq \frac{1}{N}\mathrm{Var}[h(X)], \qquad (13.5)
\end{aligned}$$

比直接用平均值法估计 $Eh(X)$ 的方差小。这里用到了条件方差的性质

$$\mathrm{Var}(X) = E\left[\mathrm{Var}(X|Y)\right] + \mathrm{Var}\left[E(X|Y)\right] \geq E\left[\mathrm{Var}(X|Y)\right], \tag{13.6}$$

如果 Y 与 X 独立则 $E(X|Y) = EX$, $\mathrm{Var}\left[E(X|Y)\right] = 0$, 这时分层抽样法比平均值法没有改进。从(13.4)可以看出，如果第 j 层样本的函数 $h(X_i^{(j)}), i = 1, 2, \ldots, Np_j$ 的样本方差为 S_j^2, 则 $\mathrm{Var}(\hat{\theta})$ 的一个无偏估计是

$$\widehat{\mathrm{Var}(\hat{\theta})} = \frac{1}{N}\sum_{j=1}^{m} S_j^2 p_j.$$

公式(13.3)取第 j 层样本数 $N_j = Np_j$，仅考虑了 Y 的取值分布，而未考虑 $X|Y = y_j$ 的条件分布情况。类似于(13.1)，应该对 $\mathrm{Var}[h(X)|Y = y_j]$ 较大的层取更多的样本。使得估计方差最小的分层样本量分配满足 $N_j \propto p_j\sqrt{\mathrm{Var}[h(X)|Y = y_j]}$, 即

$$N_j = N\frac{p_j\sqrt{\mathrm{Var}[h(X)|Y = y_j]}}{\sum_{k=1}^{m} p_k\sqrt{\mathrm{Var}[h(X)|Y = y_k]}}. \tag{13.7}$$

在 $\mathrm{Var}[h(X)|Y = y_j]$ 未知的时候，可以预先抽取一个小的样本估计 $\mathrm{Var}[h(X)|Y = y_j]$, 然后按估计的最优 N_j 分配各层的样本量。采用(13.7)的分层样本量后，

$$\begin{aligned}
\hat{\theta} &= \sum_{j=1}^{m}\frac{1}{N_j}\sum_{i=1}^{N_j} h(X_i^{(j)})p_j, \\
\mathrm{Var}(\hat{\theta}) &= \sum_{j=1}^{m}\frac{p_j^2\mathrm{Var}[h(X)|Y = y_j]}{N_j},
\end{aligned}$$

于是 $\mathrm{Var}(\hat{\theta})$ 的估计为

$$\widehat{\mathrm{Var}(\hat{\theta})} = \sum_{j=1}^{m}\frac{p_j^2 S_j^2}{N_j},$$

其中 S_j^2 是第 j 层样本函数 $\{h(X_i^{(j)}), i = 1, 2, \ldots, N_j\}$ 的样本方差。

分层抽样法的本质是把 X 的值相近的抽样分入一层，使得同层的 X 条件方差较小，从而减小估计方差。

例 13.2. 设 $U \sim \mathrm{U}(0,1)$, 用分层抽样法估计 $\theta = Eh(U) = \int_0^1 h(x)\,dx$。

令 $Y = \text{ceiling}(mU)$, 即当且仅当 $\frac{j-1}{m} < U \leq \frac{j}{m}$ 时 $Y = j$，$j = 1, 2, \ldots, m$, 可以按照 Y 分层抽样估计 θ:

$$\begin{aligned}\theta =& E[h(U)] = \sum_{j=1}^{m} E[h(U)|Y = j]\, P(Y = j) \\ =& \frac{1}{m} \sum_{j=1}^{m} E[h(U)|Y = j],\end{aligned}$$

易见 $Y = j$ 条件下 U 服从 $(\frac{j-1}{m}, \frac{j}{m})$ 上的均匀分布，设 $U_1, U_2, \ldots, U_n$ 是 U(0,1) 的独立抽样，则用分层抽样法取每层 $N_j = 1$ 估计 $\theta = Eh(U)$ 为

$$\hat{\theta} = \frac{1}{m} \sum_{j=1}^{m} h\left(\frac{j - 1 + U_j}{m}\right).$$

※※※※※

习题

习题 1

设 $\sigma_j, j = 1, 2, \ldots, m$ 为 m 个正实数,

$$f(\alpha_1, \alpha_2, \ldots, \alpha_m) = \frac{\sigma_1^2}{\alpha_1} + \frac{\sigma_2^2}{\alpha_2} + \cdots + \frac{\sigma_m^2}{\alpha_m},\ (\alpha_1, \ldots, \alpha_m) \in (0, 1)^m,$$

则在 $\alpha_1 + \alpha_2 + \cdots + \alpha_m = 1$ 条件下 $f(\alpha_1, \alpha_2, \ldots, \alpha_m)$ 的最小值点为

$$\alpha_j = \frac{\sigma_j}{\sum_{k=1}^{m} \sigma_k},\ j = 1, 2, \ldots, m$$

最小值为 $(\sigma_1 + \cdots + \sigma_m)^2$。

习题 2

考虑定积分

$$I = \int_{-1}^{1} e^x dx = e - e^{-1}.$$

(1) 用随机模拟方法计算定积分 I, 分别用随机投点法、平均值法、重要抽样法和分层抽样法计算。

(2) 设估计结果为 $\hat{I}$, 如果需要以 95% 置信度保证计算结果精度在小数点后三位小数，这四种方法分别需要计算多少次被积函数值？

(3) 用不同的随机数种子重复以上的估计 B 次，得到 $\hat{I}_j, j = 1, 2, \ldots, B$, 由此估计 $\hat{I}$ 的抽样分布方差，与 2 的结果进行验证。

(4) 称

$$\mathrm{MAE}(\hat{I}) = E|\hat{I} - I|$$

为 $\hat{I}$ 的平均绝对误差。从 3 得到的 $\hat{I}_j, j = 1, 2, \ldots, B$ 中估计 $\mathrm{MAE}(\hat{I})$。比较这四种积分方法的平均绝对误差大小。

习题 3

设 $h(x) = \frac{e^{-x}}{1+x^2}$, $x \in (0, 1)$, 用重要抽样法计算积分 $I = \int_0^1 h(x)\,dx$，分别采用如下的试抽样密度:

$$\begin{aligned}
f_1(x) =& 1,\ x \in (0, 1),\\
f_2(x) =& e^{-x},\ x \in (0, \infty),\\
f_3(x) =& \frac{1}{\pi(1 + x^2)},\ x \in (-\infty, \infty),\\
f_4(x) =& (1 - e^{-1})^{-1} e^{-x},\ x \in (0, 1),\\
f_5(x) =& \frac{4}{\pi(1 + x^2)},\ x \in (0, 1).
\end{aligned}$$

(1) 作 $h(x)$ 和各试抽样密度的图形，比较其形状。

(2) 取样本点个数 $N = 10000$, 分别给出对应于不同试抽样密度的估计 $\hat{I}_k$, $k = 1, 2, 3, 4, 5$, 以及 $\mathrm{Var}(\hat{I}_k)$ 的估计。

(3) 分析 $\mathrm{Var}(\hat{I}_k)$ 的大小差别的原因。

(4) 把 $(0, 1)$ 区间均分为 10 段，在每一段内取 $N = 1000$ 个样本点用平均值法计算积分值，把各段的估计求和得到 I 的估计 $\hat{I}_6$，估计其方差。

(5) 用例13.2的分层抽样方法计算积分的估计 $\hat{I}_7$，估计 $\mathrm{Var}(\hat{I}_7)$ 并与前面的结果进行比较。

Chapter 14

方差缩减技术

随机模拟方法虽然有着适用性广、方法简单的优点，但是又有精度低、计算量大的缺点, 一整套模拟算几天几夜也是常有的事情。如果能成倍地减小随机模拟误差方差，就可以有效地节省随机模拟时间，有些情况下可以把耗时长到不具有可行性的模拟计算（比如几个月）缩短到可行（比如几天）。

前一节的关于定积分计算的重要抽样法、分层抽样法都是降低随机模拟误差方差的重要方法，也可以用在一般的模拟问题中。本节介绍一些其它的方差缩减技巧。我们以随机变量 X 的期望 $\theta = EX$ 的估计为例，目标是降低 θ 的估计量的渐近方差。

14.1 控制变量法

设要估计随机变量 X 的期望 $\theta = EX$, 从 X 中抽取 N 个独立样本值 $X_1, X_2, \ldots, X_n$，用样本平均值 $\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$ 估计 EX。为了提高精度可以利用辅助信息。设有另外的随机变量 Y 满足

$$EY = 0, \quad \mathrm{Cov}(X, Y) < 0$$

令 $Z = X + Y$, 则

$$E(Z) = \theta, \quad \mathrm{Var}(Z) = \mathrm{Var}(X) + \mathrm{Var}(Y) + 2\mathrm{Cov}(X, Y),$$

只要 $\mathrm{Var}(Y)+2\mathrm{Cov}(X,Y)<0$ 则 $\mathrm{Var}(Z)<\mathrm{Var}(X)$，如果有 (X,Y) 成对的抽样 $(X_i,Y_i), i=1,2,\ldots,n$, 令 $Z_i=X_i+Y_i$, 则用 $\bar{Z}$ 来估计 $\theta=EX=EZ$ 的渐近方差就比用 $\bar{X}$ 估计 I 的渐近方差减小了。

为了最好地利用 Y 与 X 的相关性，令

$$Z(b)=X+bY,$$

则

$$\begin{aligned} EZ(b) =& EX=\theta, \\ \mathrm{Var}(Z(b)) =& \mathrm{Var}(X)+2b\mathrm{Cov}(X,Y)+b^2\mathrm{Var}(Y), \end{aligned}$$

求 $\mathrm{Var}(Z(b))$ 关于 b 的最小值点，得

$$b=-\mathrm{Cov}(X,Y)/\mathrm{Var}(Y)=-\rho_{X,Y}\sqrt{\mathrm{Var}(X)/\mathrm{Var}(Y)},$$

这时

$$\mathrm{Var}(Z(b))=(1-\rho_{X,Y}^2)\mathrm{Var}(X)\le\mathrm{Var}(X),$$

可见只要能找到零均值随机变量 Y 使得 $\rho_{X,Y}\neq 0$ 就可以减小 EX 的估计方差。Y 和 X 的相关性越强，改善幅度越大。这种减小随机模拟误差方差的方法叫做**控制变量法**。

实际中 $\rho_{X,Y}$ 和 $\mathrm{Var}(X),\mathrm{Var}(Y)$ 可能是未知的，可以先模拟一个小的样本估计 $\rho_{X,Y}$ 和 $\mathrm{Var}(X),\mathrm{Var}(Y)$ 从而获得 b 的估计值。

控制变量法中要求控制变量 Y 与 X 相关且 $EY=0$。如果 $EY\neq 0$ 但 $EY=\mu_Y$ 已知，只要用 $Y-\mu_Y$ 代替 Y，这需要能预先知道 Y 的期望值的真值。另一种情况是 $EY=EX$ 未知，Y 与 X 相关，这时令 $Z=\alpha X+(1-\alpha)Y$, $\alpha\in[0,1]$, 则 $EZ=EX=\theta$，可以求 α 使得 $\mathrm{Var}(Z)$ 最小。容易知道当 $\alpha=\mathrm{Cov}(Y,Y-X)/\mathrm{Var}(Y-X)$ 时 $\mathrm{Var}(Z)$ 最小。

例 14.1. 设要估计 $I=\int_0^1 e^t\,dt$。当然，可以得到积分真值为 $e-1$，这里用来演示控制变量法的优势。

设 $U\sim\mathrm{U}(0,1)$，$X=e^U$, 则 $I=Ee^U=EX$, 可以用平均值法估计 I 为

$$\hat{I}_1=\frac{1}{N}\sum_{i=1}^{N}e^{U_i}.$$

其方差为

$$\mathrm{Var}(\hat{I}_1)=\frac{1}{N}\mathrm{Var}(e^U)=\frac{1}{N}(-\frac{1}{2}e^2+2e-\frac{3}{2})\approx\frac{0.2420}{N}.$$

令 $Y=U-\frac{1}{2}$, 则 $EY=0$, X 与 Y 正相关, 可以计算出 $\mathrm{Cov}(X,Y)\approx 0.14086$, $\mathrm{Var}(Y)=1/12$(更复杂的问题中可能需要从一个小的随机抽样中近似估计), 于是 $b=-\mathrm{Cov}(X,Y)/\mathrm{Var}(Y)=-1.690$, 对 $Z(b)=e^U-1.690(U-\frac{1}{2})$ 有

$$\mathrm{Var}(Z(b))=[1-\rho^2_{X,Y}]\mathrm{Var}(X)=(1-0.9919^2)\mathrm{Var}(X)=0.016\mathrm{Var}(X)=0.0039,$$

用控制变量法估计 I 为

$$\hat{I}_2=\frac{1}{N}\sum_{i=1}^{N}\left[e^{U_i}-1.690(U_i-\frac{1}{2})\right].$$

$\hat{I}_1$ 的方差比控制变量法 $\hat{I}_2$ 的方差大 60 倍以上。

※※※※※

例 14.2 (系统可靠性估计 (*))**.** 考虑由 n 个部件组成的一个系统，用 S_i 表示第 i 个部件是否正常工作，1 表示正常工作，0 表示失效。设 $S_i\sim \mathrm{B}(1,p_i)$ 且各 S_i 相互独立。用 Y 表示系统是否工作正常，1 表示工作正常，0 表示系统失效。设 Y 为 $S_1,S_2,\dots,S_n$ 的函数 $\phi(S_1,S_2,\dots,S_n)$ 且 ϕ 关于每个 S_i 是单调不减，称 ϕ 为系统的结构函数。令 $R=P(Y=1)=E\phi(S_1,S_2,\dots,S_n)$，称 R 为系统可靠度。

例如，$\phi(s_1,s_2,\dots,s_n)=\prod_{i=1}^n s_i$, 则当且仅当所有部件正常工作时系统才正常工作，这样的系统称为串联系统，这时系统可靠度为

$$R=P(S_1=1,S_2=1,\dots,S_n=1)=\prod_{i=1}^{n}P(S_i=1)=p_1p_2\dots p_n.$$

在系统比较简单的情况下 (如串联、并联)，可以给出用 $p_1,p_2,\dots,p_n$ 表示 R 的表达式。但是更复杂的系统则很难写出 R 的表达式，这时可以用随机模拟方法估计 R。

记 $S=(S_1,S_2,\dots,S_n)$, $X=\phi(S)$, 对 S 独立抽取 N 个点 $S^{(j)}=(S_1^{(j)},S_2^{(j)},\dots,S_n^{(j)}), j=1,2,\dots,N$, R 可以用平均值法估计为

$$\hat{R}_1=\frac{1}{N}\sum_{j=1}^{N}\phi(S^{(j)}). \tag{14.1}$$

令 $Y=\sum_{i=1}^{n}(S_i-p_i)$, 则 $EY=0$, Y 与 X 正相关。用一个小的抽样先近似估计 $\mathrm{Cov}(X,Y),\mathrm{Var}(Y)$ 得到 b 的近似值，可以得到方差缩减的估计量

$$\hat{R}_2=\frac{1}{N}\sum_{j=1}^{N}\left[\phi(S^{(j)})+b\sum_{i=1}^{n}(S_i^{(j)}-p_i)\right].$$

※※※※※

14.2 对立变量法

控制变量法需要知道控制变量 Y 的期望值真值，并精确或近似知道 $\mathrm{Cov}(X,Y)$ 和 $\mathrm{Var}(Y)$。对立变量法的要求比较简单。

模拟中经常使用均匀分布随机数 U 变换产生的随机数 $X=g(U)$。下面的定理说明，如果变换 $g(\cdot)$ 是单调的，则随机变量 $Y=g(1-U)$ 就是与 X 负相关的。注意 $g(1-U)$ 与 $g(U)$ 同分布所以 $EY=EX$。

定理 14.1. *设 g 为单调函数，$U\sim U(0,1)$, 则 $Cov(g(U),g(1-U))\le 0$。*

证明： $\forall u_1,u_2\in[0,1]$，由 g 单调可知

$$(g(u_1)-g(u_2))\,(g(1-u_1)-g(1-u_2))\le 0$$

设 U_2 服从 U(0,1) 且与 U 独立，令 $X_1=g(U),Y_1=g(1-U)$, $X_2=g(U_2),Y_2=g(1-U_2)$, 则 X_1,Y_1,X_2,Y_2 的分布相同，且

$$\begin{aligned}&E(X_1-X_2)(Y_1-Y_2)\\=&\mathrm{Cov}(X_1-X_2,Y_1-Y_2)\\=&\mathrm{Cov}(X_1,Y_1)+\mathrm{Cov}(X_2,Y_2)-\mathrm{Cov}(X_1,Y_2)-\mathrm{Cov}(X_2,Y_1)\\=&2\mathrm{Cov}(X_1,Y_1)\end{aligned}$$

注意 $(X_1-X_2)(Y_1-Y_2)\le 0$ 所以 $\mathrm{Cov}(X_1,Y_1)\le 0$，即 $\mathrm{Cov}(g(U),g(1-U))\le 0$。证毕。

定理14.1可以推广到如下情形。

定理 14.2. *设 $h(x_1,x_2,\dots,x_n)$ 是关于每个自变量单调的函数，$U_1,U_2,\dots,U_n$ 相互独立同 $U(0,1)$ 分布，则 $Cov(h(U_1,U_2,\dots,U_n),\ h(1-U_1,1-U_2,\dots,1-U_n))\le 0$。*

证明见附录D.1。

对均匀随机数 U 最常见的变换是逆变换 $X = F^{-1}(U)$。下面的定理给出了提高 $I = EX$ 估计精度的方法。

定理 14.3 (对立变量法)**.** 设 $F(x)$ 为连续分布函数,$U \sim U(0,1), X = F^{-1}(U)$, $Y = F^{-1}(1-U)$, $Z = \frac{X+Y}{2}$, 则 X 与 Y 同分布 $F(x)$ 且 $Cov(X,Y) \le 0$,

$$Var(Z) \le \frac{1}{2} Var(X)$$

证明: 因为 U 和 $1-U$ 同分布所以 $X = F^{-1}(U)$ 和 $Y = F^{-1}(1-U)$ 同分布。由定理14.1, 令 $g(\cdot) = F^{-1}(\cdot)$ 可知 $\mathrm{Cov}(X,Y) = \mathrm{Cov}(g(U), g(1-U)) \le 0$, 从而

$$\begin{aligned}\mathrm{Var}(Z) =& \frac{\mathrm{Var}(X) + \mathrm{Var}(Y) + 2\mathrm{Cov}(X,Y)}{4} \\ =& \frac{\mathrm{Var}(X) + \mathrm{Cov}(X,Y)}{2} \le \frac{1}{2}\mathrm{Var}(X)\end{aligned}$$

证毕。

※※※※※

根据定理14.3的结论，为了估计 $I = EX$，产生 $U_1, U_2, \dots, U_N$ 后用

$$Z_i = \frac{1}{2}\left(F^{-1}(U_i) + F^{-1}(1-U_i)\right),\ i = 1, 2, \dots, N$$

的样本平均值估计 I 可以提高精度, 在不增加抽样个数的条件下把估计的随机误差方差降低至少一半。利用定理14.1或14.2提高随机模拟精度的方法叫做**对立变量法**。对立变量法不需要计算 X 和 Y 的方差及协方差的值，比控制变量法更简便易行。

例 14.3. 再次考虑例14.1的问题，估计 $I = \int_0^1 e^t\, dt$。下面用对立变量法改善原始的平均值法 $\hat{I}_1$ 的估计方差。

设 $U \sim \mathrm{U}(0,1)$，$X = e^U$, 令 $Y = e^{1-U}$, 用对立变量法估计 I 为

$$\hat{I}_3 = \frac{1}{N}\sum_{i=1}^{N} \frac{e^{U_i} + e^{1-U_i}}{2},$$

方差为

$$\mathrm{Var}(\hat{I}_3) = \frac{1}{N/2}\frac{\mathrm{Var}(e^U) + \mathrm{Cov}(e^U, e^{1-U})}{2} = \frac{1}{N}(-\frac{3}{4}e^2 + \frac{5}{2}e - \frac{5}{4}) \approx \frac{0.0039}{N},$$

$\hat{I}_1$ 的方差比对立变量法估计 $\hat{I}_3$ 的方差大至少 60 倍，而 $\hat{I}_3$ 的方差和例14.1中控制变量法估计量 $\hat{I}_2$ 的方差相近。

※※※※※

对立变量法和控制变量法是类似做法，一般不能结合使用。

例 14.4 (可靠性估计的对立变量法 (*))**.** 再次考虑例14.2的可靠度估计问题。用对立变量法改善估计方差。

设 $\{U_k\}$ 为标准均匀分布随机数列，取

$$S_i^{(j)} = \begin{cases} 1 & \text{当} U_{n(j-1)+i} \le p_i, \\ 0 & \text{其它} \end{cases}$$

则 $S_i^{(j)}$ 是 $U_{n(j-1)+i}$ 的单调非增函数。R 用平均值法估计为

$$\hat{R}_1 = \frac{1}{N}\sum_{j=1}^{N} \phi(S_1^{(j)}, S_2^{(j)}, \dots, S_n^{(j)}).$$

利用对立变量法，令 $h(U_1, U_2, \dots, U_n) = \phi(S_1, S_2, \dots, S_n)$, 则 h 关于每个自变量是单调非增函数，于是

$$\mathrm{Cov}(h(U_1, U_2, \dots, U_n),\ h(1-U_1, 1-U_2, \dots, 1-U_n)) \le 0,$$

估计系统可靠度 R 为

$$\hat{R}_3 = \frac{1}{N}\sum_{j=1}^{N} \frac{h(U_1^{(j)}, U_2^{(j)}, \dots, U_n^{(j)}) + h(1-U_1^{(j)}, 1-U_2^{(j)}, \dots, 1-U_n^{(j)})}{2}$$

就能比 $\hat{R}_1$ 的误差方差至少降低 $\frac{1}{2}$。

※※※※※

注意实际应用中定理14.2中的 $U_1, U_2, \dots, U_n$ 独立同 U(0,1) 分布的要求可以推广。例如，设 $X_i \sim \mathrm{N}(\mu_i, \sigma_i^2)$ 相互独立，要估计 $\theta = Eh(X_1, X_2, \dots, X_n)$, 其中 h 关于每个自变量单调不减，则 $h(2\mu_1 - X_1, 2\mu_2 - X_2, \dots, 2\mu_n - X_n)$ 与 $h(X_1, X_2, \dots, X_n)$ 同分布，对

$$Z = \frac{h(X_1, X_2, \dots, X_n) + h(2\mu_1 - X_1, 2\mu_2 - X_2, \dots, 2\mu_n - X_n)}{2}$$

抽样用平均值估计 θ 可以比仅对 $h(X_1, X_2, \dots, X_n)$ 抽样得到的估计量的方差缩减一半以上。事实上，令 $f(x_1, \dots, x_n) = h(x_1, \dots, x_n)$, $g(x_1, \dots, x_n) = -h(2\mu_1 - x_1, \dots, 2\mu_n - x_n)$，由附录D.1定理D.1可知 $h(X_1, X_2, \dots, X_n)$ 与 $h(2\mu_1 - X_1, 2\mu_2 - X_2, \dots, 2\mu_n - X_n)$ 负相关。

14.3 条件期望法

进行统计估计时，如果有额外的相关信息，利用这样的信息可以提高估计精度。比如，对随机变量 Y，如果 Y 服从某种模型，在估计 $I = EY$ 时应当尽量利用模型信息。

设变量 X 与 Y 不独立，根据 Rao-Blackwell 不等式:

$$\mathrm{Var}\left\{\mathrm{E}(Y|X)\right\} \leq \mathrm{Var}(Y)$$

又

$$\mathrm{E}\left\{\mathrm{E}(Y|X)\right\} = \mathrm{E}Y = I$$

所以，对 $Z = \mathrm{E}(Y|X)$ 抽样, 用 Z 的样本平均值来估计 $I = \mathrm{E}Y$ 比直接用 Y 的样本平均值的精度更高。这种改善随机模拟估计精度的方法叫做**条件期望法**，或 Rao-Blackwell 方法。

例 14.5. 设 $X \sim p(x)$，$\varepsilon \sim \mathrm{N}(0, \sigma^2)$ 且与 X 独立，

$$Y = \psi(X) + \varepsilon,$$

估计 $I = \mathrm{E}Y$。

可以用条件分布抽样法对二元随机向量 $Z = (X, Y)$ 抽样产生 Y 的样本。从 $p(\cdot)$ 抽样得 $X_1, X_2, \dots, X_N$, 独立地从 $\mathrm{N}(0,\sigma^2)$ 抽样得 $\varepsilon_1, \varepsilon_2, \dots, \varepsilon_N$，令

$$Y_i = \psi(X_i) + \varepsilon_i,\ i = 1, 2, \dots, N$$

然后用 $Y_1, Y_2, \dots, Y_N$ 的样本平均值估计 EY:

$$\hat{I}_1 = \frac{1}{N}\sum_{i=1}^{N} Y_i,$$

估计方差为

$$\mathrm{Var}(\hat{I}_1) = \frac{\mathrm{Var}(Y_1)}{N} = \frac{\mathrm{Var}(\psi(X_1))}{N} + \frac{\sigma^2}{N}.$$

另一方面，注意 $E(Y|X) = \psi(X)$，也可以只对 X 抽样然后用条件期望法估计 EY:

$$\hat{I}_2 = \frac{1}{N}\sum_i \psi(X_i),$$

估计方差为

$$\mathrm{Var}(\hat{I}_2) = \frac{\mathrm{Var}(\psi(X_1))}{N} < \mathrm{Var}(\hat{I}_1).$$

这个例子演示了条件期望法可以缩减误差方差的原因：对 Y 抽样分成两步进行：第一步对 X 抽样，第二步利用 $Y|X$ 的条件分布对 Y 抽样。所以使用 Y 的样本估计 EY 包含了第二步对 Y 抽样的随机误差，而使用 X 的函数 $E(Y|X) = \psi(X)$ 的抽样来估计 EY 则避免了第二步对 Y 抽样引起的随机误差。

※※※※※

例 14.6 (圆周率估计改进 (*))**.** 考虑例10.1中用随机模拟方法估计 π 的改进问题。

设 (X,Y) 服从正方形 $D = [-1,1]^2$ 上的均匀分布，令 $\eta = 1$ 表示 (X,Y) 落入单位圆 $C = \{(x,y) : x^2 + y^2 \le 1\}$, $\eta = 0$ 表示未落入单位圆, 则 $\eta \sim \mathrm{B}(1, \pi/4)$。设 $(X_i, Y_i), i = 1, 2, \ldots, N$ 是 (X,Y) 的独立重复抽样，η_i 表示 $X_i^2 + Y_i^2 \le 1$ 是否发生，则 π 的估计为

$$\hat{\pi}_1 = \frac{4}{N}\sum_{i=1}^{N} \eta_i,$$

方差为 $\mathrm{Var}(\hat{\pi}_1) = \pi(4-\pi)/N \approx \frac{2.6968}{N}$。

用 $\zeta = E(\eta|X)$ 的样本代替 η 来估计 $E\eta = \pi/4$，则由

$$\begin{aligned} E(\eta|X=x) =& P(X^2 + Y^2 \le 1 \,|\, X = x) = P(Y^2 \le 1 - x^2) \\ =& P(-\sqrt{1-x^2} \le Y \le \sqrt{1-x^2}) \\ =& \sqrt{1-x^2} \quad (\text{注意} Y \sim \mathrm{U}(-1,1)) \end{aligned}$$

可知 $\zeta = \sqrt{1-X^2}$。其方差为

$$\mathrm{Var}(\zeta) = E(1-X^2) - (E\zeta)^2 = \frac{2}{3} - \frac{\pi^2}{16},$$

估计 π 为

$$\hat{\pi}_2 = \frac{4}{N}\sum_{i=1}^{N}\sqrt{1-X_i^2},$$

方差为 $\mathrm{Var}(\hat{\pi}_2) \approx 0.7971/N$，$\mathrm{Var}(\hat{\pi}_1)/\mathrm{Var}(\hat{\pi}_2) \approx 3.4$。

另外，ζ 仅依赖于 X^2，容易发现若 $U \sim \mathrm{U}(0,1)$ 则 X^2 和 U^2 同分布，所以可取 $\xi = \sqrt{1-U^2}$, 其中 $U \sim \mathrm{U}(0,1)$。这时，函数 $h(u) = \sqrt{1-u^2}$ 是 $u \in (0,1)$ 的单调函数，可以利用对立变量法，构造 π 的估计量为

$$\hat{\pi}_3 = \frac{4}{N}\sum_{i=1}^{N}\frac{\sqrt{1-U_i^2}+\sqrt{1-(1-U_i)^2}}{2},$$

其中 $U_1, U_2, \ldots, U_N$ 为 $U(0,1)$ 随机数,则 $\mathrm{Var}(\hat{\pi}_3) \approx 0.11/N$, $\mathrm{Var}(\hat{\pi}_1)/\mathrm{Var}(\hat{\pi}_3) \approx 25$。

也可以用对立变量法来改进 $\hat{\pi}_2$。令 $W = U^2 - \frac{1}{3}$, 则 $EW = 0$ 且 W 与 ζ 负相关。可以先进行一个小规模的模拟估计 $\mathrm{Cov}(\zeta, W)$ 和 $\mathrm{Var}(W)$ 得到 $b = -\mathrm{Cov}(\zeta, W)/\mathrm{Var}(W)$ 的近似值，用 $\zeta(b) = \zeta + bW$ 代替 ζ 进行抽样，可以减小 $\hat{\pi}_2$ 的方差。

※※※※※

例 14.7. 在标准化重要抽样法中应用条件期望法减小方差。

设随机变量 $Y \sim f(y)$, 为了估计 $\theta = Eh(Y)$, 经常使用标准化重要抽样法

$$\hat{\theta}_1 = \frac{\sum_{i=1}^{N} W_i h(X_i)}{\sum_{i=1}^{N} W_i},$$

其中 X_i 为试抽样密度 $g(x)$ 的样本，权重 $W_i = f(X_i)/g(X_i)$。如果 $x = (u, v)$, $h(x) = h_1(u)$, 则只要对 $X = (U, V)$ 的分量 U 抽样得到 $U_i, i = 1, \ldots, N$ 及新的权重 $\tilde{W}_i = f_U(U_i)/g_U(U_i)$, 其中 $f_U(u) = \int f(u,v)\,dv$, $g_U(u) = \int g(u,v)\,dv$, 则可估计 θ 为

$$\hat{\theta}_2 = \frac{\sum_{i=1}^{N} \tilde{W}_i h_1(U_i)}{\sum_{i=1}^{N} \tilde{W}_i},$$

这时

$$\mathrm{Var}(\tilde{W}_i) \le \mathrm{Var}(W_i),\ i = 1, 2, \dots, N.$$

事实上，

$$\begin{aligned}\frac{f_U(u)}{g_U(u)} &= \int \frac{f(u,v)}{g_U(u)}\,dv = \int \frac{f(u,v)}{g_U(u)g_{V|U}(v|u)} g_{V|U}(v|u)\,dv \\ &= \int \frac{f(u,v)}{g(u,v)} g_{V|U}(v|u)\,dv = E_g\left\{ \frac{f(U,V)}{g(U,V)} \middle| U = u \right\}\end{aligned}$$

于是

$$\mathrm{Var}_g\left\{\frac{f(U,V)}{g(U,V)}\right\} \ge \mathrm{Var}_g\left\{E_g\left[\frac{f(U,V)}{g(U,V)}\middle| U\right]\right\} = \mathrm{Var}_g\left\{\frac{f_U(U)}{g_U(U)}\right\}.$$

※※※※※

14.4 随机数复用

在统计研究中，经常需要比较若干种统计方法的性能，如偏差、方差、覆盖率等。除了努力获取理论结果以外，可以用随机模拟方法进行比较：重复生成 N 组随机样本，对每个样本同时用不同统计方法计算结果，最后从 N 组结果比较不同方法的性能。这样比较时，并没有对每种方法单独生成 N 组样本，而是每个样本同时应用所有要比较的方法。这样不仅减少了计算量，而且在比较时具有更高的精度。

例 14.8. 对正态分布总体 $X \sim N(\mu, \sigma^2)$, 如果有样本 $X_1, X_2, \dots, X_n$, 估计 σ^2 有两种不同的公式:

$$\hat{\sigma}_1^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2, \hat{\sigma}_2^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2.$$

希望比较两个估计量的偏差和均方误差:

$$\begin{aligned} b_1 &= E\hat{\sigma}_1^2 - \sigma^2, \quad b_2 = E\hat{\sigma}_2^2 - \sigma^2, \\ s_1 &= E(\hat{\sigma}_1^2 - \sigma^2)^2, \quad s_2 = E(\hat{\sigma}_2^2 - \sigma^2)^2. \end{aligned}$$

当然，这个问题很简单，可以得到偏差和均方误差的理论值：

$$\begin{aligned} b_1 &= 0, \quad b_2 = -\frac{1}{n}\sigma^2, \\ s_1 &= \frac{2\sigma^4}{n-1}, \quad s_2 = \frac{(2n-1)\sigma^4}{n^2}, \quad s_1 - s_2 = \frac{(3n-1)\sigma^4}{n^2(n-1)}. \end{aligned}$$

我们用随机模拟来作比较。重复地生成 N 组样本 $(X_1^{(j)}, X_2^{(j)}, \dots, X_n^{(j)})$, $j = 1, 2, \dots, N$。对每组样本分别计算 $\hat{\sigma}_{1j}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i^{(j)} - \bar{X}^{(j)})^2$ 和 $\hat{\sigma}_{2j}^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i^{(j)} - \bar{X}^{(j)})^2$，得到偏差和均方误差的估计

$$\begin{aligned}\hat{b}_1 =& \frac{1}{N}\sum_{j=1}^{N}\hat{\sigma}_{1j}^2 - \sigma^2, \quad \hat{b}_2 = \frac{1}{N}\sum_{j=1}^{N}\hat{\sigma}_{2j}^2 - \sigma^2,\\ \hat{s}_1 =& \frac{1}{N}\sum_{j=1}^{N}(\hat{\sigma}_{1j}^2 - \sigma^2)^2, \quad \hat{s}_2 = \frac{1}{N}\sum_{j=1}^{N}(\hat{\sigma}_{2j}^2 - \sigma^2)^2.\end{aligned}$$

容易看出，两种方法使用相同的模拟样本得到的偏差、均方误差的估计精度与每种方法单独生成模拟样本得到的估计精度相同。

但是，如果要估计 $\Delta s = s_1 - s_2$，利用相同的样本的估计精度更好。一般地，

$$\mathrm{Var}(\hat{s}_1 - \hat{s}_2) = \mathrm{Var}(\hat{s}_1) + \mathrm{Var}(\hat{s}_2) - 2\mathrm{Cov}(\hat{s}_1, \hat{s}_2). \tag{14.2}$$

如果每种方法使用不同的样本，则(14.2)变成

$$\mathrm{Var}(\hat{s}_1 - \hat{s}_2) = \frac{1}{N}\left[\mathrm{Var}((\hat{\sigma}_1^2 - \sigma^2)^2) + \mathrm{Var}((\hat{\sigma}_2^2 - \sigma^2)^2)\right]. \tag{14.3}$$

如果两种方法利用相同的样本计算，则(14.2)变成

$$\begin{aligned}\mathrm{Var}(\hat{s}_1 - \hat{s}_2) =& \frac{1}{N}\left[\mathrm{Var}((\hat{\sigma}_1^2 - \sigma^2)^2) + \mathrm{Var}((\hat{\sigma}_2^2 - \sigma^2)^2)\right. & (14.4)\\ & \left. -2\mathrm{Cov}((\hat{\sigma}_1^2 - \sigma^2)^2, (\hat{\sigma}_2^2 - \sigma^2)^2)\right]. & (14.5)\end{aligned}$$

而 $(\hat{\sigma}_1^2 - \sigma^2)^2$ 与 $(\hat{\sigma}_2^2 - \sigma^2)^2$ 明显具有强正相关，所以两种方法针对相同样本计算时 $\hat{s}_1 - \hat{s}_2$ 的方差比两种方法使用单独样本时的方差要小得多。这说明重复利用相同的随机数或样本往往可以提高比较的精度。可以计算出(14.3)约为 $16n^{-2}/N$, (14.3)约为 $104n^{-3}/N$。

※※※※※

习题

习题 1

用随机模拟法计算二重积分 $\int_0^1\int_0^1 e^{(x+y)^2}\,dy\,dx$, 用对立变量法改善精度。

习题 2

用 R 程序实现例14.8的模拟。取 $\mu = 0, \sigma = 1, N = 100000, n = 5, 10, 30$，列出各偏差、均方误差和 $\hat{s}_1 - \hat{s}_2$ 的值。

Chapter 15

随机服务系统模拟

15.1 概述

我们在第10章中讲到，较为复杂的随机模型往往难以进行彻底的理论分析，这时常常使用随机模拟方法产生模型的大量数据，从产生的数据对模型进行统计推断。随机服务系统就是这样的一种模型，经常需要利用随机模拟方法进行研究。

随机服务系统在我们日常生活、工业生产、科学技术、军事领域中是经常遇到的随机模型，比如，研究银行、理发店、商店、海关通道、高速路收费口等服务人员个数的设置和排队规则，研究计算机网络网关、移动网络的调度规则，等等。

在概率统计理论中排队论用来研究随机服务系统的数学模型，可以用来设计适当的服务机构数目和排队规则。如下面的 M/M/1 排队系统。

例 15.1. 设某银行仅有一个柜员，并简单假设银行不休息。顾客到来间隔的时间服从独立的指数分布 $\mathrm{Exp}(\lambda)$ ($1/\lambda$ 为间隔时间的期望值), 如果柜员正在为先前的顾客服务, 新到顾客就排队等待，柜员为顾客服务的时间服从均值为 $1/\mu$ 的指数分布，设 $u = \lambda/\mu < 1$。设 X_t 表示 t 时刻在银行内的顾客数（包括正在服务的和正在排队的），则 X_t 是一个连续时马氏链。

这是一个生灭过程马氏链，有理论结果表明当系统处于稳定状态时

$$P(X_t = i) = u^i(1-u),\ i = 0, 1, 2, \ldots .$$

设随机变量 N 服从 X_t 的平稳分布。于是银行中平均顾客数为

$$EN = \frac{u}{1-u},$$

平均队列长度 EQ 等于 EN 减去平均正在服务人数，正在服务人数 Y_t 为

$$Y_t = \begin{cases} 1, & \text{当} X_t > 0, \\ 0, & \text{当} X_t = 0 \end{cases}$$

所以 $EY_t = P(Y_t = 1) = 1 - P(N = 0) = u$, 于是平均队列长度为

$$EQ = EN - EY_t = \frac{u}{1-u} - u = \frac{u^2}{1-u},$$

设顾客平均滞留时间为 ER，由关系式

$$EN = \lambda \cdot ER$$

可知平均滞留时间为

$$ER = \frac{u}{\lambda(1-u)} = \frac{1}{\mu - \lambda},$$

进一步分析还可以知道顾客滞留时间 R 服从均值为 $1/(\mu - \lambda)$ 的指数分布。

※※※※※

从上面的例子可以发现，一个随机服务系统的模型应该包括如下三个要素：

- 输入过程: 比如，银行的顾客到来的规律。
- 排队规则: 比如，银行有多个柜员时，顾客是选最短一队，还是随机选一队，还是统一排队等候叫号，顾客等得时间过长后是否会以一定概率放弃排队。
- 服务机构：有多少个柜员，服务时间的分布等。

虽然某些随机服务系统可以进行严格理论分析得到各种问题的理论解，但是，随机服务系统中存在大量随机因素，使得理论分析变得很困难以至于不可能。例如，即使是上面的银行服务问题，可能的变化因素就包括：顾客到来用齐次泊松过程还是非齐次泊松过程，柜员有多少个，是否不同时间段柜员个数有变化，柜员服务时间服从什么样的分布，顾客排队按照什么规则，是否 VIP 顾客提前服务，顾客等候过长时会不会放弃排队，等等。包含了这么多复杂因素的随机

服务系统的理论分析会变得异常复杂，完全靠理论分析无法解决问题，这时，可以用随机模拟方法给出答案。

在模拟随机服务系统时，可以按时间顺序记录发生的事件，如顾客到来、顾客接受服务、顾客结束服务等，这样的系统的模拟也叫做**离散事件模拟** (Discrete Event Simulation, DES)。

离散事件模拟算法可以分为三类：

- 活动模拟，把连续时间离散化为很小的单位，比如平均几秒发生一个新事件时可以把时间精确到毫秒，然后时钟每隔一个小的时间单位前进一步并查看所有活动并据此更新系统状态。缺点是速度太慢。
- 事件模拟。仅在事件发生时更新时钟和待发生的事件集合。这样的方法不受编程语言功能的限制，运行速度很快，也比较灵活，可以实现复杂逻辑，但是需要自己管理的数据结构和逻辑结构比较复杂，算法编制相对较难。
- 过程模拟。把将要到来的各种事件看成是不同的过程，让不同的过程并行地发生, 过程之间可以交换消息，并在特殊的软件包或编程语言支持下自动更新系统状态。在计算机实现中需要借助于线程或与线程相似的程序功能。这类软件包有 C++ 语言软件包 C++SIM 和 Python 语言软件包 SimPy, R 的 simmer 包，Julia 的 SimJulia 包。优点是系统逻辑的编码很直观，程序模块化，需要用户自己管理的数据结构和逻辑结构少。过程模拟是现在更受欢迎的离散事件模拟方式。

事件模拟算法必须考虑的变量包括：

- 当前时刻 t;
- 随时间而变化的计数变量，如 t 时刻时到来顾客人数、已离开人数；
- 系统状态，比如是否有顾客正在接受服务、排队人数、队列中顾客序号。

这些变量仅在有事件发生时（如顾客到来、顾客离开）才需要记录并更新。为了持续模拟后续事件，需要维护一个将要发生的事件的列表（下一个到来时刻、下一个离开时刻），列表仅需要在事件发生时进行更新。其它变量可以从这三种变量中推算出来，比如，顾客 i 在时间 t_1 时到达并排队，在时间 t_2 时开始接受服务，在时间 t_4 时结束服务离开，则顾客 i 的滞留时间为 $t_4 - t_1$。

例 15.2. 用事件模拟的方法来模拟例15.1。目的是估计平均滞留时间 ER。

想法是，模拟生成各种事件发生的时间，模拟很长时间，丢弃开始的一段时间 T_0 后，用 T_0 后到达的顾客的总滞留时间除以 T_0 后到达的顾客人数来估计平均滞留时间。设 T_0 时间后每位顾客滞留时间的模拟值为 $R_i, i = 1, 2, \ldots, m$，可以用 $\{R_i\}$ 作为随机变量 R 的样本来检验 R 的分布是否指数分布。

用事件模拟方法进行离散事件模拟的算法关键在于计算系统状态改变的时间，即各个事件的发生时间，这个例子中就是顾客到来、顾客开始接受服务、顾客离开这样三种事件, 由此还可以得到每个顾客排队的时间和服务的时间。

在没有明确算法构思时，可以从时间 0 开始在纸上按照模型规定人为地生成一些事件并人为地找到下一事件。这样可以找到要更新的数据结构和更新的程序逻辑。

下面的算法保持了一个将要发生的事件的集合，在每次事件发生时更新时钟，更新时钟时从事件集合中找到最早发生的事件进行处理，并生成下一事件到事件集合中，如此重复直到需要模拟的时间长度。

M/M/1 事件模拟算法

{初始化当前时钟 $t \leftarrow 0$, 柜员忙标志 $B \leftarrow 0$, 当前排队人数 $L \leftarrow 0$,
最新到来顾客序号 $i \leftarrow 0$, 正在服务顾客序号 $j \leftarrow 0$, 已服务顾客数 $n \leftarrow 0$ }
从 Exp(λ) 抽取 X, 设置下一顾客来到时间 $A \leftarrow X$
repeat {
  if($B = 0$ **or** ($B = 1$ **and** $A < E$)) { # E 是正在服务的顾客结束时刻
    $t \leftarrow A$
  } **else** {
    $t \leftarrow E$
  }
  if ($t > T_1$) **break** # T_1 是预先确定的模拟时长

  if($t == A$) { # 待处理到达事件
    $L \leftarrow L + 1$
    $i \leftarrow i + 1$, 记录第 i 位顾客到来时间 $a_i \leftarrow t$
    从 Exp(λ) 抽取 X, $A \leftarrow t + X$
    if($B == 0$) { # 不用排队，直接服务
      $B \leftarrow 1$, $L \leftarrow L - 1$

M/M/1 事件模拟算法

```
        j ← j + 1, 置第 j 位顾客开始服务时间 s_j ← t
        从 Exp(μ) 抽取 Y, 置 E ← t + Y
      }
    } else { # 待处理结束服务事件
      B ← 0
      n ← n + 1, 记录第 n 个顾客结束服务时间 e_n ← t
      if(L > 0) { # 排队顾客开始服务
        L ← L − 1
        B ← 1
        j ← j + 1, s_j ← t
        从 Exp(μ) 抽取 Y, 置 E ← t + Y
      }
    }
}
```

令 $I = \{i:\ T_0 \le s_i \le T_1\}$, 求 $\{e_i - a_i,\ i \in I\}$ 的平均值作为 ER 估计

从这个算法可以看出，事件模拟方法需要用户自己管理待处理事件集合与时钟, 算法设计难度较大。

※※※※※

用随机服务系统进行建模和模拟研究的一般步骤如下：

- 提出问题。比如，银行中顾客平均滞留时间与相关参数的关系。
- 建立模型。比如，设顾客到来服从泊松过程，设每个顾客服务时间服从独立的指数分布。
- 数据收集与处理。比如，估计银行顾客到来速率，每个顾客平均服务时间，等等。
- 建立模拟程序。一般使用专用的模拟软件包或专用模拟语言编程。
- 模拟模型的正确性确认。
- 模拟试验。
- 模拟结果分析。

离散事件模拟问题一般都比较复杂，即使借助于专用模拟软件或软件包，也很难确保实现的模拟算法与问题的实际设定是一致的。为此，需要遵循一些提高算法可信度的一般规则。算法程序一定要仔细检查，避免出现参数错误、逻辑错误。在开始阶段，可以利用较详尽的输出跟踪模拟运行一段时间，人工检查系统运行符合原始设定。尽可能利用模块化程序设计，比如，在 M/M/1 问题模拟中，顾客到来可能遵循不同的规律，比如时齐泊松过程、非时齐泊松过程，把产生顾客到来时刻的程序片段模块化并单独检查验证，就可以避免在这部分出错。问题实际设定可能比较复杂，在程序模块化以后，如果有一种较简单的设定可以得到理论结果，就可以用理论结果验证算法输出，保证程序框架正确后，再利用模块化设计修改各个模块适应实际复杂设定。

15.2 用 R 的 simmer 包进行随机服务系统模拟 (*)

R 扩展包 simmer 是仿照 python 语言的 simpy 包编写的一个用于离散事件模拟的软件包，属于过程模拟算法，利用这样的软件包可以使得离散事件模拟程序变得比较简单。

例 15.3. 用 R 的 simmer 包模拟 M/M/1 随机服务系统。比较简单的版本，进行短时间的模拟，用详细过程输出验证算法的合理性。

程序：

```
demo.mm1.simmer1 <- function(T1=20, lambda=1, mu=1.2){
  library(simmer)

  ## 建立模拟环境
  sim <- simmer("M/M/1 模拟")

  ## 顾客的轨迹
  customer <- trajectory(" 顾客") |>
    ## 进入系统
    log_(" 进入银行") |>
    ## 排队或直接服务
    seize(" 柜员", 1) |>
```

```
    log_(" 开始接受服务") |>
    timeout(function() rexp(1, rate=mu)) |>
    log_(" 结束服务") |>
    release(" 柜员", 1)

  ## 指定资源和顾客到来
  sim |>
    add_resource(" 柜员", 1) |>
    add_generator(" 顾客", customer, function() rexp(1, rate=lambda))

  ## 运行
  sim |>
    run(until=T1)
}
```

测试运行：

```
set.seed(1)
sim1 <- demo.mm1.simmer1()
## 0.755182: 顾客 0: 进入银行
## 0.755182: 顾客 0: 开始接受服务
## 0.876604: 顾客 0: 结束服务
## 1.93682: 顾客 1: 进入银行
## 1.93682: 顾客 1: 开始接受服务
## 2.07662: 顾客 2: 进入银行
## 2.30022: 顾客 1: 结束服务
## 2.30022: 顾客 2: 开始接受服务
## 3.32485: 顾客 2: 结束服务
## 4.97159: 顾客 3: 进入银行
## 4.97159: 顾客 3: 开始接受服务
## 5.51127: 顾客 4: 进入银行
## 5.65832: 顾客 5: 进入银行
## 5.76873: 顾客 3: 结束服务
```

```
## 5.76873: 顾客 4: 开始接受服务
## 6.40375: 顾客 4: 结束服务
## 6.40375: 顾客 5: 开始接受服务
## 7.04905: 顾客 6: 进入银行
## 7.43509: 顾客 5: 结束服务
## 7.43509: 顾客 6: 开始接受服务
## 8.31388: 顾客 6: 结束服务
## 11.473: 顾客 7: 进入银行
## 11.473: 顾客 7: 开始接受服务
## 12.5082: 顾客 8: 进入银行
## 13.0363: 顾客 7: 结束服务
## 13.0363: 顾客 8: 开始接受服务
## 13.163: 顾客 9: 进入银行
## 13.3171: 顾客 8: 结束服务
## 13.3171: 顾客 9: 开始接受服务
## 13.7515: 顾客 10: 进入银行
## 14.3933: 顾客 11: 进入银行
## 14.6875: 顾客 12: 进入银行
## 15.2533: 顾客 13: 进入银行
## 15.2876: 顾客 9: 结束服务
## 15.2876: 顾客 10: 开始接受服务
## 15.3371: 顾客 10: 结束服务
## 15.3371: 顾客 11: 开始接受服务
## 15.3594: 顾客 14: 进入银行
## 15.8193: 顾客 11: 结束服务
## 15.8193: 顾客 12: 开始接受服务
## 16.7971: 顾客 12: 结束服务
## 16.7971: 顾客 13: 开始接受服务
## 17.6278: 顾客 13: 结束服务
## 17.6278: 顾客 14: 开始接受服务
## 18.8239: 顾客 14: 结束服务
## 19.3183: 顾客 15: 进入银行
## 19.3183: 顾客 15: 开始接受服务
```

```
## 19.3556: 顾客 16: 进入银行
## 19.5883: 顾客 15: 结束服务
## 19.5883: 顾客 16: 开始接受服务
## 19.7579: 顾客 16: 结束服务
```

用 `get_mon_arrivals()` 获取系统监测的事件发生和持续时间，结果是数据框：

```
sim1 |>
  get_mon_arrivals() |>
  knitr::kable(digits=2)
```

name	start_time	end_time	activity_time	finished	replication
顾客 0	0.76	0.88	0.12	TRUE	1
顾客 1	1.94	2.30	0.36	TRUE	1
顾客 2	2.08	3.32	1.02	TRUE	1
顾客 3	4.97	5.77	0.80	TRUE	1
顾客 4	5.51	6.40	0.64	TRUE	1
顾客 5	5.66	7.44	1.03	TRUE	1
顾客 6	7.05	8.31	0.88	TRUE	1
顾客 7	11.47	13.04	1.56	TRUE	1
顾客 8	12.51	13.32	0.28	TRUE	1
顾客 9	13.16	15.29	1.97	TRUE	1
顾客 10	13.75	15.34	0.05	TRUE	1
顾客 11	14.39	15.82	0.48	TRUE	1
顾客 12	14.69	16.80	0.98	TRUE	1
顾客 13	15.25	17.63	0.83	TRUE	1
顾客 14	15.36	18.82	1.20	TRUE	1
顾客 15	19.32	19.59	0.27	TRUE	1
顾客 16	19.36	19.76	0.17	TRUE	1

用 `get_mon_resources()` 获取系统监测的资源占用情况：

```
sim1 |>
  get_mon_resources() |>
  knitr::kable(digits=2)
```

resource	time	server	queue	capacity	queue_size	system	limit	replication
柜员	0.76	1	0	1	Inf	1	Inf	1
柜员	0.88	0	0	1	Inf	0	Inf	1
柜员	1.94	1	0	1	Inf	1	Inf	1
柜员	2.08	1	1	1	Inf	2	Inf	1
柜员	2.30	1	0	1	Inf	1	Inf	1
柜员	3.32	0	0	1	Inf	0	Inf	1
柜员	4.97	1	0	1	Inf	1	Inf	1
柜员	5.51	1	1	1	Inf	2	Inf	1
柜员	5.66	1	2	1	Inf	3	Inf	1
柜员	5.77	1	1	1	Inf	2	Inf	1
柜员	6.40	1	0	1	Inf	1	Inf	1
柜员	7.05	1	1	1	Inf	2	Inf	1
柜员	7.44	1	0	1	Inf	1	Inf	1
柜员	8.31	0	0	1	Inf	0	Inf	1
柜员	11.47	1	0	1	Inf	1	Inf	1
柜员	12.51	1	1	1	Inf	2	Inf	1
柜员	13.04	1	0	1	Inf	1	Inf	1
柜员	13.16	1	1	1	Inf	2	Inf	1
柜员	13.32	1	0	1	Inf	1	Inf	1
柜员	13.75	1	1	1	Inf	2	Inf	1
柜员	14.39	1	2	1	Inf	3	Inf	1
柜员	14.69	1	3	1	Inf	4	Inf	1
柜员	15.25	1	4	1	Inf	5	Inf	1
柜员	15.29	1	3	1	Inf	4	Inf	1
柜员	15.34	1	2	1	Inf	3	Inf	1
柜员	15.36	1	3	1	Inf	4	Inf	1
柜员	15.82	1	2	1	Inf	3	Inf	1
柜员	16.80	1	1	1	Inf	2	Inf	1
柜员	17.63	1	0	1	Inf	1	Inf	1
柜员	18.82	0	0	1	Inf	0	Inf	1
柜员	19.32	1	0	1	Inf	1	Inf	1
柜员	19.36	1	1	1	Inf	2	Inf	1
柜员	19.59	1	0	1	Inf	1	Inf	1
柜员	19.76	0	0	1	Inf	0	Inf	1

※※※※※

例 15.4. 上一例子的改进。用 R 的 simmer 包模拟一个银行，多名顾客以指数间隔时间到来 (速率 `lambda`) 有一个服务柜台仅一名柜员，只排一队，服务时间是速率为 `mu` 的指数分布随机数。程序更容易改成其它模拟问题，还增加了一些额外的输出。

R 程序：

```
demo.mm1.simmer2 <- function(T1=20, lambda=1.0, mu=1.2){
  library(simmer)

  ## 建立模拟环境
  bank <- simmer("bank")

  ## 用 trajectory() 建立顾客，并指定顾客的一系列活动
  ## seize() 获取柜员服务资源，如果正在忙，就进入排队
  ## 服务时间用 timeout 指定，为了生成多个随机服务时间，
  ## timeout 的参数是返回随机服务时间的而函数而不是时间值本身
  customer <-
    trajectory(" 顾客") |>
    log_(" 到达") |>
    set_attribute(" 到达时刻", function() {now(bank)}) |>
    seize(" 柜员") |>
    log_(function() paste(" 开始接受服务，排队时间: ",
      now(bank) - get_attribute(bank, " 到达时刻"))) |>
    timeout( function() rexp(1, mu)) |>
    release(" 柜员") |>
    log_(function(attr) {paste(" 离开: ", now(bank))})

  ## 用 add_resource 生成柜员资源
  ## 用 add_generator() 生成顾客到来列
  bank |>
    add_resource(" 柜员") |>
    add_generator(" 顾客", customer,
```

```
      function() {rexp(1, lambda)} )

  ## 用 run() 执行模拟到指定结束时刻
  bank |>
    run(until=T1)
}
```

测试:

```
set.seed(1)
sim2 <- demo.mm1.simmer2()
## 0.755182: 顾客 0: 到达
## 0.755182: 顾客 0: 开始接受服务，排队时间： 0
## 0.876604: 顾客 0: 离开： 0.876604105381506
## 1.93682: 顾客 1: 到达
## 1.93682: 顾客 1: 开始接受服务，排队时间： 0
## 2.07662: 顾客 2: 到达
## 2.30022: 顾客 1: 离开： 2.3002151337181
## 2.30022: 顾客 2: 开始接受服务，排队时间： 0.223595259614148
## 3.32485: 顾客 2: 离开： 3.32485017823051
## 4.97159: 顾客 3: 到达
## 4.97159: 顾客 3: 开始接受服务，排队时间： 0
## 5.51127: 顾客 4: 到达
## 5.65832: 顾客 5: 到达
## 5.76873: 顾客 3: 离开： 5.76872798965007
## 5.76873: 顾客 4: 开始接受服务，排队时间： 0.257456738084932
## 6.40375: 顾客 4: 离开： 6.40375286920846
## 6.40375: 顾客 5: 开始接受服务，排队时间： 0.745435627139374
## 7.04905: 顾客 6: 到达
## 7.43509: 顾客 5: 离开： 7.43508916148749
## 7.43509: 顾客 6: 开始接受服务，排队时间： 0.386036790607969
## 8.31388: 顾客 6: 离开： 8.31387513424638
## 11.473: 顾客 7: 到达
## 11.473: 顾客 7: 开始接受服务，排队时间： 0
```

```
## 12.5082: 顾客 8: 到达
## 13.0363: 顾客 7: 离开： 13.0363492322359
## 13.0363: 顾客 8: 开始接受服务，排队时间： 0.528118697642798
## 13.163: 顾客 9: 到达
## 13.3171: 顾客 8: 离开： 13.3171271292232
## 13.3171: 顾客 9: 开始接受服务，排队时间： 0.154149957416097
## 13.7515: 顾客 10: 到达
## 14.3933: 顾客 11: 到达
## 14.6875: 顾客 12: 到达
## 15.2533: 顾客 13: 到达
## 15.2876: 顾客 9: 离开： 15.2875565067429
## 15.2876: 顾客 10: 开始接受服务，排队时间： 1.53609961348276
## 15.3371: 顾客 10: 离开： 15.3370891404057
## 15.3371: 顾客 11: 开始接受服务，排队时间： 0.943739658913893
## 15.3594: 顾客 14: 到达
## 15.8193: 顾客 11: 离开： 15.8193495265548
## 15.8193: 顾客 12: 开始接受服务，排队时间： 1.13187965742313
## 16.7971: 顾客 12: 离开： 16.7971096147388
## 16.7971: 顾客 13: 开始接受服务，排队时间： 1.54377422102827
## 17.6278: 顾客 13: 离开： 17.627787077442
## 17.6278: 顾客 14: 开始接受服务，排队时间： 2.2683790604488
## 18.8239: 顾客 14: 离开： 18.8238581969195
## 19.3183: 顾客 15: 到达
## 19.3183: 顾客 15: 开始接受服务，排队时间： 0
## 19.3556: 顾客 16: 到达
## 19.5883: 顾客 15: 离开： 19.5883493298518
## 19.5883: 顾客 16: 开始接受服务，排队时间： 0.232739934309695
## 19.7579: 顾客 16: 离开： 19.7579412882115
```

用 `get_mon_arrivals()` 获取各个顾客到来的时间、离开时间、活动时间等：

```
sim2 |>
  get_mon_arrivals() |>
```

```
  dplyr::mutate(waiting_time = end_time - start_time - activity_time) |>
  knitr::kable(digits=2)
```

name	start_time	end_time	activity_time	finished	replication	waiting_time
顾客 0	0.76	0.88	0.12	TRUE	1	0.00
顾客 1	1.94	2.30	0.36	TRUE	1	0.00
顾客 2	2.08	3.32	1.02	TRUE	1	0.22
顾客 3	4.97	5.77	0.80	TRUE	1	0.00
顾客 4	5.51	6.40	0.64	TRUE	1	0.26
顾客 5	5.66	7.44	1.03	TRUE	1	0.75
顾客 6	7.05	8.31	0.88	TRUE	1	0.39
顾客 7	11.47	13.04	1.56	TRUE	1	0.00
顾客 8	12.51	13.32	0.28	TRUE	1	0.53
顾客 9	13.16	15.29	1.97	TRUE	1	0.15
顾客 10	13.75	15.34	0.05	TRUE	1	1.54
顾客 11	14.39	15.82	0.48	TRUE	1	0.94
顾客 12	14.69	16.80	0.98	TRUE	1	1.13
顾客 13	15.25	17.63	0.83	TRUE	1	1.54
顾客 14	15.36	18.82	1.20	TRUE	1	2.27
顾客 15	19.32	19.59	0.27	TRUE	1	0.00
顾客 16	19.36	19.76	0.17	TRUE	1	0.23

※※※※※

例 15.5. 用 R 的 simmer 包模拟例15.1。一个银行，多名顾客以指数间隔时间到来 (速率 `lambda`), 有一个服务柜台仅一名柜员，只排一队, 服务时间是速率为 mu 的指数分布随机数, 比较平均等待时间与理论值。

```
demo.mm1.simmer3 <- function(T0=0, T1=10000, lambda=1.0, mu=1.2){
  library(simmer)

  ## 建立模拟环境
  bank <- simmer("bank")

  ## 用 trajectory() 建立顾客，并指定顾客的一系列活动
```

```
## seize() 获取柜员服务资源，如果正在忙，就进入排队
## 服务时间用 timeout 指定，为了生成多个随机服务时间，
## timeout 的参数是返回随机服务时间的而函数而不是时间值本身
customer <-
  trajectory(" 顾客") |>
  seize(" 柜员") |>
  timeout( function() rexp(1, mu)) |>
  release(" 柜员")

## 用 add_resource 生成柜员资源
## 用 add_generator() 生成顾客到来列
bank |>
  add_resource(" 柜员") |>
  add_generator(" 顾客", customer, function() {rexp(1, lambda)} )

## 用 run() 执行模拟到指定结束时刻
bank |>
  run(until=T1)

## 用 get_mon_arrivals() 获取各个顾客到来的时间、离开时间、活动时间等，
## 结果是数据框
resd <- bank |>
  get_mon_arrivals() |>
  dplyr::mutate(
    waiting_time = end_time - start_time - activity_time,
    stay_time = end_time - start_time)

stay_times <- resd |>
  dplyr::filter(start_time >= T0, end_time < T1) |>
  dplyr::select(stay_time) |>
  dplyr::pull()

ER <- mean(stay_times)
```

```
  ER.true <- 1/(mu-lambda)
  cat('估计的平均滞留时间 ER=', ER,
      ' 期望值 =', ER.true, '\n')
  invisible(c(ER.hat=ER, ER.true=ER.true))
}
```

测试:

```
set.seed(1)
demo.mm1.simmer3()
## 估计的平均滞留时间 ER= 4.790878  期望值 = 5
```

※※※※※

15.3 用 Julia 语言的 SimJulia 包进行离散事件模拟 (*)

Julia 的SimJulia库是一个模仿 Python 语言 SimPy 库的软件包，采用过程模拟方法实现了离散事件模拟功能。

将 SimJulia 的离散事件模拟看作一场戏剧表演，需要一个 “环境” 作为舞台，演员作为 “过程”，用 Julia 的可恢复函数实现，用 `@resumable` 前缀说明，演员与环境和演员之间的交互用 “事件” 实现。

过程是一种特殊的函数，它可以在中间搁置起来，并在某种信号下恢复运行。过程可以 `@yield` 一个事件，这时过程被暂时搁置，当它提交的事件被激发时过程从搁置的位置恢复运行。多个过程可以提交同一事件，则该事件激发时按序恢复各个过程。常见的事件是 “超时”(timeout) 事件，过程将等待或者保持当前状态指定的时间。用 `@process` 命令开始一个过程的事件激发。

过程可以要求某种资源，并在资源被其它过程占用时搁置起来等待资源，一旦资源空闲就自动占用资源并恢复运行。

作为例子，下面是 M/M/1 问题的 SimJulia 模拟实现。

```
using ResumableFunctions, SimJulia, Distributions

# 顾客行为，定义一个过程。
@resumable function customer(
    env::Environment, # 模拟环境
    arrival_time,     # 实际到达时间
    server::Resource, # 要求的柜员
    id::Integer,      # 编号
    dist_serve::Distribution) # 服务时间分布

    # 声明保存统计结果的全局变量
    global time_arr, time_ser, time_lea

    # 顾客到达事件被触发，显示到达信息
    t0 = now(env)
    DEBUG && println("Customer $id arrived: ", t0)

    # 请求柜员资源，如果没有空闲就搁置并等待
    @yield request(server)

    # 获得空闲柜员，恢复运行
    t1 = now(env)
    DEBUG && println("Customer $id entered service: ", t1)
    # 占用柜员随机的服务时间
    @yield timeout(env, rand(dist_serve))

    # 释放柜员资源，终止服务，离开银行
    @yield release(server)
    t2 = now(env)
    DEBUG && println("Customer $id exited service: ", t2)

    # 将统计分析需要的数据保存到全局变量数组
    push!(time_arr, t0)
```

```
    push!(time_ser, t1)
    push!(time_lea, t2)
end

# 这也是一个过程，用来不停地生成顾客
@resumable function customer_generator(
    env::Environment,
    T::Number, # 结束时间
    server::Resource,
    arrival_dist,
    service_dist)

    i = 0 # 顾客编号初值
    arrival_time =  0.0 # 实际到达时间，累计计算
    while true # 不停地生成顾客
        i += 1
        # 生成一个到达间隔时间
        tint = rand(arrival_dist)
        # 注意 arrival_time 是各个顾客的实际到达时间而不是间隔时间
        arrival_time += tint
        # 提交一个等待事件，等待过后才添加一个新顾客
        @yield timeout(env, tint)
        # 添加一个新顾客和此新顾客的服务事件
        proc = @process customer(env,
          arrival_time, server, i, service_dist)
    end
end

# 模拟主控程序。
function mm1_sim(  = 1.0,   = 1.2, T=1_000)
    # 随机数分布
    arrival_dist = Exponential(1 /  )
    service_dist = Exponential(1 /  )
```

```
    sim = Simulation() # 生成一个模拟控制环境
    server = Resource(sim, 1) # 柜员资源

    # 顾客生成来源也是一个过程，不停地生成顾客
    @process customer_generator(sim, T,
        server, arrival_dist, service_dist)

    # 实际运行模拟直至时间 T
    run(sim, T)
end

# 比较模拟的平均等待时间与理论期望等待时间
function mm1_stat( = 1.0,  = 1.2)
    mwait = 1 / ( - )
    arrs = [time_arr, time_ser, time_lea]
    n = minimum(length, arrs)
    arrs = [x[1:n] for x in arrs]
    time_stay = arrs[3] .- arrs[1]
    sampavg = mean(time_stay)
    println(" 理论期望等待时间 =$(mwait)")
    println(" 模拟估计 =$(sampavg)")
end

# 利用全局变量保存结果
time_arr = []
time_ser = []
time_lea = []
DEBUG = false

mm1_sim(0.2, 0.25, 100_000)
mm1_stat(0.2, 0.25)
## 理论期望等待时间 =20.000000000000004
```

```
## 模拟估计 =20.8505195476143 72
```

15.4 附录：M/M/1 系统事件模拟算法的 R 语言程序 (*)

例15.2可以用 R 语言直接按照事件模拟法实现。

这是单服务员、泊松到来、指数服务时间、先进先出服务系统。设 T0 为开始计入统计的时间，T1 为结束模拟的时间，lambda 为顾客到来速率，mu 服务结束速率。

```
demo.mm1 <- function(T0=0, T1=10000, lambda=1, mu=1.2, DEBUG=FALSE){
  ## 初始化
  t <- 0      # 时钟
  B <- FALSE # 柜员忙标志
  L <- 0      # 排队等候人数
  i <- 0      # 当前到达顾客号
  j <- 0      # 正在接受服务顾客号
  n <- 0      # 已结束服务顾客数
  max_events <- T1/lambda*2 # 设置一个较大的时间集合存储空间
  a <- numeric(max_events) # 顾客到来时间的集合
  s <- numeric(max_events) # 顾客开始服务时间的集合
  e <- numeric(max_events) # 顾客结束服务时间的集合
  X <- rexp(1, lambda); A <- X # 下一顾客到来时间
  E <- NA          # 下一个服务结束时间

  ## 时钟反复跳到下一事件发生的时间，并处理事件，更新待发生事件集合
  repeat{
    if(DEBUG){
      cat('\n')
      cat('t=', round(t,2), ' B=', as.integer(B), ' L=', L,
          ' i=', i, ' A=', round(A,2),
          ' n=', n, ' E=', round(E,2),
```

```
        ' j=', j,
        '\n', sep='')
    cat('到达序列', round(a,2), '\n')
    cat('开始序列', round(s,2), '\n')
    cat('结束序列', round(e,2), '\n')
  }
  if(!B || (B && A<E)){ # 不忙，或待处理下一到来事件
    t <- A
  } else {
    t <- E
  }
  if(t > T1) break

  if(t == A){ # 待处理到达事件
    L <- L+1
    i <- i+1; a[i] <- t
    X <- rexp(1, lambda); A <- t + X
    if(!B){
      B <- TRUE
      L <- L-1
      j <- j+1; s[j] <- t
      Y <- rexp(1, mu); E <- t + Y
    } # if !B
  } else { # 待处理结束服务事件
    B <- FALSE
    n <- n+1; e[n] <- t
    if(L > 0){
      L <- L-1
      B <- 1
      j <- j+1; s[j] <- t
      Y <- rexp(1, mu); E <- t + Y
    }
  } # end of 待处理结束服务事件
```

```
  } # end of repeat

  nn <- min(c(i, j, n))
  a <- a[seq(nn)]
  s <- s[seq(nn)]
  e <- e[seq(nn)]
  I <- a >= T0 & a <= T1
  ER <- mean(e[I] - a[I])
  ER.true <- 1/(mu-lambda)
  cat('估计的平均滞留时间 ER=', ER,
      ' 期望值 =', ER.true, '\n')
  c(ER.hat=ER, ER.true=ER.true)
}
```

测试运行：

```
set.seed(1)
demo.mm1(T1=100000, DEBUG=FALSE)
## 估计的平均滞留时间 ER= 4.783766 期望值 = 5
## ER.hat ER.true
## 4.783766 5.000000
```

※※※※※

15.5 附录：M/M/1 系统事件模拟算法的 Julia 程序 (*)

```
using Random, Distributions, Statistics

# 定义一个专用的数据类型表示模拟
abstract type DESSim end
abstract type DESState end
```

```
struct DESMM1 <: DESSim
     ::Real # 顾客到来速率
     ::Real # 服务结束速率
    make_arrival # 抽取一个到来间隔时间
    make_service # 抽取一个服务持续时间

    function DESMM1( ,  )
        ma = () -> rand(Exponential(1/ ))
        ms = () -> rand(Exponential(1/ ))
        new( ,  , ma, ms)
    end
end

mutable struct MM1State <: DESState
    model::DESMM1
    t::Real # 当前时钟
    B::Bool # 柜员忙
    L::Integer # 当前排队人数
    i::Integer  # 最新到来顾客序号
    j::Integer  # 正在服务顾客序号
    n::Integer  # 已服务顾客人数
    A::Real  # 下一顾客到来时间
    E::Real  # 下一顾客结束时间
    arr_arrival::Vector{Real} # 各位顾客的到来时间数组
    arr_service::Vector{Real} # 各位顾客的服务开始时间数组
    arr_depart::Vector{Real} # 各位顾客的离开时间数组

    function MM1State( ,  )
        model = DESMM1( ,  )
        A = model.make_arrival()
        E = A + model.make_service()
```

```
        new(model,
            0, # t
            false, # B
            0, # L
            0, # i
            0, # j
            0, # n
            A,
            E,
            [], # 到来时间数组
            [], # 服务开始时间数组
            []  # 离开时间数组
             )
    end
end

# 处理到达事件
function arrival_event!(state::MM1State)
    state.L += 1 # 排队人数
    state.i += 1 # 最新到来顾客编号
    # 记录当前新到来顾客的到来时间
    push!(state.arr_arrival, state.t)
    # 生成下一个顾客，但暂不处理此顾客
    state.A = state.t + state.model.make_arrival()

    # 处理与 i 对应的顾客
    if !state.B # 可以直接服务第 i 顾客
        state.B = true # 正在服务
        state.L -= 1 # 排队人数
        state.j += 1 # 正在接受服务顾客编号
        # 记录当前服务顾客服务开始时间
        push!(state.arr_service, state.t)
        # 生成正在服务顾客的离开时间
```

```
        state.E = state.t + state.model.make_service()
    end
    # 可以直接服务时处理服务问题,
    # 如果不可以直接服务则等待下一事件

    return state
end # function arrival_event!

# 处理离开事件
function departure_event!(state::MM1State)
    state.B = false  # 非正在服务
    state.n += 1 # 已服务顾客人数
    # 记录离开顾客的离开时间
    push!(state.arr_depart, state.t)

    # 如果有排队顾客则服务队首顾客
    if state.L > 0
        state.L -= 1
        state.B = true
        state.j += 1 # 正在服务顾客编号
        # 记录正在服务顾客服务开始时间
        push!(state.arr_service, state.t)

        # 生成正在服务顾客的离开时间, 但作为下一个事件
        state.E = state.t + state.model.make_service()
    end # if state.L > 0

    return state
end # function departure_event!


# 从当前状态向前模拟一步的函数
function step!(state::MM1State)
```

```
    model = state.model
      = model.
      = model.

    if !state.B || (state.B && state.A < state.E)
        # 如果没有顾客，或者有顾客且下一事件是到来
        state.t = state.A # 下一改变状态的时间
        arrival_event!(state)
    else
        # 下一事件是结束服务
        state.t = state.E
        departure_event!(state)
    end

    return state
end # function step!


# 函数，输入一个 MM1 系统参数和要求的时间，进行模拟
# T 是要模拟的时间终点
# T0 是要抛弃的部分
function MM1_sim( =2,  =4, T=10_000; T0 = 0)
    state = MM1State( ,  ) # 初始化
    while state.t < T
        step!(state)
    end

    # 将到来时间、开始服务时间、离开时间的顾客编号对齐，
    # 删去多余内容
    arrs = [state.arr_arrival,
        state.arr_service, state.arr_depart]
    ncommon = minimum(length, arrs)
    for i in eachindex(arrs)
```

```
        arrs[i] = arrs[i][1:ncommon]
    end
    sele = fill(true, ncommon)
    for i in eachindex(arrs)
        sele .= sele .& (T0 .<= arrs[i] .<= T)
    end
    for i in eachindex(arrs)
        arrs[i] = arrs[i][sele]
    end

    return arrs
end # function MM1_sim

# 测试 M/M/1 模拟，与理论结果比较
function test_mm1()
      = 0.2
      = 0.4
    T = 100_000
    arrs = MM1_sim( ,  , T; T0=1_000)

    # 滞留时间的数组
    t_in = arrs[3] .- arrs[1]
    mt_in = mean(t_in)
    # 理论值
    par_t_in = 1/( - )
    println(" 平均滞留时间: $(mt_in), 理论值: $(par_t_in)")

    arrs
end # function test_mm1
test_mm1();
```

习题

习题 1

选择适当的编程语言实现例15.1的算法。考虑如下的变化情形:

(1) 银行 8:00 开门，17:00 关门，关门后不再允许顾客进入但已进入的顾客会服务完;
(2) 顾客到来服从非齐次的泊松分布，其速率函数 $\lambda(t)$ 为阶梯函数；
(3) 如果顾客到来后发现队太长，有一定概率离开；如果顾客等待了太长时间，有一定概率离开，这时需要求这些离开顾客的平均人数。

习题 2

设计模拟如下离散事件的算法。设某商场每天开放 L_0 时间，有扒手在该商城出没，扒手的出现服从强度 λ_1 的齐次泊松过程，出现后作案 X 时间后离开，设 X 服从对数正态分布, $\ln X \sim \mathrm{N}(\mu_2, \sigma_2^2)$。设有一个警察每隔 L_3 时间在商城巡逻 Y 时间，Y 服从 $\mathrm{Exp}(\lambda_2)$ 分布，只要扒手和警察同时出现在商城内扒手就会被抓获。模拟估计一天时间内有扒手被抓获的概率。

习题 3

设计模拟 M/M/c 随机服务系统的算法。设服务机构共有 c 个服务窗口，顾客到来服从齐次泊松过程，速率为 λ，顾客排成一队，一旦某个窗口空闲则队头的顾客接受服务，每个窗口的服务时间均服从期望为 $1/\mu$ 的指数分布。

(1) 编程模拟上述随机服务系统；
(2) 模拟估计顾客在这个服务机构的平均滞留时间 ER;
(3) 在什么条件下这个服务机构的平均滞留时间能够稳定不变？

Chapter 16

统计研究与随机模拟

现代统计学期刊发表的论文中一半以上包括随机模拟结果（也叫数值结果）。随机模拟可以辅助说明新模型或新方法的有效性，尤其是在很难获得关于有效性的理论结果的情况下让我们也能说明自己的新方法的优点。

比如，为了掌握统计量的抽样分布经常需要依靠随机模拟。在独立正态样本情况下，我们已经有样本均值和样本方差的分布，但是对其它分布 $F(x)$ 的样本，我们一般只能得到统计量 $\hat{\theta}$ 的大样本性质，在中小样本情况下很难得到 $\hat{\theta}$ 的抽样分布理论结果，随机模拟可以解决这样的问题。在抽取 $F(x)$ 的很多批样本后，得到很多个统计量 $\hat{\theta}$ 样本值，从这些样本值可以估计 $\hat{\theta}$ 的抽样分布，并研究样本量多大时 $\hat{\theta}$ 的抽样分布与大样本极限分布相符，计算估计的均方误差，等等。

下面举一个置信区间比较的例子说明统计研究中随机模拟的做法。实际应用中经常需要计算某个百分比的置信区间，比如北京市成年人口中希望房价降低的人的百分比的置信区间。

设随机抽取了样本量为 n 样本，其中成功个数为 X 个，则 $X \sim \mathrm{B}(n,p)$, p 为成功概率，要计算 p 的置信度为 $1-\alpha$ 的置信区间。记样本中成功百分比为 $\hat{p} = X/n$。假设样本量 $n \geq 30$。

当 n 较大时由中心极限定理可知

$$W = \frac{\hat{p} - p}{\sqrt{p(1-p)}/\sqrt{n}} \tag{16.1}$$

近似服从标准正态分布。因为 $\hat{p} \to p$ a.s. $(n \to \infty)$, 所以把(16.1)式分母中的 p 换成 $\hat{p}$，仍有

$$Z = \frac{\hat{p} - p}{\sqrt{\hat{p}(1-\hat{p})}/\sqrt{n}}$$

近似服从标准正态分布。于是可以构造 p 的置信度 $1-\alpha$ 的置信区间为

$$\hat{p} \pm \frac{\lambda}{\sqrt{n}}\sqrt{\hat{p}(1-\hat{p})} \tag{16.2}$$

其中 $\lambda = z_{1-\frac{\alpha}{2}}$ 是标准正态分布的 $1-\frac{\alpha}{2}$ 分位数。当 n 很大时这个置信区间应该是比较精确的，即 p 落入置信区间的实际概率应该很接近于标称的置信度 $1-\alpha$，但是在 n 不是太大的时候，p 落入置信区间的实际概率就可能与标称的置信度 $1-\alpha$ 有一定的差距，我们要用随机模拟考察 $1-\alpha, n, p$ 的不同取值下置信区间(16.2)的覆盖率。称置信区间实际包含真实参数的概率为**覆盖率**。

参数 p 的另一个置信区间, 称为 Wilson 置信区间，是直接求解关于 p 求解不等式

$$\left|\frac{\hat{p} - p}{\sqrt{p(1-p)}/\sqrt{n}}\right| \le \lambda. \tag{16.3}$$

记

$$\tilde{p} = \frac{\hat{p} + \frac{\lambda^2}{2n}}{1 + \frac{\lambda^2}{n}}, \quad \delta = \frac{\lambda}{\sqrt{n}}\frac{\sqrt{\hat{p}(1-\hat{p}) + \frac{\lambda^2}{4n}}}{1 + \frac{\lambda^2}{n}},$$

则求解(16.3)得到的 p 的置信区间为

$$\tilde{p} \pm \delta \tag{16.4}$$

我们用随机模拟来考察(16.2)和(16.4)在不同 n, p 下的覆盖率并比较这两种置信区间。

好的置信区间应该满足如下两条要求：

(1) 覆盖率大于等于置信度，两者越接近越好；
(2) 置信区间越短越好。

下面设计这个模拟试验。首先需要估算抽样次数。设总共重复试验 M 次。M 越大，对覆盖率和置信区间长度期望估计的随机误差越小。比如我们希望模拟

计算的覆盖率误差控制在 0.1% 以下 (在 95% 置信度下)。对每组样本，真值是否落入计算得到的置信区间是一个成败型试验，设真实覆盖率为 r，M 次重复试验中置信区间包含真实参数的次数为 V，则 $V \sim \mathrm{B}(M, r)$，覆盖率的估计值为 V/M，此估计的标准误差为 $\sqrt{r(1-r)}/\sqrt{M}$。如果覆盖率 r 近似等于置信区间的置信度 $1-\alpha$，则若 $1-\alpha \geq 0.8$，近似地有 $r(1-r) \leq 0.8 \times 0.2 = 0.16$, M 次试验的覆盖率估计的标准误差为 $\sqrt{0.16}/\sqrt{M} = 0.4/\sqrt{M}$, 以 2 倍标准误差作为覆盖率估计的误差大小界限（根据中心极限定理可知有 95% 的概率使得 $|\frac{V}{M} - r|$ 小于 2 倍标准误差），令 $2 \times 0.4/\sqrt{M} \leq 0.001$ 则有 $M = 640000$，这个试验重复量很大，如果程序耗时过长我们只好降低对随机误差幅度的要求。对于更复杂的问题，如果难以得到所需重复次数 M 的理论值，可以逐步增加 M，直到结果在需要的精度上不再变化为止。

其次考虑需要模拟的不同情形，一般是不同的模型参数。比较(16.2)和(16.4)要考虑多种 $(n, p, 1-\alpha)$ 组合的影响。取 $1-\alpha = 0.99, 0.95, 0.90, 0.80$ 几种。(n, p) 的值要考虑到各种不同情况，初步选取如下的一些 (n, p) 组合:

$$
\begin{aligned}
&n = 30, p = 0.1, 0.3, 0.5, 0.7, 0.9; \\
&n = 120, p = 0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95 \\
&n = 480, p = 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95, 0.99
\end{aligned}
$$

这样，我们设计了 $(n, p, 1-\alpha)$ 的 $4 \times (5+7+9) = 84$ 种不同情况，对每种情况重复试验 M 次，M 次重复模拟中每一次可以得到四个数值：

- 置信区间(16.2)是否包含参数真值 (1 为包含，0 为不包含);
- 置信区间(16.4)是否包含参数真值;
- 置信区间(16.2)的长度;
- 置信区间(16.4)的长度。

对 $(n, p, 1-\alpha)$ 的每一种情况的 M 次重复模拟进行汇总，可以得到每种情况下的如下汇总统计量：

- 置信区间(16.2)的覆盖率 $\hat{r}_1$;
- 置信区间(16.4)的覆盖率 $\hat{r}_2$;
- 置信区间(16.2)的平均长度 $\bar{l}_1$ 和长度标准差 s_1;
- 置信区间(16.4)的平均长度 $\bar{l}_2$ 和长度标准差 s_2。

实际编程时，我们先编写 84 种不同情况中的一种情况去试验，并在开始时先试验较少的重复数，比如 $M = 100$。下面是对一种参数组合进行模拟 R 程序例子：

```
confp.sim1 <- function(n=30, p=0.3, gam=0.95, M=100){
  lam <- qnorm((1+gam)/2)
  hatp <- rbinom(M, n, p) / n
  rad1 <- lam * sqrt(hatp*(1-hatp)) / sqrt(n)
  tildep <- 0.5*lam^2 / (n + lam^2) + n*hatp / (n + lam^2)
  rad2 <- (lam / sqrt(n) *
             sqrt( hatp * (1 - hatp) + lam^2/(4*n))/
             (1 + lam^2/n))
  cov1 <- mean(abs(p - hatp) <= rad1)
  ncov1 <- M*cov1
  cov2 <- mean(abs(p - tildep) <= rad2)
  ncov2 <- M*cov2
  len1 <- 2*mean(rad1)
  sd1 <- 2*sd(rad1)
  len2 <- 2*mean(rad2)
  sd2 <- 2*sd(rad2)

  c(cover1=cov1, cover2=cov2,
    n1=ncov1, n2=ncov2,
    len1=len1, len2=len2,
    sd1=sd1, sd2=sd2)
}
```

当程序检查无误后，再逐步增大重复次数看计算时间和存储能力是否可以接受。对此例发现 $M = 640000$ 可行。

最后，用循环处理所有 84 种情况, 每种情况重复试验 $M = 640000$ 次。试验程序如下：

```r
demo.sim.confp <- function(M=640000){
  t0 <- proc.time()[3]
  ncomb <- 84
  resm <- matrix(0, nrow=ncomb, ncol=10)
  colnames(resm) <-
    c('i', 'confidence', 'n', 'p',
      'cover1', 'cover2',
      'len1', 'sd1', 'len2', 'sd2')
  resm[,'i'] <- seq(ncomb)
  resm[,'confidence'] <- rep(c(0.99, 0.95, 0.90, 0.80), each=21)
  resm[, 'n'] <- rep(rep(c(30, 120, 480), c(5, 7, 9)), 4)
  resm[, 'p'] <- rep(c(0.1, 0.3, 0.5, 0.7, 0.9,
                       0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95,
                       0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95, 0.99), 4)

  for(ii in seq(ncomb)){
    cat('组合号码 =', ii, '\n')
    gam <- resm[ii, 'confidence']
    n <- resm[ii, 'n']
    p <- resm[ii, 'p']

    res <- confp.sim1(n=n, p=p, gam=gam, M=M)
    resm[ii, 'cover1'] <- res['cover1']
    resm[ii, 'cover2'] <- res['cover2']
    resm[ii, 'len1'] <- res['len1']
    resm[ii, 'len2'] <- res['len2']
    resm[ii, 'sd1'] <- res['sd1']
    resm[ii, 'sd2'] <- res['sd2']
    save(res, file=" 置信区间比较模拟中间结果.RData")
  }

  t1 <- proc.time()[3]
  tt <- t1 - t0
```

```
  cat(" 重复次数 M=", M, " 花费时间 =", tt, "\n")

  resm
}
```

实际执行：

```
tab.stat <- demo.sim.confp(M=640000)
save(tab.stat, file=" 置信区间结果表.RData")
```

最后的结果汇总成表表格。表中每一行是一种 $(1-\alpha, n, p)$ 的组合。注意，$\bar{l}_1$ 和 $\bar{l}_2$ 的精度可以用其标准误差 $\mathrm{SE}_1 = s_1/\sqrt{M}$ 和 $\mathrm{SE}_2 = s_2/\sqrt{M}$ 来估计。

结果如下：

```
library(tidyverse)
for(j in 5:10) tab.stat[,j] <- round(tab.stat[,j], 3)
knitr::kable(tab.stat, longtable=TRUE)
```

i	confidence	n	p	cover1	cover2	len1	sd1	len2	sd2
1	0.99	30	0.10	0.957	0.992	0.265	0.083	0.287	0.045
2	0.99	30	0.30	0.968	0.992	0.422	0.038	0.391	0.027
3	0.99	30	0.50	0.984	0.995	0.462	0.012	0.420	0.008
4	0.99	30	0.70	0.967	0.991	0.422	0.038	0.390	0.027
5	0.99	30	0.90	0.957	0.992	0.265	0.083	0.287	0.045
6	0.99	120	0.05	0.942	0.993	0.100	0.020	0.109	0.016
7	0.99	120	0.10	0.983	0.991	0.139	0.018	0.142	0.015
8	0.99	120	0.30	0.986	0.988	0.214	0.009	0.210	0.008
9	0.99	120	0.50	0.986	0.992	0.234	0.001	0.228	0.001
10	0.99	120	0.70	0.987	0.988	0.214	0.009	0.210	0.008
11	0.99	120	0.90	0.983	0.991	0.139	0.018	0.142	0.015
12	0.99	120	0.95	0.942	0.993	0.100	0.020	0.109	0.016
13	0.99	480	0.01	0.952	0.990	0.023	0.006	0.026	0.005

14	0.99	480	0.05	0.981	0.991	0.051	0.005	0.052	0.005
15	0.99	480	0.10	0.987	0.988	0.070	0.004	0.071	0.004
16	0.99	480	0.30	0.990	0.989	0.108	0.002	0.107	0.002
17	0.99	480	0.50	0.991	0.991	0.117	0.000	0.117	0.000
18	0.99	480	0.70	0.990	0.989	0.108	0.002	0.107	0.002
19	0.99	480	0.90	0.987	0.988	0.070	0.004	0.071	0.004
20	0.99	480	0.95	0.981	0.991	0.051	0.005	0.052	0.005
21	0.99	480	0.99	0.953	0.990	0.023	0.006	0.026	0.005
22	0.95	30	0.10	0.809	0.974	0.201	0.063	0.215	0.041
23	0.95	30	0.30	0.953	0.930	0.321	0.029	0.307	0.023
24	0.95	30	0.50	0.957	0.957	0.352	0.009	0.332	0.007
25	0.95	30	0.70	0.953	0.930	0.321	0.029	0.307	0.023
26	0.95	30	0.90	0.808	0.974	0.201	0.063	0.215	0.041
27	0.95	120	0.05	0.935	0.946	0.076	0.016	0.080	0.013
28	0.95	120	0.10	0.954	0.954	0.106	0.013	0.107	0.012
29	0.95	120	0.30	0.937	0.942	0.163	0.007	0.161	0.006
30	0.95	120	0.50	0.945	0.945	0.178	0.001	0.175	0.001
31	0.95	120	0.70	0.937	0.942	0.163	0.007	0.161	0.006
32	0.95	120	0.90	0.954	0.955	0.106	0.013	0.107	0.012
33	0.95	120	0.95	0.935	0.946	0.076	0.016	0.080	0.014
34	0.95	480	0.01	0.855	0.967	0.017	0.004	0.019	0.004
35	0.95	480	0.05	0.937	0.955	0.039	0.004	0.039	0.004
36	0.95	480	0.10	0.947	0.943	0.054	0.003	0.054	0.003
37	0.95	480	0.30	0.947	0.948	0.082	0.002	0.082	0.002
38	0.95	480	0.50	0.950	0.950	0.089	0.000	0.089	0.000
39	0.95	480	0.70	0.947	0.948	0.082	0.002	0.082	0.002
40	0.95	480	0.90	0.947	0.943	0.054	0.003	0.054	0.003
41	0.95	480	0.95	0.937	0.954	0.039	0.004	0.039	0.004
42	0.95	480	0.99	0.855	0.968	0.017	0.004	0.019	0.004
43	0.90	30	0.10	0.790	0.884	0.169	0.053	0.178	0.038
44	0.90	30	0.30	0.883	0.930	0.270	0.024	0.261	0.021
45	0.90	30	0.50	0.901	0.901	0.295	0.007	0.283	0.006

46	0.90	30	0.70	0.883	0.930	0.269	0.024	0.261	0.021
47	0.90	30	0.90	0.791	0.885	0.169	0.053	0.178	0.038
48	0.90	120	0.05	0.838	0.864	0.064	0.013	0.066	0.012
49	0.90	120	0.10	0.892	0.909	0.089	0.011	0.090	0.011
50	0.90	120	0.30	0.887	0.910	0.137	0.006	0.136	0.005
51	0.90	120	0.50	0.879	0.917	0.150	0.001	0.148	0.001
52	0.90	120	0.70	0.888	0.910	0.137	0.006	0.136	0.005
53	0.90	120	0.90	0.892	0.909	0.089	0.011	0.090	0.011
54	0.90	120	0.95	0.838	0.863	0.064	0.013	0.066	0.012
55	0.90	480	0.01	0.834	0.898	0.014	0.004	0.016	0.003
56	0.90	480	0.05	0.891	0.886	0.033	0.003	0.033	0.003
57	0.90	480	0.10	0.886	0.890	0.045	0.003	0.045	0.003
58	0.90	480	0.30	0.899	0.901	0.069	0.001	0.069	0.001
59	0.90	480	0.50	0.890	0.909	0.075	0.000	0.075	0.000
60	0.90	480	0.70	0.898	0.900	0.069	0.001	0.069	0.001
61	0.90	480	0.90	0.886	0.891	0.045	0.003	0.045	0.003
62	0.90	480	0.95	0.891	0.886	0.033	0.003	0.033	0.003
63	0.90	480	0.99	0.834	0.898	0.014	0.004	0.016	0.003
64	0.80	30	0.10	0.743	0.884	0.132	0.041	0.137	0.032
65	0.80	30	0.30	0.757	0.839	0.210	0.019	0.206	0.017
66	0.80	30	0.50	0.799	0.799	0.230	0.006	0.224	0.005
67	0.80	30	0.70	0.756	0.839	0.210	0.019	0.206	0.017
68	0.80	30	0.90	0.744	0.885	0.132	0.041	0.137	0.032
69	0.80	120	0.05	0.777	0.864	0.050	0.010	0.051	0.009
70	0.80	120	0.10	0.769	0.831	0.069	0.009	0.070	0.008
71	0.80	120	0.30	0.805	0.805	0.107	0.004	0.106	0.004
72	0.80	120	0.50	0.764	0.829	0.116	0.001	0.116	0.001
73	0.80	120	0.70	0.805	0.805	0.107	0.004	0.106	0.004
74	0.80	120	0.90	0.770	0.833	0.069	0.009	0.070	0.008
75	0.80	120	0.95	0.776	0.864	0.050	0.010	0.051	0.009
76	0.80	480	0.01	0.805	0.747	0.011	0.003	0.012	0.003
77	0.80	480	0.05	0.788	0.828	0.025	0.002	0.025	0.002

78	0.80	480	0.10	0.798	0.805	0.035	0.002	0.035	0.002
79	0.80	480	0.30	0.805	0.788	0.054	0.001	0.053	0.001
80	0.80	480	0.50	0.814	0.814	0.058	0.000	0.058	0.000
81	0.80	480	0.70	0.803	0.786	0.054	0.001	0.053	0.001
82	0.80	480	0.90	0.797	0.805	0.035	0.002	0.035	0.002
83	0.80	480	0.95	0.787	0.828	0.025	0.002	0.026	0.002
84	0.80	480	0.99	0.804	0.747	0.011	0.003	0.012	0.003

结果内容比较多。R 的数据选择功能可以让我们比较不同参数的影响。例如，取样本量 $n = 480$，比较两种方法在置信度、p 值下的覆盖率：

```
tab1 <- as_tibble(tab.stat) |>
  dplyr::filter(n==480) |>
  select(p, confidence, cover1, cover2) |>
  arrange(p, confidence)
knitr::kable(tab1)
```

p	confidence	cover1	cover2
0.01	0.80	0.805	0.747
0.01	0.90	0.834	0.898
0.01	0.95	0.855	0.967
0.01	0.99	0.952	0.990
0.05	0.80	0.788	0.828
0.05	0.90	0.891	0.886
0.05	0.95	0.937	0.955
0.05	0.99	0.981	0.991
0.10	0.80	0.798	0.805
0.10	0.90	0.886	0.890
0.10	0.95	0.947	0.943
0.10	0.99	0.987	0.988
0.30	0.80	0.805	0.788
0.30	0.90	0.899	0.901
0.30	0.95	0.947	0.948
0.30	0.99	0.990	0.989
0.50	0.80	0.814	0.814
0.50	0.90	0.890	0.909
0.50	0.95	0.950	0.950
0.50	0.99	0.991	0.991
0.70	0.80	0.803	0.786
0.70	0.90	0.898	0.900
0.70	0.95	0.947	0.948
0.70	0.99	0.990	0.989
0.90	0.80	0.797	0.805
0.90	0.90	0.886	0.891
0.90	0.95	0.947	0.943
0.90	0.99	0.987	0.988
0.95	0.80	0.787	0.828
0.95	0.90	0.891	0.886
0.95	0.95	0.937	0.954
0.95	0.99	0.981	0.991
0.99	0.80	0.804	0.747
0.99	0.90	0.834	0.898
0.99	0.95	0.855	0.968
0.99	0.99	0.953	0.990

可以看出 $n = 480$ 时，方法 1 在 $p = 0.95, 1 - \alpha = 0.80$ 时覆盖率仅有 0.787；方法 2 在 $p = 0.01, 0.99, 1 - \alpha = 0.80$ 时覆盖率仅有 0.747。应进一步试验，研究其原因。

另一方面，从置信区间平均长度来看，两种方法差距不大。

下面做更多的模拟来确认公式(16.4)在 $n = 480, p = 0.01$，$1 - \alpha = 0.80$ 时的缺陷。写一个简单的仅模拟计算(16.4)的覆盖率的函数：

```
confp.sim2 <- function(n=480, p=0.01, gam=0.80, M=640000){
  lam <- qnorm((1+gam)/2)
  hatp <- rbinom(M, n, p) / n
  tildep <- 0.5*lam^2 / (n + lam^2) + n*hatp / (n + lam^2)
  rad2 <- (lam / sqrt(n) *
             sqrt( hatp * (1 - hatp) + lam^2/(4*n))/
             (1 + lam^2/n))
  cov2 <- mean(abs(p - tildep) <= rad2)
  ncov2 <- M*cov2

  cov2
}
```

R 的 `replicate()` 函数可以用来做简单的重复模拟：

```
set.seed(1)
tab2 <- replicate(100, confp.sim2())
summary(tab2)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.7449  0.7462  0.7465  0.7465  0.7467  0.7477
```

可见重复了 100 次，覆盖率最高也不超过 75%，说明公式(16.4)在 $n = 480$, $p = 0.01$，$1 - \alpha = 0.80$ 时不能达到标称的置信度，实际低了 5 个百分点以上。从结果也可以看出我们取 $M = 640000$ 可以使得覆盖率估计基本达到 0.001 精度。

习题

习题 1

设 $X \sim \mathrm{b}(1,p)$, $X_1, X_2, \ldots, X_n$ 为样本。令 $S_0 = \sum_{i=1}^n X_i$, $\hat{p} = S_0/n = \frac{1}{n}\sum_{i=1}^n X_i$。用模拟方法比较如下五种置信区间：

(1) 利用正态近似。当 n 很大时

$$\frac{\hat{p}-p}{\sqrt{\frac{1}{n}\hat{p}(1-\hat{p})}}$$

近似服从 $\mathrm{N}(0,1)$，于是得置信区间

$$\hat{p} \pm z_{1-\frac{\alpha}{2}}\sqrt{\frac{1}{n}\hat{p}(1-\hat{p})}.$$

(2) 利用正态近似，令

$$S^2 = \frac{1}{n-1}\sum_{i=1}^n (X_i-\hat{p})^2 = \frac{n}{n-1}\hat{p}(1-\hat{p}),$$

n 很大时

$$\frac{\hat{p}-p}{\sqrt{\frac{1}{n}S^2}}$$

近似服从 $\mathrm{N}(0,1)$，于是得置信区间

$$\hat{p} \pm z_{1-\frac{\alpha}{2}}\sqrt{\frac{1}{n}S^2}.$$

(3) Wilson 置信区间。利用正态近似，n 很大时

$$\frac{\hat{p}-p}{\sqrt{\frac{1}{n}p(1-p)}}$$

近似服从 $\mathrm{N}(0,1)$，解关于 p 的不等式

$$\left|\frac{\hat{p}-p}{\sqrt{\frac{1}{n}p(1-p)}}\right| \le z_{1-\frac{\alpha}{2}},$$

得置信区间 ($\lambda = z_{1-\frac{\alpha}{2}}$)

$$\frac{\widehat{p} + \frac{\lambda^2}{2n}}{1 + \frac{\lambda^2}{n}} \pm \frac{\lambda\sqrt{\frac{\lambda^2}{4n} + \widehat{p}(1-\widehat{p})}}{\sqrt{n}\left(1 + \frac{\lambda^2}{n}\right)}.$$

(4) Agresti and Coull(1998) 方法。先计算 Wilson 区间中心 ($\lambda = z_{1-\frac{\alpha}{2}}$)

$$\tilde{p} = \frac{\widehat{p} + \frac{\lambda^2}{2n}}{1 + \frac{\lambda^2}{n}},$$

取置信区间为

$$\tilde{p} \pm \lambda \frac{\sqrt{\tilde{p}(1-\tilde{p})}}{\sqrt{n}\sqrt{1 + \frac{\lambda^2}{n}}}.$$

(5) 用二项分布导出 p 的置信区间。令

$$p_1 = \left\{1 + \frac{n - S_0 + 1}{S_0} F_{1-\frac{\alpha}{2}}(2(n - S_0 + 1), 2S_0)\right\}^{-1}$$
$$p_2 = \left\{1 + \frac{n - S_0}{S_0 + 1} \frac{1}{F_{1-\frac{\alpha}{2}}(2(S_0 + 1), 2(n - S_0))}\right\}^{-1}$$

取置信区间为 $[p_1, p_2]$，其中 $F_q(n_1, n_2)$ 表示 $\mathrm{F}(n_1, n_2)$ 分布的 q 分位数。

对不同 $n, p, 1-\alpha$，比较这五种置信区间的优劣。

习题 2

不同的检验法有不同的功效，在难以得到功效函数的显式表达式的时候，模拟方法可以起到重要补充作用。

对无截距项的回归模型

$$y_i = bx_i + \varepsilon_i, \quad i = 1, 2, \ldots, n$$
$$\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n \text{iid} \sim \mathrm{N}(0, \sigma^2)$$

为检验 $H_0 : b = 0$，有如下两种检验方法：

(1) $b=0$ 时 $y_i=\varepsilon_i$, 于是在 H_0 下

$$t=\frac{\bar{y}}{\sqrt{\frac{1}{n}\frac{1}{n-1}\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

服从 $\mathrm{t}(n-1)$ 分布。设 λ 为 $\mathrm{t}(n-1)$ 分布的 $1-\frac{\alpha}{2}$ 分位数，取否定域为 $\{|t|>\lambda\}$。

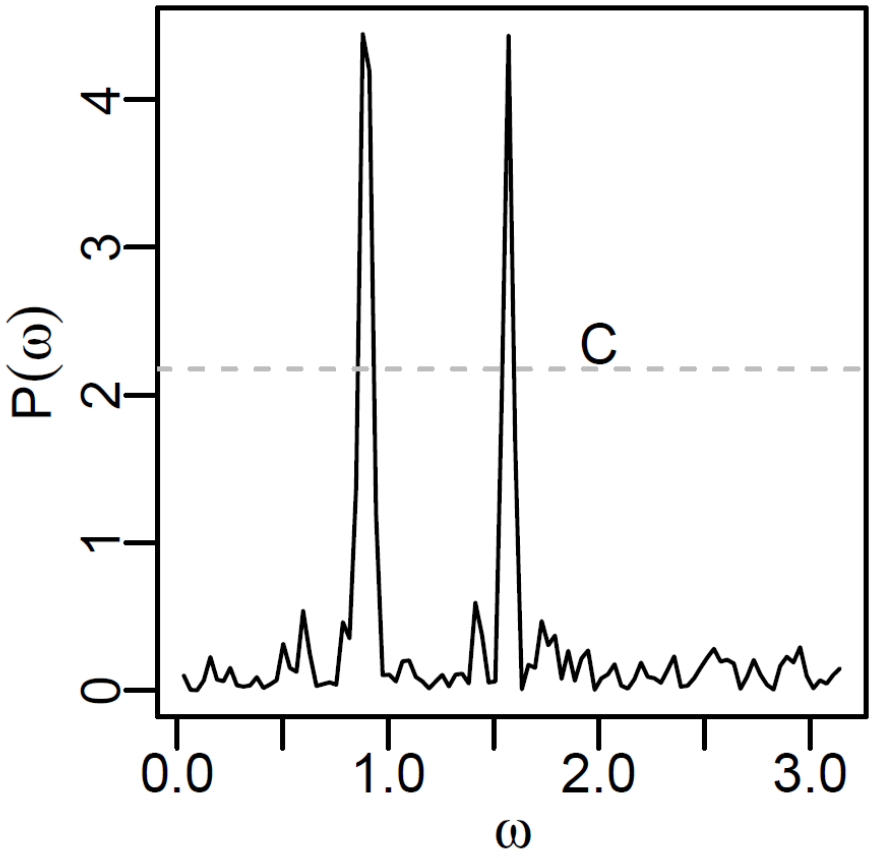
(2) 令

$$\begin{aligned}\hat{b}=&\frac{\sum_{i=1}^{n}x_iy_i}{\sum_{i=1}^{n}x_i^2}, \quad U=\hat{b}^2\sum_{i=1}^{n}x_i^2\\ Q=&\sum_{i=1}^{n}y_i^2-U, \quad F=\frac{U}{Q/(n-1)}\end{aligned}$$

设 λ' 为 $\mathrm{F}(1,n-1)$ 的 $1-\alpha$ 分位数，取否定域为 $\{F>\lambda'\}$。

对不同的 b,σ^2,n,α 以及不同的 $\{x_i\}$ 模拟比较这两种检验方法的功效。

习题 3



设时间序列 $\{y_t\}$ 有如下模型:

$$y_t = \sum_{k=1}^{m} A_k \cos(\lambda_k t + \phi_k) + x_t,\ t = 1, 2, ...$$

其中 x_t 为线性平稳时间序列，$\lambda_k \in (0, \pi)$, $k = 1, 2, ..., m$。这样的模型称为潜周期模型。如果有 $\{y_t\}$ 的一组样本 $y_1, y_2, ..., y_n$, 可以定义周期图函数 (如上图)

$$P(\omega) = \frac{1}{2\pi n} \left| \sum_{t=1}^{n} y_t e^{-it\omega} \right|^2,\ \omega \in [0, \pi].$$

在 λ_j 的对应位置 $P(\omega)$ 会有尖峰，而且当 $n \to \infty$ 时尖峰高度趋于无穷。

如下算法可以在 n 较大时估计 m 和 $\{\lambda_k\}$：首先，对 $\omega_j = \pi j/n$, $j = 1, 2, \dots, n$ 计算 $h_j = P(\omega_j)$，求 $\{h_j, j = 1, 2, \dots, n\}$ 的 3/4 分位数记为 q。令 $C = qn^{1/2}$，以 C 作为分界线，设 $\{h_j\}$ 中大于 C 的下标 j 的集合为 J, 当 J 非空时，把 J 中相邻点分入一组，但是当两个下标的差大于等于 $n^{3/4}$ 时就把后一个点归入新的一组。在每组中，以该组的 h_j 的最大值点对应的角频率 $j\pi/n$ 作为潜频率 $\{\lambda_k\}$ 中的一个的估计。

用如下 R 程序可以模拟生成一组 $\{y_t\}$ 的观测数据:

```
set.seed(1); n <- 100; tt <- seq(n)
m <- 2; lam <- 2*pi/c(4, 7); A <- c(1, 1.2)
y <- A[1]*cos(lam[1]*tt) + A[2]*cos(lam[2]*tt) + rnorm(n)
```

编写 R 程序:

(1) 编写计算 $P(\omega)$ 的函数，输入 $y = (y_1, y_2, \dots, y_n)^T$ 和 $\omega = (\omega_1, \omega_2, \dots, \omega_n)^T$, 输出 $(P(\omega_1), P(\omega_2), \dots, P(\omega_s))$。
(2) 用 R 函数 fft 计算 $h_j = P(\pi j/n), j = 1, 2, \dots, n$。
(3) 对输入的时间序列样本 $y_1, y_2, \dots, y_n$，编写函数用以上描述的算法估计 m 和 $\{\lambda_j, j = 1, 2, \dots, m\}$。

习题 4

设 $p(x, \mu, \sigma)$ 表示 $\mathrm{N}(\mu, \sigma^2)$ 的分布密度函数，设总体 X 分布密度为

$$f(x) = \lambda p(x, \mu, \sigma) + (1 - \lambda)p(x, \mu, k\sigma)$$

其中 $0 < \lambda < 1$，$k > 1$。这称为混合正态分布。从总体中抽取简单随机样本 $X_1, X_2, \dots, X_n$，对参数 μ 进行估计，有两种估计方法：

(1) 样本均值 $\bar{x}$；
(2) 取 $0 < p < 0.5$，计算删去最小的 p 比例和最大的 p 比例的样本值之后的样本平均值，记作 $\bar{x}_p$，称为 Winsorized mean。

这两个估计都是无偏的，哪一个的方差更小？理论上不容易解决。用随机模拟方法研究。需要设定一些不同的 μ, σ, λ, k, p, n 的值，估计需要的重复次数 N，比较不同情况下两种方法的估计方差。

习题 5

分位数估计。设总体 X 有密度函数 $p(x)$ 和分布函数 $F(x)$，一组简单随机样本为 $X_1, X_2, \dots, X_n$，次序统计量为 $X_{(1)}, X_{(2)}, \dots, X_{(n)}$。为估计 p 分位数 $x_p = F^{-1}(p)$，最简单的方法是 $X_{([np])}$，其中 $[np]$ 表示 np 向下取整。

更好的做法是用 Beta 分布作为线性组合系数，对次序统计量作线性组合估计 x_p。记 $G(x;p)$ 为 Beta$((n+1)p, (n+1)(1-p))$ 分布的分布函数，取 x_p 估计为

$$\tilde{x}_p = \sum_{i=1}^{n} [G(i/n;p) - G((i-1)/n;p)] X_{(i)}$$

称为 Harrell-Davis 估计量。

$p = \frac{1}{2}$ 时 x_p 称为中位数。最简单的样本中位数估计是样本中位数 $\hat{m}$: 奇数个样本取中间一个，偶数个样本取中间两个的平均值。用随机模拟方法比较样本中位数与 Harrell-Davis 估计，$F(\cdot)$ 取正态分布、柯西分布、伽马分布等多种，样本量 n 取 25 和 100 两种。用随机模拟比较两种方法的均方误差、偏差、方差，取适当的模拟重复次数 N。

Chapter 17

Bootstrap 方法 (*)

17.1 标准误差

在统计建模中，伴随着参数的估计值，应该同时给出估计的 “标准误差”。设总体 $X \sim F(x,\theta)$, $\theta \in \Theta$，$\hat{\phi}$ 是总体的一个参数 ϕ 的估计量，称 $\mathrm{SE} = \sqrt{\mathrm{Var}(\hat{\phi})}$ 为 $\hat{\phi}$ 的**标准误差**。实际工作中 SE 一般是未知的，SE 的估计也称为 $\hat{\phi}$ 的标准误差。对有偏估计，除了标准误差外我们还希望能够估计偏差。进一步地，我们还可能希望得到统计量 $\hat{\phi}$ 的分布，称为抽样分布。

例 17.1. 设 $X_i, i = 1, \ldots, n$ 是总体 $X \sim F(x)$ 的样本, 样本平均值 $\hat{\phi} = \bar{X} = \frac{1}{n}\sum_i X_i$ 为 $\phi = EX$ 的点估计，$\mathrm{SE}(\bar{X}) = \sqrt{\mathrm{Var}(X)/n}$，可以用 $S/\sqrt{n}$ 估计 SE(S^2 为样本方差)。根据中心极限定理和强大数律，当样本量 n 较大时可以取 EX 的近似 95% 置信区间为 $\bar{X} \pm 2\,\mathrm{SE}(\bar{X})$。

例 17.2. 线性模型参数估计的 SE。

考虑线性模型中参数估计的精度。设模型为

$$Y = X\beta + \varepsilon,$$

其中 $\varepsilon \sim \mathrm{N}(0, \sigma^2 I_n)$, σ^2 未知，β 是未知系数向量, X 是已知的 $n \times p$ 数值矩阵, $n > p$。在 X 列满秩时 β 的最小二乘估计为 $\hat{\beta} = (X^T X)^{-1} X^T Y$, 而 $\hat{\beta}$ 的协方差阵为 $\mathrm{Var}(\hat{\beta}) = \sigma^2 (X^T X)^{-1}$。所以，第 j 个系数 β_j 的标准误差可估计为 $\mathrm{SE}(\hat{\theta}_j) = \hat{\sigma}\sqrt{a^{(jj)}}$, 其中 $\hat{\sigma}$ 是 σ 的估计，$a^{(ij)}$ 为 $(X^T X)^{-1}$ 的 (i,j) 元素。

例 17.3. 一维参数的信息阵与 SE。

设总体 $X \sim p(x,\theta)$, $\theta \in \Theta$, $X_1, X_2, \dots, X_n$ 为 X 的简单随机样本，$\hat{\theta}$ 是真值 θ 的最大似然估计。在适当正则性条件下，$\hat{\theta}$ 渐近正态分布，渐近方差为 $\frac{1}{n}I^{-1}(\theta)$, $I(\theta)$ 为参数 θ 的信息量 (参见 [茆诗松 et al., 2006] §2.5.2 定理 2.14):

$$I(\theta) = E\left[\left(\frac{\partial \ln p(X,\theta)}{\partial \theta}\right)^2\right] = \mathrm{Var}\left(\frac{\partial \ln p(X,\theta)}{\partial \theta}\right)$$

在加强的条件下还有

$$I(\theta) = -E\left(\frac{\partial^2 \ln p(X,\theta)}{\partial \theta^2}\right)$$

可以用 $\sqrt{I^{-1}(\hat{\theta})/n}$ 估计 $\hat{\theta}$ 的 SE。

例 17.4. 多维参数的信息阵与 SE。

设总体 $X \sim p(x,\theta)$, $\theta = (\theta_1, \dots, \theta_m)$，记

$$\begin{aligned} S(\theta) =& \nabla_\theta \ln p(X,\theta) = \left(\frac{\partial \ln p(X,\theta)}{\partial \theta_1}, \dots, \frac{\partial \ln p(X,\theta)}{\partial \theta_m}\right)^T, \\ I(\theta) =& \mathrm{Var}\left(S(\theta)\right), \end{aligned}$$

称 $I(\theta)$ 为信息量矩阵，其 (i,j) 元素为

$$\mathrm{Cov}\left(\frac{\partial \ln p(X,\theta)}{\partial \theta_i}, \frac{\partial \ln p(X,\theta)}{\partial \theta_j}\right)$$

在加强的条件下 $I(\theta) = -E(\nabla_\theta^2 \ln p(X;\theta))$, $\nabla_\theta^2 \ln p(X;\theta)$ 是 $\ln p(X,\theta)$ 关于自变量 θ 的海色阵，其 (i,j) 元素为 $\frac{\partial^2 \ln p(X,\theta)}{\partial \theta_i \partial \theta_j}$。设 $X_1, X_2, \dots, X_n$ 为 X 的简单随机样本，$\hat{\theta}$ 为 θ 的最大似然估计，在适当条件下 $\hat{\theta}$ 渐近正态分布 $N(\theta, \frac{1}{n}I^{-1}(\theta))$, 可以用 $-\frac{1}{n}[\nabla_\theta^2 \ln p(X;\theta)]^{-1}$ 作为 $\mathrm{Var}(\hat{\theta})$ 的估计。

17.2 Bootstrap 方法的引入

计算参数估计的标准误差不一定总有简单的公式。例如，需要估计的参数不一定是 EX 这样的简单特征，像中位数、相关系数这样的参数估计的标准误差就比 EX 的估计的标准误差要困难得多。在线性模型估计的例子中，如果独立性、

线性或者正态分布的假定不满足则求参数估计方差阵变得很困难，比如稳健回归系数的标准误差就很难得到理论公式。在最大似然估计问题中，最大似然估计不一定总是渐近正态的，信息量有时不存在或难以计算，从而无法用上面的方法给出标准误差。

设总体 X 服从某个未知分布 $F(x)$，$X=(x_1,x_2,\dots,x_n)$ 是 X 的一个样本，ϕ 是 F 的一个参数，可以把 ϕ 看成 F 的一个泛函 $\phi(F)$，用统计量 $\hat{\phi}=g(X)$ 估计 ϕ，设 $\psi=\psi(g,F,n)$ 是统计量 $\hat{\phi}$ 的某种分布特征 ($\hat{\phi}$ 的抽样分布的数字特征)。例如 $\psi=\sqrt{\mathrm{Var}(\bar{X})}$ 为统计量 $\bar{X}$ 的标准误差，又如取 $\psi=E\hat{\phi}-\phi$ 为统计量 $\hat{\phi}$ 的偏差。可以用随机模拟的方法估计 ψ。

用随机模拟方法估计 ψ 的步骤如下。

(1) 从样本 X 估计总体分布 F 为 $\hat{F}$;
(2) 从 $\hat{F}$ 抽取 B 个独立样本 $Y^{(b)}$, $b=1,\dots,B$，每一个 $Y^{(b)}$ 样本量为 n, 称 $Y^{(b)}$ 为 bootstrap 样本。
(3) 从每个 bootstrap 样本 $Y^{(b)}$ 可以估计得到 $\hat{\phi}^{(b)}=g(Y^{(b)})$, $b=1,\dots,B$。
(4) $\hat{\phi}^{(b)}, b=1,\dots,B$ 是 $g(Y)$ 在 $\hat{F}$ 下的独立同分布样本, 可以用标准的估计方法估计关于 $g(Y)$ 在 $\hat{F}$ 下的分布特征 $\hat{\psi}=\psi(g,\hat{F},n)$, 估计结果记作 $\tilde{\psi}$，并以 $\tilde{\psi}$ 作为统计量 $\hat{\phi}$ 的抽样分布的数字特征 $\psi(g,F,n)$ 的估计值。

从样本 X 估计 $\hat{F}$ 时，可以采用参数模型，也可以采用经验分布函数 F_n。参数模型在模型正确时效率较高；经验分布法使用简单，基本不依赖于模型。从经验分布 F_n 抽样，相当于从 $X=(x_1,\dots,x_n)$ 独立有放回抽样。

估计量的标准误差可以用 bootstrap 方法估计。

例 17.5. 设 (H,W) 为某地小学五年级学生的身高和体重的总体，$(H,W)\sim F(\cdot,\cdot)$, 考虑 H 和 W 的相关系数 ϕ 的估计量的标准误差估计。

设调查了 $n=10$ 个学生的身高和体重的数据 $(h_i,w_i), i=1,2,\dots,n$:

h_i	144	166	163	143	152	169	130	159	160	175
w_i	38	44	41	35	38	51	23	51	46	51

计算得 $\hat{\phi}=g(h_1,w_1,\dots,h_n,w_n)=0.904$。令 $\mathrm{SE}(\hat{\phi})=\left[\mathrm{Var}(\hat{\phi})\right]^{1/2}=\psi(g,F,n)$。设 $\hat{F}$ 为 F 的估计，取为经验分布 F_n，则 bootstrap 方法用随机模拟方法估计 $\psi(g,F_n,n)$，然后当作 $\mathrm{SE}(\hat{\phi})$ 的估计。计算步骤如下：

(1) 从 F_n 中作 $n = 10$ 次独立抽样，即从 $\{(h_1, w_1), \dots, (h_n, w_n)\}$ 中有放回独立抽取 n 次，得到 $\hat{F} = F_n$ 的一组样本 $Y^{(1)} = ((h_1^{(1)}, w_1^{(1)}), \dots, (h_n^{(1)}, w_n^{(1)}))$;

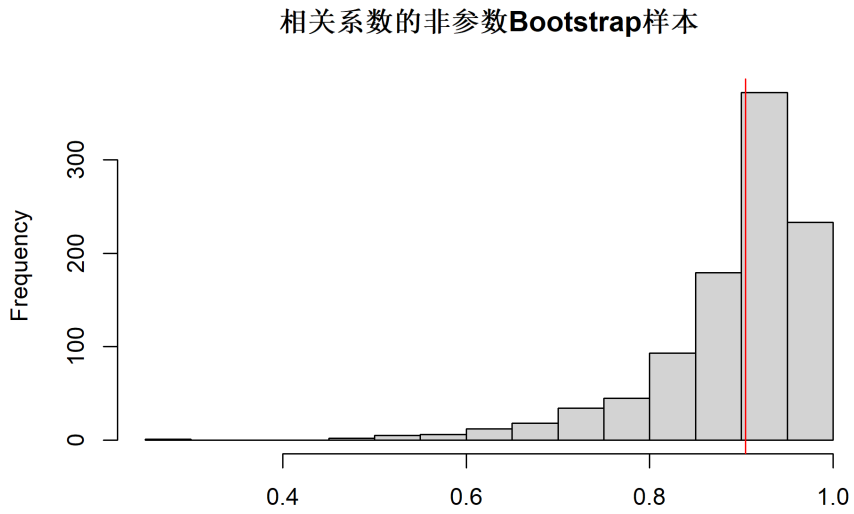
(2) 重复第 (1) 步, 直到获取了 B 组 bootstrap 样本 $Y^{(b)}, b = 1, \dots, B$;

(3) 对每一样本 $Y^{(b)}$ 计算样本相关系数 $\hat{\phi}^{(b)} = g(Y^{(b)}), b = 1, \dots, B$;

(4) 把 $\hat{\phi}^{(b)}, b = 1, \dots, B$ 作为 $\hat{F}$ 下 $n = 10$ 的样本相关系数的简单随机样本，估计其样本标准差 S，以 S 作为 $\psi(g, \hat{F}, n)$ 的估计, 进而用 S 估计 $\hat{\phi}$ 在真实的总体分布 F 下的标准误差 $\mathrm{SE}(\hat{\phi})$。

此例非参数 bootstrap 用 R 语言直接编程实现:

```
rho.boot.npar <- function(B=1000, plot.samp=FALSE){
  h <- c(144 , 166 , 163 , 143 , 152 , 169 , 130 , 159 , 160 , 175)
  w <- c(38 , 44 , 41 , 35 , 38 , 51 , 23 , 51 , 46 , 51)
  n <- length(h)
  rho0 <- cor(h, w)

  samp <- replicate(
    B, {
      ind <- sample.int(n, replace=TRUE)
      cor(h[ind], w[ind])  }  )
  if(plot.samp) {
    hist(samp, main=" 相关系数的非参数 Bootstrap 样本", xlab="")
    abline(v=rho0, col="red")
  }
  sd(samp)
}
set.seed(1)
rho.boot.npar(B=1000, plot.samp=TRUE)
```




```
## [1] 0.08671312
```

取 $B = 1000$ 的一次 bootstrap 计算得到的标准误差估计为 $S = 0.087$。当 $B \to \infty$ 时 $S \to \psi(g, F_n, n)$, 但是要注意，由于抽样误差影响，$\psi(g, F_n, n)$ 和 $\psi(g, F, n)$ 之间的误差无法避免。

R 中也提供了进行 bootstrap 估计的扩展包，比如 boot 扩展包。其中的 `boot()` 函数用来进行 bootstrap 估计，默认使用非参数方法（随机有放回抽样方法）产生 bootstrap 样本，需要输入原始数据，并输入一个用来计算统计量的函数，此统计量函数以原始数据和某一次的有放回随机抽样下标向量为输入。本例的相关系数的标准误差估计可以用如下程序计算：

```
da <- data.frame(
  h = c(144, 166, 163, 143, 152, 169, 130, 159, 160, 175),
  w = c(38,  44,  41,  35,  38,  51,  23,  51,  46,  51))
stat <- function(da, ind){
  cor(da[[1]][ind], da[[2]][ind])
}
set.seed(1)
```

```
boot::boot(data = da, statistic = stat, R = 1000)
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot::boot(data = da, statistic = stat, R = 1000)
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.9044803 -0.02006079   0.1094301
```

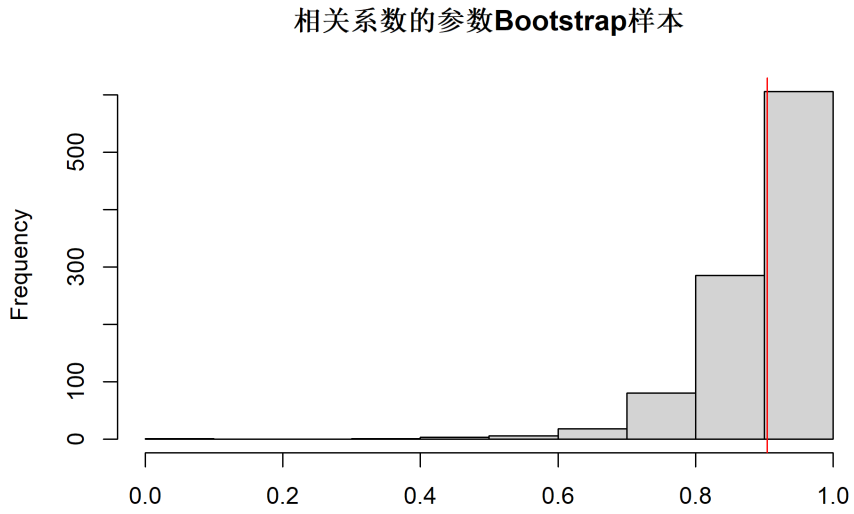
估计的标准误差为 0.100。

也可以用参数方法估计 $\hat{F}$，比如从历史经验知道总体的身高、体重服从联合正态分布，就可以按照联合正态总体模型从样本中得到参数最大似然估计后作为 $\hat{F}$ 的参数，这时 $\hat{F}$ 是一个参数确定的二元联合正态分布 $\mathrm{N}(156.1, 41.8, 13.78^2, 8.85^2, 0.904)$。从 $\hat{F}$ 中独立抽样 n 个得到一组样本，共生成 B 组这样的样本，称为 bootstrap 样本。接下来的步骤只要按照上面的 (3)、(4) 估计 $\mathrm{SE}(\hat{\phi})$ 就可以了。

参数 bootstrap 用 R 语言直接实现：

```
rho.boot.para <- function(B=1000, plot.samp=FALSE){
  h <- c(144 , 166 , 163 , 143 , 152 , 169 , 130 , 159 , 160 , 175)
  w <- c(38 , 44 , 41 , 35 , 38 , 51 , 23 , 51 , 46 , 51)
  n <- length(h)
  rho0 <- cor(h, w)
  S1 <- var(h)
  S2 <- var(w)
  Sig <- rbind(c(S1^2, rho0*S1*S2), c(rho0*S1*S2, S2^2))
  ch <- chol(Sig)
  samp <- replicate(
    B,  {
```

```
      xys <- matrix(rnorm(2*n), n, 2) %*% ch
      cor(xys[,1], xys[,2])  } )
  if(plot.samp) {
    hist(samp, main=" 相关系数的参数 Bootstrap 样本", xlab="")
    abline(v=rho0, col="red")
  }
  sd(samp)
}
set.seed(1)
rho.boot.para(B=1000, plot.samp=TRUE)
```




```
## [1] 0.08182779
```

取 $B = 1000$ 的一次参数 bootstrap 计算得到的标准误差估计为 0.082。

`boot::boot()` 也支持参数 bootstrap，这时要加 `sim="parametric"` 选项，还需要提供计算统计量的函数 `statistic`，此函数以模拟生成的数据为输入；还需要提供一个用来从参数分布生成随机样本的函数 `ran.gen`，`ran.gen()` 函

数输入原始数据以及一个列表，可以在列表中输入分布参数，而这些分布参数用 boot() 函数的 mle 自变量提供。例如：

```
get.pars <- function(da){
  rho0 <- cor(da[[1]], da[[2]])
  S1 <- var(da[[1]])
  S2 <- var(da[[2]])
  Sig <- rbind(c(S1^2, rho0*S1*S2), c(rho0*S1*S2, S2^2))
  ch <- chol(Sig)
  list(ch)
}
rg <- function(da, pars){
  ch = pars[[1]]
  n <- nrow(da)
  xys <- matrix(rnorm(2*n), n, 2) %*% ch
  xys
}
stat2 <- function(da){
  cor(da[,1], da[,2])
}
set.seed(1)
boot::boot(data = da, sim = "parametric",
  statistic = stat2,
  ran.gen = rg,
  mle = get.pars(da),
  R=200)
##
## PARAMETRIC BOOTSTRAP
##
##
## Call:
## boot::boot(data = da, statistic = stat2, R = 200, sim = "parametric",
##     ran.gen = rg, mle = get.pars(da))
##
```

```
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.9044803 -0.01153677  0.08888907
```

也可以不提供 ran.gen() 参数，而是在 statistic 函数中完成随机模拟，如：

```
stat3 <- function(da, pars){
  n <- nrow(da)
  ch <- pars[[1]]
  xys <- matrix(rnorm(2*n), n, 2) %*% ch
  cor(xys[,1], xys[,2])
}
set.seed(1)
boot::boot(data = da, sim = "parametric",
  statistic = stat3,
  R=200,
  pars = get.pars(da))
##
## PARAMETRIC BOOTSTRAP
##
##
## Call:
## boot::boot(data = da, statistic = stat3, R = 200, sim = "parametric",
##     pars = get.pars(da))
##
##
## Bootstrap Statistics :
##      original    bias    std. error
## t1* 0.7842861 0.1095859  0.08871806
```

这里利用了 boot::boot() 函数的... 参数传送了模拟需要的 pars 列表。

※※※※※

例 17.6. 考虑回归模型系数估计的标准误差计算。一般的回归模型可以写成

$$y_i = h(x_i, \beta) + \varepsilon_i, \quad i = 1, 2, \dots, n \tag{17.1}$$

其中 h 已知，β 是未知参数向量，$\{\varepsilon_i\}$ iid $F(x)$，$F(x)$ 未知, $\{x_i\}$ 是确定数值向量。可以用最小二乘等方法得到 β 的估计 $\hat{\beta} = g(y_1, \dots, y_n)$, 希望估计参数估计的协方差阵 $\mathrm{Var}(\hat{\beta})$, 协方差阵主对角线元素的平方根就是单个系数估计的标准误差。

这个模型中，未知的分布信息包括 β 和 F。β 可用 $\hat{\beta}$ 估计, F 可以用回归残差的经验分布来估计或假设一个参数模型估计模型参数。

用 bootstrap 方法估计 $\mathrm{Var}(\hat{\beta})$ 的步骤如下:

(1) 估计 β 得到 $\hat{\beta} = g(y_1, \dots, y_n)$;
(2) 计算残差 $e_i = y_i - h(x_i, \hat{\beta})$, $i = 1, \dots, n$;
(3) 对 $b = 1, \dots, B$ 重复：从 $\{e_1, \dots, e_n\}$ 中有放回独立抽取 n 次得 $\{e_i^{(b)}, i = 1, \dots, n\}$;
(4) 令 $y_i^{(b)} = h(x_i, \hat{\beta}) + e_i^{(b)}$, $i = 1, \dots, n$, $b = 1, 2, \dots, B$;
(5) 对 $b = 1, 2, \dots, B$ 重复：从 $(y_1^{(b)}, \dots, y_n^{(b)})$ 中估计 $\hat{\beta}^{(b)} = g(y_1^{(b)}, \dots, y_n^{(b)})$;
(6) 用 $\hat{\beta}^{(b)}, b = 1, \dots, B$ 的样本方差阵估计 $\mathrm{Var}(\hat{\beta})$。

※※※※※

17.3 Bootstrap 偏差校正

设 $X = (x_1, x_2, \dots, x_n)$ 为总体 $F(\cdot)$ 的样本，总体参数 $\phi = \phi(F)$ 的估计为 $\hat{\phi} = g(X)$, $\beta = E\hat{\phi} - \phi$ 为估计偏差，$\mathrm{Var}(\hat{\phi})$ 为估计方差，估计的均方误差可以分解为

$$E[\hat{\phi} - \phi]^2 = \mathrm{Var}(\hat{\phi}) + \beta^2.$$

如果 $\hat{\beta}$ 是 β 的估计，则参数 ϕ 的一个改善的估计为 $\tilde{\phi} = \hat{\phi} - \hat{\beta}$，新的估计在减小了偏差的同时一般也减小了均方误差。设 $\beta = \psi(g, F, n)$, $\hat{F}$ 是总体分布 F 的一个估计，这里 $\hat{F}$ 取为经验分布 F_n，则可以用 $\hat{\beta} = \psi(g, \hat{F}, n) = Eg(Y) - \hat{\phi}$ 来估计偏差, 其中 Y 是总体 F_n 的样本量为 n 的样本, $\hat{\phi}$ 恰好是总体分布为 F_n

时的参数 ϕ, 即 $\hat{\phi} = \phi(F_n)$。如果 $\hat{\beta}$ 不能通过理论公式计算，可以用 bootstrap 方法估计 $\hat{\beta}$，步骤如下:

(1) 从 $\{x_1, x_2, \dots, x_n\}$ 独立有放回地抽取 n 个,记为 $Y^{(1)} = (Y_1^{(1)}, Y_2^{(1)}, \dots, Y_n^{(1)})$。
(2) 重复第 (1) 步，直到获取了 B 组 bootstrap 样本 $Y^{(b)}, b = 1, 2, \dots, B$;
(3) 从每个 bootstrap 样本 $Y^{(b)}$ 可以估计得到 $\hat{\phi}^{(b)} = g(Y^{(b)})$, $b = 1, \dots, B$。
(4) 用 $\hat{\phi}^{(b)}, b = 1, \dots, B$ 作为 $g(Y)$ 在 F_n 下的独立同分布样本,
估计 $\hat{\beta} = \psi(g, F_n, n)$ 为 $\tilde{\beta} = \frac{1}{B}\sum_{b=1}^{B} \hat{\phi}^{(b)} - \hat{\phi}$。

最后，取 $\tilde{\phi} = \hat{\phi} - \tilde{b} = 2\hat{\phi} - \frac{1}{B}\sum_{b=1}^{B} \hat{\phi}^{(b)}$ 为参数 ϕ 改善的估计。

例 17.7. 设 $X \sim \mathrm{N}(\mu, \sigma^2)$, μ, σ^2 未知，$x_1, x_2, \dots, x_n$ 为 X 的样本。考虑 $\phi = \mu^2$ 的估计。

用最大似然估计法估计 ϕ 为 $\hat{\phi} = \bar{X}^2$，其中 $\bar{X}$ 为样本平均值。令 $Z = \sqrt{n}(\bar{X} - \mu)/\sigma$, 则 $Z \sim \mathrm{N}(0, 1)$。可以计算出估计偏差为

$$\beta = E\bar{X}^2 - \mu^2 = E(\mu + \frac{\sigma}{\sqrt{n}}Z)^2 - \mu^2 = \frac{\sigma^2}{n},$$

估计的均方误差为

$$\begin{aligned} L_0 =& E(\bar{X}^2 - \mu^2)^2 = E\left(\frac{2\sigma\mu}{\sqrt{n}}Z + \frac{\sigma^2}{n}Z^2\right)^2 \\ =& \frac{4\sigma^2\mu^2}{n} + \frac{3\sigma^4}{n^2}. \end{aligned}$$

估计 β 为 $\hat{\beta}_1 = S^2/n$(S^2 为样本方差)，用 $\hat{\phi}_1 = \bar{X}^2 - \hat{\beta}_1 = \bar{X}^2 - \frac{S^2}{n}$ 作为 μ^2 的改善的估计, 则 $\hat{\phi}_1$ 的均方误差比 $\hat{\phi}$ 的均方误差减小了 $\frac{n-3}{n-1}\frac{\sigma^4}{n^2}$。(设 $n > 3$)

※※※※※

如果模型更为复杂，比如，总体分布类型未知，$\hat{\beta}_1$ 这样的简单偏差估计很难得到，这种情况下可以用 bootstrap 方法进行偏差校正，步骤如下：

(1) 对 $b = 1, 2, \dots, B$ 重复: 从 $x_1, x_2, \dots, x_n$ 独立有放回地抽取 n 个，组成 bootstrap 样本 $Y^{(b)} = (y_1^{(b)}, \dots, y_n^{(b)})$;
(2) 对每个 bootstrap 样本计算 $\hat{\phi}^{(b)} = \left(\frac{1}{n}\sum_{i=1}^{n} y_i^{(b)}\right)^2$;
(3) 用 $\tilde{\phi} = 2\bar{X}^2 - \frac{1}{B}\sum_{b=1}^{B} \hat{\phi}^{(b)}$ 作为 μ^2 的改善的估计。

R 扩展包 boot 可以用来估计偏差，例如例17.5中 `boot()` 函数的输出结果中包含了相关系数估计的偏差的估计。

Jacknife 方法是另外一种对估计量的偏差和方差进行估计的方法，这种方法不需要从原来的样本重新随机抽样，而是把原来的 n 个样本点分为 r 份，每次删去其中一份后计算统计量值，利用 r 个这样的统计量值对估计量的偏差和方差进行估计。详见 [Gentle, 2002] §3.3。

17.4 Bootstrap 置信区间

枢轴量法是构造置信区间的最基本的方法。设 ϕ 是总体 $F(\cdot)$ 的一个参数, 看成 F 的一个泛函 $\phi = \phi(F)$。$X = (x_1, x_2, \dots, x_n)$ 为总体的样本，$g(X)$ 为与 ϕ 有关系的一个统计量，经常是 ϕ 的估计量。如果有变换 $W = h(g(X), \phi)$ 使得 W 的分布不依赖于任何未知参数，则设 W 的左右两侧分位数分别为 $w_{\frac{\alpha}{2}}$ 和 $w_{1-\frac{\alpha}{2}}$, 有

$$P(w_{\frac{\alpha}{2}} < h(T, \phi) < w_{1-\frac{\alpha}{2}}) = 1 - \alpha,$$

反解上面的不等式可以得到 ϕ 的置信区间。

如果对枢轴量 W 很难求分位数时，可以用 bootstrap 方法获得置信区间。设 $\hat{F}$ 为总体分布 F 的估计，设 $Y = (y_1, \dots, y_n)$ 为总体 $\hat{F}$ 的样本，$\hat{\phi} = \phi(\hat{F})$ 为与总体 $\hat{F}$ 对应的参数 ϕ 的值，实际是 ϕ 的估计值，则 $V = h(g(Y), \hat{\phi})$ 与 W 的分布相近，可以用 V 的分位数近似 W 的分位数。

例 17.8. 设总体 $X \sim F(x, \theta)$, θ 为总体的未知参数，$\phi = EX$, $X = (x_1, x_2, \dots, x_n)$ 为总体的样本，则 $g(X) = \bar{X}$ 是 ϕ 的估计，若 $W = h(g(X), \phi) = \bar{X} - EX$ 的分布与 θ 无关，求 W 的分位数 $w_{\frac{\alpha}{2}}$ 和 $w_{1-\frac{\alpha}{2}}$ 就可以构造 $\phi = EX$ 的置信区间 $(\bar{X} - w_{1-\frac{\alpha}{2}}, \bar{X} - w_{\frac{\alpha}{2}})$。

若 W 的分位数无法求得，用经验分布 F_n 作为总体分布 F 的估计，这时 $\phi(F_n) = \bar{X}$, 设 $Y = (Y_1, \dots, Y_n)$ 为 F_n 的样本，$V = h(g(Y), \bar{X}) = \bar{Y} - \bar{X}$, 这里 $\bar{X}$ 作为已知值，可以用 V 的分位数近似代替 W 的分位数。求 V 的分位数；而这只要用有放回随机抽样方法从 $x_1, x_2, \dots, x_n$ 抽取 F_n 的 B 组样本 $Y^{(b)} = (y_1^{(b)}, \dots, y_n^{(b)}), b = 1, 2, \dots, B$, 对每组样本计算平均值 $\bar{Y}^{(b)}$, 定义 $V^{(b)} = \bar{Y}^{(b)} - \bar{X}$, 用 $V^{(b)}, b = 1, 2, \dots, B$ 的样本分位数估计 V 的分位数，作为 W 的分位数 $w_{\frac{\alpha}{2}}$ 和 $w_{1-\frac{\alpha}{2}}$ 的近似。

R 的 boot 扩展包的 `boot.ci()` 函数可以从 `boot()` 函数的模拟结果中计算各种 bootstrap 置信区间。

习题

习题 1

设 $W \sim \chi^2(n-1)$，求 $\rho\left[(\frac{1}{n-1}W-1)^2,(\frac{1}{n}W-1)^2\right]$。

习题 2

对例17.7, 证明 $\hat{\phi}_1$ 的均方误差为

$$L_1 = E(\hat{\phi}_1 - \mu^2)^2 = \frac{4\sigma^2\mu^2}{n} + \frac{2n}{n-1}\frac{\sigma^4}{n^2}.$$

习题 3

考虑下的非线性回归模型 (logistic 曲线)

$$y = A(1+e^{-bx}) + \varepsilon,\ \varepsilon \sim \mathrm{N}(0,\sigma^2),$$

其中 $A>0, b>0, \sigma^2>0$ 是未知参数。设有独立样本 $(x_i,y_i), i=1,2,\ldots,n$, 可以用最小二乘方法估计 A,b,σ^2, 估计值记为 $\hat{A},\hat{b},\hat{\sigma}^2$。

(1) 设真实的 $A=10, b=1, \sigma=1$。编写程序模拟一组样本（取 $x_i=-10,-9,\ldots,9,10$），计算 $\hat{A},\hat{b},\hat{\sigma}^2$，然后用 bootstrap 方法估计 $\hat{A},\hat{b},\hat{\sigma}^2$ 的标准误差和偏差。

(2) 重复生成 N 组模型的样本，从这 N 组样本中分别得到估计值 $\hat{A}^{(j)},\hat{b}^{(j)},(\hat{\sigma}^{(j)})^2$，$j=1,2,\ldots,N$, 用得到的这些估计值作为 $\hat{A},\hat{b},\hat{\sigma}^2$ 的抽样分布的样本，估计其标准误差和偏差并与 bootstrap 方法得到的结果进行对比。

Chapter 18

置换检验 (*)

18.1 介绍

设有两个独立的总体 $X \sim F(\cdot)$ 和 $Y \sim G(\cdot)$，分别有简单随机样本 $X_1, \ldots, X_{n_1}$ 和 $Y_1, \ldots, Y_{n_2}$，$n = n_1 + n_2$。要检验如下零假设：

$$H_0 : F = G.$$

两个分布相同的检验有许多，比如，在假定两个总体都服从正态分布的情况下，可以检验其方差和均值都相等的两个假设。非参数检验方法有适用于连续分布的 Kolmogorov-Smirnov 检验。对于更一般的分布，已有的检验统计量没有零假设的理论分布或者渐近分布，无法使用已有的方法。

置换检验是基于对称性的一种计算密集型检验方法，可以利用已有的检验统计量进行检验，检验 p 值通过零假设下样本值的对称性进行计算。

18.2 肿瘤大小比较案例

我们用一个案例来讲解置换检验的做法。

例 18.1. 有 22 个肺癌病人的肿瘤大小数据，其中 10 个为腺癌，12 个为鳞癌，

数据为：

X：	13	27	33	40	43	61	125	135	161	330		
Y：	2	3	6	7	9	12	24	35	35	74	112	122

从这组数据比较这两种类型的肿瘤大小有无显著差异。

两个变量的大小比较，常用独立两样本 t 检验，但是该检验要求两个样本都服从正态分布或者样本量充分大，而这两个样本都有明显的右偏，不服从正态分布，样本量也不够大，所以如下的 t 检验给出的 p 值是不可信的：

```
dc <- data.frame(
  x = c(c(13, 27, 33, 40, 43, 61, 125, 135, 161, 330),
        c(2, 3, 6, 7, 9, 12, 24, 35, 35, 74, 112, 122)),
  type = factor(rep(c(" 腺癌", " 鳞癌"), c(10, 12)),
                levels=c(" 腺癌", " 鳞癌")))
ttc <- t.test(x ~ type, data = dc,
  var.equal=TRUE); ttc
##
##  Two Sample t-test
##
## data:  x by type
## t = 1.9433, df = 20, p-value = 0.06619
## alternative hypothesis: true difference in means between group 腺癌 and group 鳞
## 95 percent confidence interval:
##   -4.409599 124.509599
## sample estimates:
## mean in group 腺癌 mean in group 鳞癌
##              96.80              36.75
```

取零假设 H_0 为两种类型的肿瘤大小分布相同。考虑零假设成立时两样本 t 统计量的分布，t 统计量为：

$$t = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{1}{n_1+n_2-2}\left[\sum_{i=1}^{n_1}(x_i-\bar{x})^2 + \sum_{i=1}^{n_2}(y_i-\bar{y})^2\right]\left(\frac{1}{n_1}+\frac{1}{n_2}\right)}}.$$

记

$$Z_i = \begin{cases} X_i, & i = 1, 2, \dots, n_1; \\ Y_{i-n_1}, & i = n_1 + 1, \dots, n. \end{cases}$$

则在 H_0 下 $Z_i, i = 1, 2, \dots, n$ 是来自于同一总体的简单随机样本，这样，$Z_i, i = 1, 2, \dots, n$ 任意打乱次序，重新计算 t 统计量，新的 t 统计量的分布应该是不变的。因为如果仅在 $\{1, \dots, n_1\}$ 或者 $\{n_1 + 1, \dots, n\}$ 集合内部打乱次序并不改变 t 统计量的值，所以仅需考虑 $\{1, \dots, n\}$ 的 $n!$ 个排列中，$K = \binom{n}{n_1}$ 种不同的将 n 个样本值划分为前 n_1 个与后 n_2 个的划分方法，根据对称性，在 H_0 下这 K 种划分是概率相等的，每一种的概率都等于 $\frac{1}{K}$。如果我们将这 K 种划分穷举出来，每一种划分可以计算一个 t 统计量值，就可以计算 K 个 t 统计量的值 $\{t_1, \dots, t_K\}$，可以认为当零假设成立时我们从当前样本得到的 t 统计量的值是从离散均匀分布 $\{t_1, \dots, t_K\}$ 随机抽取出来的，称这个分布为**置换分布**，利用置换分布可以计算 t 统计量的双侧或者单侧 p 值。

用 T 表示服从置换分布的随机变量，定义检验的右侧 p 值为 $P(T \geq t_0)$(t_0 表示 t 统计量的值，如本例中 $t_0 = 1.9433$)，左侧 p 值为 $P(T \leq t_0)$，双侧 p 值为

$$2 \min\{P(T \leq t_0),\ P(T \geq t_0)\}.$$

如果穷举了所有 K 个置换分布的取值 $\{t_1, \dots, t_K\}$，就可以计算 p 值，比如，右侧 p 值可以计算为

$$\frac{1}{K} \sum_{j=1}^{K} I_{\{t_j \geq t_0\}},$$

其中 I_A 表示事件 A 的示性函数。

$\binom{22}{10}$ 约为 64 万种组合，虽然对本例来说可以罗列（R 的 `combn()` 函数）并进行计算，但罗列所有组合一般都计算量太大而无法完成，所以置换检验的做法一般是从所有 K 个组合中随机有放回抽取 M 个（$M < K$），然后用这 M 个对应的 t 统计量值组成一个 t 统计量的置换分布的随机样本，近似估计 p 值。比如，设抽取的 M 个组合对应的 t 统计量值为 $\{t_j, j = 1, 2, \dots, M\}$, 则双侧 p 值可以估计为

$$2 \min \left(\frac{1}{M+1} \sum_{j=0}^{M} I_{\{t_j \leq t_0\}},\ \frac{1}{M+1} \sum_{j=0}^{M} I_{\{t_j \geq t_0\}} \right).$$

这个公式中用的 $M + 1$ 是把实际样本计算出的统计量 t_0 也作为置换分布的一个随机样本点计算在内了。

R 的 sample() 函数可以从一个集合中随机无放回地抽取指定个数。如下的 R 程序重复产生 999 个服从置换分布的 t 统计量值，用置换检验计算了两样本 t 检验的双侧 p 值：

```
set.seed(1)
tsamp <- replicate(999, {
  ind <- sample(22, size=10, replace=FALSE)
  t.test(dc[["x"]][ind], dc[["x"]][-ind],
         var.equal = TRUE)$statistic })
t0 <- ttc$statistic
pvalue.2s <- 2 * min(
  mean(c(t0, tsamp) <= t0),
  mean(c(t0, tsamp) >= t0))
cat(" 置换检验 p 值 =", pvalue.2s, "\n")
## 置换检验 p 值 = 0.056
hist(tsamp)
abline(v=ttc$statistic, col="red")
```

结果得到的 p 值为 0.056，与正态假设下的 0.066 的 p 值很接近。

R 的 boot 扩展包的 boot() 函数也支持对样本进行重排列。上面的程序改用 boot::boot() 计算：

```
set.seed(1)
stat <- function(da, ind){
  n1 <- 10
  n2 <- 12
  n <- n1 + n2
  z <- da[["x"]][ind]
  t.test(z[1:n1], z[(n1+1):n],
         var.equal = TRUE)$statistic
}
bt <- boot::boot(dc, sim = "permutation",
  statistic = stat, R=999)
pvalue3 <- 2*min(mean(c(bt$t, bt$t0) <= bt$t0),
```

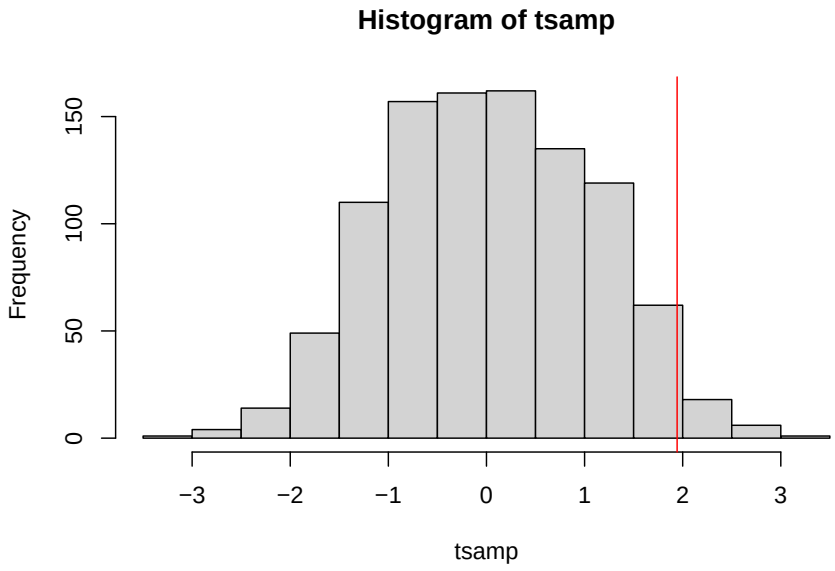


图 18.1: 置换分布与 t 统计量值

```
  mean(c(bt$t, bt$t0) >= bt$t0))
cat(" 用 boot 包计算置换检验 p 值 =", pvalue3, "\n")
## 用 boot 包计算置换检验 p 值 = 0.058
```

`boot::boot()` 函数的结果为一个列表，其中成员 `t` 是多次重复模拟每次得到的统计量值组成的向量，这是其 `statistic` 参数计算的结果；成员 `t0` 是从原始样本计算的统计量结果。加选项 `sim = "permutation"` 后 `boot::boot()` 的每次模拟就是生成样本次序的一个随机排列。

因为置换检验的 p 值是利用随机模拟计算的，所以存在随机误差。

置换检验也适用于比较两个分布的其它检验统计量的 p 值计算，包括两个多元分布的比较问题，也可以推广到多个分布的比较问题。

18.3 独立性检验

用置换检验比较两个分布是否相同，是利用了分布相同的零假设下样本次序的对称性。这种对称性也可以用来检验独立性。

设 (X,Y) 是随机向量，$X\sim F(\cdot)$, $Y\sim G(\cdot)$，有样本 (X_i,Y_i), $i=1,2,\dots,n$。要检验零假设

$$H_0: X与Y相互独立.$$

设 $\nu=(i_1,i_2,\dots,i_n)$ 是序列 $1,2,\dots,n$ 的任意一个排列，则根据对称性，(X_j,Y_{i_j}), $j=1,2,\dots,n$ 在 H_0 成立时与 (X_i,Y_i), $i=1,2,\dots,n$ 同分布，一共有 $n!$ 个这样的样本，每一个和原始样本出现概率相同。如果检验统计量 $\phi(X_1,\dots,X_n,Y_1,\dots,Y_n)$ 可以用来检验 H_0，当统计量较大时代表应拒绝 H_0，则当 H_0 成立时 $\phi(X_1,\dots,X_n,Y_1,\dots,Y_n)$ 和 $\phi(X_1,\dots,X_n,Y_{i_1},\dots,Y_{i_n})$ 同分布，一共有 $K=n!$ 个这样的统计量值，且这些统计量值概率相等。设从样本中计算得到的统计量值为 t_0，记 $\{t_1,\dots,t_K\}$ 为 $\phi(X_1,\dots,X_n,Y_{i_1},\dots,Y_{i_n})$ 的集合，则可以用置换检验法计算检验的 p 值为

$$\frac{1}{K}\sum_{j=1}^{K} I_{\{t_j\geq t_0\}}.$$

为节省计算量，可以从 $(1,2,\dots,n)$ 的所有 $K=n!$ 个排列中随机有放回地抽取 M 个，设从第 j 个随机抽取的排列计算得到的检验统计量值为 t_j，可以估计 p 值为

$$\frac{1}{M+1}\sum_{j=0}^{M+1} I_{\{t_j\geq t_0\}}.$$

独立性检验的统计量 ϕ 可以用相关系数、独立性卡方统计量等。

Chapter 19

MCMC

19.1 马氏链和 MCMC 介绍

实际工作中经常遇到分布复杂的高维随机向量抽样问题。12的重要抽样法可以应付维数不太高的情况，但是对于维数很高而且分布很复杂（比如，分布密度多峰而且位置不易确定的情况）则难以处理。

MCMC(马氏链蒙特卡洛) 是一种对高维随机向量抽样的方法，此方法模拟一个马氏链，使马氏链的平稳分布为目标分布，由此产生大量的近似服从目标分布的样本，但样本不是相互独立的。MCMC 的目标分布密度函数或概率函数可以只计算到差一个常数倍的值。MCMC 方法适用范围广，近年来获得了广泛的应用。

先介绍马氏链的概念（参考 钱敏平 et al. [2011] ）。

设 $\{X_t, t = 0, 1, ...\}$ 为随机变量序列，称为一个**随机过程**。称 X_t 为 "系统在时刻 t 的**状态**"。为讨论简单起见，设所有 X_t 均取值于有限集合 $S = \{1, 2, ..., m\}$，称 S 为**状态空间**。如果 $\{X_t\}$ 满足

$$\begin{aligned} &P(X_{t+1} = j | X_0 = k_0, ... X_{t-1} = k_{t-1}, X_t = i) \\ =&P(X_{t+1} = j | X_t = i) = p_{ij},\ t = 0, 1, ...,\ k_0, ..., k_{t-1}, i, j \in S, \end{aligned} \tag{19.1}$$

则称 $\{X_t\}$ 为**马氏链**, p_{ij} 为**转移概率**，矩阵 $P = (p_{ij})_{m\times m}$ 为**转移概率矩阵**。显然 $\sum_{j=1}^{m} p_{ij} = 1,\ i = 1, 2, ..., m$。对马氏链，$P(X_{t+k} = j | X_t = i) = p_{ij}^{(k)}$ 也

不依赖于 t，称为k **步转移概率**。如果对任意 $i, j \in S, i \neq j$ 都存在 $k \geq 1$ 使得 $p_{ij}^{(k)} > 0$ 则称 $\{X_t\}$ 为**不可约马氏链**。不可约马氏链的所有状态是互相连通的，即总能经过若干步后互相转移。对马氏链 $\{X_t\}$ 的某个状态 i, 如果存在 $k \geq 0$ 使得 $p_{ii}^{(k)} > 0$ 并且 $p_{ii}^{(k+1)} > 0$，则称 i 是**非周期**的。如果一个马氏链所有状态都是非周期的，则该马氏链称为非周期的。不可约马氏链只要有一个状态是非周期的则所有状态是非周期的。对只有有限个状态的非周期不可约马氏链有

$$\lim_{n\to\infty} P(X_n = j|X_0 = i) = \pi_j,\ i, j = 1, 2, \ldots, m,$$

其中 $\{\pi_j, j = 1, 2, \ldots, m\}$ 为正常数, 满足 $\sum_{j=1}^{m} \pi_j = 1$，称为 $\{X_t\}$ 的**极限分布**。$\{\pi_j\}$ 满足方程组

$$\begin{cases} \sum_{i=1}^{m} \pi_i p_{ij} = \pi_j,\ j = 1, 2, \ldots, m \\ \sum_{j=1}^{m} \pi_j = 1, \end{cases} \tag{19.2}$$

称满足(19.2)的分布 $\{\pi_j\}$ 为**平稳分布**或**不变分布**。对只有有限个状态的非周期不可约马氏链，极限分布和平稳分布存在且为同一分布。

如果允许状态空间 S 为可列个元素，比如 $S = \{0, 1, 2, ...\}$, 极限分布的条件需要更多的讨论。对状态 i，如从状态 i 出发总能再返回状态 i，则称状态 i 是**常返**的 (recurrent)。状态 i 常返可等价地定义为 $P(X_n = i, \text{i.o.}|X_0 = i) = 1$，其中 i.o. 表示事件发生无穷多次，$A_n$ i.o. 即 $\cap_{k=1}^{\infty} \cup_{n=k}^{\infty} A_n$。

对常返状态 i，如果从 i 出发首次返回 i 的时间的期望有限，称 i 是**正常返**的。对于不可约马氏链，只要存在正常返状态则所有状态都是正常返，这时存在唯一的平稳分布 π。非周期、正常返的不可约马氏链存在极限分布，极限分布就是平稳分布:

$$\lim_{n\to\infty} p_{ij}^{(n)} = \pi_j,\ \forall i, j \in S$$

设正常返的不可约马氏链的平稳分布为 π, 设 $h(\cdot)$ 是状态空间 S 上的有界函数，则

$$P\left(\lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n} h(X_k) = \sum_{x\in S} \pi_x h(x)\right) = 1 \tag{19.3}$$

这类似于独立同分布随机变量平均值的强大数律。当 Y 服从平稳分布 π 时，上式中的极限就等于 $Eh(Y)$。如果能设计转移概率矩阵满足正常返、不可约性质的马氏链且其平稳分布为 $\pi(\cdot)$, 则从某个初始分布出发按照转移概率模拟一列马氏链，由(19.3)可以估计关于 $Y \sim \pi(\cdot)$ 的随机变量的函数的期望 $Eh(Y)$。

如何得到有平稳分布的转移概率矩阵？一个充分条件是转移概率满足细致平衡条件。如果存在 $\{\pi_j, j \in S\}$, $\pi_j \geq 0$, $\sum_{j\in S} \pi_j = 1$, 使得

$$\pi_i p_{ij} = \pi_j p_{ji},\ \forall i \neq j,$$

称这样的马氏链为**细致平衡**的 (detailed balanced)，这时 $\{\pi_j\}$ 是 $\{X_t\}$ 的平稳分布。事实上，若 $P(X_t = i) = \pi_i, i \in S$, 则

$$P(X_{t+1} = j) = \sum_i \pi_i p_{ij} = \sum_i \pi_j p_{ji} = \pi_j \sum_i p_{ji} = \pi_j,\ \forall j \in S.$$

只要再验证转移概率矩阵 (p_{ij}) 满足不可约性和正常返性就可以利用(19.3)估计服从平稳分布的随机变量 Y 的函数的期望 $Eh(Y)$ 了。

马氏链的概念可以推广到 X_t 的取值集合 $\mathcal{X}$ 为 $\mathbb{R}^d$ 的区域的情形。如果各 $\{X_t\}$ 的有限维分布是连续型的，则(19.1)可以改用条件密度表示, 这时的 $\{X_t\}$ 按照随机过程论中的习惯应该称作马氏过程, 但这里还是叫做马氏链。

如果非周期、正常返的不可约马氏链 $\{X_t\}$ 的平稳分布为 $\pi(x), x \in \mathcal{X}$, 则从任意初值出发模拟产生序列 $\{X_t\}$，当 t 很大时，X_t 的分布就近似服从 π，抛弃开始的一段后的 X_t 序列可以作为分布 π 的相关的样本，抛弃的一段序列叫做**老化期**。如果只是为了估计 $Y \sim \pi(\cdot)$ 的函数的期望，可以仅要求马氏链为正常返不可约马氏链。设 $Y \sim \pi(\cdot)$, $h(y), y \in \mathcal{X}$ 是有界函数, 为估计 $\theta = Eh(Y)$，用 $X_t, t = k+1, \ldots, n$ 作为 π 的样本，用估计量 $\hat{\theta} = \frac{1}{n-k}\sum_{t=k+1}^{n} h(X_t)$ 来估计 θ，则 $\hat{\theta}$ 是 θ 的强相合估计。老化期长度 k 可以从 $\hat{\theta}$ 的变化图形经验地选取。

这样的估计量 $\hat{\theta}$ 是相关样本的平均值，无法用原来独立样本的公式估计 $\mathrm{Var}(\hat{\theta})$ 从而得到 $\hat{\theta}$ 的标准误差。为了估计 $\mathrm{Var}(\hat{\theta})$, 可以采用如下的分段平均法。把样本 $X_{k+1}, \ldots, X_n$ 分为 s 段，每段 r 个 (设 $n - k = sr$)。设第 j 段的 r 个 $h(X_t)$ 的平均值为 $Z_j, j = 1, 2, \ldots, s$, 设 $\{Z_j, j = 1, 2, \ldots, s\}$ 的样本方差为 $\hat{\sigma}^2 = \frac{1}{s-1}\sum_{j=1}^{s}(Z_j - \bar{Z})^2$，因为 $\hat{\theta}$ 等于 $\{Z_j, j = 1, 2, \ldots, s\}$ 的样本均值 $\bar{Z}$，当 r 足够大时，可以认为各 Z_j 相关性已经很弱，这时 $\hat{\theta}$ 的方差可以用 $\hat{\sigma}^2/s$ 估计。r 的大小依赖于不同时刻的 X_t 的相关性强弱，相关性越强，需要的 r 越大。

以上的方法就是 MCMC 方法（马氏链蒙特卡洛）。一般地，对高维或取值空间 $\mathcal{X}$ 结构复杂的随机向量 X, MCMC 方法构造取值于 $\mathcal{X}$ 的马氏链，使其平稳分布为 X 的目标分布。模拟此马氏链，抛弃开始的部分抽样值，把剩余部分作为 X 的非独立抽样。非独立抽样的估计效率比独立抽样低。

MCMC 方法的关键在于如何从第 t 时刻转移到第 $t+1$ 时刻。好的转移算法应该使得马氏链比较快地收敛到平稳分布，并且不会长时间地停留在取值空间 $\mathcal{X}$ 的局部区域内（在目标分布是多峰分布且峰高度差异较大时容易出现这种问题）。

Metropolis-Hasting 方法 (MH 方法) 是一个基本的 MCMC 算法，此算法在每一步试探地进行转移 (如随机游动)，如果转移后能提高状态 x_t 在目标分布 π 中的密度值则接受转移结果，否则以一定的概率决定是转移还是停留不动。

Gibbs 抽样是另外一种常用的 MCMC 方法，此方法轮流沿各个坐标轴方向转移，且转移概率由当前状态下用其它坐标预测转移方向坐标的条件分布给出。因为利用了目标分布的条件分布，所以 Gibbs 抽样方法的效率比 MH 方法效率更高。

19.2 Metropolis-Hasting 抽样

设随机变量 X 分布为 $\pi(x), x \in \mathcal{X}$。为论述简单起见仍假设 $\mathcal{X}$ 是离散集合。算法需要一个试转移概率函数 $T(y|x), x, y \in \mathcal{X}$，满足 $0 \leq T(y|x) \leq 1$, $\sum_y T(y|x) = 1$，并且

$$T(y|x) > 0 \Leftrightarrow T(x|y) > 0.$$

算法首先从 $\mathcal{X}$ 中任意取初值 $X^{(0)}$。设经过 t 步后算法的当前状态为 $X^{(t)}$, 则下一步由试转移分布 $T(y|X^{(t)})$ 抽取 Y，并生成 U ∼U(0,1), 然后按如下规则转移:

$$X^{(t+1)} = \begin{cases} Y & \text{如果} U \leq r(X^{(t)}, Y) \\ X^{(t)} & \text{其它.} \end{cases}$$

其中

$$r(x, y) = \min\left\{1, \frac{\pi(y)T(x|y)}{\pi(x)T(y|x)}\right\}. \tag{19.4}$$

在 MH 算法中如果取 $T(y|x) = T(x|y)$，则 $r(x, y) = \min\left(1, \frac{\pi(y)}{\pi(x)}\right)$, 相应的算法称为 **Metropolis 抽样法**。

如果取 $T(y|x)=g(y)$(不依赖于 x)，则 $r(x,y)=\min\left(1,\frac{\pi(y)/g(y)}{\pi(x)/g(x)}\right)$, 相应的算法称为 **Metropolis 独立抽样法**, 和重要抽样有相似之处, 试抽样分布 $g(\cdot)$ 经常取为相对重尾的分布。

在 MH 算法中，目标分布 $\pi(x)$ 可以用差一个常数倍的 $\tilde{\pi}(x)=C\pi(x)$ 代替，这样关于目标分布仅知道差一个常数倍的 $\tilde{\pi}(x)$ 的情形，也可以使用此算法。

下面说明 MH 抽样方法的合理性。
我们来验证 MH 抽样的转移概率 $A(x,y)=P(X^{(t+1)}=y|X^{(t)}=x)$ 满足细致平衡条件。易见

$$A(x,y)=\begin{cases}T(y|x)r(x,y), & y\neq x,\\ T(x|x)+\sum_{z\neq x}T(z|x)[1-r(x,z)], & y=x,\end{cases}$$

于是当 $x\neq y$ 时

$$\pi(x)A(x,y)=\pi(x)T(y|x)\min\left\{1,\frac{\pi(y)T(x|y)}{\pi(x)T(y|x)}\right\}=\min\left\{\pi(x)T(y|x),\pi(y)T(x|y)\right\},$$

等式右侧关于 x,y 是对称的，所以等式左侧把 x,y 交换后仍相等。所以，MH 构造的马氏链以 $\{\pi(x)\}$ 为平稳分布。多数情况下 MH 构造的马氏链也以 $\{\pi(x)\}$ 为极限分布。

(19.4)中的 $r(x,y)$ 还可以推广为如下的形式

$$\tilde{r}(x,y)=\frac{\alpha(x,y)}{\pi(x)T(y|x)},$$

其中 $\alpha(x,y)$ 是任意的满足 $\alpha(x,y)=\alpha(y,x)$ 且使得 $\tilde{r}(x,y)\leq 1$ 的函数。易见这样的 $\tilde{r}(x,y)$ 仍使得生成的马氏链满足细致平衡条件。

例 19.1. X 的取值集合 $\mathcal{X}$ 可能是很大的，以至于无法穷举，目标分布 $\pi(x)$ 可能是只能确定到差一个常数倍。

例如，设

$$\begin{aligned}\mathcal{X}=&\{x=(x_1,x_2,\dots,x_n)\ :\\ &(x_1,x_2,\dots,x_n)\text{是}(1,2,\dots,n)\text{的排列，使得}\sum_{j=1}^{n}jx_j>a\},\end{aligned}$$

其中 a 是一个给定的常数。用 $|\mathcal{X}|$ 表示 $\mathcal{X}$ 的元素个数，当 n 较大时 $\mathcal{X}$ 是 $(1,2,\dots,n)$ 的所有 $n!$ 个排列的一个子集，$|\mathcal{X}|$ 很大，很难穷举 $\mathcal{X}$ 的元素, 从而 $|\mathcal{X}|$ 未知。

设 X 服从 $\mathcal{X}$ 上的均匀分布，即 $\pi(x) = C,\ x \in \mathcal{X}, C = 1/|\mathcal{X}|$ 但 C 未知。要用 MH 方法产生 X 的抽样序列。

试抽样 $T(y|x)$ 如果允许转移到所有的 y 是很难执行的，因为 y 的个数太多了。我们定义一个 x 的近邻的概念，仅考虑转移到 x 的近邻。一种定义是，如果把 x 的 n 个元素中的某两个交换位置后可以得到 $y \in \mathcal{X}$，则 y 称为 x 的一个近邻，记 $N(x)$ 为 x 的所有近邻的集合，记 $|N(x)|$ 为 x 的近邻的个数。当 n 很大时，求 $N(x)$ 也需要从 $C_n^2 = \frac{1}{2}n(n-1)$ 个可能的元素中用穷举法选择。取试转移概率函数为

$$T(y|x) = \frac{1}{|N(x)|},\ x, y \in \mathcal{X},$$

即从 x 出发，等可能地试转移到 x 的任何一个近邻上。

因为目标分布 $\pi(x)$ 是常数，所以这时

$$r(x, y) = \min\left(1, \frac{|N(x)|}{|N(y)|}\right),$$

即从 x 试转移到 y 后，如果 y 的近邻数不超过 x 的近邻数则确定转移到 y，否则，仅按概率 $|N(x)|/|N(y)|$ 转移到 y。这就构成了对 X 抽样的 MH 算法。

Julia 实现：

```
# x: 初始排列
# nout: 需要输出的抽样次数
# n: 排列的元素个数
# bound: 即 $\sum_j j x_j \geq a$ 的界限 $a$。
function demo_permsub(x, nout::Int=100, n::Int=10, bound::Int=100)
  DEBUG = false
  sum1 = dot(1:n, x) # 要判断的条件的当前值

  ## 计算合法近邻数的函数
  ## 注意：内嵌函数内的局部变量与其所在函数的同名变量是同一变量
  function nneighb(x)
    nn = 1
    sum0 = dot(1:n, x)
    for i=1:(n-1)
```

```
    for j=(i+1):n
      sum0b = sum0 - i*x[i] - j*x[j] + i*x[j] + j*x[i]
      if sum0b >= bound
        nn += 1
      end
    end
  end
  nn
end

## 输出为 nout 行 n 列矩阵
resmat = zeros(Int, nout, n)
nsucc = 0 # 已经成功找到的元素个数
nfail = 0 # 上一次成功后的累计失败次数
ntry = 0  # 总试投次数
nn1 = nneighb(x) # 当前合法近邻数
while nsucc < nout
  ntry += 1
  DEBUG && println("ntry: ", ntry)
  nfail += 1
  if nfail > 1000
    println(" 尝试失败超过 1000 次: nsucc=", nsucc, " ntry=", ntry)
    println(x)
    break
  end
  # 进行一次尝试，随机选取两个元素下标，交换对应元素的位置
  # 看是否合法元素。注意，原本算法应该在合法近邻中抽取，这样是等价的
  I1::Int = ceil(rng()*n) # 第一个随机下标
  I2::Int = ceil(rng()*(n-1))
  if I2 >= I1
    I2 += 1 # I2 是第二个随机下标
  end
  DEBUG && println(" 尝试交换: ", I1, " ", I2, " ", x[I1], " ", x[I2])
```

```
    # 按照新的位置，计算目标的新值，比较是否超过界限
    sum2 = sum1 - I1*x[I1] - I2*x[I2] + I1*x[I2] + I2*x[I1]
    DEBUG && println("sum1: ", sum1, " sum2: ", sum2)
    if sum2 >= bound # 交换得到新的合法元素，看是否要转移
      # 先跳过去，如果不满足概率要求再跳回来
      tmp1 = x[I1]; x[I1] = x[I2]; x[I2] = tmp1
      tran = false
      nn2 = nneighb(x)
      if nn2 <= nn1
        tran = true
        DEBUG && println("nn1: ", nn1, " nn2: ", nn2)
      else
        R = rng()
        DEBUG && println("nn1: ", nn1, " nn2: ", nn2, " U(0,1): ", R)
        if R < nn1 / nn2
          tran = true
        end
      end
      if tran
        nsucc += 1
        nfail = 0
        resmat[nsucc, :] = x
        nn1 = nn2
        sum1 = sum2
      else
        tmp1 = x[I1]; x[I1] = x[I2]; x[I2] = tmp1
        nfail += 1
      end
    else
      nfail += 1
    end
  end
```

```
  resmat
end
```

测试运行：

```
srand(101)
rng = rand # 使用 Julia 官方的均匀随机数发生器
demo_permsub(collect(1:5), 10, 5, 50)
```

```
10×5 Array{Int64,2}:
 1  2  3  5  4
 1  2  3  4  5
 1  2  5  4  3
 2  1  5  4  3
 2  1  5  3  4
 1  2  5  3  4
 1  2  4  3  5
 1  2  4  5  3
 1  2  5  4  3
 1  2  3  4  5
```

```
demo_permsub(collect(1:10), 20, 10, 370)
```

```
20×10 Array{Int64,2}:
 1  2  3  4  5   9   7  8   6  10
 1  2  3  4  5  10   7  8   6   9
 1  2  3  4  5  10   7  9   6   8
 1  2  3  4  5  10   7  6   9   8
 1  2  3  4  5   9   7  6  10   8
 1  2  3  4  5  10   7  6   9   8
 1  2  3  4  5   7  10  6   9   8
 1  2  3  4  5  10   7  6   9   8
 1  2  3  4  6  10   7  5   9   8
 1  2  3  4  6   7  10  5   9   8
```

```
 1  2  3  6  4   7  10  5   9   8
 1  2  3  6  4   7  10  5   8   9
 1  2  3  6  4   5  10  7   8   9
 1  2  3  5  4   6  10  7   8   9
 1  2  5  3  4   6  10  7   8   9
 1  2  6  3  4   5  10  7   8   9
 1  2  6  3  4   5  10  8   7   9
 1  2  3  6  4   5  10  8   7   9
 1  2  3  6  4   5  10  8   9   7
 1  2  3  6  4   5  10  8   7   9
```

有了抽样后可以考察 $\mathcal{X}$ 上的均匀分布的一些问题。例如，求 X_n 的概率分布和期望值。取 $n = 100, a = 330000$，用模拟方法求解。在模拟量过大时，存储所有的结果会超出可用内存，这时一般分批模拟，下一批从上一批最后的状态开始模拟。因为模拟是从一个规则状态开始的，丢弃初始的 1 万次抽样，使用随后的 100 万次抽样的结果。

```
rng = rand # 使用 Julia 官方的均匀随机数发生器
srand(101)
n = 100
a = 330000
batch = 100 # 每批次抽取 100 个样本
nb = 10100 # 总共 101 万个抽样
nb0 = 100 # 丢弃的初始模拟样本 1 万个
tab = zeros(n) # 用来记录每个数值在 $X_n$ 出现的次数
resm = zeros(Int, batch, n) # 一个批次的存储，在各批次之间复用
x = collect(1:n)
for ib = 1:nb
  println("Batch NO. ", ib)
  resm[:,:] = demo_permsub(x, batch, n, a)
  x[:] = resm[batch, :]
  if ib > nb0 # 跳过 nb0 批次
    for j=1:batch
      tab[resm[j,n]] += 1
```

```
    end
  end
end
```

估计 X_n 的概率分布为：

```
println(tab ./ (batch*(nb-nb0)))
```

```
[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, 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, 1.5e-5, 0.0, 5.4e-5, 1.4e-5, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0e-6, 0.0,
9.2e-5, 0.0, 6.0e-6, 0.000156, 0.000253, 0.000149, 0.000307, 0.000687, 0.000853, 0.000813,
0.000644, 0.00121, 0.001393, 0.001503, 0.002198, 0.001811, 0.002581, 0.002672, 0.003203, 0.0
0.004883, 0.006127, 0.006593, 0.008228, 0.010117, 0.010797, 0.010435, 0.015785, 0.014116, 0.
0.017558, 0.021747, 0.024416, 0.026553, 0.026831, 0.028487, 0.035404, 0.032202, 0.037334, 0.
0.041615, 0.042638, 0.044291, 0.051675, 0.057559, 0.059117, 0.066057, 0.06187, 0.076104, 0.0
```

EX_n 的估计：

```
exn_est = dot(tab, (1:n)) / (batch*(nb-nb0))
exn_est
## 90.418805
```

※※※※※

19.2.1 连续型分布的 MH 抽样法

对于连续型的目标分布, 设 $\pi(x)$ 为目标分布的密度，这时 $T(y|x)$ 改为给定 x 条件下的试抽样密度，$r(x,y)$ 定义不变，算法和离散型目标分布的情形相同。

例 19.2. 考虑一个贝叶斯推断问题。在金融投资中，投资者经常把若干种证券组合在一起来减少风险。假设有 5 支股票的 $n = 250$ 个交易日的收益率记录，

每个交易日都找出这 5 支股票收益率最高的一个，设 X_i 表示第 i 支股票在 n 个交易日中收益率为最高的次数 $(i = 1, 2, \dots, 5)$。设 $(X_1, \dots, X_5)$ 服从多项分布，相应的概率假设为

$$p = \left(\frac{1}{3}, \frac{1-\beta}{3}, \frac{1-2\beta}{3}, \frac{2\beta}{3}, \frac{\beta}{3}\right),$$

其中 $\beta \in (0, 0.5)$ 为未知参数。假设 β 有先验分布 $p_0(\beta) \sim \mathrm{U}(0, 0.5)$。设 $(x_1, \dots, x_5)$ 为 $(X_1, \dots, X_5)$ 的观测值，则 β 的后验分布为

$$\begin{aligned}
&f(\beta|x_1, \dots, x_5) \propto p(x_1, \dots, x_5|\beta)p_0(\beta) \\
=&\binom{n}{x_1, \dots, x_5} \left(\frac{1}{3}\right)^{x_1} \left(\frac{1-\beta}{3}\right)^{x_2} \left(\frac{1-2\beta}{3}\right)^{x_3} \left(\frac{2\beta}{3}\right)^{x_4} \left(\frac{\beta}{3}\right)^{x_5} \frac{1}{0.5} I_{(0,0.5)}(\beta) \\
\propto&(1-\beta)^{x_2}(1-2\beta)^{x_3}\beta^{x_4+x_5} I_{(0,0.5)}(\beta) = \tilde{\pi}(\beta).
\end{aligned}$$

为了求 β 后验均值，需要产生服从 $f(\beta|x_1, \dots, x_5)$ 的抽样。从 β 的后验分布很难直接抽样，采用 Metropolis 抽样法。

设当前 β 的状态为 $\beta^{(t)}$, 取试抽样分布 $T(y|\beta^{(t)})$ 为 U(0,0.5), 则 $T(y|x) = T(x|y)$,

$$r(\beta^{(t)}, y) = \min\left(1, \frac{\tilde{\pi}(y)}{\tilde{\pi}(\beta^{(t)})}\right) = \min\left(1, \left(\frac{1-y}{1-\beta^{(t)}}\right)^{x_2} \left(\frac{1-2y}{1-2\beta^{(t)}}\right)^{x_3} \left(\frac{y}{\beta^{(t)}}\right)^{x_4+x_5}\right),$$

从 U(0, 0.5) 试抽取 y，以概率 $r(\beta^{(t)}, y)$ 接受 $\beta^{(t+1)} = y$ 即可。

※※※※※

19.2.2 随机游动 MH 算法

MH 抽样中试转移概率函数 $T(y|x)$ 较难找到，容易想到的是从 $x^{(t)}$ 作随机游动的试转移方法，叫做随机游动 Metropolis 抽样。

设 X 的目标分布 $\pi(x)$ 取值于欧式空间 $\mathcal{X} = \mathbb{R}^d$。从 $x^{(t)}$ 出发试转移，令

$$y = x^{(t)} + \varepsilon_t,$$

其中 $\varepsilon_t \sim g(x; \sigma)$ 对不同 t 是独立同分布的, $T(y|x) = g(y - x)$。设 g 是关于 $x = 0$ 对称的分布，则 $T(y|x) = T(x|y)$。

常取 g 为 $\mathrm{N}(0, \sigma^2 I)$ 和半径为 σ 的中心为 0 的球内的均匀分布。

转移法则为: 从 $x^{(t)}$ 出发试转移到 y 后，若 $\pi(y) > \pi(x^{(t)})$ 则令 $x^{(t+1)} = y$; 否则, 独立地抽取 $U \sim \mathrm{U}(0,1)$, 取

$$x^{(t+1)} = \begin{cases} y, & \text{当} U \le \pi(y)/\pi(x^{(t)}), \\ x^{(t)}, & \text{其它}. \end{cases}$$

随机游动 MH 算法是一种 Metropolis 抽样方法。随机游动的步幅 σ 是重要参数，步幅过大导致拒绝率大，步幅过小使得序列的相关性太强，收敛到平衡态速度太慢。一个建议选法是试验各种选法，使得试抽样被接受的概率在 0.25 到 0.35 之间。见 Liu [2001] § 5.4 P. 115.

例 19.3. 考虑如下的简单气体模型: 在平面区域 $G = [0, A] \times [0, B]$ 内有 K 个直径为 d 的刚性圆盘。随机向量 $X = (x_1, y_1, \ldots, x_K, y_k)$ 为这些圆盘的位置坐标。分布 $\pi(x)$ 是 G 内所有允许位置的均匀分布。希望对 π 抽样。

先找一个初始的允许位置 $x^{(0)}$。比如，把圆盘整齐地排列在左上角。

设已得到 $x^{(t)}$，随机选取一个圆盘 i，把圆盘 i 的位置试移动到 $(x_i', y_i') = (x_i + \delta_i, y_i + \epsilon_i)$，其中 δ_i, ϵ_i 独立同 $\mathrm{N}(0, \sigma^2)$ 分布。如果得到的位置是允许的则接受结果，否则留在原地不动。

下面给出 R 实现。运行参数：

```
N <- 100 # 重复抽样个数
A <- 3.5 # 摆放区域宽度
B <- 3.5 # 摆放区域高度
d <- 0.8 # 圆盘直径
K <- 6   # 圆盘个数
r <- d/2 # 圆盘半径
d2 <- d^2
sigma <- 0.5 # 随机移动的标准差
oldpar <- par(pty="s", bty="n", mar=c(0,0,1,0))
```

用来画出一个圆盘的函数。`x`, `y` 为圆心，`r` 为半径。

```
circ <- function(x,y,r,...){
  n <- 30
  theta <- seq(0, 2*pi, length=n)
  xx <- x + r * cos(theta)
  yy <- y + r * sin(theta)
  lines(xx,yy,...)
}
```

初始摆放，将所有 6 个圆盘整齐地摆放在区域的左下角位置：

```
x <- rep(seq(r+0.01, length=3, by=d+0.01),2)
y <- c(rep(r+0.01, 3), rep(3*r+0.01, 3))
plot(c(-d,A+d), c(-d,B+d), type="n", axes=FALSE,
     xlab="", ylab="")
lines(c(0,A,A,0,0), c(0,0,B,B,0), lwd=2)
for(i in 1:K) circ(x[i], y[i], r)
locator(1)
```

下面定义一个函数，对要移动的圆盘，看要移动到的新位置是否与其他圆盘矛盾。x, y 是新的圆心位置，i 是要移动的圆盘序号。

```
fit <- function(xnew, ynew, i){
  res <- xnew > r && xnew < A-r && ynew > r && ynew < B-r
  res <- res && all((x[-i] - xnew)^2 + (y[-i] - ynew)^2 > d2)
  res
}
```

其中的 x, y 将使用外部的变量值。

下面产生 $N-1$ 个新的位置。每次尝试移动，直到移动成功。

```
set.seed(1)
nfit <- 0 # 总成功次数
tt <- 0   # 总尝试次数
batch <- FALSE
```

```
while(nfit <= N){
  tt <- tt + 1
  i <- sample(1:K, 1) # 随机选一个圆盘移动
  xnew <- x[i] + rnorm(1, 0, sigma) # 产生尝试的新位置
  ynew <- y[i] + rnorm(1, 0, sigma)
  isfit <- fit(xnew, ynew, i)
  ## 画出尝试移动
  plot(c(-d,A+d), c(-d,B+d), type="n", axes=FALSE,
       xlab="", ylab="")
  text(A, -0.1*d, "BATCH") # 画简单的图上按钮
  lines(c(0,A,A,0,0), c(0,0,B,B,0), lwd=2) # 画边框
  for(j in 1:K) if(j != i) circ(x[j], y[j], r) # 画不移动的那些圆盘
  circ(x[i], y[i], r, lty=2) # 画要移动的圆盘的原始位置，虚线
  if(isfit){ # 可移动
    nfit <- nfit + 1
    circ(xnew, ynew, r, col="blue") # 画出要移动的圆盘的新位置
    x[i] <- xnew; y[i] <- ynew
  } else {
    circ(xnew, ynew, r, col="red") # 尝试的新位置是不可行的
  }
  title(main=paste("t = ", tt, "   success = ", nfit, sep=""))
  if(tt <= 40){ # 前 40 次要求用户点击
    menu <- locator(1)
  }
  if(!batch && tt > 40){ # 40 次以后，允许不再点击就继续模拟
    menu <- locator(1)
    if(menu$x > A-0.2 && menu$y < 0.2) batch=TRUE
  }
}
cat("Acception rate = ", round( N / tt * 100, 1), "%\n")
par(oldpar)
```

※※※※※

19.3 Gibbs 抽样

一般的 MH 抽样每一步首先进行尝试从上一步的状态进入一个新的状态，然后根据新的状态是否靠近目标分布来接受或拒绝试抽样点，如果拒绝，就停留在上一步的状态，效率较低。

Gibbs 抽样是另外一种 MCMC 方法，它仅在坐标轴方向尝试转移，用当前点的条件分布决定下一步的试抽样分布，所有试抽样都被接受，不需要拒绝，所以效率可以更高。

设状态用 $x = (x_1, x_2, \dots, x_n)$ 表示, 设目标分布为 $\pi(x)$, 用 $x_{(-i)}$ 表示 (x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)，假设 $\pi(\cdot)$ 的条件分布 $p(x_i|x_{(-i)})$ 都能够比较容易地抽样。

Gibbs 抽样每一步从条件分布中抽样，可以轮流从每一分量抽样，这样的算法称为系统扫描 Gibbs 抽样算法:

从$\pi(x)$的取值区域任意取一个初值$X^{(0)}$
for(t in $0 : (N-1)$){
    for(i in $1 : n$){
        从条件分布$p(x_i|X_1^*, \dots, X_{i-1}^*, X_{i+1}^{(t)}, \dots, X_n^{(t)})$抽取$X_i^*$
    }
    $X^{(t+1)} \leftarrow (X_1^*, \dots, X_n^*)$
}

从条件分布抽样的次序也可以是随机选取各个分量，这样的算法称为随机扫描 Gibbs 抽样算法:

从$\pi(x)$的取值区域任意取一个初值$X^{(0)}$
for(t in $0 : (N-1)$){
    按概率$\alpha = (\alpha_1, \dots, \alpha_n)$随机抽取下标$i$
    从条件分布$p(x_i|X_{(-i)}^{(t)})$抽取X_i^*
    $X^{(t+1)} \leftarrow (X_1^{(t)}, \dots, X_{i-1}^{(t)}, X_i^*, X_{i+1}^{(t)}, \dots, X_n^{(t)})$
}

其中下标的抽样概率 α 为事先给定。

容易看出，无论采用系统扫描还是随机扫描的 Gibbs 抽样，如果 $X^{(t)}$ 服从目标分布，则 $X^{(t+1)}$ 也服从目标分布。以系统扫描方法为例，设在第 $t+1$ 步已经抽取了 $X_1^*, \dots, X_{i-1}^*$, 令 $Y = (X_1^*, \dots, X_{i-1}^*, X_i^{(t)}, \dots, X_n^{(t)})$, 设 $Y \sim \pi(\cdot)$。下一步从 $\pi(\cdot)$ 的边缘密度 $p(x_i|X_1^*, \dots, X_{i-1}^*, X_{i+1}^{(t)}, \dots, X_n^{(t)})$ 抽取 X_i^*, 则 $Y^* = (X_1^*, \dots, X_{i-1}^*, X_i^*, X_{i+1}^{(t)}, \dots, X_n^{(t)})$ 的分布密度在 Y^* 处的值为

$$\begin{aligned}
&p(X_1^*, \dots, X_{i-1}^*, X_i^*, X_{i+1}^{(t)}, \dots, X_n^{(t)}) \\
=&p(X_i^*|X_1^*, \dots, X_{i-1}^*, X_{i+1}^{(t)}, \dots, X_n^{(t)})\, p(X_1^*, \dots, X_{i-1}^*, X_{i+1}^{(t)}, \dots, X_n^{(t)}) \\
=&\pi(X_1^*, \dots, X_{i-1}^*, X_i^*, X_{i+1}^{(t)}, \dots, X_n^{(t)})
\end{aligned}$$

即 $Y \sim \pi(\cdot)$ 则 $Y^* \sim \pi(\cdot)$。

例 19.4. 设目标分布为二元正态分布，设 $X \sim \pi(x)$ 为

$$\mathrm{N}\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \right\}$$

采用系统扫描 Gibbs 抽样方案，每一步的迭代为，

$$\begin{aligned}
&\text{抽取} X_1^{(t+1)}|X_2^{(t)} \sim \mathrm{N}(\rho X_2^{(t)},\, 1-\rho^2) \\
&\text{抽取} X_2^{(t+1)}|X_1^{(t+1)} \sim \mathrm{N}(\rho X_1^{(t+1)},\, 1-\rho^2)
\end{aligned}$$

递推可得

$$\begin{pmatrix} X_1^{(t)} \\ X_2^{(t)} \end{pmatrix} \sim \mathrm{N}\left\{ \begin{pmatrix} \rho^{2t-1} X_2^{(0)} \\ \rho^{2t} X_2^{(0)} \end{pmatrix}, \begin{pmatrix} 1-\rho^{4t-2} & \rho-\rho^{4t-1} \\ \rho-\rho^{4t-1} & 1-\rho^{4t} \end{pmatrix} \right\} \tag{19.5}$$

当 $t \to \infty$ 时，$(X_1^{(t)}, X_2^{(t)})$ 的期望与目标分布期望之差为 $O(|\rho|^{2t})$, 方差与目标分布方差之差为 $O(|\rho|^{4t})$。

例 19.5. 设目标分布为

$$\pi(x,y) \propto \binom{n}{x} y^{x+\alpha-1}(1-y)^{n-x+\beta-1},\ x = 0, 1, \dots, n,\ 0 \le y \le 1,$$

则 $X|Y \sim \mathrm{B}(n,y)$, $Y|X \sim \mathrm{Beta}(x+\alpha, n-x+\beta)$。易见 Y 的边缘分布为 $\mathrm{Beta}(\alpha,\beta)$。可以用 Gibbs 抽样方法模拟生成 (X,Y) 的样本链。

下面用 Julia 语言实现这个模拟，并用抽样样本估计 $\theta = EY$。在这个例子中，我们重点演示如何选取抽样量 N 与老化期（预热期）长度 N_1。

先取定模拟参数 ， 和 n。调入有各种分布随机数发生器的包 Distributions，与作曲线图的包 Plots:

```
# 参数:
 =5
 =2
n=10
using Distributions
using Plots
```

在下面为了度量分段平均值是否不相关，需要一个计算一阶自相关系数的函数，这里自己定义一个这样的函数：

```
function autocor1(x::Vector)
    n = length(x)
    s = 0.0
    s2 = 0.0
    sc = 0.0
    for k=1:(n-1)
        s += x[k]
        s2 += x[k]^2
        sc += x[k]*x[k+1]
    end
    xm = (s + x[n])/n # 均值
    s2 += x[n]^2
    s2 = (s2 - n*xm^2)/(n-1) # 方差
    sc = (sc - (n+1)*xm^2 + xm*(x[1]+x[n]))/(n-1) # 协方差
    scor = sc/(s2*(n-1)/n)
    scor
end
```

针对不同的总抽样长度 N，可以写出模拟生成样本的函数：

```
function simtrial(N=1000)
    x = Vector{Int}(N+1)
    y = Vector{Float64}(N+1)
    x[1] = 0
    y[1] = rand(Beta( , ))
    for k=1:N
      # 系统扫描 Gibbs 算法
      x[k+1] = rand(Binomial(n, y[k]))
      y[k+1] = rand(Beta(x[k+1]+ , n-x[k+1]+ ))
    end
    (x[2:(N+1)], y[2:(N+1)])
end
```

在生成模拟样本后，交互地确定预热期长度。一种办法是用逐步平均值作图查看需要的预热期长短。令

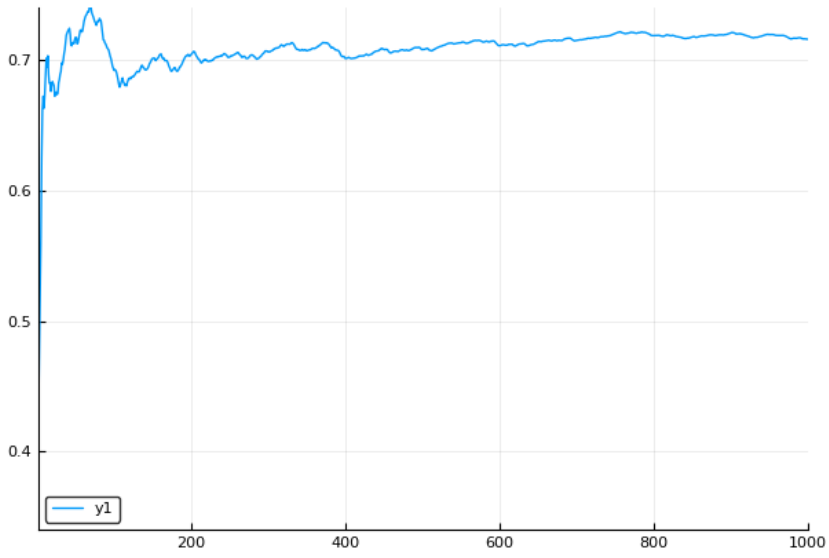
$$\bar{y}_k = \frac{1}{k}\sum_{i=1}^{k} y_i,\ k = 1, 2, \dots, N$$

为逐步平均值。当逐步平均值相比开始的时候不再剧烈震荡，就可以认为已经过了预热期。先定义一个计算序列的逐步平均序列的函数：

```
function mavg(x)
    n = length(x)
    avg = zeros(n)
    s = 0
    for k=1:n
        s += y[k]
        avg[k] = s/k
    end
    avg
end
```

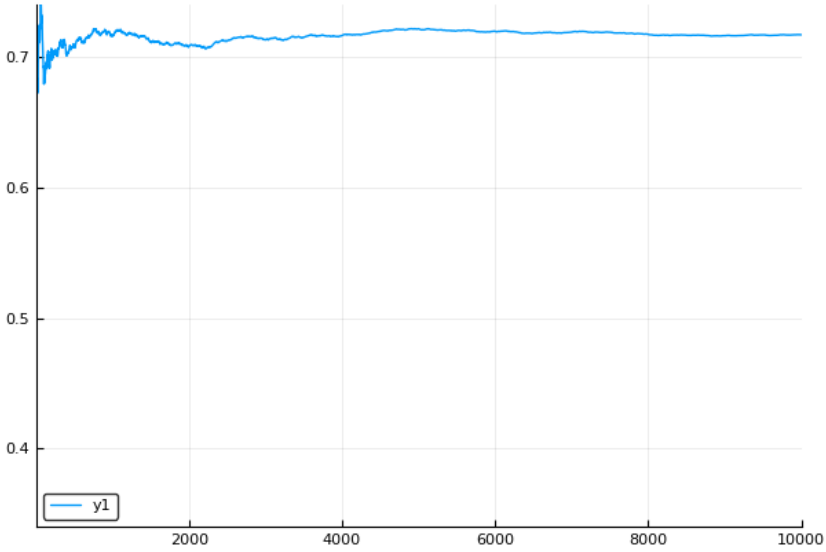
暂时取总长度 $N = 1000$ 进行模拟，然后用逐步平均值图形查看预热期长度：

```
N = 1000
srand(101)
x, y = simtrial(N)
avg = mavg(y)
plot(1:N, avg)
```



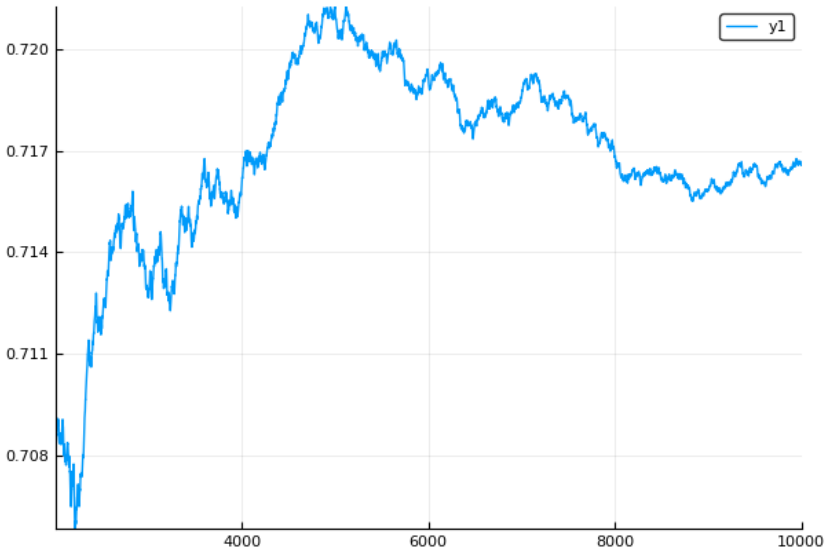
可以看出，在经过 200 次迭代后平均值已经比较稳定，但是一直到 $N = 1000$ 时，平均值都有一定的上升趋势。为了进一步确认向平稳分布的收敛，增大到 $N = 10000$:

```
N = 10000
srand(101)
x, y = simtrial(N)
avg = mavg(y)
plot(1:N, avg)
```



从这个图看出，$N = 1000$ 确实有些不足，取预热期为 $N_1 = 2000$，利用最后的 8000 点作为抽样。要注意的是，从局部看，经过预热期之后，这样的迭代平均值仍会有波动，只不过波动幅度变小了许多而已：

```
N1=2000
plot((N1+1):N, avg[(N1+1):N])
```



从上图看仍在剧烈震荡，但实际上变动幅度已经仅在 0.705 到 0.722 之间了。

现在用第 2001 到 10000 点的观测值估计 $\theta = EY$, 令 $\hat{\theta}$ 为最后 $N_2 = N - N_1$ 的 Y 样本的平均值：

```
N2 = N - N1
yout = y[(N1+1):N]
hat_ey = mean(yout)
## 0.7188284012983895
```

用 8000 个抽样给出了 EY 的一个估计。问题是，这个估计精度有多少？

将样本分成若干段，每段 r 个样本点，对每段计算平均值，在每段长度 r 足够长以后这些分段平均值应该是近似不相关的。

```
# 分段计算均值的函数
function segavg(x::Vector, r)
    # x 是输入向量
    # r 是每段长度
    n = length(x)
    m = Int(floor(n/r))
    z = Vector{Float64}(m)
    for k=1:m
        z[k] = mean(x[((k-1)*r+1):(k*r)])
    end
    z
end
```

问题是，需要多长一段？采用试验的方法。先试验 $r = 100$ 一段，利用一阶自相关系数估计段之间的相关性。

```
r = 100 # 试验使用的每段长度
m = Int(floor(N2/r)) # 段数
z = segavg(yout, r)
autocor1(z)
## 0.031074953357238148
```

现在分段均值序列长度 $m = 80$，如果序列是独立同分布列的话，由时间序列中的结论其一阶自相关系数应近似服从 $\mathrm{N}(0, \frac{1}{m})$，所有取值在 $\pm 2/\sqrt{m}$ 范围是正常的：

```
2/sqrt(m)
## 0.22360679774997896
```

所以取 $r = 100$ 为一段是可以的。但是，分段均值独立条件的上述标准误差界太大了，提示我们模拟长度可能需要更长才可以做出比较可信的独立性推断。

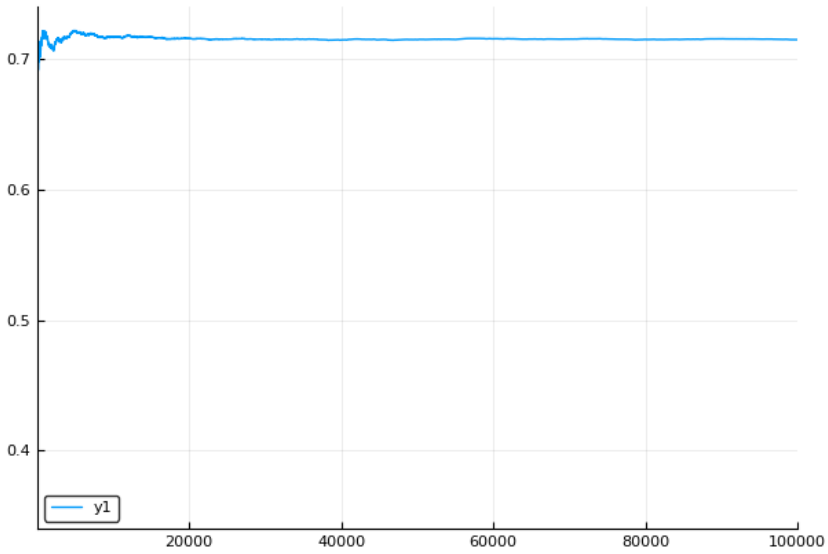
求得分段均值的样本标准差, 用分段均值的平均值的标准误差作为 EY 的估计 $\hat{\theta}$ 的标准误差：

```
std(z)/sqrt(m)
## 0.003241594936484582
```

这说明上面用 8000 个抽样对 EY 的估计大约精确到小数点后两位，即估计为 0.72。

上面模拟的总长度是 $N = 10000$，预热期是 $N_1 = 2000$，给出估计的标准误差时使用每 100 点的均值的统计量来估计，从精度和均值独立性假设检验的角度看还是不够满意。为此，增大样本量 N 到 10 万：

```
N = 100_000
srand(101)
x, y = simtrial(N)
avg = mavg(y)
plot(1:N, avg)
```



因为总模拟量增大，我们也可以相应地加大预热期，取 $N_1 = 20000$。给出 $\theta = EY$ 的点估计：

```
N1 = 20_000
N2 = N - N1
yout = y[(N1+1):N]
hat_ey = mean(yout)
## 0.7141137928323931
```

这也验证了前面取 $N = 10000$ 时给出的 0.72 估计值是基本正确的。

尝试计算每段 $r = 100$ 个点的分段平均值，再次评估这样的分段长度是否合适：

```
r = 100 # 试验使用的每段长度
m = Int(floor(N2/r)) # 段数
z = segavg(yout, r)
(autocor1(z), 2/sqrt(m))
## (0.045651, 0.070710)
```

这样的自相关与分段均值之间不相关是吻合的，所以取每段长度为 $r = 100$ 合适。用分段平均值的标准差来估计 $\hat{\theta}$ 的标准误差, 用 $N_2 = 80000$ 个样本点给出的点估计 $\hat{\theta}$ 的标准误差估计为

```
std(z)/sqrt(m)
## 0.0011348637276127777
```

这说明 EY 的估计值可报告为 $\hat{\theta} = 0.714$，精度在小数点后两位到三位。为了增加一位有效数字精度，需要抽样量增大 100 倍，到 1 千万级别。

※※※※※

例 19.6. 在 Gibbs 抽样中，每次变化的可以不是单个的分量，而是两个或多个分量。例如，设某个试验有 r 种不同结果，相应概率为 $p = (p_1, \dots, p_r)$(其中 $\sum_{i=1}^{r} p_i = 1$)，独立重复试验 n 次，各个结果出现的次数 $X = (X_1, \dots, X_r)$ 服从多项分布。设 $A = \{X_1 \geq 1, \dots, X_r \geq 1\}$，假设 $P(A)$ 概率很小，要在条件 A 下对 X 抽样，如果先生成 X 的无条件样本再舍弃不符合条件 A 的部分则效率太低，可以采用如下的 Gibbs 抽样方法。

首先，任取初值 $X^{(0)}$, 如 $X^{(0)} = (1, \dots, 1)$。假设已生成了 $X^{(t)}$，下一步首先从 $(1, \dots, r)$ 中随机地抽取两个下标 (i, j), 令 $s = X_i^{(t)} + X_j^{(t)}$，在给定 $X_k, k \neq i, j$ 为 $X^{(t)}$ 对应元素的条件下，(X_i, X_j) 的条件分布实际是 (X_i, X_j) 在 $X_i + X_j = s$ 以及 $X_i \geq 1, X_j \geq 1$ 条件下的分布。于是，在以上条件下，X_i 服从 $\mathrm{B}(s, p_i/(p_i + p_j))$ 分布限制在 $1 \leq X_i \leq s - 1$ 条件下的分布，即

$$\begin{aligned} q_k =& P(X_i = k | X_i + X_j = s, X_i \geq 1, X_j \geq 1) \\ =& \frac{C_s^k \left(\frac{p_i}{p_i+p_j}\right)^k \left(\frac{p_j}{p_i+p_j}\right)^{s-k}}{1 - \left(\frac{p_j}{p_i+p_j}\right)^s - \left(\frac{p_i}{p_i+p_j}\right)^s}, \ k = 1, 2, \dots, s-1. \end{aligned}$$

要生成这样的 X_i 的抽样只要用生成离散型随机数的逆变换法。设抽取的 X_i 值为 X_i^*, 取 $(X_i^{(t+1)}, X_j^{(t+1)}) = (X_i^*, s - X_i^*)$, 取 $X^{(t+1)}$ 的其它元素为 $X^{(t)}$ 的对应元素。如此重复就可以生成所需的 X 在条件 A 下的抽样链。

※※※※※

19.4 MCMC 计算软件 (*)

MCMC 是贝叶斯统计计算中最常用的计算工具。OpenBUGS 是一个成熟的 MCMC 计算开源软件（见 [Lunn et al., 2009], [Cowles, 2013], 另一个类似的

有关软件是 WinBUGS[Lunn et al., 2000]），能够进行十分复杂的贝叶斯模型的计算, 可以在 R 中直接调用 OpenBUGS 进行计算。

OpenBUGS 采用 Gibbs 抽样方法从贝叶斯后验分布中抽样，用户只需要指定先验分布和似然函数以及观测数据、已知参数，以及并行地生成多少个马氏链、链的一些初值、运行步数。软件自动计算 Gibbs 抽样所需的条件分布, 产生马氏链，并可以用图形和数值辅助判断收敛性，给出后验推断的概括统计。

在 R 中通过 BRugs 包调用 OpenBUGS 的功能。BRugs 用三类输入文件指定一个贝叶斯模型，第一类文件指定似然函数和参数先验密度，第二类文件指定已知参数、样本值，第三类文件指定马氏链初值（并行产生多个链时需要指定多组初值）。R 的 coda 软件包可以帮助对 MCMC 抽样结果进行分析和诊断。下面用一个简单例子介绍在 R 中用 BRugs 和 OpenBUGS 从贝叶斯后验中抽样的基本步骤。

例 19.7. 对例19.2,用 R 的 BRugs 包调用 OpenBUGS 来计算。设 $(X_1, \dots, X_5)$ 的观测值为 $(74, 85, 69, 17, 5)$。

OpenBUGS 用模型文件描述随机变量分布和对参数的依赖关系，以及参数之间的关系。首先建立如下模型文件, 保存在文件 `pfl-model.txt` 中:

```
model
{
    p[1] <- 1/3
    p[2] <- (1-b)/3
    p[3] <- (1-2*b)/3
    p[4] <- 2*b/3
    p[5] <- b/3
    b <- b2 / 2
    b2 ~ dbeta(1,1)
    x[1:5] ~ dmulti(p[1:5], N)
}
```

文件中用向左的箭头表示确定性的关系，用方括号加序号表示下标，用 $\sim$ 表示左边的变量服从右边的分布。这里，b 为参数 β, b2 $= 2b$ 服从 Beta(1,1) 分布即 U(0,1) 分布，于是 β 有先验分布 U(0,0.5)。`x[1:5]` 表示向量 $(x_1, \dots, x_5)$，

服从多项分布，参数为 $(p_1, \ldots, p_5)$, 试验次数为 N。N 在模型文件中没有指定，将在数据文件中给出。

建立如下的数据文件，保存在文件 `pfl-data.txt` 中:

```
list(x=c(74, 85, 69, 17, 5),
    N=250)
```

这样的数据文件是 R 软件的列表格式，列表中的标量和向量为通常的 R 程序写法，矩阵用如

```
list(M=structure(.Data=c(1,2,3,4,5,6), .Dim=c(3,2)))
```

表示，其中`.Dim` 给出矩阵的行、列数（3×2），`.Data` 给出按行排列的所有元素（第一行为 1,2, 第二行为 3,4，第三行为 5,6）。

OpenBUGS 需要用户指定各个链的要抽样的参数的初值，这里我们要抽样的参数是 `b`, 但 `b` 是由 `b2` 计算得到的，所以对 `b2` 设置初值。设有如下两个初值文件，不同链的初值应尽可能不同，文件 `pfl-inits1.txt` 内容为:

```
list(b2=0.2)
```

文件 `pfl-inits2.txt` 内容为:

```
list(b2=0.8)
```

在 R 中，首先调用 BRugs 软件包:

```
library(BRugs)
```

然后，读入并检查模型文件：

```
modelCheck('pfl-model.txt')
```

读入数据文件:

```
modelData('pfl-data.txt')
```

下面，对模型和数据进行编译，得到抽样方案, 下面的语句要求并行运行两个链:

```
modelCompile(numChains=2)
```

准备迭代地生成 MCMC 抽样了，首先指定初值:

```
modelInits(c('pfl-inits1.txt', 'pfl-inits2.txt'))
```

下面，先试验性地运行 1000 次，作为老化期:

```
modelUpdate(1000)
```

现在才指定抽样要输出那些随机变量的随机数:

```
samplesSet(c('b', 'p[1:5]'))
```

现在可以抽样了，指定运行 10000 次，两个链并行运行:

```
modelUpdate(10000)
```

得到抽样后，可以把抽样的结果保存在 R 的变量中，比如

```
b2chains <- samplesHistory('b', plot=FALSE)
```

得到一个列表，有唯一的元素 `b`, 为 2×10000 的矩阵，每行是一个链的记录。`sampleHistory` 可以抽样链的曲线图，如

```
samplesHistory('b')
```

OpenBUGS 提供了一系列的简单统计和收敛诊断功能。如下程序列出各抽样变量的简单统计:

```
print(samplesStats("*"))
```

结果为:

```
        mean       sd  MC_error val2.5pc  median val97.5pc start sample
b    0.08757 0.016900 1.168e-04  0.05737 0.08668   0.12330  1001  20000
p[2] 0.30410 0.005632 3.892e-05  0.29230 0.30440   0.31420  1001  20000
p[3] 0.27500 0.011260 7.784e-05  0.25120 0.27550   0.29510  1001  20000
p[4] 0.05838 0.011260 7.784e-05  0.03824 0.05778   0.08217  1001  20000
p[5] 0.02919 0.005632 3.892e-05  0.01912 0.02889   0.04109  1001  20000
```

统计使用所有链的数据。可以看出，β 的后验均值为 0.08757。表中的 `MC_error` 表示估计后验均值时由于随机模拟导致的误差的标准差的估计，这个标准差估计针对抽样自相关性进行了校正。在老化期之后的运行次数越多 `MC_error` 越小，一个常用的经验规则是保证 `MC_error` 小于后验标准差的 5，这里的结果提示还需要更多的运行次数。`val2.5pc` 和 `val97.5pc` 是抽样的后验分布的 2.5% 和 97.5% 分位数的估计值, 由此得到 β 的水平 95% 的可信区间 (credible interval) 为 $(0.0574, 0.1233)$。

如下程序画出 `b` 的后验密度估计:

```
samplesDensity('b', mfrow=c(1,1))
```

如下程序画抽样的 `b` 的自相关函数估计:

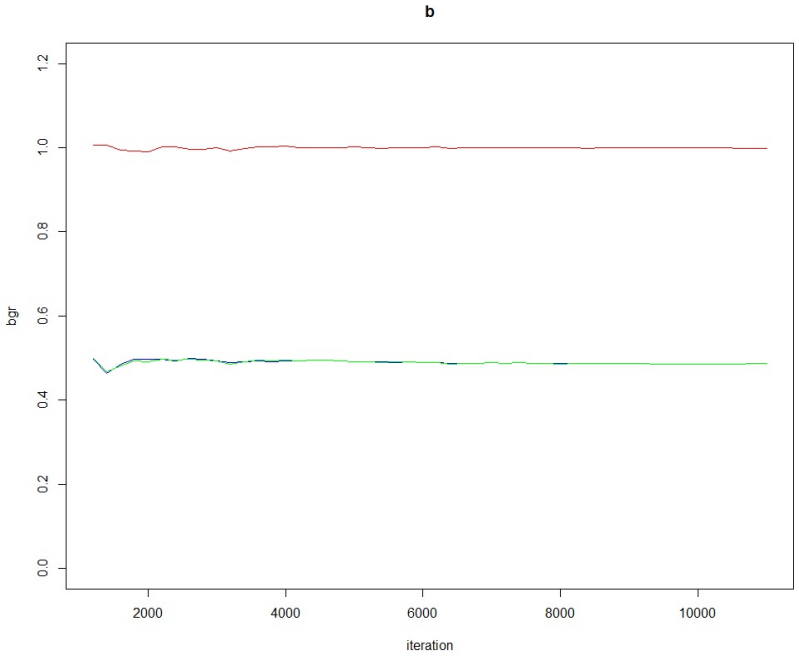
```
samplesAutoC('b', 1, mfrow=c(1,1))
```

当自相关函数很大而且衰减缓慢时生成的抽样链的效率较低。本例中的自相关函数基本表现为不相关列。

为了检查链是否收敛，OpenBUGS 提供了 BGR 统计量图:

```
samplesBgr('b', mfrow=c(1,1))
```

BGR 统计量的原理是考虑并行运行的每个链内部的变化情况，以及把所有的链合并在一起的变化情况，对这两种变化情况进行比较，当链收敛时，每个链内部的变化情况应该与合并在一起的变化情况很类似。类似于单因素方差分析中组间平方和与组内平方和的比较。当 BGR 图中的红色线接近于 1 并且三条线都保持稳定时就提示链收敛了。



※※※※※

习题

习题 1

设 $\{X_t, t = 0, 1, ...\}$ 为状态取值于 $S = \{1, 2, ..., m\}$ 的马氏链，有平稳分布 $\pi_j, j = 1, 2, ..., m$，若 X_0 服从平稳分布，证明所有 X_t 都服从平稳分布 $(t \geq 0)$。

习题 2

在例19.1中，设 $n = 100$, $a = 10000$, 编写 R 程序产生 MCMC 抽样链。

习题 3

在例19.2中，设 $(x_1, x_2, x_3, x_4, x_5) = (82, 72, 45, 34, 17)$, 选取适当的预热期和模拟，编写 R 程序计算 β 的后验均值估计。从不同初值出发多做几次考察估计的误差大小。

习题 4

在例19.3中，设 $A = 10, B = 10$, $d = 1$, $K = 11$，选取适当的预热期和步幅 σ，编写 R 程序产生 MCMC 抽样链。

习题 5

设向量 $x = (x_1, \ldots, x_n)$, 每个 x_i 取值为 $+1$ 或 -1，记这样的 x 的集合为 $\mathcal{X}$。设随机向量 X 有如下概率质量函数

$$\pi(x) = P(X = x) = \frac{1}{Z} \exp\left\{\mu \sum_{i=1}^{n-1} x_i x_{i+1}\right\}$$

其中 μ 已知，Z 为归一化常数。这样的模型称为一维 Ising 模型。取 $\mu = 1, n = 50$, 设计 MCMC 算法从 $\pi(x)$ 抽样。

习题 6

证明例19.4的(19.5)式。

习题 7

对例19.5, 取 $n = 20$, $\alpha = \beta = 0.5$, 生成 (X, Y) 的 Gibbs 抽样链，比较 Y 的样本的直方图和 Beta(α, β) 分布密度。

习题 8

设随机变量 X 和 Y 都取值于 $(0, B)$ 区间 (B 已知)。设 $Y = y$ 条件下 X 的条件分布密度为

$$f(x|y) \propto e^{-yx},\ x \in (0, B),$$

$X = x$ 条件下 Y 的条件分布密度为

$$f(y|x) \propto e^{-xy},\ y \in (0, B),$$

编写 R 程序用 Gibbs 抽样方法对 (X, Y) 抽样，估计 EX 和 $\rho(X, Y)$。

习题 9

设 $X = (X_1, X_2, \dots, X_n)$ 独立同 U(0,1) 分布，对 $0 < d < \frac{1}{n-1}$, 用条件 A 表示这 n 个点两两的距离都超过 d，可以证明 A 的概率为 $[1 - (n-1)d]^n$。设 $n = 9$, $d = 0.1$，设计 Gibbs 抽样方法生成满足条件 A 的 X 的抽样链。

Chapter 20

序贯重要抽样 (*)

MCMC 是目前广泛使用的随机模拟方法，其中 Gibbs 抽样方法（见 §19.3）在确定了各个分量的条件分布后可以轮流产生各个分量的抽样。序贯重要抽样方法是重要抽样法（见12）的一种推广，其做法与 Gibbs 抽样方法有些相似，也是每次从一个分量抽样。

回顾多维随机变量抽样的条件分布法（见 §7.1）。设随机向量 X 的密度 $\pi(x) = \pi(x_1, x_2, \dots, x_n)$ 可以逐次地分解为条件密度乘积

$$\pi(x_1, x_2, \dots, x_n) = \pi(x_1)\pi(x_2|x_1)\pi(x_3|x_1, x_2)\cdots\pi(x_n|x_1, \dots, x_{n-1})$$

则可以用条件分布法产生 X 的抽样。当数据是时间序列或者可以依次增加对 π 的信息 (比如 π 是 Bayes 后验分布) 时这种方法很自然。但是，条件分布 $\pi(x_t|x_1, \dots, x_{t-1})$ 可能是难以得到的或难以抽样的。为此，采用重要抽样思想：取一系列辅助分布 $\pi_t(x_t) = \pi_t(x_1, \dots, x_t), t = 1, \dots, n$，使其近似于 $(X_1, \dots, X_t)$ 的分布，$\pi_n = \pi$，各 π_t 可以分别有一个未知的归一化常数。对 $t = 1, 2, \dots, n$ 用重要抽样法逐次抽取 $X_1, X_2, \dots, X_n$，使得 $(X_1, \dots, X_t)$ 是关于 π_t 适当加权的样本。抽取 X_t 时一般是给定 $X_1, \dots, X_{t-1}$ 后从一个试抽样分布 $g_t(x_t|x_{t-1})$ 抽取。适当计算权重就可以得到关于 π 适当加权的抽样 $X_n = (X_1, X_2, \dots, X_n)$。

这种抽样方法叫做**序贯重要抽样**（sequential importance sampling，SIS），算法如下:

置$t \leftarrow 1$

从$g_1(\cdot)$抽取X_1, 置$W_1 \leftarrow \pi_1(X_1)/g_1(X_1)$

for(t in $2:n$){

从$g_t(x_t|X_1,\dots,X_{t-1})$抽取$X_t$, 记$X_t = (X_1,\dots,X_{t-1},X_t)$

计算步进权重$U_t \leftarrow \dfrac{\pi_t(X_t)}{\pi_{t-1}(X_{t-1})g_t(X_t|X_{t-1})}$

令$W_t \leftarrow W_{t-1}U_t$

}

输出(X_n, W_n)为关于$\pi(x)$适当加权的样本

SIS 一般同时独立进行 N 组，得到 $\{(X_n^{(j)}, W_n^{(j)})\}_{j=1}^N$，每一组称为一个 “**流**” 或一个 “**粒子**”。

按照 SIS 步骤，有

$$
\begin{aligned}
X_1 \sim g_1(x_1), \quad & w_1 = \frac{\pi_1(X_1)}{g_1(X_1)} \\
X_2 \sim g_2(x_2|x_1), \quad & U_2 = \frac{\pi_2(X_2)}{\pi_1(X_1)g_2(X_2|X_1)}, \\
& W_2 = W_1U_2 = \frac{\pi_2(X_2)}{g_1(X_1)g_2(X_2|X_1)} \\
X_3 \sim g_3(X_3|X_2), \quad & U_3 = \frac{\pi_3(X_3)}{\pi_2(X_2)g_3(X_3|X_2)}, \\
& W_3 = \frac{\pi_3(X_3)}{g_1(X_1)g_2(X_2|X_1)g_3(X_3|X_2)} \\
\cdots\cdots & \\
X_n \sim g_n(x_n|X_{n-1}), \quad & W_n = \frac{\pi_n(X_n)}{g_1(X_1)g_2(X_2|X_1)\cdots g_n(X_n|X_{n-1})}.
\end{aligned}
$$

记

$$g_t(x_t) = g_1(x_1)g_2(x_2|x_1)\cdots g_t(x_t|x_{t-1})$$

则 $X_t \sim g_t(\cdot)$ 而

$$W_t = \frac{\pi_t(X_t)}{g_t(X_t)},$$

因此 (X_t, W_t) 关于 π_t 适当加权 $(t = 1,\dots,n)$。注意 $\pi_n = \pi$ 故这样得到的 (X_n, W_n) 关于 $\pi(\cdot)$ 适当加权。

试抽样分布的一种常见取法为 $g_t(x_t|x_{t-1}) = \pi_t(x_t|x_{t-1})$。

20.1 非线性滤波平滑

SIS 方法可以用在很多统计模型的计算中, 作为示例，考虑如下的非线性滤波平滑问题。

设不可观测的 “状态”X_t 服从如下状态方程

$$X_t \sim q_t(\cdot|X_{t-1}, \theta),\ t = 1, 2, \dots, n$$

X_t 的信息可以反映在可观测的 Y_t 中，其关系满足如下观测方程

$$Y_t \sim f_t(\cdot|X_t, \phi),\ t = 1, 2, \dots, n$$

已知 $Y_n = (Y_1, \dots, Y_n) = y_n$ 和 θ, ϕ 后估计 $X = (X_1, \dots, X_n)$ 的问题称为滤波平滑问题, 只需要求后验分布 $\pi(x_n) = p_{X_n|Y_n}(x_n|y_n)$。如果能从 π 大量抽样 $X^{(j)}, j = 1, \dots, N$ 则可以用随机模拟方法对 $X_n = (X_1, \dots, X_n)$ 的后验分布进行推断。下面用 SIS 方法产生 π 的样本。

记

$$\pi_t(x_t) = p_{X_t|Y_t}(x_t|y_t),\ t = 1, 2, \dots, n$$

注意

$$\pi_t(x_t) \propto f_t(y_t|x_t) q_t(x_t|x_{t-1}) \pi_{t-1}(x_{t-1}) \tag{20.1}$$

取试抽样分布 $g_t(x_t|x_{t-1}) = q_t(x_t|x_{t-1})$, 对 $t = 1$, $\pi_1(x_1) = p_{X_1|Y_1}(x_1|y_1) \propto f_1(y_1|x_1)p(x_1)$, 其中 $p(x_1)$ 为 X_1 的分布密度，抽取 $X_1 \sim g_1(x_1)$, 如果可能应取 $g_1(x_1) = \pi_1(x_1)$。

产生关于 π 适当加权的 X_n 抽样的 SIS 步骤如下:

置$t \leftarrow 1$, 从$g_1(x_1)$抽取X_1, 置$W_1 \leftarrow \pi_1(X_1)/g_1(X_1)$
for(t in $2 : n$){
    从$q_t(x_t|X_{t-1})$抽取X_t, 记$X_t = (X_1, \dots, X_{t-1}, X_t)$
    令步进权重$U_t \leftarrow f_t(y_t|X_t)$
    令$W_t \leftarrow W_{t-1}U_t$
}
输出(X_n, W_n)作为关于$\pi(x)$适当加权的样本

这种方法相当于用状态方程前进一步获得下一分量的抽样，用同时刻的观测值 y_t 的似然作为步进权重。这样，t 步以后得到的 X_t 服从

$$g_t(x_t) = g_{t-1}(x_{t-1})q_t(x_t|x_{t-1})$$

其中 $g_1(x_1) \propto f_1(y_1|x_1)$。于是

$$\begin{aligned} \frac{\pi_t(x_t)}{g_t(x_t)} &= \frac{f_t(y_t|x_t)q_t(x_t|x_{t-1})\pi_{t-1}(x_{t-1})}{g_{t-1}(x_{t-1})q_t(x_t|x_{t-1})} \\ &= f_t(y_t|x_t)\frac{\pi_{t-1}(x_{t-1})}{g_{t-1}(x_{t-1})}, \\ W_t &= f_t(y_t|X_t)W_{t-1} = \frac{\pi_t(X_t)}{g_t(X_t)}, \end{aligned}$$

可见 (X_t, W_t) 关于 π_t 适当加权, (X_n, W_n) 关于 $\pi_n(x) = p_{X_n|Y_n}(x_n|y_n)$ 适当加权。

设第 t 步的抽样为 $X_t^{(i)}, i = 1, \ldots, N$, 对应权重为 $W_t^{(i)}, i = 1, \ldots, n$。以上的方法在抽取 $(X_1, \ldots, X_n)$ 时不考虑 $(y_1, \ldots, y_n)$ 的具体取值，这样的试抽样分布虽然很容易抽样，但是效果很差，权重 $\{W_t^i, i = 1, \ldots, N\}$ 随着 t 的增加会把变得差异很大，以至于 N 个流中只有极少数流能起作用。

由(20.1)可见

$$\pi_t(x_t|x_{t-1}) = p_{X_t|X_{t-1},Y_t}(x_t|x_{t-1}) \propto f_t(y_t|x_t)q_t(x_t|x_{t-1}),$$

如果能取试抽样分布 $g_t(x_t) \propto f_t(y_t|x_t)q_t(x_t|x_{t-1})$ 则每次抽取 X_t 都利用了同期的观测值 y_t 的信息，会大大改善抽样效率。更进一步，设 $\pi_{t+1}(x_{t+1})$ 关于 x_t 的边缘分布为 $\pi_{t,t+1}(x_t)$, 如果在第 t 步能从 $\pi_{t,t+1}(x_t)$ 的条件分布 $p(x_t|x_{t-1})$ 抽样，就可以利用 y_t, y_{t+1} 的信息，抽样效率可以进一步改善。

另外一种改进的办法是再抽样，增加权重大的流，舍弃权重小的流。

20.2 再抽样

如果试抽样分布选取不适当，最后的权重可能会差别很大，体现在权重 $\{W_t^{(i)}, i = 1, \ldots, N\}$ 的样本变异系数很大，称为权重偏斜严重。这时，权重小的流基本不起作用。出现这样的情况时，可以把一些权重太小的流舍弃而增加权重大的流，这样的技术称为**再抽样**。

20.2.1 简单随机再抽样

设进行 SIS 时在时刻 t 已经得到了 N 个部分的流 $\{X_t^{(i)}, i = 1, \dots, N\}$ 以及相应的权重 $\{W_t^{(i)}, i = 1, \dots, N\}$。可以在每一步都按照权重对流再抽样，也可以仅当权重偏斜严重时才进行再抽样。如果在第 t 步再抽样，只要以正比于 $W_t^{(i)}$ 的概率从 $\{X_t^{(j)}, j = 1, \dots, N\}$ 中抽取 $X_t^{(i)}$, 独立有放回地抽取 N 个，记作 $\{X_t^{*(i)}, i = 1, \dots, N\}$, 并把权重调整为相等的 $W_t^{*(i)} = \frac{1}{N}\sum_{j=1}^{N} W_t^{(j)}$, $i = 1, \dots, N$。这样的方法称为**简单随机再抽样**。这样再抽样后各个流不再是独立的。

20.2.2 剩余再抽样

为了达到以上的简单随机抽样的效果，还可以把大权重的的流直接复制多份，小权重的流仅以一定比例保留，其它舍弃。这种方法称为**剩余再抽样** (residual resampling)，其计算量更小而且模拟误差更小。算法描述如下。如果在第 t 步需要再抽样，则计算 $\bar{W}_t = \frac{1}{N}\sum_{j=1}^{N} W_t^{(j)}$, 然后按如下做法从 N 个流中重新抽取 N 个。首先，对 $i = 1, \dots, N$, 直接保留 $k_i = [W_t^{(i)}/\bar{W}_t]$ 份 $X_t^{(i)}$([·] 表示向下取整)；其次，令 $N_r = N - \sum_{i=1}^{N} k_i$ 为缺额个数，随机有放回地按照正比于 $\frac{W_t^{(i)}}{W_t} - k_i$(取整后的小数部分) 的概率从 $\{X_t^{(j)}, j = 1, \dots, N\}$ 中抽取 $X_t^{(i)}$, 共抽取 N_r 个。这样得到了 N 个新的流, 记作 $\{X_t^{*(i)}, i = 1, \dots, N\}$, 并调整其权重为相等的 $W_t^{*(i)} = \bar{W}_t$, $i = 1, \dots, N$。这种方法的第一步把权重大的流直接复制了若干份，第二步对按剩余的权重再抽取到满 N 个流为止。

20.2.3 舍选控制再抽样

简单随机再抽样和剩余再抽样都使得结果各个流不再独立。另外一种想法是采用 §12中介绍的舍选控制方法，对权重小的流从头重新抽样并适当调整权重。首先，设定若干个要执行再抽样的时间点 $0 < t_1 < \cdots < t_k \le n$, 以及相应的权重阈值 $c_1, \dots, c_k$。在 $t = t_j$ 时，进行舍选控制再抽样。若流 i 的权重 $W_t^{(i)} \ge c_j$, 则保留此流 $X_t^{(i)}$ 和权重 $W_t^{(i)}$ 不变；若流 i 的权重 $W_t^{(i)} < c_j$, 则以概率 $W_t^{(i)}/c_j$ 保留此流，如果决定保留，则修改其权重为 c_j，如果决定舍弃此流，则从 $t = 1$ 重新生成这个流，同样也需要经过 $t_1, \dots, t_j$ 处的舍选判断，如果被舍弃就还从 $t = 1$ 重新开始，直到被接受。

20.2.4 部分舍选控制再抽样

如果从头开始重新抽样的情况发生比较多则模拟的效率会比较低，为此可以采用如下的部分舍选控制再抽样的 SIS 方法。

首先，设定若干个要执行再抽样的时间点 $0 < t_1 < \cdots < t_k \leq n$, 以及相应的权重阈值 $c_1, \ldots, c_k$。在 $t = t_j$ 时，进行部分舍选控制再抽样。若流 i 的权重 $W_t^{(i)} \geq c_j$, 则保留此流 $X_t^{(i)}$ 和权重 $W_t^{(i)}$ 不变。若流 i 的权重 $W_t^{(i)} < c_j$, 则以概率 $W_t^{(i)}/c_j$ 保留此流, 如果决定保留，则修改其权重为 c_j。如果决定舍弃此流，则不是从头重新生成这个流，而是按照概率正比于 $W_{t_{j-1}}^{(s)}$ 从 $\{X_{t_{j-1}}^{(s)}, s = 1, \ldots, N\}$ 中随机抽取一个替换原来的 $X_{t_{j-1}}^{(i)}$, 用 t_{j-1} 时的权重 $\{W_{t_{j-1}}^{(s)}\}$ 的平均值 $\bar{W}_{t_{j-1}}$ 代替原来的权重 $W_{t_{j-1}}^{(i)}$，然后继续按照 SIS 标准步骤将此流经 $t = t_{j-1}+1, \ldots, t_j$ 延伸得到新的 $X_{t_j}^{(i)}$ 和权重 $W_{t_j}^{(i)}$，然后再进行 t_j 处的舍选控制，如果被舍弃就再从 t_{j-1} 处随机再抽样后继续，直到在 t_j 处被接受。

关于序贯重要抽样更详细的讨论参见 Liu [2001] 。

习题

习题 1

在调相通讯中，考虑如下的状态空间模型

$$\begin{aligned} X_t =& \phi_1 X_{t-1} + \eta_t,\ \eta_t \sim N(0, \sigma_\eta^2),\ t = 1, 2, \ldots, n \\ y_t =& A\cos(ft + X_t) + \varepsilon_t,\ \eta_t \sim N(0, \sigma_\varepsilon^2),\ t = 1, 2, \ldots, n \end{aligned} \tag{*}$$

其中 $\phi_1 = 0.6, \sigma_\eta^2 = 1/6, A = 320, f = 1.072 \times 10^7, \sigma_\varepsilon^2 = 1$, $(y_1, \ldots, y_n)$ 为观测值，$(X_1, \ldots, X_n)$ 为不可观测的随机变量。

(1) 设 $X_0 = 0$，$n = 128$，模拟生成 $(X_t, y_t), t = 1, 2, \ldots, n$。
(2) 设计 SIS 算法产生关于已知 $y_1, \ldots, y_n$ 条件下 $X_1, \ldots, X_n$ 的条件分布的适当加权样本，共生成 $N = 10000$ 组, 试抽样采用从 (*) 向前一步的方法。
(3) 考察以上得到的权重 $\{W_i\}$ 的分布情况。
(4) 在 SIS 抽样的每一步进行剩余再抽样；

(5) 根据后验均值方法利用上述改进的抽样估计 $(X_1, \ldots, X_n)$;

(6) 对每个 X_t，计算上述后验估计的标准误差;

(7) 独立地重复 $M = 400$ 次估计过程，从 M 次不同的后验估计计算新的估计标准误差, 与 6 得到的结果进行比较。

Part IV

近似计算

Chapter 21

函数逼近

在统计计算和其它科学计算中，经常需要计算各种函数的值，对函数进行逼近，用数值方法计算积分、微分。

21.1 多项式逼近

数学中的超越函数如 e^x、$\ln x$、$\sin x$ 在计算机中经常用泰勒级数展开来计算，这就是用多项式来逼近函数。多项式的高效算法见例2.11。数学分析中的 Weirstrass 定理表明，闭区间上的连续函数可以用多项式一致逼近。泰勒展开要求函数有多阶导数，我们需要找到对更一般函数做多项式逼近的方法。

考虑如下的函数空间

$$L^2[a,b] = \{g(\cdot) : g(x) \text{ 定义域为}[a,b], \text{ 且} \int_a^b g^2(x)w(x)dx < \infty\}$$

则 $L^2[a,b]$ 是线性空间，在 $L^2[a,b]$ 中定义内积

$$< f, g >= \int_a^b f(x)g(x)w(x)dx, \tag{21.1}$$

其中 $w(x)$ 是适当的权重函数，则 $L^2[a,b]$ 为 Hilbert 空间。对 $g(x) \in L^2[a,b]$，假设希望用 n 阶多项式 $f_n(x)$ 逼近 $g(x)$，使得

$$\|f_n - g\|^2 = \int_a^b |f_n(x) - g(x)|^2 w(x)dx \tag{21.2}$$

最小。如何求这样的多项式？

用 Gram-Schmidt 正交化方法可以在 $L^2[a,b]$ 中把多项式序列 $\{1,x,x^2,...\}$ 正交化为正交序列 $\{P_0,P_1,P_2,...\}$，序列中函数彼此正交，且 P_k 是 k 阶多项式，称 $\{P_0,P_1,P_2,...\}$ 为正交多项式。设 $H_n[a,b]$ 为函数 $\{1,x,x^2,...,x^n\}$ 的线性组合构成的线性空间，则 $\{P_0,P_1,...,P_n\}$ 构成 $H_n[a,b]$ 的正交基且 $P_n[a,b]$ 是 $L^2[a,b]$ 的子 Hilbert 空间，使得加权平方距离(21.2)最小的 $f_n(x)$ 是 $g(\cdot)$ 在子空间 $H_n[a,b]$ 的投影，记为 $\text{Proj}_{H_n[a,b]}(g)$, 投影可以表示为 $\{P_0,P_1,...,P_n\}$ 的线性组合

$$\underset{H_n[a,b]}{\text{Proj}}(g)=\sum_{j=0}^{n}\frac{<g,P_j>}{\|P_j\|^2}P_j. \tag{21.3}$$

这样，只要预先找到 $[a,b]$ 上的多项式的正交基，通过计算内积就可以很容易地找到使得(21.2)最小的 $f_n(x)$。对于 $L^2[a,b]$ 中的任意函数 $g(x)$ 有

$$\lim_{n\to\infty}\|\underset{H_n[a,b]}{\text{Proj}}(g)-g\|^2=0,$$

于是有

$$g=\lim_{n\to\infty}\underset{H_n[a,b]}{\text{Proj}}(g)=\sum_{j=0}^{\infty}\frac{<g,P_j>}{\|P_j\|^2}P_j.$$

因为 $L^2[a,b]$ 依赖于定义域 $[a,b]$ 和权重函数 $w(\cdot)$，所以正交多项式也依赖于 $[a,b]$ 和 $w(\cdot)$。针对定义域 $[-1,1]$, $[0,\infty)$ 和 $(-\infty,\infty)$ 和几种不同的权重函数可以得到不同的正交多项式序列，表21.1列出了这些正交多项式的定义域、权重函数和名称。

表 21.1: 正交多项式

函数空间	权函数	正交多项式	记号
$L^2[-1,1]$	1	Legendre	$P_n(x)$
$L^2[-1,1]$	$(1-x^2)^{-1/2}$(边界加重)	Chebyshev I 型	$T_n(x)$
$L^2[-1,1]$	$(1-x^2)^{1/2}$(中心加重)	Chebyshev II 型	$U_n(x)$
$L^2[0,\infty)$	$\exp(-x)$	Laguerre	$L_n(x)$

函数空间	权函数	正交多项式	记号
$L^2(-\infty,\infty)$	$\exp(-x^2)$	Hermite	$H_n(x)$
$L^2(-\infty,\infty)$	$\exp(-x^2/2)$	修正 Hermite	$He_n(x)$

Legendre 多项式是权重函数 $w(x)=1$ 的 $L^2[-1,1]$ 中的正交多项式，此权重函数在整个区间是等权重的，内积为

$$< f,g >= \int_{-1}^{1} f(x)g(x)\,dx.$$

有一般公式

$$P_n(x)=\frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n,$$

整个序列可以递推计算如下：

$$\begin{aligned} P_0(x) =& 1, \quad P_1(x)=x, \quad P_2(x)=(3x^2-1)/2,\\ P_3(x) =& (5x^3-3x)/2, \quad P_4(x)=(35x^4-30x^2+3)/8,\\ (n+1)P_{n+1}(x) =& (2n+1)xP_n(x)-nP_{n-1}(x), \end{aligned}$$

且 $\|P_n\|^2=2/(2n+1)$。

Chebyshev I 型多项式是权重函数为 $w(x)=(1-x^2)^{-1/2}$ 的 $L^2[-1,1]$ 中的正交多项式，此权重函数强调两端，内积为

$$< f,g >= \int_{-1}^{1} f(x)g(x)(1-x^2)^{-1/2}\,dx,$$

$T_n(x)$ 有一般表达式 $\cos(n\cos^{-1}(x))$, 整个序列可以递推计算如下：

$$\begin{aligned} T_0(x) =& 1, \quad T_1(x)=x, \quad T_2(x)=2x^2-1,\\ T_3(x) =& 4x^3-3x, \quad T_4(x)=8x^4-8x^2+1,\\ T_{n+1}(x) =& 2xT_n(x)-T_{n-1}(x) \end{aligned}$$

且 $\|T_0\|^2=\pi$, $\|T_n\|^2=\frac{\pi}{2}(n>0)$。

Chebyshev II 型多项式是权重函数为 $w(x)=(1-x^2)^{1/2}$ 的 $L^2[-1,1]$ 中的正交多项式，此权重函数强调区间中间，内积为

$$< f,g >= \int_{-1}^{1} f(x)g(x)(1-x^2)^{1/2}\,dx,$$

整个序列可以递推计算如下：

$$\begin{aligned} U_0(x) =& 1, \quad U_1(x) = 2x, \quad U_2(x) = 4x^2 - 1, \\ U_3(x) =& 8x^3 - 4x, \quad U_4(x) = 16x^4 - 12x^2 + 1, \\ U_{n+1}(x) =& 2xU_n(x) - U_{n-1}(x), \end{aligned}$$

且 $\|U_n\|^2 = \pi/2$。

Laguerre 多项式是权重函数为 $w(x) = e^{-x}$ 的 $L^2[0, \infty)$ 中的正交多项式，此权重函数强调区间左边，内积为

$$< f, g >= \int_0^\infty f(x)g(x)e^{-x}\, dx,$$

整个序列可以递推计算如下：

$$\begin{aligned} L_0(x) =& 1, \quad L_1(x) = 1 - x, \quad L_2(x) = (2 - 4x + x^2)/2, \\ L_3(x) =& (6 - 18x + 9x^2 - x^3)/6, \\ L_4(x) =& (24 - 96x + 72x^2 - 16x^3 + x^4)/24, \\ (n+1)L_{n+1}(x) =& (2n + 1 - x)L_n(x) - nL_{n-1}(x), \end{aligned}$$

且 $\|L_n\|^2 = 1$。

Hermite 多项式是权重函数为 $w(x) = e^{-x^2}$ 的 $L^2(-\infty, \infty)$ 中的正交多项式，此权重函数强调区间中央，内积为

$$< f, g >= \int_{-\infty}^\infty f(x)g(x)e^{-x^2}\, dx,$$

整个序列可以递推计算如下：

$$\begin{aligned} H_0(x) =& 1, \quad H_1(x) = 2x, \quad H_2(x) = 4x^2 - 2, \\ H_3(x) =& 8x^3 - 12x, \quad H_4(x) = 16x^4 - 48x^2 + 12, \\ H_{n+1}(x) =& 2xH_n(x) - 2nH_{n-1}(x), \end{aligned}$$

且 $\|H_n\|^2 = 2^n n! \pi^{1/2}$。

修正 Hermite 多项式与 Hermite 多项式的区别仅仅是权重函数由 e^{-x^2} 改成了 $w(x) = e^{-x^2/2}$，内积为

$$< f, g >= \int_{-\infty}^\infty f(x)g(x)e^{-x^2/2}\, dx,$$

整个序列可以递推计算如下：

$$\begin{aligned} He_0(x) =& 1, \quad He_1(x) = x, \quad He_2(x) = x^2 - 1, \\ He_3(x) =& x^3 - 3x, \quad He_4(x) = x^4 - 6x^2 + 3, \\ He_{n+1}(x) =& xHe_n(x) - nHe_{n-1}(x), \end{aligned}$$

且 $\|He_n\|^2 = n!\sqrt{2\pi}$。

统计计算中常常会遇到没有解析表达式的函数的计算问题，比如，标准正态分布函数 $\Phi(x) = \int_{-\infty}^{x} \phi(u)du = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2} du$，我们可以用数值积分方法把 $\Phi(x)$ 计算到很高精度，但是这样计算会极为耗时。为此，我们可以用 Hermite 多项式或修正 Hermite 多项式逼近 $\Phi(x)$ 到一定的精度，计算多项式系数时可以用数值积分计算，这些系数计算可能耗时很多但是只要找到了逼近多项式就可以一劳永逸了。当然，经过多年研究，常见分布函数都已经有了很好的近似公式，比如

$$\Phi(x) = \begin{cases} \frac{1}{2} + \phi(x) \sum_{n=0}^{\infty} \frac{1}{(2n+1)!!} x^{2n+1}, & 0 \le x \le 3, \\ 1 - \phi(x) \sum_{n=0}^{\infty} (-1)^n \frac{(2n-1)!!}{x^{2n+1}}, & x > 3 \end{cases} \tag{21.4}$$

但是，在各种各样的统计模型中我们还是会遇到很多需要快速计算的函数，这些函数很可能没有高阶导数，每计算一次花费很长时间，我们可以用多项式逼近或插值方法给出近似公式，然后每次用近似公式计算函数值。

有些统计方法需要估计一个函数，这时可以用正交多项式级数表示这个函数，只要估计级数展开所需的系数。

21.2 有理函数与连分式逼近

用正交多项式近似函数的好处是公式和计算简单，容易计算展开系数，容易微分、积分。但是多项式仅能逼近有限区间上的函数，容易出现很强的波动，尤其是在区间边界处, 而且多项式没有渐近线，没有无穷函数值的点，很多函数，尤其是取值区间包含正负无穷的函数，用正交多项式很难得到高精度。

有理函数能克服多项式的这些缺点。分子和分母都是多项式的分式叫做有理函数。逼近函数时，在相同参数个数条件下有理函数经常比多项式逼近好。有理函数可以从幂级数展开表示中解出。有理函数在计算时可以化为连分式来计算，连分式可以高效地计算。

连分式是像

$$1+\cfrac{1}{2+\cfrac{1}{3+\cfrac{1}{4}}}=1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}$$

这样的分式，等号右侧是对左侧的分式的简写。一般地有

$$R_n = b_0+\cfrac{a_1}{b_1+\cfrac{a_2}{b_2+\ddots+\cfrac{a_n}{b_n}}}=b_0+\frac{a_1}{b_1}+\frac{a_2}{b_2}+\cdots+\frac{a_n}{b_n}. \tag{21.5}$$

计算连分式(21.5), 可以使用如下反向算法或正向算法，正向算法适用于无穷连分式的计算。

连分数计算反向算法

$z \leftarrow b_n$
for $(k$ **in** $n:1)$ {
    $z \leftarrow b_{k-1}+\frac{a_k}{z}$
}
输出 $R_n \leftarrow z$

连分数计算正向算法

$A_0 \leftarrow b_0$, $B_0 \leftarrow 1$
$A_1 \leftarrow b_1b_0+a_1$, $B_1 \leftarrow b_1$
for$(k=2,\ldots,n)$ {
    $A_k = b_kA_{k-1}+a_kA_{k-2}$
    $B_k = b_kB_{k-1}+a_kB_{k-2}$
}
输出 $R_n = A_n/B_n$，同时 $R_k = A_k/B_k, k=1,2,\ldots,n$

计算幂级数时，如果预先知道需要计算的项数，可以把幂级数化为无穷连分数，用连分数的正向算法计算:

$$\sum_{i=0}^{\infty}a_ix^i = \frac{b_0}{1}-\cfrac{b_1x}{1+b_1x}-\cfrac{b_2x}{1+b_2x}-\cdots \tag{21.6}$$

其中 $b_0 = a_0$, $b_i = a_i/a_{i-1} (i > 0)$.

有理分式可以化为连分式计算。转化的一般规则是:

- 如果分子分母都有常数项，如 $\dfrac{1+2x+x^3}{1+2x}$ 从分式中减去此常数项，使得分子以 x 为因子 (如 $1+\dfrac{x^3}{1+2x}$)。
- 如果分子中有 x 因子而分母中有常数项，则把分式变成分子只有 x。(如 $\dfrac{\dfrac{x}{1+2x}}{x^2}$)
- 如果分母中有 x 因子而分子中有常数项 (如 $\dfrac{1+2x}{x^2}$)，写成倒数形式 (如 $\dfrac{\dfrac{1}{x^2}}{1+2x}$) 再处理。
- 转换的想法是分子上只留常数项或一次项。

例如

$$\begin{aligned}
\frac{1+2x+x^3}{1+2x} =& \frac{1}{1} + \left(\frac{1+2x+x^3}{1+2x} - \frac{1}{1}\right) \text{ (减去都有的常数项)} \\
=& 1 + \frac{x^3}{1+2x} = 1 + \frac{x}{\dfrac{1+2x}{x^2}} \text{ (分子仅留 } x\text{)} \\
=& 1 + \frac{x}{\dfrac{1}{\dfrac{x^2}{1+2x}}} \text{ (分母有 } x \text{ 但分子有常数加项时化为倒数)} \\
=& 1 + \frac{x}{\dfrac{1}{\dfrac{x}{2+\dfrac{1}{x}}}} \text{ (分子仅留 } x\text{)} \\
=& 1 + \frac{x}{0} + \frac{1}{0} + \frac{x}{2} + \frac{1}{x}.
\end{aligned}$$

对于一般有理分式

$$g(x) = \frac{\sum_{i=0}^{\infty} a_{1i} x^i}{\sum_{i=0}^{\infty} a_{0i} x^i} \tag{21.7}$$

可化为

$$g(x) = \frac{a_{10}}{a_{00}} + \frac{\dfrac{a_{20}x}{a_{10}}}{} + \frac{\dfrac{a_{30}x}{a_{20}}}{} + \cdots \tag{21.8}$$

其中

$$a_{mi} = a_{m-1,0}a_{m-2,i+1} - a_{m-2,0}a_{m-1,i+1},\ i = 0, 1, 2, ...;\ m = 2, 3, 4, ...$$

有理分式化为连分式还可以用如下公式：

$$g(x) = \frac{\sum_{i=0}^{\infty} b_{1i}x^i}{\sum_{i=0}^{\infty} b_{0i}x^i} = \frac{1}{d_0} + \frac{x}{d_1} + \frac{x}{d_2} + \cdots \tag{21.9}$$

其中 $d_m = \frac{b_{m,0}}{b_{m+1,0}}$,

$$b_{m+2,i} = b_{m,i+1} - d_m b_{m+1,i+1},\ i = 0, 1, 2, ...; m = 0, 1, 2, ...$$

例如，计算标准正态分布函数的(21.4)可以用 (21.9)化为如下连分式：

$$\Phi(x) = \begin{cases} \frac{1}{2} + \frac{\phi(x)x}{1} - \frac{x^2}{3} + \frac{2x^2}{5} - \cdots + \frac{(-1)^k k x^2}{2k+1} + \cdots, & 0 \le x \le 3, \\ 1 - \frac{\phi(x)}{x} + \frac{1}{x} + \frac{2}{x} + \cdots + \frac{k}{x} + \cdots, & x > 3 \end{cases}. \tag{21.10}$$

21.3 逼近技巧

大多数计算机语言都提供了 $\sqrt{x}$、$\ln x$、e^x、$\sin x$ 等函数，R、SAS 等系统更提供了大多数概率密度、分布函数、分位数函数和随机数函数。我们应该尽量利用这些经过很多人验证的函数实现, 但是，也不能盲目相信已有的计算代码。

函数的数学定义、计算公式和计算机算法这三者是有很大差别的。计算机只能计算加减乘除，即使有些计算公式已经是泰勒展开式这样的形式，在实际编程计算时也要考虑浮点表示误差、误差积累等问题。另外，一个常用函数的算法需要同时满足精确性和高效率，算法的一点小改进就可能在反复调用中放大为很显著的效率改进。

以 $\sqrt{x}$ 计算为例，如果 y^2 的值和 x 差距在机器单位 U 以下，则可以用 y 作为 $\sqrt{x}$ 的值，尽管这样的 y 可能有多个。构造一个达到如此精度要求的高效率算法可能需要很多努力。我们知道，泰勒展开、多项式逼近等一般只适用于小范围而不适用于无穷区间，这时，“范围缩减” 技术就可以帮我们把能够精确计算的范围扩大到全定义域。对 $\sqrt{x}$，我们可以用 $x_0 = 1$ 处的泰勒展开精确计算 $x \in [\frac{1}{2}, 1]$ 区间的 $\sqrt{x}$:

$$\sqrt{x} = 1 + \frac{1}{2}(x-1) - \frac{1}{8}(x-1)^2 + \frac{1}{16}(x-1)^3 - \frac{5}{128}(x-1)^4 + \cdots$$

实际设计算法时要考虑这个公式中正负项互相抵消和误差积累问题。用 $\sqrt{2} = \sqrt{4 \times \frac{1}{2}} = 2\sqrt{\frac{1}{2}}$ 计算出 $\sqrt{2}$，对于一般的 x，都可以写成

$$x = a2^k,\ a \in [\frac{1}{2}, 1]$$

从而

$$\sqrt{x} = \sqrt{a} \cdot 2^{k/2}$$

如果 k 是奇数，设 $k = 2m + 1$，则 $2^{k/2} = 2^m\sqrt{2}$。

习题

习题 1

把 Legendre 多项式推广到 $L^2[a, b]$ 中, 给出递推公式。

习题 2

设计计算 $L^2[a, b]$ 上的 Legendre 多项式 P_n 系数的算法并编写 R 程序。

习题 3

设计计算 Laguerre 多项式 L_n 系数的算法并编写 R 程序。

习题 4

用 Legendre 正交多项式找到标准正态分布函数 $\Phi(x), x \in [0, 3]$ 的绝对误差在 10^{-6} 以下的近似公式并用 R 程序验证。可以使用 R 中的 `pnorm` 函数作为已知的精确值。

习题 5

用 Laguerre 正交多项式方法找到标准正态分布函数 $\Phi(x), x \in [3, \infty)$ 的绝对误差在 10^{-6} 以下的近似公式并用 R 程序验证。

习题 6

证明连分式计算的正向算法。

习题 7

编写用连分式公式(21.10)计算标准正态分布函数 $\Phi(x)$ 的 R 程序，要求绝对误差控制在 10^{-10} 以下。

Chapter 22

插值

22.1 多项式插值

例 22.1. 下表是 $F(1,n)$ 分布的 0.95 分位数的部分值：

n	$\cdots$	20	$\cdots$	29	30	40	60	120	∞
$h(n)$	$\cdots$	4.35	$\cdots$	4.18	4.17	4.08	4.00	3.92	3.84

设关于函数 $h(n)$ 我们只有上述知识，但是还知道 $h(n)$ 具有一定光滑性。这里 $h(30)=4.17, h(40)=4.08$, 为了计算 $h(32)$，只要把 $(30,h(30))$ 和 $(40,h(40))$ 这两个点用线段连接，在 $(30,40)$ 之间用连接的线段作为 $h(x)$ 的近似值，就可以计算

$$h(32)=h(30)+\frac{h(40)-h(30)}{40-30}(32-30)\approx 4.15$$

称这样的近似计算为**线性插值**。

还可以进一步地近似。比如我们还知道 $h(20)=4.35$，则存在唯一的二次多项式 $f(x)$ 使得 $f(x)$ 穿过 $(20,h(20))$, $(30,h(30))$, $(40,h(40))$ 这三个点，用 $f(32)$ 来近似 $h(32)$，这叫做**抛物线插值**。

一般地，设已知函数 $h(x)$ 在点 $x_1,x_2,\ldots,x_n$ 上的值 $z_1,z_2,\ldots,z_n$ $(z_i=h(x_i))$，在假定 $h(x)$ 具有一定光滑度的条件下可以用函数 $f(x)$ 近似计算 $h(x)$ 在自变量 x 处的值。称 $h(x)$ 为**被插值函数**，$x_1,x_2,\ldots,x_n$ 叫做节点, $f(x)$ 叫做**插值**

函数，一般使用多项式或分段多项式作插值函数。如果我们只能获得 $h(x)$ 在有限个点上的值，可以用插值方法近似计算 $h(x)$ 在其它自变量处的值。如果函数 $h(x)$ 每计算一个函数值到所需精度都要花费很长时间，可以预先在一些自变量处计算函数值并保存好，然后需要用到 $h(x)$ 的值时用插值法近似计算整个定义域上的函数值。如果 $h(x)$ 的计算精度要求不高，也可以预先计算并保存部分自变量处的函数值，然后用插值来近似函数值。插值得到的表达式可以用于积分、微分计算。

例22.1中用连接两点的线段作为插值公式的方法叫做线性插值。一般地，设已知 $(x_i, z_i), i = 1, \dots, n,\ z_i = h(x_i)$。对 $x \in [x_{i-1}, x_i]$, 令

$$h(x) \approx f(x) = z_{i-1} + \frac{z_i - z_{i-1}}{x_i - x_{i-1}}(x - x_{i-1}) \tag{22.1}$$

(这就是直线方程的斜截式)。如果记 $\alpha = \frac{x-x_{i-1}}{x_i-x_{i-1}}$，即点 x 在插值区间 $[x_{i-1}, x_i]$ 中前后位置比例，则

$$f(x) = (1-\alpha)z_{i-1} + \alpha z_i. \tag{22.2}$$

当 $h(x)$ 本身在 $[x_{i-1}, x_i]$ 就是一次多项式时插值是精确的。

R 中用函数 `approx` 对给定的 n 个点在相邻点之间做线性插值。

当已知三个点的函数值时可以找到穿过这三个点的抛物线,用对应的二次多项式作为 $h(x)$ 的近似值。设 x_{-1}, x_0, x_1 是三个点且距离为 $x_0 - x_{-1} = x_1 - x_0 = d$，函数值分别为 z_{-1}, z_0, z_1。对 $x \in [x_{-1}, x_1]$，令 $\alpha = \frac{x-x_0}{d}$，则 $\alpha \in [-1, 1]$, 令

$$f(\alpha) = \frac{z_{-1} - 2z_0 + z_1}{2}\alpha^2 + \frac{z_1 - z_{-1}}{2}\alpha + z_0,$$

则 $f(\alpha)$ 是 x 的二次函数，且经过 $(x_{-1}, z_{-1}), (x_0, z_0), (x_1, z_1)$ 这三个点, 可以用 $f(\alpha)$ 近似 $h(x)$ 的值。

如果有 n 个等距格点及对应函数值 $(x_i, z_i), i = 1, 2, \dots, n$，$z_i = h(x_i)$, 为了用抛物线插值方法近似计算 $h(x)$, $x \in [x_1, x_n]$, 取 m_i 为区间 $[x_i, x_{i+1}]$ 的中点，对 $x \in [m_i, m_{i+1}]$ 用 (x_i, x_{i+1}, x_{i+2}) 三点插值 $(m = 2, 3, \dots, m_{n-2})$, 在 $[x_1, m_1]$ 内用 (x_1, x_2, x_3) 三个点插值，在 $[m_{n-1}, x_n]$ 内用 x_{n-2}, x_{n-1}, x_n 三个点插值。

下面的定理保证了插值多项式的存在唯一性。

定理 22.1. 设有 $\{(x_i, z_i), i = 1, 2, \dots, n\}$, $\{x_i, i = 1, 2, \dots, n\}$ 互不相同，则存在唯一的不超过 $n-1$ 阶的多项式 $P_{n-1}(x)$ 使得

$$P_{n-1}(x_i) = z_i,\ i = 1, 2, \dots, n.$$

证明: 设 $P_{n-1}(x) = a_0 + a_1 x + \dots + a_{n-1}x^{n-1}$ 为多项式, 其中 $a_0, a_1, \dots, a_{n-1}$ 为待定常数。为使 $P_{n-1}(x_i) = z_i$, $i = 1, 2, \dots, n$，应有

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} \tag{22.3}$$

注意到此方程组的系数矩阵行列式为 Vandermonde 行列式，行列式值为

$$\prod_{1 \le i < j \le n} (x_j - x_i) \neq 0$$

所以方程组存在唯一解, 即存在唯一的次数不超过 $n-1$ 的多项式 $P_{n-1}(x)$ 使 $P_{n-1}(x)$ 通过 $(x_i, z_i), i = 1, 2, \dots, n$ 这 n 个点。

由定理22.1的证明可见求插值多项式的方法之一是解线性方程组(22.3)，但是这样难以得到用初始值 $(x_i, z_i), i = 1, 2, \dots, n$ 表示的函数表达式。

22.2 Lagrange 插值法

获得插值多项式表达式的一种方法是 Lagrange 插值法。令

$$d_i(x) = \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}$$

则 $d_i(x)$ 是 $n-1$ 阶多项式, 在各 $\{x_i\}$ 上满足:

$$\begin{cases} d_i(x_i) = 1 \\ d_i(x_k) = 0, k \neq i \end{cases}$$

令

$$f(x) = \sum_{i=1}^{n} z_i d_i(x) \tag{22.4}$$

则 $f(x)$ 是通过指定的 n 个点的不超过 $n-1$ 阶的多项式，根据定理22.1可知(22.4)给出的多项式是唯一的插值多项式。

为了用 Lagrange 方法插值，首先需要确定 n 个多项式 $d_i(x), i = 1, 2, \ldots, n$ 的系数。为了便于计算，记

$$g(x) = (x - x_1)(x - x_2)\cdots(x - x_n) = \prod_{i=1}^{n}(x - x_i), \tag{22.5}$$

则有

$$g'(x) = \sum_{k=1}^{n}\prod_{j\neq k}(x - x_j),$$

$$g'(x_i) = \sum_{k=1}^{n}\prod_{j\neq k}(x_i - x_j) = \prod_{j\neq i}(x_i - x_j)$$

于是可知

$$d_i(x) = \frac{g(x)/(x - x_i)}{g'(x_i)}. \tag{22.6}$$

这样，只要算出 $g(x)$ 的系数和 $g'(x)$ 的系数, 然后用多项式除法写出 $g(x)/(x - x_i)$ 的系数，就可以用(22.6)得到 $d_i(x)$ 的系数。

计算插值多项式 $f(x)$ 的系数的算法
写出 $g(x)$ 的展开式 $g(x) = \sum_{i=0}^{n} a_i x^i$.
写出 $g'(x)$ 的展开式 $g'(x) = \sum_{i=0}^{n-1}(i+1)a_{i+1}x^i$.
计算各 $g'(x_i)$.
写出各 $g(x)/(x - x_i)$ 的展开式.
用下式给出插值多项式表达式：
$f(x) = \sum_{i=1}^{n} \frac{z_i}{g'(x_i)}[g(x)/(x - x_i)]$

上述算法中用到了多项式加减法、乘法和除法。

为计算多项式乘法 $\sum_{i=0}^{n} a_i x^i \sum_{j=0}^{m} b_j x^j$, 记 $a_i = 0 (i > n)$, $b_j = 0 (j > m)$, 有

$$\begin{aligned}
&\sum_{i=0}^{n} a_i x^i \sum_{j=0}^{m} b_j x^j = \sum_{i=0}^{n} \sum_{j=0}^{m} a_i b_j x^{i+j} \text{ (令 } k = i + j) \\
=&\sum_{i=0}^{n} \sum_{k=i}^{i+m} a_i b_{k-i} x^k = \sum_{k=0}^{m+n} \left(\sum_{i=0}^{k} a_i b_{k-i} \right) x^k \\
=&\sum_{k=0}^{m+n} \left(\sum_{i=\max(0,k-m)}^{\min(k,n)} a_i b_{k-i} \right) x^k \text{ (注意 } 0 \le i \le n, 0 \le k-i \le m) \\
=&\sum_{k=0}^{m+n} c_k x^k,
\end{aligned}$$

乘积多项式的系数为

$$c_k = \sum_{i=\max(0,k-m)}^{\min(k,n)} a_i b_{k-i}, \ k = 0, 1, \ldots, m+n. \tag{22.7}$$

$\{c_k\}$ 是 $\{a_k\}$ 和 $\{b_k\}$ 的卷积。如果 $\{a_k\}$, $\{b_k\}$ 是两个数列，称

$$c_k = \sum_{i=-\infty}^{\infty} a_i b_{k-i}, \ k \in \mathbb{Z}$$

($\mathbb{Z}$ 为所有整数) 为 $\{a_k\}$, $\{b_k\}$ 的卷积。

注: R 函数 `filter` 和 `convolve` 可以计算两个有限序列的卷积。

关于多项式插值的误差有如下定理。

定理 22.2. *设 $h(x) \in C^{n-1}[a,b]$，$h^{(n)}(x)$ 在 (a,b) 内存在，设 $\{x_i, i = 1, 2, \ldots, n\}$ 互不相同，$f(x)$ 和 $g(x)$ 分别由(22.4)、(22.5)定义，$\{x_i, i = 1, 2, \ldots, n\}$ 的最小值到最大值的区间为 I，则对 $\forall x \in I$, 存在 $\xi \in I$ 使*

$$R(x) = h(x) - f(x) = \frac{h^{(n)}(\xi)}{n!} g(x). \tag{22.8}$$

证明不妨设 $x_1, x_2, \ldots, x_n$ 从小到大排列。对 $x \in \{x_1, x_2, \ldots, x_n\}$, 显然 $R(x) = 0$。对给定的 $x \in [a,b] \backslash \{x_1, x_2, \ldots, x_n\}$, 定义辅助函数

$$H(t) = R(t) - \frac{g(t)}{g(x)} R(x)$$

易见 $H(x_i) = 0, i = 1, 2, \ldots, n$，且 $H(x) = 0$，所以 $H(t)$ 有 $n+1$ 个相异零点。由 Rolle 定理，$H(t)$ 的每两个相邻零点中间必有一个点导数等于零，所以

$H^{(1)}(t)$ 至少有 n 个相异零点。类似讨论可知 $H^{(j)}(t)$ 至少有 $n+1-j$ 个相异零点，$j=1,2,\dots,n$。因此 $H^{(n)}(t)$ 至少有一个零点。设 ξ 是 $H^{(n)}(t)$ 的一个零点，则

$$\begin{aligned}0 =&H^{(n)}(\xi)=R^{(n)}(\xi)-\frac{g^{(n)}(\xi)}{g(x)}R(x)\\ =&h^{(n)}(\xi)-f^{(n)}(\xi)-\frac{n!}{g(x)}R(x)\\ =&h^{(n)}(\xi)-\frac{n!}{g(x)}R(x)\end{aligned}$$

于是

$$R(x)=\frac{h^{(n)}(\xi)}{n!}g(x)$$

定理得证。

※※※※※

$g(x)$ 在靠近 $\{x_i\}$ 最小值和最大值的位置绝对值较大，说明多项式插值在边界处误差较大, 见图22.1的左下图。所以，多项式插值并不是多项式的阶数越高越好，阶数太高的多项式的曲线形状会变得很复杂，插值误差反而会很大。

为了使(22.8)中的 $g(x)$ 不要太大，还可以考虑适当选取 x_i 的位置。对 $(-1,1)$ 区间，取 x_i 为 Chebyshev I 型多项式 $T_n(x)$ 的零点 $x_i=\cos\left(\frac{2i-1}{2n}\pi\right)$ $(i=1,2,\dots,n)$ 可以使 $g(x)$ 最小。

例 22.2 (用 Lagrange 法进行二次插值)**.** 设已知 $(x_i,z_i),i=1,2,3$ 三个点，求通过这三个点的抛物线方程。按照 Lagrange 方法，令

$$\begin{aligned}u_1(x)=&(x-x_2)(x-x_3)=x^2-(x_2+x_3)x+x_2x_3\\ u_2(x)=&(x-x_1)(x-x_3)=x^2-(x_1+x_3)x+x_1x_3\\ u_3(x)=&(x-x_1)(x-x_2)=x^2-(x_1+x_2)x+x_1x_2\end{aligned}$$

则 $d_i(x)=u_i(x)/u_i(x_i)$。记 $b_i=z_i/u_i(x_i)$ 则 $f(x)=\sum_{i=1}^3 b_iu_i(x)$, 于是抛物线插值公式为

$$\begin{aligned}f(x)=&(b_1+b_2+b_3)x^2\\ &-\left(b_1(x_2+x_3)+b_2(x_1+x_3)+b_3(x_1+x_2)\right)x\\ &+(b_1x_2x_3+b_2x_1x_3+b_3x_1x_2)\end{aligned}$$

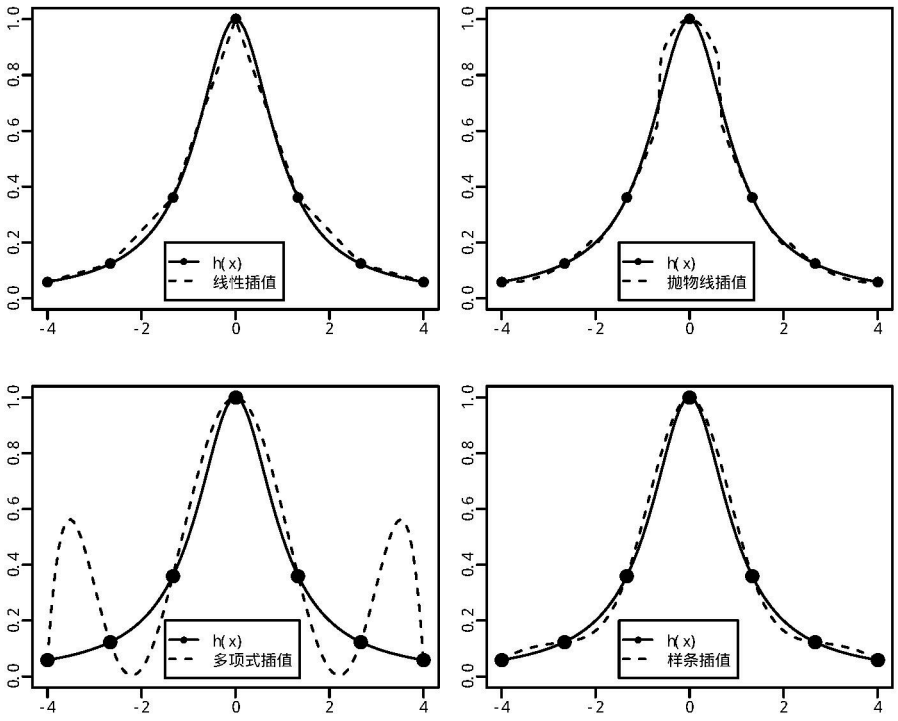


图 22.1: 函数 $1/(1+x^2)$ 的线性插值、抛物线插值、$n-1$ 阶多项式插值和样条插值

例如，对 $h(x)=\sqrt{x}$ 在 $x=1/16,1/4,1$ 用抛物线插值，有

$$b_1=\frac{1/4}{u_1(\frac{1}{16})}=\frac{64}{45},\ b_2=\frac{1/2}{u_2(\frac{1}{4})}=-\frac{32}{9}=-\frac{160}{45},\ b_3=\frac{1}{u_3(1)}=\frac{64}{45}$$

于是插值函数

$$\begin{aligned}h(x)=&\left(\frac{64}{45}-\frac{160}{45}+\frac{64}{45}\right)x^2\\&-\left(\frac{64}{45}(\frac{1}{4}+1)-\frac{160}{45}(\frac{1}{16}+1)+\frac{64}{45}(\frac{1}{16}+\frac{1}{4})\right)x\\&+\left(\frac{64}{45}\frac{1}{16}\frac{1}{4}-\frac{160}{45}\frac{1}{16}\cdot 1+\frac{64}{45}\frac{1}{4}\cdot 1\right)\\=&\frac{1}{45}(7+70x-32x^2)\end{aligned}$$

在 $[0,1]$ 插值的平均绝对误差约为 0.033。

在 $(-1,1)$ 区间按照 Chebyshev I 型零点取 x_i 为 $-\sqrt{3}/2,0,\sqrt{3}/2$，于是在 $[0,1]$ 区间取 x_i 分别为 $\frac{2-\sqrt{3}}{4},\frac{1}{2},\frac{2+\sqrt{3}}{4}$, 这时在 $[0,1]$ 区间插值的平均绝对误差约为 0.015。

也可以考虑用有理函数插值。设

$$f(x)=\frac{a_0+a_1x}{1+b_1x},$$

令 $f(x_i)=z_i$, $i=1,2,3$，仍取 x_i 为 $1/16,1/4,1$, 可得方程

$$\begin{aligned}a_0+\frac{1}{16}a_1=&\frac{1}{4}\left(1+\frac{1}{16}b_1\right)\\a_0+\frac{1}{4}a_1=&\frac{1}{2}\left(1+\frac{1}{4}b_1\right)\\a_0+a_1=&1\cdot(1+b_1)\end{aligned}$$

解得

$$f(x)=\frac{\frac{1}{7}+2x}{1+\frac{8}{7}x},$$

这时在 $[0,1]$ 区间插值的平均绝对误差约为 0.014。

R 的 polynom 包的 `poly.calc(x,y)` 用给定的 x, y 坐标计算插值多项式。

22.3 牛顿差商公式 (*)

牛顿差商公式是另外一种给出插值多项式表达式的方法。对函数 $h(x)$，分点 $x_i, i = 1, \dots, n$，定义各阶差商为：

$$\begin{aligned}
h[x_i] =& h(x_i) \\
h[x_i, x_j] =& \frac{h(x_j) - h(x_i)}{x_j - x_i} \\
h[x_i, x_j, x_k] =& \frac{h[x_j, x_k] - h[x_i, x_j]}{x_k - x_i} \\
& \cdots\cdots \\
h[x_1, x_2, \dots, x_n] =& \frac{h[x_2, x_3, \dots, x_n] - h[x_1, x_2, \dots, x_{n-1}]}{x_n - x_1}
\end{aligned}$$

则 $h[x_1, x_2, \dots, x_n]$ 关于自变量对称，即以任意自变量次序计算 $h[x_1, x_2, \dots, x_n]$ 结果都相同。另外，在定理22.2条件下存在 $\xi \in [\min\{x_i\}, \max\{x_i\}]$ 使得

$$h[x_1, x_2, \dots, x_n] = \frac{h^{(n-1)}(\xi)}{(n-1)!}.$$

利用牛顿差商公式也可以得到多项式插值公式

$$\begin{aligned}
h(x) =& P_{n-1}(x) + R_n(x) \\
P_{n-1}(x) =& h(x_1) + (x - x_1)h[x_1, x_2] \\
& + (x - x_1)(x - x_2)h[x_1, x_2, x_3] \\
& + \cdots \\
& + (x - x_1)(x - x_2)\cdots(x - x_{n-1})h[x_1, x_2, \dots, x_n] \\
R_n(x) =& (x - x_1)(x - x_2)\cdots(x - x_{n-1})(x - x_n) \\
& \cdot h[x_1, x_2, \dots, x_n, x]
\end{aligned}$$

其中 $P_{n-1}(x)$ 是 $n-1$ 次插值多项式，根据定理22.1可知 $P_{n-1}(x)$ 与 Lagrange 插值多项式相同。$R_n(x)$ 为余项，关于余项估计有:

$$h[x_1, x_2, \dots, x_n, x] = \frac{h^{(n)}(\eta)}{n!},$$

其中 $\eta \in [\min(x_1, x_2, \dots, x_n, x), \max(x_1, x_2, \dots, x_n, x)]$。

22.4 样条插值介绍 (*)

如果有 n 个点 $\{(x_i, z_i), i = 1, 2, \dots, n\}$ 要插值，当 n 较大时，用 $n-1$ 阶插值多项式可以通过这些点，但是得到的曲线会过于复杂, 与真实值差距很大，如果用插值函数计算 $\{(x_i, z_i), i = 1, 2, \dots, n\}$ 所在区间外的点则误差更大。如果仅仅用线段连接或者用抛物线连接，那么在两段交界的地方又不够光滑。图22.1用几种方法对函数 $h(x) = 1/(1 + x^2)$ 在 $x \in [-4, 4]$ 进行了插值，只用到了 7 个已知点。从例图可以看出，线性插值可能在连接两段的地方形成尖角转弯，抛物线插值在两段连接的地方也有明显转折，用通过给定的 7 个点的 6 阶多项式插值，出现了很大的波折，与真实函数差距较大。**样条插值**也是用分段的多项式进行插值，但样条插值保证连接处是光滑的，同时构成的曲线又不会太复杂。

设已知 n 个点 $\{(x_i, z_i), i = 1, \dots, n\}$，$x_1 < x_2 < \cdots < x_n$, 求有二阶连续导数的 $S(x)$ 使得 $S(x)$ 通过这 n 个点，即求 $S(x)$ 使

$$\begin{cases} S(x_i) = z_i, i = 1, \dots, n \\ \min\limits_{S} \int |S''(x)|^2 dx \end{cases}$$

满足条件的函数就是三次样条函数。三次样条函数是分段函数，以各 x_i 为分界点，称 $\{x_1, x_2, \dots, x_n\}$ 为结点 (knots)，三次样条函数 $S(x)$ 在每一段内为三次多项式，其一阶导数 $S'(x)$ 连续，为分段二次多项式，其二阶导数 $S''(x)$ 连续，为分段线性函数。

样条插值优点是保证了曲线光滑，可以对比较光滑的各种形状的函数进行插值同时插值曲线不会太复杂，得到的函数表达式可以用于数值积分和数值微分。

为了求样条函数 $S(x)$ 的每一段的表达式，记 $d_j = x_j - x_{j-1}, j = 2, \dots, n$。因为 $S''(x)$ 分段线性，可记 $S''(x_j) = M_j$，用线性插值公式，对 $x \in [x_{j-1}, x_j]$ 有

$$S''(x) = \frac{M_{j-1}(x_j - x)}{d_j} + \frac{M_j(x - x_{j-1})}{d_j},$$

积分得

$$\begin{aligned} S'(x) =& c_1 + (-1)\frac{M_{j-1}(x_j - x)^2}{2d_j} + \frac{M_j(x - x_{j-1})^2}{2d_j}, \\ S(x) =& c_1 x + c_2 + \frac{M_{j-1}(x_j - x)^3}{6d_j} + \frac{M_j(x - x_{j-1})^3}{6d_j}, \end{aligned}$$

改写为

$$S(x) = \frac{M_{j-1}(x_j - x)^3 + M_j(x - x_{j-1})^3}{6d_j} + c_1' \frac{x_j - x}{d_j} + c_2' \frac{x - x_{j-1}}{d_j}.$$

根据 $S(x_{j-1}) = z_{j-1}$ 和 $S(x_j) = z_j$ 解出 c_1', c_2'，有

$$\begin{aligned} S(x) =& \frac{M_{j-1}(x_j - x)^3 + M_j(x - x_{j-1})^3}{6d_j} \\ &+ \left(z_{j-1} - \frac{M_{j-1}d_j^2}{6} \right) \frac{x_j - x}{d_j} + \left(z_j - \frac{M_j d_j^2}{6} \right) \frac{x - x_{j-1}}{d_j}, \\ & x \in [x_{j-1}, x_j]. \end{aligned}$$

只要求出 $M_j, j = 1, \ldots, n$，则三次样条插值函数 $S(x)$ 在每一段的表达式就完全确定。

用 $S'(x_j - 0) = S'(x_j + 0), j = 2, \ldots, n-1$ 可以得到 $\{M_j\}$ 的 $n-2$ 个线性方程，每一方程只涉及 M_{j-1}, M_j, M_{j+1}. 再增加两个线性限制条件可以解出 $\{M_j, j = 1, \ldots, n\}$。比如，加限制条件 $M_1 = 0, M_n = 0$，即在边界处函数为线性函数，这样解出的样条插值函数称为**自然样条**。

R 中样条插值函数为 `spline(x, y, n, method="natural")`, 结果为包含元素 `x` 和 `y` 的列表，这是在等间隔的 x 点上计算的样条插值结果。

样条函数除了可以用于插值，还可以用于平滑点 $(x_i, z_i), i = 1, 2, \ldots, n$。插值函数需要穿过所有已知点，而平滑则只要与已知点很接近就可以了。用样条函数在点 $(x_i, z_i), i = 1, 2, \ldots, n$ 处进行平滑叫做样条回归，结果是以 $x_1, x_2, \ldots, x_n$ 为结点的三次样条函数，但在 x_i 处的值不需要等于 z_i。样条回归是使得

$$Q(S) = \sum_{i=1}^{n} (z_i - S(x_i))^2 + \lambda n \int |S''(x)|^2 dx$$

最小的函数，其中 $\lambda > 0$ 是代表光滑程度的参数，λ 越大得到的样条函数越光滑。

R 的 `smooth.spline()` 函数计算样条平滑。

样条函数用作回归函数时也可以不取已知点 $\{(x_i, z_i), i = 1, 2, \ldots, n\}$ 的横坐标为结点，而是单独取结点 $k_1, k_2, \ldots, k_m$，一般 m 比 n 小得多，结点 k_j 作为回归关系改变的分界点。

习题

习题 1

设计 R 程序，用向量保存多项式的系数，设计多项式加减法、乘法、除法的程序（除法结果包括商和余式）。

习题 2

编写 R 程序，对输入的 n 个点在给定的自变量 x 处进行抛物线插值，x 为向量。考虑 n 个点的自变量等距和不等距两种情况。

Chapter 23

数值积分

23.1 数值积分的用途

在11给出了用随机模拟积分的方法。随机模拟积分在高维或者积分区域不规则时有优势，在低维、积分区域规则且被积函数比较光滑时用数值积分方法可以得到更高精度，也不会引入随机误差。

积分、最优化、矩阵计算都是在统计问题中最常见的计算问题，在统计计算中经常需要计算积分。比如，从密度 $p(x)$ 计算分布函数 $F(x)$，如果没有解析表达式和精确的计算公式，需要用积分来计算：

$$F(x) = \int_{-\infty}^{x} p(u)du,$$

用积分给出部分函数值后可以用插值和函数逼近得到 $F(x)$ 的近似公式。

已知联合密度 $p(x_1, x_2)$ 要求边缘密度 $p(x_1)$，要用积分计算

$$p(x_1) = \int p(x_1, x_2)dx_2.$$

贝叶斯分析的主要问题是已知先验密度 $\pi(\theta)$ 和似然函数 $p(x|\theta)$ 后求后验密度 $p(\theta|x)$:

$$p(\theta|x) = \frac{p(\theta, x)}{p(x)} = \frac{\pi(\theta)p(x|\theta)}{\int \pi(u)p(x|u)du},$$

大多数情况下不能得到后验密度 $p(\theta|x)$ 的解析表达式，也可能需要计算积分，用后验密度求期望、平均损失函数也需要计算积分。

R 的 `integrate(f, lower, upper)` 函数可以计算有限区间或无穷区间定积分，如果要指定无穷区间，取 `lower` 为`-Inf` 或取 `upper` 为 `Inf`。R 的 MASS 包中的 `area` 函数可以计算有限区间上的定积分，结果比 `integrate` 更为可靠。

对于可以积分得到显示表达式的情况，可以用一些符号计算软件来辅助推导积分，这样的软件有 Mathematica, Maple, Maxima 等，其中 Maxima 是自由软件。

23.2 一维数值积分

数值积分的最简单方法是直接用达布和计算。更精确的积分方法是对被积函数进行多项式逼近然后对近似多项式用代数方法求积分。这些近似多项式可以不依赖于被积函数，只需要用被积函数的若干值。多项式逼近可以是在全积分区间上进行，也可以把积分区间分为很多小区间在小区间上逼近。分为小区间的方法适用性更好。

23.2.1 计算达布和的积分方法

为求定积分

$$I = \int_a^b f(x)dx,$$

把区间 $[a, b]$ 均匀分为 n 段，分点为 $x_0 = a, x_1, \ldots, x_{n-1}, x_n = b$，间隔为 $h = (b - a)/n$, 可以用

$$D_n = \sum_{i=1}^{n} f(x_i)h \tag{23.1}$$

近似计算 I。把 D_n 改写成

$$D_n = \sum_{i=0}^{n-1} f(x_{i+1})h$$

可以看出相当于把曲边梯形面积

$$\int_{x_i}^{x_{i+1}} f(x)dx$$

用以 $[x_i, x_{i+1}]$ 为底、右侧高度 $f(x_{i+1})$ 为高的矩形面积近似（见图23.1)。(23.1)中的 $f(x_i)$ 是小区间右端点的函数值。

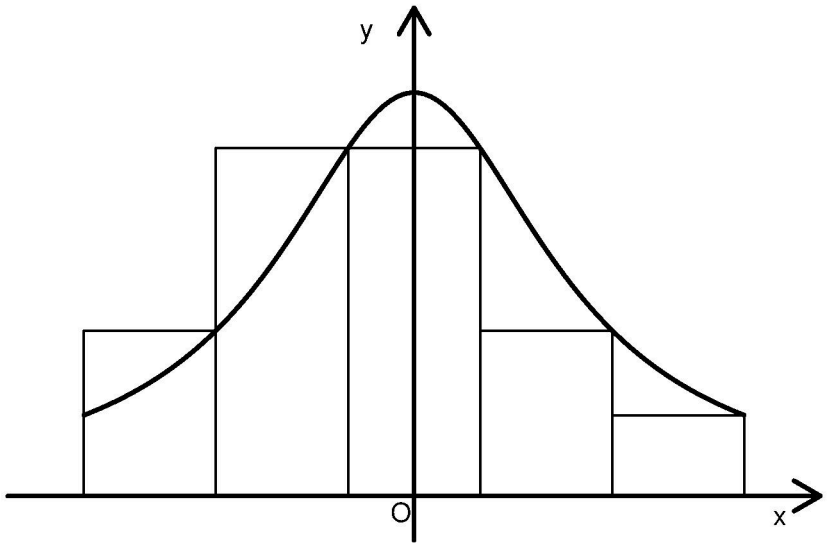


图 23.1: 达布和数值积分图示

达布和积分方法容易理解，但用区间端点近似整个区间的函数值误差较大，精度不好，基本不使用公式(23.1)计算一元函数积分。如果使用区间中点作为代表，公式变成

$$M_n = h\sum_{i=0}^{n-1} f\left(a + \left(i + \frac{1}{2}\right)h\right) \quad (h = \frac{b-a}{n}) \tag{23.2}$$

这个公式称为中点法则，余项为

$$R_n = I - M_n = \frac{(b-a)^3}{24n^2} f''(\xi),\ \xi \in [a, b], \tag{23.3}$$

当 $f''(x)$ 有界时中点法则的精度为 $O(n^{-2})$，精度比(23.1)高得多，与下面的梯形法则精度相近。

23.2.2 梯形法则

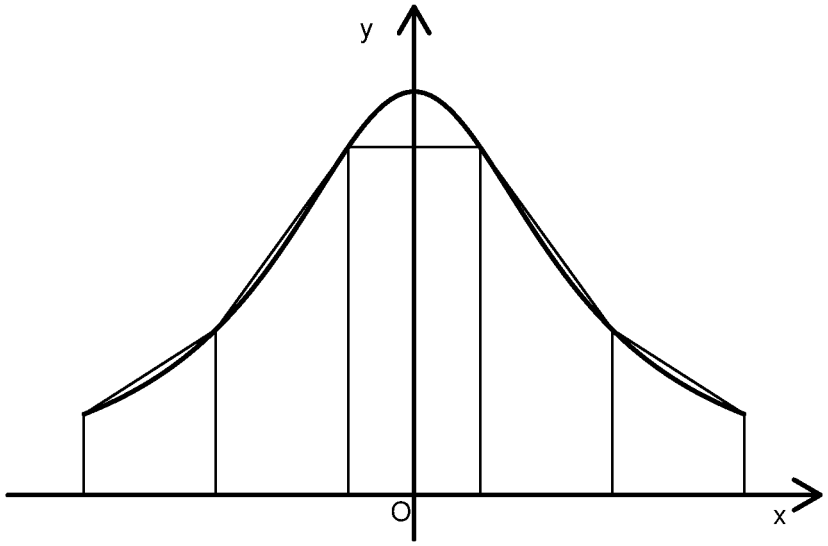


图 23.2: 梯形法则数值积分图示

在小区间 $[x_i, x_{i+1}]$ 内不是用常数而是用线性插值代替 $f(x)$，即用梯形代替曲边梯形 (参见图23.2)，可以得到如下的积分公式

$$\begin{aligned} f(x) \approx & f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{h}(x - x_i),\ x \in [x_i, x_{i+1}] \\ \int_{x_i}^{x_{i+1}} f(x)dx \approx & \frac{f(x_i) + f(x_{i+1})}{2} h \end{aligned} \tag{23.4}$$

$$\begin{aligned} \int_a^b f(x)\,dx = & \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\,dx \\ \approx & \frac{h}{2}\left\{ f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b) \right\} \triangleq T_n \end{aligned} \tag{23.5}$$

余项为

$$R_n = I - T_n = -\frac{(b-a)^3}{12n^2} f''(\xi), \xi \in (a, b).$$

(23.5)叫做复合梯形公式，在 $f''(x)$ 有界时算法精度为 $O(n^{-2})$。

23.2.3 辛普森 (Simpson) 法则 (*)

在等距的 (x_{-1}, x_0, x_1) 的区间中用抛物线插值公式近似 $f(x)$:

$$\begin{aligned} f(x) \approx & \frac{1}{2}\left(f(x_{-1}) - 2f(x_0) + f(x_1)\right)\left(\frac{x-x_0}{h/2}\right)^2 \\ & + \frac{1}{2}\left(f(x_1) - f(x_{-1})\right)\left(\frac{x-x_0}{h/2}\right) + f(x_0) \end{aligned}$$

其中 $h = x_1 - x_{-1} = 2(x_0 - x_{-1})$, 积分得

$$\int_{x_{-1}}^{x_1} f(x)\,dx = \frac{h}{6}\left\{f(x_{-1}) + 4f(x_0) + f(x_1)\right\}, h = x_1 - x_{-1}. \tag{23.6}$$

把区间 $[a,b]$ 等分为 n 份, 记 $h = (b-a)/n$, 记 $x_i = a + ih$, $i = 0, 1, 2, \ldots, n$, 则$_{x_0, x_1, \ldots, x_n}$把$[a,b]$ 等分为 n 份，然后在每个小区间内取中点，记为 $x_{i+\frac{1}{2}} = a + (i + \frac{1}{2})h$, $i = 0, 1, \ldots, n-1$, 在 $[x_i, x_{i+1}]$ 内用公式(23.6)并把 n 个小区间的积分相加，得

$$\begin{aligned} I = & \int_a^b f(x)\,dx \\ \approx & \frac{h}{6}\left(f(a) + 4\sum_{i=0}^{n-1} f(x_{i+\frac{1}{2}}) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\right) \triangleq S_n, \end{aligned} \tag{23.7}$$

余项为

$$R_n = I - S_n = -\frac{(b-a)^5}{2880n^4} f^{(4)}(\xi), \xi \in (a,b).$$

(23.7)叫做复合辛普森公式，在 $f^{(4)}(x)$ 有界时复合辛普森公式误差为 $O(n^{-4})$。

23.2.4 牛顿-柯蒂斯 (Newton-Cotes 公式)(*)

公式(23.4)和(23.6)分别是在小区间用线性插值和抛物线插值近似被积函数 $f(x)$ 得到的积分公式。一般地，对等距节点 $(x_0 = a, x_1, \ldots, x_n = b)$ 可以进行 Lagrange 插值，得到 n 阶插值多项式 $P_n(x)$:

$$P_n(x) = \sum_{j=0}^{n} f(x_j) \frac{\prod_{k\neq j}(x - x_k)}{\prod_{k\neq j}(x_j - x_k)}$$

用 $P_n(x)$ 在 $[a,b]$ 上的积分近似 $\int_a^b f(x)dx$，有

$$\begin{aligned}\int_a^b f(x)\,dx \approx& \int_a^b P_n(x)\,dx \\ =&\sum_{j=0}^{n}\left\{\int_a^b \frac{\prod_{k\neq j}(x-x_k)}{\prod_{k\neq j}(x_j-x_k)}dx\right\} f(x_j)\end{aligned}$$

注意到 $x_j = a+jh$, $h=(b-a)/n$, 做变量替换 $x=a+th$, 上式变成

$$\begin{aligned}\int_a^b f(x)\,dx \approx&\frac{b-a}{n}\sum_{j=0}^{n}\left\{\int_0^n \frac{\prod_{k\neq j}(t-k)}{\prod_{k\neq j}(j-k)}dt\right\} f(x_j) \\ =&(b-a)\sum_{j=0}^{n} C_j^{(n)} f(x_j)\end{aligned} \tag{23.8}$$

这是被积函数在节点上函数 $f(x_i)$ 值的线性组合，其中各线性组合系数 $C_j^{(n)}$ 不依赖于 f 和积分区间:

$$C_j^{(n)} = \frac{1}{n}\int_0^n \frac{\prod_{k\neq j}(t-k)}{\prod_{k\neq j}(j-k)}dt = \frac{1}{n}\frac{(-1)^{n-j}}{j!(n-j)!}\int_0^n \prod_{k\neq j}(t-k)dt. \tag{23.9}$$

公式(23.8)称为牛顿-柯蒂斯 (Newton-Cotes) 公式, 公式(23.4)和(23.6)分别是 $n=1$ 和 $n=2$ 的特例。一些 $C_j^{(n)}$ 的值见表23.1。$n=4$ 的牛顿-柯蒂斯公式叫做柯蒂斯法则，积分公式为

$$I=\int_a^b f(x)dx \approx \frac{b-a}{90}\left\{7f(x_0)+32f(x_1)+12f(x_2)+32f(x_3)+7f(x_4)\right\}, \tag{23.10}$$

$$x_i = a+i\cdot\frac{b-a}{4},\ i=0,1,2,3,4. \tag{23.11}$$

复合柯蒂斯公式把 $[a,b]$ 等分为 n 份，令 $h=(b-a)/n$, $x_i=a+ih, i=0,1,\ldots,n$, 然后把每个小区间 $[x_i,x_{i+1}]$ 等分为 4 份，内部分点为 $x_{i+\frac{1}{4}}=x_i+\frac{1}{4}h$, $x_{i+\frac{1}{2}}=x_i+\frac{1}{2}h$, $x_{i+\frac{3}{4}}=x_i+\frac{3}{4}h$, $i=0,1,\ldots,n-1$，在每个小区间 $[x_i,x_{i+1}]$ 内用(23.11)积分然后相加，得复合柯蒂斯公式

$$\begin{aligned}\int_a^b f(x)dx \approx&\frac{h}{90}\left\{7f(a)+32\sum_{i=0}^{n-1}f(x_{i+\frac{1}{4}})+12\sum_{i=0}^{n-1}f(x_{i+\frac{1}{2}})\right. \\ &\left.+32\sum_{i=0}^{n-1}f(x_{i+\frac{3}{4}})+14\sum_{i=1}^{n-1}f(x_i)+7f(b)\right\}\end{aligned}$$

余项为

$$R = -\frac{(b-a)^7}{1013760n^6} f^{(6)}(\xi),\ \xi \in (a,b).$$

表 23.1: 牛顿-柯蒂斯积分系数

n	$C_j^{(n)}$
1	$\frac{1}{2},\ \frac{1}{2}$
2	$\frac{1}{6},\ \frac{2}{3},\ \frac{1}{6}$
3	$\frac{1}{8},\ \frac{3}{8},\ \frac{3}{8},\ \frac{1}{8}$
4	$\frac{7}{90},\ \frac{32}{90},\ \frac{12}{90},\ \frac{32}{90},\ \frac{7}{90}$
5	$\frac{19}{288},\ \frac{75}{288},\ \frac{50}{288},\ \frac{50}{288},\ \frac{75}{288},\ \frac{19}{288}$
6	$\frac{41}{840},\ \frac{216}{840},\ \frac{27}{840},\ \frac{272}{840},\ \frac{27}{840},\ \frac{216}{840},\ \frac{41}{840}$

虽然我们可以得到高阶的牛顿-柯蒂斯公式，但是高阶的插值多项式一般在边界处会有很大误差，实际中我们一般最多用到 4 阶，然后用复合方法把 $[a,b]$ 划分为小区间，在小区间内用低阶的牛顿-柯蒂斯公式。

23.2.5 数值积分的代数精度 (*)

为计算积分

$$I = \int_a^b f(x)d\,x,$$

可以在 $[a,b]$ 的 n 个节点 $a \le x_1 < x_2 < \cdots < x_n \le b$ 上对 $f(x)$ 用 $n-1$ 阶多项式进行插值，用插值多项式的积分近似积分 I。由 Lagrange 插值公式(22.4)得

$$I \approx \sum_{j=1}^{n} \left\{ \int_a^b \frac{\prod_{k\neq j}(x-x_k)}{\prod_{k\neq j}(x_j-x_k)} d\,x \right\} f(x_j) \tag{23.12}$$

$$= \sum_{j=1}^{n} A_j f(x_j), \tag{23.13}$$

其中 $\{A_j\}$ 不依赖于被积函数 $f(x)$。

在理想情况下(23.13)可以有较高精度。如果某插值方法对不超过 $n-1$ 次的多项式可精确表示而不能精确表示 n 次多项式，则称该插值方法有 $n-1$ 次**代数**

精度。(23.13)若对不超过 $n-1$ 次的多项式积分可以得到准确值而 n 次则不行，则称此数值积分方法具有 $n-1$ 次代数精度。(23.13)至少具有 $n-1$ 次代数精度, 所以牛顿-柯蒂斯积分公式(23.8) 至少具有 n 次代数精度。

23.2.6 高斯-勒让德 (Gauss-Legendre) 积分公式 (*)

公式(23.13)至少有 $n-1$ 次代数精度。在(23.13)中适当选取节点 $\{x_j\}$ 的位置可以得到更高的代数精度, n 个节点可以达到 $2n-1$ 次代数精度, 还可以在无穷区间上积分。这样的数值积分方法叫做高斯型数值积分。

对至多 $2n-1$ 次多项式 $f(x)$，取 n 个节点 $\{x_k\}$ 为 n 次 Legendre 正交多项式 $P_n(x)$ 的 n 个零点，即

$$P_n(x) = c(x-x_1)\cdots(x-x_n),$$

则

$$f(x) = q(x)P_n(x) + r(x)$$

其中 $q(x)$ 和 $r(x)$ 都是至多 $n-1$ 次的多项式。由正交多项式性质, $P_n(x)$ 与任意至多 $n-1$ 次多项式都正交，于是

$$\int_{-1}^{1} q(x)P_n(x)\,dx = 0.$$

故

$$\int_{-1}^{1} f(x)\,dx = \int_{-1}^{1} r(x)\,dx,$$

而 $r(x)$ 为至多 $n-1$ 阶，其插值方法的积分是精确的，即

$$\int_{-1}^{1} f(x)\,dx = \int_{-1}^{1} r(x)\,dx = \sum_{k=1}^{n} A_k r(x_k)$$

等式成立，注意

$$P_n(x_k) = 0, k = 1, 2, \ldots, n,$$

所以 $r(x_k) = f(x_k)$, 于是

$$\int_{-1}^{1} f(x)\,dx = \sum_{k=1}^{n} A_k f(x_k)$$

即当 $f(x)$ 为至多 $2n-1$ 次多项式时这样选取节点的 @ref{eq:fun-quad-interp} 是精确的。可预先把 x_k 和 A_k 做表。

对于其他的积分区间及权函数 $w(x)$，要计算

$$I = \int_a^b f(x)w(x)\,dx,$$

也可以可以利用相应的正交多项式及零点解决。比如 Gauss-Laguerre 积分

$$\int_0^\infty f(x)e^{-x}\,dx$$

可以利用 Laguerre 多项式的零点作为节点进行插值，Gauss-Hermite 积分

$$\int_{-\infty}^\infty f(x)e^{-x^2}dx$$

可以利用 Hermite 多项式的零点作为节点进行插值。这样可以计算无穷区间上的积分。

高斯型积分适用于函数足够光滑，需要反复计算积分但每次计算被积函数值都耗时很多的情况，不适用于一般的被积函数。

R 的 gss 包的 `gauss.quad()` 函数可以对给定点数和区间端点计算高斯-勒让德积分的节点位置和相应积分权重。

23.2.7 变步长积分 (*)

对一般的函数，提高插值多项式阶数并不能改善积分精度，复合方法如(23.5)和(23.7)则取点越多精度越高。因为很难预估需要的点数，所以我们可以逐步增加取点个数直至达到需要的积分精度。一种节省取点个数的方法是每次取点仅增加原来小区间的中点，这样点数每次增加一倍但是仅需要再多计算一半点上的函数值。

比如，使用复合梯形公式(23.5):

$$\begin{aligned} T_n =& \frac{d}{2}\left\{f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\right\}, \\ & x_i = a + id,\ i = 0, 1, \dots, n,\ d = \frac{b-a}{n}, \end{aligned}$$

变步长为 $d/2$，节点增加到 $2n+1$ 个，原来的一个小区间 $[x_i, x_{i+1}]$ 中又增加了一个分点 $x_{i+\frac{1}{2}} = x_i + \frac{d}{2}$，原来在 $[x_i, x_{i+1}]$ 的梯形法则积分结果为

$$I_1 = \frac{d}{2}\left\{f(x_i) + f(x_{i+1})\right\},$$

增加 $x_{i+\frac{1}{2}}$ 节点后在 $[x_i, x_{i+1}]$ 的积分变成 $x_{i+\frac{1}{2}}$ 前后两半的积分之和，为

$$\begin{aligned} I_2 =& \frac{d}{4}\left\{f(x_i) + f(x_{i+\frac{1}{2}})\right\} + \frac{d}{4}\left\{f(x_{i+\frac{1}{2}}) + f(x_{i+1})\right\} \\ =& \frac{d}{4}\left\{f(x_i) + 2f(x_{i+\frac{1}{2}}) + f(x_{i+1})\right\} \\ =& \frac{1}{2}I_1 + \frac{d}{2}f(x_{i+\frac{1}{2}}) \end{aligned}$$

于是 $n+1$ 个节点的梯形公式结果 T_n 与 $2n+1$ 个节点的 T_{2n} 的关系为

$$T_{2n} = \frac{1}{2}T_n + \frac{d}{2}\sum_{i=0}^{n-1} f(x_{i+\frac{1}{2}}). \tag{23.14}$$

只需要计算新增的 n 个节点上的函数值。实际计算积分时预先确定一个误差限 ε，当 $|T_{2n} - T_n| < \varepsilon$ 时停止增加节点, 以 T_{2n} 为积分近似值。注意，如果被积函数在很大范围基本不变，比如等于零，这样的停止规则可能给出完全错误的积分计算结果。

23.3 多维数值积分 (*)

多元函数 $f(x_1, x_2, \ldots, x_d)$ 的积分比一元函数积分困难得多，这时很难再取多元的插值多项式然后对插值函数积分，计算量也要比一维积分大得多。

对定义在矩形 $[a,b]\times[c,d]$ 上的二元函数 $f(x,y)$，为计算二重积分

$$I = \int_a^b \int_c^d f(x,y)\, dy\, dx$$

可以用如下的中点法则近似：

$$\begin{aligned} M_{n,m} =& \frac{b-a}{n}\frac{d-c}{m}\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} f\left(a + \left(i+\frac{1}{2}\right)h,\ c + \left(j+\frac{1}{2}\right)g\right), \\ & \left(h = \frac{b-a}{n},\ g = \frac{d-c}{m}\right). \end{aligned}$$

对于 R^d 空间中超立方体上的积分也可以类似计算。

高维空间的积分需要大量的计算，计算量可能会大到不适用。比如，如果一维需要 n 个节点，计算 n 次被积函数值，那么 d 维积分就需要计算 $N = n^d = e^{d \ln n}$ 次被积函数值，计算量随着 d 的增长而指数增长。随机模拟积分 (见11) 的精度为 $O_p(N^{-1/2})$，受维数 d 的影响较小，但高维时也需要计算很多点上的函数值才能相对准确地计算积分。

拟随机积分是另外一种计算积分的方法，它结合了数值积分和随机模拟积分的优点，在特殊选取的格子点上计算函数值并计算积分，这样的布点位置称为拟随机数 (quasi-random)。我国数学家方开泰和王元 [Fang and Wang, 1994] 开创了基于数论并在试验设计、高维积分有良好应用效果的 “均匀设计” 学科，利用他们提出的拟随机数布点可以进行高维积分计算。均匀设计布点有 “均匀分散” 和 “整齐可比” 两个特点，使布点既具有代表性，又是有规律的。随机模拟积分的随机误差界为 $O(\sqrt{N^{-1} \log\log N})$，而拟随机积分的误差可以达到 $O(N^{-1}(\log N)^d)$ (见 Monahan [2001] §10.6, Lemieux [2009])。

习题

习题 1

证明复合辛普森积分公式(23.7)。

习题 2

编写变步长梯形法则积分算法和相应 R 程序。

习题 3

把变步长梯形法则积分算法推广到 $[0, \infty)$ 上的定积分：设已用 n 等分计算 $[0, b]$ 上的积分，下一步增加原有 n 个区间的中点作为节点，并在 $(b, 2b]$ 增加 $2n$ 个节点，计算 $[0, 2b]$ 上的定积分，直到结果变化小于给定的误差限 ε。用 $\int_0^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} dx = \frac{1}{2}$ 验证。

习题 4

在贝叶斯分析中先验分布为 $\pi(\theta) = 6\theta(1-\theta)$, 似然函数为 $L(\theta) \propto \theta^{15}/(-\log(1-\theta))^{10}$, 后验密度为（差一个比例系数）

$$p^*(\theta) \propto \theta^{16}(1-\theta)(-\log(1-\theta))^{10}.$$

分别对 $h_0(\theta) = 1$, $h_1(\theta) = \theta$, $h_2(\theta) = \theta^2$ 计算数值积分 $\int_0^1 h_i(\theta)p^*(\theta)d\theta$, 分别用中点法则、复合梯形法则和复合辛普森法则计算，比较达到小数点后 1 位的精度所需要的节点个数。

Chapter 24

数值微分

24.1 三种差商方法

在没有函数微分的解析表达式时，可以用数值方法由函数值近似计算函数的微分值。按照微分定义，当 $f(x)$ 在点 x 处可微时，对很小的 h 有

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h) \tag{24.1}$$

$$= \frac{f(x) - f(x-h)}{h} + O(h) \tag{24.2}$$

$$= \frac{f(x+h) - f(x-h)}{2h} + O(h^2) \tag{24.3}$$

这三种近似分别称为数值微分的**向前差商**、**向后差商**和**中心差商公式**, 其中中心差商公式有更好的精度。h 选取应该很小，使得继续减小 h 时按差商公式计算的近似导数值不再变化或变化量小于给定误差界限。但是，h 也不能过分小，因为计算 $f(x)$ 值只能到一定精度，当 h 太小时 $f(x+h)$ 与 $f(x)$ 的差别已经不是由 h 引起的而是由 $f(x)$ 计算误差引起的。如果 $f(x)$ 的计算精度可以达到机器单位 U 量级，取 h 的一个经验法则是取 $h/|x|$ 为 $U^{1/2}$ 左右, 对中心差商公式取 $h/|x|$ 为 $U^{1/3}$ 左右 (见 Monahan [2001] §8.6)。多元函数的梯度可以类似地计算。

可以类似地计算高阶微分，如

$$f''(x) = \frac{[f(x+h) - f(x)] - [f(x) - f(x-h)]}{h^2} + O(h), \tag{24.4}$$

公式分子有意地写成了两个差分的差分，以避免有效位数损失。

公式(24.1)–(24.3)是用割线斜率近似切线斜率，相当于用两点的函数值作线性插值后用插值函数的微分近似原始函数微分。一般地，对函数 $f(x)$ 可以计算插值多项式 $P_n(X)$，用 $P_n'(x)$ 近似 $f'(x)$。这种计算微分的方法称为**插值型求导公式**。设多项式 $P_{n-1}(x)$ 满足

$$P_{n-1}(x_i) = f(x_i),\ i = 1, 2, \dots, n$$

由定理22.2, 插值近似的余项为

$$f(x) - P_{n-1}(x) = \frac{f^{(n)}(\xi)}{n!} g(x)$$

其中 $g(x) = \prod_{i=1}^{n}(x - x_i)$, ξ 依赖于 x 的位置。插值型求导公式的余项为

$$f'(x) - P_{n-1}'(x) = \frac{f^{(n)}(\xi)}{n!} \cdot g'(x) + \frac{g(x)}{n!} \cdot \frac{d}{dx}\frac{f^{(n)}(\xi)}{n!} \tag{24.5}$$

因为(24.5)中的第二项中 ξ 与 x 的函数关系未知，对任意 x 处用 $P_{n-1}'(x)$ 近似 $f'(x)$ 可能有很大误差。如果仅在节点 $x_i, i = 1, 2, \dots, n$ 处计算 $P_{n-1}'(x)$，则有

$$f'(x_i) = P_n'(x_i) + \frac{f^{(n)}(\xi)}{n!} g'(x_i),\ i = 1, 2, \dots, n. \tag{24.6}$$

根据(24.6), 我们导出对等距节点 $(x_i, z_i)(\ i = 1, 2, \dots, n)$ 计算数值微分的公式。设 $z_i = f(x_i)$，$x_i - x_{i-1} = h$。

用线性插值近似求导的公式为

$$f'(x_i) = \frac{z_{i+1} - z_i}{h} - \frac{h}{2} f''(\xi_1),\ i = 1, 2, \dots, n-1\ (\xi_1 \in (x_i, x_{i+1})$$

或

$$f'(x_i) = \frac{z_i - z_{i-1}}{h} + \frac{h}{2} f''(\xi_2),\ i = 2, 3, \dots, n\ (\xi_2 \in (x_{i-1}, x_i)$$

用二次多项式插值近似求导的公式为

$$\begin{aligned} f'(x_i) =& \frac{-3z_i + 4z_{i+1} - z_{i+2}}{2h} + \frac{h^2}{3} f^{(3)}(\xi_1), \\ & i = 1, 2, \dots, n-2,\ \xi_1 \in (x_i, x_{i+2}) \end{aligned}$$

或

$$\begin{aligned} f'(x_i) =& \frac{-z_{i-1} + z_{i+1}}{2h} - \frac{h^2}{6} f^{(3)}(\xi_2), \\ & i = 2, 3, \dots, n-1,\ \xi_2 \in (x_{i-1}, x_{i+1}) \end{aligned}$$

或

$$f'(x_i) = \frac{z_{i-2} - 4z_{i-1} + 3z_i}{2h} + \frac{h^2}{3} f^{(3)}(\xi_3),$$
$$i = 3, 4, \ldots, n,\ \xi_3 \in (x_{i-2}, x_i).$$

用二次多项式插值近似求二阶导数的公式为

$$f''(x_i) = \frac{z_{i-1} - 2z_i + z_{i+1}}{h^2} - \frac{h^2}{12} f^{(4)}(\xi_1),$$
$$i = 2, 3, \ldots, n-1,\ \xi_1 \in (x_{i-1}, x_{i+1}).$$

用多项式插值方法近似计算函数微分，在一般的 x 值处计算的误差可能很大。可以采用样条函数来进行插值，用样条函数的导数来计算被插值函数的导数。

Mathmatica、Maple、Maxima 等软件系统支持符号运算，可以对用数学表达式表示的函数获得数学表达式表示的微分、积分。

Julia 语言的 ForwardDiff 包使用自动微分算法，可以高效地对 Julia 程序表示的函数计算梯度、海色阵。自动微分不是符号计算，但也不同于简单的差商近似，精度和效率更高。

24.2 复数差商 (*)

如果函数 $f(x)$ 可以对 $x \in \mathbb{C}$(复数域) 计算，则由泰勒展开公式

$$f(x + ih) = f(x) + ihf'(x) - \frac{1}{2}h^2 f''(x) - \frac{1}{6}ih^3 f'''(x) + O(h^4),$$

在两边取虚部，有

$$\Im(f(x + ih)) = hf'(x) - \frac{1}{6}h^3 f'''(x) + O(h^4),$$

于是

$$f'(x) = \frac{\Im(f(x + ih))}{h} + \frac{1}{6}h^2 f'''(x) + O(h^3),$$

可见，不需要计算 $f(x + ih) - f(x)$ 就可以用虚部得到 $O(h^2)$ 精度的近似。同时，因为

$$\Re(f(x + ih)) = f(x) - \frac{1}{2}h^2 f''(x) + O(h^4),$$

所以还得到了 $f(x)$ 的一个精度为 $O(h^2)$ 的近似值。h 的选取也不需要依赖于 x 的绝对值大小。局限是 $f(x)$ 必须对复数有定义且程序允许 x 取复数。

24.3 自动微分 (*)

用计算机程序表示的函数，在许多程序语言中可以使用 “自动微分” 技术，获得微分的精确表示。这种方法的基础是微分的链式法则

$$\frac{dg(f(x))}{dx} = \frac{dg}{df} \cdot \frac{df}{dx},$$

以及实数的一阶精度表示：将函数的每个自变量 x 表示为 $x + h_x\epsilon$，其中 ϵ 是一个虚拟的一阶无穷小量，将运算或者函数的结果 y 用运算规则计算为 y 和 ϵ 的倍数。对于函数 $f(x)$，有

$$f(x + h_x\epsilon) = f(x) + h_x f'(x)\epsilon + o(\epsilon),$$

于是只要将 $f(x + h_x\epsilon)$ 忽略 ϵ 的高阶项近似表示成

$$f(x + h_x\epsilon) = f(x) + h_x f'(x)\epsilon, \tag{24.7}$$

就可以从 ϵ 的系数中提取出 $f'(x)$ 的值。具体方法是在编程时将每个实数都编码为 (x, h_x) 的格式，用来表示 $x + h_x\epsilon + o(\epsilon)$。

在四则运算中，也要对这样编码的实数进行计算。比如，两个这样的实数的乘法：

$$(a + h_a\epsilon) \times (b + h_b\epsilon) = ab + (bh_a + ah_b)\epsilon + h_a h_b\epsilon^2 \approx ab + (bh_a + ah_b)\epsilon.$$

计算机程序中计算的函数，可以看成是四则运算、数学函数的复合。对用程序表示的函数进行改造，使其接受 (x, h_x) 这样的编码格式以及运算规则，对数学函数变换结果也用(24.7)这样的方法进行计算，最后程序函数执行结束后的 $h_x\epsilon$ 的倍数就是所需的导数值。这样的计算不像差商方法那样受步长选取的影响，其误差仅限于数值计算误差，所以可以认为是精确求导。

因为只能对四则运算和常用数学函数进行这样的编码的运算，对一般的用计算机程序表示的函数，有程序分析工具可以将这样的计算机程序表示成有向图，节点是运算或函数，以及输入、输出，连接是复合过程。例如，$\sin(x-1)$ 就可以看成节点 x 和节点 1 用运算 “$-$” 连接，结果可记为 z_1 节点，z_1 节点再连接到 sin 节点进行函数变换。在进行运算和函数调用时都需要按照特殊编码 $x + h_x\epsilon$ 的运算规则来计算。

自动微分的向前累积方法，是从有向图的源头方向开始计算微分，并利用链式法则将计算过程向结果方向传递。在编程时，同时保存分解后的值和微分值。在计算梯度时，每一个分量的微分都需要用向前累积方法分别计算一遍。

自动微分的另一种方法是向后累积方法，计算梯度时只需要沿有向图向前和向后各一遍就可以完成多个分量微分的计算。此方法是从计算结果方向朝源头方向应用链式法则的。

参见 [Kochenderfer]。

24.4 附录：误差阶的推导 (*)

用泰勒展开得

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + O(h^3), \tag{24.8}$$

可得

$$\frac{f(x+h) - f(x)}{h} = f'(x) + \frac{1}{2}hf''(x) + O(h^2),$$

即向前差商公式的误差为 $O(h)$ 阶。向后差商公式类似。

对于中心差商公式，因为

$$f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) + O(h^3), \tag{24.9}$$

将(24.8)与(24.9)相减, 则 $\frac{1}{2}h^2 f''(x)$ 项被约掉，于是有

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + O(h^2).$$

※※※※※

24.5 附录：Julia 程序 (*)

24.5.1 三种差商公式的计算程序和误差演示

用三种差商公式计算数值微分的 Julia 函数如下：

```
diff_forward(f, x; h=sqrt(eps(Float64))) = (f(x+h) - f(x))/h
diff_central(f, x; h=cbrt(eps(Float64))) = (f(x+h/2) - f(x-h/2))/h
diff_backward(f, x; h=sqrt(eps(Float64))) = (f(x) - f(x-h))/h
```

其中 `eps(FLoat64)` 给出双精度值的机器 ε 值，`cbrt()` 是立方根函数。程序没有考虑 x 的大小，实际上，当 x 的绝对值很大时，$x+h$ 会无法与 x 区分开来，这时缺省的 h 值也不合适。

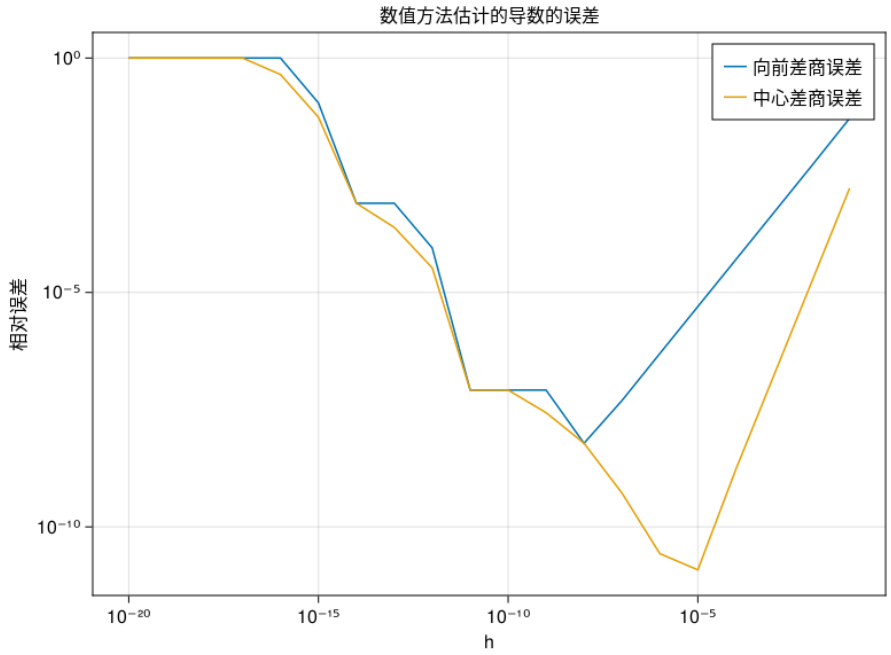
例 24.1. 当 h 太小时，反而会因为 $f(x+h)-f(x)$ 出现两个过于接近的值相减造成有效位数损失，使得估计的微分不准确。以 $f(x)=e^x$ 在 $x=0$ 处的数值微分为例查看相对误差情况。

取 h 为 10 的 -1 次方到 -20 次方，计算误差如下：

```
using CairoMakie
let
    f(x) = exp(x)
    hvec = 10.0 .^ (-1:-1:-20)
    err1 = abs.((f.(0.0 .+ hvec) .- f(0.0)) ./ hvec .- 1.0)
    err2 = abs.((f.(0.0 .+ hvec) .- f.(0.0 .- hvec)) ./
           (2 .* hvec) .- 1.0)
    display([hvec err1 err2])
    fig, ax, plt = lines(hvec, err1,
        axis = (; title=" 数值方法估计的导数的误差",
        xlabel="h", ylabel=" 相对误差",
        xscale=log10, yscale=log10),
        label = " 向前差商误差"    )
    lines!(hvec, err2,
        label = " 中心差商误差"    )
    axislegend()
    fig
end
```

结果：

```
20×3 Matrix{Float64}:
 0.1       0.0517092    0.0016675
 0.01      0.00501671   1.66667e-5
 0.001     0.000500167  1.66667e-7
 0.0001    5.00017e-5   1.66689e-9
 1.0e-5    5.00001e-6   1.21023e-11
 1.0e-6    4.99962e-7   2.67555e-11
 1.0e-7    4.94337e-8   5.26356e-10
 1.0e-8    6.07747e-9   6.07747e-9
 1.0e-9    8.27404e-8   2.72292e-8
 1.0e-10   8.27404e-8   8.27404e-8
 1.0e-11   8.27404e-8   8.27404e-8
 1.0e-12   8.89006e-5   3.33894e-5
 1.0e-13   0.000799278  0.000244166
 1.0e-14   0.000799278  0.000799278
 1.0e-15   0.110223     0.0547119
 1.0e-16   1.0          0.444888
 1.0e-17   1.0          1.0
 1.0e-18   1.0          1.0
 1.0e-19   1.0          1.0
 1.0e-20   1.0          1.0
```



可以看出，h 较大时误差较大，减小 h 可以显著地减小误差，且中心差商公式误差比向前差商公式的误差减小速度快得多；但是，当 h 达到某个范围（向前差商是 10^{-8}，中心差商是 10^{-5}）时，误差不再减小，反而增大。当 $h = 10^{-16}$ 时，向前差商公式估计的 $f'(0)$ 已经等于 0，这是因为这时 $x + h$ 和 x 已经不可区分。

※※※※※

24.5.2 借助于复数自变量计算

如果函数 $f(x)$ 对复数有定义且可微，则计算其实函数的微分的 Julia 程序如下：

```
diff_complex(f, x; h=1e-20) = imag(f(x + h*im)) / h
```

如：

```
diff_complex(exp, 0.0) - 1.0
## 0.0
```

结果完全精确。

24.5.3 自动微分

Julia 有许多包提供了自动微分功能，其中 ForwardDiff 是功能比较完善可靠的。使用自动微分，要求被微分的函数允许输入一般的自变量，而不能指定自变量类型为某种特定的浮点类型。

如：

```
using ForwardDiff
let
    f1(x) = x^3
    f1d(x) = 3 * x^2
    x = [-1, 0, 1, 2]
    @show f1d.(x);
    @show ForwardDiff.derivative.(f1, x);
end
## f1d.(x) = [3, 0, 3, 12]
## ForwardDiff.derivative.(f1, x) = [3, 0, 3, 12]
```

又如函数

$$f(x) = \sum_i \sin(x_i) + \prod_i \sqrt{x_i}.$$

```
f(x) = sum(sin, x) + prod(sqrt, x)
x = [1, 2, 3]
ForwardDiff.gradient(f, x)
## 3-element Vector{Float64}:
##  1.7650471777259729
##  0.19622559914865206
## -0.5817442061365823
```

习题

习题 1

令 $x = (x_1, x_2, x_3, x_4)^T$, $f(x) = [\log(x_1 x_2)/(x_3 x_4)]^2$, $x_i > 0, i = 1, 2, 3, 4$。求 $f(x)$ 的梯度 $\nabla f(x)$ 的表达式，并编写程序用数值微分方法估计 $\nabla f(x)$，比较两者的结果，并比较不同的数值微分方法的结果。

Chapter 25

R 中的近似计算函数 (*)

25.1 多项式

R 用扩展包 polynom 提供了一元多项式类，用多项式系数表示一个多项式，提供了多项式运算、微分、积分等功能。

用 `polynomial(coef)` 生成一个多项式，自变量 `coef` 是多项式的 0 次、1 次、2 次、……项系数的向量。例如，$P(x) = -3 + 2x + x^2$ 可表示为：

```
library(polynom)
P2 = polynomial(c(-3,2,1))
```

也可以利用多项式的根定义多项式，如 $(x+1)(x-1)$:

```
poly.from.roots(c(-1, 1))
```

```
## -1 + x^2
```

用 `as.character(p)` 可以将多项式 `p` 转换成 R 的表达式字符串，如：

```
as.character(P2)
```

```
## [1] "-3 + 2*x + x^2"
```

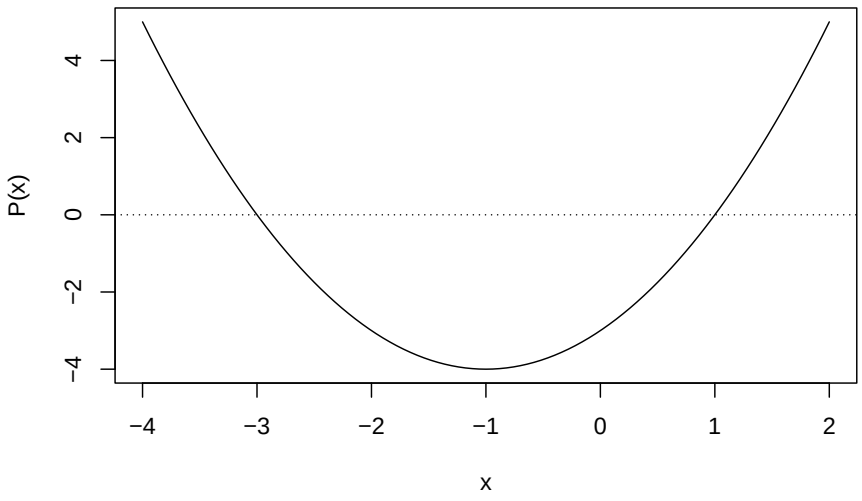
用 `coef(p)` 可以提取多项式的升幂表示的各个系数为一个向量，如

```
coef(P2)
```

```
## [1] -3  2  1
```

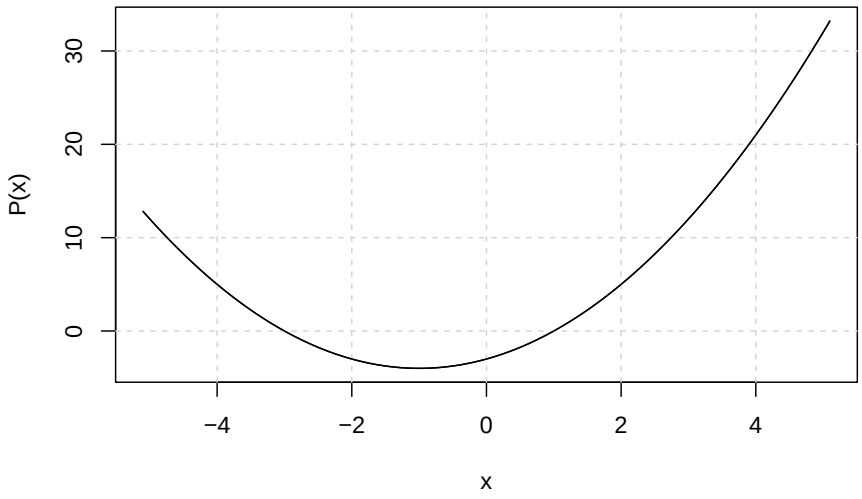
用 `as.function(p)` 可以将多项式转换成一个 R 函数，这时它才能对自变量值求函数值，如

```
curve(as.function(P2)(x), -4, 2, ylab="P(x)"); abline(h=0, lty=3)
```



实际上，可以直接对多项式作图，如

```
plot(P2, xlim=c(-5, 5))
```



多项式可以进行通常的加、减、乘法，如

```
P1 <- polynomial(c(-2, 1))
print(P1 + P2)
```

```
## -5 + 3*x + x^2
```

```
print(P1 * P2)
```

```
## 6 - 7*x + x^3
```

多项式除法比较特殊。除尽时：

```
P2 / polynomial(c(3,1))
```

```
## -1 + x
```

除不尽时：

```
P3 = P2 / polynomial(c(1,1)); print(P3)
```

```
## 1 + x
```

```
P4 = P2 - P3*polynomial(c(1,1)); print(P4)
```

```
## -4
```

这说明

$$\frac{-3+2x+x^2}{1+x}=1+x+\frac{-4}{1+x}$$

可以用函数 GCD() 求多项式的最大公因式，用 LCM() 求多项式的最小公倍式。

用 deriv(p) 求多项式 p 的一阶导数，如

```
deriv(P2)
```

```
## 2 + 2*x
```

用 integral(p) 求多项式 p 的不定积分，看成是 $\int_0^x p(t)\,dt$。integral(p, limits) 则可求定积分，其中 limits 是表示积分上界和下界的两个元素的向量。如

```
integral(P2)
```

```
## -3*x + x^2 + 0.3333333*x^3
```

```
integral(P2, c(-3, 1))
```

```
## [1] -10.66667
```

poly.calc(x, y) 可以从给定的 x 值和 y 值的组合计算相应的 Lagrange 插值多项式，阶数由点数决定。

例 25.1. 对 $h(x) = \sqrt(x)$ 在 $x = 1/16, 1/4, 1$ 用抛物线插值。

```
ph <- poly.calc(x=c(1/16, 1/4, 1), y=sqrt(c(1/16, 1/4, 1)))
print(ph)
```

```
## 0.1555556 + 1.555556*x - 0.7111111*x^2
```

可以用 MASS 包的 `fractions()` 函数得到分数表示：

```
MASS::fractions(coef(ph))
```

```
## [1]   7/45   14/9 -32/45
```

即此插值多项式的理论值是

$$\frac{1}{45}\left(7 + 70x - 32x^2\right).$$

用 `polylist()` 可以生成若干个多项式的列表，可以用 `sum()` 函数求多项式的和，用 `prod()` 求多项式乘积，用 `deriv()` 对其中每个微分，用 `integral()` 对其中每个积分。

25.2 插值

R 函数 `approx(x, y)` 对给定的 x 和 y 坐标计算线性插值，输出格子点上的 x, y 坐标的列表。可以用 `n=` 指定输出的格子点的个数，可以用 `xout=` 指定需要计算的 x 坐标。

`approx.fun(x, y)` 计算插值函数，结果是类型为 R 函数的插值函数。

25.3 样条插值

R 函数 `spline(x, y)` 对给定的 x 和 y 坐标计算样条插值，输出格子点上的 x, y 坐标的列表。可以用 `n=` 指定输出的格子点的个数，可以用 `xout=` 指定需

要计算的 x 坐标。默认使用 fmm 样条，这时在两端用各自最边缘处的 4 格点分别拟合三次多项式。选 `method="natural"` 用自然样条。

`splinefun(x, y)` 计算样条插值函数，结果是类型为 R 函数的插值函数，得到的插值函数可以用来计算 x 处的插值函数值及导数值，在调用生成的插值函数时加选项 `deriv=` 可以指定导数阶数，最多 3 阶。

函数 `spline.smooth()` 可以计算样条曲线拟合。

在用 `lm()` 模型，可以用 `bs()` 函数计算给定的自变量 x 的若干阶的样条基函数值作为非线性项。

25.4 积分

R 函数 `integrate()` 用自适应方法计算一元定积分。但是，当被积函数在大部分区域为常数时，自适应算法的收敛判断可能会误判，得到完全错误的结果。

25.5 Gauss-Legendre 积分

R 的 gss 包的 `gauss.quad()` 函数可以对给定点数和区间端点计算高斯-勒让德积分的节点位置和相应积分权重。用法为

```
gauss.quad(size, interval)
```

25.6 微分

R 的 `deriv()` 函数计算表达式的导数,有符号显示表示时用符号计算得到结果，否则计算数值导数。如 `{rapprox-rlang-deriv01, cache=TRUE} deriv(~sin(x)^2, "x")`

Chapter 26

Julia 中的近似计算函数 (*)

待完成。

Part V

矩阵计算

Chapter 27

统计计算中的矩阵计算

27.1 矩阵计算的应用例子

统计模型和统计计算中广泛使用矩阵运算和线性方程组求解。例如，如下的线性模型

$$y = X\beta + \varepsilon \tag{27.1}$$

中，y 为 $n \times 1$ 向量, X 为 $n \times p$ 矩阵, 一般第一列元素全是 1, 代表截距项；β 为 $p \times 1$ 未知参数向量; ε 为 $n \times 1$ 随机误差向量, ε 的元素独立且方差为相等的 σ^2(未知)。参数 β 的最小二乘估计是如下正规方程的解：

$$X^T X\beta = X^T y, \tag{27.2}$$

当 X 为列满秩矩阵时 β 的最小二乘估计可以表示为

$$\widehat{\beta} = (X^T X)^{-1} X^T y, \tag{27.3}$$

因变量 y 的拟合值（预报值）可以表示为

$$\widehat{y} = X(X^T X)^{-1} X^T y = Hy,$$

其中 $H = X(X^T X)^{-1} X^T$ 是对称幂等矩阵。

再比如，在时间序列分析问题中需要对多元序列建模，常用的一种模型是向量自回归模型，p 阶向量自回归模型可以写成

$$x_t = \sum_{j=1}^{p} A_j x_{t-j} + \varepsilon_t,\ t \in \mathbb{Z}$$

其中 x_t 是 m 元随机向量，$A_1, A_2, \ldots, A_p$ 是 $m \times m$ 矩阵，ε_t 是 m 元白噪声。

可见，统计模型中广泛使用矩阵作为模型表达工具，统计计算中有大量的矩阵计算。我们需要研究稳定、高效的矩阵计算方法。例如，按照(27.3)计算 $\hat{\beta}$ 从理论上很简单，但需要计算逆矩阵，实际计算量比较大，而且当 X 的各列之间有近似线性关系时算法不稳定；如果把(27.2)看成是线性方程组来求解，或者直接考虑最小化 $\|y - X\beta\|^2$ 的问题，则可以找到各种快速且高精度的计算方法。

有许多编程语言有成熟的矩阵计算软件包，比如 FORTRAN 和 C 语言的 LAPACK 程序包 [E. and ten others, 1999]、IMSL[Inc., 1985] 程序包。在常用统计软件系统一般内建了高等矩阵计算功能，比如 R 软件 (见A)、Julia(见B)、Python 的 scipy 模块、SAS 软件中的 IML 模块、MATLAB 软件，等等。我们可能不需要再去自己编写矩阵乘法、解线性方程组这些基础的计算程序，但是还是要了解这里面涉及的算法，这样遇到高强度、高维数等复杂情形下的矩阵计算问题才能给出稳定、高效的解决方案。对于反复使用的矩阵运算，1% 的速度提升也是难能可贵的；对于高阶矩阵，应该尽可能少产生中间结果矩阵，尽可能把输入和输出保存在同一存储位置。

例 27.1. 设 A 为 $n \times n$ 矩阵，x 为 n 维列向量，计算矩阵乘法 Ax 需要 n^2 次乘法和 n^2 次加法。如果 A 有特殊结构 $A = I_n + uv^T$，其中 u 和 v 是 n 维列向量，则

$$Ax = x + (v^T x)u$$

只需要 $2n$ 次乘法和 $2n$ 次加法，并且矩阵 A 也不需要保存 n^2 个元素，而只需 u 和 v 的 $2n$ 个元素。若 A 是上三角矩阵或下三角矩阵，则 Ax 只需要 $\frac{1}{2}n(n+1)$ 次乘法和加法，计算量比一般的 A 减少一半。

在 R 语言中，用 `matrix` 函数定义一个矩阵，用 `cbind` 和 `rbind` 进行横向和纵向合并，用 `t(A)` 表示 A 的转置，用 `A %*% B` 表示矩阵 A 和 B 相乘。R 语言的矩阵是多维数组中的二维数组。

Julia 语言也支持多维数组，矩阵就是二维数组。

27.2 矩阵记号与特殊矩阵

在本书中我们用黑体小写字母表示向量，且缺省为列向量，如 a, v，用 a_i 表示 a 的第 i 元素。用 $\mathbb{R}^n$ 表示所有 n 维实值向量组成的 n 维欧式空间, 用 (a,b) 表示 a^Tb，称为 a,b 的内积，记 $\|a\| = (a,a)^{1/2}$，称为 a 的欧式模。用大写字母表示矩阵，如 A, M，用 a_{ij} 表示 A 的第 i 行第 j 列元素，用 $a_{\cdot j}$ 表示 A 的第 j 列组成的列向量，用 $a_{i\cdot}$ 表示 A 的第 i 行组成的行向量。用 A^T 表示 A 的转置, $\det(A)$ 表示 A 的行列式。另外，一些特殊的矩阵定义如下：

- 设 e_i 为 n 维列向量，如果其第 i 个元素为 1，其它元素为 0，称 e_i 为 n 维**单位向量**。
- 记 1 为元素都是 1 的列向量。
- 用 I_n 表示 n 阶单位阵，用 I 表示单位阵。
- 若矩阵 A 的元素满足 $a_{ij} = 0, \forall i < j$，称 A 为**下三角矩阵**。
- 若矩阵 A 的元素满足 $a_{ij} = 0, \forall i > j$，称 A 为**上三角矩阵**。
- 若矩阵 A 的元素满足 $a_{ij} = 0, \forall i \neq j$，称 A 为**对角矩阵**。
- 若矩阵 A 的元素满足 $a_{ij} = 0, \forall |i-j| > 1$，称 A 为**三对角矩阵**。
- 若实方阵 A 满足 $A^TA = I$，则称 A 为**正交阵**。
- 若 A 为 n 阶实对称矩阵，对任意 n 维非零实数向量 x 有 $x^TAx > 0$，称 A 为**正定阵**。
- 若 A 为 n 阶实数对称矩阵，对任意 n 维实数向量 x 有 $x^TAx \geq 0$，称 A 为**非负定阵**或**半正定阵**。
- 若 $P(i,j)$ 是把 I_n 的第 i 行和第 j 行交换位置后得到的 n 阶方阵，称 $P(i,j)$ 为**基本置换阵**。$P(i,j)A$ 把 A 的第 i、j 两行互换, $AP(i,j)$ 把 A 的第 i、j 两列互换。$P(i,j)P(i,j) = I_n$。
- 设 $\pi = (k_1, k_2, \ldots, k_n)^T$ 是由 $(1,2,\ldots,n)$ 的一个排列组成的 n 维向量，方阵 $P = (e_{k_1}, e_{k_2}, \ldots, e_{k_n})$, 称 P 为**置换阵**。P 是一个正交阵，对任意 $m \times n$ 矩阵 A, AP 是把 A 的各列按照 $k_1, k_2, \ldots, k_n$ 次序重新排列得到的矩阵。
- 设 A 为 $n \times m$ 矩阵 $(n \geq m)$，称 $\mu(A) = \{y \in \mathbb{R}^n : y = Ax, x \in \mathbb{R}^m\}$ 为由 A 的列向量张成的线性子空间。
- 设 n 阶实对称矩阵 P 满足 $P^2 = P$，称 P 为**对称幂等矩阵**，P 是 $\mathbb{R}^n$ 到 $\mu(P)$ 的 **(正交) 投影矩阵**，对任意 $x \in \mathbb{R}^n$, Px 与 $x - Px = (I-P)x$ 正交。若 A 为 $n \times m$ 列满秩矩阵, 则 $P = A(A^TA)^{-1}A^T$ 是 $\mathbb{R}^n$ 到 $\mu(A)$

的正交投影矩阵。

例 27.2. 考虑如下置换：

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix} \tag{27.4}$$

表示将 1 换成 4，2 换成 3，3 换成 1，4 换成 2。

对应的置换阵，用 Julia 表示:

```
P = eye(4)[:, [4,3,1,2]]
P * [1;2;3;4]

## 4-element Array{Float64,1}:
##  3.0
##  4.0
##  2.0
##  1.0
```

用 P 左乘一个列向量，将第 1 行换到了第 4 行，第 2 行换到了第 3 行，第 3 行换到了第 1 行，第 4 行换到了第 2 行。

以上的置换变换的逆变换是将(27.4)的两行调换，然后按照第一行排序：

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix} \tag{27.5}$$

```
Prev = eye(4)[:, [3,4,2,1]]
Prev * [3;4;2;1]

## 4-element Array{Float64,1}:
##  1.0
##  2.0
##  3.0
##  4.0
```

实际上，P 是正交阵，所以 $P^{-1} = P^T$：

```
maximum(Prev - P')
## 0.0
```

27.3 矩阵微分

对 $f : \mathbb{R}^p \to \mathbb{R}$, 记 $\nabla f(x)$ 为 f 的 p 个一阶偏导数组成的列向量，称为 f 的梯度。记 $\nabla^2 f(x)$ 为 f 的二阶偏导数组成的 $p \times p$ 矩阵，称为 f 的海色阵 (Hessian)。$\nabla^2 f(x)$ 的第 i 行第 j 列元素为 $\frac{\partial^2 f(x)}{\partial x_i \partial x_j}$。

设 a 为 p 维列向量，A 为 $p \times p$ 对称阵，则

$$\nabla(a^T x) = a, \quad \nabla(x^T a) = a,$$
$$\nabla(x^T A x) = 2Ax, \quad \nabla^2(x^T A x) = 2A.$$

如果 A 是 $p \times p$ 矩阵，则

$$\nabla(x^T A x) = (A + A^T)x, \quad \nabla^2(x^T A x) = A + A^T$$

对 $f : \mathbb{R}^p \to \mathbb{R}^q$, 设 $y = f(x)$, 记 $q \times p$ 个一阶偏导数 $\frac{\partial y_i}{\partial x_j}$ 组成的 $q \times p$ 矩阵为 $\frac{\partial f(x)}{\partial x}$，称为 f 的 Jacobi 矩阵，当 $q = 1$ 时，Jacobi 矩阵是行向量，是梯度向量（列向量）的转置。

若 $y = f(x) = Ax$, 其中 A 是 $q \times p$ 矩阵，则 $\frac{\partial f(x)}{\partial x} = A$。

对 $f : \mathbb{R}^{n \times m} \to \mathbb{R}$，设 $y = f(A)$, 其中 A 是 $n \times m$ 矩阵，定义 $\frac{\partial f(A)}{\partial A}$ 为矩阵 $\left(\frac{\partial y}{\partial a_{ij}}\right)_{n \times m}$。若 $f(A) = a^T A b$，其中 a 和 b 是列向量，则 $\frac{\partial f(A)}{\partial A} = ab^T$。

27.4 矩阵期望

这里列出概率论中关于随机向量和随机矩阵的几个基本公式。设 X 为 n 元随机向量，Y 为 m 随机向量，M 为 $m \times n$ 随机矩阵，A, B, C 为普通非随机的实数矩阵。a, b 为非随机的向量。EX 是 X 的各个分量的期望组成的列向量，$\mathrm{Var}(X)$ 表示 X 的协方差阵，其 (i, j) 元素为 X_i 和 X_j 的协方差，当 $i = j$ 时

为 X_i 的方差。$\mathrm{Cov}(X,Y)$ 是一个 $n\times m$ 矩阵，其 (i,j) 元素为 $\mathrm{Cov}(X_i,Y_j)$。有如下常用公式:

$$\begin{aligned}
E(AMB+C) =& AE(M)B+C\\
\mathrm{Var}(X) =& E\left[(X-EX)(X-EX)^T\right]\\
\mathrm{Var}(AX+a) =& A\mathrm{Var}(X)A^T\\
\mathrm{Cov}(X,Y) =& E\left[(X-EX)(Y-EY)^T\right]\\
\mathrm{Cov}(AX+a,BY+b) =& A\mathrm{Cov}(X,Y)B^T.
\end{aligned}$$

习题

习题 1

设 A 为 $n\times m$ 矩阵，x 为 m 维向量，为了计算 Ax，可以逐个计算结果的 n 个元素，也可以把这个乘法看成是对 A 的各列用 x 作为线性组合系数作线性组合，逐次把 A 的第 j 列与 x_j 相乘累加到结果向量中。写出这两种算法，尽可能减少不必要的存储。计算这两种算法所需的乘法和加法的次数。如果使用 C 语言或 FORTRAN 语言来实现这两种算法，在 n,m 很大时，两种算法会有不同的计算效率，设法验证。

Chapter 28

线性方程组求解与 LU 分解

统计计算和其它科学工程计算中很多的问题会转化为线性方程组求解问题。本节讨论稳定、高效的线性方程组求解方法。

28.1 三角形线性方程组求解

我们熟知的解线性方程组的一种方法是消元法，用线性变化把增广矩阵化为阶梯形然后用回代法求解。我们先给出系数矩阵为三角形矩阵的线性方程组解法。

设有如下的三角形 n 阶线性方程组

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1,n-1}x_{n-1} + a_{1n}x_n &= b_1 \\ a_{22}x_2 + \cdots + a_{2,n-1}x_{n-1} + a_{2n}x_n &= b_2 \\ \ddots \qquad\qquad & \vdots \\ a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n &= b_{n-1} \\ a_{nn}x_n &= b_n \end{aligned}$$

设其系数矩阵满秩（当且仅当 $a_{11}a_{22}\dots a_{nn}\neq 0$），则求解过程可以写成：

$$\begin{aligned}
x_n =& b_n/a_{nn}\\
x_{n-1} =& \left(b_{n-1}-a_{n-1,n}x_n\right)/a_{n-1,n-1}\\
&\vdots\\
x_2 =& \left(b_2-a_{23}x_3-\cdots-a_{2n}x_n\right)/a_{22}\\
x_1 =& \left(b_1-a_{12}x_2-\cdots-a_{1,n}x_n\right)/a_{11}
\end{aligned}$$

这种求解过程叫做**回代法**。需要的话可以把解出的 x 元素保存在 b 原来的存储空间中。

如果系数矩阵是下三角的，也可以类似求解。

28.2 高斯消元法

高斯消元法是众所周知的线性方程组解法，配合主元元素选取可以得到稳定的解。

例 28.1. 考虑线性方程组

$$\left\{\begin{array}{rcrcrcrcr}
3x_1 & + & x_2 & + & 2x_3 & + & x_4 & = & 5\\
6x_1 & + & 4x_2 & + & 7x_3 & + & 11x_4 & = & 5\\
15x_1 & + & 11x_2 & + & 18x_3 & + & 34x_4 & = & 6\\
18x_1 & + & 16x_2 & + & 25x_3 & + & 56x_4 & = & -4
\end{array}\right. \tag{28.1}$$

只要把方程组变成上三角形就可以用 §28.1描述的方法求解。记

$$A=\begin{pmatrix}3&1&2&1\\6&4&7&11\\15&11&18&34\\18&16&25&56\end{pmatrix}\quad b=\begin{pmatrix}5\\5\\6\\-4\end{pmatrix}\quad \overline{A}=(A\,|\,b)\quad x=\begin{pmatrix}x_1\\x_2\\x_3\\x_4\end{pmatrix}$$

令 $m^{(1)}=(0,2,5,6)^T$，把第一个方程乘以 $-m_2^{(1)}=-2$ 加到第二个方程上，可以消去第二个方程中 x_1 的系数。类似地，把第一个方程乘以 $-m_3^{(1)}=-5$ 加到第三个方程上，把第一个方程乘以 $-m_4^{(1)}=-6$ 加到第四个方程上，可以消

去第三、第四方程中 x_1 的系数。方程组变成

$$\begin{cases} 3x_1 + x_2 + 2x_3 + x_4 = 5 \\ 2x_2 + 3x_3 + 9x_4 = -5 \\ 6x_2 + 8x_3 + 29x_4 = -19 \\ 10x_2 + 13x_3 + 50x_4 = -34 \end{cases} \tag{28.2}$$

记 $m^{(2)} = (0, 0, 3, 5)^T$，把第二个方程乘以 $-m_3^{(2)} = -3$ 加到第三个方程上，可以消去第三个方程中 x_2 的系数；把第二个方程乘以 $-m_4^{(2)} = -5$ 加到第四个方程上，可以消去第四个方程中 x_2 的系数。方程组变成

$$\begin{cases} 3x_1 + x_2 + 2x_3 + x_4 = 5 \\ 2x_2 + 3x_3 + 9x_4 = -5 \\ -x_3 + 2x_4 = -4 \\ -2x_3 + 5x_4 = -9 \end{cases} \tag{28.3}$$

记 $m^{(3)} = (0, 0, 0, 2)^T$, 把第三个方程乘以 $-m_4^{(3)} = -2$ 加到第四个方程上，可以消去第四个方程中 x_3 的系数。方程组变成

$$\begin{cases} 3x_1 + x_2 + 2x_3 + x_4 = 5 \\ 2x_2 + 3x_3 + 9x_4 = -5 \\ - x_3 + 2x_4 = -4 \\ x_4 = -1 \end{cases} \tag{28.4}$$

用回代法可以解出 $x = (1, -1, 2, -1)^T$。

28.3 LU 分解

考虑例28.1把系数矩阵变成上三角形的过程。第 j 步用初等变换把系数矩阵第 j 列的主对角线下方的元素都变成零。设 $A^{(0)} = A$，A 为 $n \times n$ 系数矩阵，第 j 步把矩阵 $A^{(j-1)}$ 变成矩阵 $A^{(j)}$，实际是左乘了一个初等变换矩阵 $M^{(j)}$:

$$A^{(j)} = M^{(j)} A^{(j-1)},\ j = 1, 2, \ldots, n-1$$

其中 $M^{(j)}$ 是一个与 I_n 仅在第 j 列不同的方阵，其第 j 列为

$$m_{ij}^{(j)} = \begin{cases} 0, & \text{当} i < j \\ 1, & \text{当} i = j \\ -a_{ij}^{(j-1)} / a_{jj}^{(j-1)}, & \text{当} i > j \end{cases}$$

定义 n 维向量 $m^{(j)}$ 为

$$m_i^{(j)} = \begin{cases} 0, & \text{当} i \le j \\ a_{ij}^{(j-1)}/a_{jj}^{(j-1)}, & \text{当} i > j \end{cases} \tag{28.5}$$

则初等变换矩阵 $M^{(j)}$ 可以表示为

$$M^{(j)} = I_n - m^{(j)} e_j^T \tag{28.6}$$

经过 $n-1$ 步消去变换后，得到的上三角形系数矩阵为

$$A^{(n-1)} = M^{(n-1)} \dots M^{(2)} M^{(1)} A$$

如果每一步中的分母 $a_{jj}^{(j-1)}$ 都不等于零 (注意：浮点运算中不等于零的判断是不可靠的)，则上述步骤可以把方程组的系数矩阵变成上三角形, 对线性方程组右边的 b 也作相同的变换就得到三角形线性方程组，可以用回代法求解。

但是，$a_{jj}^{(j-1)} = 0$ 的情况是存在的，例如

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

就无法用例28.1的方法化为上三角形。

这种情况怎么办？当第 j 步的 $a_{jj}^{(j-1)} = 0$ 时，我们可以看第 $j+1, j+2, \dots, n$ 个方程中 x_j 的系数是否非零，如果第 s_j 个方程的 x_j 系数不等于零，可以把第 j 个方程与第 s_j 个方程互换，然后继续消元。把第 j 个方程与第 s_j 个方程互换的操作相当于对系数矩阵左乘简单置换阵 $P(j, s_j)$。

那么，是不是第 $j+1, j+2, \dots, n$ 个方程中只要 x_j 的系数非零就可以与第 j 个方程互换？要注意的是，第 j 个方程与第 s_j 个方程互换后原来的 $a_{s_j j}^{(j-1)}$ 就变成了第 j 列的第 j 行元素，要作为消去的分母 (称为**主元**)；而数值计算中 “非零” 的判断是很不可靠的，应该等于零的值在数值计算中很可能因为舍入误差变成非零。所以，在轮到对第 j 列消元时，不论 $a_{jj}^{(j-1)}$ 是不是非零值，都要从系数矩阵第 j 列的第 $j, j+1, \dots, n$ 行元素中找到绝对值最大的一个，假设此元素在第 j 列的第 s_j 行，就把第 s_j 行与第 j 行互换，然后进行消去。这种解方程的方法叫做**列主元的高斯消元法**。

得到三角形矩阵及相应方程组的过程为:

$$\begin{aligned} A^{(n-1)} =& M^{(n-1)} P(n-1, s_{n-1}) \dots M^{(2)} P(2, s_2) M^{(1)} P(1, s_1) A \\ A^{(n-1)} x =& M^{(n-1)} P(n-1, s_{n-1}) \dots M^{(2)} P(2, s_2) M^{(1)} P(1, s_1) b \end{aligned}$$

注意，其中的 $M^{(j)}$ 是基于行置换后的 $P(j,s_j)A^{(j-1)}$ 计算的。实际求解时，通行的算法是把每一步的 $A^{(j)}$ 保存在 A 的存储空间，但已经消去变成零的元素不保存，代之以将 $m^{(j)}$ 的第 $j+1,j+2,\ldots,n$ 号元素存放在 $A^{(j)}$ 第 j 列的最后 $n-j$ 个元素中 (这 $n-j$ 本来被消去变成了零), 解 x 的元素可以存放在 b 的存储空间中。要注意的是，在第 $j+1,j+2,\ldots$ 步时的行置换会打乱这样保存的 $m^{(j)}$ 的元素次序。

可以证明，按照以上所述的步骤，把原始矩阵 A 做了如下 LU 分解:

$$PA = LU \tag{28.7}$$

其中 P 是一个置换矩阵:

$$P = P(n-1,s_{n-1})\ldots P(2,s_2)P(1,s_1),$$

只要保存向量 $(s_1,s_2,\ldots,s_{n-1})$ 就可以恢复矩阵 P。U 是一个上三角矩阵，其上三角元素保存在原来输入的 A 存储空间的上三角部分，为列主元法消元最后得到的 $A^{(n-1)}$ 矩阵。L 是一个单位下三角矩阵（主对角线元素都等于 1 的下三角矩阵），它严格下三角部分（不包括主对角线在内的下三角部分）的元素保存在原来输入的 A 的存储空间的严格下三角部分，第 j 列的最后的 $n-j$ 个元素保存了被第 $j+1,j+2,\ldots,n-1$ 次行置换打乱次序的 $m^{(j)}$, 记为 $m_*^{(j)}$:

$$m_*^{(j)} = P(n-1,s_{n-1})\ldots P(j+1,s_{j+1})m^{(j)}, \tag{28.8}$$

所以 L 可以表示为

$$L = I_n + m_*^{(1)}e_1^T + \cdots + m_*^{(n-1)}e_{n-1}^T.$$

公式(28.7)称为矩阵 A 的三角分解或 LU 分解。

在得到 LU 分解(28.7)后，对任何一个常数向量 b，要求解 $Ax=b$，可以化为 $PAx=Pb$，注意左乘矩阵 P 只是进行了 $n-1$ 次行置换。由 $PA=LU$ 得 $L\,U\,x=(Pb)$, 令 $y=Ux$，先用回代法解下三角形方程组 $Ly=Pb$，再用回代法解上三角形方程组 $Ux=y$ 就可以得到方程组的解。

求解线性方程组 $Ax=b$ 时，为什么不先求逆矩阵 A^{-1} 再求 $x=A^{-1}b$? 仔细分析上述算法的运算次数可以发现，得到 LU 分解(28.7)只需 $\frac{2}{3}n^3+O(n^2)$ 次浮点运算，额外地再用两遍回代法对一个 b 求解 x 仅需 $2n^2$ 次浮点运算，但是在得到(28.7)后如果要求 A^{-1} 则需要额外的 $\frac{2}{3}n^3$ 次浮点运算，即求 A^{-1} 需要

$\frac{4}{3}n^3 + O(n^2)$ 次浮点运算。直接用消元法求 A^{-1} 也需要 $\frac{4}{3}n^3 + O(n^2)$ 次浮点运算。得到 A^{-1} 后计算 $x = A^{-1}b$ 需要 $2n^2$ 次浮点运算，与两遍回代法求 x 计算量相同, 但多出了求逆的 $\frac{2}{3}n^3$ 次浮点运算。所以如果仅需要解线性方程组的话不需要计算逆矩阵。

得到 LU 分解(28.7)后可以给出 A 的行列式：

$$\det(P)\det(A) = \det(PA) = \det(LU) = \det(U)$$

$\det(U)$ 等于 U 的主对角线元素乘积，$\det(P)$ 等于 1 或 -1，当 P 代表偶数次行置换时 $\det(P) = 1$，否则 $\det(P) = -1$。注意 P 是 $n-1$ 个行置换的乘积，当 $s_j = j$ 时 $P(j, s_j) = I_n$，$\det(P(j, s_j)) = 1$；当 $s_j \neq j$ 时 $\det(P(j, s_j)) = -1$。所以，如果向量 $(s_1, s_2, \dots, s_{n-1})$ 与向量 $(1, 2, \dots, n-1)$ 不相等的元素个数是偶数，则 $\det(P) = 1$，否则 $\det(P) = -1$。

在 R 语言中，用 `solve(A, b)` 求解 $Ax = b$, 用 `solve(A)` 求 A 的逆矩阵，用 Matrix 包的 `lu` 函数求 LU 分解。

在 Julia 语言中用 `A \ b` 求解 $Ax = b$。

28.4 Cholesky 分解

统计计算中比一般矩阵更常用到的是非负定矩阵和正定矩阵。比如，随机向量 X 的协方差阵 $\Sigma = \mathrm{Var}(X)$ 是非负定的，如果满秩，就是正定的。回归分析中正规方程(27.2)中的叉积阵 X^TX 是非负定的，如果 X 列满秩则 X^TX 是正定的。如果方程 $Ax = b$ 的系数矩阵 A 是正定矩阵，虽然还可以用高斯消元法或 LU 分解来求解，但这样不能利用 A 的特殊结构。正定矩阵 A 可以进行所谓 Cholesky 分解:

$$A = LL^T \tag{28.9}$$

其中 L 是一个主对角线元素都取正值的下三角阵。

先承认(28.9)的存在性, 设法求 L。设 L 的元素为 l_{ij}, $i \geq j$, 则

$$A = \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ & l_{22} & \cdots & l_{n2} \\ & & \ddots & \vdots \\ & & & l_{nn} \end{pmatrix}$$

显然 $a_{11} = l_{11}^2$。用 $A^{[k]}$ 表示 A 的前 k 行和前 k 列组成的子矩阵，$L^{[k]}$ 表示 L 的前 k 行和前 k 列组成的子矩阵，易见 $A^{[k]} = L^{[k]}(L^{[k]})^T$。归纳地，设 $L^{[k-1]}$ 已经求得，要求 L 的第 k 行, 记 $(l^{[k]})^T = (l_{k1}, l_{k2}, \dots, l_{k,k-1})$, $a^{[k]} = (a_{1,k}, a_{2,k}, \dots, a_{k-1,k})^T$, 由 $A^{[k]} = L^{[k]}(L^{[k]})^T$ 有

$$\begin{aligned} A^{[k]} &= \begin{pmatrix} A^{[k-1]} & a^{[k]} \\ (a^{[k]})^T & a_{kk} \end{pmatrix} \\ &= L^{[k]}(L^{[k]})^T = \begin{pmatrix} L^{[k-1]} & 0 \\ (l^{[k]})^T & l_{kk} \end{pmatrix} \begin{pmatrix} (L^{[k-1]})^T & l^{[k]} \\ 0 & l_{kk} \end{pmatrix} \end{aligned}$$

得方程

$$L^{[k-1]} l^{[k]} = a^{[k]} \tag{28.10}$$

$$l_{kk}^2 = a_{kk} - (l^{[k]})^T l^{[k]} \tag{28.11}$$

只要用回代法解下三角方程组(28.10)得到 $l^{[k]}$, 然后开平方根得到 $l_{kk} = \left(a_{kk} - (l^{[k]})^T l^{[k]}\right)^{1/2}$。

例 28.2. 矩阵

$$A = \begin{pmatrix} 4 & 2 & 0 & 2 \\ 2 & 10 & 12 & 1 \\ 0 & 12 & 17 & 2 \\ 2 & 1 & 2 & 9 \end{pmatrix}$$

是正定阵，来求它的 Cholesky 分解。

- $k = 1$: $l_{11} = \sqrt{a_{11}} = \sqrt{4} = 2$.
- $k = 2$: 方程 $L^{[1]} l^{[2]} = a^{[2]}$ 即 $l_{11} l_{21} = a_{21}$(注意 $a_{12} = a_{21}$)，所以 $2l_{21} = 2$, $l_{21} = 1$；于是用(28.11)得 $l_{22} = \sqrt{a_{22} - l_{21}^2} = \sqrt{10 - 1^2} = 3$;
- $k = 3$: 方程 $L^{[2]} l^{[3]} = a^{[3]}$ 即

$$\begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} l_{31} \\ l_{32} \end{pmatrix} = \begin{pmatrix} 0 \\ 12 \end{pmatrix}$$

解得 $l_{31} = 0$, $l_{32} = 4$, 再求出 $l_{33} = \sqrt{a_{33} - (l_{31}^2 + l_{32}^2)} = \sqrt{17 - (0^2 + 4^2)} = 1$。

- $k=4$: 方程 $L^{[3]}l^{[4]}=a^{[4]}$ 即

$$\begin{pmatrix} 2 & 0 & 0 \\ 1 & 3 & 0 \\ 0 & 4 & 1 \end{pmatrix}\begin{pmatrix} l_{41} \\ l_{42} \\ l_{43} \end{pmatrix}=\begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix}$$

解得 $l_{41}=1, l_{42}=0, l_{43}=2$, 再开平方根得到 $l_{44}=\sqrt{a_{44}-(l_{41}^2+l_{42}^2+l_{43}^2)}=2$。

于是，

$$L=\begin{pmatrix} 2 & & & \\ 1 & 3 & & \\ 0 & 4 & 1 & \\ 1 & 0 & 2 & 2 \end{pmatrix},\quad A=LL^T.$$

※※※※※

当 A 为正定阵时，各 $A^{[k]}$ 也是正定阵，取向量

$$\alpha=\begin{pmatrix} -(A^{[k-1]})^{-1}a^{[k]} \\ 1 \\ 0 \end{pmatrix} \tag{28.12}$$

则(28.11)的右边是二次型 $\alpha^TA\alpha$，应该为正值，所以解方程(28.10)–(28.11)逐次得到的 $a_{kk}>0$，这样 $L^{[k]}$ 是满秩的，于是下一步的方程(28.10)有唯一解，先计算 $l_{11}=\sqrt{a_{11}}$ 然后对 $k=2,3,\ldots,n$ 重复解(28.10)–(28.11) 一定每步都可以进行且解是唯一的。于是，正定阵 A 的 Cholesky 分解是存在唯一的。

Cholesky 分解需要 $\frac{1}{3}n^3+O(n^2)$ 次浮点运算和 n 次开平方根，比 LU 分解的 $\frac{2}{3}n^3$ 次浮点运算次数少，开平方根所需的时间只是 n 的倍数。

编写 Cholesky 算法的程序时，如果输入了一个正定阵 A, 因为正定阵是对称的，可以只输入 A 的下三角部分，返回的 Cholesky 分解 L^TL 的下三角矩阵 L 可以保存在输入的矩阵 A 的下三角部分的存储空间中。对于对称矩阵和下三角矩阵也可以只保存其下三角部分的元素，这样按行排列的话，第 (i,j) 元素存放在第 $\frac{1}{2}i(i-1)+j$ 号位置。

统计中经常需要计算正定矩阵逆矩阵的二次型 $\alpha^TA^{-1}\alpha$, 就可以用 Cholesky 分解转化为

$$\alpha^TA^{-1}\alpha=\alpha^T(LL^T)^{-1}\alpha=(L^{-1}\alpha)^TL^{-1}\alpha,$$

只要用回代法解 $Lx=\alpha$, 则有 $\alpha^TA^{-1}\alpha=x^Tx$。这样计算只需要 Cholesky 分解的 $\frac{1}{3}n^3+O(n^2)$ 次运算，如果直接计算逆矩阵，则需要 $\frac{4}{3}n^3+O(n^2)$ 次运算。

在 R 软件中，用 `chol()` 函数求正定阵的 Cholesky 分解。

设 A 的 Cholesky 分解 $A=LL^T$ 中 L 的主对角线元素组成的对角阵为 $D=\mathrm{diag}(l_{11},l_{22},\dots,l_{nn})$, 令 $\tilde{L}=LD^{-1}$, $\tilde{D}=D^2$, 则 $\tilde{L}$ 为单位下三角阵，$\tilde{D}$ 是主对角线元素为正值的对角阵，A 有如下分解:

$$A=LL^T=\tilde{L}DD\tilde{L}^T=\tilde{L}\tilde{D}\tilde{L}^T, \tag{28.13}$$

当 A 为对称正定阵时此分解存在唯一，称为矩阵 A 的 LDL 分解。

28.5 线性方程组求解的稳定性 (*)

设 A 为 n 阶方阵，则线性方程组 $Ax=b$ 存在唯一解的充分必要条件是 A 满秩，即 A 的列向量的任何一个非零线性组合都不等于零向量。但是，如果存在 A 的列向量的一个非零线性组合与零向量十分接近会怎样？这时，方程求解的误差很大而且不稳定（计算误差和舍入误差的影响很大）。

例 28.3. 方程组

$$\begin{pmatrix} 3 & 1 \\ 3.0001 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}=\begin{pmatrix} 4 \\ 4.0001 \end{pmatrix}$$

有精确解 $(1,1)^T$。考虑扰动对其的影响。

对 A,b 作微小变化：

$$\begin{pmatrix} 3 & 1 \\ 2.9999 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}=\begin{pmatrix} 4 \\ 4.0002 \end{pmatrix}$$

则精确解变为 $(-2,10)$。这里, $\det(A)=-0.0001$, A 已经很接近于不满秩。

为了考虑线性方程组求解的适定性，首先回顾一些关于向量和矩阵运算的内容。设 $x\in\mathbb{R}^n$, 定义

$$\|x\|_p=\left(\sum_{i=1}^n|x_i|^p\right)^{1/p},$$

这叫做向量 x 的 p 范数。特别地，$p = 1, 2, +\infty$ 时有

$$\begin{aligned}\|x\|_1 &= \sum_{i=1}^{n} |x_i|,\\ \|x\|_2 &= \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{x^T x},\\ \|x\|_\infty &= \max_{1\le i\le n} |x_i|.\end{aligned}$$

向量范数作为向量空间中广义的长度，满足:

- $\|x\| \ge 0$, 等号成立当且仅当 $x = 0$;
- 对任意 $\lambda \in \mathbb{R}$ 有 $\|\lambda x\| = |\lambda| \cdot \|x\|$;
- 有三角不等式 $\|x + y\| \le \|x\| + \|y\|$。

给定矩阵 A, $f(x) = Ax$ 是一个多元线性函数，或称为一个线性算子。为了考察 x 的变化引起的 Ax 的变化大小，引入矩阵范数 $\|A\|$。定义

$$\|A\|_p = \sup_{\|x\|_p=1} \|Ax\|,$$

称为 A 的 p 范数。可以证明，

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}|, \tag{28.14}$$

$$\|A\|_2 = \sqrt{A^T A\text{的最大特征值}}, \tag{28.15}$$

$$\|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|. \tag{28.16}$$

定义矩阵 p 范数后有

$$\|Ax\|_p \le \|A\|_p \cdot \|x\|_p, \ \forall x \in \mathbb{R}^n,$$

进一步

$$\|ABx\|_p \le \|A\|_p \cdot \|Bx\|_p \le \|A\|_p \cdot \|B\|_p \cdot \|x\|_p.$$

所以 $\|AB\|_p \le \|A\|_p \cdot \|B\|_p$。

一般的矩阵范数定义如下。

定义 28.1 (矩阵范数)**.** 若对任意 n 阶实方阵 A，都有一个实数 $\|A\|$ 与之对应，且满足

(1) 非负性: $\|A\| \geq 0$, 且等号成立当且仅当 A 为零矩阵;
(2) 齐次性: 对任意 $\lambda \in \mathbb{R}$, 有 $\|\lambda A\| = |\lambda| \, \|A\|$;
(3) 三角不等式: 对任意 n 阶实方阵 A 和 B，都有 $\|A + B\| \leq \|A\| + \|B\|$;
(4) 相容性: 对任意 n 阶实方阵 A, B, 都有 $\|AB\| \leq \|A\| \, \|B\|$,

则称 $\|A\|$ 为 A 的矩阵范数。

定义

$$\|A\|_F = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2 \right)^{1/2},$$

则 $\|A\|_F$ 是 A 的一个矩阵范数，叫做 Frobenius 范数。$\|A\|_F$ 实际是将矩阵 A 按列拉直成一个列向量计算其欧式模, 所以算法中为了判断两个矩阵 A, B 是否近似相当，只要看 $\|A - B\|_F$ 是否小于给定的误差界限 ϵ。

定义了矩阵范数就可以用来评估解线性方程组 $Ax = b$ 的适定性，即输入的 b 的误差导致的解 x 的误差大小。理论上，当 A 满秩时 $x = A^{-1}b$，设 b 有一个输入误差 Δb，令

$$Ax = b, \quad A(x + \Delta x) = b + \Delta b,$$

则 b 输入的误差 Δb 造成的解的误差 Δx 满足 (见 关治 and 陆金甫 [1998] §5.4.2)

$$\frac{\|\Delta x\|_p}{\|x\|_p} \leq \kappa_p(A) \cdot \frac{\|\Delta b\|_p}{\|b\|_p}, \tag{28.17}$$

其中 $\kappa_p(A) = \|A\|_p \cdot \|A^{-1}\|_p$ 叫做 A 的**条件数**。条件数用来衡量以 A 为系数矩阵的线性方程组求解的稳定性，A 的条件数越大，求逆和解线性方程组越不稳定, 称系数矩阵条件数很大的线性方程组是**病态**的。对 $p = 2$，$\kappa_2(A)$ 是 $A^T A$ 的最大特征值与最小特征值的比值的算数平方根，当 A 是正定阵时 $\kappa_2(A)$ 是 A 的最大特征值与最小特征值的比值的平方根。

对于病态的线性方程组，某些变换可以解决一些明显的问题，比如，每一个方程都乘以一个调整因子使得各行元素大小相近，A 的每列都乘以一个调整因子使得各列元素大小相近（最后对应的未知数要作反向的调整），但是没有完全通用的方法解决病态问题。

求解线性方程组的误差还涉及到系数矩阵 A 的误差，更详细的讨论见 [关治 and 陆金甫, 1998] §5.4.2。

习题

习题 1

对满秩 n 阶上三角矩阵 A，编写算法求解线性方程组 $Ax = b$, 要求把 x 的解保存在 b 原来的存储空间中。计算算法需要的乘除法和加减法次数。

习题 2

对 n 阶单位下三角矩阵 A，编写算法求解线性方程组 $Ax = b$, 要求把 x 的解保存在 b 原来的存储空间中。计算算法需要的乘除法和加减法次数。

习题 3

设 A 为上三角矩阵，$e_k = (0, \dots, 0, 1, 0, \dots, 0)^T$ 为单位向量，写算法求解 $Ax = e_k$。解 x 中那些元素是一定等于零的？如何简化求解过程？编写算法求 A^{-1} 并把 A^{-1} 保存在 A 原来的存储空间中。

习题 4

证明用列主元法进行矩阵 LU 分解(28.7)的算法正确。

(1) 对 $M^{(i)} = I_n - m^{(i)} e_i^T$ 矩阵，证明当 $k, j > i$ 时有

$$P(k,j)M^{(i)} = [I_n - P(k,j)m^{(i)}e_i^T]P(k,j).$$

(2) 令 $M_*^{(k)} = I_n - m_*^{(k)} e_k^T$ ($m_*^{(k)}$ 定义见(28.8))，证明

$$\begin{aligned} &M^{(n-1)}P(n-1,s_{n-1}) \dots M^{(2)}P(2,s_2)M^{(1)}P(1,s_1) \\ =&M_*^{(n-1)} \dots M_*^{(1)}P(n-1,s_{n-1}) \dots P(1,s_1). \end{aligned}$$

(3) 证明

$$(M_*^{(k+1)}M_*^{(k)})^{-1} = I_n + m_*^{(k)}e_k^T + m_*^{(k+1)}e_{k+1}^T.$$

(4) 证明 $PA = LU$ 成立。

习题 4

编写用列主元法作高斯消元同时得到矩阵 LU 分解的算法程序。

习题 5

编写 Cholesky 分解算法的程序。对输入的正定阵 A（只有下三角部分需要输入值）, 求 Cholesky 分解 $A = LL^T$ 并把矩阵 L 存放在 A 的下三角部分中作为函数返回值。(如果使用 R 语言的编写一个输入 A 输出 L 的函数，由于 R 语言的设计特点，这样修改的是矩阵 A 的一个副本)。

习题 6

编写 Cholesky 分解算法的程序。设正定阵 A 只保存了下三角部分，元素按行次序保存在一个一维数组中输入，Cholesky 分解 $A = LL^T$ 得到的下三角矩阵 L 保存到 A 所用的存储空间中返回。

习题 7

对正定阵 A，证明(28.12)定义的向量与 A 组成的二次型 $\alpha^T A\alpha$ 等于(28.11)的右边。

习题 8

设 A 为正定阵，给出计算 $x^T A^{-1} y$ 的有效算法。

习题 9

考虑线性回归模型 $y = X\beta + \varepsilon$，其中 X 为 $n \times p$ 矩阵 $(n > p)$, 设 X 列满秩，则回归系数 β 的最小二乘估计为 $\widehat{\beta} = (X^T X)^{-1} X^T y$, 残差平方和 SSE 为 $y^T y - y^T X(X^T X)^{-1} X^T y$。写出利用 Cholesky 分解方法计算 $\widehat{\beta}$, SSE 和 $(X^T X)^{-1}$ 的算法。

习题 10

证明矩阵范数的三个公式(28.14)–(28.16)。

Chapter 29

特殊线性方程组求解 (*)

对于特殊的系数矩阵，比如稀疏的、带状的、巨大的系数矩阵，需要利用其特殊结构以提高运算效率或减轻对存储空间要求。

29.1 带状矩阵

如果矩阵 A 的元素 a_{ij} 满足 $a_{ij} = 0$ 对 $i > j + p$ 和 $j > i + q$，则称 A 为**带状矩阵**，称 p 为下带宽，q 为上带宽。储存带状矩阵时可以排除零元素，仅保存其它元素，这样每行至多有 $p + q + 1$ 个元素，整个矩阵只保存 $n(p + q + 1)$ 个元素。带状矩阵如果保存在一个 $n \times (p + q + 1)$ 矩阵中，其 a_{ij} 元素在新的存储矩阵的 $(i, j - (i - p - 1))$ 位置。

线性方程组 $Ax = b$ 的系数矩阵如果是带状矩阵，用列主元的高斯消元法求解会在交换行时失去带状结构。如果 A 有 LU 分解 $A = LU$，易见下三角矩阵 L 是一个下带宽 p 的带状单位下三角矩阵，上三角矩阵 U 是上带宽 q 的带状上三角矩阵。

$p = q = 1$ 的带状矩阵叫做三对角矩阵，只需要把主对角线、下副对角线、上副对角线分别保存在三个 n 维向量中。三对角矩阵如果有 LU 分解，必为如下形

式:

$$M = \begin{pmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_n & b_n \end{pmatrix} \tag{29.1}$$

$$= \begin{pmatrix} 1 & & & & \\ l_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & l_{n-1} & 1 & \\ & & & l_n & 1 \end{pmatrix} \begin{pmatrix} d_1 & c_1 & & & \\ & d_2 & c_2 & & \\ & & \ddots & \ddots & \\ & & & d_{n-1} & c_{n-1} \\ & & & & d_n \end{pmatrix} \tag{29.2}$$

所以只要求解 $l_2, \ldots, l_n$ 和 $d_1, \ldots, d_n$，输入 a, b, c 三个向量后，求解得到 $l_2, \ldots, l_n$ 可以放在 $a_2, \ldots, a_n$ 的存储空间中，$d_1, \ldots, d_n$ 放在 $b_1, \ldots, b_n$ 的存储空间中，有如下的递推算法:

三对角矩阵递推算法

输入 a, b, c
for(i **in** $2 : n$) {
    $a_i \leftarrow a_i / b_{i-1}$
    $b_i \leftarrow b_i - a_i c_{i-1}$
}
输出 $(a_2, \ldots, a_n)$ 作为 $(l_2, \ldots, l_n)$, b 作为 d

三对角的线性方程组在 LU 分解后，用两次回代法解方程也可以利用 L 和 U 的带状结构简化算法。

22.4中求自然样条函数的未知参数 $M_1, M_2, \ldots, M_n$ 的线性方程组就是三对角的。

带状的正定阵的 Cholesky 分解的结果也是带状的，可以简化计算。

例 29.1. 设 $\{\varepsilon_t, t \in \mathbb{Z}\}$ 是随机变量序列，其中 $\mathbb{Z}$ 表示所有整数组成的集合。$E\varepsilon_t \equiv 0$, $\mathrm{Var}(\varepsilon_t) \equiv \delta > 0$, $\mathrm{Cov}(\varepsilon_t, \varepsilon_s) = 0$ 对 $s \neq t$，则称 $\{\varepsilon_t, t \in \mathbb{Z}\}$ 是白噪

声列。若 $0<|b|<1$，

$$X_t=\varepsilon_t+b\varepsilon_{t-1},\ t\in\mathbb{Z}, \tag{29.3}$$

称 $\{X_t\}$ 为 MA(1) 序列。讨论其方差结构。

设各 ε_t 服从 N(0,δ) 分布。若有 $\{X_t\}$ 的观测 $X_1,X_2,\dots,X_n$，则其联合分布完全依赖于 b,δ。令 $X=(X_1,X_2,\dots,X_n)^T$，则 $EX=0$, $\mathrm{Var}(X)=\Sigma$, Σ 是一个三对角正定矩阵，$\sigma_{ii}=\delta(1+b^2)$, $\sigma_{i+1,i}=\sigma_{i,i+1}=\delta b$。为了求 b,δ 的最大似然估计，注意到样本 $X=x$ 的对数似然函数为

$$\ln L(b,\delta)=-\frac{n}{2}\ln(2\pi)-\frac{1}{2}\ln\det(\Sigma)-\frac{1}{2}x^T\Sigma^{-1}x, \tag{29.4}$$

计算似然函数涉及给定参数 (b,δ) 后求相应的 Σ 的行列式以及计算二次型 $x^T\Sigma^{-1}x$，只要对 Σ 作 Cholesky 分解 $\Sigma=LL^T$, L 也是带状的，只有主对角线和下副对角线存在非零元素，L 满秩，这时 $\det(\Sigma)=[\det(L)]^2=(l_{11}l_{22}\dots l_{nn})^2$, $x^T\Sigma^{-1}x=(L^{-1}x)^T(L^{-1}x)$，只要用回代法解得 $y=L^{-1}x$ 则 $x^T\Sigma^{-1}x=y^Ty$。

29.2 Toeplitz 矩阵

在时间序列分析中，若 $\{\xi_t,t\in\mathbb{Z}\}$ 是平稳时间序列，则 $\xi=(\xi_1,\xi_2,\dots,\xi_n)^T$ 有协方差阵 $\Gamma_n=\big(\gamma(|i-j|)\big)_{n\times n}$, 其中 $\gamma(k)=\mathrm{Cov}(\xi_1,\xi_{k+1})$ 叫做 $\{\xi_t\}$ 的协方差函数。这样形式的矩阵叫做 Toeplizt 矩阵，形式为

$$\Gamma_n=\begin{pmatrix}\gamma(0)&\gamma(1)&\cdots&\gamma(n-1)\\ \gamma(1)&\gamma(0)&\cdots&\gamma(n-2)\\ \vdots&\vdots&\ddots&\vdots\\ \gamma(n-1)&\gamma(n-2)&\cdots&\gamma(0)\end{pmatrix} \tag{29.5}$$

在很一般的条件下 Γ_n 正定。在本小节中假定 Γ_n 正定。如果要求解如下形式的方程，

$$\Gamma_n a^{(n)}=\gamma^{(n)}, \tag{29.6}$$

其中 $a^{(n)}=(a_{n1},a_{n2},\dots,a_{nn})^T$, $\gamma^{(n)}=(\gamma(1),\gamma(2),\dots,\gamma(n))^T$, 可以用一个 Levinson 递推算法求解，只需要 $O(n^2)$ 次运算 (见何书元 (2003) 2.4.3 小节)。

$a^{(n)}$ 叫做时间序列 $\{\xi_t\}$ 的 n 阶 Yule-Walker 系数，可用在 AR 建模或最优线性预报中。

在另外一些问题中可能需要对一般的 $y^{(n)}$ 求解

$$\Gamma_n x^{(n)} = y^{(n)}, \tag{29.7}$$

其中 $x^{(n)} = (x_{n1}, x_{n2}\ \ldots, x_{nn})^T$, $y^{(n)} = (y_1, y_2, \ldots, y_n)^T$, 比如计算 ξ 的似然函数时。(29.6)是(29.7)的特例。下面利用 Toeplitz 矩阵的特殊结构构造(29.7)的高效算法。

首先考虑 Y-W 方程(29.6)的求解。令 P_n 表示把 $(1, 2, \ldots, n)$ 排列为 $(n, n-1, \ldots, 1)$ 的置换阵，则 $P_n P_n = I_n$, $P_n \Gamma_n^{-1} = \Gamma_n^{-1} P_n$, $P_n \Gamma_n^{-1} P_n = \Gamma_n^{-1}$。记 $a_*^{(n+1)} = (a_{n+1,1}, a_{n+1,2}, \ldots, a_{n+1,n})^T$, 则 $n+1$ 阶的 Y-W 方程可以写成如下分块形式

$$\begin{pmatrix} \Gamma_n & P_n\gamma^{(n)} \\ \gamma^{(n)T} P_n & \gamma(0) \end{pmatrix} \begin{pmatrix} a_*^{(n+1)} \\ a_{n+1,n+1} \end{pmatrix} = \begin{pmatrix} \gamma^{(n)} \\ \gamma(n+1) \end{pmatrix} \tag{29.8}$$

利用矩阵消元的想法把上面的第 $n+1$ 个方程中的 $\gamma^{(n)T} P_n$ 变成零，只要用 $-\gamma^{(n)T} P_n \Gamma_n^{-1}$ 左乘前 n 个方程然后加到第 $n+1$ 个方程即可，这时有

$$\Gamma_n a_*^{(n+1)} + a_{n+1,n+1} P_n \gamma^{(n)} = \gamma^{(n)}, \tag{29.9}$$

$$\left(\gamma(0) - \gamma^{(n)T} P_n \Gamma_n^{-1} P_n \gamma^{(n)}\right) a_{n+1,n+1} = \gamma(n+1) - \gamma^{(n)T} P_n \Gamma_n^{-1} \gamma^{(n)}, \tag{29.10}$$

于是

$$a_{n+1,n+1} = \frac{\gamma(n+1) - \gamma^{(n)T} P_n a^{(n)}}{\gamma(0) - \gamma^{(n)T} a^{(n)}} = \frac{\gamma(n+1) - a_{n1}\gamma(n) - \cdots - a_{nn}\gamma(1)}{\gamma(0) - a_{n1}\gamma(1) - \cdots - a_{nn}\gamma(n)}, \tag{29.11}$$

$$a_*^{(n+1)} = a^{(n)} - a_{n+1,n+1} P_n a^{(n)}. \tag{29.12}$$

为求解(29.7)，类似对 $n+1$ 阶的方程进行分块消元，分块形式为

$$\begin{pmatrix} \Gamma_n & P_n\gamma^{(n)} \\ \gamma^{(n)T} P_n & \gamma(0) \end{pmatrix} \begin{pmatrix} x_*^{(n+1)} \\ x_{n+1,n+1} \end{pmatrix} = \begin{pmatrix} y^{(n)} \\ y_{n+1} \end{pmatrix} \tag{29.13}$$

其中 $x_*^{(n+1)}$ 是 $x^{(n+1)}$ 的前 n 个元素组成的列向量。消去第 $n+1$ 个方程中的

$\gamma^{(n)T}P_n$，得

$$x_{n+1,n+1} = \frac{y_{n+1} - \gamma^{(n)T}P_n x^{(n)}}{\gamma(0) - \gamma^{(n)T}a^{(n)}} = \frac{y_{n+1} - x_{n1}\gamma(n) - \cdots - x_{nn}\gamma(1)}{\gamma(0) - a_{n1}\gamma(1) - \cdots - a_{nn}\gamma(n)}, \quad (29.14)$$

$$x_*^{(n+1)} = x^{(n)} - x_{n+1,n+1}P_n a^{(n)}. \quad (29.15)$$

为求解 $x^{(n)}$ 需要同时计算 $a^{(n)}$，总共只需要 $O(n^2)$ 次运算。

29.3 稀疏系数矩阵方程组求解

系数矩阵为带状矩阵时解方程组计算量可以从 $O(n^3)$ 降低到 $O(n)$。更一般的稀疏矩阵的非零元素分布不一定有固定的规律，而且在求解消元过程中可能变得不再稀疏。现代统计模型中经常有数千自变量的情形，涉及的矩阵经常是稀疏矩阵，减少不必要的存储并加速运算可以使这样的问题求解变得可行或者更加高效。在经典的统计问题中，有许多个因素的试验的设计阵是稀疏矩阵。

稀疏矩阵可以只保存非零元素与其所在的行列位置。在设计存储方案时，要考虑到如何使用此矩阵，比如，仅用来计算 Ax 这样的矩阵和向量的乘法，要求解线性方程组 $Ax = b$, 矩阵是否会添加、删除元素或修改元素值，访问矩阵时需要按行访问还是按列访问，等等。

下面假设对稀疏矩阵 A 主要需要计算 $y \leftarrow Ax$ 这样的矩阵和向量的乘法。如果 A, x, y 都可以保存到内存中，不需要修改 A 的内容，则可以把 A 的非零元素存入一个长度为 m 的数组 a(m 是 A 的非零元素个数), 把非零元素的行号存入长度为 m 的数组 r, 列号存入数组 c，即 $a_{ij} \neq 0$ 保存在 a 的第 k 元素中，则 $r_k = i$, $c_k = j$。乘法 $y \leftarrow Ax$ 的伪代码为:

稀疏矩阵乘以向量的算法

输入 a, r, c, x
$y \leftarrow 0$
for(k **in** $1 : m$) { # 计算与第 k 个 A 非零元素有关的乘法
    $i \leftarrow r_k$, $j \leftarrow c_k$
    $y_i \leftarrow y_i + a_k x_j$
}

稀疏矩阵乘以向量的算法
输出 y

以上算法稍作修改就可以适用于 A 的所有非零元素无法同时调入内存，需要分批调入的情形。

如果 A 可能被修改，就要设计存储方法使得能迅速定位元素。比如，可以把 A 的非零元素按行存储为链表，然后保存每行的非零元素个数，并把每行的非零元素所在的列号保存为链表。这样可以在一行中迅速找到某个元素的值, 也可以删除非零元素、添加非零元素、修改元素值。

29.4 用迭代法求解线性方程组

用消元法求解 $Ax = b$ 需要 $O(n^3)$ 次运算, 可以在有限步结束。迭代算法每次给出解 x 的一个近似，下次迭代对此近似进行改进，每次迭代仅需要 $O(n^2)$ 次运算，当达到需要的精度时停止迭代。如果迭代很快收敛，这种方法可以比消元法更快求解。如果 A 是稀疏矩阵，消元法会在消元过程中使矩阵变得稠密，而迭代法则可以保持使用稀疏矩阵，从而减少计算量。

Jacobi 方法是求解线性方程组的一种迭代算法。对线性方程组 $Ax = b$, 首先把 A 分解为 $A = L + D + U$, 其中 L 仅有 A 的严格下三角部分，D 为 A 的主对角部分，U 仅有 A 的严格上三角部分。设迭代的第 k 步已经得到了近似解 $x^{(k)}$, 则在第 $k+1$ 步时对 $i = 1, 2, \ldots, n$ 计算

$$x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}}{a_{ii}} \tag{29.16}$$

写成向量形式为

$$x^{(k+1)} = D^{-1} \left[b - (L + U) x^{(k)} \right]. \tag{29.17}$$

在迭代收敛时，(29.17)变成

$$Dx = b - Lx - Ux \Longleftrightarrow (L + D + U)x = b \tag{29.18}$$

即 $Ax = b$。

显然，Jacobi 方法要求 A 的主对角线元素都不等于零。为了使得迭代收敛，即 $\lim_{k\to\infty}\|x^{(k)}-x\|=0$, 注意到第 $k+1$ 步的误差为

$$\left(x^{(k+1)}-x\right)=-D^{-1}(L+U)\left(x^{(k)}-x\right)=C^{k+1}(x^{(0)}-x),$$

其中 $C=D^{-1}(L+U)$，所以迭代收敛的充分条件为所有 $a_{ii}\neq 0$ 且 $\|C\|<1$，这时

$$\|x^{(k+1)}-x\|\leq\|C\|^{k+1}\|x^{(0)}-x\|\to 0\quad(k\to\infty).$$

其中 $\|C\|$ 是 C 的任意一种矩阵范数。可以证明，在所有 $a_{ii}\neq 0$ 时，Jacobi 算法收敛的充分必要条件为 $\|C\|_2<1$ (参见 [关治 and 陆金甫, 1998] §6.1)。

在某些问题中以上收敛性条件可以得到验证。为了判断迭代是否已经收敛，一般预先指定一个精度 ϵ, 当 $\|x^{(k+1)}-x^{(k)}\|<\epsilon$ 时停止迭代。由于舍入误差的影响，ϵ 只要取为问题需要的精度即可，取过小的 ϵ 有可能会造成算法无法停止。

以上的 Jacobi 方法在第 $k+1$ 步利用 $x^{(k)}$ 得到整个 $x^{(k+1)}$。另外一种迭代方法叫做 Gauss-Seidel 迭代，每次仅更新一个分量，而且更新后的分量值马上在下一步中就可以利用。迭代公式为

$$x_i^{(k+1)}=\frac{b_i-\sum_{j<i}a_{ij}x_j^{(k+1)}-\sum_{j>i}a_{ij}x_j^{(k)}}{a_{ii}},\tag{29.19}$$

写成向量形式为

$$(L+D)x^{(k+1)}=b-Ux^{(k)}\Longleftrightarrow x^{(k+1)}=(L+D)^{-1}(b-Ux^{(k)}),$$

这种方法收敛的充分必要条件为 $L+D$ 可逆且 $\|(L+D)^{-1}U\|_2<1$。特别地，当 A 为对称正定阵时 Gauss-Seidel 迭代必定收敛 (参见 [关治 and 陆金甫, 1998] §6.2)。

对于 Gauss-Seidel 方法，一种改进的方法是 $x_i^{(k+1)}$ 取为(29.19)与 $x_i^{(k)}$ 的加权平均，这种方法称为超松弛 (SOR) 迭代法，某些特殊的系数矩阵可以找到最优的加权因子使得收敛速度达到最优。

当 A 为对称正定阵时，$Ax=b$ 的解等价于 $f(x)=\frac{1}{2}x^TAx-b^Tx$ 的最小值点，可以用函数最优化方法如共轭梯度法求解。

习题

习题 1

设方阵 A 满秩，$a_{ij} = 0$ 对 $i > j + p$ 和 $j > i + q$, 若 A 有 LU 分解 $A = LU$，证明当 $i > j + p$ 时 $l_{ij} = 0$, 当 $j > i + q$ 时 $u_{ij} = 0$。

习题 2

证明29.1关于三对角矩阵的递推算法。

习题 3

编写用 LU 分解方法求解三对角矩阵的算法程序。要在程序中设置分母等于零时快速算法失败的预防措施。

习题 4

对例29.1, 编写输入 b 和 δ 计算对数似然函数(29.4)的高效算法程序, 尽可能不占用额外存储空间。

习题 5

编写算法程序，输入时间序列自协方差函数值 $(\gamma(0), \gamma(1), ..., \gamma(n))$ 和实数值 $(y_1, y_2, ..., y_n)$，用递推算法求解方程(29.6)和(29.7), 输出 $a^{(n)}$ 和 $x^{(n)}$。注意避免中间结果不必要的存储。

Chapter 30

正交三角分解

考虑如下线性模型的估计问题:

$$y = X\beta + \varepsilon, \tag{30.1}$$

其中 X 是 $n \times p$ 已知矩阵 $(n > p)$，y 是 n 维因变量观测值向量，β 是 p 维未知模型系数向量，ε 是 n 维未知的模型随机误差向量。用最小二乘法估计 β，即求 β 使得

$$g(\beta) = \|y - X\beta\|^2 \tag{30.2}$$

最小 (本节中的向量范数 $\|x\|$ 都表示欧式空间中的长度 $\sqrt{x^T x}$)，归结为求解如下正规方程

$$X^T X\beta = X^T y. \tag{30.3}$$

此方程一定有解，当 X 列满秩时有唯一解 $\widehat{\beta} = (X^T X)^{-1} X^T y$。

考虑正规方程(30.3)的数值计算问题。我们需要解出 β(记作 $\widehat{\beta}$), 并计算残差平方和

$$\text{SSE} = \|y - X\widehat{\beta}\|^2. \tag{30.4}$$

在 X 列满秩时，正规方程(30.3)的系数矩阵 $X^T X$ 是正定阵，可以用 Cholesky 分解方法高效地计算出 $\widehat{\beta}$ 和 SSE，并且所需的存储空间也只有 $O(p^2)$ 阶。但

是，X^TX 相当于做了平方，当 X^TX 条件数 $\kappa_2(X^TX)$ 很大时，直接求解正规方程(30.3)会引入很大误差。有结果表明（参见 Stewart [1973] 定理 5.2.4），$\widehat{\beta}$ 的相对误差有如下控制式:

$$\frac{\|\Delta\widehat{\beta}\|}{\|\widehat{\beta}\|} \leq \sqrt{\kappa_2(X^TX)}\frac{\|\Delta\widehat{y}\|}{\|\widehat{y}\|}, \tag{30.5}$$

其中 $\widehat{y} = X\widehat{\beta}$, $\Delta\widehat{y}$ 是观测值 y 中的误差引起的 $\widehat{y}$ 的误差，$\Delta\widehat{\beta}$ 是观测值 y 中的误差引起的 $\widehat{\beta}$ 的误差。所以，应该寻找计算更稳定的算法来求解最小二乘问题(30.2)。另外，在样本量 n 很大时，计算 X^TX 时的累积误差可能导致其实际不正定，使得 Cholesky 分解算法失败。新的算法最好能够在 X 非列满秩时也可以求出适当的最小二乘解。直接利用 X 计算而不是对 X^TX 计算可以提高稳定性。

注意，在统计数据分析中，数据中的随机误差和舍入误差往往超过计算中的误差，但不稳定的算法会放大这些数据中的误差。一般只要控制计算中造成的误差比随机误差小一个数量级就可以满足要求。

30.1 Gram-Schmidt 正交化方法

定义 30.1 (QR 分解). 对 $n \times p$ 矩阵 X，若有 $n \times p$ 矩阵 Q 满足 $Q^TQ = I_p$，以及 p 阶上三角矩阵 R 使得

$$X = QR, \tag{30.6}$$

则称(30.6)为矩阵 X 的 QR 分解，或正交—三角分解。

假设最小二乘问题(30.2)中的 X 有 QR 分解 $X = QR$ 且 R 满秩。设 $Q^* = (Q \,|\, Q_2)$ 是 n 阶正交阵，则

$$\|y - X\beta\|^2 = \|(Q^*)^T(y - QR\beta)\|^2 = \|Q^Ty - R\beta\|^2 + \|Q_2^Ty\|^2, \tag{30.7}$$

$\|y - X\beta\|^2$ 与 $\|Q^Ty - R\beta\|^2$ 有相同的最小值点，用回代法求解

$$R\beta = Q^Ty \tag{30.8}$$

就可以得到最小二乘估计 $\widehat{\beta}$, 而(30.7)式右边的 $\|Q_2^Ty\|^2$ 就是残差平方和。由于 $X^TX = R^TR$，R^T 是下三角形矩阵，所以从 QR 分解也可以得到 Cholesky 分解 (可能符号有差别)。

(30.6)实际是把 X 的各列进行了正交化。把 X 的第一列标准化为 Q 的第一列，把 X 的第一、二列正交化和标准化得到 Q 的第二列，以此类推，这正是线性代数中的 Gram-Schmidt 正交化过程, 算法用伪代码表示如下:

QR 分解的 Gram-Schmidt 算法 (RGS)

令 $r_{11} \leftarrow \|X_{\cdot 1}\|$, $Q_{\cdot 1} \leftarrow r_{11}^{-1} X_{\cdot 1}$.
for(j **in** $2:p$) { # 求解 $Q_{\cdot j}$ 和 R 的第 j 列
    $Q_{\cdot j} \leftarrow X_{\cdot j}$
    for(k **in** $1:(j-1)$){
        $r_{kj} \leftarrow X_{\cdot j}^T Q_{\cdot k}$
        $Q_{\cdot j} \leftarrow Q_{\cdot j} - r_{kj} Q_{\cdot k}$
    }
    $r_{jj} \leftarrow \|Q_{\cdot j}\|$, $Q_{\cdot j} \leftarrow r_{jj}^{-1} Q_{\cdot j}$
} # end for j
输出 Q 和 R

以上的 Gram-Schmidt 算法 (记作 RGS) 虽然得到了 $X^T X$ 的 QR 分解，但是该算法得到的 Q 矩阵由于计算误差影响可能会正交性不够好，使得用这样的到 Q 矩阵以及(30.8)求解 $\widehat{\beta}$ 的条件数与直接求解正规方程(30.3)的条件数相同。对 Gram-Schmidt 算法略作修改就可以得到更高精度的分解结果。

修正的 Gram-Schmidt 算法 (记作 MGS) 仅仅调整了计算的次序，并把 Q 的分解结果保存在了 X 的存储空间中。注意

$$r_{ij} = Q_{\cdot i}^T X_{\cdot j} = Q_{\cdot i}^T \left(X_{\cdot j} - \sum_{k=1}^{i-1} b_{ik} Q_{\cdot k} \right),\ j = i+1, \dots, p$$

其中 b_{ik} 为任意实数。在 MGS 的第 i 步，Q 的前 $i-1$ 列已经求得并保存在 X 的前 $i-1$ 列中，并且 R 的前 $i-1$ 行作为 Q 的前 $i-1$ 列与 X 的所有列的内积已经求得，另外，X 的第 i 到第 p 列中已经扣除了 Q 的前 $i-1$ 列的投影。所以，第 i 步只需要求当前保存在 X 的第 i 列中的向量长度作为 r_{ii}，然后将当前保存在 X 第 i 列中的向量标准化作为 Q 的第 i 列，并计算 $Q_{\cdot i}$(保存在 $X_{\cdot i}$ 中) 与当前 X 的第 $i+1, \dots, p$ 列的内积，作为 $r_{i,i+1}, r_{i,i+2}, \dots, r_{i,p}$ 的值，然后从 X 的第 $i+1, \dots, p$ 列中扣除 $Q_{\cdot i}$(保存在 $X_{\cdot i}$ 中) 的投影。所以，算法的第 i 步将求得 Q 的第 i 列 (保存在 $X_{\cdot i}$ 中) 和 R 的第 i 行，并把 X 的

第 $i+1,\ldots,p$ 列减去其在 $Q_{\cdot i}$(保存在 $X_{\cdot i}$ 中) 上的投影。这样，循环到某后续列 $j'>j$ 时，X 的第 j' 列中该减去的投影就都已经减去了，只需要标准化第 j' 列。算法用伪代码表示如下:

QR 分解的修正 GS 算法 (MGS)

---

for(i **in** $1:p$) { # 求解 $Q_{\cdot i}$ 和 R 的第 i 行
    $r_{ii} \leftarrow \|X_{\cdot i}\|$, $X_{\cdot i} \leftarrow r_{ii}^{-1} X_{\cdot i}$
    for(j **in** $(i+1):p$){
        $r_{ij} \leftarrow X_{\cdot i}^T X_{\cdot j}$
        $X_{\cdot j} \leftarrow X_{\cdot j} - r_{ij} X_{\cdot i}$
    }
} # end for i
输出 $Q = X$ 和 R

上面的伪代码用了 R 语言的语法，但是没有特别针对 R 语言特点进行定制，所以伪代码不能作为真正的 R 语言程序。在 R 语言中，`for j in (i+1):p` 在运行到 `i=p` 时会出错，因为这时相当于 `for j in (p+1):p`，在其他语言中这会执行 0 次循环，但 R 语言会执行 `j=p+1` 和 `j=p` 两步。

在以上的 RGS 和 MGS 算法过程中，r_{jj} 是 X 的第 j 列对 X 的第 $1,2,\ldots,j-1$ 列作线性回归 (无截距项) 得到的残差平方和的平方根, 所以某一步中如果 $r_{jj}=0$ 说明 X 的第 j 列与前面的列共线。在 MGS 算法过程中，第 j 步以后，X 的第 $j+1,j+2,\ldots,p$ 列中保存的是原始的 X 的那些列关于 Q 的前 j 列回归 (无截距项) 后的残差, 这时类似于解线性方程组时的主元方法, 可以优先选后续列中向量范数最大的列对应的自变量进入模型。

在 X 列满秩时，RGS 和 MGS 算法得到的上三角阵 R 的主对角线元素都为正值，所以 $X^TX = R^TR$ 是 X^TX 的 Cholesky 分解。

如果 X 不满秩，$\mathrm{rank}(X) = r < \min(n,m)$，则存在 $V_{n\times r}$, $U_{m\times r}$, $V^TV = I_r$, $U^TU = I_r$，使得 $X = VDU^T$，其中 D 是 $n\times m$ 对角阵，前 r 个主对角线元素为正，其余元素均为零，这称为矩阵的奇异值分解，见特征值和奇异值。

30.2 Householder 变换 (*)

Householder 变换方法逐次对 X 进行正交变换把 X 变成上三角形，这样就得到了 X 的 QR 分解。

设 $x = (x_1, x_2, ..., x_m)^T$ 是一个 m 维向量，找一个正交变换把 x 最后 $m-1$ 个元素都变成零。取 $U = I_m - duu^T$, 其中 $d = 2/u^Tu$，易见对任意的 m 维非零向量 u，U 都是对称的正交方阵 ($U^TU = UU^T = I_m$), 且对任意非零常数 b，bu 和 u 对应相同的 U 矩阵。给定向量 x 后，来求 u 使得 Ux 仅有第一个元素非零，即 $Ux = se_1$(e_1 是仅有第一个元素为 1，其它元素都等于 0 的 m 维向量)。

由于正交变换是使得向量长度不变的，有 $\|x\|^2 = \|Ux\|^2 = s^2\|e_1\|^2 = s^2$, 即 $s^2 = \|x\|^2 = x^Tx$。由 $Ux = se_1$ 得 $(du^Tx)u = x - se_1$, 即 u 是 $x - se_1$ 的数乘结果，因为 u 和 u 的非零数乘结果对应相同的矩阵 U, 所以不妨取 $u = x - se_1$，其中 $s = \|x\|$。容易验证，这样选取的 u 使得 $Ux = \|x\|e_1$。对向量 x 的这种正交变换叫做 **Householder 变换**。因为矩阵 uu^T 秩为 1，这种变换也称为秩 1 的更新。

可以看出，对任何的 m 维向量 x 都可以用 Householder 变换把最后 $m-1$ 个元素都变成零。对一个 $n \times p$ 矩阵 $X(n > p)$, 可以左乘 $X_{\cdot 1}$ 的 Householder 变换阵，把 $X_{\cdot 1}$ 的最后 $n-1$ 个元素变成零；然后对变换后的 X, 左乘一个变换矩阵，使得 X 的第 1 行不变，而使得 $X_{\cdot 2}$ 的最后 $n-2$ 个元素变成零，即仅对 X 的第 2 到 n 行作 $n-1$ 维的 Householder 变换，而且第一列中的最后 $n-1$ 个零不变。以此类推，在第 j 次变换时仅对 X 的第 j 到 n 行作 $n-j+1$ 维的 Householder 变换。于是，变换 p 次后，X 变成了一个上三角矩阵 R ($n \times p$ 的上三角矩阵是指当 $i > j$ 时总有 $r_{ij} = 0$)。

令 $X^{(0)} = X$,

$$U_j = \begin{pmatrix} I_{j-1} & 0 \\ 0 & U^{(j)} \end{pmatrix}, \quad X^{(j)} = U_j X^{(j-1)},$$

其中 U_j 是 $X_{\cdot j}^{(j-1)}$ 的最后 $n-j+1$ 个元素的 Householder 变换矩阵。用 $x^{(j)}$ 表示 $X_{\cdot j}^{(j-1)}$ 的最后 $n-j+1$ 个元素，令 $s_j = \|x^{(j)}\|$, $u^{(j)} = x^{(j)} - s_j e_1^{(n-j+1)}$, $U^{(j)} = I_{n-j+1} - \frac{2}{\|u^{(j)}\|^2} u^{(j)}(u^{(j)})^T$，其中 $e_1^{(n-j+1)}$ 表示仅有第一个元素等于 1，

所有其它元素等于 0 的 $n+1-j$ 维向量。在 p 步变换后得到

$$R^* = U_p U_{p-1} ... U_1 X$$

是一个 $n \times p$ 的上三角阵，设 R^* 的前 p 行组成的矩阵为 R, 则 R 是 p 阶上三角方阵，而 R^* 的后 $n-p$ 行均为零。令 $U^* = U_p U_{p-1} \cdot U_1$, 则 U^* 是一个 n 阶对称正交阵, 设 U^* 的前 p 列组成的 $n \times p$ 矩阵为 U，则

$$X = U^* R^* = (U \quad *) \begin{pmatrix} R \\ 0 \end{pmatrix} = UR,$$

于是用 p 次 Householder 变换得到了 X 的 QR 分解。用 Householder 变换作 QR 分解来求解最小二乘问题是计算精度较高的方法。

编程实现时，在 X 的存储空间中保存每次得到的 $X^{(j)}$，因为第 j 列的最后 $n-j$ 个元素为零，第 j 个元素为 s_j, 所以可以单独保存 $s_1, s_2, ..., s_p$, 并把 $u^{(j)}$ 保存在 X 的第 j 列的最后 $n-j+1$ 个元素的存储空间中。

当 X 列满秩时，R 的主对角线元素都为正值，$X^T X = R^T R$ 是 $X^T X$ 的 Cholesky 分解。

如果 X 存在共线问题，比如，$X_{\cdot j}$ 可以用 X 的前 $j-1$ 列线性表示，则 Householder 变换过程进行了 $j-1$ 次以后，因为 X 的前 $j-1$ 列的最后 $n-j+1$ 行的元素都变成了零，所以 X 的第 j 列的最后 $n-j+1$ 个元素也都变成了零，这时 $s_j = \|x^{(j)}\| = 0$。这样在回归计算中可以判断共线是否发生，也可以用类似选主元的方法，在第 j 步看后面的 $n-j+1$ 列的后 $n-j+1$ 行组成的子矩阵中哪一列的向量范数最大，就把那一列对应的自变量与第 j 个自变量交换位置。

30.3 Givens 变换 (*)

Givens 变换也是一种正交变换，是二维向量的旋转变换的推广，可以把向量的指定元素变成零。

对二维非零向量 $x = (x_1, x_2)^T$, 把 x 看成直角坐标平面上的向量，作旋转变换把新的 x 轴旋转到 x 的方向上，设旋转角度为 θ, 则

$$\cos\theta = \frac{x_1}{\sqrt{x_1^2 + x_2^2}}, \quad \sin\theta = \frac{x_2}{\sqrt{x_1^2 + x_2^2}},$$

旋转变换矩阵和变换结果为

$$R = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = (x_1^2 + x_2^2)^{-\frac{1}{2}} \begin{pmatrix} x_1 & x_2 \\ -x_2 & x_1 \end{pmatrix},$$

$$Rx = \begin{pmatrix} \sqrt{x_1^2 + x_2^2} \\ 0 \end{pmatrix}.$$

对 n 维向量 x, 为了把 x_k 变成零，可以对 $(x_i, x_k)(i < k)$ 作旋转使得 x_k 变成零，保持其它元素不变。这相当于左乘了一个正交变换矩阵 U_{ik}, U_{ik} 与单位阵 I_n 仅在 4 个元素上有差别: 其 $(i,i),(i,k),(k,i),(k,k)$ 元素组成的子矩阵恰好为 (x_i, x_k) 对应的旋转阵

$$R_{ik} = (x_i^2 + x_k^2)^{-\frac{1}{2}} \begin{pmatrix} x_i & x_k \\ -x_k & x_i \end{pmatrix}, \tag{30.9}$$

则 $U_{ik}x$ 除去第 i,k 号元素以外不变，第 i 号元素变成 $\sqrt{x_i^2 + x_k^2}$, 第 k 号元素变成 0。

仿照用 Householder 变换进行 QR 分解的方法，利用 Givens 变换也可以对 $n \times p$ 矩阵 $X(n > p)$ 作 QR 分解。首先，用 $n-1$ 次 Givens 变换可以把 X 的第 1 列的最后 $n-1$ 个元素变成零，每次用该列主对角线元素与其下面的元素构成 Givens 变换。然后，用 $n-2$ 次 Givens 变换可以把 X 的第 2 列的最后 $n-2$ 个元素变成零，每次用该列主对角线元素与其下面的一个元素构成 Givens 变换。设已经把 X 的前 $j-1$ 列变成了上三角形 (前 $j-1$ 列主对角线下方的元素都变成了零), 在第 j 步，用 $n-j$ 次 Givens 变换把主对角线下方的元素都变成零，每次用该列主对角先元素与其下面的一个元素构成 Givens 变换，注意此变换仅影响到第 j 行和第 k 行 $(k > j)$, 所以不会影响到所有列的前 $j-1$ 行元素，也就不会影响到前 $j-1$ 列的上三角部分，前 $j-1$ 列已经变成零的那些元素也保持为零，第 j 列中已经变成零的元素也保持为零。注意，第 j 步关于第 j 和第 k 元素作 Givens 变换时，仅需对当前 X 的第 j,k 两行和第 $j, j+1, \ldots, p$ 列组成的两行的子矩阵作二维的 Givens 变换即可, 并且对第 j 列的两个元素的变换结果分别为 $\sqrt{x_j^2 + x_k^2}$ 和 0，不用重复计算。如此进行 p 步后就把 X 变成了上三角形，共需要进行 $(n-1) + (n-2) + \cdots + (n-p) = np - \frac{1}{2}p(p+1)$ 次 Givens 变换。

用 Givens 变换作 QR 分解，还可以按照每次消除主对角下方的一行的方式: 首先消除主对角线下方第 2 行 (只有 $(2,1)$ 号元素), 然后消除主对角线下方第 3

行 (有 $(3,1,),(3,2)$ 两个元素), 如此重复直到消除了主对角线下方第 n 行元素 (该行除主对角元素以外的所有元素)。容易看出，按这种次序变换，在消去第 $i+1$ 行下三角部分的元素时，不会影响前面的第 $1,2,\ldots,i$ 行中下三角部分已经变成零的元素，但是会改变第 $1,2,\ldots,i$ 行的上三角部分 (包含对角线) 的元素。在回归计算中采用这种变换次序可以适应不断获得新的观测需要更新回归结果的情况（参见 Monahan [2001] §5.8）。

借助于 QR 分解可以很容易地计算帽子矩阵 $X(X^TX)^{-1}X^T$ 的主对角线元素 h_i 的值，并由此计算多种回归诊断统计量，比如外学生化残差。回归的一般线性约束的假设检验也可以借助于正交分解的方法计算。参见 [Monahan, 2001] §5.9 和 §5.10。**消去变换 (sweep)** 是解线性方程组时完全消元法（把系数矩阵通过初等变换化为单位阵）的变种，在原来系数矩阵的存储空间中保存得到的逆矩阵，为回归变量选择的计算提供了很多便利，参见 [高惠璇, 1995] §5.6 和 [Monahan, 2001] §5.12。

在 R 软件中，用 `qr()` 函数可以计算矩阵的 QR 分解。

Julia 程序示例见Julia 的 Givens 变换示例 。

习题

习题 1

对 $n \times p$ 矩阵 X（设 $n > p$）编写用 Gram-Schmidt 正交化方法和修正的 Gram-Schmidt 方法作 QR 分解的程序。

习题 2

对 $n \times p$ 矩阵 X（设 $n > p$）编写用 Householder 变换方法作 QR 分解的程序。

习题 3

对 $n \times p$ 矩阵 X（设 $n > p$）编写用 Givens 变换方法作 QR 分解的程序。

习题 4

对线性模型 (30.1), 设 X 列满秩，编写用 QR 分解求 $\widehat{\beta}$、SSE、回归残差 $e = y - X\beta$ 和 $(X^TX)^{-1}$ 的 R 程序。

Chapter 31

特征值和奇异值

31.1 特征值和奇异值定义

在多元统计和时间序列分析中会用到特征值和奇异值, 比如，主成分分析、典型相关分析、对应分析、多元自回归模型等。

先简单回顾线性代数中特征值的定义和性质。设 A 为 n 阶方阵，若有非零向量 α 和复数 λ 使得

$$A\alpha = \lambda\alpha, \tag{31.1}$$

则称 λ 是矩阵 A 的一个**特征值**，α 是特征值 λ 对应的**特征向量**。特征向量具有某种不变性：矩阵 A 左乘特征向量，不改变特征向量的方向（没有正反）。

当 A 为 n 阶实对称阵时，A 恰有 n 个实特征值，记作 $\lambda_1, \lambda_2, \dots, \lambda_n$, 相应的特征向量也都是实向量, 且存在 n 阶正交阵 U 使得

$$A = U\Lambda U^T = \sum_{j=1}^{n} \lambda_j u_{\cdot j} u_{\cdot j}^T, \tag{31.2}$$

其中 $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$ 是对角线元素为 A 的特征值的对角阵, U 的第 j 列 $u_{\cdot j}$ 是 λ_j 对应的特征向量。

当 A 为非负定阵时，所有特征值非负；当 A 为正定阵时，所有特征值都是正数。A 为非负定阵时，令 $\Lambda^{1/2} = \mathrm{diag}(\lambda_1^{1/2}, \lambda_2^{1/2}, \dots, \lambda_n^{1/2})$, 令 $A^{1/2} = U\Lambda^{1/2}U^T$，

则

$$(A^{1/2})^2 = U\Lambda^{1/2}U^TU\Lambda^{1/2}U^T = U\Lambda^{1/2}\Lambda^{1/2}U^T = U\Lambda U^T = A.$$

如果 A 是对称阵且 $\lambda_1, \dots, \lambda_n$ 中仅有 $\lambda_1, \dots, \lambda_r$ 绝对值较大，其余特征值接近于零，则矩阵 A 有如下的近似：

$$A \approx \sum_{j=1}^{r} \lambda_j u_{\cdot j} u_{\cdot j}^T, \tag{31.3}$$

在统计问题如典型相关分析中还会遇到广义的特征值问题: 设 A, B 为 n 阶方阵，若有复数 λ 和非零向量 α 使得

$$A\alpha = \lambda B\alpha, \tag{31.4}$$

则称 λ 和 α 分别为矩阵 A 相对于矩阵 B 的**广义特征值**和**广义特征向量**。实际问题中，B 通常是正定阵，A 是实对称阵，这时(31.4)等价于 $B^{-1}A\alpha = \lambda\alpha$，可以化为普通特征值问题，并可利用 Cholesky 分解进行计算。设 B 有 Cholesky 分解 $B = LL^T$, 则由 $A\alpha = \lambda LL^T\alpha$ 得 $L^{-1}A(L^T)^{-1}(L^T\alpha) = \lambda(L^T\alpha)$, 求解普通特征值问题 $\left(L^{-1}A(L^T)^{-1}\right)\beta = \lambda\beta$ 得 λ 和 β 再求解 $L^T\alpha = \beta$ 即可得广义特征值和广义特征向量。

对 n 阶非奇异矩阵 A, 必存在 n 阶正交阵 U 和 V，使得

$$A = V\mathrm{diag}(d_1, d_2, \dots, d_n)U^T, \tag{31.5}$$

其中 $d_1, d_2, \dots, d_n$ 是正定阵 A^TA 的 n 个特征值的算数平方根, 称(31.5)为矩阵 A 的**奇异值分解**, $d_1, d_2, \dots, d_n$ 称为 A 的**奇异值**。

若 A 是一般的 $n \times m$ 非零矩阵，A 的秩为 $\mathrm{rank}(A) = r \le \min(n, m)$, A^TA 的非零特征值为 $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_r > 0$, 令 $d_i = \sqrt{\lambda_i}$, $i = 1, 2, \dots, r$, 则称 d_i 为 A 的**奇异值**, 且一定有 m 阶正交阵 U 和 n 阶正交阵 V 使得

$$A = VDU^T = V_1D_1U_1^T, \tag{31.6}$$

其中 $U = (U_1|U_2)$, $V = (V_1|V_2)$, U_1 和 V_1 分别为正交阵 U 和 V 的前 r 列，$D_1 = \mathrm{diag}(d_1, d_2, \dots, d_r)$,

$$D = \begin{pmatrix} D_1 & 0 \\ 0 & 0 \end{pmatrix}_{n\times m}, \tag{31.7}$$

称(31.6)为 A 的**奇异值分解** (singular value decomposition, SVD)。可以看出

$$A = \sum_{i=1}^{r} d_i v_i u_j^T, \tag{31.8}$$

$D_1 = (v_1, \dots, v_r)$，$U_1 = (u_1, \dots, u_r)$。当 r 比较小或者奇异值中仅有少数几个较大而其它奇异值都接近于零时，可以近似 A 为

$$A \approx \sum_{i=1}^{r'} d_i v_i u_j^T. \tag{31.9}$$

其中 r' 远小于 $\max(n, m)$。

详见 高惠璇 [1995] §5.4 和 Monahan [2001] §6.6。

31.2 对称阵特征值分解的 Jacobi 算法 (*)

矩阵 A 的特征值 λ 是 A 的特征多项式 $A - \lambda I$ 的根，但直接求多项式的根并不容易，特征值和特征向量的计算一般都通过迭代算法实现。

§30.3引入的 Givens 变换是一个旋转变换，可以仅改变向量中指定的两个元素并使得第二个指定元素变成零。类似这样仅改变向量中第 i, j 两个元素的旋转变换矩阵可以写成

$$G_{ij}(\theta) = \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & \cos\theta & & & & \sin\theta & & \\ & & & & 1 & & & & & \\ & & & & & \ddots & & & & \\ & & & & & & 1 & & & \\ & & & -\sin\theta & & & & \cos\theta & & \\ & & & & & & & & 1 & \\ & & & & & & & & & \ddots \\ & & & & & & & & & & 1 \end{pmatrix}. \tag{31.10}$$

若 A 为对称阵，适当选取角度 θ 对 A 作如下变换

$$A^* = G_{ij}(\theta) A \left(G_{ij}(\theta)\right)^T \tag{31.11}$$

可以使得 $a^*_{ij} = a^*_{ji} = 0$，这样的变换叫做 **Jacobi** 变换。对 A 反复地作 Jocobi 变换可以使得非对角线元素趋于零。

考虑 Jacobi 变换(31.11)中角度 θ 的确定。显然，A^* 和 A 的不同仅体现在第 i,j 行和第 i,j 列，其它元素保持不变; $G_{ij}(\theta)A$ 与 A 仅在第 i,j 行有差别，$A\left(G_{ij}(\theta)\right)^T$ 与 A 仅在第 i,j 列有差别，A^* 与 $G_{ij}(\theta)A$ 仅在第 i,j 列有差别。简单推导可得

$$a^*_{ij} = \frac{1}{2}(a_{jj} - a_{ii})\sin 2\theta + a_{ij}\cos 2\theta, \tag{31.12}$$

只要取 $\theta \in (-\frac{\pi}{4}, \frac{\pi}{4})$ 使得

$$\tau = \cot 2\theta = \frac{a_{ii} - a_{jj}}{2a_{ij}} \tag{31.13}$$

即可使 $a^*_{ij} = a^*_{ji} = 0$。

注意到 $G_{ij}(\theta)$ 仅依赖于 $\cos\theta$ 和 $\sin\theta$, 设 $x = \tan\theta$，由三角函数公式得

$$x^2 + 2\tau x - 1 = 0. \tag{31.14}$$

x 有两个根，为保证 $|\theta| \le \frac{\pi}{4}$ 取其中绝对值较小的一个，为

$$x = \tan\theta = \mathrm{sgn}(\tau)(-|\tau| + \sqrt{\tau^2+1}) = \frac{\mathrm{sgn}(\tau)}{|\tau| + \sqrt{\tau^2+1}}, \tag{31.15}$$

(其中 sgn(·) 表示符号函数，对非负数取 1，对负数取-1) 从 $x = \tan\theta$ 再计算出

$$\cos\theta = (1+x^2)^{-\frac{1}{2}}, \quad \sin\theta = x\cos\theta. \tag{31.16}$$

这样求 $\cos\theta$ 和 $\sin\theta$ 避免了三角函数计算并且 x 的计算方法考虑到了避免两个相近数相减造成精度损失的问题。

设 A 为实对称阵，J_{ij} 是 A 关于下标 (i,j) 的的 Jacobi 变换阵，令 $A^* = J_{ij}AJ_{ij}^T$，则 A^* 有如下性质:

i) A^*仍为对称阵;

ii) $a^*_{ij} = a^*_{ji} = 0$, 且对$k,t \ne i,j$有$a^*_{kt} = a_{kt}$;

iii) 对$t \ne i,j$有$(a^*_{it})^2 + (a^*_{jt})^2 = a^2_{it} + a^2_{jt}$;

iv) $\sum_i \sum_j (a^*_{ij})^2 = \sum_i \sum_j a^2_{ij}$;

v) $\mathrm{off}(A^*) = \mathrm{off}(A) - 2a^2_{ij}$, 其中 $\mathrm{off}(A)$ 表示 A 中非对角线元素的平方和.

Jacobi 算法从 $A^{(0)}=A$ 和 $U^{(0)}=I_n$ 出发反复作 Jacobi 变换，设已有 $A^{(k-1)}$, 则在第 k 步求 $A^{(k-1)}$ 的非对角元素中绝对值最大者，设为其 (i_k,j_k) 元素，用(31.13)和(31.16)针对 $A^{(k-1)}$ 和 (i_k,j_k) 求出 Jacobi 变换矩阵 $J^{(k)}$, 则令 $U^{(k)}=U^{(k-1)}(J^{(k)})^T$, $A^{(k)}=J^{(k)}A^{(k-1)}(J^{(k)})^T$, 如此重复直到 $A^{(k)}$ 的非对角元素的绝对值最大值小于预定的精度 ϵ。这时有 $A=U\Lambda U^T$, $U=U^{(k)}$ 是正交阵，Λ 近似为对角阵。

可以证明上述 Jacobi 算法的 $A^{(k)}$ 收敛到一个对角阵且对角线元素为 A 的特征值。此算法收敛较快，但每次寻找非对角元素中绝对值最大的一个比较耗时。改进的 Jacobi 算法从 A 和 $U=I$ 出发，在第 k 步时基于上一步的 A 计算一个界限 $\epsilon_k=\sqrt{\text{off}(A^{(k)})}/[n(n-1)]$, 然后对 A 的每个严格上三角元素都做一次 Jacobi 变换，用变换后的矩阵代替原来的 A 并更新矩阵 U，但是若该严格上三角元素绝对值小于 ϵ_k 就跳过该元素。所有严格上三角元素都处理过一遍才进入第 $k+1$ 步并计算新的 ϵ_{k+1}, 重复运算直到 ϵ_{k+1} 小于预先指定的误差限 ϵ 为止。

在 R 软件中，用 `eigen()` 函数计算特征值和特征向量。

31.3 用 QR 分解方法求对称矩阵特征值分解 (*)

计算实对称矩阵特征值分解的一种较好的方法是利用 Householder 变换和 Givens 变换。首先，若 A 为 n 阶实对称矩阵，可以用 $n-2$ 个 Householder 变换把它变成对称三对角矩阵: 设 H_1 为一个分块对角矩阵，主对角线的第一块为 1 阶单位阵，第二块是把 A 的第一列最后 $n-1$ 个元素中后 $n-2$ 个元素变成零的 Householder 变换阵，则 H_1A 第一列为 $(a_{11},a_{21}^{(1)},0,\dots,0)^T$, 其中 $a_{21}^{(1)}=\sqrt{a_{21}^2+\cdots+a_{n1}^2}$，且 H_1A 的第一行与 A 的第一行完全相同，于是，H_1AH_1 的第一行为 $(a_{11},a_{21}^{(1)},0,\dots,0)^T$, 注意到 H_1 的对称性与正交性，H_1AH_1 仍为对称阵，但是第一列和第一行的最后 $n-2$ 个元素已经变成了零。在第二步，可以构造一个分块对角矩阵 H_2，对角线第一块为 I_2，第二块是把矩阵 H_1AH_1 的第二列中最后 $n-3$ 个元素变成零的 Householder 变换矩阵，把 H_1AH_1 变成 $H_2H_1AH_1H_2$, 易见第一列和第一行不变，第二列和第二行的最后 $n-3$ 个元素变成了零。如此进行下去得到 $A^{(0)}=H_{n-2}\cdots H_1AH_1\cdots H_{n-2}$，使得 $A^{(0)}$ 为三对角对称矩阵。

得到三对角对称矩阵 $A^{(0)}$ 以后，进行 QR 迭代。设经过 $k-1$ 次迭代后得到矩

阵 $A^{(k-1)}$，在第 k 步，先选一个平移量 t_k，对矩阵 $A^{(k-1)}-t_kI$ 用 Givens 变换方法作 QR 分解得到 $A^{(k-1)}-t_kI=Q_kR_k$, 把得到的上三角阵 R_k 右乘 Q_k 再反向平移，得到

$$A^{(k)}=R_kQ_k+t_kI=Q_k^T(A^{(k-1)}-t_kI)Q_k+t_kI=Q_k^TA^{(k-1)}Q_k, \quad (31.17)$$

这样的 $A^{(k)}$ 仍是三对角对称矩阵，如此迭代直到 $A^{(k)}$ 变成对角形。收敛时，$A^{(k)}$ 的对角线元素为各个特征值，$H_1\cdots H_{n-2}Q_1\cdots Q_k$ 的各列为相应特征向量。

具体的算法比较复杂，详见 [Monahan, 2001] §6.5, [Gentle, 2007] §7.4。

31.4 奇异值分解的计算 (*)

设 A 为任意非零 $n\times m$ 实值矩阵，先说明 A 的奇异值分解的存在性。以下用 $\|\cdot\|$ 表示向量的长度，即 $\|\cdot\|_2$。首先，非负定阵 A^TA 和 AA^T 有共同的正特征值 $\lambda_1\geq\lambda_2\geq\cdots\geq\lambda_r>0$ 且个数 r 为矩阵 A 的秩。设 $u^{(i)}$ 是 A^TA 的特征值 λ_i 对应的单位特征向量（长度为 1），则 $A^TAu^{(i)}=\lambda_iu^{(i)}$, 由此式可得 $\|Au^{(i)}\|^2=\lambda_i$。在 $A^TAu^{(i)}=\lambda_iu^{(i)}$ 两边左乘矩阵 A 得到 $(AA^T)(Au^{(i)})=\lambda_i(Au^{(i)})$, 即 $Au^{(i)}$ 是矩阵 AA^T 的对应于特征值 λ_i 的特征向量，长度为 $d_i=\sqrt{\lambda_i}$。设 AA^T 的所有特征向量组成的正交阵为 V, $v^{(j)}$ 是 V 的第 j 列，适当构造的 V 可使得

$$(v^{(j)})^TAu^{(i)}=d_i\delta_{i-j},\ i=1,2,\dots,m,\ j=1,2,\dots,n, \quad (31.18)$$

其中 δ_{i-j} 是 Kronecker 记号，当 $i=j$ 时表示 1，当 $i\neq j$ 时表示 0, 当 $i>r$ 时令 $d_i=0$。把(31.18)式写成矩阵形式即 $V^TAU=D$，$A=VDU^T$(D 的定义见(31.7))，说明任何非零矩阵 A 均有奇异值分解。

以上的证明给出了求奇异值分解的一种方法：先选 A^TA 和 AA^T 中阶数较低一个, 不妨设是 A^TA，求其特征值分解得到 A 的所有奇异值和矩阵 U，然后利用上面的关系得到 AA^T 的对应于非零特征值的特征向量，如果需要再补充适当列向量组成正交方阵 V 即可。这种方法比较简单，但是计算 A^TA 会造成累积误差。

当 A 为 $n\times m$ 的列满秩矩阵时，可以用类似 §31.3的 QR 分解方法来求 A 的奇异值分解。方法描述如下。仿照正交三角分解把一个对称矩阵变成三对角矩阵的做法，我们可以先在 A 的左边乘以 $n-2$ 个对称正交阵把 A 变成上三角形，然后在此上三角形矩阵右边乘以 $n-1$ 个对称正交阵将其变成上双对角阵，

即除对角线和上副对角线外的元素都是零的矩阵。总之，存在 n 阶正交阵 P 和 m 阶正交阵 Q 使得

$$PAQ = \begin{pmatrix} B_{m\times m} \\ 0_{(n-m)\times m} \end{pmatrix}, \tag{31.19}$$

其中 B 的元素满足当 $i > j$ 或 $j > i+1$ 时 $b_{ij} = 0$, 且 B 满秩。

得到上双对角阵 B 后，先求 B^TB 的特征值分解。B^TB 是一个三对角对称阵，可以用 §31.3的 QR 方法求特征值分解。设 $B^TB = U_1D_m^2U_1^T$, 其中 $D_m = \mathrm{diag}(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_m})$, $\lambda_1 \geq \cdots \geq \lambda_m > 0$ 为 B^TB 的所有特征值，U_1 各列为 B^TB 的特征向量。令 $V_1 = BU_1D_m^{-1}$，则 $V_1^TV_1 = I_m$, 于是 $B = V_1D_mU_1^T$ 是 B 的奇异值分解。

这时，

$$\begin{aligned} PAQ = \begin{pmatrix} B \\ 0 \end{pmatrix} = \begin{pmatrix} V_1D_mU_1^T \\ 0 \end{pmatrix} &= \begin{pmatrix} V_1 & 0 \\ 0 & I_{n-m} \end{pmatrix}_{n\times n} \begin{pmatrix} D_m \\ 0 \end{pmatrix}_{n\times m} U_1^T \\ &\triangleq V_2 \begin{pmatrix} D_m \\ 0 \end{pmatrix}_{n\times m} U_1^T, \end{aligned} \tag{31.20}$$

则 V_2 为 n 阶正交阵，可得 A 有奇异值分解

$$A = (P^TV_2)D(QU_1)^T \triangleq VDU^T, \tag{31.21}$$

其中 $V = P^TV_2$ 为 n 阶正交阵，$U = QU_1$ 为 m 阶正交阵，D 为 $n \times m$ 对角阵，对角线元素为 B^TB 的特征值的算数平方根。

习题

习题 1

编写关于实对称矩阵 A 用 Jacobi 方法求特征值分解的程序。

习题 2

编写关于实对称矩阵 A 用改进的 Jacobi 方法求特征值分解的程序。

习题 3

设 A 为 n 阶实对称方阵，编写程序用正交相似变换 $B = QAQ$ 把 A 变换为三对角对称矩阵 B，其中 Q 为 n 阶对称正交阵, 输出 B 和 Q 的值。

习题 4

设 n 方阵 A 为上双对角矩阵: 当 $j \neq i, i+1$ 时总有 $a_{ij} = 0$。证明 $B = A^T A$ 为三对角矩阵。

习题 5

设 A 为 n 阶非负定对称矩阵，用拉格朗日乘子法求 $x^T x = 1$ 条件下 $x^T Ax$ 的最大值点。

习题 6

设 A 为 n 阶非负定对称矩阵，用 A 的特征值分解求 $x^T x = 1$ 条件下 $x^T Ax$ 的最大值点。

Chapter 32

广义逆矩阵

32.1 广义逆的定义和性质

当 A 为满秩方阵时，线性方程组 $Ax = b$ 有唯一解 $x = A^{-1}b$。当 A 为不满秩的方阵或长方形 $n \times m$ 矩阵时，A^{-1} 不存在，这时能否用类似逆矩阵的方式表示线性方程组 $Ax = b$ 的解？$Ax = b$ 可能有唯一解、无穷多个解或无解（无解时可以找最小二乘解），用广义逆矩阵可以统一地给出这些问题的解。

一些统计计算问题的理论研究和数值计算也用到广义逆矩阵，比如，线性模型中参数最小二乘估计的表示, 典型相关分析中典型相关系数和典型变量的计算。

广义逆有多种定义，最常用的一种是加号逆。

定义 32.1 (加号逆和减号逆)**.** 设 A 为 $n \times m$ 矩阵，若 $m \times n$ 矩阵 G 满足

$$\begin{aligned} &\text{i)}\ AGA = A; \\ &\text{ii)}\ GAG = G; \\ &\text{iii)}\ (AG)^T = AG; \\ &\text{iv)}\ (GA)^T = GA, \end{aligned}$$

则称矩阵 G 为矩阵 A 的**加号逆**或 Moore-Penrose 广义逆，记作 A^+。如果 G 满足定义中的第一个条件，则称 G 为 A 的**减号逆**，记作 A^-。

显然，如果 A 本身就是 n 阶可逆方阵，则 A^{-1} 满足上述四个条件且是唯一的。一般情况下有：

定理 32.1. *$n \times m$ 矩阵 A 的加号逆存在且唯一。*

证明: 若 A 不是零矩阵（所有元素都等于零的矩阵），则 A 有奇异值分解 $A = VDU^T$，取 $G = UD^+V^T$, 其中 D^+ 是把对角阵 D 的主对角线中非零元素换成相应元素的倒数，其它元素保持为零，容易验证 G 是 A 的加号逆。当 A 为零矩阵时，$m \times n$ 的零矩阵是 A 的加号逆。

若 A_1^+ 和 A_2^+ 是 $n \times m$ 矩阵 A 的两个加号逆，则

$$\begin{aligned}
&A_1^+ = A_1^+AA_1^+ = A_1^+(AA_1^+)^T = A_1^+(A_1^+)^T(A)^T = A_1^+(A_1^+)^T(AA_2^+A)^T \\
=&A_1^+(A_1^+)^TA^T(AA_2^+)^T = A_1^+(AA_1^+)^TAA_2^+ = A_1^+AA_1^+AA_2^+ = A_1^+AA_2^+ \\
=&(A_1^+A)^TA_2^+ = A^T(A_1^+)^TA_2^+ = (AA_2^+A)^T(A_1^+)^TA_2^+ = (A_2^+A)^TA^T(A_1^+)^TA_2^+ \\
=&(A_2^+A)^T(A_1^+A)^TA_2^+ = A_2^+AA_1^+AA_2^+ = A_2^+AA_2^+ = A_2^+
\end{aligned}$$

可见加号逆存在唯一。证毕。

※※※※※

以上定理证明中给出了用奇异值分解计算加号逆的方法。另一种计算加号逆的方法是利用 “满秩分解”。设 A 为非零的 $n \times m$ 实矩阵，设 $\mathrm{rank}(A) = r$, 则存在 $n \times r$ 的满秩矩阵 B 和 $r \times m$ 的满秩矩阵 C 使得 $A = BC$, 这称为 A 的 **满秩分解** (见习题 3)。这时，A 的加号逆可表示为

$$A^+ = C^T(CC^T)^{-1}(B^TB)^{-1}B^T. \tag{32.1}$$

加号逆也是减号逆，所以减号逆一定存在，但是当 A 不是满秩方阵时 A 的减号逆有无穷多个。

容易看出，当且仅当 A 为满秩方阵时 $A^+ = A^{-1}$。当且仅当 A 为 $n \times m$ 列满秩阵时 $A^+A = I_m$, 这时 $A^+ = (A^TA)^{-1}A^T$；当且仅当 A 为 $n \times m$ 行满秩阵时 $AA^+ = I_n$，这时 $A^+ = A^T(AA^T)^{-1}$。

定理 32.2. *加号逆有如下的性质:*

i) $(A^+)^+ = A$;

ii) $(A^T)^+ = (A^+)^T$;

iii) $(\lambda A)^+ = \lambda^{-1}A^+,\ \forall\lambda \neq 0$;

iv) $rank(A^+) = rank(A) = rank(AA^+) = rank(A^+A)$;

v) $(A^TA)^+ = A^+(A^+)^T$;

vi) $A^+ = (A^TA)^+A^T = A^T(AA^T)^+$;

vii) AA^+和A^+A都是对称幂等矩阵;

viii) 若 A 是对称幂等矩阵，则 $A^+ = A$。

证明留给读者。

定理 32.3. 若 $n \times m$ 矩阵 A 是列满秩矩阵，有 QR 分解 $A = QR$, Q 为 $n \times m$ 矩阵使得 $Q^TQ = I$, R 为 m 阶满秩上三角方阵，则 $A^+ = R^{-1}Q^T$。

证明: 令 $G = R^{-1}Q^T$，则

$$\begin{aligned} AGA =& QRR^{-1}Q^TQR = QR = A \\ GAG =& R^{-1}Q^TQRR^{-1}Q^T = R^{-1}Q^T = G \\ GA =& R^{-1}Q^TQR = I_m \text{ 对称} \\ AG =& QRR^{-1}Q^T = QQ^T \text{ 对称} \end{aligned}$$

证毕。

32.2 叉积阵 (*)

设矩阵 A 为 $n\times m$，矩阵 A^TA 经常出现在统计计算中，比如线性模型的正规方程，协方差阵估计，等等。A^TA 是对称半正定矩阵。A^TA 与 A 之间有密切的联系。$\text{rank}(A^TA) = \text{rank}(A)$, $A = 0$ 当且仅当 $A^TA = 0$，当 $\text{rank}(A) = m \leq n$ 时 A^TA 是对称正定阵。

若 G 是 A^TA 的一个减号逆，则 G^T 也是 A^TA 的减号逆（注意对称阵的减号逆不一定对称）：

$$(A^TA)G^T(A^TA) = [(A^TA)G(A^TA)]^T = (A^TA)^T = (A^TA)$$

A^TA 的广义逆与 A 的广义逆密切相关。

定理 32.4. *对 A^TA 的任一个减号逆 $(A^TA)^-$，$(A^TA)^-A^T$ 是 A 的减号逆。*

证明: 由定理32.2性质 vi),

$$A^+ = (A^TA)^+A^T, \quad A = AA^+A = A(A^TA)^+A^TA,$$

所以

$$\begin{aligned}
&A[(A^TA)^-A^T]A \\
=&AA^+A\cdot[(A^TA)^-A^T]A \\
=&A[(A^TA)^+A^T]A\cdot[(A^TA)^-A^T]A \\
=&A(A^TA)^+\cdot[(A^TA)(A^TA)^-(A^TA)] \\
=&A(A^TA)^+\cdot(A^TA) \\
=&A[(A^TA)^+A^T]A \\
=&AA^+A = A.
\end{aligned}$$

定理证毕。

※※※※※

32.3 正交投影 (*)

设 $\mathbb{R}^n$ 为 n 维欧式空间，即 $x=(x_1,x_2,\dots,x_n)^T$, $x_i\in\mathbb{R}$, $i=1,2,\dots,n$ 这样的元素组成的集合并定义了加法和数乘，又定义了内积

$$(x,y)=x^Ty=\sum_{i=1}^n x_iy_i$$

当 $(x,y)=0$ 时称 x 和 y 正交，记作 $x\perp y$。

设 A 为 $n\times m$ 实数矩阵，记 $A\in\mathbb{R}^{n\times m}$。称

$$\mu(A)=\{A\beta\in\mathbb{R}^n:\ \beta\in\mathbb{R}^m\}$$

为 A 的各列张成的线性子空间。$\mathbb{R}^n$ 的线性子空间是 $\mathbb{R}^n$ 的子集且加法和数乘在此子集中封闭。记

$$\mu(A)^\perp=\{z\in\mathbb{R}^n:\ A^Tz=0\}$$

这是 $\mathbb{R}^n$ 中所有与 A 的各列都正交的向量组成的集合，也是 $\mathbb{R}^n$ 的子空间。

定理 32.5. 设 $A \in \mathbb{R}^{n\times m}$，则对任意 $y \in \mathbb{R}^n$，有 $\widehat{y} = AA^+y \in \mu(A)$, $z = (I - AA^+)y \in \mu(A)^\perp$，使得

$$\begin{cases} y = \widehat{y} + z, \\ \widehat{y} \in \mu(A),\ z \in \mu(A)^\perp,\ \widehat{y} \perp z. \end{cases}$$

证明: 易见 $\widehat{y} \in \mu(A)$，而

$$\begin{aligned} A^Tz =& A^Ty - A^TAA^+y = A^Ty - A^T(AA^+)^Ty \\ =& A^Ty - A^T(A^+)^TA^Ty = A^Ty - A^T(A^T)^+A^Ty \\ =& A^Ty - A^Ty = 0 \end{aligned}$$

所以 $z \in \mu(A)^\perp$，证毕。

※※※※※

定义 32.2. 设 $A \in \mathbb{R}^{n\times m}$，$y \in \mathbb{R}^n$，称 $\widehat{y} = AA^+y$ 为 y 在子空间 $\mu(A)$ 中的**正交投影**。

正交投影满足 $y = \widehat{y} + (y - \widehat{y})$, 其中 $\widehat{y} \in \mu(A)$，而 $y - \widehat{y} \in \mu(A)^\perp$。

定理 32.6. $y \in \mu(A)$ 当且仅当 $AA^+y = y$; $y \in \mu(A)^\perp$ 当且仅当 $AA^+y = 0$。

证明: 若 $y \in \mu(A)$，则存在 β 使得 $y = A\beta$，于是

$$AA^+y = AA^+A\beta = A\beta = y.$$

反之，若 $AA^+y = y$, 记 $\beta = A^+y$，则 $y = A\beta \in \mu(A)$。

若 $y \in \mu(A)^\perp$，即 $A^Ty = 0$，则

$$AA^+y = (AA^+)^Ty = (A^+)^TA^Ty = 0.$$

反之，若 $AA^+y = 0$，则

$$A^Ty = (AA^+A)^Ty = A^T(AA^+)^Ty = A^TAA^+y = 0.$$

证毕。

※※※※※

另外，$\widehat{y}$ 还是 $\mu(A)$ 中与 y 距离最近的元素：

定理 32.7. 设 $A \in \mathbb{R}^{n\times m}$，$y \in \mathbb{R}^n$，$\hat{y} = AA^+y$ 是 y 在 $\mu(A)$ 中的正交投影，则

$$\|y - \hat{y}\|^2 \leq \|y - A\beta\|,\ \forall \beta \in \mathbb{R}^m.$$

证明:

$$\|y - A\beta\|^2 = \|(y - \hat{y}) + (\hat{y} - A\beta)\|^2$$

易见 $\hat{y} - A\beta \in \mu(A)$ 而 $y - \hat{y} \in \mu(A)^\perp$，所以

$$\|y - A\beta\|^2 = \|y - \hat{y}\|^2 + \|\hat{y} - A\beta\|^2 \geq \|y - \hat{y}\|^2$$

证毕。

※※※※※

称 AA^+ 是 $\mathbb{R}^n$ 向 $\mu(A)$ 投影的正交投影阵。

定义 32.3. 设 P 为 n 阶实对称矩阵，若 $P = P^2$，则称 P 为**对称幂等阵**。

AA^+ 是对称幂等阵：

$$(AA^+)^2 = (AA^+A)A^+ = AA^+.$$

定理 32.8. 若 P 是对称幂等阵，则 P 是 $\mathbb{R}^n$ 向 $\mu(P)$ 的正交投影阵。

证明: 要证明 $P = PP^+$，事实上

$$\begin{aligned} PP^+ =& PP^+PP^+ = PP^+PPP^+ \\ =& PP^+P(PP^+)^T = PP^+P(P^+)^TP^T \\ =& PP^+P(P^T)^+P = PP^+PP^+P \\ =& PP^+P = P \end{aligned}$$

证毕。

※※※※※

定理 32.9. 设 $(A^TA)^-$ 是 A^TA 的任一个减号逆，则

$$A(A^TA)^-A^T = AA^+$$

即 $A(A^TA)^-A^T$ 是向 A 的各列张成的线性空间 $\mu(A)$ 的正交投影阵，不依赖于减号逆的选择。

证明:

$$\begin{aligned}A(A^TA)^-A^T =& AA^+A(A^TA)^-(AA^+A)^T\\ =& A(A^TA)^+A^TA(A^TA)^-[A(A^TA)^+A^TA]^T\\ =& A(A^TA)^+A^TA(A^TA)^-A^TA(A^TA)^+A^T\\ =& A(A^TA)^+[A^TA(A^TA)^-A^TA](A^TA)^+A^T\\ =& A(A^TA)^+A^TA(A^TA)^+A^T\\ =& A(A^TA)^+A^T = AA^+\end{aligned}$$

证毕。

※※※※※

32.4 线性方程组通解 (*)

利用广义逆可以讨论线性方程组解的表示。

定理 32.10. *系数矩阵为 $n\times m$ 矩阵的线性方程组 $Ax=b$ 有解的充分必要条件为*

$$AA^+b=b. \tag{32.2}$$

在方程组有解时，对 A 的任何一个减号逆 A^-, $x=A^-b$ 是方程组的解，且方程组的通解为

$$x=A^+b+(I-A^+A)y,\ \forall y\in\mathbb{R}^m. \tag{32.3}$$

证明: 若(32.2)成立，则 $x=A^+b$ 是一个解。反之，若 x 满足 $Ax=b$, 则

$$b=Ax=AA^+Ax=AA^+b$$

成立，即(32.2)成立。另外，方程有解当且仅当 $b\in\mu(A)$，这当且仅当 $AA^+b=b$。

设 A^- 是 A 的任意一个减号逆，若 $x=A^-b$，则

$$\begin{aligned}Ax =& AA^-b=AA^-(AA^+b)\quad(\text{由解的存在性})\\ =& (AA^-A)A^+b=AA^+b=b\quad(\text{由解的存在性}).\end{aligned}$$

即当解存在时 A^-b 是方程组的解。

令 x 由 (32.3) 定义，容易验证它是一个解。反之，若 x 满足 $Ax = b$，则

$$x = A^+b + x - A^+b = A^+b + x - A^+Ax = A^+b + (I - A^+A)x,$$

满足通解形式。实际上，通解 (32.3) 中的 A^+ 也可以替换成 A 的任何一个减号逆。证毕。

※※※※※

定理 32.11. *当线性方程组 $Ax = b$ 有解时，$x = A^+b$ 是所有解中唯一的长度最小的解。*

证明: 有解时通解中 $A^+b \perp (I - A^+A)y$：

$$y^T(I-A^+A)^TA^+b = y^T(I-A^+A)A^+b = y^T(A^+b-A^+AA^+b) = y^T(A^+b-A^+b) = 0,$$

所以

$$\|A^+b + (I - A^+A)y\|^2 = \|A^+b\|^2 + \|(I - A^+A)y\|^2 \geq \|A^+b\|^2.$$

证毕。

※※※※※

32.5 最小二乘问题通解

广义逆可以用来分析回归分析和线性模型问题中最小二乘解的结构。设 X 为 $n \times m$ 矩阵 $(n > m)$，则当 X 列满秩时矩阵 $P = X(X^TX)^{-1}X^T$ 是对称幂等矩阵，可以把向量 y 正交投影到 X 的各列张成的线性空间 $\mu(X)$ 中, 这时最小二乘问题

$$\min_{\beta \in \mathbb{R}^m} \|y - X\beta\|_2^2 \tag{32.4}$$

有唯一解 $\widehat{\beta} = (X^TX)^{-1}X^Ty$。对一般情况有如下结论。

定理 32.12. *设 X 为 $n \times m$ 矩阵 $(n > m)$, 则最小二乘问题(32.4)的所有的最小二乘解可以写成*

$$\widehat{\beta} = X^+y + (I - X^+X)z, \ \forall z \in \mathbb{R}^m. \tag{32.5}$$

在这些最小二乘解中 $\beta_0 = X^+y$ 是唯一的长度最短的解。

证明: 令 $P = XX^+$，则 P 是对称幂等矩阵，Py 是向量 y 到 X 的各列张成的线性子空间 $\mu(X)$ 的正交投影, 且

$$X\hat{\beta} = XX^+y + (X - XX^+X)z = XX^+y = Py,$$

于是

$$\begin{aligned}\|y - X\beta\|_2^2 =& \|y - X\hat{\beta} + X(\hat{\beta} - \beta)\|^2 \\ =& \|y - Py + X(\hat{\beta} - \beta)\|^2 \\ =& \|y - Py\|^2 + \|X(\hat{\beta} - \beta)\|^2 \quad \text{(因为 } Py \text{ 是正交投影)} \\ =& \|y - X\hat{\beta}\|^2 + \|X(\hat{\beta} - \beta)\|^2 \\ \geq& \|y - X\hat{\beta}\|^2,\end{aligned}$$

所以(32.5)是最小二乘问题(32.4)的解, 等号成立当且仅当 $X(X^+y - \beta) = 0$。

若 $\tilde{\beta}$ 是最小二乘问题(32.4)的解，则 $X(X^+y - \tilde{\beta}) = 0$, 于是 $\tilde{\beta}$ 是线性方程组 $X\tilde{\beta} = XX^+y$ 的解，由定理32.10可知存在 z 使得

$$\tilde{\beta} = X^+(XX^+y) + (I - X^+X)z = X^+y + (I - X^+X)z.$$

设 $\hat{\beta}$ 为最小二乘解(32.5)，则

$$\begin{aligned}\|\hat{\beta}\|_2^2 =& \|X^+y\|^2 + \|(I - X^+X)z\|^2 + 2z^T(I - X^+X)X^+y \\ =& \|X^+y\|^2 + \|(I - X^+X)z\|^2 \\ \geq& \|X^+y\|^2,\end{aligned}$$

等号成立当且仅当 $(I - X^+X)z = 0$ 即 $\hat{\beta} = X^+y$。

定理证毕。

※※※※※

习题

习题 1

设 $n \times m$ 非零矩阵 A 的有奇异值分解 $A = VDU^T$，证明 UD^+V^T 为其加号逆, 其中 D^+ 是把对角阵 D 的主对角线中非零元素换成相应元素的倒数，其它元素保持为零。

习题 2

设 A 为 n 阶实对称方阵，B 为 n 阶正定阵。写出用 Cholesky 分解的方法求解广义特征值问题 $A\alpha = \lambda B\alpha$ 的算法，并用编写程序实现该算法。

习题 3

设 A 为非零的 $n \times m$ 实矩阵，且 $\mathrm{rank}(A) = r$, 证明存在 $n \times r$ 的满秩矩阵 B 和 $r \times m$ 的满秩矩阵 C 使得 $A = BC$。

习题 4

设 $n \times m$ 非零的实矩阵 A 有满秩分解 $A = BC$, 证明 A 的加号逆可表示为

$$A^+ = C^T(CC^T)^{-1}(B^TB)^{-1}B^T.$$

习题 5

设 X 为 $n \times p$ 矩阵（$n > p$），y 为 n 维向量，证明正规方程 $X^TX\beta = X^Ty$ 的解 β 中长度最小的一个为 X^+y。

习题 6

证明加号逆的性质 i)—viii)。

习题 7

对 $n \times m$ 矩阵 A，令 $P = A(A^TA)^+A^T$, 证明 $P = AA^+$, 且 P 是对称幂等阵。记 $\mu(A) = \{Ax : \ x \in \mathbb{R}^m\}$, 证明对任意 $y \in \mathbb{R}^n$ 有 $Py \in \mu(A)$ 且

$$z^T(y - Py) = 0, \ \forall z \in \mu(A).$$

习题 8

设 A 为 $n \times m$ 矩阵，若线性方程组 $Ax = b$ 有解，证明 $x_0 = A^+b$ 是所有解中唯一的长度最小的解。

Chapter 33

R 中的矩阵计算 (*)

33.1 Base 包的矩阵功能

可以用 `matrix()` 函数生成一个矩阵，如

```
matrix(1:12, nrow=4, ncol=3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

`A + B` 是矩阵加法，`A - B` 是矩阵减法，`A %*% B` 是矩阵乘法。`x * A` 若 `x` 是标量，`A` 是矩阵，作矩阵数乘。如果 `x` 是向量，`A` 是矩阵，则 `x %*% A` 表示行向量 `x` 左乘矩阵 `A`，`A %*% x` 表示列向量 `x` 右乘矩阵 `A`。

对矩阵 A, B，`crossprod(A)` 计算 A^TA, `crossprod(A, B)` 计算 A^TB。注意，如果 `x` 是向量，为了计算 `x` 的元素平方和，不应该使用 `crossprod` 而应写成 `sum(x^2)`。为了计算两个向量 `x`, `y` 的内积，应该写成 `sum(x * y)`。

`solve(A)` 求方阵 `A` 的逆矩阵；`solve(A, x)` 解线性方程组 $Ax = b$ 求 x。`solve(A, B)`，若 `B` 也是矩阵，求 $A^{-1}B$。

R 函数 `backsolve (R, b)` 求解上三角方程组 $Rx = b$，`forwardsolve(L, b)` 求解下三角方程组 $Lx = b$。

R 函数 `chol(x)` 对正定矩阵作 Cholesky 分解 R^TR，结果 R 是上三角矩阵。默认不使用主元，加 `pivot=TRUE` 选项可以使用主元法，可以对半正定矩阵分解。

如果对正定矩阵 A 做了 Cholesky 分解 $A = R^TR$，可以从 R 计算 $A = R^{-1}R^{-T}$;如果对矩阵 X 得到了 QR 分解 $X = QR$,可以从 R 计算 $(X^TX)^{-1} = (R^TR)^{-1} = R^{-1}R^{-T}$。R 函数 `chol2inv(x)` 计算这两种逆矩阵，其中 `x` 是满秩上三角阵。

`qr(A)` 计算 QR 分解 $A = QR$，并返回压缩格式的计算结果与辅助信息的列表，设结果为 `z`, 可以用 `qr.Q(z)` 返回矩阵 Q, `qr.R(z)` 返回矩阵 R, `qr.solve(z, b)` 利用得到的 QR 分解求解线性方程组 $Ax = b$。

在统计计算中 QR 分解经常用来计算回归分析的最小二乘估计，或用在 Newton-Raphson 优化时计算逆矩阵乘以梯度。设 `z` 为 `qr(X)` 的结果，则 `qr.coef(z, y)` 利用 QR 分解求最小二乘估计 $(X^TX)^{-1}X^Ty$, `qr.qy(z, y)` 计算 Qy, `qr.qty(z, y)` 计算 Q^Ty，`qr.resid(z, y)` 计算残差，`qr.fitted(z, y)` 计算拟合值。

`eigen(A)` 求方阵 A 的特征值和特征向量，结果返回一个列表，其中元素 `values` 为各特征值，`vectors` 是一个方阵，矩阵的各列为对应的特征向量。可以加选项 `symmetric=TRUE` 表示输入的 `A` 是对称阵（输入复数矩阵时表示是 Hermite 阵）。加选项 `only.values=TRUE` 表示仅要求计算并输出特征值，不需要输出特征向量。

`svd(A, nu, nv)` 求奇异值分解 $A = Q\Lambda V^T$，其中 A 是 $n \times p$ 矩阵，`nu` 表示要计算的左奇异向量个数，`nv` 表示要计算的右奇异向量个数，默认 `nu`, `nv` 都是 A 的行、列数最小值。因为计算奇异向量比较慢，所以不需要所有奇异向量时可以取较小的 `nu` 和 `nv`。结果是包含 `d`, `u`, `v` 三个元素的列表，`d` 是长度为`\min(n,p)` 的保存奇异值的向量，`u` 是行数为 n、列数为 `nu` 的左奇异值列向量组成的矩阵，当 `nu=0` 时不输出此项；`v` 是行数为 n、列数为 `nv` 的右奇异值列向量组成的矩阵，当 `nv=0` 时不输出此项。

33.2 Matrix 包的矩阵功能

Matrix 扩展包补充了许多 base 部分没有提供的稠密矩阵功能，并提供了 base 部分没有的稀疏矩阵的支持。Matrix 包使用特有的 Matrix 类型，此类型又有对角阵、三角阵、分块对角阵、稀疏矩阵等变种，Matrix 包的矩阵类型在做了分解后会将分解结果保存在原始矩阵的变量中，有利用分解结果的重复利用。

`Matrix(x, nr, nc)` 生成一个稠密矩阵，与 `matrix()` 函数类似。`cBind()` 和 `rBind()` 与 `cbind()` 和 `rbind()` 类似。`t(A)` 返回转置。

`as(A, "sparseMatrix")` 返回 `A` 转换为稀疏矩阵的结果。系数矩阵的运算结果尽可能保持为稀疏矩阵。

详见 Matrix 包的文档。

33.3 其它包的矩阵功能

`MASS::ginv(X)` 求 `X` 的加号逆 (Moore-Penrose 广义逆)。

Chapter 34

Julia 中的矩阵计算功能 (*)

34.1 基本矩阵运算

Julia 是一个向 Matlab 软件一样强调科学计算，尤其是线性代数运算的编程语言，所以其矩阵功能比 R 还要强大。

Julia 的基本矩阵构造、加减法、乘法、求逆、求解线性方程组等运算，见第 @(jintro) 章。

`size(A)` 返回包含行数和列数的二元组 (tuple)。

`eye(n)` 生成 `n` 阶的元素类型为 Float64 的单位阵。`ones(n, m)` 生成 $n \times m$ 的元素类型为 Float64 的都是 1 的矩阵，`zeros(n,m)` 生成 $n \times m$ 的元素类型为 Float64 的都是 0 的矩阵。

`A'`表示 `A` 的共轭转置。`A + B` 和 `A - B` 表示矩阵加减，`A * B` 表示矩阵乘法。`inv(A)` 返回逆矩阵。`A \ b` 求解 $Ax = b$ 中的 x。

`dot(x, y)` 计算两个向量的内积，与 `sum(x .* y)` 相同。`norm(x)` 求向量的欧式模，等于 `sqrt(sum(x .^ 2))`。

`factorize(A)` 对 `A` 计算适合的分解，自动判断 `A` 的特殊结构。

`det(A)` 求行列式。`trace(A)` 计算迹。

`eigvals(A)` 求特征值。`eig(A)` 返回由特征值、特征向量组成的二元组，特征值储存在一个向量中，特征向量储存在矩阵的各列中。

`eig(A, B)` 求解广义特征值问题 $Av = \lambda Bv$。

`rank(A)` 求秩。

Julia 的线性代数模块提供了丰富的矩阵类型和矩阵计算功能，详见 Base.LinAlg 模块的文档。

34.2 用 Givens 变换作 QR 分解示例

```
using Distributions # 随机数函数
srand(101) # 指定起始种子
A=round.(rand(4,4), 2)

##  4×4 Array{Float64,2}:
##   0.09  1.0   0.97  0.42
##   0.83  0.86  0.62  0.8
##   0.63  0.8   0.75  0.23
##   0.16  0.95  0.73  0.07

## Givens 变换矩阵，在 i1,i2 两行和两列作用，将 (a, b) 长度集中到 a 所在元素
function givensmat(n, a, b, i1, i2)
    submat = [a b; -b a]/sqrt(a*a+b*b)
    resmat = eye(n)
    resmat[[i1,i2], [i1,i2]] = submat
    resmat
end

n=4
Q1 = givensmat(n, A[1,1], A[2,1], 1, 2)

##     4×4 Array{Float64,2}:
```

```
##       0.107802  0.994172  0.0  0.0
##      -0.994172  0.107802  0.0  0.0
##       0.0       0.0       1.0  0.0
##       0.0       0.0       0.0  1.0

A12 = Q1 * A

##     4×4 Array{Float64,2}:
##      0.834865      0.96279    0.720955   0.840615
##      1.38778e-17  -0.901463  -0.89751   -0.331311
##      0.63          0.8        0.75       0.23
##      0.16          0.95       0.73       0.07

Q2 = givensmat(n, A12[1,1], A12[3,1], 1, 3)

##     4×4 Array{Float64,2}:
##       0.798229  0.0  0.602354  0.0
##       0.0       1.0  0.0       0.0
##      -0.602354  0.0  0.798229  0.0
##       0.0       0.0  0.0       1.0

A13 = Q2 * A12

##     4×4 Array{Float64,2}:
##      1.0459        1.25041     1.02725    0.809545
##      1.38778e-17  -0.901463   -0.89751   -0.331311
##      0.0           0.0586429   0.164402  -0.322755
##      0.16          0.95        0.73       0.07

Q3 = givensmat(n, A13[1,1], A[4,1], 1, 4)

##     4×4 Array{Float64,2}:
##       0.9885   0.0  0.0  0.15122
```

```
##       0.0      1.0  0.0  0.0
##       0.0      0.0  1.0  0.0
##      -0.15122  0.0  0.0  0.9885

A14 = Q3 * A13

##     4×4 Array{Float64,2}:
##       1.05806       1.37969     1.12583    0.81082
##       1.38778e-17  -0.901463   -0.89751   -0.331311
##       0.0           0.0586429   0.164402  -0.322755
##      -2.77556e-17   0.749989    0.566264  -0.0532239

Q4 = givensmat(n, A14[2,2], A14[3,2], 2, 3)

##     4×4 Array{Float64,2}:
##      1.0   0.0         0.0         0.0
##      0.0  -0.997891    0.0649159   0.0
##      0.0  -0.0649159  -0.997891    0.0
##      0.0   0.0         0.0         1.0

A23 = Q4 * A14

##     4×4 Array{Float64,2}:
##       1.05806       1.37969       1.12583    0.81082
##      -1.38485e-17   0.903368      0.906289   0.30966
##      -9.00889e-19  -6.93889e-18  -0.105793   0.343581
##      -2.77556e-17   0.749989      0.566264  -0.0532239

Q5 = givensmat(n, A23[2,2], A23[4,2], 2, 4);
A24 = Q5 * A23

##     4×4 Array{Float64,2}:
##       1.05806       1.37969       1.12583    0.81082
```

```
##      -2.83844e-17   1.17412       1.05901    0.204255
##      -9.00889e-19  -6.93889e-18  -0.105793   0.343581
##      -1.25092e-17   0.0          -0.143223  -0.238751

Q6 = givensmat(n, A24[3,3], A24[4,3], 3, 4);
A34 = Q6 * A24

##     4×4 Array{Float64,2}:
##       1.05806       1.37969      1.12583    0.81082
##      -2.83844e-17   1.17412      1.05901    0.204255
##       1.05971e-17   4.1227e-18   0.178059  -0.012095
##       6.70761e-18  -5.58136e-18  0.0        0.418215
```

到 A34 已经变成了上三角矩阵。各个 Givens 旋转合并起来也是一个正交阵：

```
Q = Q6*Q5*Q4*Q3*Q2*Q1

##     4×4 Array{Float64,2}:
##      0.085061   0.784451   0.595427    0.15122
##      0.751748  -0.189333  -0.0183152   0.631421
##      0.43877   -0.351868   0.556258   -0.611757
##      0.484892   0.474318  -0.579403   -0.451877

Q'*Q

##     4×4 Array{Float64,2}:
##       1.0          -5.55112e-17  -5.55112e-17  0.0
##      -5.55112e-17   1.0          -2.22045e-16  0.0
##      -5.55112e-17  -2.22045e-16   1.0          0.0
##       0.0           0.0           0.0          1.0

B = Q * A

##     4×4 Array{Float64,2}:
```

```
##       1.05806      1.37969      1.12583       0.81082
##       0.0          1.17412      1.05901       0.204255
##       4.16334e-17  0.0          0.178059     -0.012095
##      -4.16334e-17  5.55112e-17  1.11022e-16   0.418215

maximum(B - A34)

##     2.220446049250313e-16
```

上面相当于 $QA = R$，所以 $A = Q'R$。Julia 的 `qr()` 函数可以计算 QR 分解：

```
Q1, R1 = qr(A);
R1

##     4×4 Array{Float64,2}:
##      -1.05806  -1.37969  -1.12583   -0.81082
##       0.0       1.17412   1.05901    0.204255
##       0.0       0.0      -0.178059   0.012095
##       0.0       0.0       0.0       -0.418215
```

`qr()` 函数给出的上三角矩阵不满足对角线元素为正的要求。需要时可以将负的主对角元素所在行反号并将对应的 Q 的列反号。如：

```
R2 = R1; Q2 = Q1
R2[1,:] = -R2[1,:]; Q2[:,1] = -Q2[:,1]
R2[3,:] = -R2[3,:]; Q2[:,3] = -Q2[:,3]
R2[4,:] = -R2[4,:]; Q2[:,4] = -Q2[:,4]
R2

##     4×4 Array{Float64,2}:
##       1.05806   1.37969   1.12583    0.81082
##       0.0       1.17412   1.05901    0.204255
##      -0.0      -0.0       0.178059  -0.012095
##      -0.0      -0.0      -0.0        0.418215
```

```
Q2
```

```
##     4×4 Array{Float64,2}:
##      0.085061   0.751748    0.43877    0.484892
##      0.784451  -0.189333   -0.351868   0.474318
##      0.595427  -0.0183152   0.556258  -0.579403
##      0.15122    0.631421   -0.611757  -0.451877
```

```
Q'
```

```
##     4×4 Array{Float64,2}:
##      0.085061   0.751748    0.43877    0.484892
##      0.784451  -0.189333   -0.351868   0.474318
##      0.595427  -0.0183152   0.556258  -0.579403
##      0.15122    0.631421   -0.611757  -0.451877
```

Part VI

最优化与方程求根

Chapter 35

最优化问题

在统计计算中广泛用到求最小（最大）值点或求方程的根的算法，比如，参数最大似然估计、置信区间的统计量法、回归分析参数估计、惩罚似然估计、惩罚最小二乘估计，等等。求最小值点或最大值点的问题称为**最优化**问题（或称优化问题），最优化问题和求方程的根经常具有类似的算法，所以在这一章一起讨论。有些情况下，可以得到解的解析表达式；更多的情况只能通过数值迭代算法求解。

本章将叙述最小值点存在性的一些结论，并给出一些常用的数值算法。还有很多的优化问题需要更专门化的算法。关于最优化问题的详细理论以及更多的算法，读者可以参考这方面的教材和专著，如 徐成贤 et al. [2002], 高立 [2014], Lange [2013] 。

因为求最大值的问题和求最小值的问题完全类似，而且最小值问题涉及到凸函数、正定海色阵等，比最大值问题方便讨论，所以我们只考虑最小值问题。

35.1 优化问题的类型

设 $f(x)$ 是 $\mathbb{R}^d$ 上的多元函数，如果 $f(x)$ 有一阶偏导数，则记 $f(x)$ 的梯度向量为

$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \cdots, \frac{\partial f(x)}{\partial x_d} \right)^T,$$

有的教材记梯度向量为 $g(x)$。如果 $f(x)$ 的二阶偏导数都存在，则记它的海色阵为

$$\nabla^2 f(x) = \left(\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \right)_{\substack{i=1,\dots,d \\ j=1,\dots,d}},$$

有的教材记海色阵为 $H_f(x)$ 或 $H(x)$。当所有二阶偏导数连续时，海色阵是对称阵。

定义 35.1 (无约束最优化)**.** 设 $f(x), x = (x_1, \dots, x_d)^T \in \mathbb{R}^d$ 是实数值的 d 元函数，求

$$\operatorname*{argmin}_{x \in \mathbb{R}^d} f(x)$$

的问题称为（全局的）**无约束最优化**问题。

这里，符号 argmin 表示求最小值点的运算, 称 $f(x)$ 为**目标函数**，自变量 x 称为决策变量。x^* 是问题的解等价于

$$f(x) \geq f(x^*),\ \forall x \in \mathbb{R}^d,$$

称这样的 x^* 为 $f(x)$ 的一个**全局最小值点**。全局最小值点不一定存在，存在时不一定唯一。如果

$$f(x) > f(x^*),\ \forall x \in \mathbb{R}^d \backslash \{x^*\},$$

则称 x^* 为 $f(x)$ 的一个**全局严格最小值点**, 全局严格最小值点如果存在一定只有一个。上式中 "\" 表示集合的差。

对 $x \in \mathbb{R}^d$ 和 $\delta > 0$，定义

$$U(x, \delta) = \{y \in \mathbb{R}^d : \|y - x\| < \delta\},$$

称为 x 的**邻域**或开邻域，定义

$$U_0(x, \delta) = \{y \in \mathbb{R}^d : \|y - x\| < \delta, y \neq x\},$$

为 x 的**空心开邻域**。

如果存在 x^* 的开邻域 $U(x, \delta)$ 使得

$$f(x) \geq f(x^*),\ \forall x \in U(x, \delta),$$

称 x^* 是 $f(x)$ 的一个**局部极小值点**；如果存在 x^* 的空心开邻域 $U_0(x,\delta)$ 使得

$$f(x) > f(x^*),\ \forall x \in U_0(x,\delta),$$

称 x^* 是 $f(x)$ 的一个**局部严格极小值点**。

定义 35.2 (约束最优化)**.** 设 $c_i(\cdot),\ i=1,\dots,p+q$ 是 d 元实值函数, 求

$$\begin{cases} \underset{x\in\mathbb{R}^d}{\operatorname{argmin}} f(x),\ \text{s.t.} \\ c_i(x)=0,\ i=1,\dots,p \\ c_i(x)\le 0,\ i=p+1,\dots,p+q \end{cases} \tag{35.1}$$

的问题称为（全局的）**约束最优化**问题, 这里 s.t.(subject to) 是 "满足如下约束条件" 的意思, $c_i, i=1,\dots,p$ 称为**等式约束**，$c_i, i=p+1,\dots,p+q$ 称为**不等式约束**。满足约束的 $x\in\mathbb{R}^d$ 称为一个**可行点**, 所有可行点的集合 D 称为**可行域**:

$$D=\{x\in\mathbb{R}^d:\ c_i(x)=0, i=1,\dots,p;\ c_i(x)\le 0, i=p+1,\dots,p+q\}.$$

最优化问题(35.1)也可以写成 $\min\limits_{x\in D} f(x)$。若 $x^*\in D$ 使得

$$f(x)\ge f(x^*),\ \forall x\in D,$$

称 x^* 为约束优化问题(35.1)的一个**全局最小值点**。如果 $x^*\in D$ 使得

$$f(x) > f(x^*),\ \forall x\in D\backslash\{x^*\},$$

称 x^* 为约束优化问题(35.1)的一个**全局严格最小值点**。

如果 $x^*\in D$ 且存在 x^* 的开邻域 $U(x,\delta)$ 使得

$$f(x)\ge f(x^*),\ \forall x\in U(x,\delta)\cap D,$$

称 x^* 是约束优化问题(35.1)的一个**局部极小值点**；如果 $x^*\in D$ 且存在 x^* 的空心开邻域 $U_0(x,\delta)$ 使得

$$f(x) > f(x^*),\ \forall x\in U_0(x,\delta)\cap D,$$

称 x^* 是约束优化问题(35.1)的一个**局部严格极小值点**。

最优化问题经常使用数值迭代方法求解，数值迭代方法大多要求每一步得到比上一步更小的目标函数值，这样最后迭代结束时往往得到的不是全局的最小值点，而是局部的极小值点。要注意的是，全局的最小值点一定也是局部极小值点, 另外，极小值点有时不存在，存在时可以不唯一。本书中的最优化算法一般只能收敛到局部极小值点。

因为等式约束 $c_i(x)=0$ 可以写成两个不等式约束：

$$c_i(x)\le 0,\quad -c_i(x)\le 0,$$

所以约束优化问题(35.1)有时写成如下的简化形式：

$$\begin{cases} \underset{x\in\mathbb{R}^d}{\operatorname{argmin}} f(x),\ \text{s.t.} \\ c_i(x)\le 0,\ i=1,\dots,p \end{cases}, \tag{35.2}$$

可行域为

$$D=\{x\in\mathbb{R}^d:\ c_i(x)\le 0, i=1,\dots,p\}.$$

例 35.1. 函数 $f(x)=\frac{1}{1+x^2}, x\in\mathbb{R}$ 没有极小值点，$f(x)>0, \lim_{x\to\pm\infty} f(x)=0$。

例 35.2. 函数 $f(x)=\max(|x|,1)$ 最小值为 1, 且当 $x\in[-1,1]$ 时都取到最小值。

在最优化问题中如果要求解 x 的所有或一部分分量取整数值，这样的问题称为**整数规划**问题。如果要同时考虑具有相同自变量的多个目标函数的优化问题，则将其称为**多目标规划**问题。对约束最优化问题(35.1)，如果 $f(x), c_i(x)$ 都是线性函数，称这样的问题为**线性规划**问题。线性规划是运筹学的一个重要工具，在工业生产、经济计划等领域有广泛的应用。如果(35.1)中的 $f(x), c_i(x)$ 不都是线性函数，称这样的问题为**非线性规划问题**。对非线性规划问题，如果 $f(x)$ 是二次多项式函数，$c_i(x)$ 都是线性函数，称这样的问题为**二次规划**问题。本书主要讨论在统计学中常见的优化问题，不讨论线性规划、整数规划、多目标规划等问题。

35.2 统计计算与最优化

最优化是统计计算中最常用的工具之一，另外常用的是随机模拟和积分，矩阵计算也是统计计算中最基本的组成部分。

许多统计方法的计算都归结为一个最优化问题。比如，在估计问题中，最大似然估计和最小二乘估计都是最优化问题。在试验或者抽样调查方案的设计时，也需要用最优化方法制作误差最小的方案。在现代的深度神经网络等机器学习方法中，也要用最优化方法训练模型。

35.2.1 回归模型的估计与最优化

最常见的统计模型是回归问题

$$y = f(x;\theta) + \varepsilon, \tag{35.3}$$

其中 θ 是未知的不可观测但非随机的**模型参数**，ε 是随机误差，模型拟合归结到估计 θ 的问题。

函数 f 最简单的是线性函数，模型(35.3)变成

$$y = x^T\beta + \varepsilon, \tag{35.4}$$

其中 β 是未知参数，ε 服从正态分布。模型的样本为 $(x_i, y_i), i = 1, 2, \ldots, n$, 在估计模型参数 β 时，认为 β 是客观存在的未知、非随机值，先用一个变量 b 代替 β，定义拟合误差

$$r_i(b) = y_i - x_i^T b,$$

按照某种使得拟合误差最小的准则求一个 b 作为真实参数 β 的估计值。注意拟合误差 $r_i(b)$ 是样本与变量 b 的函数，这既不是随机误差 $\varepsilon_i = y_i - x_i^T\beta$，也不是 ε_i 的观测值（ε_i 是不可观测的）。

因为拟合误差有 n 个，需要定义某种规则才能同时最小化 n 个拟合误差。一个自然的考虑是最小化各个拟合误差绝对值的总和：

$$R_1(b) = \sum_{i=1}^{n} |y_i - x_i^T b|,$$

但是这个函数关于 b 不可微，最优化的数值算法也比较复杂，所以通常考虑最小化拟合误差的平方和：

$$R_2(b) = \sum_{i=1}^{n} |y_i - x_i^T b|^2 = \sum_{i=1}^{n} \left(y_i - x_i^T b\right)^2,$$

此函数是 b 的二次多项式函数，必存在最小值点且适当条件下最小值点有显式解。最小二乘估计就是如下的无约束最优化问题

$$\min_b \sum_{i=1}^{n} \left(y_i - x_i^T b\right)^2.$$

最小二乘问题也可能会是带有约束的，比如在线性回归中要求 β 中的各个回归系数非负，记作 $\beta \succeq 0$，估计 β 就变成了约束最小二乘问题

$$\min_{b \succeq 0} \sum_{i=1}^{n} \left(y_i - x^T b\right)^2.$$

约束优化问题比无约束优化问题困难得多，也不存在显式解。

回归模型也可以是非线性的，这时最小二乘问题为

$$\min_t \sum_{i=1}^{n} \left(y_i - f(x_i; t)\right)^2,$$

此问题比线性模型的最小二乘问题困难，一般也没有解的表达式。

在 $R_1(\cdot)$ 中用 $|r_i(b)|$ 度量误差大小，在 $R_2(\cdot)$ 中用 $r_i^2(b)$ 度量误差大小，也可以将误差大小的度量推广为一般的 $\rho(r_i(b))$，其中 $\rho(z)$ 随 $|z|$ 增大而增大。模型估计转换为如下的最优化问题

$$\min_t \sum_{i=1}^{n} \rho(y_i - f(x_i; t)), \tag{35.5}$$

这一般比最小二乘在理论上和计算上都复杂得多。

不同的观测可能具有不同的误差大小，在最小化拟合误差时可以给不同的观测加不同的权重，误差大的观测权重小，估计问题变成如下最优化问题：

$$\min_t \sum_{i=1}^{n} w(y_i, x_i, t)\rho(y_i - f(x_i; t)), \tag{35.6}$$

其中 $w(y_i, x_i, t)$ 是第 i 个观测的权重，经常不直接依赖于观测值和参数变量，所以往往写成 w_i。例如，线性模型的加权最小二乘估计：

$$\min_b \sum_{i=1}^n w_i \left(y_i - x_i^T b\right)^2,$$

这个问题也有显式解。

在现代的对回归问题的研究中，又提出了对估计施加某种惩罚的做法，目的是克服过拟合问题。估计问题变成如下最优化问题：

$$\min_t \sum_{i=1}^n w(y_i, x_i, t)\rho(y_i - f(x_i; t)) + \lambda g(t), \tag{35.7}$$

其中 $g(\cdot)$ 是对估计参数的复杂程度的一种取非负值的惩罚函数，$\lambda > 0$ 是调整估计的模型的复杂程度的一个调节因子。例如，岭回归问题对应于如下的最优化问题：

$$\min_b \left\{ \sum_{i=1}^n \left(y_i - x_i^T b\right)^2 + \lambda b^T b \right\},$$

此惩罚倾向于产生分量的绝对值较小的估计 b。Lasso 回归实际是如下的最优化问题：

$$\min_b \left\{ \sum_{i=1}^n \left(y_i - x_i^T b\right)^2 + \lambda \|b\|_1 \right\},$$

其中 $\|b\|_1 = \sum_{j=1}^p |b_j|$。此惩罚倾向于产生分量的绝对值较小的估计 b，而且 λ 较大时会使得参数估计 b 的部分分量退化到 0。

模型(35.3)中的 f 是形式已知，参数 θ 未知的。还可以认为 $f(\cdot)$ 是未知的，直接用一个函数 $\tilde{f}(\cdot)$ 来近似 f。这称为非参数回归。但是，如果仅要求 $y_i - \tilde{f}(x)$ 小，有许多函数可以使得这些拟合误差完全等于零（当相同的 x 都对应相同的 y）时。所以，必须对 $\tilde{f}$ 施加一定的限制，称为正则化 (regularization)。比如，要求 $\tilde{f}$ 二阶连续可微且比较 “光滑”，最优化问题变成

$$\min_{\tilde{f} \text{ 二阶连续可微}} \sum_{i=1}^n (y_i - \tilde{f}(x_i))^2 + \lambda \int [\tilde{f}''(x)]^2 \, dx.$$

$\lambda > 0$ 是光滑度的调节参数，λ 越大，最后得到的回归函数估计 $\tilde{f}$ 越光滑。

35.2.2 最大似然估计与最优化

参数估计问题中，最大似然估计是将观测的联合密度函数或者联合概率函数看成是未知参数的函数，称为**似然函数**, 通过最大化似然函数来估计参数。

在(35.3)的估计中，除了可以最小化残差的方法，还可以用最大似然估计方法。设 ε 有密度函数 $p(\cdot)$, 且各个观测相互独立，则参数 θ 的最大似然估计问题是

$$\max_{t} \prod_{i=1}^{n} p(y_i - f(x_i; t)).$$

但这样的优化问题可能很难计算求解。对某些密度 $p(\cdot)$, 最大似然估计与最小化拟合残差是等价的。

如果 $f(\cdot)$ 是非参数的未知回归函数，也可以用最大似然估计求解，但仍需要施加正则化，如

$$\max_{f \text{ 二阶连续可微}} \prod_{i=1}^{n} p(y_i - f(x_i; t)) \ \exp\left\{-\lambda \int [f''(x)]^2 \, dx\right\}.$$

35.2.3 将统计问题表述为最优化问题的缺点

许多统计问题的解归结为一个最优化问题。但是，产生的最优化问题会有解的存在性、唯一性、局部最优等问题。要解决的统计问题与要解决的最优化问题应该是一致的，不能因为可以求解的最优化问题的限制而修改原来的统计问题。由于约束优化问题的特点，最优值常常在边界处达到，所以用优化问题求解统计模型对模型的假定更为敏感，数据不满足某些模型假定时用最优化方法解出的结果可能不能很好地解释数据。

许多统计问题需要同时对多个目标优化，或在某些约束下优化，这就需要适当地设计策略，不同策略产生不同的统计方法。最简单的做法是对各个目标或约束的加权和进行优化。

35.3 一元函数的极值

先复习一些数学分析中的结论。设 $f(x)$ 为定义在闭区间 $[a, b]$ 上的连续函数，则 $f(x)$ 在 $[a, b]$ 内有界，且能达到最小值和最大值, 极值可以在区间内部或边界取到。设 $f(x)$ 在区间 (a, b)（有限或无限区间）可微。

定理 35.1 (一阶必要条件)**.** *若 $x^* \in (a, b)$ 是 $f(x)$ 的一个局部极小值点，则 $f'(x^*) = 0$。*

这只是必要条件，不是充分条件。局部极大值点也满足同样的条件。满足 $f'(x^*)=0$ 的 x^* 叫做 $f(x)$ 的一个**稳定点**，它可能是局部极小值点、局部极大值点，也可能不是局部极值点。证明略。

定理 35.2 (二阶必要条件)**.** *若 $f(x)$ 在 (a,b) 内有二阶导数，$x^*\in(a,b)$ 是 $f(x)$ 的一个局部极小值点，则 $f'(x^*)=0$ 且 $f''(x^*)\geq 0$。*

这可以用下面的二阶充分条件来证明。对局部极大值点，条件 $f''(x^*)\geq 0$ 变成了 $f''(x^*)\leq 0$。

定理 35.3 (二阶充分条件)**.** *若 $f(x)$ 在 (a,b) 内有二阶导数，对 $x^*\in(a,b)$ 同时有 $f'(x^*)=0$ 且 $f''(x^*)>0$，则 x^* 是 $f(x)$ 的一个局部极小值点。*

若 $f'(x^*)=0$ 且 $f''(x^*)<0$ 则 x^* 是局部极大值点。当 $f''(x^*)=0$ 时，不能确定 x^* 是否极值点。

当一元连续函数 $f(x)$ 的定义域为区间但不是闭区间时，$f(x)$ 在定义域内不一定取到最小值，通常可以通过检查边界处的极限并与内部的所有极小值的比较来确定。

例 35.3. 对 $f_1(x)=x^2, x\in\mathbb{R}$, $x^*=0$ 使得 $f_1'(x^*)=2x^*=0$, $f_1''(x^*)=2>0$，是一个局部极小值点。因为 $x^*=0$ 是唯一的极小值点，当 $x\to\pm\infty$ 时函数值增大，所以这个局部极小值点也是全局最小值点。

对 $f_2(x)=x^3$, $x^*=0$ 使得 $f_2'(x^*)=3(x^*)^2=0$, 是一个稳定点。但是 $f_2''(x^*)=6x^*=0$，不能保证 x^* 是极值点。事实上，$x^*=0$ 不是 $f_2(x)=x^3$ 的极值点。

对 $f_3(x)=\frac{1-x}{1+x^2}, x\geq 0$, 有 $f_3(0)=1$, $\lim_{x\to+\infty}f_3(x)=0$, $f_3'(x)=\frac{x^2-2x-1}{(x^2+1)^2}$, 令 $f_3'(x)=0$ 得稳定点 $x^*=1+\sqrt{2}$，且 $f_3(x^*)=-\sqrt{2}/(4+2\sqrt{2})<0$，所以 x^* 是 $f_3(x)$ 的全局最小值点。

例 35.4. 存在不可微点时极值点也不一定出现在稳定点。例如，设总体 $X\sim \mathrm{U}(0,b)$, 样本 $X_1,\ldots,X_n$, 似然函数为

$$L(b)=\prod_{i=1}^{n}\frac{1}{b}I_{[0,b]}(X_i)=\frac{1}{b^n}I_{[0,b]}(X_{(n)}),$$

其中 $X_{(n)} = \max(X_1, \dots, X_n)$。导数

$$L'(b) = \begin{cases} 0 & b < X_{(n)}, \\ \text{不存在}, & b = X_{(n)}, \\ -nb^{-n-1}, & b > X_{(n)}, \end{cases}$$

$L(b)$ 的最大值点为 $\hat{b} = X_{(n)}$，$L'(\hat{b})$ 不存在。

35.4 凸函数

35.4.1 凸集

定义 35.3 (凸集)**.** 设 S 是 $\mathbb{R}^d$ 的子集，如果 $\forall x, y \in S$, $\alpha \in [0,1]$, 都有 $\alpha x + (1-\alpha)y \in S$，则称 S 为**凸集**。

等价地，若对任意正整数 $n \geq 2$ 和 S 中的 n 个互不相同的点 $x^{(j)}, j = 1, 2, \dots, n$，以及任意的 $\{\alpha_j, j = 1, \dots, n\}$ 满足 $\alpha_j \geq 0, \sum_{j=1}^n \alpha_j = 1$ 都有 $\sum_{j=1}^n \alpha_j x^{(j)} \in S$，则 S 是凸集。

S 是凸集当且仅当 S 中任意两个点的连线都属于 S。例如，实数轴上的凸集和区间是等价的; 平面上边界为椭圆、三角形、平行四边形的集合是凸集, 圆环则不是凸集; 球体、长方体是凸集。

定义 35.4 (凸壳)**.** 设 A 是 $\mathbb{R}^d$ 的子集，称包含 A 的最小的凸集 S 为 A 的**凸壳**。

A 的凸壳的等价定义为

$$\text{conv}(A) = \left\{ \sum_{i=1}^m \alpha_i x_i : m = 1, 2, \dots; \right. \\ \left. x_i \in \mathbb{R}^d, \alpha_i \geq 0, i = 1, 2, \dots, m; \sum_{i=1}^m \alpha_i = 1 \right\}$$

凸集有如下性质:

(1) 凸集的闭包是凸集。

(2) 凸集的内核（所有内点组成的集合）是凸集。
(3) 凸集是连通集合。
(4) 任意多个凸集的交集是凸集。
(5) 关于仿射变换 $f(x) = Ax + b(A$ 为矩阵, b 为常数向量)，凸集的像和原像还是凸集。
(6) 两个凸集 S, T 的笛卡尔积 $S \times T = \{(x, y) : x \in S, y \in T\}$ 是凸集。
(7) 设 S 是 $\mathbb{R}^d$ 中的凸集，λ 为实数，则 $\lambda S \triangleq \{y \in \mathbb{R}^d : y = \lambda x, x \in S\}$ 是凸集。
(8) 设 S, T 是 $\mathbb{R}^d$ 中的两个凸集，则 $S+T \triangleq \{z \in \mathbb{R}^d : z = x+y, x \in S, y \in T\}$ 是凸集。
(9) 对 $\mathbb{R}^d$ 的凸集 S 以及任意 $y \in \mathbb{R}^d$, 令 y 到 S 的距离为 $\mathrm{dist}(y, S) \triangleq \inf_{x \in S} \|y - x\|$, 则至多有一个 $x_0 \in S$ 可以满足 $\|y - x_0\| = \mathrm{dist}(y, S)$。当 S 是闭的凸集时，存在唯一的 $x_0 \in S$ 使得 $\|y - x_0\| = \mathrm{dist}(y, S)$。
(10) 设 S 为凸集，x_0 是 S 的边界点，则存在单位向量 $v \in \mathbb{R}^d$ 使得 $v^T x_0 \geq v^T x, \forall x \in S$。

35.4.2 凸函数

定义 35.5 (凸函数)**.** 定义在凸集 S 上的 d 元函数 $f(x)$ 称为**凸函数**, 如果对任意 $x, y \in S$ 和 $\alpha \in [0, 1]$ 都有

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y), \tag{35.8}$$

如果(35.8)对任意 $x \neq y \in S$ 和 $\alpha \in (0, 1)$ 都满足严格不等式，则称 $f(x)$ 为**严格凸函数**。

若 $-f(x)$ 是定义在凸集 S 上的凸函数，则称 $f(x)$ 为**凹函数**。

对凸函数 $f(x)$，设 $x^{(j)}, j = 1, \ldots, n$ 是 S 中的 n 个点，$\alpha_j \geq 0, j = 1, \ldots, n$, $\sum_{j=1}^n \alpha_j = 1$，则

$$f\left(\sum_{j=1}^n \alpha_j x^{(j)}\right) \leq \sum_{j=1}^n \alpha_j f(x^{(j)}).$$

设 $f(x)$ 是正值函数，如果 $\log f(x)$ 是凸函数，则称 $f(x)$ 为**对数凸函数**。

例 35.5. 所有的范数（模）都是凸函数。

事实上，

$$\|\alpha x+(1-\alpha)y\| \leq \|\alpha x\|+\|(1-\alpha)y\| = \alpha\|x\|+(1-\alpha)\|y\|.$$

下面给出凸函数的另外一些性质。

(1) 设 S 是 $\mathbb{R}^d$ 的凸集，$f(x), x \in S$ 是凸函数当且仅当 $\{(x,y):\ f(x) \leq y,\ x \in S, y \in \mathbb{R}\}$ 是凸集。

(2) 若 $f(x)$ 是定义在 $\mathbb{R}^d$ 的开凸集 S 上的可微函数，则 $f(x)$ 是凸函数当且仅当

$$f(y)-f(x) \geq \nabla f(x)^T\ (y-x),\ \forall x, y \in S.$$

(3) 若 $f(x)$ 是 $\mathbb{R}^d$ 上的凸函数，$c \in \mathbb{R}$, 则 $\{x \in \mathbb{R}^d:\ f(x) \leq c\}$ 和 $\{x \in \mathbb{R}^d:\ f(x) < c\}$ 都是凸集。

(4) 设 $f(x)$ 为定义在某区间 S 上的二阶可微的一元实函数，如果 $f(x)$ 满足 $f''(x) \geq 0, \forall x \in S$，则 $f(x)$ 在 S 上为凸函数；如果 $f(x)$ 满足 $f''(x) > 0, \forall x \in S$，则 $f(x)$ 在 S 上为严格凸函数。

(5) 设 $f(x)$ 是定义在开凸集 $S \subset \mathbb{R}^d$ 上的二阶可微函数，若 $\nabla^2 f(x), x \in S$ 都是非负定阵，则 $f(x)$ 为凸函数；若 $\nabla^2 f(x), x \in S$ 都是正定阵，则 $f(x)$ 为严格凸函数。

(6) 设 $f(x)$ 是定义在凸集 $S \subset \mathbb{R}^d$ 上的凸函数，则 $f(x)$ 在 S 的内核 $\mathring{S}$ 上连续且满足局部 Lipschitz 条件，即 $\forall x \in \mathring{S}$, 存在常数 c 使得对 x 的一个邻域内的 y, z 都有 $|f(y)-f(z)| \leq c\|y-z\|$。

(7) 设 $f(x,y)$ 是凸集 $S \subset X \times Y$ 上的凸函数，集合 $B \subset Y$ 是一个凸集，集合 $A = \{x \in X:\ \exists y \in B$使得$(x,y) \in S\}$ 为非空集，则 A 是凸集，$g(x) \triangleq \inf_{y \in B} f(x,y)$ 是 A 上的凸函数。

(8) 对随机变量 X，若 $f(x)$ 是凸函数，EX 和 $Ef(X)$ 存在，则 $f(EX) \leq Ef(X)$。这称为 Jenson 不等式。

(9) 设 $f(x)$ 为凸函数，$g(t), t \in \mathbb{R}$ 为单调增的凸函数，则 $g(f(x))$ 为凸函数。

(10) 设 A 为矩阵，b 为向量，$f(x)$ 为凸函数，则 $f(Ax+b)$ 为凸函数。

(11) 如果 $f(x), g(x)$ 为凸函数，$\alpha \geq 0, \beta \geq 0$, 则 $\alpha f(x)+\beta g(x)$ 为凸函数。

(12) 如果 $f(x), g(x)$ 为凸函数，则 $\max(f(x), g(x))$ 为凸函数。

(13) 若 $f(x,y)$ 定义于 $x \in A, y \in B$，A 是 $\mathbb{R}^d$ 中的凸集，B 是某足标集，若对任意 $y \in B$，$f(x,y)$ 关于 $x \in A$ 是凸函数，则 $g(x) \triangleq \sup_{y \in B} f(x,y)$

是 A 上的凸函数；若对任意 $y \in B$，$f(x,y)$ 关于 $x \in A$ 是凹函数，则 $g(x) \triangleq \inf_{y\in B} f(x,y)$ 是 A 上的凹函数。

(14) 设 $f_n(x), n = 1,2,...$ 都是凸函数且 $\lim_{n\to\infty} f_n(x)$ 存在，则 $f(x) \triangleq \lim_{n\to\infty} f_n(x)$ 是凸函数。

(15) 对数凸函数一定也是凸函数。

(16) 设 $f(x)$ 为凸函数，$g(t), t \in \mathbb{R}$ 为单调增的对数凸函数，则 $g(f(x))$ 是对数凸函数。

(17) 若 $f(x)$ 是对数凸函数，则 $f(Ax+b)$ 也是对数凸函数。

(18) 设 $f(x)$ 为对数凸函数，$\alpha > 0$，则 $f(x)^\alpha$ 和 $\alpha f(x)$ 为对数凸函数。

(19) 设 $f(x), g(x)$ 为对数凸函数,则 $f(x)+g(x)$, $f(x)g(x)$ 和 $\max(f(x),\ g(x))$ 都是对数凸函数。

(20) 设 $f_n(x), n = 1,2,...$ 都是对数凸函数且 $\lim_{n\to\infty} f_n(x)$ 存在，则 $f(x) \triangleq \lim_{n\to\infty} f_n(x)$ 是对数凸函数。

例 35.6. 设 $x \in \mathbb{R}^d$, S 为 $\mathbb{R}^d$ 的凸集，则 $f(x) = \mathrm{dist}(y, S)$ 是凸函数。

利用性质 7 可得此结论。

例 35.7. $|x|$ 是凸函数。可以直接证明，或利用 $|x| = \max(x,-x)$ 和性质 12。

$x^+ = \max(x, 0)$ 是凸函数。

x^2 是凸函数。$\frac{d^2}{dx^2}x^2 = 2 > 0$。

例 35.8. $f(x) = |x|^\gamma$ 是凸函数 $(\gamma \geq 1)$。首先, $g(x) = x^\gamma, x \in [0,\infty)$ 满足 $g'(x) = \gamma x^{\gamma-1} \geq 0$, $g''(x) = \gamma(\gamma-1)x^{\gamma-2} \geq 0$，所以 $g(x)$ 在 $[0,\infty)$ 上是单调增凸函数，而 $|x|$ 是凸函数，所以 $f(x) = g(|x|)$ 是凸函数。

例 35.9. 对线性函数 $f(x) = a^Tx + b, x \in \mathbb{R}^n$，其中 $a \in \mathbb{R}^d, b \in \mathbb{R}$, 有 $\nabla f(x) = a$, $\nabla^2 f(x) = 0$，$f(x)$ 是凸函数，且(35.8)中的等式成立。

例 35.10. 对 $f(x) = \frac{1}{2}x^TAx + b^Tx + c, x \in \mathbb{R}^n$, 其中 A 为非负定阵，$b \in \mathbb{R}^d, c \in \mathbb{R}$, 有 $\nabla f(x) = Ax + b$, $\nabla^2 f(x) = A$, 所以 $f(x)$ 是凸函数。当 A 是正定阵时，$f(x)$ 是严格凸函数。

例 35.11. 欧式模 $f(x) = \|x\| = \sqrt{x^Tx}$ 是凸函数。

事实上，当 $x \neq 0$ 时 $\nabla f(x) = \|x\|^{-1}x$, $\nabla^2 f(x) = \|x\|^{-3}\left(\|x\|^2 I_d - xx^T\right)$, 其中 I_d 为 d 阶单位阵。易见 $\nabla^2 f(x)$ 是非负定阵，但不是正定阵。进一步地，所有的范数（模）都是凸函数，见例35.5。

例 35.12. 设 $z_j, j = 1, \dots, n$ 是 $\mathbb{R}^d$ 的 n 个点。定义

$$f(x) = \log \left[\sum_{j=1}^{n} \exp(z_j^T x) \right],$$

由于 $z_j^T x$ 是凸函数，所以 $\exp(z_j^T x)$ 是对数凸函数，由性质 19 可知 $\sum_{j=1}^{n} \exp(z_j^T x)$ 是对数凸函数，即 $f(x)$ 是凸函数。

35.4.3 凸规划

凸函数在最优化问题中有重要作用，原因是, 若 $f(x)$ 是凸集 $S \subset \mathbb{R}^d$ 上的凸函数, 那么：

(1) 当 $f(x)$ 有一个局部极小值点 x^* 时，这个极小值点也是全局最小值点，但不一定是唯一的全局最小值点，这时集合 $\{x \in S : \ f(x) = f(x^*)\}$ 是一个凸集。
(2) 如果 x^* 是 $f(x)$ 的一个稳定点，则 x^* 是 $f(x)$ 的一个全局最小值点。
(3) 如果 $f(x)$ 是严格凸函数而且全局最小值点存在，这个全局最小值点是唯一的。

对于约束优化问题(35.1), 如果等式约束 $c_i, i = 1, \dots, p$ 都是线性函数，不等式约束 $c_i, i = p+1, \dots, p+q$ 都是凸函数，目标函数 $f(x)$ 是凸函数，则称(35.1)为**凸规划问题**或凸优化问题。凸规划问题的可行域 D 是凸集 (因为凸集的交集是凸集和凸函数性质 4)，其局部极小值点一定是全局最小值点。

35.5 无约束极值点的条件

设 $f(x)$ 的定义域 $\mathcal{A}$ 为 $\mathbb{R}^d$ 中的开集。如果 x^* 使得 $\nabla f(x^*) = 0$，称 x^* 为 $f(\cdot)$ 的一个**稳定点**。无约束极值点的必要条件和充分条件是数学分析中熟知的结论，这里仅罗列这些结论备查。

定理 35.4 (一阶必要条件)**.** *如果 $f(x), x \in \mathcal{A}$ 有一阶连续偏导数，$x^* \in \mathcal{A}$ 是 $f(\cdot)$ 的一个局部极小值点，则 x^* 是 $f(\cdot)$ 的稳定点：*

$$\nabla f(x^*) = 0. \tag{35.9}$$

定理 35.5 (二阶必要条件). 如果 $f(x), x \in \mathcal{A}$ 有二阶连续偏导数，$x^* \in \mathcal{A}$ 是 $f(\cdot)$ 的一个局部极小值点，则 x^* 是 $f(\cdot)$ 的稳定点，且海色阵 $\nabla^2 f(x^*)$ 为非负定阵。

定理 35.6 (二阶充分条件). 如果 $f(x), x \in \mathcal{A}$ 有二阶连续偏导数，$x^* \in \mathcal{A}$ 是 $f(\cdot)$ 的一个稳定点，且海色阵 $\nabla^2 f(x^*)$ 为正定阵，则 x^* 是 $f(\cdot)$ 的局部严格极小值点。

稳定点不一定是极值点。如果 x^* 是稳定点但是 $\nabla^2 f(x^*)$ 同时有正特征值和负特征值，则 x^* 一定不是 $f(\cdot)$ 的极值点；如果 x^* 是稳定点而 $\nabla^2 f(x^*)$ 非负定但不正定，这时 x^* 可能是 $f(\cdot)$ 的极值点，也可能不是。

当所有 $\nabla^2 f(x)$ 都是非负定（也称半正定）矩阵时，$f(x)$ 是凸函数；当所有 $\nabla^2 f(x)$ 都是正定矩阵时，$f(x)$ 是严格凸函数。由凸函数的性质可得如下结论。

定理 35.7 (全局最小值点的二阶充分条件). 如果 $f(x), x \in \mathcal{A}$ 有二阶连续偏导数，所有 $\nabla^2 f(x)$ 都是非负定阵，$x^* \in \mathcal{A}$ 是 $f(\cdot)$ 的一个稳定点，则 x^* 是 $f(\cdot)$ 的全局最小值点; 如果进一步设 $\nabla^2 f(x^*)$ 是正定阵，则稳定点 x^* 是全局严格最小值点。

定义 35.6 (方向导数). 对向量 $v \in \mathbb{R}^d$, 若 $\|v\| = 1$，定义一元函数

$$h(\alpha) = f(x + \alpha v), \ \alpha \geq 0,$$

如果 $h(\alpha)$ 在 $\alpha = 0$ 的右导数存在，则称其为函数 $f(x)$ 在点 x 沿方向 v 的**方向导数**，记为 $\frac{\partial f(x)}{\partial v}$；类似可定义二阶方向导数 $\frac{\partial^2 f(x)}{\partial v^2}$。

如果 $f(x)$ 有一阶连续偏导数，$\|v\| = 1$, 则 $\frac{\partial f(x)}{\partial v} = v^T \nabla f(x)$。如果 $f(x)$ 有二阶连续偏导数，$\|v\| = 1$, 则 $\frac{\partial^2 f(x)}{\partial v^2} = v^T \nabla^2 f(x) v$。

求无约束极值通常使用迭代法。设迭代中得到了一个近似极小值点 $x^{(t)}$，如果 $\nabla f(x) \neq 0$，则从 $x^{(t)}$ 出发沿 $-\nabla f(x)$ 方向前进可以使函数值下降。而且，对任意非零向量 $v \in \mathbb{R}^d$, 只要 $v^T \nabla f(x) < 0$，则 $\frac{\partial f(x)}{\partial v} < 0$, 从 $x^{(t)}$ 出发沿 v 方向前进也可以使函数值下降，称这样的方向 v 为**下降方向**。

例 35.13. 设 Y 为 $n \times 1$ 向量, X 为 $n \times p$ 矩阵 $(n > p)$，β 为 $p \times 1$ 向量。求函数

$$Q(\beta) = \|Y - X\beta\|^2 \tag{35.10}$$

的极小值点的问题称为线性最小二乘问题。

易见

$$\nabla Q(\beta) = -2X^TY + 2X^TX\beta, \tag{35.11}$$

$$\nabla^2 Q(\beta) = 2X^TX, \tag{35.12}$$

当 X 列满秩时 X^TX 为正定矩阵，这时稳定点 $\widehat{\beta} = (X^TX)^{-1}XY$ 是 $Q(\beta)$ 全局严格最小值点。

如果 X 不满秩，因为 X^TX 是非负定矩阵，所以 $Q(\beta)$ 是凸函数，可以证明 $Q(\beta)$ 存在无穷多个稳定点，由定理35.7可知这些稳定点都是全局最小值点。

※※※※※

例 35.14. 求

$$\min f(x) = \frac{1}{4}x_1^4 + \frac{1}{2}x_2^2 - x_1x_2 + x_1 - x_2,\ x \in \mathbb{R}^2.$$

易见

$$\nabla f(x) = \begin{pmatrix} x_1^3 - x_2 + 1 \\ x_2 - x_1 - 1 \end{pmatrix}, \quad \nabla^2 f(x) = \begin{pmatrix} 3x_1^2 & -1 \\ -1 & 1 \end{pmatrix},$$

令 $\nabla f(x) = 0$ 得到三个稳定点 $(-1,0),(1,2),(0,1)$。计算这三个点处的海色阵得

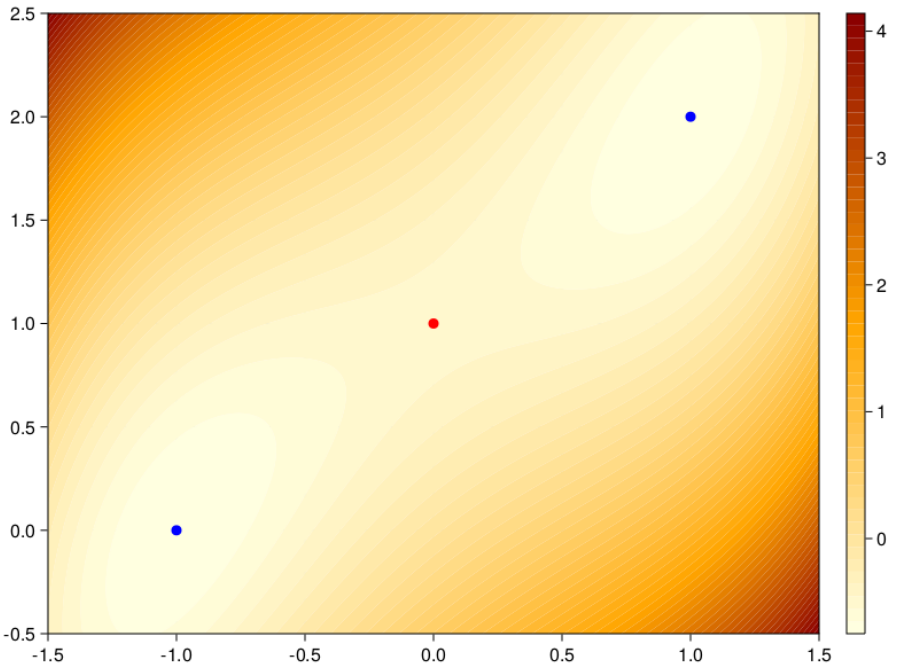
$$\nabla^2 f(-1,0) = \nabla^2 f(1,2) = \begin{pmatrix} 3 & -1 \\ -1 & 1 \end{pmatrix}, \quad \nabla^2 f(0,1) = \begin{pmatrix} 0 & -1 \\ -1 & 1 \end{pmatrix},$$

前两个稳定点海色阵的特征值为 $2 \pm \sqrt{2} > 0$，是正定阵，所以 $(-1,0)$ 和 $(1,2)$ 是 $f(\cdot)$ 的极小值点，$f(-1,0) = f(1,2) = -\frac{3}{4}$，这两个点都是全局最小值点；稳定点 $(0,1)$ 处的海色阵的特征值为 $\frac{1}{2}(1 \pm \sqrt{5})$，一正一负，所以 $(0,1)$ 不是极小值点，且 $f(0,1) = -\frac{1}{2}$。

用 Julia 语言作函数的等高线图，标出三个稳定点：

```
using CairoMakie

function f(x, y)
    1/4*x^4 + 1/2*y^2 - x*y + x - y
end
```

```
xg = -1.5:0.01:1.5
yg = -0.5:0.01:2.5
zg = [f(x, y) for x in xg, y in yg]
fig, ax, plt = contourf(xg, yg, zg, colormap=:heat, levels=50)
scatter!([-1, 1], [0, 2], color=:blue)
scatter!([0], [1], color=:red)
Colorbar(fig[1,2], plt)
fig
```



※※※※※

35.6 约束极值点的条件

与无约束最优化问题不同，约束极值点一般不满足梯度向量为零的条件，这是因为约束极值经常在可行域的边界达到，这时在极值点处如果不考虑约束条件，函数值仍可下降，所以约束极值点处的梯度不需要等于零向量。

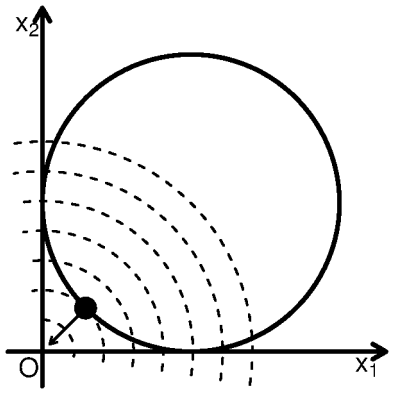


图 35.1: 例35.15图形。虚线为 $f(x)$ 的等值线，箭头是约束最小值点的负梯度方向

例 35.15. 考虑如下约束优化问题

$$\begin{cases} \operatorname{argmin} f(x) = x_1^2 + x_2^2, \text{ s.t.} \\ (x_1-1)^2 + (x_2-1)^2 \le 1, \end{cases} \tag{35.13}$$

可行域 D 是以 $(1,1)^T$ 为圆心的半径等于 1 的圆面。容易看出约束最小值在 $f(x)$ 的等值线与 D 外切的点 $x^* = \left(1-\frac{\sqrt{2}}{2}, 1-\frac{\sqrt{2}}{2}\right)^T$ 处达到 (见图35.1)，这时 $\nabla f(x^*) = (2-\sqrt{2}, 2-\sqrt{2})^T \neq 0$，沿着其反方向 $(-1,-1)^T$ 搜索仍可使函数值下降，但会离开可行域。

在仅有等式约束的情形，如下的拉格朗日乘子法给出了约束极值点的必要条件。

定理 35.8. *考虑约束最优化问题*

$$\begin{cases} \underset{x\in\mathbb{R}^d}{argmin}\, f(x), \ s.t. \\ c_i(x) = 0, \ i = 1, \dots, p \end{cases} \tag{35.14}$$

设 $f(\cdot), c_i(\cdot)$ 在 $\mathbb{R}^d$ 存在一阶连续偏导数，定义拉格朗日函数

$$L(x,\lambda) = f(x) + \sum_{i=1}^{p} \lambda_i c_i(x), \tag{35.15}$$

其中 $\lambda=(\lambda_1,\dots,\lambda_p)^T$，若 x^* 是(35.14)的一个约束局部极小值点，则必存在 λ^* 使得 (x^*,λ^*) 为 $L(x,\lambda)$ 的稳定点，即

$$\nabla f(x^*)+\sum_{i=1}^{p}\lambda_i^*\nabla c_i(x^*)=0, \tag{35.16}$$

$$c_i(x^*)=0,\ i=1,2,\dots,p. \tag{35.17}$$

条件(35.16)说明在约束局部极小值点，目标函数的梯度是各个约束的梯度的线性组合。这个定理将求等式约束的约束优化问题，转化成了关于拉格朗日函数 $L(x,\lambda)$ 的无约束优化问题。

考虑如(35.1)那样的一般约束最优化问题。设可行域 D 为非空集。关于不等式约束 $c_i, i=p+1,\dots,p+q$，对 $x\in D$，如果 $c_i(x)=0$，称 c_i 在 x 处是**起作用**的，否则称 c_i 在 x 处是**不起作用**的。实际上，假设 f,c_i 都连续，如果 x^* 是一个约束局部极小值点，则去掉 x^* 处不起作用的约束得到的新优化问题也以 x^* 为一个约束局部极小值点。起作用的和不起作用的不等式约束的下标集合是随 x 而变的，在本书中，为记号简单起见，设 $c_i, i=p+1,\dots,p+r(0\le r\le q)$ 是起作用的不等式约束，$c_i, i=p+r+1,\dots,p+q$ 是不起作用的不等式约束，对实际问题则应使用两个随 x 而变化的下标集合或每次重新排列约束次序。

由于可行域形状可能是多种多样的，需要特别地定义可行方向的概念。

定义 35.7 (可行方向)**.** 设 x 是问题(35.1)的可行点，如果有 $d\in\mathbb{R}^d$，$\|d\|=1$，以及序列 $d^{(k)}\in\mathbb{R}^d$, $\|d^{(k)}\|=1$, 和 $\alpha_k>0$，使得 $x^{(k)}=x+\alpha_k d^{(k)}\in D$, 且 $\lim_{k\to\infty}x^{(k)}=x$，$\lim_{k\to\infty}d^{(k)}\to d$, 则称 d 是问题(35.1)在 x 处的一个**可行方向**，记 x 处所有可行方向的集合为 $\mathscr{F}(x)$。可行方向也称为**序列可行方向**。如果 $d\in\mathscr{F}(x)$ 满足 $d^T\nabla f(x)<0$，称 d 是 x 处的一个**可行下降方向**。

另一种可行方向定义比较简单。

定义 35.8 (线性化可行方向)**.** 设 x 是问题(35.1)的可行点，

$$\begin{aligned}\mathscr{L}(x)\triangleq\{d\in\mathbb{R}^d:\|d\|=1,\ d^T\nabla c_i(x)=0, i=1,\dots,p;\\ d^T\nabla c_i(x)\le 0, i=p+1,\dots,p+r\}\end{aligned}$$

称为 x 处的线性化可行方向集，其中的元素称为 x 处的**线性化可行方向**。

注意线性化可行方向定义只对等式约束和起作用的不等式约束有要求，不关心不起作用的不等式约束。可以证明，序列可行方向一定是线性化可行方向，反

之不然。在线性化可行方向定义中，要求 d 与等式约束 $c_i(x)$ 的梯度正交，所以沿 d 略微移动使得 $c_i(x)$ 既不增加也不减少，保持等式不变；要求 d 与起作用的不等式约束 $c_i(x)$ 的梯度不能成锐角，所以沿 d 略微移动不能使得 $c_i(x)$ 增加，所以可以保持不等式成立。

如果 x^* 是问题(35.1)的一个约束局部极小值点，则 x^* 处没有可行下降方向，否则在 x^* 附近可以找到函数值比 $f(x^*)$ 更小的可行点。另一方面，如果可行点 x^* 处所有可行方向 $d \in \mathscr{F}(x^*)$ 都是上升方向 (即 $d^T\nabla f(x) > 0$)，则 x^* 是一个约束严格局部极小值点。

鉴于可行方向集难以确定，以下的定理给出了约束极值点的一个比较容易验证的必要条件。

定理 35.9 (Karush-Kuhn-Tucker)**.** *对约束最优化问题(35.1)，假定目标函数 $f(x)$ 和约束函数 $c_i(x)$ 在 $\mathbb{R}^d$ 有一阶连续偏导数，x^* 是一个约束局部极小值点，如果 $c_i, i = 1, \dots, p+r$ 都是线性函数，或者 $\{\nabla c_i(x^*), i = 1, \dots, p+r\}$ 构成线性无关向量组，则必存在常数 $\lambda_1^*, \dots, \lambda_{p+q}^*$, 使得*

$$\nabla f(x^*) + \sum_{i=1}^{p+q} \lambda_i^* \nabla c_i(x^*) = 0, \tag{35.18}$$

$$\lambda_i^* \geq 0,\ i = p+1, \dots, p+q, \tag{35.19}$$

$$\lambda_i^* c_i(x^*) = 0,\ i = p+1, \dots, p+q. \tag{35.20}$$

对问题(35.1)仍可定义拉格朗日函数

$$L(x, \lambda) = f(x) - \sum_{i=1}^{p+q} \lambda_i c_i(x),$$

定理结论中的(35.20)表明 $L(x^*, \lambda^*) = f(x^*)$, (35.18)可以写成 $\nabla_x L(x^*, \lambda^*) = 0$, 其中 $\nabla_x L(x, \lambda)$ 表示 $L(x, \lambda)$ 的梯度向量中与分量 x 对应的部分。

定理证明略去 (见 高立 [2014] §6.3, Lange [2013] §5.2)。(35.18)–(35.20)称为 **KKT 条件**或 KT 条件，迭代时满足 KKT 条件点 x^* 称为 **KKT 点**, 对应的 λ^* 称为**拉格朗日乘子**，(x^*, λ^*) 称为 **KKT 对**。KKT 点类似于无约束优化问题中的稳定点，很多基于导数的算法在达到 KKT 点后就不能再继续搜索。

注意(35.20)保证了不起作用的不等式约束对应的乘子 $\lambda_i^* = 0, i = p+r+1, \dots, p+q$。这样，我们把 KKT 对 (x^*, λ^*) 处的约束条件 $c_i(x^*), i = 1, \dots, p+q$ 和相应的拉格朗日乘子 λ^* 分为四个部分:

(1) $c_i(x^*)=0$, λ_i^* 可取正、负、零, $i=1,\dots,p$;
(2) $c_i(x^*)=0$, $\lambda_i^*>0$, $i=p+1,\dots,p+r'(r'\le r)$;

(3) $c_i(x^*)=0$, $\lambda_i^*=0$, $i=p+r'+1,\dots,p+r$;
(4) $c_i(x^*)<0$, $\lambda_i^*=0$, $i=p+r+1,\dots,p+q$。

其中，第一部分对应于等式约束，第二、第三部分对应于起作用的不等式约束，第四部分对应于不起作用的不等式约束。注意，这里为了记号简单起见用序号分开了这四部分，实际上第二、三、四部分的组成下标可以是 $\{p+1,\dots,p+q\}$ 的任意分组。把(35.18)写成

$$\nabla f(x^*)=-\sum_{i=1}^{p+r'}\lambda_i^*\nabla c_i(x^*),$$

对线性化可行方向 $d\in\mathscr{L}(x^*)$, 有

$$d^T\nabla f(x^*)=-\sum_{i=1}^{p}\lambda_i^*d^T\nabla c_i(x^*)-\sum_{i=p+1}^{p+r'}\lambda_i^*d^T\nabla c_i(x^*)\ge 0,$$

所以在 KKT 点没有线性化可行下降方向; 没有线性化可行下降方向是约束局部极小值点的必要条件，但不是充分条件。如果某个点 x^* 处所有的可行方向 $d\in\mathscr{F}(x^*)$ 都是严格上升方向，即 $d^T\nabla f(x^*)>0$，则 x^* 是约束局部极小值点。如果存在与 $\nabla f(x^*)$ 正交的可行方向，就需要进一步的信息来判断 x^* 是不是约束极小值点。

在 KKT 对 (x^*,λ^*) 处定义

$$\begin{aligned}\mathscr{L}_1(x^*,\lambda^*)\triangleq\{d\in\mathbb{R}^d:\|d\|=1, d^T\nabla c_i(x^*)=0,\ i=1,\dots,p+r';\\ d^T\nabla c_i(x^*)\le 0, i=p+r'+1,\dots,p+r\},\end{aligned}$$

显然 $\mathscr{L}_1(x^*,\lambda^*)\subset\mathscr{L}(x^*)$, 且对 $d\in\mathscr{L}_1(x^*,\lambda^*)$ 有

$$d^T\nabla f(x^*)=-\sum_{i=1}^{p+r'}\lambda_i^*d^T\nabla c_i(x^*)=0,$$

即 $\mathscr{L}_1(x^*,\lambda^*)$ 是 $\mathscr{L}(x^*)$ 中与梯度 $\nabla f(x^*)$ 正交的方向。因为这样的方向可能存在，判断极小值点可能还需要二阶偏导数条件。这个问题的理由与一元函数导数等于零的点不一定是极值点的理由类似。

定理 35.10 (二阶必要条件)**.** *在定理35.9的条件下，进一步设在 x^* 的一个邻域内 f 与各 c_i 有二阶连续偏导数，则有*

$$d^T\nabla^2_{xx}L(x^*,\lambda^*)d \geq 0,\ \forall d \in \mathscr{L}_1(x^*,\lambda^*). \tag{35.21}$$

定理35.10给出了约束极小值点处拉格朗日函数关于 x 的海色阵的必要条件，类似于无约束情形下要求目标函数海色阵非负定，(35.21)针对方向 $d \in \mathscr{L}_1(x^*,\lambda^*)$ 要求 $\nabla^2_{xx}L(x^*,\lambda^*)$ 非负定。$\mathscr{L}_1(x^*,\lambda^*)$ 是 $\mathscr{L}(x^*)$ 中与梯度 $\nabla f(x^*)$ 正交的方向，为了 x^* 是局部最小值点，对这样的方向需要加二阶导数条件，因为在这样的方向上变化仅从一阶导数看不出会不会下降。

定理 35.11 (二阶充分条件)**.** *对约束最优化问题(35.1)，假定目标函数 f 和各约束函数 c_i 在 x^* 的邻域内有二阶连续偏导数，若 (x^*,λ^*) 是 KKT 对，且*

$$d^T\nabla^2_{xx}L(x^*,\lambda^*)d > 0,\ \forall d \in \mathscr{L}_1(x^*,\lambda^*),$$

则 x^* 是问题(35.1)的一个严格局部极小值点。

如果在 KKT 对 (x^*,λ^*) 处的 $\mathscr{L}_1(x^*,\lambda^*)$ 为空集, 定理35.11结论成立。

定理35.10和定理35.11的证明参见 高立 [2014] §6.4。

例 35.16. 利用 Karush-Kuhn-Tucker 定理可以证明如下不等式：

$$\frac{1}{4}(x_1^2+x_2^2) \leq e^{x_1+x_2-2},\ x_1 \geq 0, x_2 \geq 0.$$

只要证明 $f(x) = -(x_1^2+x_2^2)e^{-x_1-x_2}$ 的最小值是 $-4e^{-2}$。取 $p=0$, $q=2$, 约束函数 $c_1(x) = -x_1$, $c_2(x) = -x_2$, 约束函数都是线性函数，极小值点应该满足条件

$$e^{-x_1-x_2}\begin{pmatrix} -2x_1+x_1^2+x_2^2 \\ -2x_2+x_1^2+x_2^2 \end{pmatrix} + \lambda_1\begin{pmatrix} -1 \\ 0 \end{pmatrix} + \lambda_2\begin{pmatrix} 0 \\ -1 \end{pmatrix} = 0,$$

$$\lambda_1 \geq 0,\ \lambda_2 \geq 0,$$

$$-\lambda_1 x_1 = 0,\ -\lambda_1 x_2 = 0.$$

解集为 $\{(0,0),(1,1),(0,2),(2,0)\}$, 对应的目标函数值分别为 0, $-2e^{-2}$, $-4e^{-2}$, $-4e^{-2}$。易见 $\lim_{\|x\|\to\infty} f(x) = 0 > -4e^{-2}$, 所以目标函数 $f(x)$ 能达到最小值, 所以全局约束最小值点是 $(0,2)$ 和 $(2,0)$，最小值是 $-4e^{-2}$。

作为示例，用二阶条件来判断 $x^* = (2,0)^T$ 是否局部极小值点。解出对应的拉格朗日乘子为 $\lambda^* = (0, 4e^{-2})^T$, 两个约束条件中第二个起作用且对应的拉格朗日乘子为正，第一个不起作用，又 $\nabla c_1(x^*) = (-1,0)^T$, $\nabla c_2(x^*) = (0,-1)^T$,

$$\mathscr{L}_1(x^*, \lambda^*) = \{d : \|d\| = 1, d^T(-1,0)^T \le 0, d^T(0,-1)^T = 0\} = \{(1,0)^T\},$$

计算可得 $(1,0)\nabla^2 L_{xx}(x^*, \lambda^*)(1,0)^T = 2e^{-2} > 0$，所以 $x^* = (2,0)^T$ 是局部极小值点。

※※※※※

35.7 迭代收敛

最优化和方程求根普遍使用迭代算法，从上一步的近似值 $x^{(t)}$ 通过迭代公式得到下一步的近似值 $x^{(t+1)}$。设 $\lim_{t\to\infty} x^{(t)} = x^*$，下面给出收敛速度的一些度量。

定义 35.9. 如果存在 $r > 0$ 和 $c > 0$ 使得

$$\lim_{n\to\infty} \frac{\|x^{(t+1)} - x^*\|}{\|x^{(t)} - x^*\|^r} = c$$

称算法为 r 阶收敛，收敛常数为 c。

定义 35.10. 令

$$Q_1 = \overline{\lim_{t\to\infty}} \frac{\|x^{(t+1)} - x^*\|}{\|x^{(t)} - x^*\|},$$

则 $Q_1 = 0$ 时称算法**超线性收敛**, 当 $0 < Q_1 < 1$ 时称算法**线性收敛**, 当 $Q_1 = 1$ 时称算法**次线性收敛**。

线性收敛是 1 阶收敛，收敛常数为 Q_1。

定义 35.11. 令

$$Q_2 = \overline{\lim_{t\to\infty}} \frac{\|x^{(t+1)} - x^*\|}{\|x^{(t)} - x^*\|^2},$$

则 $Q_2 = 0$ 时称算法**超平方收敛**, 当 $0 < Q_2 < +\infty$ 时称算法**平方收敛**或**二次收敛**, 当 $Q_2 = +\infty$ 时称算法**次平方收敛**。

二次收敛即 2 阶收敛，收敛常数为 Q_2。

最优化算法至少应该有线性收敛速度，否则收敛太慢，实际中无法使用。

实际的最优化问题不一定满足算法收敛的条件，而且往往难以预先判断是否能够收敛。所以，最优化算法对于何时停止迭代，通常使用目标函数值 $f(x^{(t)})$ 两次迭代之间的变化量 $|f(x^{(t+1)}) - f(x^{(t)})|$ 小于预定精度值、迭代近似点 $x^{(t)}$ 两次之间的变化量 $\|x^{(t+1)} - x^{(t)}\|$ 小于预定精度值、梯度向量长度 $\|\nabla f(x^{(t)})\|$ 小于预定精度值等作为停止法则。注意，按照这样的法则停止时并不能保证求解序列逼近了真实的全局或局部极小值点，迭代序列是否收敛到最优解依赖于问题与优化算法。

另外，为了保险起见，当迭代次数超出一个预定最大次数时也停止，但是这时给出算法失败的结果。

在统计计算问题中，目标函数 $f(x)$ 常常不能计算到很高精度，有时目标函数值甚至是由随机模拟方法计算得到的，这样，迭代停止法则不能选取过高的精度，否则算法可能在极值点附近不停跳跃而无法收敛。

35.8 R 软件中的优化和方程求根功能 (*)

`uniroot(f, interval)` 可以对一元函数在指定区间求根。

`optimize()` 函数进行一元函数优化求解。

`optim()` 函数可以进行多元函数无约束最优化，支持 Nelder-Mead、BFGS、模拟退化等算法，也支持简单的区间约束问题求解。

`nlm()` 求解最小二乘问题。

例 35.17. 例35.14的求解。

`optim(x0, f)` 可以从初值 `x0` 出发用 Nelder-Mead 方法对目标函数 `f` 求解无约束优化问题。

```
f <- function(x) 1/4*x[1]^4 + 1/2*x[2]^2 - x[1]*x[2] + x[1] - x[2]
optim(c(0, 0), f)
## $par
```

```
## [1] -1.000000e+00 -1.250914e-08
##
## $value
## [1] -0.75
##
## $counts
## function gradient
##      123       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

它找到了两个最小值点之一的 $(-1, 0)$。

使用 BFGS 方法：

```
optim(c(0, 0), f, method="BFGS")
```

结果类似。

35.9 Julia 软件中的优化和方程求根功能 (*)

JuMP是 Julia 语言的一个最优化问题建模求解软件包，在不同后端计算软件支持下可以计算广泛的最优化问题。Convex 包也是一个功能强大的优化软件包。

Optim是一个纯粹用 Julia 语言编写的优化求解软件包，功能比较少，主要针对无约束优化问题。

例 35.18. 例35.14的求解。

用 Optim 包求解：

```
using Optim

function f(xvec)
    x, y = xvec
    1/4*x^4 + 1/2*y^2 - x*y + x - y
end

# 使用默认的 Nelder-Mead 方法求解
ores = Optim.optimize(f, [0.0, 0.0])
@show Optim.minimizer(ores);
## Optim.minimizer(ores) = [-1.0001049651060008, -2.3876814817729273e-5]
```

其中 ores 显示的结果为：

```
 * Status: success

 * Candidate solution
    Final objective value:     -7.500000e-01

 * Found with
    Algorithm:     Nelder-Mead

 * Convergence measures
    √(Σ(y-ȳ)²)/n   1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    38
    f(x) calls:    77
```

35.10 附录：证明补充 (*)

定理35.1（一阶必要条件）的证明：

已知 $x^* \in (a,b)$，存在 $\delta > 0$，使得当 $0 < h < \delta$ 时

$$f(x^* \pm h) \geq f(x^*).$$

做泰勒展开得

$$f(x^* \pm h) = f(x^*) \pm hf'(x^*) + o(h) \geq f(x^*),$$

即

$$\pm hf'(x^*) + o(h) \geq 0,$$

于是

$$o(h) \geq \pm hf'(x^*), \quad o(h) \geq h|f'(x^*)|,$$

取极限

$$0 = \lim_{h \downarrow 0} \frac{o(h)}{h} \geq |f'(x^*)| \geq 0,$$

故 $f'(x^*) = 0$，得证。

※※※※※

定理35.2（二阶必要条件）的证明：

类似地作泰勒展开

$$f(x^* \pm h) = f(x^*) \pm hf'(x^*) + \frac{1}{2}h^2 f''(x^*) + o(h^2) \geq f(x^*),$$

其中 $f'(x^*) = 0$，有

$$\frac{1}{2}h^2 f''(x^*) + o(h^2) \geq 0,$$

从而

$$\frac{1}{2}f''(x^*) + o(1) \geq 0, \ (h \downarrow 0),$$

令 $h \downarrow 0$ 得 $f''(x^*) \geq 0$。

※※※※※

定理35.3（二阶充分条件）的证明：

由

$$f''(x^*) = \lim_{h \to 0} \frac{f'(x^* + h) - f'(x^*)}{h} = \lim_{h \to 0} \frac{f'(x^* + h)}{h} > 0,$$

存在 $\delta > 0$ 使得 $0 < |h| < \delta$ 时 $f'(x^* + h)$ 与 h 同号，因此函数 $f(x)$ 在 $(x^* - \delta, x^*)$ 为减函数，在 $(x^*, x^* + \delta)$ 为增函数，所以 x^* 是局部极小值点。

※※※※※

习题

习题 1

设 $x = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$, $y = (y_1, y_2, \dots, y_n) \in \mathbb{R}^n$，证明如下的 Cauchy-Schwarz 不等式:

$$\big|(x, y)\big| \leq \|x\| \cdot \|y\|$$

其中 $(x, y) = \sum_{i=1}^{n} x_i y_i$, $\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}$，不等式中等号成立当且仅当存在实数 a, b 使得 $ax + by = 0$。

习题 2

证明凸集性质 1–10。

习题 3

证明无约束和有约束的凸规划问题的局部最优解一定是全局最优解。

习题 4

对例35.13, 证明当 X 不满秩时方程 $X^T X\beta = X^T Y$ 有无穷多解, 任一个解 β^* 都是 $Q(\beta)$ 的全局最小值点。

习题 5

在定理35.9条件下，如果 $\{\nabla c_i(x^*), i = 1, \dots, p + r\}$ 构成线性无关向量组，则 λ^* 唯一。

习题 6

对约束优化问题

$$\begin{cases} \operatorname{argmin} f(x_1, x_2) \triangleq 4x_1 - 3x_2, \quad \text{s.t.} \\ 4 - x_1 - x_2 \geq 0, \\ x_2 + 7 \geq 0, \\ -(x_1 - 3)^2 + x_2 + 1 \geq 0 \end{cases}$$

求其 KKT 点，并根据二阶条件判断 KKT 点是否最小值点。

习题 7

设函数 $f(x)$ 定义在 $\mathbb{R}^d$ 上，有一阶连续偏导数，$v \in \mathbb{R}^d$, $\|v\| = 1$。令 $h(\alpha) = f(x + \alpha v)$, $\alpha \geq 0$, 证明 $h'(0) = v^T \nabla f(x)$。对一般的 $u \in \mathbb{R}^d$，若 $\|u\| > 0$，仍有

$$\frac{d}{d\alpha} f(x + \alpha u)|_{\alpha=0} = u^T \nabla f(x).$$

Chapter 36

一维搜索与求根

即使目标函数 $f(x)$ 是一元函数，求最小值点也经常需要使用数值迭代方法。另外，在多元目标函数优化中，一般每次迭代从上一步的 $x^{(t)}$ 先确定一个下降方向 $d^{(t)}$, 然后对派生出的一元函数 $h(\alpha) = f(x^{(t)} + \alpha d^{(t)})$, $\alpha \geq 0$ 求最小值点得到下降的步长 α_t，并令 $x^{(t+1)} = x^{(t)} + \alpha_t d^{(t)}$，求步长的过程称为**一维搜索**，搜索可以是求一元函数 $h(\alpha)$ 的精确最小值点，也可以求一个使得目标函数下降足够多的 α 作为步长。对于多元目标函数 $f(x)$，$x \in \mathbb{R}^d$, 有一种优化方法是先沿 x_1 坐标轴方向搜索，再沿 x_2 坐标轴方向搜索，直到沿 x_d 坐标轴方向搜索后再重复地从 x_1 坐标轴方向开始，这种方法叫做**坐标循环下降法**。

一元函数的求根问题和优化问题使用的算法类似,下面讨论一元函数的优化和求根问题。在 R 语言中,`uniroot()` 函数可以用来进行一元函数求根,`polyroot()` 可以求多项式的所有复根，`optimize()` 可以进行一元函数优化。

36.1 二分法求根

优化问题与求根问题密切相关，很多优化问题会归结为一个求根问题。对一元目标函数 $f(x)$，若 $f(x)$ 连续可微，极值点一定是 $f'(x) = 0$ 的根。二分法是实际数值计算中一元函数求根使用最多的方法，这种方法简单而又有比较高的收敛速度。

设函数 $f(x)$ 在闭区间 $[a, b]$ 连续，且 $f(a)f(b) \leq 0$，由中间值定理，至少有一

个点 $x^* \in [a,b]$ 使得 $f(x^*)=0$。如果 $f(x)$ 在 $[a,b]$ 上还是严格单调函数则解 x^* 存在唯一。

二分法求根用区间 $[a,b]$ 中点 c 处函数值 $f(c)$ 的正负号来判断根在区间中点的左边还是右边，显然，如果 $f(c)$ 与 $f(a)$ 异号，则 $[a,c]$ 中有根；如果 $f(c)$ 与 $f(b)$ 异号，则 $[c,b]$ 中有根。根据这样的想法，可以迭代地每次用区间的左半部分或右半部分代替原来的含根区间，逐步缩小含根的区间。设求根的绝对误差限规定为 ϵ。算法如下:

二分法求根算法

令 $f_a = f(a),\ f_b = f(b)$
until $(b-a<\epsilon)$ {
    令 $c \leftarrow \frac{1}{2}(a+b),\ f_c \leftarrow f(c)$
    if $(f_a f_c \le 0)$ {
        $b \leftarrow c,\ f_b \leftarrow f_c$
    } **else** {
        $a \leftarrow c,\ f_a \leftarrow f_c$
    }
}
输出 c 作为根

设真实根为 x^*，第 k 步的区间中点为 $x^{(k)}$，则

$$|x^{(k)} - x^*| \le (b-a)\left(\frac{1}{2}\right)^k,$$

二分法具有线性收敛速度。

如果 $f(x)$ 是 $[a,\infty)$ 区间上的严格单调的连续函数, 且 $\lim_{x\to\infty} f(x)$ 与 $f(a)$ 反号，则 $f(x)$ 在 $[a,\infty)$ 上存在唯一的根，这时需要先找到闭区间 $[a,b]$ 使得 $f(a)$ 与 $f(b)$ 反号，二分法算法需要略作调整:

区间右侧无穷的二分法求根算法

取步长 $h>0$, 倍数 $\gamma>1$
until $(f(a)f(b)\le 0)$ {

区间右侧无穷的二分法求根算法

$b \leftarrow a + h$

$h \leftarrow \gamma h$

}

用闭区间上的二分法在 $[a, b]$ 内求根

其它的情况，比如 $f(x)$ 定义在 $(-\infty, \infty)$, (a, b) 的情况可类似处理，都需要先找到使得区间端点函数值符号相反的闭区间。

例 36.1. 设某种建筑钢梁的强度 X 服从正态 $\mathrm{N}(\mu, \sigma^2)$ 分布，μ, σ^2 未知，根据设计要求，当 $X > L(L$ 已知$)$ 时钢梁的强度才达到要求，称 L 为强度的下规范限，称强度达到要求的概率 $R = P(X > L)$ 为单侧性能可靠度, 在可靠性评估中需要在给了总体 X 的一组样本 $X_1, X_2, \ldots, X_n$ 以后求可靠度 R 的 $1 - \alpha$ 置信下限 $(0 < \alpha < 1)$。

记 $K = (\mu - L)/\sigma$，则 $R = \Phi(K)$, 所以 R 是 K 的严格单调增连续函数，只要求 K 的置信下限。

假设从样本 $X_1, X_2, \ldots, X_n$ 中计算了样本平均值 $\bar{X}$ 和样本方差 S^2，定义 $\hat{K} \triangleq (\bar{X} - L)/S$，$\hat{K}$ 的分布仅依赖于 K，经过简单推导可以得知 $\sqrt{n}\hat{K}$ 服从非中心 $\mathrm{t}(n-1, \sqrt{n}K)$ 分布, 设其分布函数为 $\mathrm{pt}(\cdot, n-1, \sqrt{n}K)$。

按照置信下限的统计量法 (见 [陈家鼎、孙山泽、李东风、刘力平, 2006] §2.3)，只要定义

$$\begin{aligned} G(u, K) =& P(\hat{K} \geq u),\ u \in (0, 1),\ K \in \mathbb{R}, \\ g(u) =& \inf_{K \in (-\infty, \infty)} \{K :\ G(u, K) > \alpha\},\ u \in (0, 1), \end{aligned}$$

则 $g(\hat{K})$ 是 K 的 $1 - \alpha$ 置信下限, 从而 $\Phi(g(\hat{K}))$ 是 R 的置信下限。而

$$G(u, K) = 1 - \mathrm{pt}(\sqrt{n}u, n-1, \sqrt{n}K)$$

是 K 的严格单调增函数，且 $\lim_{K \to -\infty} G(u, K) = 0$, $\lim_{K \to +\infty} G(u, K) = 1$, 所以在 $G(u, K) > \alpha$ 约束下求 K 的下确界，等价于在固定 u 的情况下对关于 K 严格单调增的连续函数 $f(K) \triangleq G(u, K) - \alpha$ 求根，易见根存在唯一，只要用二分法求根即可。需要先求得包含根的闭区间。

求含根区间的算法

取初始步长 $h_0 > 0$, 倍数 $\gamma > 1$
$h \leftarrow h_0$, $a \leftarrow 0$, $b \leftarrow 0$
while $(f(a) > 0)$ { # 这时不需要单独找 b
    $b \leftarrow a$
    $a \leftarrow a - h$
    $h \leftarrow \gamma h$
}
$h \leftarrow h_0$
while $(f(b) \leq 0)$ { # 这时不需要单独找 a
    $a \leftarrow b$
    $b \leftarrow b + h$
    $h \leftarrow \gamma h$
}
用闭区间上的二分法在 $[a, b]$ 内求根

※※※※※

36.2 牛顿法

设目标函数 $f(x)$ 有连续二阶导数，求最小值点可以通过求解 $f'(x) = 0$ 的根实现。记 $g(x) = f'(x)$。设 x^* 是 $f(x)$ 的一个最小值点，则 $g(x^*) = 0$，设 x_0 是 x^* 附近的一个点。在 x_0 附近对 $g(x)$ 作一阶泰勒近似得

$$g(x) \approx g(x_0) + g'(x_0)(x - x_0),$$

令 $g(x) = 0$，得 x 近似为

$$x_1 = x_0 - \frac{g(x_0)}{g'(x_0)},$$

写成迭代公式，即

$$x_{t+1} = x_t - \frac{g(x_t)}{g'(x_t)},\ t = 0, 1, 2, ... \tag{36.1}$$

或

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)},\ t = 0, 1, 2, ... \tag{36.2}$$

如果 $g'(x)$ 在 x^* 的一个邻域 $[x^*-\delta, x^*+\delta]$ 内都为正，初值 $x_0 \in [x^*-\delta, x^*+\delta]$，则牛顿法产生的迭代数列 $\{x_t\}$ 收敛到 x^*。如果初值接近于最小值点并且目标函数满足一定正则性条件，牛顿法有二阶收敛速度。但是，如果迭代过程中遇到 $g'(x)$ 接近于零的点，下一个迭代点可能会被抛到远离根 x^* 的地方，造成不收敛。另外，牛顿法迭代的停止法则，可以取为 $|x_{t+1} - x_t| < \epsilon_x$ 或 $|g(x_t)| < \epsilon_g$，ϵ_x 和 ϵ_g 是预定的精度值。在实际数值计算中，函数 $g(x)$ 和 $g'(x)$ 只有一定的计算精度，所以算法中对 ϵ_x 和 ϵ_g 的选取并不是越小越好，一般够用就可以了，取 ϵ 太小有可能导致算法在 x^* 附近反复跳跃而不能满足收敛法则。判断算法收敛，可以使用绝对变化量也可以使用相对变化量。实际算法经常使用 $U^{1/4}$ 作为相对变化量的界限，其中 U 是双精度值的机器单位。为了保险起见，还应该设置一个迭代最大次数，迭代超过最大次数时宣告算法失败。

牛顿法是方程求根方法, 也可以通过求 $f(x)$ 的导数 $g(x)$ 的根来求最小值点。这里给出牛顿法的一个几何解释。设 x_t 是根 x^* 附近的一个点，从曲线 $g(x)$ 上的点 $(x_t, g(x_t))$ 作切线 (见图36.1)，切线方程为

$$y = g(x_t) + g'(x_t)(x - x_t),$$

用切线在 x_t 附近作为曲线 $g(x)$ 的近似，把求 $g(x) = 0$ 的根近似地变成求切线与 x 轴交点的问题，令

$$0 = g(x_t) + g'(x_t)(x - x_t),$$

则交点 x 的值就是公式(36.1)的下一个迭代值 x_{t+1}。

例 36.2. 对函数 $f(x) = \frac{1}{4}x^4 - \frac{1}{8}x$，求 $\operatorname{argmin}_{x\geq 0} f(x)$。

令 $g(x) = f'(x) = x^3 - \frac{1}{8}$，显然 $g(x) = 0$ 在 $x \geq 0$ 有唯一的根 $x^* = \frac{1}{2}$。取 $x_0 = 2$,用(36.1)进行迭代的过程如图36.1。迭代的前几个近似值为 $x_1 = 1.3438$, $x_2 = 0.9189$, $x_3 = 0.6619$。从图中可以看出，牛顿法当 $g(x)$ 斜率大且形状接近直线时收敛最快。

用 R 的 `uniroot()` 求解：

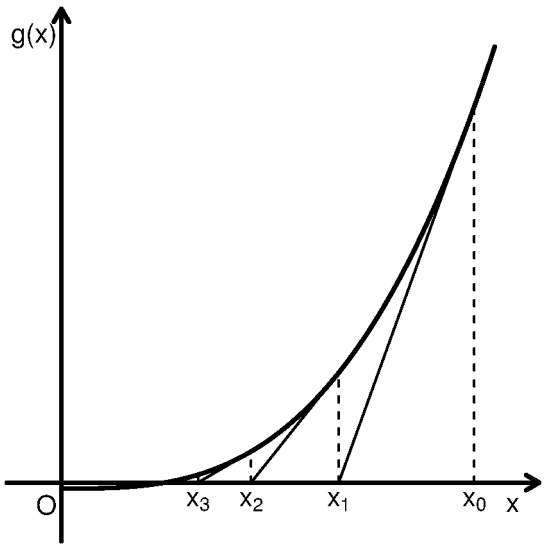


图 36.1: 用牛顿法解一元非线性方程

```
uniroot(function(x) x^3 - 1/8, c(0, 10))
```

结果：

```
$root
[1] 0.5000279

$f.root
[1] 2.095328e-05

$iter
[1] 13

$init.it
[1] NA

$estim.prec
```

```
[1] 6.103516e-05
```

※※※※※

如果待求根的函数 $g(x)$ 的导数 $g'(x)$ 没有表达式或很难推导，也可以用数值微分方法计算 $g'(x)$。另一种方法是用函数图像上连接两次迭代的两个点 $(x_{t-1}, g(x_{t-1}))$ 和 $(x_t, g(x_t))$ 的连线斜率代替公式(36.1)中的 $g'(x_t)$，公式变成

$$x_{t+1} = x_t - \frac{x_t - x_{t-1}}{g(x_t) - g(x_{t-1})} g(x_t), \tag{36.3}$$

这种方法称为**割线法**，适当正则条件下具有超线性收敛速度但比牛顿法差一些。割线法简单易行，但是和牛顿法一样，遇到 $g'(x)$ 接近于零的点会使得近似点被抛离真值 x^*。有一些方法结合割线法与二分法的优点，具有超线性收敛速度而且保证根总在迭代的区间内，参见 Monahan [2001] §8.3。

36.3 一维搜索的区间

一维搜索一般需要先确定包含极小值点的区间, 方程求根时可能也需要确定根所在的区间。

对一元连续函数 $f(x)$，若存在 $A < x^* < B$ 使得 $f(x)$ 在 $(A, x^*]$ 单调下降，在 $[x^*, B)$ 单调上升，则称 $f(x)$ 在 (A, B) 是**单谷**的。这时，若存在 $A < a < c < b < B$ 使得 $f(c) < f(a)$, $f(c) < f(b)$，则 x^* 必在 (a, b) 中。这是因为如果 $x^* \le a$，则 a, c, b 三个点在单调上升分支，必有 $f(a) \le f(c) \le f(b)$，如果 $x^* \ge b$ 则 a, c, b 三个点在单调下降分支，必有 $f(a) \ge f(c) \ge f(b)$，都会导致矛盾。

设定义于 $(-\infty, \infty)$ 或 $(0, \infty)$ 的连续目标函数 $f(x)$ 存在局部极小值点 x^*, 且设 $f(x)$ 在 x^* 的一个邻域内是单谷的, 则只要能找到三个点 $a < c < b$ 使得 $f(a) > f(c)$, $f(c) < f(b)$，则在 (a, b) 内必存在 $f(x)$ 的一个极小值点。

为了求 a, b, 从两个初始点 x_0, x_1 出发，如果 $f(x_0) > f(x_1)$，则最小值点可能在 x_1 右侧，向右侧搜索找到一个函数值超过 $f(x_1)$ 的点 x_2，这时最小值点应该在 (x_0, x_2) 内部；如果 $f(x_0) < f(x_1)$，则最小值点可能在 x_0 左侧，向左侧搜索 (参考图36.2)。

求含根区间 $[a,b]$ 的算法

取步长 $h>0$，步长倍数 $\gamma>1$, 初始点 x_0，$x_1 \leftarrow x_0+h$, $k \leftarrow 0$
if $(f(x_0)>f(x_1))$ { # 这时可以向右搜索得到 a,c,b
    until $(f(x_{k+1})>f(x_k))$ {
        $k \leftarrow k+1$
        $x_{k+1} \leftarrow x_k+\gamma^k h$
    }
    $a \leftarrow x_{k-1}, c \leftarrow x_k, b \leftarrow x_{k+1}$
} **else** { # 这时可以向左搜索
    交换 x_0 与 x_1，这样 $x_1<x_0$, $f(x_0)>f(x_1)$
    until $(f(x_{k+1})>f(x_k))$ {
        $k \leftarrow k+1$
        $x_{k+1} \leftarrow x_k-\gamma^k h$
    }
    $a \leftarrow x_{k+1}, c \leftarrow x_k, b \leftarrow x_{k-1}$
}
输出 $[a,b]$ 作为包含极小值点的区间

如果函数 $f(x)$ 的定义域为 $x\in(0,\infty)$，以上的算法应取初值 $x_0>0$，如果出现向左搜索，可取迭代公式为 $x_{k+1}=\frac{1}{\gamma}x_k$。

如果目标函数 $f(x)$ 连续可导，则求最小值点所在区间也可以通过求 $a<b$ 使得 $f'(a)<0$, $f'(b)>0$ 来解决。可取初值 x_0，如果 $f'(x_0)<0$，则类似上面算法那样向右搜索找到 b 使得 $f'(b)>0$；如果 $f'(x_0)>0$，则向左搜索找到 $a<x_0$ 使得 $f'(a)<0$。用最后得到的两个点作为 a,b。可以看出，这里描述的搜索方法适用于一元函数求根时寻找包含根的区间的问题。

36.4 0.618 法 (*)

0.618 法（黄金分割法）是一种常用的不需要导数信息的一维搜索方法。假设 $f(x)$ 在闭区间 $[a,b]$ 内是单谷的，存在最小值点 $x^*\in(a,b)$，这时可以用 0.618 法求最小值点。

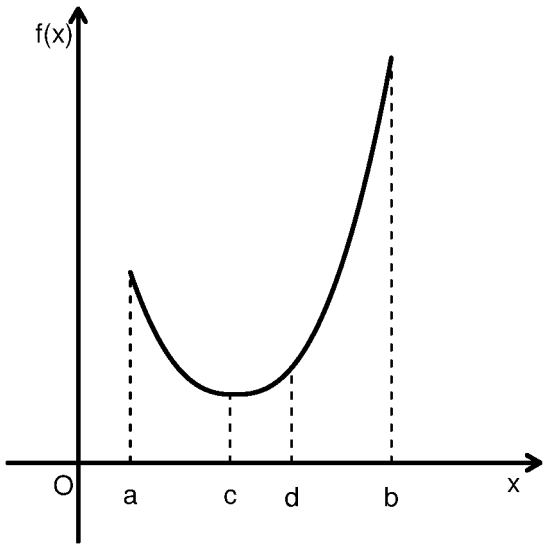


图 36.2: 0.618 法

在区间 $[a,b]$ 内对称地取两个点 $c<d$ 在所谓的黄金分割点上 (见图36.2)，即

$$\frac{d-a}{b-a}=\rho\triangleq\frac{\sqrt{5}-1}{2}\approx 0.618,\quad \frac{b-c}{b-a}=\rho,$$

黄金分割比例 ρ 满足

$$\frac{1-\rho}{\rho}=\rho,$$

这样的比例的特点是，当区间缩小到 $[a,d]$ 时，c 仍为缩小后的区间的右侧黄金分割点；当区间缩小到 $[c,b]$ 时，d 仍为缩小后的区间的左侧黄金分割点。这样选了 c,d 两个点后，比较 $f(c)$ 和 $f(d)$ 的值，如果 $f(c)$ 较小，则因为 $a<c<d$, $f(a)>f(c)$, $f(c)<f(d)$, 最小值点 x^* 一定在区间 $[a,d]$ 内，可以把搜索区间缩小到 $[a,d]$, 以 $[a,d]$ 作为含有最小值点的区间再比较其中两个黄金分割点的函数值，而且这时 c 是 $[a,d]$ 中右侧的黄金分割点，只要再添加左侧的黄金分割点就可以了。如果比较 $f(c)$ 和 $f(d)$ 时发现 $f(d)$ 较小，则把搜索区间缩小到 $[c,b]$, 这时 d 是 $[c,b]$ 内的左侧黄金分割点，只要再添加右侧黄金分割点。每次迭代使得区间长度缩小到原来的 0.618 倍, 缩小后的两个黄金分割点中一个的坐标和函数值是已经算过的。

设规定的最小值点绝对误差限为 ϵ，算法如下:

0.618 法

令 $\rho \leftarrow 0.618,\ t \leftarrow 0$
令 $a_0 \leftarrow a,\ b_0 \leftarrow b$
令 $c_0 \leftarrow \rho a_0 + (1-\rho)b_0,\ d_0 \leftarrow (1-\rho)a_0 + \rho b_0$
until $(b_t - a_t < \epsilon))$ {
    if $(f(c_t) < f(d_t))$ { # 保留左侧
        $a_{t+1} \leftarrow a_t,\ b_{t+1} \leftarrow d_t$
        $d_{t+1} \leftarrow c_{t+1},\ c_{t+1} \leftarrow \rho a_{t+1} + (1-\rho)b_{t+1}$
    } **else** { # 保留右侧
        $a_{t+1} \leftarrow c_t,\ b_{t+1} \leftarrow b_t$
        $c_{t+1} \leftarrow d_t,\ d_{t+1} \leftarrow (1-\rho)a_{t+1} + \rho b_{t+1}$
    }
    $t \leftarrow t+1$
} # end until
令 $x^* \leftarrow \frac{1}{2}(a_t + b_t)$
输出 x^* 作为最小值点近似值

0.618 法具有线性收敛速度。

36.5 抛物线法 (*)

牛顿法需要计算导数，在初值合适的情况下收敛快。牛顿法本质上是利用 $f(x)$ 的一、二阶导数对 $f(x)$ 用二次多项式进行近似并求近似函数的最小值点作为下一个近似值。事实上，为了用二次多项式逼近 $f(x)$，只要有 $f(x)$ 函数图像上的三个点就可以。仍假设 $f(x)$ 在闭区间 $[a,b]$ 内是单谷的，存在最小值点 $x^* \in (a,b)$。设 $c \in (a,b)$, 利用 $(a,f(a)),\ (c,f(c)),\ (b,f(b))$ 三个点求得二次插值多项式 $\varphi(x)$，用 $\varphi(x)$ 的最小值点作为下一个近似值。

设插值多项式为

$$\varphi(x) = \beta_0 + \beta_1 x + \beta_2 x^2,$$

其中 $\beta_2 > 0$, 在给定了 $a < c < b$ 和相应函数值后，可以解出

$$\begin{aligned}\beta_2 =& \frac{1}{b-a}\left[\frac{f(b)-f(c)}{b-c} - \frac{f(c)-f(a)}{c-a}\right],\\ \beta_1 =& \frac{f(c)-f(a)}{c-a} - \beta_2(c+a),\end{aligned}$$

从而求得 $\varphi(\cdot)$ 的最小值点

$$\tilde{x} = -\frac{\beta_1}{2\beta_2}$$

如果 $\tilde{x} < c$, 则以 $a, \tilde{x}, c$ 为新的插值节点继续计算插值多项式并求插值多项式的最小值点；如果 $\tilde{x} > c$, 则以 $c, \tilde{x}, b$ 为新的插值节点继续计算插值多项式并求插值多项式的最小值点，如此迭代直至三个插值点足够接近。

适当条件下抛物线法达到超线性收敛速度。

在 R 软件中，用 `optimize` 函数求一元函数极值点, 函数结合使用了 0.618 法和抛物线法。

36.6 Brent 法求根 (*)

二分法求根只有线性收敛速度，参考上面的抛物线法，利用二次多项式反向插值方法得到根的下一个近似值。

设已知含根区间为 (a, c)，令 $x_1 = a$, $x_2 = c$，$x_3 = (x_1 + x_2)/2$。设迭代 k 步以后，记 $a = x_{k-2}$, $b = x_k$, $c = x_{k-1}$，应有 $a < b < c$。不妨设 $f(a)f(b) < 0$(否则应有 $f(b)f(c) < 0$，可类似处理)。在区间 (a, c) 内设函数 $y = f(x)$ 可逆，将 x 看成 y 的函数，利用 Lagrange 插值法得到插值公式：

$$\begin{aligned}x = & a\frac{[y-f(b)][y-f(c)]}{[f(a)-f(b)][f(a)-f(c)]}\\ & + b\frac{[y-f(a)][y-f(c)]}{[f(b)-f(a)][f(b)-f(c)]}\\ & + c\frac{[y-f(a)][y-f(b)]}{[f(c)-f(a)][f(c)-f(b)]}\end{aligned}$$

上式中令 $y=0$，得

$$\begin{aligned} x_{k+1}^{*} = & a\frac{f(b)f(c)}{[f(a)-f(b)][f(a)-f(c)]} \\ & + b\frac{f(a)f(c)}{[f(b)-f(a)][f(b)-f(c)]} \\ & + c\frac{f(a)f(b)}{[f(c)-f(a)][f(c)-f(b)]} \end{aligned}$$

如果 $x_{k+1}^{*} \in (a,b)$（在含根区间内），则取 $x_{k+1}=x_{k+1}^{*}$; 如果 $x_{k+1}^{*} \notin (a,c)$，则退回到二分法，取 $x_{k+1}=\frac{a+b}{2}$。如此迭代直到区间长度小于指定精度值。

这种方法称为 Brent 求根法，它和二分法一样需要找到含根区间并设在区间内函数连续，在此条件下算法保证收敛且至少有线性收敛速度，但此方法又能够利用局部二次多项式逼近，所以收敛速度一般比二分法快。

R 的 `uniroot()` 函数实现了 Brent 法。

36.7 沃尔夫准则

在求多元函数优化问题时经常需要进行一维搜索，一维搜索可以在搜索方向上求精确最小值点，也可以求一个使得函数值下降足够多的点，称为不精确一维搜索。

设目标函数 $f(x)$ 有连续二阶偏导数，当前的最小值点近似值为 $x^{(t)}$，当前的搜索方向为 $d^{(t)}$, 满足 $[\nabla f(x^{(t)})]^T d^{(t)}<0$。令 $h(\alpha)=f(x^{(t)}+\alpha d^{(t)}), \alpha \geq 0$，则

$$\begin{aligned} h'(\alpha) =& [\nabla f(x^{(t)}+\alpha d^{(t)})]^T d^{(t)}, \\ h'(0) =& [\nabla f(x^{(t)})]^T d^{(t)} < 0, \\ h(\alpha) =& h(0)+h'(0)\alpha + o(\alpha), \end{aligned}$$

由 $h'(\alpha)$ 的连续性可知 $h(\alpha)$ 在 0 的一个右侧邻域中严格单调下降。

取 $\mu \in (0,\frac{1}{2}), \sigma \in (\mu,1)$，不精确一维搜索的沃尔夫准则要求步长 α 满足如下两个条件:

$$h(0)-h(\alpha) \geq \mu[-h'(0)]\alpha, \tag{36.4}$$

$$-h'(\alpha) \leq \sigma[-h'(0)]. \tag{36.5}$$

第一个限制条件要求函数值下降足够多，第二个限制条件要求 $|h'(\alpha)|$ 已经比较接近于零（精确一维搜索要求 $h'(\alpha)=0$）。有许多个 α 可以使这两个条件成立。把这两个条件用目标函数 $f(\cdot)$ 表示，可以写成

$$\begin{aligned} &f(x^{(t)})-f(x^{(t)}+\alpha d^{(t)})\geq \mu\left\{-[\nabla f(x^{(t)})]^T d^{(t)}\right\}\alpha,\\ &-[\nabla f(x^{(t)}+\alpha d^{(t)})]^T d^{(t)}\leq \sigma\left\{-[\nabla f(x^{(t)})]^T d^{(t)}\right\}. \end{aligned}$$

σ 取值越小，一维搜索越精确，花费的时间也越长。实际中常取 $\mu=0.1$, $\sigma\in[0.6,0.8]$。

下面的算法可以找到满足沃尔夫准则的两个条件的步长 α。想法是，如果条件不满足，就利用现有的 $h(\alpha)$ 的函数值和一阶导数值构造 $h(\alpha)$ 的二阶插值多项式，计算插值多项式的最小值点 $\tilde{\alpha}$ 作为 α 的下一个试验值。算法如下:

不精确一维搜索的 Wolfe 法

取定 $\mu\in(0,\frac{1}{2})$, $\sigma\in(\mu,1)$, 最大步长 $\overline{\alpha}$, 令 $\alpha_0\leftarrow 0$, $\alpha_2\leftarrow\overline{\alpha}$
计算 $y_0\leftarrow h(0)$, $y_0'=h'(0)=[\nabla f(x^{(t)})]^T d^{(t)}$
令 $\alpha_1\leftarrow\alpha_0$, $y_1\leftarrow y_0$, $y_1'\leftarrow y_0'$
取 $\alpha\in(0,\alpha_2)$
令 finished $\leftarrow$ FALSE
until (finished) {
　　计算 $y\leftarrow h(\alpha)$
　　if($y_0-y\geq-\mu y_0'\alpha$) { # 满足沃尔夫条件 1
　　　　计算 $y'\leftarrow h'(\alpha)$
　　　　if($-y'\leq-\sigma y_0'$) { # 满足沃尔夫条件 1,2
　　　　finished $\leftarrow$ TRUE; break
　　　　} **else** { # 满足条件 1 但不满足条件 2, 延伸搜索
　　　　　　由 α_1,y_1,y_1' 和 α,y' 计算 $\tilde{\alpha}\leftarrow\alpha_1-\dfrac{\alpha-\alpha_1}{y'-y_1'}y_1'$
　　　　　　令 $\alpha_1\leftarrow\alpha$, $y_1\leftarrow y$, $y_1'\leftarrow y'$, $\alpha\leftarrow\tilde{\alpha}$
　　　　}
　　} **else** { # 不满足沃尔夫条件 1, 收缩搜索
　　　　由 α_1,y_1,y_1' 和 α,y 计算 $\tilde{\alpha}\leftarrow-\dfrac{(\alpha^2-\alpha_1^2)y_1'-2\alpha_1(y-y_1)}{2[y-y_1-(\alpha-\alpha_1)y_1']}$
　　　　令 $\alpha_2\leftarrow\alpha$, $\alpha\leftarrow\tilde{\alpha}$

不精确一维搜索的 Wolfe 法

```
    }
} # until
```

输出 α 作为步长

36.8 附录：二分法求根的程序 (*)

输入要求根的函数，含有根的区间 a, b，根的误差界限 ，输出缩小的含根区间。参见 [Kochenderfer]。

```
## 二分法求根
function bisection(f, a, b, )
    if a > b
        # 确保 a < b
        a,b = b,a
    end
    ya, yb = f(a), f(b)

    if ya == 0 # 已经是根
        b = a
    end
    if yb == 0 # 已经是根
        a = b
    end
    while b - a >
        x = (a+b)/2
        y = f(x)
        if y == 0
            a, b = x, x # 已经是根
        elseif sign(y) == sign(ya)
            a = x # 根在右半边
        else
```

```
            b = x # 根在左半边
        end
    end

    # 返回缩小的含根区间
    return (a,b)
end
```

下面是一个简化的求含根区间的程序，设 f 定义在 $(-\infty, \infty)$ 上：

```
## 求含根区间
function bracket_sign_change(f, a, b; k=2)
    if a > b
        # 确保 a < b
        a,b = b,a
    end
    center= (b+a)/2
    half_width = (b-a)/2
    # 不断从中心向两端扩张直至含根
    while sign(f(a))  sign(f(b))
        half_width *= k
        a = center - half_width
        b = center + half_width
    end
    return (a,b)
end
```

36.9 附录：求最小值所在区间的程序 (*)

输入要求最大值的函数 f，初值 x，设 f 是凸函数，求含最小值的区间。参见 [Kochenderfer]。

```
function bracket_minimum(f, x=0; s=1e-2, k=2.0)
    a, ya = x, f(x) # 初始值
    # 向右一步，希望下降
    b, yb = a + s, f(a + s)
    if yb > ya
        # 如果非下降，就改为向左
        a, b = b, a
        ya, yb = yb, ya
        s = -s
    end

    # 此处已经满足 ya > yb 条件，只要找到 yc > yb
    while true
        c, yc = b + s, f(b + s)
        if yc > yb
            # 找到 c 使得 ya>yb, yc>yb 则满足条件
            return a < c ? (a, c) : (c, a)
        end
        # 没有找到 c，则增大步长，继续沿原方向查找
        a, ya, b, yb = b, yb, c, yc
        s *= k
    end # while
end
```

36.10 附录：0.618 法的程序 (*)

输入单谷函数 f，含最小值的区间 (a, b)，迭代次数 n，求迭代 n 次后的含最小值区间。每次迭代区间长度减小到原来的 0.618 倍。参见 [Kochenderfer]。

```
# 一元凸函数黄金分割法求最小值。
function golden_section_search(f, a, b, n)
    # 0.618 黄金分割比
```

```
      = (sqrt(5)-1)/2

    # d: (a,b) 中靠近 b 的黄金分割点
    d =   * b + (1 -  )*a
    yd = f(d)

    for i = 1 : n-1
        # c: (a, b) 中靠近 a 的黄金分割点
        c =  *a + (1 -  )*b
        yc = f(c)
        # 比较 a, b, c, d 四个点的函数值
        if yc < yd
            # 这时 (a, d) 是含最小值区间
            # c 是区间 (a,d) 的靠近 d 的黄金分割点
            # 仍用 a, b 表示含最小值区间，d 表示靠近 b 的黄金分割点
            d, b, yd = c, d, yc
        else
            # 这时 (c, b) 是含最小值区间
            # d 是 (c, b) 的中靠近 c 的黄金分割点
            # 用 b, a 表示含最小值区间，
            # 则 d 相对于 (b, a) 仍是靠近 b 的黄金分割点
            a, b = b, c
        end
    end
    # 返回含最小值的区间。
    return a < b ? (a, b) : (b, a)
end
```

36.11 附录：抛物线法的程序 (*)

这里的程序比节36.5的做法略有改进，对一元单谷函数 `f`，需要输入三个点 $a < b < c$ 使得 $f(a) > f(b)$, $f(c) > f(b)$，迭代次数 n，输出缩减后的

$a < b < c$。在迭代过程中要求每一步产生的新的 $a < b < c$ 都满足 $f(a) > f(b)$, $f(c) > f(b)$，即最小值一定在 (a, c) 内。直接利用 Lagrange 插值公式写出二次多项式，并用导数等于零解出多项式的最小值点，可得从 a, b, c 计算下一个点的迭代公式：

$$x^* = \frac{1}{2} \frac{y_a(b^2 - c^2) + y_b(c^2 - a^2) + y_c(a^2 - b^2)}{y_a(b - c) + y_b(c - a) + y_c(a - b)}$$

产生 x^* 后，利用 a, b, c, x^* 这四个点中找到含最小值的缩小的区间并仍用 a, b, c 三个点表示。节36.5的算法迭代不保证最小值点在拟合抛物线所用的三个点构成的区间内。参见 [Kochenderfer]。

```
## 抛物线法的程序。
function quadratic_fit_search(f, a, b, c, n)
    ya, yb, yc = f(a), f(b), f(c)
    for i in 1:n-3
        # x: 逼近序列中的下一个点
        x = 0.5*(ya*(b^2-c^2)+yb*(c^2-a^2)+yc*(a^2-b^2)) /
            (ya*(b-c) +yb*(c-a) +yc*(a-b))
        yx = f(x)

        if x > b
            if yx > yb
                # 用 a < b < x 作为新的起点
                c, yc = x, yx
            else
                # # 用 b < x < c 作为新的起点
                a, ya, b, yb = b, yb, x, yx
            end
        elseif x < b
            if yx > yb
                # 用 x < b < c 作为新的起点
                a, ya = x, yx
            else
                # 用 a < x < b 作为新的起点
                c, yc, b, yb = b, yb, x, yx
```

```
            end
        end
    end # for i

    # 返回 a < b < c, 使得 f(a)>f(b), f(c)>f(b) .
    return (a, b, c)
end
```

36.12 附录：Brent 法程序 (*)

R 语言程序：

```
brent <- function(f, lower, upper){
  ## 初始化，先生成三个点
  x1 <- lower; y1 <- f(x1)
  x2 <- upper; y2 <- f(x2)
  x3 <- (x1 + x2)/2; y3 <- f(x3)

  eps <- .Machine$double.eps^0.25
  while(abs(x2 - x1) > eps){
    xs <- ( x1*y3*y2/(y1-y3)/(y1-y2) +
            x3*y1*y2/(y3-y1)/(y3-y2) +
            x2*y1*y3/(y2-y1)/(y2-y3) )
    if(y3*y1 < 0){ # 含根区间在 (x1, x3) 中
      if(xs > x1 && xs < x3) { # 近似根在含根区间中
        x4 <- xs
      } else {
        x4 <- (x1 + x3)/2
      }
    } else { # 含根区间在 (x3, x2) 中
      if(xs > x3 && xs < x2){
        x4 <- xs
      } else {
```

```
        x4 <- (x3 + x2)/2
      }
    }
    y4 <- f(x4)
    x1 <- x2; y1 <- y2
    x2 <- x3; y2 <- y3
    x3 <- x4; y3 <- y4
  }
  x3
}
```

习题

习题 1

设一元函数 $f(x) = \frac{4}{3}\log(1+x) - x$ 定义在 $(0,1)$ 中，先求其最大值点的精确值，然后编写 R 程序，分别用 0.618 法和二分法求其最大值点, 比较 5 次和 10 次迭代后，两种方法的绝对误差大小。

习题 2

对例36.1，编写程序，给定 $n, \bar{X}, S, U, 1-\alpha$ 后计算 K 的 $1-\alpha$ 置信下限。

习题 3

对二次多项式 $\varphi(x) = \beta_0 + \beta_1 x + \beta_2 x^2$，$\beta_2 > 0$, $\beta_0, \beta_1, \beta_2$ 都未知，若已知 $a < c < b$ 和 $y_a = \varphi(a)$, $y_c = \varphi(c)$, $y_b = \varphi(b)$, 求 $\varphi(\cdot)$ 的最小值点。

习题 4

对二次多项式函数 $\varphi(x) = \beta_0 + \beta_1 x + \beta_2 x^2$，设 $\beta_2 > 0$, $\beta_0, \beta_1, \beta_2$ 未知。若已知 $x_1 \neq x_2$ 和 $y_1 = \varphi(x_1)$, $y_2 = \varphi(x_2)$，$y_1' = \varphi'(x_1)$，求 $\varphi(x)$ 的最小值点。

习题 5

对二次多项式函数 $\varphi(x) = \beta_0 + \beta_1 x + \beta_2 x^2$，设 $\beta_2 > 0$, $\beta_0, \beta_1, \beta_2$ 未知。若已知 $x_1 \neq x_2$ 和 $y_1 = \varphi(x_1)$, $y_1' = \varphi'(x_1)$，$y_2' = \varphi'(x_2)$，求 $\varphi(x)$ 的最小值点。

习题 6

设 $f(x)$ 在 (a, b) 内为凸函数且 x^* 为 $f(x)$ 的局部最小值点，证明 $f(x)$ 在 (a, b) 是单谷的。

Chapter 37

无约束优化方法

37.1 分块松弛法

对多元目标函数 $f(x), x \in \mathbb{R}^d$, 可以反复地分别沿每个坐标轴方向进行一维搜索，称为坐标循环下降法。按这种方法进行引申，可以把 x 的分量分成若干组，每次固定其它分量不变，针对某一组分量进行优化，然后轮换其它组进行优化，这样的方法叫做**分块松弛法**。这种方法常常得到比较简单易行的算法，只不过其收敛速度不一定是最优的。

例 37.1 (k 均值聚类)**.** 设在 $\mathbb{R}^d$ 中有 n 个点 $x^{(i)}, i = 1, \dots, n$，要把这 n 个点按照距离分为 k 个类。设 $C = \{C_1, \dots, C_k\}$, C_j 是分到第 j 个类中的点的集合，$\mu = \{\mu_1, \dots, \mu_k\}$ 是各类的中心（该类中的点的均值）, 聚类问题就是求 C 和 μ 使得

$$f(\mu, C) = \sum_{j=1}^{k} \sum_{x^{(i)} \in C_j} \|x^{(i)} - \mu_j\|^2 \tag{37.1}$$

最小。

如果已知 μ，则对每个点计算该点到每个 μ_j 的距离，把该点归类到距离最近的类中。如果已知 C，则 μ_j 取第 j 类所有点的平均值。

在 μ 和 C 都未知的情况下求 $f(\mu, C)$ 的最小值点，可以使用分块松弛法。作为初始中心，可以随机地选 k 个不同的 $x^{(i)}$ 点当作初始的 $\mu_j, j = 1, \dots, k$，但

是最好能让这 k 个初始中心相互距离远一些。然后，轮流地，先把所有点按照到类中心的距离分配到 k 个类中，再重新计算类中心 $\{\mu_j\}$, 如此重复直到结果不再变动。这个算法简单有效。

※※※※※

37.2 梯度法

37.2.1 最速下降法

在目标函数 $f(x)$ 一阶可导时，应利用导数（梯度）的信息，向负梯度方向搜索前进，使得每一步的目标函数值都减小。最速下降法沿负梯度方向搜索找到函数值下降最多的步长，基本的算法为:

最速下降法

取初值 $x^{(0)} \in \mathbb{R}^d$, 置 $t \leftarrow 0$

until (迭代收敛) {

求 $g^{(t)} \leftarrow -\nabla f(x^{(t)})$

求最优步长 $\lambda_t > 0$ 使得 $f(x^{(t)} + \lambda_t g^{(t)}) = \min\limits_{\lambda>0} f(x^{(t)} + \lambda g^{(t)})$

$x^{(t+1)} \leftarrow x^{(t)} + \lambda_t g^{(t)}$

}

输出 $x^{(t+1)}$ 作为极小值点

在算法中，“迭代收敛” 可以用梯度向量长度小于某个预定值，或者两次迭代间函数值变化小于某个预定值来判断。迭代中需要用某种一维搜索方法求步长 λ_t。

在适当正则性条件下，最速下降法可以收敛到局部极小值点并且具有线性收敛速度。但是，最速下降法连续两次的下降方向 $-\nabla f(x^{(t)})$ 和 $-\nabla f(x^{(t+1)})$ 是正交的，这是因为在第 t 步沿 $-\nabla f(x^{(t)})$ 搜索时找到点 $x^{(t+1)}$ 时已经使得函数值在该方向上不再变化，下一个搜索方向如果与该方向不正交就不是下降最快的方向。这样，最速下降法收敛速度比较慢。所以，在每次迭代的一维搜索时，可以不要求找到一维搜索的极小值点，而是函数值足够降低就结束一维搜索。见沃尔夫准则。

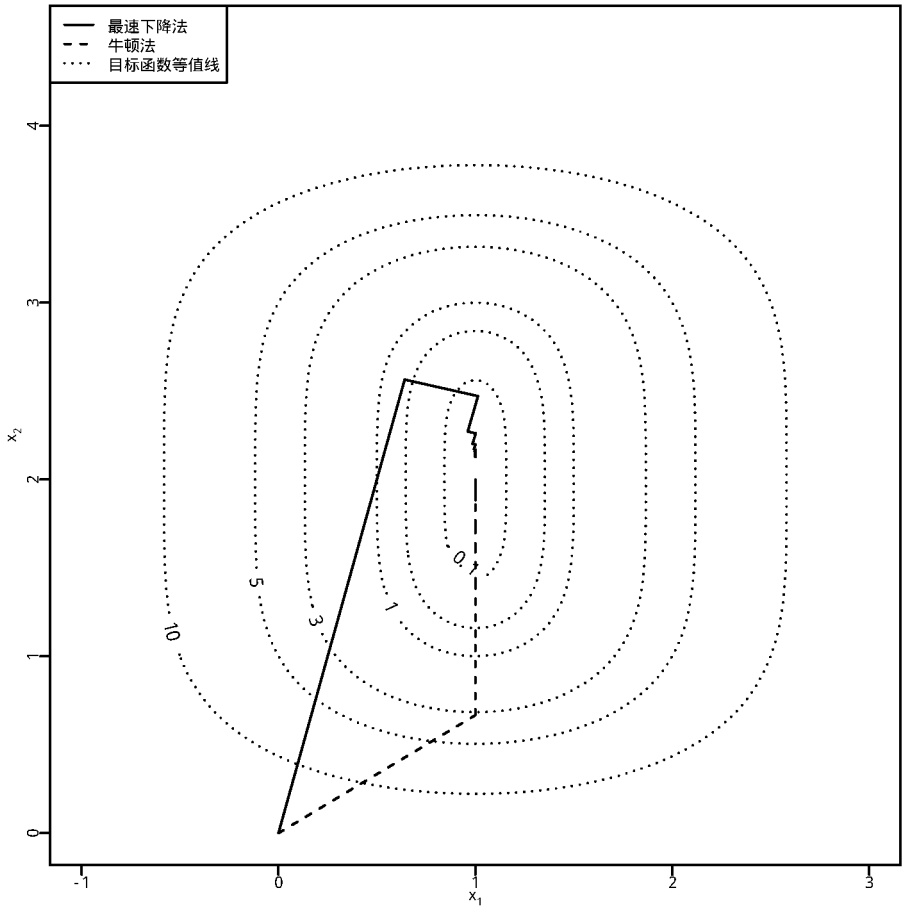


图 37.1: 最速下降法和牛顿法

例 37.2. 求解无约束最优化问题

$$\underset{(x_1,x_2)\in\mathbb{R}^2}{\operatorname{argmin}}\ 4(x_1-1)^2+(x_2-2)^4.$$

显然，函数有全局最小值点 $x^*=(1,2)$。

用最速下降法求解。易见

$$\nabla f(x)=\left(8(x_1-1),\ 4(x_2-2)^3\right)^T,$$

假设从点 $x^{(0)}=(0,0)$ 出发，函数的等值线图以及迭代轨迹见图37.1中的实线，
轨迹中部分点坐标如下：

$x^{(1)}=(0.641,2.565),\ x^{(2)}=(1.014,2.471),\ x^{(3)}=(0.962,2.471),\ x^{(4)}=(1.002,2.271),$
$x^{(5)}=(0.986,2.202),\ x^{(6)}=(1.001,2.197),\ \ldots,\ x^{(10)}=(1.001,2.156),$
$x^{(20)}=(1.001,2.127),\ x^{(40)}=(1.001,2.099),\ x^{(60)}=(1.001,2.083),\ x^{(80)}=(1.001,2.074),$
$x^{(100)}=(1.001,2.067),\ x^{(150)}=(1.001,2.055),\ x^{(200)}=(1.001,2.048)$

使用 R 函数 `optim` 求解（不是最速下降法）：

```
f <- function(x) 4*(x[1]-1)^2 + (x[2]-2)^4
optim(c(0,0), f, method="BFGS")
```

结果：

```
$par
[1] 1.000000 2.001276

$value
[1] 2.648515e-12

$counts
function gradient
     121       98

$convergence
```

```
[1] 0

$message
NULL
```

使用 Julia 的 Optim 包求解：

```
using Optim

function f(xvec)
    x, y = xvec
    4*(x-1)^2 + (y-2)^4
end
# 使用 BFGS 方法求解，使用自动微分方法
ores = optimize(f, [0.0, 0.0], Newton(); autodiff = :forward)
@show Optim.minimizer(ores);
```

结果：

```
 * Status: success

 * Candidate solution
    Final objective value:     7.322602e-29

 * Found with
    Algorithm:     Newton's Method

 * Convergence measures
    |x - x'|               = 4.63e-08   0.0e+00
    |x - x'|/|x'|          = 2.31e-08   0.0e+00
    |f(x) - f(x')|         = 2.86e-17   0.0e+00
    |f(x) - f(x')|/|f(x')| = 3.90e+11   0.0e+00
    |g(x)|                 = 3.17e-21   1.0e-08
```

```
 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    10
    f(x) calls:    31
     f(x) calls:   31
     ²f(x) calls:  10

Optim.minimizer(ores) = [1.0, 1.9999999074947545]
```

※※※※※

从例子可以看出，最速下降法的相邻两次搜索方向正交，而且越到最小值点附近接近得越慢。所以，一维搜索不必找到一维的最小值点，而是使得函数值足够降低就可以了。

37.2.2 梯度法和随机梯度法

现代机器学习（统计学习）算法中普遍使用最速下降法的各种变种，称为梯度法，公式为：

$$x^{(t+1)} \leftarrow x^{(t)} + \lambda_t g^{(t)}, \tag{37.2}$$

其中 λ_t 不是通过一维搜索得到而是预先设定，一般取为很小的正数，称为学习速率。如果取 α_t 为常数，则较大的 α_t 使得收敛较快，但有可能跨越最小值点；较小的 α_t 收敛比较稳定但速度太慢。也可以取 $\lambda_t \to 0$，比如取 $\lambda_t = \lambda_0 \rho^t (0 < \rho < 1)$。这避免了两次搜索方向总是正交的问题，在统计学习中可以减少过度拟合。

37.2.3 随机梯度法

许多机器学习算法基于最小化平均损失：

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(y_i; \theta), \tag{37.3}$$

其中 y_i 是独立同分布样本观测，$L(y_i; \theta)$ 是模型参数为 θ 时第 i 个观测对损失函数的贡献，如残差平方，负对数似然值。如果对上述目标函数用梯度法求最

小值点，当样本量 N 特别巨大时，计算每一个 $\frac{\partial}{\partial\theta}L(y_i;\theta)$ 会占用过多的计算资源使得算法不可行。这时，注意到

$$\frac{1}{N}\sum_{i=1}^{N}L(y_i;\theta)\approx E[L(y;\theta)],$$

在迭代的第 t 步，可以从 N 个样本点中随机抽取 $N'\ll N$ 个样本, 记这些抽取的子样本下标集合为 S_t，用

$$\frac{1}{N'}\sum_{i\in S_t}L(y_i;\theta^{(t)})$$

估计(37.3)目标函数，取

$$g^{(t)}=-\frac{1}{N'}\sum_{i\in S_t}\frac{\partial}{\partial\theta}L(y_i;\theta^{(t)}),$$

作迭代

$$\theta^{(t+1)}=\theta^{(t)}+\lambda_t g^{(t)}. \tag{37.4}$$

每一步重新抽样并利用(37.4)进行优化逼近，称这种方法为随机梯度法。当 N 特别巨大时，N' 可以不随 N 的增大而增大，使得梯度法能适用于训练样本量过于巨大的训练数据集。

37.2.4 共轭梯度法 (*)

最速下降法收敛速度较慢，而且越靠近最小值点步长越短，速度越慢。对于一个函数曲面为狭长的山谷形状的目标函数，为了达到山谷的最低点，会沿相互正交的梯度方向反复跳跃。

一种解决的方法是前面所说的不使用最优一维搜索而是使用预先确定的步长或步长序列，还有一种方法是将两步之间的梯度正交，推广到两步之间为 “共轭” 关系，即对某个矩阵 A_t 使得

$$[g^{(t)}]^T A_t g^{(t+1)}=0.$$

这种方法的来源是对目标函数 $f(x)$ 用二次多项式近似：

$$f(x)\approx\frac{1}{2}x^TAx+b^Tx+c,$$

而为了将一个 d 元的凸二次多项式达到最小值点，只要从负梯度方向出发，沿着 d 个相互都关于 A 共轭的方向进行一维搜索就保证达到最小值点。连续可微函数在最小值点附近一般都很接近于一个凸二次多项式函数，所以共轭梯度法能够克服最速下降法在靠近最小值点时收敛变慢的问题。

A_t 矩阵选取最好是海色阵 $\nabla^2 f(x^{(t)})$，但海色阵一般比梯度更难获得，使用海色阵的方法见节37.3和节37.4。共轭梯度法实际的算法并不去求 A_t，而是从 $x^{(1)}$ 的负梯度出发，然后每次用当前 $x^{(t)}$ 处的负梯度和上一步的共轭方向的线性组合作为 t 到 $t+1$ 步的搜索方向。即：

$$\begin{aligned} d^{(1)} &= -g^{(1)}; \\ d^{(t)} &= -g^{(t)} + \beta^{(t)} d^{(t-1)},\ t = 2, 3, \ldots . \end{aligned}$$

对 d 元凸二次多项式，其中的系数 $\beta^{(t)}$ 可以用 A 解出精确公式，使得各个 $d^{(t)}$ 彼此共轭。但是对一般目标函数 $f(x)$，只能使用一些近似的 $\beta^{(t)}$，方法如：

$$\text{Fletcher-Reeves: } \beta^{(k)} = \max\left(0,\ \frac{[g^{(k)}]^T g^{(k)}}{[g^{(k-1)}]^T g^{(k-1)}}\right).$$

$$\text{Polak-Ribière: } \beta^{(k)} = \frac{[g^{(k)}]^T [g^{(k)} - g^{(k-1)}]}{[g^{(k-1)}]^T g^{(k-1)}}.$$

Fletcher-Reeves 共轭梯度法在极小值点附近可以确保收敛。

参见：[Kochenderfer]。

37.2.5 动量方法 (*)

梯度方法在梯度很接近于 0 的地方也会移动很慢。“动量” 方法可以将前面的梯度方向进行保存，与当前的梯度方向作线性组合，从而可以积累朝向一个搜索方向的 “动量”，就好比一个球在比较平坦的曲面滚动，如果下降方向是相对固定的，即使开始时滚动比较慢，积累的朝向正确下降方向的动量也会使得速度变快。算法迭代公式为：

$$\begin{aligned} v^{(k+1)} &= \beta v^{(k)} - \alpha g^{(k)}, \\ x^{(k+1)} &= x^{(k)} + v^{(k+1)}. \end{aligned}$$

其中 $v^{(k)}$ 起到了将以往的梯度进行累积的作用，系数 β 代表这种记忆的衰减因子，α 代表对当前负梯度方向的贡献的衰减。最速下降法会在相互正交的方向上反复转折，而动量方法则会积累并放大正确的下降方向的作用，使得在梯度接近于零的位置的下降方向变快。

动量法的缺点是在靠近最小值点处的前进速度不会下降，可能会跨过最小值点。一种改进（称为 Nesterov 方法）是使用沿上一步相同方向前进一步处梯度，而不是当前点的梯度，这样如果前进一步后的梯度与当前搜索方向变得相反，就可以起到减慢前进速度的作用。迭代公式为：

$$\begin{aligned} v^{(k+1)} =& \beta v^{(k)} - \alpha \nabla f(x^{(k)} + \beta v^{(k)}), \\ x^{(k+1)} =& x^{(k)} + v^{(k+1)}. \end{aligned}$$

动量法还有许多针对学习速率的改进，比如，对于每个自变量分量分别使用不同步长的 Adagrad 算法，Adagrad 的改进 RMSProp 算法，Adadelta 算法，Adam 算法。

37.2.6 超梯度下降方法 (*)

梯度下降方法中的学习速率 α 是决定梯度下降法性能的关键。超梯度下降法将学习速度 α 作为优化任务的自变量之一，使用如下的学习速度：

$$\alpha^{(k)} = \alpha^{(k-1)} + \mu [g^{(k)}]^T g^{(k-1)}, \tag{37.5}$$

其中参数 μ 称为超梯度学习速率。

这个公式是将 α 作为最优化 f 的自变量使用梯度下降方法产生的。令

$$h(\alpha) = f(x^{(k-1)} - \alpha g^{(k-1)}),$$

则

$$h(\alpha^{(k-1)}) = f(x^{(k)}),$$

可计算得

$$h'(\alpha) = \frac{\partial f(x^{(k-1)} - \alpha g^{(k-1)})}{\partial(-g^{(k-1)})}$$

于是

$$h'(\alpha^{(k-1)}) = [\nabla f(x^{(k)})]^T(-g^{(k-1)}) = -[g^{(k)}]^T g^{(k-1)}.$$

然后对 $h(\alpha)$ 使用学习率为 μ 的梯度下降法，即

$$\alpha^{(k)} = \alpha^{(k-1)} - \mu h'(\alpha^{(k-1)}).$$

此方法的优点是克服了梯度下降方法对学习速率的敏感问题。公式(37.5)中的 $g^{(k)}$ 也可以取为其它的下降方向如动量法的方向。

参见：[Kochenderfer] 第 5 章。

37.3 牛顿法

牛顿法利用海色阵和梯度向量求下一步，不需要一维搜索，对于二次多项式函数可以一步得到最小值点。许多光滑的目标函数在靠近极小值点的邻域内可以很好地用凸二阶多项式逼近。对一个存在二阶连续偏导的 d 元函数 $f(x)$, 有如下的泰勒近似

$$f(x) \approx f(x^{(0)}) + \nabla f(x_0)^T(x - x^{(0)}) + \frac{1}{2}(x - x^{(0)})^T \nabla^2 f(x^{(0)})(x - x^{(0)}), \tag{37.6}$$

若 $\nabla^2 f(x^{(0)})$ 为正定矩阵，上式右侧的二次多项式函数的最小值点在

$$x^* = x^{(0)} - \left[\nabla^2 f(x^{(0)})\right]^{-1} \nabla f(x^{(0)})$$

处达到。所以牛顿法取初值 $x^{(0)}$ 后，用公式

$$x^{(t+1)} = x^{(t)} - \left[\nabla^2 f(x^{(t)})\right]^{-1} \nabla f(x^{(t)}),\ t = 0, 1, 2, ... \tag{37.7}$$

进行迭代，直至收敛。收敛的判断准则可以取为 $\|\nabla f(x)\|$ 足够小，或者取为 $|f(x^{(t+1)}) - f(x^{(t)})|$ 足够小。

(37.7)在算法实现时，应将逆矩阵表示改为如下的线性方程组求解：

$$\nabla^2 f(x^{(t)}) d^{(t)} = -\nabla f(x^{(t)}), \quad x^{(t+1)} = x^{(t)} - d^{(t)} \tag{37.8}$$

如果初值接近于最小值点并且目标函数满足一定正则性条件，牛顿法有二阶收敛速度。

牛顿法的性能对海色阵 $H(x)$ 十分依赖。如果 $H(x^{(t)})$ 不满秩，则(37.8)的解不唯一，算法无法进行；如果 $H(x^{(t)})$ 很接近于不满秩，则 $H(x^{(t)})^{-1}\nabla f(x^{(t)})$ 的计算很不稳定，而且会将下一个近似点 $x^{(t+1)}$ 远离 $x^{(t)}$，这很可能会使得逼近失败。另外，从(37.6)看出，海色阵也是函数的局部性质的重要特征。如果 x_* 是一个稳定点，即 $\nabla f(x_*)=0$，海色阵与此处目标函数的曲面形状密切相关。如果 $H(x_*)$ 是正定的，则 x_* 是局部极小值点。如果 $H(x_*)$ 是负定的，则 x_* 是局部极大值点。如果 $H(x_*)$ 既有正特征值又有负特征值，则 x_* 是目标函数的鞍点，不是局部极值点。

使用牛顿法求最小值时，最好的情况是海色阵始终为正定阵，这时函数是严格凸函数，极小值点也是全局最小值点。在线性最小二乘问题中，目标函数的海色阵是 X^TX 矩阵。

如果目标函数非负定但是接近于不满秩，可以考虑给 $H(x^{(t)})$ 加一个对角的正定阵，例如，岭回归就是将 X^TX 替换成 $X^TX+\lambda I$。

例 37.3. 再次考虑例37.2，用牛顿法。

易见 $f(x)$ 的海色阵为

$$\nabla^2 f(x)=\begin{pmatrix} 8 & 0 \\ 0 & 12(x_2-2)^2 \end{pmatrix},$$

用牛顿法，迭代公式为

$$x^{(t)}=x^{(t-1)}-\left[\nabla^2 f(x^{(t-1)})\right]^{-1}\nabla f(x^{(t-1)})=x^{(t-1)}-(x_1-1,\ \frac{1}{3}(x_2-2))^T.$$

迭代过程见图37.1中的虚线。轨迹中部分点坐标如下：

$$x^{(1)}=(1,0.667),\ x^{(2)}=(1,1.111),\ x^{(3)}=(1,1.407),\ x^{(4)}=(1,1.605),$$
$$x^{(5)}=(1,1.737),\ x^{(6)}=(1,1.824),\ \ldots,\ x^{(10)}=(1,1.965),$$
$$x^{(15)}=(1,1.995),\ x^{(20)}=(1,1.999)$$

没有发生最速下降法那样反复折向前进的问题，而且收敛速度相对较快。

※※※※※

37.3.1 阻尼牛顿法

牛顿法在目标函数具有较好光滑性而且海色阵正定时有很好的收敛速度。但是，牛顿法需要目标函数有二阶导数，还需要在每步迭代都计算海色阵的逆矩阵，在

海色阵非正定时可能会失败。

在公式(37.7)中，把其中的海色阵 $\nabla^2 f(x^{(t)})$ 替换成一个保证正定的矩阵 H_t，则 $-H_t^{-1}\nabla f(x^{(t)})$ 是 $f(x)$ 在 $x^{(t)}$ 处的一个下降方向，即当 $\lambda > 0$ 充分小而且 $\nabla f(x^{(t)}) \neq 0$ 时一定有

$$f\left(x^{(t)} - \lambda H_t^{-1}\nabla f(x^{(t)})\right) < f(x^{(t)}),$$

于是可以用一维搜索方法求步长 λ_{t+1} 使得

$$\begin{cases} \lambda_{t+1} = & \operatorname{argmin}_{\lambda>0} f\left(x^{(t)} - \lambda H_t^{-1}\nabla f(x^{(t)})\right), \\ x^{(t+1)} = & x^{(t)} - \lambda_t H_t^{-1}\nabla f(x^{(t)}). \end{cases} \tag{37.9}$$

这样得到了保证函数值每次迭代都下降的算法。

这里正定矩阵 H_t 的选取有各种各样的方法，比如，取 H_t 为单位阵，则(37.9)变成最速下降法。

在(37.9)中取 H_t 为海色阵 $\nabla^2 f(x^{(t)})$，算法就变成增加了一维搜索的牛顿法，称之为**阻尼牛顿法**。其中的步长 λ_{t+1} 可以用一种一维搜索方法来求，最简单的做法是，依次考虑 $\lambda = 1, 1/2, 1/4, ...$，一旦 $f\left(x^{(t)} - \lambda H_t^{-1}\nabla f(x^{(t)})\right) < f(x^{(t)})$ 就停止搜索。阻尼牛顿法也需要海色阵正定，所以实际使用阻尼牛顿法时，如果某步的海色阵不可逆或者 $\left[\nabla f(x^{(t)})\right]^T \left[\nabla^2 f(x^{(t)})\right]^{-1} \nabla f(x^{(t)}) \leq 0$, 可以在此步以 $-\nabla f(x^{(t)})$ 为搜索方向 (最速下降法)，这样修改后可以使得阻尼牛顿法每步都减小目标函数值。

最速下降法、牛顿法、阻尼牛顿法在迭代中遇到不是局部最小值点的稳定点时都不能正常运行。这时需要一些特殊的方法求得下降方向。参见 徐成贤 et al. [2002] §3.2.2。

37.4 拟牛顿法

另外一些构造公式(37.9)中 H_t 的方法可以不用计算二阶偏导数而仅需要计算目标函数值和一阶偏导数值，可以迭代地构造 H_t 或 H_t^{-1}，这样的算法有很多，统称为**拟牛顿法**或**变尺度法**。这样的 H_t 应该满足如下条件:

(1) H_t 是对称正定阵，从而 $d^{(t)} \triangleq -H_t^{-1}\nabla f(x^{(t)})$ 是下降方向 (称 $d^{(t)}$ 为拟牛顿方向)。

(2) H_{t+1} 由 H_t 迭代得到:

$$H_{t+1} = H_t + \Delta H_t,$$

称 ΔH_t 是修正矩阵。

(3) H_{t+1} 必须满足如下拟牛顿条件:

$$H_{t+1}\delta^{(t)} = \zeta^{(t)}. \tag{37.10}$$

其中 $\delta^{(t)} \triangleq x^{(t+1)} - x^{(t)}$, $\zeta^{(t)} \triangleq \nabla f(x^{(t+1)}) - \nabla f(x^{(t)})$。

第二条可以替换成 H_{t+1}^{-1} 可以用 H_t^{-1} 递推得到。前两个条件比较自然，下面解释第三个条件的理由。

设迭代已经得到了 H_t 和 $x^{(t+1)}$，需要给出下一步的 H_{t+1}。把 $f(x)$ 在 $x^{(t+1)}$ 处作二阶泰勒展开得

$$\begin{aligned} f(x) \approx & f(x^{(t+1)}) + \nabla f(x^{(t+1)})^T (x - x^{(t+1)}) \\ & + \frac{1}{2}(x - x^{(t+1)})^T \nabla^2 f(x^{(t+1)})(x - x^{(t+1)}), \end{aligned}$$

这意味着

$$\nabla f(x) \approx \nabla f(x^{(t+1)}) + \nabla^2 f(x^{(t+1)})(x - x^{(t+1)}),$$

在上式中取 $x = x^{(t)}$ 得

$$\nabla f(x^{(t)}) \approx \nabla f(x^{(t+1)}) + \nabla^2 f(x^{(t+1)})(x^{(t)} - x^{(t+1)}),$$

则有

$$\nabla^2 f(x^{(t+1)})(x^{(t+1)} - x^{(t)}) \approx \nabla f(x^{(t+1)}) - \nabla f(x^{(t)}) = \zeta^{(t)},$$

因为 H_{t+1} 是用来代替海色阵的，所以根据上式需要 H_{t+1} 满足拟牛顿条件(37.10)。

37.4.1 BFGS 拟牛顿法

修正矩阵 ΔH_t 有多种取法，其中一种比较高效而稳定的算法称为 BFGS 算法 (Broyden-Fletcher-Goldfarb-Shanno)，在多元目标函数无约束优化问题中被广泛应用。BFGS 算法的修正公式为

$$H_{t+1} = H_t + \frac{\zeta^{(t)}(\zeta^{(t)})^T}{(\zeta^{(t)})^T \delta^{(t)}} - \frac{(H_t\delta^{(t)})\,(H_t\delta^{(t)})^T}{(\delta^{(t)})^T H_t \delta^{(t)}}, \tag{37.11}$$

一般取初值 H_0 为单位阵，这样按(37.9)和(37.11)迭代，如果一维搜索采用精确一维搜索或沃尔夫准则，(37.11)中的分母 $(\zeta^{(t)})^T\delta^{(t)}$ 可以保证为正值，使得 $\{H_t\}$ 总是对称正定矩阵，搜索方向 $-H_t^{-1}\nabla f(x^{(t)})$ 总是下降方向。在适当正则性条件下 BFGS 方法具有超线性收敛速度。详见 徐成贤 et al. [2002] §3.3。

在(37.9)中计算 $H_t^{-1}\nabla f(x^{(t)})$ 要解一个线性方程组，需要 $O(d^3)$ 阶的计算量 (d 是 x 的维数)。事实上，在 BFGS 算法中 H_t^{-1} 可以递推计算而不需直接求逆或求解线性方程组。记 $V_t = H_t^{-1}$, 这些逆矩阵有如下递推公式:

$$V_{t+1} = V_t + \left[1 + \frac{(\zeta^{(t)})^T V_t \zeta^{(t)}}{(\zeta^{(t)})^T \delta^{(t)}}\right] \frac{\delta^{(t)}(\delta^{(t)})^T}{(\zeta^{(t)})^T \delta^{(t)}} - \frac{H_t\zeta^{(t)}(\delta^{(t)})^T + \left[H_t\zeta^{(t)}(\delta^{(t)})^T\right]^T}{(\zeta^{(t)})^T \delta^{(t)}}.$$

37.5 Nelder-Mead 方法

在导数不存在或很难获得时，求解多元函数无约束最小值经常使用 Nelder-Mead 单纯型方法 [Nelder and Mead, 1965]。**单纯型** (simplex) 指在 $\mathbb{R}^d$ 空间中的 $d+1$ 个点作为顶点的图形构成的凸集，比如，在 $\mathbb{R}^2$ 中给定了三个顶点的一个三角形是一个单纯型，在 $\mathbb{R}^3$ 中给定了四个顶点的一个四面体是一个单纯型。单纯型是 $\mathbb{R}^d$ 中最简单的多面体。

Nelder-Mead 方法是一种直接搜索算法，不使用导数或数值导数。算法开始时要找到 $d+1$ 个点 $x_0, x_1, \dots, x_d$ 构成一个单纯型（要求这 $d+1$ 个点不能位于一个超平面内），计算相应的目标函数值 $y_j = f(x_j), j = 0, 1, \dots, d$。然后，进行反复迭代，对当前的单纯型进行变换使得目标函数值变小。单纯型在开始没有找到最小值点附近时可以沿目标函数值下降的方向扩大，以包含最小值点所在区域；可以保持大小不变移动位置，以接近最小值点；在靠近最小值点后可以缩小以获得精确搜索结果。迭代在单纯型变得足够小或者函数值的变化变得足够小时结束，保险起见，如果迭代次数超过一个预定次数就认为算法失败而结束。

选取初值时，可以任意选取初值 x_0，然后其他 d 个点可以从 x_0 出发分别沿 d 个正坐标轴方向前进一段距离得到。初始的单纯型不要取得太小，否则很容易停留在附近的局部极小值点。

在算法迭代的每一步，假设当前单纯型为 $(x_0, x_1, \dots, x_d)$, 相应的函数值为 $(y_0, y_1, \dots, y_d)$，首先从中找到最坏的、次坏的、最好的函数值 y_h, y_s, y_l，如果每次得到新的单纯型都把单纯型 $d+1$ 个点次序重排使其函数值由小到大排

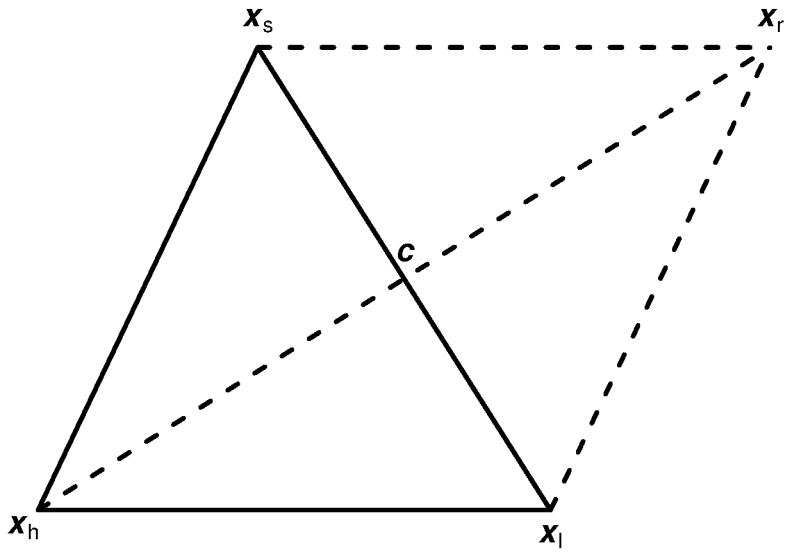


图 37.2: Nelder-Mead 算法反射变换图示，其它变换类似

列，则 $y_l = y_0, y_s = y_{d-1}, y_h = y_d$。然后，计算除最坏点之外的 d 个顶点的重心 $c = \frac{1}{d}\sum_{j\neq h} x_j$。然后对单纯型进行变换以减小函数值。

单纯型的变换有反射、退缩 (contraction)、延伸 (expansion) 和缩小 (shrinkage) 等四种，变化量分别由四个参数 $\alpha > 0$, $\beta \in (0,1)$, $\gamma > \alpha$, $\delta \in (0,1)$ 控制，这四个参数一般取为 $\alpha = 1$, $\beta = \frac{1}{2}$, $\gamma = 2$, $\delta = \frac{1}{2}$。

首先考虑反射是否有效，计算反射点 $x_r \triangleq c + \alpha(c - x_h)$, 这实际是把最坏点 x_h 与中心 c 连线后把连线延长得到点 x_r(见图37.2)，这样点 x_h, c, x_r 在一条直线上但是最坏点 x_h 与反射点 x_r 分别在 c 的两侧。这样沿最坏点的反方向搜索期望能改善目标函数值。如果这时 $y_r \triangleq f(x_r)$ 满足 $y_l \leq y_r < y_s$ （反射点比原次坏点好、但不优于原最好点），则接受 x_r 为本迭代步的新顶点，替换原来的最坏点 x_h 即可。

如果反射点 x_r 的目标函数值比原最好点还小 $(y_r < y_l)$, 则考虑把生成 x_r 的延长线进一步延伸，得到延伸点 $x_e \triangleq c + \gamma(x_r - c)$。令 $y_e = f(x_e)$，如果比反射点进一步改善 $(y_e < y_r)$, 则接受延伸点 x_e 为本迭代步的新顶点，在单纯型中替换原来的最坏点 x_h 即可，本步迭代结束。如果延伸后 x_e 的函数值不如反射点 x_r 的函数值 $(y_e \geq y_r)$, 则接受反射点 x_r 为本迭代步的新顶点，在单纯型

中替换原来的最坏点 x_h 即可，本步迭代结束。

在尝试反射 x_r 后如果发现其不优于次坏点 $x_s(y_r \geq y_s)$, 则反射点不能使用。这时，可以在 x_h, c, x_r 构成的线段上另找一个点，看这个点是否可以接受，这样的变换称为退缩。新的点可以在 c 到 x_r 之间（单纯型外部），也可以在 x_h 和 c 之间（单纯型内部）。这时，如果反射点的函数值 y_r 介于原次坏点与最坏点之间 $(y_s \leq y_r \leq y_h)$, 那么单纯型外部有希望改善目标函数，所以取 $x_c = c + \beta(x_r - c)$(在单纯型外部)；否则，若 y_r 比原最坏点还差 $(y_r > y_h)$，则只能在单纯型内部考虑，取 $x_c = c - \beta(c - x_h)$（在单纯型内部）。得到 x_c 后令 $y_c = f(x_c)$，如果比反射点有改善 $(y_c < y_r)$, 则接受退缩点 x_c 为本迭代步的新顶点，在单纯型中替换原来的最坏点 x_h 即可，本步迭代结束。否则，就要执行第四种变换：缩小。

当反射、延伸、退缩都失败时，执行缩小变换。缩小变换是保持原来的最好点 x_l 不动，其它点都按比例向 x_l 收缩: $x_j \leftarrow x_l + \delta(x_j - x_l), j \neq l$。单纯型缩小后希望其它 d 个点的目标函数值能有所改善。

算法的停止法则可以取为最后的 $d+1$ 个定点处的 $d+1$ 个函数值的样本标准差小于某一预先设定的精度阈值，或迭代次数超过某一界限后宣布算法失败。

Nelder-Mead 方法的收敛性很难确定。

在 R 软件中，用 `optim` 函数求解多元函数无约束优化问题，该函数提供了 BFGS、Nelder-Mead、共轭梯度、模拟淬火等算法。

37.6 模拟退火算法 (*)

退火是一种机械加工技术，加热金属零件到一定温度后进行加工，然后控制温度缓慢地下降到常温。在高温时，分子运动速度较快，在缓慢降温时可以调整机械应力、减少缺陷。

模拟退火算法的名称是利用了“缓慢降温”的做法和“温度”参数，这种算法利用随机性进行全局优化，用类似于 Metropolis-Hasting 的 MCMC 算法那样的方法在目标函数定义域进行随机游动，试图找到全局最小值点。算法从当前位置 $x^{(t)}$ 按照某种抽样分布生成随机数 $\epsilon^{(t)}$, 令 $x' = x^{(t)} + \epsilon^{(t)}$，记

$\Delta y = f(x') - f(x^{(t)})$，以如下概率令 $x^{(t+1)} = x'$:

$$p^{(t+1)} = \begin{cases} 1, & 若\Delta y \le 0, \\ \exp(-\Delta y / T^{(t+1)}), & 若\Delta y > 0. \end{cases}$$

否则令 $x^{(t+1)} = x^{(t)}$。即随机移动一步，如果目标函数值变小，则确认移动；如果目标函数值变大，就以一定正概率移动，否则停留不动，等待下一次随机转移。这里 $T^{(t+1)}$ 是转移概率的参数，类似于退火时的温度，$T^{(t+1)}$ 越大，则 $p^{(t+1)}$ 越大，逆向的随机跳转越频繁，可以用来从局部极小值点跳开；随着迭代的进行，$T^{(t+1)}$ 应该逐步变小，使得逆向随机跳转变稀少，这样迭代能最终收敛。

$T^{(t+1)}$ 应选取为趋于 0 的数列，在一定条件下如果取

$$T^{(t)} = T^{(0)} \frac{\ln 2}{\ln(t+1)},$$

可以确保迭代收敛到全局最小值点，但显然这个速度太慢了。实际应用中往往用

$$T^{(t+1)} = \gamma T^{(t)},$$

其中 $0 < \gamma < 1$，或取 $T^{(t)} = T^{(0)}/t$。

$\epsilon^{(t+1)}$ 的分布可取为多元标准正态分布，或取与各自变量的取值更匹配的一元分布合成多元分布。

参见 [Kochenderfer] 第 8 章。

37.7 遗传算法 (*)

模拟退火算法是利用随机移动来寻找全局最优点。遗传算法是另一类算法的典型代表，它产生多个搜索点，利用这些搜索点以及搜索点之间的交互去接近全局最优点。

遗传算法利用了生物学中择优配对、交叉遗传、遗传突变的思想使得整个种群（搜索点集合）不断进化（接近全局最优点）。设要求 $\min_{x\in\mathbb{R}^d} f(x)$，算法的核心包括：

- 初始化。可以预先获得包含最优点的范围，比如一个超长方体，然后在其中均匀产生 m 个初始的搜索点 $x^{(i)}$, $i = 1, 2, \dots, m$。也可以用多元正态

分布抽样，方差阵用对角阵。也可以用重要抽样思想，使用对于可能包含最优点的区域的先验知识构造初始搜索点的抽样分布。每个搜索点称为一个“染色体”。

- 每一步先选出 m 对父本，从父本用交叉遗传产生下一代的搜索点。在选父本时，要体现择优配对的思想，比如，仅从 m 个现有搜索点中的 k 个表现最优的点（目标函数值最小）中抽取父本；或者，令被抽取为父本的概率，与目标函数值反向相关，等等。
- 对选出的 m 对父本中的每一对，如 $x^{(i)}, x^{(j)}$，进行染色体交叉，获得一个子代的搜索点（染色体）。交叉的方法有多种，比如 d 维的每一分量随机地从两个父本对应分量中任选一个。
- 经过交叉遗传获得下一代染色体后，如果没有别的步骤，这个种群（搜索点集合）就会仅在有限个点上活动，不可能搜索全定义域。所以，还要有一个“突变”步骤，使得搜索点能够转移到附近的位置。一般是选择对每个搜索点加多元标准正态噪声乘以一个小的步长。

详见 [Kochenderfer] 第 9 章。这种利用多个搜索点的方法还有许多，例如“粒子蜂拥”方法，记录每一个搜索点的位置、速度、历史最优点，更新速度为倾向于自己的历史最优点和群体的历史最优点。“萤火虫”方法，是每个搜索点都逐次被每一个优于自己的点吸引。

这些利用群体随机变化寻找全局最优的方法有利用查找比较广泛的定义域，但是收敛到最小值点比较慢，所以可以配合一些局部搜索算法；可以对每一个搜索点都作局部搜索，并更新搜索点和目标函数值；也可以对每一个搜索点都作局部搜索后，仅更新其搜索得到的局部更优的目标函数值，但并不更新搜索点本身，这种做法更能避开局部最小点的吸引。

习题

习题 1

编写牛顿法的程序，当没有输入偏导数函数时，程序用数值微分方法近似计算偏导数。

习题 2

编写 BFGS 方法的程序，当没有输入偏导数函数时，程序用数值微分方法近似计算偏导数。

习题 3

编写 Nelder-Mead 算法的程序。

习题 4

设 $f(x)$ 偏导数连续，证明最速下降法的两个方向 $g^{(t)}$ 和 $g^{(t+1)}$ 正交。

Chapter 38

约束优化方法

考虑约束最优化问题。约束最优化问题比无约束优化问题复杂得多，也没有很好的通用解决方法，在这个方面有大量研究。现有的方法大致上有如下三类：

- 把约束问题转化为一系列无约束问题，用这些无约束问题的极小值点去逼近约束极小值点，称这样的方法为序列无约束优化方法 (SUMT)，如惩罚函数法、乘子罚函数法。
- 在迭代的每一步用一个二次函数逼近目标函数，用线性约束近似一般约束，构造一系列二次规划来逼近原问题，称这样的方法为序列二次规划 (SQP) 法。
- 每次迭代找到可行下降方向，保证迭代始终处于可行域内，这样的方法称为可行方向法。

38.1 约束的化简

最简单的一些约束可以通过参数变换转化成无约束问题。例如，对 $f(x)$ 在 $x \in (0,\infty)$ 求最小值，可令 $x = e^t$, $h(u) = f(e^u), u \in (-\infty,\infty)$，若求得 $\operatorname{argmin}_{u\in\mathbb{R}} h(u) = u^*$，则 $x^* \triangleq e^{u^*}$ 是 $f(x)$ 在 $(0,\infty)$ 的最小值点。

类似地，对 x 限制在有限开区间 (a,b) 的情形，可以作变换

$$x = a + \frac{b-a}{1+e^{-u}},\ u \in (-\infty, \infty).$$

对 x 限制在有限闭区间 $[a,b]$ 的情形，可以做变换

$$x = a + (b-a)\sin^2 u,\ u \in (-\infty, \infty).$$

也可以不使用三角函数，而是作变换

$$x = \frac{a+b}{2} + \frac{b-a}{2}\frac{2u}{1+u^2},\ u \in (-\infty, \infty).$$

对 $x = (x_1, x_2, x_3)$ 限制为 $0 \le x_1 \le x_2 \le x_3$ 的情形，可以做变换

$$x_1 = u_1^2, \quad x_2 = u_1^2 + u_2^2, \quad x_3 = u_1^2 + u_2^2 + u_3^2,\ (u_1, u_2, u_3) \in \mathbb{R}^3.$$

对于 $x = (x_1, x_2)$ 限制为 $x_1 > 0, x_2 > 0, x_1 + x_2 < 1$ 的情形，可以做变换

$$x_1 = \frac{e^{t_1}}{1+e^{t_1}+e^{t_2}}, \quad x_2 = \frac{e^{t_2}}{1+e^{t_1}+e^{t_2}},\ (t_1, t_2) \in \mathbb{R}^2.$$

这些变换可以把原来的约束优化问题转化为无约束优化问题，或者减少约束个数。

对于等式约束，如果有显式解，可以从中解出部分自变量，简化原来的问题。

38.2 线性规划的单纯形法介绍

对约束最优化问题(35.1)，如果 $f(x), c_i(x)$ 都是线性函数（可以带有截距项），称这样的问题为**线性规划**问题。等式约束 $c_i^T x = b_i$ 可以写成两个不等式约束 $c_i^T x \ge b_i$ 和 $(-c_i)^T x \ge b_i$，所以线性规划问题可以写成

$$\begin{aligned} &\operatorname*{argmin}_{x \in \mathbb{R}^d} c^T x, \quad \text{s.t.} \\ &Ax \ge b, \end{aligned} \tag{38.1}$$

其中 $c \in \mathbb{R}^d$, A 为 $m \times d$ 矩阵 $(m < d)$，$b \in \mathbb{R}^m$，$Ax \ge b$ 中的大于等于号表示对每一行成立。

线性规划问题(38.1)不够通用，如下的一般形式更容易在实际建模中使用：

$$\begin{aligned} &\operatorname*{argmin}_{x\in\mathbb{R}^d} c^T x, \quad \text{s.t.} \\ &A_{\mathrm{GE}} x \geq b_{\mathrm{GE}}, \\ &A_{\mathrm{LE}} x \leq b_{\mathrm{LE}}, \\ &A_{\mathrm{EQ}} x = b_{\mathrm{EQ}}. \end{aligned} \tag{38.2}$$

在线型规划的文献中还经常使用如下的标准形式：

$$\begin{aligned} &\operatorname*{argmin}_{x\in\mathbb{R}^d} c^T x, \quad \text{s.t.} \\ &Ax \leq b, \\ &x \geq 0. \end{aligned} \tag{38.3}$$

其中 A 为 $m \times d$ 行满秩矩阵，$m < d$。

在讨论求解时，对标准形式(38.3)，可以增加松弛变量 $s = b - Ax \geq 0$，使得问题变成如下等式约束形式：

$$\begin{aligned} &\operatorname*{argmin}_{x\in\mathbb{R}^d} c^T x, \quad \text{s.t.} \\ &Ax = b, \\ &x \geq 0. \end{aligned} \tag{38.4}$$

这四种形式都可以互相转化，本质上是等价的。

问题(38.4)的可行域是凸多面体，边界为超平面（当 $d = 2$ 时为线段或射线），凸多面体的顶点，在二维平面和三维空间就是初等几何学中的顶点，在高维空间，定义为不在可行域的任意两个点之间的点。

约束最小值点如果存在，可能在可行域的什么位置出现？

可行域的内点一定不是最优点，因为目标函数 $c^T x$ 沿负梯度方向 $-c^T$ 下降，内点可以向负梯度方向移动一个小的步长，不超出可行域但使得目标函数下降。所以约束最小值如果存在，一定出现在可行域的边界。

可行域的边界超平面图形（比如，三维空间中的四面体，边界是平面三角形）的内点（比如，三角形内部的点）如果是约束最小值点，这个超平面的法线方向必须是 c，因为如果这个超平面的法线方向不是 c，内点一定可以沿 $-c$ 在这个超平面的投影方向移动一个小的步长，使得目标函数变小。

同理可行域的边界超平面图形的棱线（比如，四面体的边）的内部的点如果是约束最小值点，则此棱线必垂直于 c。

约束最小值如果存在，还可能存在于形状为 d 维凸多面体的可行域的顶点处，称这些顶点为基本可行点，基本可行点只有有限个；如果线性规划问题存在约束最小值点，则可行域至少有一个基本可行点，而且基本可行点中至少有一个是最优点（可能存在多个最优点，甚至于无穷多个最优点）。根据上面的讨论，如果边界超平面图形内部、棱线内部有最优值，则相应的各个顶点也都是最优点，所以只需要在这有限个顶点中寻找最优值。线性规划的问题是当问题规模很大时，如何不利用穷举法快速找到目标函数值全局最优的顶点。

问题(38.4)的顶点，其结构一定满足 $x = (x_1, \ldots, x_d)$ 的分量中有 $d - m$ 个 0，记这些分量的下标集合为 V，其它分量的下标集合为 B，在给定取 0 的分量的条件下解方程 $Ax = b$ 可以求出 B 中的分量的值，这些值大于等于零，如此可以求出顶点坐标。但是，满足这样条件的点不一定都是顶点，而且不容易判断是否顶点。

单纯形法是线性规划问题的经典求解方法，算法先初始化，找到一个基本可行点（顶点），这可以通过构造一个辅助的初始可行点已知的线性规划求解得到，用这种方法得到初始可行点，需要对原始线性规划问题进行适当的变换才能继续迭代。有了初始可行点后，单纯形算法在相邻基本可行点之间迭代，每次在下标集合 B 和 C 之间交换一个下标并使得这样交换产生的目标函数下降最多，这样每次迭代都使得目标函数值下降，一般可以很快收敛。

对于线性规划，满足 KKT 条件的点一定是约束全局最小值点，所以可以用 KKT 条件检测每一个迭代经过的顶点是否最优点。只要问题有可行点，且在可行域内目标函数有下界，则单纯型法一定能找到约束的全局最小值点。

单纯型算法是针对(38.4)问题求解的，其它几种形式可以转换成(38.4)的形式，求解软件一般能自动进行转换。

R 的 boot 包中提供了 `simplex()` 函数，可以用来求解线性规划问题。

例 38.1. 考虑如下的线性规划问题：

$$\begin{cases} \underset{(x_1,x_2)\in\mathbb{R}^3}{\operatorname{argmin}} \quad x_1 + x_2, \text{ s.t.} \\ 2x - y \geq 0, \\ -x + 2y \geq 3, \\ 3x - y \leq 13. \end{cases}$$

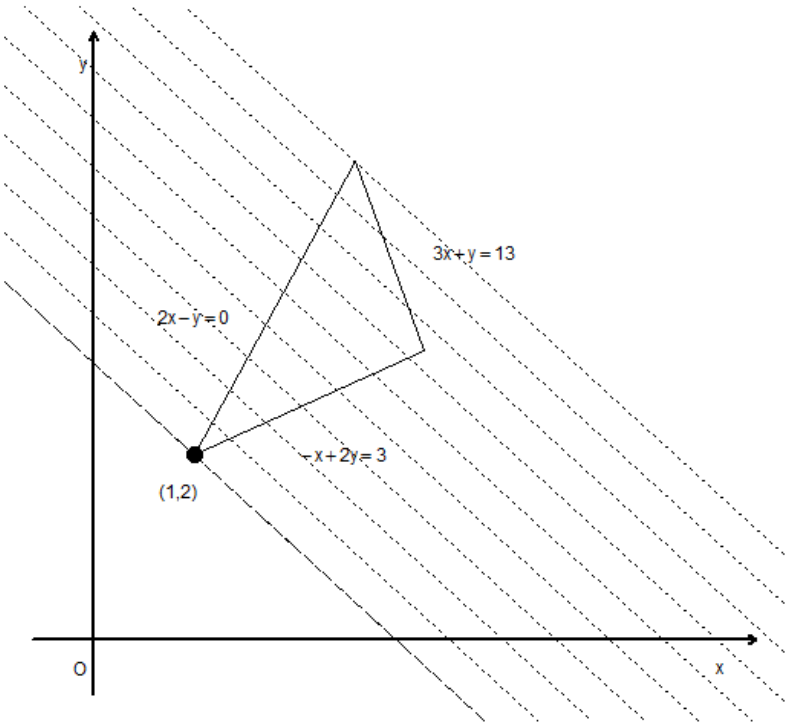


图 38.1: 例38.1图形。三角形是可行域，虚线是目标函数的等值线，圆点是最小值点。

用 R 的 `boot::simplex()` 求解：

```
boot::simplex(
  a = c(1, 1),            # 目标函数系数
  A1 = rbind(c(3, -1)),
  b1 = 13,                # <= 约束
  A2 = rbind(c(2, -1),
             c(-1, 2)),
  b2 = c(0, 3),           # >= 约束，所有自变量非负约束为默认
  A3 = NULL, b3 = NULL)   # = 约束
```

```
##
## Linear Programming Results
##
## Call : boot::simplex(a = c(1, 1), A1 = rbind(c(3, -1)), b1 = 13, A2 = rbind(c(2
##     -1), c(-1, 2)), b2 = c(0, 3), A3 = NULL, b3 = NULL)
##
## Minimization Problem with Objective Function Coefficients
## x1 x2
##  1  1
##
##
## Optimal solution has the following values
## x1 x2
##  1  2
## The optimal value of the objective  function is 3.
```

最小值点为 $(1,2)$，最小值为 3。

使用 Julia 的 JuMP 包求解，利用 GLPK 包作为实际求解的后端：

```
using JuMP, GLPK
# 建立最优化模型框架
om = Model(GLPK.Optimizer)
# 定义优化问题要求解的变量
@variable(om, x)
```

```
@variable(om, y)
# 定义优化目标
@objective(om, Min, x + y)
# 定义三个约束
@constraint(om, 2x - y >= 0)
@constraint(om, -x + 2y >= 3)
@constraint(om, 3x - y <= 13)
# 显示定义的优化问题
print(om)
# 进行优化求解
JuMP.optimize!(om)
# 检查是否得到最优解
println(termination_status(om))
## OPTIMAL
println(" 最优目标函数值: ", objective_value(om))
## 最优目标函数值: 3.0
println(" 最优解: x=", value(x), " y=", value(y))
## 最优解: x=1.0 y=2.0
```

38.3 仅含线性等式约束的情形 (*)

如果约束仅包含线性方程组，可以把问题(35.1)用矩阵形式写成

$$\begin{cases} \underset{x\in\mathbb{R}^d}{\operatorname{argmin}} f(x), \text{ s.t.} \\ A^T x = b, \end{cases} \tag{38.5}$$

其中 A 为 $d \times p$ 矩阵, 各列为 $a_i, i = 1, \dots, p$，$b = (b_1, \dots, b_p)^T$, 即问题仅有 p 个线性约束 $a_i^T x = b_i, i = 1, 2, \dots, p$。设 $p < d$ 且 A 列满秩，当 A 规模很小时，可以预先用消元法把 x 的 p 个分量用其余的 $d-p$ 个线性表出，把问题转化为 $d-p$ 维的无约束优化问题。

下面讨论当 A 规模较大时的处理方法。

记 $\mu(A)$ 为 A 的各列在 $\mathbb{R}^d$ 中张成的线性空间, 即

$$\mu(A) \triangleq \{x \in \mathbb{R}^d : x = Au, u \in \mathbb{R}^p\} = \{x \in \mathbb{R}^d : x = \sum_{i=1}^{p} c_i a_i,\ c_1, \dots, c_p \in \mathbb{R}\}$$

则 $\mathbb{R}^d$ 可以分解为 $\mu(A)$ 与 $\mu(A)$ 的正交补空间

$$\mu(A)^\perp \triangleq \{x \in \mathbb{R}^d : A^T x = 0\}$$

的直和:

$$\mathbb{R}^d = \mu(A) \oplus \mu(A)^\perp.$$

设 A 有 QR 分解

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix} = (Q_1, Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R,$$

其中 Q 为 $d \times d$ 正交阵，R 为 $p \times p$ 上三角阵，Q_1 为 $d \times p$ 矩阵, 易见 Q_1 各列构成 $\mu(A)$ 的标准正交基，Q_2 各列构成 $\mu(A)^\perp$ 的标准正交基。若 x 满足约束 $Ax = b$, 设

$$x = Q_1 u \oplus Q_2 v,\ u \in \mathbb{R}^p,\ v \in \mathbb{R}^{d-p},$$

则

$$A^T x = R^T Q_1^T x = R^T u = b$$

可得 $u = R^{-T} b$(简记 $(R^T)^{-1}$ 为 R^{-T})，于是

$$x = Q_1 R^{-T} b + Q_2 v,\ v \in \mathbb{R}^{d-p}, \tag{38.6}$$

只要对目标函数 $f(Q_1 R^{-T} b + Q_2 v)$, $v \in \mathbb{R}^{d-p}$ 求解无约束最优化问题即可。

在 x 的分解式(38.6)中，易见

$$\begin{aligned} P \triangleq & Q_2 Q_2^T = I_n - Q_1 Q_1^T = I_n - A R^{-1} R^{-T} A^T = I_n - A(R^T R)^{-1} A^T \\ = & I_n - A(A^T A)^{-1} A^T, \end{aligned}$$

这是向正交补空间 $\mu(A)^\perp$ 投影的投影矩阵，所以(38.6)中属于正交补空间 $\mu(A)^\perp$ 的部分 $Q_2 v$，在已知 x 时可以用投影表示为 Px。后文中的投影梯度法利用了这样的思想。

如果 $x^{(t)}$ 是一个可行点，为了使得 $x^{(t+1)} = x^{(t)} + d$ 也是可行点，需要 $A^T d = 0$。注意问题(38.5)是一般约束问题(35.1)仅含等式约束的版本，这里 $c_i(x) = a_i^T x$, a_i 是 A 的第 i 列，可见 $\nabla c_i(x) = a_i$，上述对 d 的要求 $A^T d = 0$ 可以表示为

$$d^T \nabla c_i(x) = 0, i = 1, \dots, p,$$

这个要求可以推广到非线性等式约束的情形，即为了 $x^{(t)} + d$ 继续满足等式约束，搜索方向 d 必须和等式约束函数的梯度都正交，否则会使得 $c_i(x)$ 的值改变。

38.4 线性约束最优化方法 (*)

在约束最优化问题中，如果所有约束都是线性函数，这样的问题相对比较容易处理，但是也能体现约束优化问题的困难。如下的约束优化问题称为**线性约束优化问题**:

$$\begin{cases} \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} f(x), \text{ s.t.} \\ a_i^T x = b_i, \ i = 1, \dots, p, \\ a_i^T x \le b_i, \ i = p+1, \dots, q, \end{cases} \tag{38.7}$$

其中 $a_i \in \mathbb{R}^d, i = 1, \dots, p+q$, $b_i \in \mathbb{R}, i = 1, \dots, p+q$。

前面已经讨论了仅有线性等式约束的情形，下面讨论含有线性不等式约束的情形。

38.4.1 投影梯度法

从(38.6)看出，当约束为线性等式约束时，搜索时应该在和各 a_i 正交的方向上搜索。这提示我们，对于包含不等式约束的优化问题(38.7), 如果 $x^{(t)}$ 是一个可行点，希望找到使得函数值下降的可行方向，只需考虑所有等式约束和起作用的不等式约束。假设 $x^{(t)}$ 处起作用的不等式约束下标为 $p+1, \dots, p+r$，记

$$\mathcal{B} = \{d \in \mathbb{R}^d : \ a_i^T d = 0, \ i = 1, \dots, p+r\} \tag{38.8}$$

设负梯度 $-\nabla f(x^{(t)})$ 投影到 $\mathcal{B}$ 中得到 d，如果 $d \neq 0$, d 就是一个可行下降方向, 沿方向 d 搜索使得函数值下降且保持所有 $p+q$ 个约束成立。有了可行下

降方向 d 以后，从 $x^{(t)}$ 出发沿 d 方向进行一维搜索，得到下一个近似极小值点 $x^{(t+1)}$，如此迭代逼近。

如果投影得到的 $d=0$, 必存在 $\lambda_i^{(t)}, i=1,\dots,p+r$, 满足

$$\sum_{i=1}^{p+r}\lambda_i^{(t)}a_i=-\nabla f(x^{(t)}).$$

解出 $\lambda_i^{(t)}, i=1,\dots,p+r$, 如果 $\lambda_i^{(t)}\geq 0, i=p+1,\dots,p+r$, 只要令 $\lambda_i^{(t)}=0, i=p+r+1,\dots,p+q$, $\lambda^{(t)}=(\lambda_1^{(t)},\dots,\lambda_{p+q}^{(t)})^T$, 则 $(x^{(t)},\lambda^{(t)})$ 是问题(38.7)的KKT 对，算法不再继续，只要设法判断 $x^{(t)}$ 是否极小值点。

如果 $d=0$ 但是解出的 $\lambda_i^{(t)}, i=p+1,\dots,p+r$ 中有负值，则设 $\lambda_{i_0}^{(t)}=\min\{\lambda_i^{(t)}, i=p+1,\dots,p+r\}$, 令

$$\mathcal{B}'=\{d\in\mathbb{R}^d:\ a_i^Td=0,\ i=1,\dots,p+r, i\neq i_0\}$$

令 $\tilde{d}$ 为 $-\nabla f(x^{(t)})$ 向 $\mathcal{B}'$ 的投影，则 $\tilde{d}$ 一定是可行下降方向。

这种方法称为**投影梯度法**，是一种比较早期的可行方向法，具体算法略。投影梯度法以及与之类似的简约梯度法优点是比较简单，缺点是仅利用梯度信息而没有试图采用高阶逼近，收敛速度慢。

38.4.2 简约梯度法

在一般线性约束优化问题(38.7)中，一般无约束的分量 x_i 可以写成 $x_j=x_j^+-x_j^-$, 其中 $x_j^+=\max(x_j,0)$, $x_j^-=-\min(x_j,0)$，这样，可以设所有的分量都满足 $x_j\geq 0$。对不等式约束 $a_i^Tx\geq b_i$, 可以引入新自变量 $z_i\geq 0$ 使得 $a_i^Tx-z_i=b_i$, 这样，可以设所有自变量非负，所有的其它约束都是等式约束，于是问题(38.7)可以写成

$$\begin{cases}\underset{x\in\mathbb{R}^d}{\operatorname{argmin}} f(x),\ \text{s.t.}\\ Cx=b,\\ x\geq 0.\end{cases}\tag{38.9}$$

设 C 为 $p\times d$ 矩阵，并设 C 行满秩，不妨设 $C=(C_B,C_N)$，其中 C_B 为 $p\times p$ 满秩方阵，把 x 拆分为 (x_B,x_N), 由 $Cx=b$ 可解出

$$x_B=x_B(x_N)=C_B^{-1}b-C_B^{-1}C_Nx_N,\ x_N\in\mathbb{R}^{d-p},$$

问题可简化为

$$\begin{cases} \underset{x_N\in\mathbb{R}^{d-p}}{\operatorname{argmin}} h(x_N) = f(x_B(x_N), x_N), \text{ s.t.} \\ x_N \geq 0, x_B(x_N) \geq 0. \end{cases}$$

记 $f(x)$ 的梯度前 p 个为 $G_B(x)$, 后 $d-p$ 个为 $G_N(x)$，则目标函数 $h(x_N)$ 的梯度为

$$\nabla h(x_N) = G_N(x) - (C_B^{-1}C_N)^T G_B(x),$$

称为 $f(x)$ 的简约梯度，其中 $x = (x_B(x_N), x_N)$。记 $\tilde{d}_N = -\nabla h(x_N)$, 这是目标函数 $h(x_N)$ 的下降方向。在迭代逼近约束最小值点的过程中，如果负简约梯度方向 $\tilde{d}_N$ 所有分量都大于零或等于零，则它指向 x_N 的分量都增加或不变的方向，关于 x_N 部分是可行下降方向，x_N 部分只要延 $\tilde{d}$ 方向搜索即可；如果 $\tilde{d}_N$ 的某个分量（设为第 j 个）小于零，因为则该分量指向 x_j 减小的方向，有可能使得搜索跳出可行域，所以要把 $\tilde{d}_N$ 的这样的分量修改为 $x_j\tilde{d}_j$, 这样，当 $x_j > 0$ 时，搜索方向是使得 x_j 减少的方向，当 $x_j = 0$ 时，搜索方向使得 x_j 不变从而不会离开可行域。设修改后的 $\tilde{d}_N$ 为 d_N, 对于 x_B 部分，令相应的搜索方向为 $d_B = -C_B^{-1}C_N d_N$，令 $d = (d_B^T, d_N^T)^T$ 为搜索方向，进行一维搜索，搜索要限制在可行域内。在每次迭代中，都重新把 x 拆分为 x_B 和 x_N，x_B 部分要求分量都取正值。利用这样的方法对原目标函数 $f(x)$ 进行迭代搜索，可以保证每次迭代延可行下降方向搜索, 当满足 KKT 条件时停止。这样的方法叫做**简约梯度法**。详见 徐成贤 et al. [2002] §6.1.2。

投影梯度法和简约梯度法优点是比较简单，缺点是仅利用梯度信息而没有试图采用高阶逼近，收敛速度慢。

38.4.3 有效集方法

有效集方法可以看成是投影梯度法的推广。对问题(38.7)，假设已有约束最小值点的近似值 $x^{(t)}$ 是可行点 (满足约束条件)，设 $x^{(t)}$ 处不等式约束中 $c_i, i = p+1, \ldots, p+r$ 是起作用约束，$c_i, i = p+r+1, \ldots, p+q$ 不起作用。如前所述，搜索方向必须与 $a_i, i = 1, \ldots, p+r$ 正交，仍按(38.8)定义 $\mathcal{B}$ 为这些方向的集合。投影梯度法是把负梯度向量投影到线性子空间 $\mathcal{B}$ 上作为搜索方向，有效集方法推广了这种方法，仍要求搜索方向在 $\mathcal{B}$ 中，但不限于负梯度投影方向，即

在第 $t+1$ 步求解如下的仅含线性约束的子问题

$$\begin{cases} \underset{d\in\mathbb{R}^d}{\operatorname{argmin}} f(x^{(t)}+d), \text{ s.t.} \\ a_i^T d = 0,\ i=1,\dots,p+r \end{cases} \tag{38.10}$$

这个子问题可以比较容易地求解。

设解(38.10)得到 $d^{(t)}$。如果 $d^{(t)}=0$，这说明仅含等式约束的子问题

$$\begin{cases} \underset{x\in\mathbb{R}^d}{\operatorname{argmin}} f(x), \text{ s.t.} \\ a_i^T x = b_i,\ i=1,\dots,p+r \end{cases} \tag{38.11}$$

以 $x^{(t)}$ 为最小值点，由定理35.8，可以解出拉格朗日乘子 $\lambda_i^{(t)}, i=1,\dots,p+r$ 使得

$$\sum_{i=1}^{p+r}\lambda_i^{(t)}a_i = -\nabla f(x).$$

如果 $\lambda_i^{(t)}\geq 0, i=p+1,\dots,p+r$, 只要令 $\lambda_i^{(t)}=0, i=p+r+1,\dots,p+q$, $\lambda^{(t)}=(\lambda_1^{(t)},\dots,\lambda_{p+q}^{(t)})^T$，则由定理35.9可知 $(x^{(t)},\lambda^{(t)})$ 是问题(38.7)的 KKT 对，只要判断 $x^{(t)}$ 是不是极小值点。

如果 $d^{(t)}=0$ 但是解出的 $\lambda_i^{(t)}, i=p+1,\dots,p+r$ 中有负值，设 $\lambda_{i_0}^{(t)}=\min\{\lambda_i^{(t)}, i=p+1,\dots,p+r\}$, 只要从子问题(38.10)的约束中删去第 i_0 个重新求解。

如果子问题(38.10)的解 $d^{(t)}\neq 0$，这时如果 $x^{(t)}+d^{(t)}$ 是可行点，满足所有的等式约束和不等式约束（包括不起作用的不等式约束），则令 $x^{(t+1)}=x^{(t)}+d^{(t)}$，否则就把 d 当作一个可行下降方向做一维搜索，一维搜索时要注意不起作用约束也要满足。

具体算法从略，详见 徐成贤 et al. [2002] §6.3.1。

38.5 二次规划问题 (*)

在仅含线性约束的非线性规划问题中，如果 $f(x)$ 是二次多项式函数，称这样的问题为**二次规划**问题，这是最简单的非线性规划问题，这种问题有很多实际应用，另外，非线性约束优化问题的求解也往往需要通过求解一系列二次规划子问题实现。

38.5.1 仅含等式约束的二次规划问题

考虑如下的二次规划问题:

$$\begin{cases} \underset{x\in\mathbb{R}^d}{\operatorname{argmin}} q(x) \triangleq \frac{1}{2}x^T Hx + g^T x, \quad \text{s.t.} \\ A^T x = b, \end{cases} \tag{38.12}$$

其中 H 为 d 阶实对称矩阵，$g \in \mathbb{R}^d$, A 为 $n \times p$ 列满秩矩阵，各列为 $a_i, i = 1, \ldots, p$，$b \in \mathbb{R}^p$。此问题可以用 §38.3的方法求解。

设 A 有如下 QR 分解

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix} = (Q_1, Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R,$$

其中 Q 为 $d \times d$ 正交阵，R 为 $p \times p$ 满秩上三角阵，Q_1 为 $d \times p$ 矩阵, $Q_1^T Q_1 = I_p$, $Q_1^T Q_2 = 0$。满足约束的可行点 x 的通解为

$$x = Q_1 R^{-T} b + Q_2 v, \quad v \in \mathbb{R}^{d-p}, \tag{38.13}$$

令

$$\tilde{q}(v) = q(Q_1 R^{-T} b + Q_2 v) \triangleq \frac{1}{2} v^T \tilde{H} v + \tilde{g}^T v + \tilde{c}, \quad v \in \mathbb{R}^{d-p}, \tag{38.14}$$

其中

$$\tilde{H} = Q_2^T H Q_2, \quad \tilde{g} = Q_2^T g + Q_2^T H Q_1 R^{-T} b,$$

$\tilde{c}$ 为与 v 无关的常数项。只要求解无约束优化问题 $\operatorname{argmin} \tilde{q}(v)$ 得到 v^*，再用(38.13)就可以得到问题(38.12)的最小值点 x^*。

在(38.14)的目标函数 $\tilde{q}(v)$ 中，海色阵 $\tilde{H}$ 如果有负特征值，则 $\tilde{q}(v)$ 的最小值是 $-\infty$，问题无解。事实上，设有 $\lambda < 0$ 以及 $d \neq 0$ 使得 $\tilde{H}d = \lambda d$，取 $v^{(k)} = kd$, 则

$$\tilde{q}(v^{(k)}) = \frac{1}{2}\lambda \|d\|^2 \cdot k^2 + \tilde{g}^T d \cdot k \to -\infty, \quad \text{当} k \to \infty.$$

所以，不妨设 $\tilde{H}$ 为非负定阵。

如果 $\tilde{H}$ 是正定阵，则 $\tilde{q}(v)$ 是严格凸函数，有全局严格最小值点 $v^* = -\tilde{H}^{-1}\tilde{g}$，代入(38.13)可得问题(38.12)的最小值点 x^*。这时，再考虑拉格朗日函数

$$L(x, \lambda) = q(x) - \lambda^T (A^T x - b),$$

来求拉格朗日函数的稳定点，其中 x^* 已知，由定理35.8知拉格朗日乘子 λ^* 满足

$$\begin{aligned}\nabla q(x^*) - A\lambda^* =& 0,\\ A\lambda^* =& Hx^* + g,\end{aligned}$$

由 $A = Q_1 R$ 得

$$R\lambda^* = Q_1^T(Hx^* + g),$$

用回代法解此方程可得拉格朗日乘子 λ^*。

如果 $\tilde{H}$ 仅为非负定阵而不是正定阵，$\tilde{q}(v)$ 仍然是凸函数，最小值点与稳定点等价，稳定点存在当且仅当 $\tilde{H}v = -\tilde{g}$ 有解，这等价于 $\tilde{g}$ 属于 $\mu(\tilde{H})$ （$\mu(\tilde{H})$ 是 $\tilde{H}$ 的各列张成的线性子空间），又等价于

$$(I - \tilde{H}\tilde{H}^+)\tilde{g} = 0, \tag{38.15}$$

其中 $\tilde{H}^+$ 是 $\tilde{H}$ 的加号逆。由定理32.12，当(38.15)成立时 $\tilde{q}(v)$ 有无穷多个最小值点，可以表达为

$$v^* = -\tilde{H}^+\tilde{g} + (I - \tilde{H}^+\tilde{H})u, \quad \forall u \in \mathbb{R}^{n-p},$$

再代入(38.13)可得问题(38.12)的无穷多个最小值点 x^*。

在问题(38.12)规模很大的时候，求解线性方程组计算量很大，可以用共轭梯度法迭代地求解，由于二次目标函数 $q(x)$ 的特殊性，迭代最多只需要 $n-p$ 次就可以收敛到最小值点。参见 徐成贤 et al. [2002] §6.4.1。

38.5.2 含有不等式约束的二次规划问题

考虑如下的二次规划问题:

$$\begin{cases}\underset{x\in\mathbb{R}^d}{\operatorname{argmin}}\, q(x) \triangleq \frac{1}{2}x^T Hx + g^T x, \quad \text{s.t.}\\ a_i^T x = b_i,\ i = 1, \dots, p\\ a_i^T x \le b_i,\ i = p+1, \dots, p+q\end{cases} \tag{38.16}$$

其中 H 为 d 阶实对称矩阵，$g \in \mathbb{R}^d$, $a_i \in \mathbb{R}^d, i = 1, \dots, p+q$, $b_i \in \mathbb{R}, i = 1, \dots, p+q$。

用前面所述关于一般线性约束的有效集方法可以把含有不等式约束的二次规划问题转化为一系列仅含等式约束的二次规划问题。假设经过 t 步迭代得到问题(38.16)的一个可行点 $x^{(t)}$，$x^{(t)}$ 处不等式约束中 $c_i, i=p+1,\dots,p+r$ 是起作用约束，$c_i, i=p+r+1,\dots,p+q$ 不起作用。考虑如下只有等式约束的子问题

$$\begin{cases} \underset{x\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{2}x^THx+g^Tx, \quad \text{s.t.} \\ a_i^Tx=b_i,\ i=1,\dots,p+r \end{cases} \tag{38.17}$$

令 $x=x^{(t)}+d$, 问题(38.17)可以等价地写成

$$\begin{cases} \underset{d\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{2}d^THd+(g+Hx^{(t)})^Td, \quad \text{s.t.} \\ a_i^Td=0,\ i=1,\dots,p+r \end{cases} \tag{38.18}$$

记 $\mathcal{B}=\{d\in\mathbb{R}^d : a_i^Td=0, i=1,\dots,p+r\}$，如果以 $a_i, i=1,\dots,p+r$ 为列向量组成矩阵 A, 则 $\mathcal{B}=\mu(A)^\perp$。

通过前面对仅含等式约束的二次规划问题的讨论可以知道，如果

$$d^THd>0,\ \forall d\in\mathcal{B}\backslash\{0\}, \tag{38.19}$$

则子问题(38.18)存在唯一的全局严格最小值点，设其为 $d^{(t)}$。如果 $d^{(t)}=0$，就说明 $x^{(t)}$ 已经是子问题(38.17)的约束最小值点, 可以解出相应的拉格朗日乘子 $\lambda_i^{(t)}, i=1,\dots,p+r$。如果解出的 $\lambda_i^{(t)}\geq 0, i=p+1,\dots,p+r$, 则令 $\lambda_i^{(t)}=0, i=p+r+1,\dots,p+q$, $\lambda^{(t)}=(\lambda_1^{(t)},\dots,\lambda_{p+q}^{(t)})^T$, 这时 $(x^{(t)},\lambda^{(t)})$ 为原始问题(38.16)的 KKT 对，由于二次目标函数的特殊性以及(38.19)条件可知 $x^{(t)}$ 是问题(38.16)的约束全局严格最小值点。

如果 $d^{(t)}=0$ 但是 $\lambda_{i_0}^{(t)}=\min\{\lambda_i^{(t)} : i=p+1,\dots,p+r\}<0$，把对应于 i_0 的约束从子问题(38.18)中删除，可以证明新的子问题有非零解 $\tilde{d}^{(t)}$，且 $\tilde{d}^{(t)}$ 是原问题(38.16)的可行下降方向。

如果 $d^{(t)}\neq 0$, 则 $d^{(t)}$ 是原问题(38.16)的可行下降方向。

设 $d^{(t)}$ 是原问题(38.16)的可行下降方向。若 $x^{(t)}+d^{(t)}$ 是原问题(38.16)的可行点，直接取为 $x^{(t+1)}$ 继续迭代即可。如果 $x^{(t)}+d^{(t)}$ 超出了(38.16)的可行域，这一定是不起作用的不等式约束中的某些被突破了，可以从 $x^{(t)}$ 出发沿 $d^{(t)}$ 方

向进行线性搜索，由二次目标函数的特点知道线性搜索的最小值点在可行域边界处达到，直接取步长为

$$\alpha_t = \min\left\{ \frac{b_i - a_i^T x^{(t)}}{a_i^T d^{(t)}} :\ p+r+1 \le i \le p+q且a_i^T d^{(t)} > 0 \right\} \qquad (38.20)$$

注意到 $d^{(t)}$ 已经是子问题(38.18)的约束最小值点，所以如果得到的 $\alpha_t > 1$ 只要取 $\alpha_t = 1$。如果 $\alpha_t < 1$，设(38.20)的最小值是在第 i_1 个约束处达到的，则 $x^{(t+1)} = x^{(t)} + \alpha_t d^{(t)}$ 比 $x^{(t)}$ 增加了一个起作用的不等式约束 i_1，下一步迭代时子问题(38.18)的约束中需要增加约束 $a_{i_1}^T x = b_{i_1}$。

详细算法从略。

38.6 非线性约束优化问题

当约束中含有非线性函数时，问题比无约束和线性约束问题都要复杂得多。这时，很难求得可行的下降方向，而且由于数值误差的因素，对于不等式约束很难判断是否起作用。所以一般的非线性约束问题没有通用的解决方法，更没有可靠的软件。来自于线性约束优化的一些做法仍可以采用，但是算法会很复杂。比较容易实现的方法是各类**罚函数法**，定义一系列增加了惩罚项的目标函数，用一系列无约束优化问题的解逼近约束问题的解，这样的方法又称为**序列无约束最小化方法** (Sequential Unconstrained Minimization Teckniques, SUMT)，特点是实现简单，但是收敛慢而且惩罚很重时问题适定性很差，所以难以求得比较精确的结果。另一类方法是在每步迭代时构造一个二次规划子问题作为近似，通过求解一系列二次规划问题逼近一般非线性约束问题的解，这种方法称为**序列二次规划法** (Sequential Quadratic Programming, SQP)，是现在较好的方法。

38.6.1 外点罚函数法

罚函数法引入包括原来的目标函数和对偏离约束的惩罚两部分的新目标函数，对新目标函数求无约束最小值点。

对一般的约束优化问题(35.1), 等式约束可以用 $|c_i(x)|$ 惩罚，不等式约束可以

用 $\max(0, c_i(x))$ 惩罚，取惩罚函数为

$$\bar{p}(x) = \sum_{i=1}^{p} c_i^2(x) + \sum_{i=p+1}^{p+q} \left[\max(0, c_i(x))\right]^2, \tag{38.21}$$

当且仅当 x 是可行点时 $\bar{p}(x) = 0$。定义新的目标函数

$$\bar{f}(x) = f(x) + \sigma\bar{p}(x),$$

其中 $\sigma > 0$ 称为罚因子。若 $\bar{x}$ 是 $\bar{f}(x)$ 的一个无约束极值点且 $\bar{x}$ 是(35.1)的可行点，则对(35.1)的任意可行点 $\tilde{x}$，都有

$$\bar{f}(\bar{x}) = f(\bar{x}) + \sigma\bar{p}(\bar{x}) = \min\left[f(x) + \sigma\bar{p}(x)\right] \leq f(\tilde{x}) + \sigma\bar{p}(\tilde{x}) = f(\tilde{x})$$

即只要无约束问题 $\operatorname{argmin} \bar{f}(x)$ 的解是约束问题的可行点就是约束极小值点。σ 越大，对无约束极小值点不在可行域内的惩罚越大，所以 $\operatorname{argmin} \bar{f}(x)$ 也越可能落在可行域内。算法迭代进行，只要找到的无约束解不可行就增大罚因子 σ 然后继续求解无约束极值点。算法如下:

取精度 $\epsilon > 0$, 倍数 $c > 1$
取 $t \leftarrow 0$, 初始罚因子 $\sigma_0 > 0$, 任取初始点 $x^{(0)} \in \mathbb{R}^d$
until $(\sigma_t\bar{p}(x^{(t)}) < \epsilon)$ {
    $t \leftarrow t + 1$, $\sigma_t \leftarrow c\sigma_{t-1}$
    以 $x^{(t-1)}$ 为初值求解无约束问题 $\operatorname{argmin}\left[f(x) + \sigma_t\bar{p}(x)\right]$ 得 $x^{(t)}$
}

算法实现时可以取 $\sigma_0 = 1$，$c = 10$。求解无约束问题时迭代停止的条件可以取为梯度向量长度 $\|\nabla\bar{f}(x)\|$ 小于预定精度值。

运用这样的算法，可以任意选取初始点，不需要初始点在可行域内，每次迭代得到的近似解都在可行域外，只有最后一个解近似地落入可行域而且是近似约束最小值点，所以这种方法称为**外点罚函数法**。这种方法要求目标函数在 $\mathbb{R}^d$ 上都有定义，如果目标函数仅在与可行域有关的子集上有定义则无法使用；最后的解不一定严格满足可行性条件，如果要求解严格可行此种方法也无法使用。另外，随着 σ 的增大，在新目标函数 $\bar{f}(x)$ 中目标函数占的比例越来越小，约束惩罚占的比例越来越大，使得优化问题的适定性变得越来越差，求解无约束优化问题会变得很困难，计算误差变得很大。所以，罚函数法虽然看起来很简单，但是不一定能得到好的结果。

38.6.2 内点罚函数法

内点罚函数法要求每次迭代严格地在可行域内进行。定义新的无约束目标函数，当近似极小值点接近可行域边界时新目标函数变得很大（通常在可行域边界处趋于无穷大），相当于在可行域边界处设立了无限高的墙壁不允许搜索越过可行域边界，所以这种方法又称为**障碍罚函数法**。内点罚函数法不能处理含有等式约束的问题，而且要求可行域含有内点。对于如下仅含不等式约束的优化问题

$$\begin{cases} \underset{x\in\mathbb{R}^d}{\operatorname{argmin}} f(x), \text{ s.t.} \\ c_i(x) \le 0, \ i = 1, \dots, q \end{cases} \tag{38.22}$$

定义罚函数为 $\bar{p}(x) = -\sum_{i=1}^{q} \log |c_i(x)|$ 或 $\bar{p}(x) = \sum_{i=1}^{q} \frac{1}{|c_i(x)|}$，$x$ 只能是严格可行点，即所有 $c_i(x) < 0$, 当某个 $c_i(x) = 0$ 时 $\bar{p}(x) = +\infty$。取新的目标函数

$$\bar{f}(x) = f(x) + \sigma \bar{p}(x),$$

因为真正的约束最小值点允许取边界处的值，需要取 $\{\sigma_t\}$ 使得 $\sigma_t \to 0$, 如此减轻对接近可行域边界的惩罚。内点罚函数法和外点罚函数法类似，先取较大的 σ，求 $\bar{f}(x)$ 的无约束最小值点，每次迭代减小 σ 的值后求无约束极值点，直到近似约束最小值点达到足够精度。

由于可行域边界处的高墙的存在，内点罚函数法在近似点靠近可行域边界时加罚的目标函数 $\bar{f}(x)$ 也会变得病态，难以求得最小值点，并且求 $\bar{f}(x)$ 的最小值点时要注意搜索不能跳出可行域。内点罚函数法要求初值在可行域内，可以设法先找可行点。

求初始的可行点的一种方法是以二次惩罚函数(38.21)为目标函数进行无约束优化，求得令 $\bar{p}(x) = 0$ 的点 x，就是满足约束的可行点。

惩罚函数法的一个变种是把内点罚函数和外点罚函数结合使用，对等式约束和初始点不满足的不等式约束，可以采用外点罚函数，其它不等式约束采用内点罚函数，这样的方法称为**混合罚函数法**。

乘子罚函数法针对普通罚函数方法的病态问题做了改进，参见 [高立, 2014] §7.3、§7.4。

38.6.3 序列二次规划法 (SQP)(*)

序列二次规划法 (SQP 方法) 每一步迭代解决一个二次规划子问题，这种方法在中小规模的非线性约束规划问题中是比较好的方法。SQP 的思想类似于无约束规划中的牛顿法，在局部对目标函数用二次函数逼近，对约束用线性函数逼近，得到二次规划子问题。

对一般约束优化问题(35.1)，设 f 有二阶连续偏导数, 设各 c_i 有一阶连续偏导数。设 $x^{(t)}$ 为迭代 t 步后得到的近似约束最小值点，在 $x^{(t)}$ 处对 $f(x)$ 作如下的二阶泰勒展开近似:

$$f(x^{(t)}+d) \approx f(x^{(t)}) + [\nabla f(x^{(t)})]^T d + \frac{1}{2} d^T [\nabla^2 f(x^{(t)})]^T d,\ d \in \mathbb{R}^d,$$

对 $c_i(x)$ 作如下的一阶泰勒展开近似:

$$c_i(x^{(t)}+d) \approx c_i(x^{(t)}) + [\nabla c_i(x^{(t)})]^T d,\ d \in \mathbb{R}^d.$$

进行这样的近似后，考虑如下的二次规划子问题

$$\begin{cases} \underset{d\in\mathbb{R}^d}{\operatorname{argmin}}\ \frac{1}{2} d^T [\nabla^2 f(x^{(t)})]^T d + [\nabla f(x^{(t)})]^T d, \quad \text{s.t.} \\ c_i(x^{(t)}) + [\nabla c_i(x^{(t)})]^T d = 0, i = 1, \dots, p, \\ c_i(x^{(t)}) + [\nabla c_i(x^{(t)})]^T d \le 0, i = p+1, \dots, p+q \end{cases} \tag{38.23}$$

通过求解这样的二次规划子问题得到下一个近似点 $x^{(t+1)}$。

SQP 算法需要考虑的问题比较多，这里略去详细的讨论，感兴趣的读者可以参考 高立 [2014] 第九章。

38.7 用 Julia 的 JuMP 包进行优化 (*)

Julia 语言提供了多个进行最优化建模计算的扩展程序包，其中 JuMP 是比较优秀的一个，JuMP 适用范围广，使用简单，支持多种求解后端。

考虑如下的目标函数优化：

$$\min (1-x)^2 + 100(y-x^2)^2.$$

显然无约束全局最小值点为 $(1,1)$。

用 JuMP 调用 Ipopt 后端求解无约束优化问题：

```
# JuMP 优化例子
using JuMP
using Ipopt # 一个优化后端，自由软件
om = JuMP.Model(Ipopt.Optimizer)
@variable(om, x, start=0.0)
@variable(om, y, start=0.0)
@NLobjective(om, Min, (1 - x)^2 + 100 * (y - x^2)^2)
optimize!(om)
## .....
## EXIT: Optimal Solution Found.
println(" 最小化目标函数值 = ", objective_value(om))
## 最小化目标函数值 = 1.3288608467480825e-28
println(" 最小值点: x = ", value(x), " y = ", value(y))
## 最小值点: x = 0.9999999999999899 y = 0.9999999999999792
```

增加一个线性等式约束 $x + y = 10$ 后求解：

```
@constraint(om, x + y == 10.0)
optimize!(om)
println(" 最小化目标函数值 = ", objective_value(om))
## 最小化目标函数值 = 2.89460755048946
println(" 最小值点: x = ", value(x), " y = ", value(y))
## 最小值点: x = 2.701147124098218 y = 7.2988528759017814
```

继续增加两个线性不等式约束 $x \geq 1.25$, $y \geq -2.1$ 后求解：

```
@constraint(om, x >= 1.25)
@constraint(om, y >= -2.1)
optimize!(om)
println(" 最小化目标函数值 = ", objective_value(om))
## 最小化目标函数值 = 2.89460755048946
println(" 最小值点: x = ", value(x), " y = ", value(y))
## 最小值点: x = 2.701147124098218 y = 7.2988528759017814
```

习题

习题 1

用外点罚函数法求解如下约束优化问题:

$$\begin{cases}\operatorname{argmin} x_1^2 + x_2^2, & \text{s.t.} \\ x_1 - x_2 + 1 = 0. \end{cases}$$

习题 2

用外点罚函数法求解如下约束优化问题:

$$\begin{cases}\operatorname{argmin} x_1^2 + x_2^2, & \text{s.t.} \\ -x_1 + x_2 - 1 \geq 0. \end{cases}$$

习题 3

用内点罚函数法求解上一题目。

Chapter 39

统计计算中的优化问题

统计中许多问题的计算最终都归结为一个最优化问题，典型代表是最大似然估计 (MLE)、各种拟似然估计方法、非线性回归、惩罚函数方法（如 svm、lasso）等。

在统计优化问题中，目标函数往往不象数学函数优化问题那样可以计算到机器精度，目标函数精度一般都是有限的，甚至于是用随机模拟方法计算的。这样，统计问题优化需要稳定性比较高的算法，对于目标函数的小的扰动应该具有良好适应能力，算法迭代收敛条件也不要取得过于严苛，否则可能无法收敛。比如，在最大似然估计问题中，样本值精度受限于测量精度，实际上都存在较大的测量误差，如果优化算法结果对于样本值的微小变化就产生很大变化，这样的算法就是不可取的。

39.1 最大似然估计

最大似然估计经常需要用最优化算法计算，最大似然估计问题有自身的特点，可以直接用一般优化方法进行最大似然估计的计算，但是利用最大似然估计的特点可以得到更有效的算法。

设总体 X 有密度或概率函数 $p(x|\theta)$, θ 为 m 维的分布参数。有了一组样本

$X_1, X_2, \ldots, X_n$ 后，似然函数为

$$L(\theta) = \prod_{i=1}^{n} p(X_i|\theta), \tag{39.1}$$

对数似然函数为

$$l(\theta) = \sum_{i=1}^{n} \log p(X_i|\theta), \tag{39.2}$$

R 的 stats4 包的 `mle()` 提供了最大似然估计计算功能，除了参数估计值以外还能给出参数标准误差的估计。

39.1.1 得分法

设对数似然函数 $l(\theta)$ 有二阶连续偏导数, $\nabla l(\theta)$ 称为得分函数 (score function), 求最大似然估计可以通过解方程 $\nabla l(\theta) = 0$ 实现，这个方程称为**估计方程**。设参数真值为 θ_*, $l(\theta)$ 最大值点为 $\hat{\theta}$, 在适当正则性条件下 $\hat{\theta}$ 渐近正态分布:

$$\sqrt{n}(\hat{\theta} - \theta_*) \xrightarrow{\mathrm{d}} \mathrm{N}(0, I^{-1}(\theta_*)),\ n \to \infty, \tag{39.3}$$

其中

$$I(\theta_*) = \mathrm{Var}(\nabla \log p(X|\theta_*)) = \frac{1}{n}\mathrm{Var}(\nabla l(\theta_*)) = E\left[-\nabla^2 \log p(X|\theta_*)\right] = \frac{1}{n}E\left[-\nabla^2 l(\theta_*)\right] \tag{39.4}$$

是 X 的信息阵。记 $I_n(\theta_*) = \mathrm{Var}(\nabla l(\theta_*))$，称 $I_n(\theta_*)$ 为 $(X_1, X_2, \ldots, X_n)$ 的信息阵, 当 $X_1, X_2, \ldots, X_n$ 独立同分布时 $I_n(\theta_*) = nI(\theta_*)$。

注意最大似然估计求最大值点，与前面求最小值点的问题略有差别，有时讨论负对数似然函数更方便。因为多元正态分布的负对数似然函数是正定二次型，所以如果初值取得比较合适，负对数似然函数 $-l(\theta)$ 与多元正态分布的负对数似然函数相近，接近于正定二次型，这时求 $-l(\theta)$ 的最小值点会比较容易。求得最大似然估计 $\hat{\theta}$ 后，可以用 $\left[I_n(\hat{\theta})\right]^{-1}$ 估计 $\hat{\theta}$ 的协方差阵。

$l(\theta)$ 最大值点 $\hat{\theta}$ 的求解可以用通常的牛顿法、BFGS 法，用到的对数似然函数偏导数和二阶偏导数可以推导解析表达式来计算，也可以用数值微分代替。用数值微分可以避免偏导数推导和编程时的错误。如果用牛顿法求最大值点，$\hat{\theta}$ 的协方差阵可以用 $[-\nabla^2 l(\hat{\theta})]^{-1}$ 来估计。

注意到当样本量充分大且 θ 接近于真值 θ_* 时海色阵 $\nabla^2 l(\theta) \approx -I_n(\theta)$，如果信息阵的公式很容易得到，在牛顿法的迭代中可以用负信息阵代替对数似然函数的海色阵，迭代为

$$\hat{\theta}^{(t+1)} = \hat{\theta}^{(t)} - \left[-I_n(\hat{\theta}^{(t)})\right]^{-1} \nabla l(\hat{\theta}^{(t)}). \tag{39.5}$$

这种方法叫做**得分法** (scoring)。

例 39.1 (逻辑斯谛回归参数估计)**.** 设 $Y_i \sim \mathrm{B}(m_i, \pi_i)$, $i = 1, 2, \ldots, n$, $Y_1, Y_2, \ldots, Y_n$ 相互独立，其中 $\pi_i = \exp(\beta^T x_i)/[1 + \exp(\beta^T x_i)]$, β 为未知参数向量，自变量 $x_i, i = 1, 2, \ldots, n$ 已知。这样的模型称为**逻辑斯谛回归模型**，函数 $\mathrm{logit}(\pi) \triangleq \log \frac{\pi}{1-\pi}$, $\pi \in (0, 1)$ 称为逻辑斯谛函数，其反函数为 $\mathrm{logit}^{-1}(\gamma) = \frac{\exp(\gamma)}{1+\exp(\gamma)}$。上面模型中的 π_i 满足 $\mathrm{logit}(\pi_i) = \beta^T x_i$, $\pi_i = \mathrm{logit}^{-1}(\beta^T x_i)$。

Y_i 的概率函数及其对数为

$$P(Y_i = y_i) = \binom{m_i}{y_i} \pi_i^{y_i} (1-\pi_i)^{m_i - y_i} = \binom{m_i}{y_i} \left[1 + \exp(\beta^T x_i)\right]^{-m_i} \left[\exp\left(\beta^T x_i\right)\right]^{y_i},$$

$$\log P(Y_i = y_i) = \log \binom{m_i}{y_i} + y_i \cdot \beta^T x_i - m_i \log\left[1 + \exp(\beta^T x_i)\right],$$

对数似然函数为（省略了不随 β 变化的部分）

$$l(\beta) = \sum_{i=1}^{n} \left\{ y_i \cdot \beta^T x_i - m_i \log\left[1 + \exp(\beta^T x_i)\right] \right\}$$

梯度和海色阵为

$$\nabla l(\beta) = \sum_{i=1}^{n} \left(y_i x_i - \frac{m_i \exp(\beta^T x_i)}{1 + \exp(\beta^T x_i)} x_i \right) = \sum_{i=1}^{n} (y_i - m_i \pi_i) x_i,$$

$$\nabla^2 l(\beta) = -\sum_{i=1}^{n} m_i \frac{\exp(\beta^T x_i)}{[1 + \exp(\beta^T x_i)]^2} \cdot x_i x_i^T = -\sum_{i=1}^{n} m_i \pi_i (1 - \pi_i) x_i x_i^T.$$

因为海色阵不包含随机成分，所以 $Y = (Y_1, \ldots, Y_n)$ 的信息阵

$$I_n(\beta) = E\left(-\nabla^2 l(\beta)\right) = -\nabla^2 l(\beta),$$

得分法和普通牛顿法是相同的, 迭代公式同为

$$\begin{cases} \pi_i^{(t)} = \mathrm{logit}^{-1}\left([\beta^{(t)}]^T x_i\right), \ i = 1, 2, \ldots, n, \\ \beta^{(t+1)} = \beta^{(t)} - \left[-\sum_{i=1}^{n} m_i \pi_i^{(t)} (1 - \pi_i^{(t)}) x_i x_i^T\right]^{-1} \left[\sum_{i=1}^{n} (y_i - m_i \pi_i^{(t)}) x_i\right], \\ \qquad t = 0, 1, 2, \ldots \end{cases}$$

应该用一个合理的初始估计作为 β 的初值 $\beta^{(0)}$。记 $\hat{\pi}_i = y_i/m_i$, 以 $(x_i, \hat{\pi}_i)$, $i = 1, 2, \ldots, n$ 为自变量和因变量做普通最小二乘回归得到回归系数估计可以用作 $\beta^{(0)}$。这里不用 $\text{logit}(\hat{\pi}_i)$ 作为因变量是因为 $\hat{\pi}_i = 0$ 或 1 时 $\text{logit}^{-1}(\hat{\pi}_i)$ 无定义。

※※※※※

39.1.2 精简最大似然估计

在最大似然估计问题中，如果能分步求得最大值，则可以减少问题的维数从而降低难度。设参数 θ 分为 θ_1 和 θ_2 两部分, 如果对任意 θ_1，都能比较容易地求得

$$\underset{\theta_2}{\text{argmax}}\, l(\theta_1, \theta_2) = \hat{\theta}_2(\theta_1),$$

则

$$\max_{\theta_1, \theta_2} l(\theta_1, \theta_2) = \max_{\theta_1} \max_{\theta_2} l(\theta_1, \theta_2) = \max_{\theta_1} l(\theta_1, \hat{\theta}_2(\theta_1)),$$

问题简化为以 θ_1 为自变量的优化问题。这样的方法称为**精简最大似然方法** (concentrated MLE)。

如果给定 θ_2 后也很容易得到关于 θ_1 的最大值点 $\hat{\theta}_1(\theta_2)$，那么，可以迭代地求两部分的最大值。首先取 θ_1 的适当初值 $\theta_1^{(0)}$，求得 $\theta_2^{(0)} \triangleq \hat{\theta}_2(\theta_1^{(0)})$, 再令 $\theta_1^{(1)} \triangleq \hat{\theta}_1(\theta_2^{(0)})$, $\theta_2^{(1)} \triangleq \hat{\theta}_2(\theta_1^{(1)})$, 如此迭代直至收敛，这就构成了一种分块松弛法（见37.1）。这样的方法实现简单，但有可能速度比较慢。

例 39.2. 设总体 X 服从 $\Gamma(\alpha, \lambda)$ 分布，密度为

$$p(x; \alpha, \lambda) = \begin{cases} \frac{\lambda^\alpha x^{\alpha-1}}{\Gamma(\alpha)} e^{-\lambda x}, & x > 0, \alpha > 0, \lambda > 0, \\ 0, & \text{其它} \end{cases}$$

设有 X 的简单随机样本 $X_1, X_2, \ldots, X_n$，则对数似然函数为

$$l(\alpha, \lambda) = n\alpha \log \lambda - n \log \Gamma(\alpha) + \alpha \sum \log X_i - \lambda \sum X_i - \sum \log X_i,$$

给定 $\alpha > 0$，先关于 λ 求最大值。易见

$$\frac{\partial l}{\partial \lambda} = \frac{n\alpha}{\lambda} - \sum X_i,$$

解得稳定点 $\hat{\lambda} = \hat{\lambda}(\alpha) = \alpha/\bar{X}$, 其中 $\bar{X} = \frac{1}{n}\sum X_i$。又 $\frac{\partial^2 l}{\partial \lambda^2} = -\frac{n\alpha}{\lambda^2} < 0$, $\forall \lambda > 0$, 所以 $\hat{\lambda}(\alpha)$ 是给定 α 情形下 $l(\alpha, \lambda)$ 关于自变量 λ 的最大值点。令 $\tilde{l}(\alpha) = l(\alpha, \hat{\lambda}(\alpha))$, 则

$$\begin{aligned}\tilde{l}(\alpha) =& n\alpha \log \alpha - n \log \Gamma(\alpha) + \alpha \left[\sum \log \frac{X_i}{\bar{X}} - n\right] - \sum \log X_i,\\ \tilde{l}'(\alpha) =& n \log \alpha - n\psi(\alpha) + \sum \log \frac{X_i}{\bar{X}},\end{aligned}$$

其中 $\psi(\alpha) = \frac{d}{d\alpha} \log \Gamma(\alpha)$ 称为 digamma 函数。可以证明 $\tilde{l}'(\alpha)$ 严格单调减，方程 $\tilde{l}'(\alpha) = 0$ 存在唯一解 $\hat{\alpha}$，且 $\hat{\alpha} = \operatorname{argmax}_{\alpha>0} \tilde{l}(\alpha)$，从而 $(\hat{\alpha}, \hat{\lambda}(\hat{\alpha}))$ 为总体参数的最大似然估计。原来二维的优化问题简化为一个一元函数 $\tilde{l}(\alpha)$ 的优化问题，可以用二分法或牛顿法求解方程 $\tilde{l}'(\alpha) = 0$。

※※※※※

39.2 非线性回归

首先要注意有些非线性回归可以轻易地转换为线性回归。比如，$y = Ae^{-\beta x}$ 可以变成 $\log y = a - \beta x$, 其中 $a = \log A$。又如，$y = A\cos(2\pi f t + \varphi)(f$ 已知) 可以变成 $y = a\cos(2\pi f t) + b\sin(2\pi f t)$, 其中 $a = A\cos\varphi$, $b = -A\sin\varphi$。

考虑如下的非线性回归问题

$$Y_i = \varphi(x_i, \beta) + \varepsilon_i,\ i = 1, 2, \ldots, n,$$

其中 ε_i 独立同 $\mathrm{N}(0,\sigma^2)$ 分布，$\beta \in \mathbb{R}^p$ 为未知参数向量，Y_i 为因变量，x_i 为非随机的自变量，φ 为关于 β 非线性的函数。

记 $\gamma = \sigma^{-2}$(这可以使得似然函数偏导数形式更简单), $Y = (Y_1, \ldots, Y_n)^T$, $g_i(\beta) = \varphi(x_i, \beta)$, $g = g(\beta) = (g_1(\beta), \ldots, g_n(\beta))^T$,

$$\begin{aligned}G =& G(\beta) = \left(\frac{\partial g_i(\beta)}{\partial \beta_j}\right)_{\substack{i=1,\ldots,n\\ j=1,\ldots,p}},\\ S(\beta) =& \sum_{i=1}^{n} \left[Y_i - g_i(\beta)\right]^2 = \|Y - g(\beta)\|^2.\end{aligned}$$

最小化 $S(\beta)$ 可以得到参数 β 的最小二乘估计，容易验证最小二乘估计也是最大似然估计。求非线性最小二乘估计可以用通常的牛顿法、拟牛顿法、Nelder-

Mead 等方法，下面结合非线性回归模型最大似然估计的特点讨论更为高效的参数估计算法。

上述模型的对数似然函数为 (省略常数项)

$$l(\beta,\gamma) = \frac{n}{2}\log\gamma - \frac{\gamma}{2}S(\beta),$$

梯度和海色阵为

$$\nabla l(\beta,\gamma) = \begin{pmatrix} \gamma G^T(Y-g) \\ \frac{n}{2\gamma} - \frac{1}{2}S(\beta) \end{pmatrix},$$

$$\nabla^2 l(\beta,\gamma) = \begin{pmatrix} \gamma\sum_{i=1}^n [Y_i - g_i(\beta)]\nabla^2 g_i(\beta) - \gamma G^T G & G^T(Y-g) \\ (Y-g)^T G & -\frac{n}{2}\gamma^{-2} \end{pmatrix},$$

由此得 Y 的信息阵为

$$I_n(\beta,\gamma) = E[-\nabla^2 l(\beta,\gamma)] = \begin{pmatrix} \gamma G^T G & 0 \\ 0 & \frac{n}{2}\gamma^{-2} \end{pmatrix},$$

可见信息阵比海色阵简单得多，用得分法迭代估计参数比牛顿法更简单，迭代公式为

$$\beta^{(t+1)} = \beta^{(t)} + \left[G^T(\beta^{(t)})G(\beta^{(t)})\right]^{-1} G^T(\beta^{(t)})\left[Y - g(\beta^{(t)})\right], \tag{39.6}$$

这种方法称为高斯–牛顿法。如果拟合误差 $Y_i - g_i(\beta)$ 都比较小，这时 $-I_n(\beta,\gamma)$ 与海色阵 $\nabla^2 l(\beta,\gamma)$ 会很接近，这种情况下高斯–牛顿法具有很好的效果。与牛顿法的缺点类似，高斯–牛顿法有可能步长过大，使得 $S(\beta)$ 变大而不是变小，这时可以在(39.6)中增加一个步长 α，从 $\alpha = 1$ 开始每次步长减半直到迭代使 $S(\beta)$ 变小。

当拟合误差 $Y_i - g_i(\beta)$ 较大时，高斯–牛顿法效果会变差。把公式(39.6)改为

$$\beta^{(t+1)} = \beta^{(t)} + \left[G^T(\beta^{(t)})G(\beta^{(t)}) + \lambda_t I_p\right]^{-1} G^T(\beta^{(t)})\left[Y - g(\beta^{(t)})\right], \tag{39.7}$$

其中 $\lambda_t \geq 0$, $\lambda_t \geq 0$ 可以迭代地更新。此算法称为 LMF(Levenberg-Marquardt-Fletcher) 算法。公式(39.7) 可以看成是取搜索方向为高斯–牛顿法的搜索方向和最速下降法的搜索方向之间的一个方向, 记搜索方向为

$$d^{(t)} = \left[G^T(\beta^{(t)})G(\beta^{(t)}) + \lambda_t I_p\right]^{-1} G^T(\beta^{(t)})\left[Y - g(\beta^{(t)})\right] \tag{39.8}$$

并记

$$\begin{aligned}\Delta S_t =& S(\beta^{(t)}) - S(\beta^{(t)} + d^{(t)}),\\ \Delta q_t =& (d^{(t)})^T \left[\lambda_t d^{(t)} + G^T(\beta^{(t)}) \left(Y - g(\beta^{(t)})\right)\right],\end{aligned}$$

LMF 算法

LMF 算法

取初值 $\beta^{(0)}$, $\lambda_0 > 0$, 精度 $\epsilon > 0$, $t \leftarrow 0$
until (迭代收敛) {
    用(39.8)求 $d^{(t)}$
    计算 $\gamma_t = \Delta S_t / \Delta q_t$
    if($\gamma_t < 0.25$) {
        $\lambda_{t+1} \leftarrow 4\lambda_t$
    } **else if**($\lambda_t > 0.75$) {
        $\lambda_{t+1} \leftarrow \frac{1}{2}\lambda_t$
    }
    if($\gamma_t \le 0$) {
        $\beta^{(t+1)} \leftarrow \beta^{(t)}$
    } **else** {
        $\beta^{(t+1)} \leftarrow \beta^{(t)} + d^{(t)}$
    }
    $t \leftarrow t + 1$
} # until
输出 $\beta^{(t)}$ 作为最小二乘估计

参见 高立 [2014] §5.3。

39.3 EM 算法

EM 算法最初用于缺失数据模型参数估计，现在已经用在许多优化问题中。设模型中包含 X_{obs} 和 X_{mis} 两个随机成分，有联合密度函数或概率函数 $f(x_{\text{obs}}, x_{\text{mis}}|\theta)$, θ 为未知参数。称 $f(x_{\text{obs}}, x_{\text{mis}}|\theta)$ 为完全数据的密度，一般具

有简单的形式。实际上我们只有 X_{obs} 的观测数据 $X_{\text{obs}} = x_{\text{obs}}$，$X_{\text{mis}}$ 不能观测得到，这一部分可能是缺失观测数据，也可能是潜在影响因素。所以实际的似然函数为

$$L(\theta) = f(x_{\text{obs}}|\theta) = \int f(x_{\text{obs}}, x_{\text{mis}}|\theta)\, dx_{\text{mis}}, \tag{39.9}$$

这个似然函数通常比完全数据的似然函数复杂得多，所以很难直接从 $L(\theta)$ 求最大似然估计。

EM 算法的想法是，已经有了参数的近似估计值 $\theta^{(t)}$ 后，假设 $(X_{\text{obs}}, X_{\text{mis}})$ 近似服从完全密度 $f(x_{\text{obs}}, x_{\text{mis}}|\theta^{(t)})$, 这里 $X_{\text{obs}} = x_{\text{obs}}$ 已知，所以认为 X_{mis} 近似服从由 $f(x_{\text{obs}}, x_{\text{mis}}|\theta^{(t)})$ 导出的条件分布

$$f(x_{\text{mis}}|x_{\text{obs}}, \theta^{(t)}) = \frac{f(x_{\text{obs}}, x_{\text{mis}}|\theta^{(t)})}{f(x_{\text{obs}}|\theta^{(t)})},$$

其中 $f(x_{\text{obs}}|\theta^{(t)})$ 是由 $f(x_{\text{obs}}, x_{\text{mis}}|\theta^{(t)})$ 决定的边缘密度。据此近似条件分布，在完全数据对数似然函数 $\log f(X_{\text{obs}}, X_{\text{mis}}|\theta)$ 中，把 $X_{\text{obs}} = x_{\text{obs}}$ 看成已知，关于未知部分 X_{mis} 按密度 $f(x_{\text{mis}}|x_{\text{obs}}, \theta^{(t)})$ 求期望，得到 θ 的函数 $Q_t(\theta)$，再求 $Q_t(\theta)$ 的最大值点作为下一个 $\theta^{(t+1)}$。

EM 算法每次迭代有如下的 E 步（期望步）和 M 步（最大化步）：

- **E 步**: 计算完全数据对数似然函数的期望 $Q_t(\theta) = E\{\log f(x_{\text{obs}}, X_{\text{mis}}|\theta)\}$, 其中期望针对随机变量 X_{mis}，求期望时假定 X_{mis} 服从条件密度 $f(x_{\text{mis}}|x_{\text{obs}}, \theta^{(t)})$ 决定的分布。
- **M 步**: 求 $Q_t(\theta)$ 的最大值点，记为 $\theta^{(t+1)}$，迭代进入下一步。

定理 39.1. *EM 算法得到的估计序列 $\theta^{(t)}$ 使得(39.9)中的似然函数值 $L(\theta^{(t)})$ 单调不减。*

证明: 对任意参数 θ，有

$$\begin{aligned}\ln L(\theta) =& \ln f(x_{\text{obs}}|\theta) \cdot \int f(x_{\text{mis}}|x_{\text{obs}},\theta^{(t)})\,dx_{\text{mis}}\\ =& \int [\log f(x_{\text{obs}},x_{\text{mis}}|\theta) - \log f(x_{\text{mis}}|x_{\text{obs}},\theta)] f(x_{\text{mis}}|x_{\text{obs}},\theta^{(t)})\,dx_{\text{mis}}\\ =& \int \log f(x_{\text{obs}},x_{\text{mis}}|\theta) f(x_{\text{mis}}|x_{\text{obs}},\theta^{(t)})\,dx_{\text{mis}}\\ & - \int \log f(x_{\text{mis}}|x_{\text{obs}},\theta) f(x_{\text{mis}}|x_{\text{obs}},\theta^{(t)})\,dx_{\text{mis}}\\ =& Q_t(\theta) - \int \log f(x_{\text{mis}}|x_{\text{obs}},\theta) f(x_{\text{mis}}|x_{\text{obs}},\theta^{(t)})\,dx_{\text{mis}}\end{aligned}$$

由信息不等式知

$$\begin{aligned}&\int \log f(x_{\text{mis}}|x_{\text{obs}},\theta) f(x_{\text{mis}}|x_{\text{obs}},\theta^{(t)})\,dx_{\text{mis}}\\ \le& \int \log f(x_{\text{mis}}|x_{\text{obs}},\theta^{(t)}) f(x_{\text{mis}}|x_{\text{obs}},\theta^{(t)})\,dx_{\text{mis}}\end{aligned}$$

又 EM 迭代使得 $Q_t(\theta^{(t+1)}) \ge Q_t(\theta^{(t)})$, 所以

$$\begin{aligned}\log L(\theta^{(t+1)}) \ge& Q_t(\theta^{(t)}) - \int \log f(x_{\text{mis}}|x_{\text{obs}},\theta^{(t)}) f(x_{\text{mis}}|x_{\text{obs}},\theta^{(t)})\,dx_{\text{mis}}\\ =& \log L(\theta^{(t)}).\end{aligned}$$

定理证毕。

※※※※※

在适当正则性条件下，EM 算法的迭代序列 $\theta^{(t)}$ 依概率收敛到 $L(\theta)$ 的最大值点 $\hat{\theta}$。但是，定理39.1仅保证 EM 算法最终能收敛，但不能保证 EM 算法会收敛到似然函数的全局最大值点，算法也可能收敛到局部极大值点或者鞍点。

在实际问题中，往往 E 步和 M 步都比较简单，有时 E 步和 M 步都有解析表达式，这时 EM 算法实现很简单。EM 算法优点是计算稳定，可以保持原有的参数约束，缺点是收敛可能很慢，尤其是接近最大值点时可能收敛更慢。如果(39.9)中的似然函数不是凸函数，算法可能收敛不到全局最大值点，遇到这样的问题可以多取不同初值比较，用矩估计等合适的近似值作为初值。

例 39.3 (多项分布)**.** 设 X 取值于 $\{1,2,3,4\}$, 分布概率为 $p(x|\theta) \triangleq \pi_x(\theta)$:

$$\begin{aligned}\pi_1(\theta) =& \frac{1}{4}(2+\theta), \quad \pi_2(\theta) = \frac{1}{4}(1-\theta),\\ \pi_3(\theta) =& \frac{1}{4}(1-\theta), \quad \pi_4(\theta) = \frac{1}{4}\theta.\end{aligned}$$

设 n 次试验得到的 X 值有 n_j 个 $j(j=1,2,3,4)$，求参数 θ 的最大似然估计。

这是 [Dempster et al., 1977] 文章中的例子。观测数据的对数似然函数为 (去掉了与参数无关的加性常数)

$$l(\theta) = n_1 \log(2+\theta) + (n_2+n_3)\log(1-\theta) + n_4 \log\theta.$$

令 $l'(\theta)=0$ 得到一个关于 θ 的二次方程，由此写出 $\operatorname{argmax} l(\theta)$ 的解析表达式:

$$l'(\theta) = \frac{n_1}{2+\theta} - \frac{n_2+n_3}{1-\theta} + \frac{n_4}{\theta}$$

令 $l'(\theta)=0$，即求解二次方程

$$n\theta^2 + (-n_1+2n_2+2n_3+n_4)\theta - 2n_4 = 0.$$

得最大似然估计为

$$\hat{\theta} = \frac{-b+\sqrt{b^2+8nn_4}}{2n}$$

其中 $b=-n_1+2n_2+2n_3+n_4$。

例如，$n=400$, $n_1=80$, $n_2=120$, $n_3=110$, $n_4=90$ 时：

```
n = 400; n1=80; n2=120; n3=110; n4=90
b = -n1 + 2*n2 + 2*n3 + n4
hattheta = (-b + sqrt(b^2 + 8*n*n4))/(2*n)
hattheta
```

```
## [1] 0.3042153
```

下面用 EM 算法迭代估计 θ。将结果 1 分解为 11 和 12，n_1 分解为 Z_1+Z_2，其中 $P(X=11)=\frac{1}{2}$, $P(X=12)=\frac{1}{4}\theta$。这时 Z_2+n_4 代表结果 12 和结果 4 的出现次数，这两种结果出现概率为 $\frac{1}{2}\theta$，其它结果 (11, 2, 3) 的出现概率为 $1-\frac{1}{2}\theta$。令 $Y=Z_2+n_4$，则 $Y\sim \mathrm{B}(n,\frac{1}{2}\theta)$。

数据 $(Z_1, Z_2, n_2, n_3, n_4)$ 的全似然函数为

$$L_c(\theta) \propto (\frac{1}{2})^{Z_1}(\frac{\theta}{4})^{Z_2}(\frac{1}{4}-\frac{\theta}{4})^{n_2+n_3}(\frac{\theta}{4})^{n_4}$$

对数似然函数（差一个与 θ 无关的常数项）为

$$l_c(\theta) = \ln L_c(\theta) = (Z_2 + n_4)\ln\theta + (n_2 + n_3)\ln(1-\theta)$$

在 EM 迭代中，假设已经得到的参数 θ 近似值为 $\theta^{(t)}$，设 $\theta = \theta^{(t)}$，在给定 n, n_1, n_2, n_3, n_4 条件下求 $l_c(\theta)$ 的条件期望，这时 Z_2 的条件分布为

$$\mathrm{B}\left(n_1, \frac{\theta^{(t)}}{2+\theta^{(t)}}\right),$$

于是

$$Z_2^{(t)} = E(Z_2|\theta^{(t)}, n, n_1, n_2, n_3, n_4) = n_1\frac{\theta^{(t)}}{2+\theta^{(t)}},$$

从而完全对数似然函数的条件期望为

$$Q_t(\theta) = E(\ln L_c(\theta)|\theta^{(t)}, n, n_1, n_2, n_3, n_4) = (Z_2^{(t)}+n_4)\ln\theta+(n_2+n_3)\ln(1-\theta).$$

求解 $Q_t(\theta)$ 的最大值，令

$$\frac{d}{d\theta}Q_t(\theta) = \frac{n_4 + Z_2^{(t)}}{\theta} - \frac{n_2+n_3}{1-\theta} = 0,$$

得下一个参数近似值为

$$\theta^{(t+1)} = \frac{n_4 + Z_2^{(t)}}{n_2 + n_3 + n_4 + Z_2^{(t)}}.$$

于是, EM 迭代步骤从某个 $\theta^{(0)}$ 出发，比如 $\theta^{(0)} = \frac{1}{2}$，在第 t 步计算

$$Z_2^{(t)} = n_1\frac{\theta^{(t)}}{2+\theta^{(t)}}, \quad \theta^{(t+1)} = \frac{n_4 + Z_2^{(t)}}{n_2 + n_3 + n_4 + Z_2^{(t)}}$$

迭代到两次的近似参数值变化小于 $\epsilon = 10^{-6}$ 为止。

程序例：

```
n = 400; n1=80; n2=120; n3=110; n4=90
logL <- function(th) {
  n1*log(2+th) + (n2+n3)*log(1-th) + n4*log(th)
}
eps <- 1E-6
```

```
theta <- 0.5
k <- 0
repeat{
  k <- k+1
  theta0 <- theta # 保存上一个 $\theta$ 值
  z2t <- n1 * theta0 / (2 + theta0)
  theta <- (n4 + z2t) / (n2+n3+n4 + z2t) # 新的 $\theta$ 估计
  cat("log L(theta[", k, "]) = log L(",
      theta, ") = ", logL(theta), "\n", sep="")
  if(abs(theta - theta0) < eps || k >= 500) break
}
```

```
## log L(theta[1]) = log L(0.3154762) = -123.8386
## log L(theta[2]) = log L(0.3049254) = -123.7474
## log L(theta[3]) = log L(0.3042604) = -123.747
## log L(theta[4]) = log L(0.3042182) = -123.747
## log L(theta[5]) = log L(0.3042155) = -123.747
## log L(theta[6]) = log L(0.3042154) = -123.747
```

```
cat(" 迭代次数 =", k, " EM 参数估计 =", theta, "\n")
```

```
## 迭代次数= 6  EM参数估计= 0.3042154
```

仅需要 6 次迭代就达到了小数点后 6 位精度。

※※※※※

例 39.4 (指数分布右删失 (*))**.** 设某种设备的寿命总体 $Y \sim \mathrm{Exp}(1/\theta)$，$EY = \theta$，对 n 个这样的设备进行寿命试验，其中 n_1 个观测到失效时间 $t_1, \dots, t_{n_1}$，另外的 $n_2 = n - n_1$ 个没有观测到失效，仅知道失效时间分别超过 $c_j, j = n_1 + 1, \dots, n$。求参数的最大似然估计。

观测数据为

$$x = (x_1, \dots, x_{n_1}, x_{n_1+1}, \dots, x_n) = (t_1, \dots, t_{n_1}, c_{n_1+1}, \dots, c_n)$$

观测数据的似然函数为

$$L(\theta)=(\frac{1}{\theta})^{n_1}\exp\{-\frac{1}{\theta}\sum_{1}^{n_1}t_i\}\exp\{-\frac{1}{\theta}\sum_{n_1+1}^{n}c_j\}=\theta^{-n_1}\exp\{-\frac{1}{\theta}\sum_{1}^{n}x_i\}$$

对数似然函数为

$$\ln L(\theta)=-n_1\ln\theta-\frac{1}{\theta}\sum_{1}^{n}x_i$$

令 $\frac{d}{d\theta}\ln L(\theta)=0$ 得最大似然估计为

$$\hat{\theta}=\frac{1}{n_1}\sum_{i=1}^{n}x_i$$

下面用 EM 算法估计 θ。设完全数据为 $(t_1,\dots,t_{n_1},t_{n_1+1},\dots,t_n)$ 都是失效时间，完全数据的似然函数为

$$L_c(\theta)=\theta^{-n}\exp\{-\frac{1}{\theta}\sum_{1}^{n}t_i\}$$

完全的对数似然函数为

$$\ln L_c(\theta)=-n\ln\theta-\frac{1}{\theta}\sum_{1}^{n_1}t_i-\frac{1}{\theta}\sum_{n_1+1}^{n}t_j$$

取适当初值 $\theta^{(0)}$，如 $\theta^{(0)}=\frac{1}{n}x_i$。在迭代中设已有 $\theta^{(t)}$，在已知 $\theta=\theta^{(t)}$，$t_i,i=1,\dots,n_1$，$t_j>c_j,j=n_1+1,\dots,n$ 的条件下求 $\ln L_c(\theta)$ 的条件期望，先求 $E(t_j|\theta^{(t)},t_j>c_j)$，$j=n_1+1,\dots,n$。易见 t_j-c_j 的条件分布为 $\mathrm{Exp}(\frac{1}{\theta^{(t)}})$，所以

$$E(t_j|\theta^{(t)},t_j>c_j)=c_j+E(t_j-c_j|\theta^{(t)},t_j>c_j)=c_j+\theta^{(t)}$$

所以

$$\begin{aligned}Q_t(\theta)=&E\left[\ln L_c(\theta)\mid x\right]=-n\ln\theta-\frac{1}{\theta}\sum_{1}^{n_1}t_i-\frac{1}{\theta}\sum_{n_1}^{n}(c_j+\theta^{(t)})\\=&-n\ln\theta-\frac{1}{\theta}\sum_{1}^{n}x_i-n_2\frac{\theta^{(t)}}{\theta}\end{aligned}$$

令 $\frac{d}{d\theta}Q_t(\theta)=0$ 得参数的迭代公式为

$$\theta^{(t+1)}=\frac{1}{n}\left(\sum_{1}^{n}x_i+n_2\theta^{(t)}\right)$$

模拟实例：

```
set.seed(101)
true.theta <- 2.0
n <- 25
t.comp <- rexp(n, 1/true.theta)
t.cens <- rexp(n, 1/true.theta)
sele <- t.comp <= t.cens
x.comp <- t.comp[sele]
x.cens <- t.cens[!sele]
x <- c(x.comp, x.cens) # 观测数据
n1 <- length(x.comp)
n2 <- n - n1
cat("n=", n, " n1=", n1, " n2=", n2, "\n")
```

```
## n= 25  n1= 9  n2= 16
```

```
## MLE:
sumx <- sum(x)
hat.theta <- sumx / n1
cat("MLE=", hat.theta, "\n")
```

```
## MLE= 2.555051
```

```
## EM:
eps <- 1E-6
max.iter <- 500
theta <- mean(x)
k <- 0
repeat{
  k <- k+1
  theta0 <- theta # 上一个 $\theta$ 值
  theta <- 1/n*(sumx + n2*theta0)
  if(abs(theta - theta0) < eps || k >= max.iter) break
}
cat(" 迭代次数 =", k, " EM estimate=", theta, "\n")
```

```
## 迭代次数= 31  EM estimate= 2.555049
```

※※※※※

例 39.5 (另一指数分布右删失)**.** 设某种设备的寿命总体 $Y \sim \text{Exp}(1/\theta), EY = \theta$，设对 n 个这样的设备进行寿命试验，得到了失效时间 $x_1, \dots, x_n$，对另外的 m 个这样的设备进行寿命试验，仅知道其中 r 个在时刻 s 之前失效，$m-r$ 个未失效。用 EM 方法求 θ 的最大似然估计。

观测数据为

$$x = (x_1, \dots, x_n, r),$$

观测数据的似然函数为

$$L(\theta) \propto \theta^{-n} \exp\{-\frac{1}{\theta}\sum_1^n x_i\}(1-e^{-s/\theta})^r e^{-(m-r)s/\theta},$$

对数似然函数为

$$\ln L(\theta) = -n\ln\theta + r\ln(1-e^{-s/\theta}) - \frac{1}{\theta}\left[\sum_1^n x_i + (m-r)s\right],$$

直接求 MLE 有困难。

下面用 EM 算法估计 θ。设第二批的 m 个试验的具体失效时间为 $u_j, j = 1, \dots, m$。完全数据的似然函数为

$$L_c(\theta) = \theta^{-n-m}\exp\left\{-\frac{1}{\theta}\left[\sum_1^n x_i + \sum_1^m u_j\right]\right\},$$

完全的对数似然函数为

$$\ln L_c(\theta) = -(n+m)\ln\theta - \frac{1}{\theta}\left[\sum_1^n x_i + \sum_1^m u_j\right].$$

取适当初值 $\theta^{(0)}$，如 $\theta^{(0)} = \frac{1}{n}x_i$。在迭代中设已有 $\theta^{(t)}$，在已知 $\theta = \theta^{(t)}$ 和观测数据 x 的条件下求 $\ln L_c(\theta)$ 的条件期望。先求 $E(u_j|\theta^{(t)}, x)$，$j = n_1+1, \dots, n$。在给定 $\theta = \theta^{(t)}$，观测数据 x 和 $u_j < s$ 条件下，u_j 的条件分布是 $\text{Exp}(1/\theta^{(t)})$ 在 $u_j < s$ 下的条件分布，

$$P(u_j > u | u_j < s) = \frac{P(u < u_j < s)}{P(u_j < s)} = \frac{e^{-u/\theta^{(t)}} - e^{-s/\theta^{(t)}}}{1-e^{-s/\theta^{(t)}}},$$

这时

$$E(u_j|\theta^{(t)}, u_j < s) = \int_0^s P(u_j > u|u_j < s)\, du = \theta^{(t)} + s - \frac{s}{1 - e^{-s/\theta^{(t)}}}.$$

在给定 $\theta = \theta^{(t)}$，观测数据 x 和 $u_j \geq s$ 条件下，$u_j - s$ 条件分布是 $\text{Exp}(1/\theta^{(t)})$，所以

$$E(u_j|\theta^{(t)}, u_j \geq s) = s + \theta^{(t)}.$$

于是

$$\begin{aligned} E(u_j|\theta^{(t)}, x) =& E(u_j|\theta^{(t)}, u_j < s)P(u_j < s) + E(u_j|\theta^{(t)}, u_j \geq s)P(u_j \geq s) \\ =& \left(\theta^{(t)} + s - \frac{s}{1 - e^{-s/\theta^{(t)}}}\right)\frac{r}{m} + (s + \theta^{(t)})\frac{m - r}{m}. \end{aligned}$$

所以

$$\begin{aligned} Q_t(\theta) =& E\left[\ln L_c(\theta)|\theta^{(t)}, x\right] \\ =& -(n + m)\ln\theta - \frac{1}{\theta}\left[\sum_1^n x_i \right. \\ & \left. + \left(\theta^{(t)} + s - \frac{s}{1 - e^{-s/\theta^{(t)}}}\right) r + (s + \theta^{(t)})(m - r)\right]. \end{aligned}$$

求

$$\begin{aligned} & \frac{d}{d\theta}Q_t(\theta) \\ =& -\frac{n + m}{\theta} + \frac{1}{\theta^2}\left[\sum_1^n x_i \right. \\ & \left. + \left(\theta^{(t)} + s - \frac{s}{1 - e^{-s/\theta^{(t)}}}\right) r + (s + \theta^{(t)})(m - r)\right] \\ =& 0, \end{aligned}$$

解得迭代公式为

$$\theta^{(t+1)} = \frac{1}{n + m}\left[\sum_1^n x_i + m(\theta^{(t)} + s) - \frac{rs}{1 - e^{-s/\theta^{(t)}}}\right].$$

模拟实例：

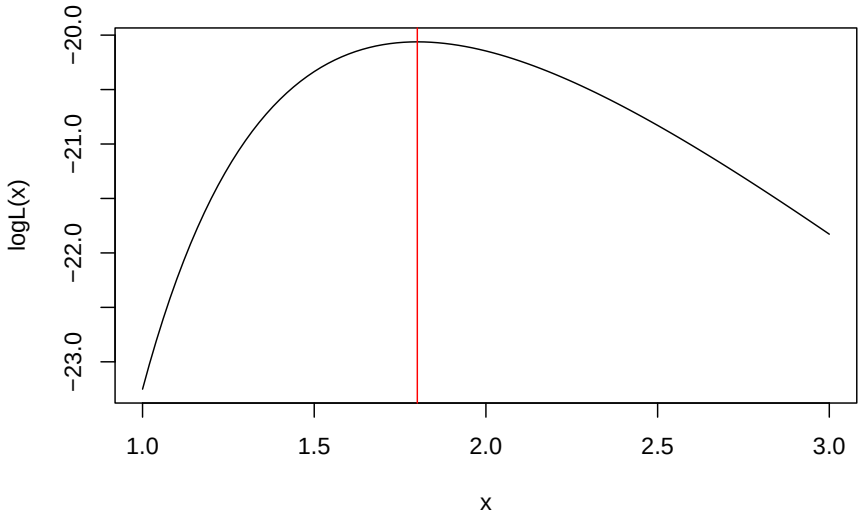
```
set.seed(101)
true.theta <- 2.0
```

```
n <- 10
m <- 8
s <- 1.5*true.theta
x1 <- rexp(n, 1/true.theta)
r <- sum(rexp(m, 1/true.theta) < s)

## EM:
eps <- 1E-6
max.iter <- 500
theta <- mean(x1)
sumx1 <- sum(x1)
k <- 0
repeat{
  k <- k+1
  theta0 <- theta # 上一个 $\theta$ 值
  theta <- 1/(n+m)*(sumx1 + m*(theta0 + s) - r*s/(1 - exp(-s/theta0)))
  if(abs(theta - theta0) < eps || k >= max.iter) break
}
cat(" 迭代次数 =", k, " EM 估计 =", theta, "\n")
```

```
## 迭代次数= 7  EM 估计= 1.800224
```

图形验证：

```
logL <- function(th){
  (-n * log(th) + r*log(1 - exp(-s/th))
   - 1/th*(sumx1 + (m-r)*s) )
}
curve(logL, 1.0, 3.0)
abline(v=theta, col="red")
```



※※※※※

例 39.6 (正态分布右删失 (*))**.** 设总体为 N(θ,1) 分布，观测样本 $X_1,\dots,X_n$ 有观测值 $x_1,\dots,x_n$, 另外有 $Z_1,\dots,Z_m$ 仅知道 $Z_j > a, j = 1,\dots,m$，称 Z_j 是右删失观测。假设这些观测都相互独立，则实际的似然函数为

$$L(\theta) = f(x|\theta) = [1-\Phi(a-\theta)]^m \prod_{i=1}^{n} \phi(x_i-\theta), \tag{39.10}$$

其中 $\phi(\cdot)$ 和 $\Phi(\cdot)$ 分别为标准正态分布的分布密度和分布函数。

用 EM 算法求最大似然估计，以 $X = (X_1,\dots,X_n)$ 为有观测的部分，以 $Z = (Z_1,\dots,Z_m)$ 为没有观测的部分。完全数据的联合密度为

$$f(x,z|\theta) = \prod_{i=1}^{n} \phi(x_i-\theta) \prod_{j=1}^{m} \phi(z_j-\theta).$$

在 E 步，设 $\theta^{(t)}$ 为参数的一个估计值，在已知 $X = x$ 条件下，Z 的条件密度可以从完全数据联合密度导出:

$$f(z|x,\theta^{(t)}) = \frac{f(x,z|\theta^{(t)})}{f(x|\theta^{(t)})} = [1-\Phi(a-\theta^{(t)})]^{-m} \prod_{j=1}^{m} \phi(z_j-\theta^{(t)}),$$

这个条件分布可以看成是与 X 独立，且 $Z_1,\dots,Z_m$ 独立，有共同的条件密度 $\phi(z-\theta^{(t)})/[1-\Phi(a-\theta^{(t)})]$, $z>a$。求完全数据对数似然关于 Z 的条件期望, 有

$$\begin{aligned}Q_t(\theta) =& E\log\left[\prod_{i=1}^{n}\phi(x_i-\theta)\prod_{j=1}^{m}\phi(Z_j-\theta)\right]\\ =&\sum_{i=1}^{n}\log\phi(x_i-\theta)+\sum_{j=1}^{m}E\log\phi(Z_j-\theta),\end{aligned}$$

其中 Z_j 独立，有共同密度 $\phi(z-\theta^{(t)})/[1-\Phi(a-\theta^{(t)})]$,$z>a$, 所以

$$Q_t(\theta)=\sum_{i=1}^{n}\log\phi(x_i-\theta)+m\int_a^{\infty}\log\phi(z-\theta)\cdot\frac{\phi(z-\theta^{(t)})}{1-\Phi(a-\theta^{(t)})}\,dz.$$

在 M 步，要求 $Q_t(\theta)$ 的最大值点。注意 $\frac{d}{dx}\log\phi(x)=\phi'(x)/\phi(x)=-x$, 得

$$\begin{aligned}Q_t'(\theta) =&\sum_{i=1}^{n}(x_i-\theta)+m\int_a^{\infty}(z-\theta)\frac{\phi(z-\theta^{(t)})}{1-\Phi(a-\theta^{(t)})}\,dz\\ =&n\bar{X}-n\theta+m\int_a^{\infty}(z-\theta^{(t)})\frac{\phi(z-\theta^{(t)})}{1-\Phi(a-\theta^{(t)})}\,dz+m(\theta^{(t)}-\theta)\\ =&n\bar{X}-n\theta+m\theta^{(t)}-m\theta+m\frac{1}{1-\Phi(a-\theta^{(t)})}\int_{a-\theta^{(t)}}^{\infty}u\phi(u)\,du\\ &(\text{令}u=z-\theta^{(t)})\\ =&-(n+m)\theta+n\bar{X}+m\theta^{(t)}+m\frac{\phi(a-\theta^{(t)})}{1-\Phi(a-\theta^{(t)})}.\end{aligned}$$

令 $Q_t'(\theta)=0$ 得

$$\theta^{(t+1)}=\frac{1}{n+m}\left[n\bar{X}+m\theta^{(t)}+m\frac{\phi(a-\theta^{(t)})}{1-\Phi(a-\theta^{(t)})}\right]\tag{39.11}$$

为 $Q_t(\theta)$ 的最大值点。选取适当初值 $\theta^{(0)}$ 按(39.11)迭代就可以求得带有右删失数据的正态总体参数 θ 的最大似然估计。

※※※※※

例 39.7 (混合分布)**.** EM 算法可以用来估计混合分布的参数。设随机变量 $Y_1\sim \mathrm{N}(\mu_1,\delta_1)$, $Y_2\sim \mathrm{N}(\mu_2,\delta_2)$, Y_1,Y_2 独立。记 $N(\mu,\delta)$ 的密度为 $f(x|\mu,\delta)$。设随机变量 $W\sim \mathrm{b}(1,\lambda)$, $0<\lambda<1$, W 与 Y_1,Y_2 独立，令

$$X=(1-W)Y_1+WY_2,$$

则 $W=0$ 条件下 $X\sim \mathrm{N}(\mu_1,\delta_1)$, $W=1$ 条件下 $X\sim \mathrm{N}(\mu_2,\delta_2)$, 但 X 的边缘密度为

$$f(x|\theta)=(1-\lambda)f(x|\mu_1,\delta_1)+\lambda f(x|\mu_2,\delta_2),$$

其中 $\theta=(\mu_1,\delta_1,\mu_2,\delta_2,\lambda)$。

设 X 有样本 $X=(X_1,\dots,X_n)$, 样本值为 x，实际观测数据的似然函数为

$$L(\theta)=\prod_{i=1}^{n} f(x_i|\theta),$$

这个函数是光滑函数但是形状很复杂，直接求极值很容易停留在局部极值点。

用 EM 算法，以 $W=(W_1,\dots,W_n)$ 为没有观测到的部分, 完全数据的似然函数和对数似然函数为

$$\begin{aligned}
\tilde{L}(\theta|x,W)=&\prod_{W_i=0} f(x_i|\mu_1,\delta_1)\ \prod_{W_i=1} f(x_i|\mu_2,\delta_2)\ \lambda^{\sum_{i=1}^n W_i}(1-\lambda)^{n-\sum_{i=1}^n W_i},\\
\tilde{l}(\theta|x,W)=&\sum_{i=1}^{n}\Big[(1-W_i)\log f(x_i|\mu_1,\delta_1)+W_i\log f(x_i|\mu_2,\delta_2)\Big]\\
&+\left(\sum_{i=1}^{n} W_i\right)\log\lambda+\left(n-\sum_{i=1}^{n} W_i\right)\log(1-\lambda).
\end{aligned}$$

在 E 步，设已有 θ 的近似值 $\theta^{(t)}=(\mu_1^{(t)},\delta_1^{(t)},\mu_2^{(t)},\delta_2^{(t)},\lambda^{(t)})$, 以 $\theta^{(t)}$ 为分布参数，在 $X=x$ 条件下，W_i 的条件分布为

$$\begin{aligned}
\gamma_i^{(t)}\triangleq& P(W_i=1|x,\theta^{(t)})=P(W_i=1|X_i=x_i,\theta^{(t)})\\
=&\frac{\lambda^{(t)}f(x_i|\mu_2^{(t)},\delta_2^{(t)})}{(1-\lambda^{(t)})f(x_i|\mu_1^{(t)},\delta_1^{(t)})+\lambda^{(t)}f(x_i|\mu_2^{(t)},\delta_2^{(t)})}.
\end{aligned}\tag{39.12}$$

这里的推导类似于逆概率公式。利用 W_i 的条件分布求完全数据对数似然的期望，得

$$\begin{aligned}
Q_t(\theta)=&\sum_{i=1}^{n}\Big[(1-\gamma_i^{(t)})\log f(x_i|\mu_1,\delta_1)+\gamma_i^{(t)}\log f(x_i|\mu_2,\delta_2)\Big]\\
&+\left(\sum_{i=1}^{n}\gamma_i^{(t)}\right)\log\lambda+\left(n-\sum_{i=1}^{n}\gamma_i^{(t)}\right)\log(1-\lambda).
\end{aligned}$$

令 $\nabla Q_t(\theta)=0$，求得 $Q_t(\theta)$ 的最大值点 $\theta^{(t+1)}$ 为

$$\begin{cases}\mu_1^{(t+1)}=\frac{\sum_{i=1}^n(1-\gamma_i^{(t)})x_i}{\sum_{i=1}^n(1-\gamma_i^{(t)})}\\ \delta_1^{(t+1)}=\frac{\sum_{i=1}^n(1-\gamma_i^{(t)})(x_i-\mu_1^{(t+1)})^2}{\sum_{i=1}^n(1-\gamma_i^{(t)})}\\ \mu_2^{(t+1)}=\frac{\sum_{i=1}^n\gamma_i^{(t)}x_i}{\sum_{i=1}^n\gamma_i^{(t)}}\\ \delta_2^{(t+1)}=\frac{\sum_{i=1}^n\gamma_i^{(t)}(x_i-\mu_2^{(t+1)})^2}{\sum_{i=1}^n\gamma_i^{(t)}}\\ \lambda^{(t+1)}=\frac{1}{n}\sum_{i=1}^n\gamma_i^{(t)}\end{cases}\tag{39.13}$$

适当选取初值 $\theta^{(0)}$ 用(39.12)和(39.13)迭代就可以计算 θ 的最大似然估计。

※※※※※

习题

习题 1

设 $f(x)$, $g(x)$ 是定义在集合 A 上的两个密度，$f(x)$, $g(x)$ 在 A 上都为正值。证明如下信息不等式:

$$\int_A[\log f(x)]f(x)\,dx\geq\int_A[\log g(x)]f(x)\,dx.$$

习题 2

设总体 X 服从如下的 logistic 分布:

$$f(x|\theta)=\frac{\exp(-(x-\theta))}{(1+\exp(-(x-\theta)))^2},\ -\infty<x<\infty,\ -\infty<\theta<\infty,$$

设 $X=(X_1,\dots,X_n)$ 为 X 的简单随机样本。

(1) 写出对数似然函数 $l(\theta)$ 和导数 $l'(\theta)$。
(2) 证明方程 $l'(\theta)=0$ 存在唯一解，解为最大似然估计。
(3) 写出用牛顿法求解方程 $l'(\theta)=0$ 的迭代公式。
(4) 设一个样本为 $-0.63,0.18,-0.84,1.6,0.33,-0.82,0.49,0.74,0.58,-0.31$, 计算 $\widehat{\theta}$。

习题 3

设总体 X 取值于正整数集合 $\mathbb{Z}_+ \triangleq \{1,2,3,\dots\}$，有概率函数

$$p(x;\theta) = \theta^x x^{-1}[-\log(1-\theta)]^{-1},\ x = 1,2,\dots,$$

其中 $\theta \in (0,1)$ 为未知参数。设 $x_1, x_2, \dots, x_n$ 为 X 的简单随机样本，设 $\sum_{i=1}^n x_n = s$。

(1) 令 $f(\theta)$ 为负对数似然函数，写出 $f(\theta)$ 和 $g(\theta) = f'(\theta)$ 以及 $g'(\theta)$ 的表达式。
(2) 设 $n = 10$, $s = 15$, 编写程序，作 $f(x)$ 和 $g(x)$ 的图像;
(3) 编写程序，分别用二分法、牛顿法和割线法求 θ 的最大似然估计 (即 $f(\theta)$ 的最小值点)。

习题 4

考虑如下最大似然估计计算问题。设 X 取值于 $\{1,2,3,4\}$, 分布概率为 $p(x|\theta_1,\theta_2) \triangleq \pi_x(\theta_1,\theta_2)$:

$$\pi_1(\theta_1,\theta_2) = 2\theta_1\theta_2, \qquad \pi_2(\theta_1,\theta_2) = \theta_1(2-\theta_1-2\theta_2),$$
$$\pi_3(\theta_1,\theta_2) = \theta_2(2-\theta_2-2\theta_1), \qquad \pi_4(\theta_1,\theta_2) = (1-\theta_1-\theta_2)^2.$$

设 n 次试验得到的 X 值有 n_j 个 $j(j = 1,2,3,4)$，则对数似然函数为 (去掉了与参数无关的加性常数)

$$l(\theta_1,\theta_2) = \sum_{j=1}^4 n_j \log \pi_j(\theta_1,\theta_2)$$

设 $n = 435$, $n_1 = 17$, $n_2 = 182$, $n_3 = 60$, $n_4 = 176$。编写程序，分别用牛顿法、阻尼牛顿法、BFGS 方法、得分法和 Nelder-Mead 方法求最大似然估计，比较这几种方法的收敛速度。

习题 5

设 $Y_i \sim \mathrm{B}(m_i, \mathrm{logit}^{-1}(\beta_0 + \beta_1 x_i))$, $i = 1,\dots,n$ 相互独立，$n = 4$，(m_i, y_i, x_i) 的 4 组数据为 $(55,0,7)$, $(157,2,14)$, $(159,7,27)$, $(16,3,51)$。编写程序用牛顿法求 $\beta = (\beta_0,\beta_1)$ 的最大似然估计。写出模型的对数似然函数及其梯度、海色阵，写出最大似然估计的牛顿法迭代公式。

习题 6

设随机变量 $Y_i \sim \text{Poisson}(\exp[\beta^T x_i])$, $i = 1, 2, \ldots, n$, $Y_1, \ldots, Y_n$ 相互独立，β 为未知参数向量，自变量 $x_i, i = 1, \ldots, n$ 已知。写出模型的对数似然函数及其梯度、海色阵，写出最大似然估计的牛顿法迭代公式。

习题 7

写出例39.2中的简化似然函数 $\tilde{l}(\alpha)$，编写程序，分别用二分法和牛顿法求 $\tilde{l}(\alpha)$ 的最大值点。设有样本 $0.08, 0.36, 0.35, 0.21, 0.39, 0.25, 0.23, 0.11, 0.07, 0.08$, 求参数最大似然估计。

习题 8

设总体 X 服从威布尔分布 $\text{W}(\alpha, \eta)$，密度函数为

$$p(x; \alpha, \eta) = \begin{cases} \frac{\alpha}{\eta}\left(\frac{x}{\eta}\right)^{\alpha-1} \exp\left\{-\left(\frac{x}{\eta}\right)^{\alpha}\right\}, & x > 0, \alpha > 0, \eta > 0, \\ 0, & \text{其它} \end{cases}$$

设 $X_1, X_2, \ldots, X_n$ 为 X 的简单随机样本。

(1) 求对数似然函数 $l(\alpha, \eta)$ 及其梯度 $\nabla l(\alpha, \eta)$、海色阵 $\nabla^2 l(\alpha, \eta)$;
(2) 编写程序，用牛顿法求最大似然估计；
(3) 用分步求最大值的方法，先对固定 α 求最大值点 $\hat{\eta}(\alpha)$，再关于 α 求最大值，编写 R 程序实现算法；
(4) 用随机模拟方法生成多组样本，比较两种算法方法的收敛速度。比较随机模拟得到的最大似然估计的方差与牛顿法结束迭代时用海色阵得到的渐近方差。

习题 9

考虑如下非线性回归模型：

$$y_i = A\cos(\omega x_i + \theta) + \varepsilon_i, \ i = 1, 2, \ldots, n,$$

其中 ε_i 独立同 $\mathrm{N}(0,\sigma^2)$ 分布，A,ω,θ,σ^2 为未知参数。设 $n=100$, $x_i=2\pi i/n$, $A=10$, $\omega=2$, $\theta=0.5$, $\sigma^2=2^2$。编写程序，生成随机模拟样本, 分别用高斯–牛顿法和 LMF 方法给出参数最小二乘估计。

附录 A

R 软件基础 (*)

R 是一个用于统计计算、绘图和数据分析的自由软件，最初由新西兰 Auckland 大学的 Ross Ihaka 和 Robert Gentleman 于 1997 年发布，现在由 R 核心团队开发维护，全世界的用户都可以贡献软件包。R 软件使用 R 语言，R 语言直接提供了向量、矩阵、数据框、一般对象等高级数据结构，其语法简单易学，功能强大，尤其适用于开发新的统计算法, 用 R 语言编写程序就像写公式一样简单，也可以调用 C、C++、Fortran 等编译语言的代码实现附加功能或提高计算效率。R 软件通过附加软件包 (package) 的方式提供了各种经典的统计方法以及最新的统计方法，R 的网站 http://www.r-project.org/上已经有超过 1.8 万个软件包（截止 2022 年 4 月），还有一些软件包发布在 github、Bioconductor 等网站。R 软件已经成为全世界统计学家研究新算法和进行统计计算的首选软件之一，在生物、金融、医药、工农业等各行各业的数据分析中也获得了广泛应用。

R 语言可以看成是 S 语言的一个变种，S 语言是 Rick Becker, John Chambers 等人在贝尔实验室开发的一个用于交互数据分析和作图的软件，最初的版本开发于 1976-1980 年，后来又有改进（见 [Becker and Chambers, 1984], [Becker et al., 1988], [Chambers and Hastie, 1992]）。

这一节讲述 R 软件的基础用法, 这样读者可以用 R 语言实现自己的算法, 本书的算法也用类似 R 语言的语法描述。R 软件的进一步使用需要读者自己阅读 R 手册和 R 软件的相关书籍，可参考李东风的统计软件教程。

A.1 向量

R 软件可以在图形界面内以命令方式交互运行，也可以整体地运行存放在扩展名为.R 或.r 的程序文件中的代码。如果有某个 R 程序存放在当前目录中的 mysrc.R 文件中，如下命令可以运行这样的程序文件：

```
source("mysrc.R")
```

R 语言最基本的数据类型是**向量** (vector)，这是 R 的**对象**中最简单的一种。标量（单个的数值或字符串）可以看成是长度为 1 的向量。数值型的 R 向量基本可以看成是线性代数中的向量，但不区分行向量和列向量；与其它程序语言相比，R 向量相当于其它程序语言中的一维数组, 可以保存若干个相同基本类型的值，各个元素用向量名和序号（下标）访问，下标从 1 开始计数。

R 中用函数 c() 来把若干个元素组合为一个向量，如

```
marks <- c(10, 6, 4, 7, 8)
```

定义了一个长度为 5、数据类型为数值 (numeric) 的向量，保存在名为 marks 的**变量**中。

R 中把向量等数据保存在变量中，如上例的 marks，用 <-为变量赋值。R 中的变量名可以由字母、数字、句点、下划线组成，变量名的第一个字符只能是字母或句点，如果以句点开头必须有第二个字符并且第二个字符不能是数字, 变量名长度不限。变量名是大小写区分的，所以 Amap 和 amap 是不同的名字。

为了访问向量 x 的第 3 个元素，只要用 x[3] 这种方法，下标是从 1 开始计数的。也可以直接修改一个元素，如

```
marks[3] <- 0
```

在 R 命令行环境中输入变量名然后按 Enter 键可以查看变量的值。如果在程序文件中要显示变量值 x 的值，应当使用 print(x)。更一般的输出可以用 cat 函数显示文本和数据值，如

```
cat(" 第三个分数 =", marks[3], "\n")
## 第三个分数 = 0
```

其中"\n" 表示换行。cat 的各项输出自动用空格分隔，在 cat 中加 sep="" 选项表示各项之间不分隔。

可以用 “start:stop” 的方法定义一个由连续整数值组成的向量，比如 11:13 为向量 $(11, 12, 13)$。用 rep(x, times) 可以把向量 x 重复 times 次，比如 rep(c(1,3), 2) 结果为向量 $(1, 3, 1, 3)$。用 rep(x, each=...) 可以把向量 x 的每个元素重复 each 次, 如 rep(c(1,3), each=2) 结果为向量 $(1, 1, 3, 3)$。

如果一个文本文件中用空格和换行分隔保存了多个数值，用函数 scan 可以把这些数值读入到一个数值型向量中，如

```
x <- scan('mydata.txt')
```

其中 mydata.txt 是当前目录下的一个文本文件，保存了用空格和换行分隔的数值。

R 的基本数据类型包括数值型 (numeric)、字符型 (character)、逻辑型 (logical)、复数型 (complex)。为了定义一个指定长度、元素初始化为 0 数值型向量，方法如

```
x <- numeric(4)
```

则变量 x 保存了由 4 个零组成的数值型向量。

字符型常量可以用两个单撇号或两个双撇号界定，如

```
tnames <- c('王思明', "John Kennedy")
```

R 的逻辑型只有真值 TRUE、假值 FALSE 和缺失值 NA 三个不同取值。

复数型常量写成如 1.2 - 3.3i 这样的形式，虚数单位 i 可以写成 1i。

和 C、C++、Fortran、Java 这些严格类型的程序设计语言不同，R 的变量不需要事先声明数据类型。为了动态地获知某个向量 x 当前所保存的基本数据类型，可

以调用函数 typeof(x)，结果为字符串"integer"、"double"、"character"、"logical" 或 "complex"。"integer"、"double" 可以归为"numeric" 类别。基本类型之间可以用 as.xxx 类的函数进行转换，比如 as.numeric(c(TRUE, FALSE)) 结果为数值型向量 $(1, 0)$。

向量元素中用 NA 表示缺失值，如

```
x <- c(1, NA, 3)
s <- c(" 王思明", NA, "John Kennedy")
```

但是没有上下文的一个 NA 被认为是逻辑型的。

向量是 R 对象中的一种。R 对象用**属性**用来区分不同数据类型或者对数据附加额外描述信息，基本类型 (mode) 就是所有 R 对象都有的基本属性，另一个基本属性是长度 (length)，用函数 length(x) 可以返回向量 x 的元素个数。

为了访问 R 向量的子集，除了一个元素可以用 “ ” 的格式访问之外，还可以用一个正整数向量作为下标，比如 marks[c(1,5)] 结果为 $(10, 8)$,

```
marks[3:5] <- 0
```

把 marks 的第 3 到第 5 号元素赋值为零。

向量下标还可以取为负整数或负整数向量，如 (11:15)[-5] 是 (11:15) 去掉了第 5 个元素之后的结果 $(11, 12, 13, 14)$, (11:15)[-c(1,5)] 是 (11:15) 去掉了第 1 和第 5 号元素之后的结果 $(12, 13, 14)$。

R 向量可以定义元素名并使用元素名作为下标，比如

```
marks <- c(" 李明"=10, " 张红艺"=6, " 王思明"=4,
           " 张聪"=7, " 刘颖"=8)
```

则可以用得到元素值 7，用得到结果 $(10, 7)$。

所有元素的元素名组成一个字符型向量，这是向量的对象属性之一。可以用 names(x) 取得向量 x 的元素名向量，也可以用如

```
names(marks) <- c(" 李明", " 张红艺", " 王思明", " 张聪", " 刘颖")
```

这样的方法定义或修改元素名属性。注意，这里赋值符号左边的 `names(marks)` 是对象属性的一种表示方法，不能看成是给一个函数结果赋值。

R 程序允许自动续行。只要前一行程序明显地没有完成，就可以直接拆分到下一行，没有配对的括号是前一行程序没有完成的一种常见情况。在一行的开头或中间以 # 号开始注释，# 以及该行的后续内容都作为注释。另外，两个语句可以用分号分隔后写在同一行中，比如

```
x <- 1; y < 2; z <- x+y
```

因子 (factor) 是一种特殊的向量：元素只能在有限个“水平”中取值，元素值编码为正整数值保存，可以用来表示分类变量的观测值。例如

```
fac <- factor(c(" 男", " 男", " 女", " 男"))
```

生成一个水平为“男”和“女”的因子，用 `levels(fac)` 可以查看其各个水平，用 `as.numeric(fac)` 可以查看其编码值，此例为 $(1,1,2,1)$；用 `as.character(fac)` 可以转换为字符型。

A.2 向量运算

R 的算术运算符为

```
+  -  *  /  %%  %/%  ^
```

分别表示加、减、乘、除、除法余数、整除、乘方。算术运算的优先级为先乘方、再乘除、最后加减，可以用圆括号来改变运算次序。有缺失值参加的四则运算结果还是缺失值。

向量可以和一个标量进行算术运算，结果为向量的每个元素与标量进行计算得到的新向量。如 `(11:14) + 100` 的结果为向量 $(111,112,113,114)$，`2^(0:4)` 的结果为向量 $(1,2,4,8,16)$。

两个长度相同的向量进行算术运算，结果为对应元素进行算术运算后得到的新向量。比如 `(11:14) - (1:4)` 结果为向量 $(10, 10, 10, 10)$。

如果两个向量长度不同，但较大长度是较小长度的整数倍，则这两个向量也可以进行算术运算，在运算时把短的一个循环重复使用。比如，`(11:14) + c(100, 200)` 结果为向量 $(111, 212, 113, 214)$。

在 R 中，允许对一个向量调用数学中的一元函数，函数输出结果为每个元素取函数值后组成的新向量，比如 `sqrt(c(1,4,9))` 的结果为向量 $(1, 2, 3)$。类似的函数还有 `abs(x)` ($|x|$), `exp(x)` (e^x), `log(x)` ($\ln x$), `log10(x)` ($\lg x$), `sin(x)`, `cos(x)`, `tan(x)`, `asin(x)`(反正弦), `acos(x)`(反余弦), `atan(x)`(反正切), `atan2(y,x)`。`atan2(y,x)` 函数求平面直角坐标系中原点到点 (x, y) 的向量与 x 轴的夹角，对第 I、II 象限和 x 轴上的点，结果取值于 $[0, \pi]$，对第 III、IV 象限的点，结果取值于 $(-\pi, 0)$。

R 的比较运算符为

```
==  !=  <  <=  >  >=
```

可以表示两个标量比较，两个等长向量比较，或两个长度不等但是长度为倍数的两个向量比较，结果是对应元素比较的结果组成的逻辑型向量。如 `(1:4) > (4:1)` 结果为逻辑型向量 `(FALSE, FALSE, TRUE, TRUE)`, `(1:3) > 1.5` 的结果为逻辑型向量 `(FALSE, TRUE, TRUE)`, `(1:4) > (3:2)` 结果为逻辑型向量 `(FALSE, FALSE, FALSE, TRUE)`。

R 用`%in%` 运算符表示元素 “属于” 集合的判断，如果 `a` 是一个标量，`A` 是一个向量，把 `A` 看成一个集合（集合的元素就是 `A` 的各个分量），`a %in% A` 的结果就是 $a \in A$ 是否成立的结果。如果 `B` 是一个向量，`B %in% A` 就是对 `B` 的每个元素分别判断是否属于 `A` 的结果。比如, `c(2,-2) %in% (1:3)` 结果为 `(TRUE, FALSE)`。为了判断向量 `y` 的所有元素都属于向量 `x` 是否成立，可以用 `setdiff(y,x)` 求集合 y 减去集合 x 的差集，如果 `length(setdiff(y,x))==0` 为真则说明向量 `y` 的所有元素都属于向量 `x`。集合运算函数还有 `union(x,y)`(并集)、`intersect(x,y)`(交集)、`setequal(x,y)`(相同集合)、`is.element(el, set)`(属于判断)。

R 定义了如下的逻辑运算符：

```
&  |  !
```

其中 & 表示逻辑与（同时为真才为真），| 表示逻辑或（任一为真则为真），! 表示逻辑非（真变成假，假变成真），两个逻辑向量之间可以进行逻辑运算，含义为对应元素的运算。一般用来把比较运算连接起来得到复杂条件，比如

```
(age >= 18) & (age <= 59)
```

表示年龄在 18 和 59 之间（含 18 和 59），

```
sex==" 女" | age <= 3
```

表示妇女或年龄在 3 岁以下（含 3 岁），

```
!((age >= 18) & (age <= 59))
```

表示年龄在 18 岁以下或 59 岁以上（不含 18 和 59）。

为了表示所有元素都满足一个条件，用函数 all()，如 all(age >= 18) 表示向量 age 中所有年龄都在 18 岁以上。类似地，函数 any() 表示向量中有任一元素满足条件。

R 中比较和逻辑运算的一个重要作用是按条件挑选向量元素子集。比如，要挑选向量 marks 中分数在 5 分以下（含 5 分）的元素，可以用如

```
marks[marks <= 5]
## 王思明
##      4
```

这是因为 R 的向量下标允许取为和向量长度相同的一个逻辑向量。这样取子集可能取到空集，结果表示为 numeric(0)，即长度为零的数值型向量。

A.3 矩阵

R 语言支持矩阵 (matrix) 和数组 (array) 类型, 矩阵是数组的特例。数组有一个属性 dim, 比如

```
arr <- array(1:24, dim=c(2,3,4))
```

定义了一个数组，元素分别为 1 到 24，每个元素用 3 个下标访问，第一个下标可取 1 和 2, 第二个下标可取 1, 2, 3, 第三个下标可取 1, 2, 3, 4。数组元素在填入各下标位置时，第一下标变化最快，第三下标变化最慢，这种排序叫做 FORTRAN 次序或列次序。比如，arr 中元素的次序为 arr[1,1,1], arr[2,1,1], arr[1,2,1], arr[2,2,1], ..., arr[2,3,4]。

矩阵是 dim 属性为两个元素的向量的数组，矩阵元素用两个下标访问，第一下标可以看作是行下标，第二下标可以看作列下标，元素按列优先次序存储。因为矩阵是最常用的数组，所以单独提供一个 matrix 函数用来定义矩阵，如

```
M <- matrix(1:6, nrow=2, ncol=3); M
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

为了让矩阵元素按行填入，定义时可指定 byrow=TRUE，如

```
M <- matrix(1:6, nrow=2, ncol=3, byrow=TRUE); M
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

单个数组元素用两个下标访问，如上述 M 的第 2 行第 3 列元素用 M[2,3] 访问。

可以取出 M 的一行或一列组成一个向量，格式如 M[1,], 表示 M 的第一行组成的向量（不再是矩阵，所以不区分行向量和列向量），M[,2] 表示 M 的第二列组成的向量。

可以取出若干行或若干列组成子矩阵，比如 M[,2:3] 取出 M 的第 2 列和第 3 列组成的 2×2 子矩阵。取出子矩阵时如果仅有一行或仅有一列，子矩阵会退化成 R 向量而不再有维数属性，可以用 M[,1,drop=FALSE] 这样的方法要求取子集时数组维数不变。

用函数 `cbind` 可以把一个向量转换为列向量（列数等于 1 的矩阵），或者把若干个向量、矩阵横向合并为一个矩阵。用函数 `rbind` 可以转换行向量或进行纵向合并。

矩阵有两维（行维、列维），每一维都可以定义维名当作下标使用。对上述矩阵 `M`，如下命令定义了矩阵列名：

```
colnames(M) <- c("X1", "X2", "X3")
```

这样，`M` 的第二列可以用 `M[,"X2"]` 访问，`M` 的第一列和第二列组成的子矩阵可以用 `M[,c("X1", "X3")]` 访问。类似地可以定义行名并用行名代替行下标：

```
rownames(M) <- c("John", "Mary")
```

这时 `M` 显示如下：

```
M
##      X1 X2 X3
## John  1  2  3
## Mary  4  5  6
```

列名和行名没有命名规则限制，但不应有重复值以免不能唯一地标识列和行。

矩阵首先是二维数组，形状相同的矩阵可以进行对应元素之间的四则运算，运算符与向量的四则运算符相同。比如，设

```
M1 <- rbind(c(1,-1,1), c(-1,1,1))
```

则

```
M + M1
##      X1 X2 X3
## John  2  1  4
## Mary  3  6  7
M - M1
##      X1 X2 X3
```

```
## John  0  3  2
## Mary  5  4  5
M * M1
##      X1 X2 X3
## John  1 -2  3
## Mary -4  5  6
M / M1
##      X1 X2 X3
## John  1 -2  3
## Mary -4  5  6
M ^ M1
##        X1  X2 X3
## John 1.00 0.5  3
## Mary 0.25 5.0  6
```

分别对 M 和 M2 的对应元素作加法、减法、乘法、除法、乘方，结果是形状相同的矩阵。对于加法和减法，结果与线性代数中两个矩阵相加和相减是一致的。

矩阵可以和标量作四则运算，结果是矩阵的每个元素与该标量进行四则运算。如果是矩阵与标量相乘，结果与线性代数中数乘结果相同。

两个矩阵相乘用%*% 运算符表示。比如

```
M %*% t(M1)
##      [,1] [,2]
## John    2    4
## Mary    5    7
```

表示矩阵 M 左乘矩阵 M1 的转置 (函数 t() 表示矩阵转置)。另外，crossprod(A) 表示 $A'A$, crossprod(A, B) 表示 $A'B$。

如果 A 是可逆方阵，用 solve(A) 求逆矩阵 A^{-1}。对线性方程组 $Ax = b$，用 solve(A, b) 可以求出线性方程组的解 x。

A.4 数据框

数据框 (data frame) 是类似于矩阵的数据类型，但是它允许同时保存数值型、字符型和因子型数据，每列的类型必须相同。数据框每列必须有变量名，用 `names(df)` 访问数据框 `df` 的各个变量名。统计数据经常以数据框的形式保存。

用函数 `read.csv` 可以把 CSV(逗号分隔) 格式的文件转换为 R 数据框，用函数 `write.csv` 可以把 R 数据框保存为 CSV 格式的文件。

A.5 分支和循环

R 语言因为内置了向量、矩阵这样的高级数据类型，所以编程比较容易。很多其它语言中必须用分支、循环结构解决的问题在 R 语言中可以用更简洁的代码处理。

R 中用

```
if(条件) ... else ...
```

处理分支结构，其中“条件”应该是一个逻辑型标量。`else` 部分可省略, 但是有 `else` 部分时两部分必须在同一个语句中。例如

```
if(x>0) y <- 1  else y <- 0
```

也可写成

```
if(x>0){
  y <- 1
} else {
  y <- 0
}
```

上例中使用了**复合语句**。R 中可以用左右大括号把若干个语句组合成一个复合语句，在程序中当作一个语句使用。

if 结构中的条件必须是标量。在上述问题中如果 x 是一个向量，可以改用如下的逻辑下标的方法：

```
y <- numeric(length(x))
y[x>0] <- 1
```

这样可以根据 x 元素的不同取值而对 y 的相应元素赋值。

R 语言中用 for 结构进行对向量下标或向量元素进行遍历（循环），格式为

```
for(idvar in 范围){
  循环体语句
  }
```

其中 idvar 是循环变量，“范围”是下标向量或一般向量。比如，下例中对向量 x 的下标循环以计算元素总和：

```
x <- 11:15
s <- 0
for(k in seq(along=x)){
  s <- s + x[k]
  cat('k=', k, 'x[k]=', x[k], 's=', s, '\n')
}
## k= 1 x[k]= 11 s= 11
## k= 2 x[k]= 12 s= 23
## k= 3 x[k]= 13 s= 36
## k= 4 x[k]= 14 s= 50
## k= 5 x[k]= 15 s= 65
```

其中 seq(along=x) 获得向量 x 的下标向量（本例中为 1:5）。使用 seq(along=x) 而不是 1:length(x)，是因为 x 可以是零长度的，这时 1:length(x) 为 1:0, 即 1 和 0 两个元素，是错误的。

还可以直接对 x 的元素循环，如

```
x <- 11:15
s <- 0
for(item in x){
  s <- s + item
  cat('item=', item, 's=', s, '\n')
}
## item= 11 s= 11
## item= 12 s= 23
## item= 13 s= 36
## item= 14 s= 50
## item= 15 s= 65
```

在不能预知循环次数时，可以使用**当型循环**，格式为

```
while(循环继续条件){
  循环体语句
}
```

当循环继续条件成立时反复执行循环体语句，直到条件不成立时才不再执行。如果一开始循环继续条件就不成立，就一次也不执行循环体语句。

在计算机算法中还有所谓 * 直到型循环 **，R 没有直接提供直到型循环的语法结构，在本书的算法描述中有时用如下的直到型结构:

```
until(循环退出条件){
  循环体语句
}
```

执行循环直到循环退出条件成立时才退出循环，循环至少执行一次。R 中可以用如下方法模仿直到型循环:

```
repeat{
  循环体语句
  if(循环退出条件) break
}
```

其中 repeat 是无条件循环语句，break 跳出一重循环。

在本书的算法描述中，使用了类似于 R 语言语法的**伪代码**, 伪代码与程序类似，但允许使用描述性的操作说明。伪代码中主要用到 if ... else ..., for 循环, while 循环，until 循环等控制结构。

在 R 中，for、while、repeat 循环效率较低，比用 FORTRAN、C 等编译型语言的计算速度可能慢几个数量级。R 中的循环常常是可以设法避免的。比如，sum(x) 可以直接得到 x 的元素和。类似的函数还有 mean(x) 求平均值，prod(x) 求所有元素的乘积，min(x) 求最小值，max(x) 求最大值，sd(x) 求样本标准差，cumsum(x) 计算元素累加和（包括各中间结果），cumprod(x) 计算元素累乘积，等等。

R 的 apply、lapply、sapply、mapply、vapply、replicate、tapply、rapply 等函数隐含了循环。函数 apply 可以对矩阵的各行或各列循环处理，比如，apply(M, 1, sum) 表示对 M 的各行求和，而 apply(M, 2, mean) 表示对 M 的各列求平均。函数 lapply(X,FUN, ...) 可以把函数 FUN 分别作用于 X 的每一个元素 (比如数据框的每个变量)，输出一个包含这些作用结果的列表，... 是 FUN 所需要的其它自变量。sapply(X,FUN,...) 与 lapply(X,FUN...) 类似但尽可能把结果转换为向量或数组。vapply 与 sapply 作用类似，但需要自己指定输出类型。mapply(FUN, ...) 是把 sapply 推广到 FUN 为多元函数的情况。replicate(n, expr) 经常用于重复 n 次模拟，expr 是调用一次模拟的表达式。rapply 可以把函数作用到嵌套列表的最底层。

R 中提供了若干个函数可以进行快速计算，如 fft 计算离散傅立叶变换，convolve 计算离散卷积，filter 计算滤波或自回归迭代。R 这样的函数很多，需要用户能够根据函数的帮助信息，自己构造一些简单可手工验证算例，快速掌握这些函数的使用。

对于必须使用循环的情况，如果需要重复的次数不多，用 R 的循环不会造成效率损失；如果重复次数很多，可以考虑把重复很多的循环改为 C 代码或 C++ 代码，R 的 Rcpp 软件包支持在 R 代码内嵌入 C 代码和 C++ 代码。

A.6 函数

R 内置了许多函数，比如，`sin(x)` 可以计算向量 `x` 的每个元素的正弦值，`seq(a,b)` 可以生成从 `a` 到 `b` 的等差数列，`sum(x)` 可以计算 `x` 的所有元素的总和。

R 的函数在调用时可以按照自变量位置调用，如 `seq(2,5)` 结果为 (2,3,4,5)，也可以在调用时指定自变量名，这时自变量次序不再重要，如 `seq(to=5, from=2)` 结果还是 (2,3,4,5)；还可以前一部分按自变量位置对应，后一部分指定自变量名，如 `seq(2, length=4)` 结果仍为 (2,3,4,5)。

模块化程序设计是保证软件正确、可复用、易升级的重要方法，而自定义函数是模块化设计的主要组成部分。在 R 中用如下格式定义一个新的函数：

```
函数名 <- function(形式参数表){
  函数体
}
```

其中函数体内最后一个表达式的值是函数的返回值。可以为空，表示不需要自变量，但是括号不能省略。形式参数表可以是用逗号分开的形式参数名，并可以带有用等于号给出的缺省值。例如

```
demean <- function(x, xbar=mean(x)){
  x - xbar
}
```

调用如 `demean(c(1,2,3))`, 则缺省参数 `xbar` 用给出的表达式计算，返回值为 (-1,0,1)。还可以调用如 `demean(c(1,2,3), xbar=1)` 或 `demean(c(1,2,3), 1)`，结果为 (0,1,2)。

函数只能返回一个表达式的结果，可以是标量、向量、矩阵、数据框或其它更复杂的类型。为了能够同时返回多个结果，可以把这些结果组合成为一个**列表** (list) 类型。列表定义如

```
list(coefficients=b, Fvalue=F, pvalue=pv)
```

则此列表包含了回归系数向量、检验的 F 统计量和相应的 p 值。设上述列表保存在列表变量 `li` 中,为了访问 `li` 的 `pvalue` 元素,可以用 `li[["pvalue"]]`、`li$pvalue` 或 `li[[3]]` 的格式。用 `names(li)` 查询或修改列表的元素名。为了删除列表中某项，只要将其赋值为特殊的 `NULL` 值，表示不存在的值，`NULL` 与 `NA` 不同，`NA` 表示存在但是缺失的值。

R 自定义函数内，形式参数是局部的，即形式参数和函数定义外其它变量重名时不会有冲突。在函数定义内赋值变量是局部的，即使函数定义外部有同名变量也不会给外部的同名变量赋值。然而，如果在函数定义内读取某个变量的值，此变量的值在函数内无定义但是在它的外部环境中有定义，运行时可以读取外部定义的值，这是一个需要小心的特点，因为在较长的函数定义中可能忘记给变量赋值而使用了外部环境中同名变量的值。

在数据分析时，经常将前一步的输出作为后一步的输入，比如，

```
y1 <- f1(x)
y <- f2(y1)
```

可以简写成

```
y <- f2(f1(x))
```

但这样写就不容易辨认处理步骤是先进行 `f1()` 变换，再进行 `f2()` 变换，R 提供了如下的“管道”运算符用来表示函数复合：

```
y <- x |>
  f1() |>
  f2()
```

也可以有更多的复合运算。

在 `f1()` 和 `f2()` 中也可以有其他选项，如：

```
y <- x |>
  f1(na.rm=TRUE) |>
  f2(simplify=TRUE)
```

A.7 简单输入输出

对一个 R 变量 x，用 print(x) 显示其内容；如果在命令行状态，直接用 x 可以显示变量或表达式 x 的值。

用 cat() 函数显示若干个字符串和向量值，如

```
x <- c(2, 3, 5, 7, 11)
cat(" 前 5 个素数: ", x, "\n")
## 前 5 个素数:  2 3 5 7 11
```

其中最后的"\n" 表示换行。

在 cat() 中加 file= 选项要求将欲显示的内容保存到指定文件中，加选项 append=TRUE 使得写入到文件时在文件末尾添加而不是替换原有内容。如：

```
x <- c(2, 3, 5, 7, 11)
cat(x, file="c:/work/prime5.txt")
```

用 scan() 函数将用空格和换行分隔的保存在文件中的数值读入为向量。如

```
x2 <- scan("c:/work/prime5.txt")
```

用 write.csv() 将一个数据框或矩阵保存为 CSV 格式，这是用逗号分隔两项的格式，第一行是列名，这样的格式可以被 Excel 软件自动判读为表格，也被大多数的数据处理软件支持。例如：

```
df <- data.frame(
  name=c(" 张三", " 李四", " 王五"),
  age=c(30, 28, 34)
```

```
)
write.csv(df, file="empages.csv")
```

用 `read.csv()` 读入一个 CSV 文件。要求第一行是列名，后续行的每一个单元格是一个单独的值，不能有合并格或者公式、图形。如

```
df2 <- read.csv("empages.csv", header=TRUE)
```

A.8 RStudio 介绍

RStudio 是 R 软件的一个功能强大的集成环境，对 R 软件进行了很多功能增强。这是一个商业软件，但提供了免费的功能缩减版本，免费版本也能满足一般用户的需要。尤其适用于用 Rmd 格式制作 HTML、Word、PDF、网站、图书、演示等。

软件的界面分为四个窗格，主要有编辑窗格、命令行窗格、文件窗格等。在屏幕分辨率较低时可以用 Ctrl+Shift+1 等快捷键使得某一个窗格最大化，Ctrl+Shift+1 使得编辑窗口最大化，Ctrl+Shift+2 使得命令行窗口最大化，Ctrl+Shift+5 使得文件窗口最大化，等等。下面是 RStudio 界面的一个截图：

使用 RStudio 进行一个统计研究或者统计数据分析项目，一般将有关的文件存放在计算机的一个子目录（文件夹）中，然后从 RStudio 的文件菜单，建立一个新“项目”（project），将项目与文件存放的子目录关联在一起。子目录中包括数据、R 脚本文件（扩展名.r 或.R）、R markdown 文件（扩展名.Rmd）等。关于使用 RStudio 制作文档、网站、图书、演示稿，参见李东风的R 语言教程。

习题

习题 1

设 x 是一个长度为 n 的向量, 写一段程序, 计算 x 的长度为 s 的滑动和：

$$S_x(t) = \sum_{i=0}^{s-1} x_{t-i}, \quad t = s, s+1, \ldots, n$$

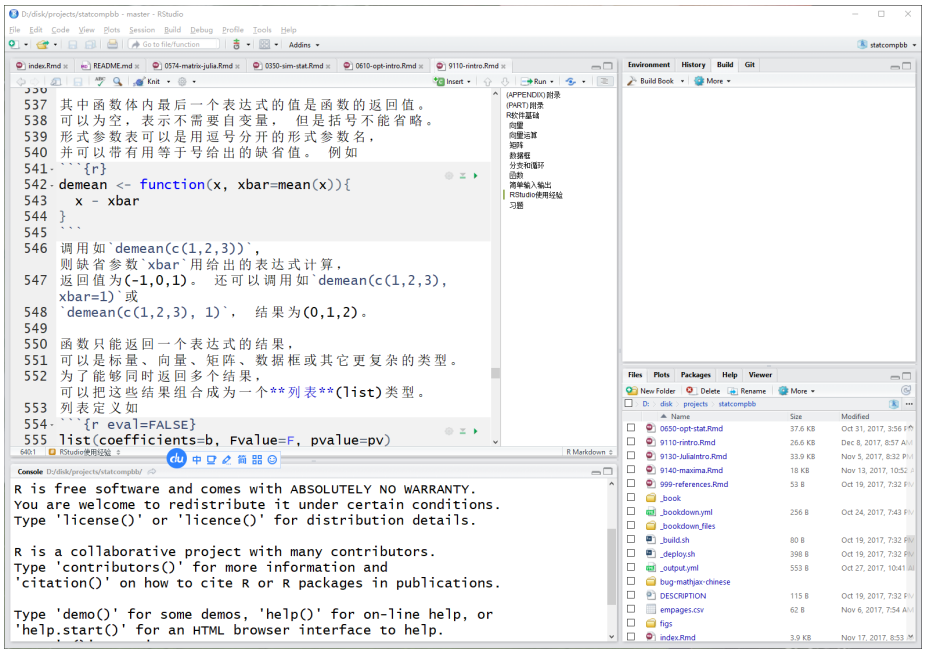


图 A.1: RStudio 界面

(提示：使用 R 的 `filter()` 函数)

习题 2

写一个 AR(1) 的模拟函数：

$$x_t = a + bx_{t-1} + \varepsilon_t, t = 1, 2, \ldots, n, \mathrm{Var}(\varepsilon_t) = \sigma^2$$

函数的参数为 n、a、b、σ 和 x_0, 缺省时 n=100, a=0, b=1, σ=1, $x_0 = 0$。(提示：使用 `filter` 函数)

习题 3

设随机变量 $X \sim \mathrm{B}(30, 0.5)$, 计算 $p = P(X > 20)$，估计计算精度。用正态近似方法计算 p 并比较两个结果。

附录 B

Julia 语言入门 (*)

B.1 Julia 的安装和运行

B.1.1 Julia 程序语言介绍

Julia 程序语言是一种计算机编程语言，就像 C、C++、Fortran、Java、R、Python、Matlab 等程序语言一样。Julia 语言历史比较短，发布于 2012 年，是 MIT 的几位作者和全世界的参与者共同制作的。主网站在https://julialang.org/。

Julia 与 R、Python 等一样是动态类型语言，程序与 R、Python 一样简单，但是它先进的设计使得 Julia 程序的效率基本达到和 C、Fortran 等强类型语言同样的高效率。尤其适用于数值计算，在现今的大数据应用中也是特别合适的语言，排在 Python、R 之后，已经获得广泛的关注，现在用户较少只是因为历史还太短。

B.1.2 Julia 软件安装和运行

从 Julia 网站下载安装 Julia 的命令行程序，称为 REPL 界面。

运行方式是在一个字符型窗口中，在提示行 `julia>` 后面键入命令，回车后在下面显示结果。

在 MS Windows 操作系统中，设存放 Julia 源程序和数据文件的目录为 `c:\work`, 将安装后显示在桌面上的 Julia 图标复制到目录中，从右键选 “属性”，将 “起始位置” 栏改为空白。这样读写数据和程序文件的默认位置就是图标所在的目录。

Julia 源程序用`.jl` 作为扩展名，为了运行当前目录中的 “myprog.jl” 文件，在命令行用命令

```
julia> include("myprog.jl")
```

(其中 `julia>` 是自动显示的提示)。

B.1.3 Julia 的其它运行方式

1. 安装 REPL 界面后，下载安装微软的软件集成开发界面Visual Studio Code，然后安装 Julia 语言支持模块。
2. 安装 Anaconda 软件，并安装 Julia 的 IJulia 模块，将 IJulia 所需要的 Python 和 Jupyter 可执行程序的路径指定为安装的 Anaconda 软件的位置。这可以同时获得 Python 语言和一些科学计算、数据整理、统计建模、机器学习等软件。
3. 在 REPL 界面，安装 Pluto 笔记本软件，方法如：

   ```
   using Pkg; Pkg.add("Pluto")
   ```

   安装好以后，只要在 REPL 中运行

   ```
   using Pluto; Pluto.run()
   ```

   就可以进入一个笔记本形式的软件开发环境。

B.1.4 附加软件包安装和调用

Julia 语言安装好以后，已经默认安装了许多标准的软件包（library），称为标准库，比如线性代数计算的 LinearAlgebra 标准库。其中的 Base 库是不需要

额外声明就可以使用的，其它的库则需要用 `using` 库名声明，然后才可以访问其中的功能。

为了安装某个额外的软件包（库），方法如：

```
using Pkg; Pkg.add(" 库名")
```

参见：

- http://www.math.pku.edu.cn/teachers/lidf/docs/Julia/JuliaTut1.html。

B.2 Julia 的基本数据和相应运算

B.2.1 整数与浮点数

Julia 程序中的整数值可以直接写成如 `123` 或者`-123` 这样。虽然整数有多种类型，一般程序中不必特别关心整数常量的具体类型。Julia 允许使用特别长的整数，这时其类型为 BigInt。

Julia 的浮点数可以写成带点的形式如 `123.0`, `1.23`, 也可以写成带有 10 的幂次如 `1.23e3`(表示 1.23×10^3), `1.23e-3`(表示 1.23×10^{-3})。这些写法都属于 Float64 类型的浮点数。Julia 还有其他类型的浮点数，但是科学计算中主要使用 Float64 类型，在别的语言中这称为双精度浮点数。

B.2.2 四则运算

表示加、减、乘、除、乘方的运算符分别为：

```
+  -   *  /  ^
```

浮点数的四则运算遵循传统的算数运算规则和优先级规定。如

```
1.3 + 2.5*2.0 - 3.6/1.2 + 1.2^2
```

```
4.74
```

表示

$$1.3 + 2.5 \times 2.0 - 3.6 \div 1.2 + 1.2^2$$

B.2.3 整数的四则运算

整数加法、减法、乘法结果仍为整数，这样因为整数的表示范围有限，有可能发生溢出。如

```
10 + 2*3 - 3*4
```

```
4
```

整数用“/”作的除法总是返回浮点数，即使结果是整数也是一样：

```
10/2
```

```
5.0
```

整数用 `a % b` 表示 a 整除 b 的余数，结果符号总是取 a 的符号。如

```
10 % 3
```

```
1
```

整数与浮点数的混合运算会将整数转换成浮点数再计算。

B.2.4 字符串

单个字符在两边用单撇号界定，如'A'，'囧'。字符都是用 Unicode 编码存储，具体使用 UTF-8 编码。

零到多个字符组成字符串，程序中的字符串在两边用双撇号界定，如 "A cat"，" 泰囧"。

对于占据多行的字符串，可以在两侧分别用三个双撇号界定。如

```
"""
这是第一行
这是第二行
三个双撇号界定的字符串中间的单个双撇号" 不需要转义
"""
```

"这是第一行\n这是第二行\n三个双撇号界定的字符串中间的单个双撇号\"不需要转义\n"

B.2.5 字符串连接

用星号 “*” 连接两个字符串，如

```
" 过去的" * " 历史" * " 不能忘记"
```

"过去的历史不能忘记"

B.3 变量

B.3.1 变量

变量名是一个标识符，用来指向某个值在计算机内存中的存储位置。变量名可以用英文大小写字母、下划线、数字、允许的 Unicode 字符。变量名不允许使用空格、句点以及其它标点符号和井号之类的特殊字符。

给变量赋值，即将变量名与一个内存中的内容联系起来，也称为绑定（binding），使用等号 “=”，等号左边写变量名，右边写要保存到变量中的值。如

```
x = 123
```

```
123
```

```
y = 1+3/2
```

```
2.5
```

```
addr100871 = " 北京市海淀区颐和园路 5 号"
```

```
"北京市海淀区颐和园路5号"
```

变量的类型是由它保存的（指向的内存中的）值的类型决定的，不需要说明变量类型（Julia 允许说明变量类型，但一般不需要）。

变量赋值后，就可以参与运算，如：

```
x = 123
y = 1+3/2
x + y*2
```

```
128.0
```

B.3.2 简单的输出

在 Julia 命令行，键入变量名或者计算表达式直接在下面显示结果。可以用 `println()` 函数显示指定的变量和结果。如

```
println(x + y*2)
```

```
128.0
```

```
println("x=", x, " y=", y, " x + y*2 =", x+y*2)
```

```
x=123 y=2.5 x + y*2 =128.0
```

B.4 向量

B.4.1 向量

在程序中直接定义一个向量，用方括号内写多个逗号分隔的数值，如

```
v1 = [1, 3, 4, 9, 13]
```

```
5-element Array{Int64,1}:
  1
  3
  4
  9
 13
```

```
v2 = [1.5, 3, 4, 9.12]
```

```
4-element Array{Float64,1}:
 1.5
 3.0
 4.0
 9.12
```

其中 v1 是整数型的向量，v2 是浮点型 Float64 的向量。

向量实际上是一维数组。

可以用 `1:5` 定义一个范围，在仅使用其中的元素值而不改写时作用与 `[1, 2, 3, 4, 5]` 类似。`1:2:9` 定义带有跨度的范围，与 `[1, 3, 5, 7, 9]` 类似。

```
1:5
```

```
1:5
```

```
1:2:9
```

```
1:2:9
```

B.4.2 向量下标

若 x 是向量，i 是正整数，x[i] 表示向量的第 i 个元素。如

```
v1[2]
```

```
3
```

对元素赋值将在原地修改元素的值，如

```
v1[2] = -999; v1
```

```
5-element Array{Int64,1}:
    1
 -999
    4
    9
   13
```

B.4.3 用范围作为下标

下标可以是一个范围，如

```
v1[2:4]
```

```
3-element Array{Int64,1}:
 -999
    4
    9
```

熟悉 Python 语言的读者要注意，这里下标范围的终点是包含在内的，而 Python 用这种方法取子集时不包括范围的终点。

用 `end` 表示最后一个下标，如

```
v1[4:end]
```

```
2-element Array{Int64,1}:
  9
 13
```

```
v1[1:(end-1)]
```

```
4-element Array{Int64,1}:
    1
 -999
    4
    9
```

B.4.4 数组作为下标

向量的下标也可以是一个下标数组，如

```
v1[[1, 3, 5]]
```

```
3-element Array{Int64,1}:
  1
  4
 13
```

取出的多个元素可以修改，可以赋值为同一个标量，如：

```
v1[1:3] = 0; v1
```

```
5-element Array{Int64,1}:
  0
  0
  0
  9
 13
```

也可以分别赋值，如

```
v1[[1, 3, 5]] = [101, 303, 505]; v1
```

```
5-element Array{Int64,1}:
 101
   0
 303
   9
 505
```

B.4.5 向量与标量的运算

向量与一个标量作四则运算，将运算符前面加句点 “.”：

```
.+   .-   .*   ./   .^
```

表示向量的每个元素分别与该标量作四则运算，结果仍是向量。如

```
v1 = [1, 3, 4, 9, 13]
v1 .+ 100
```

```
5-element Array{Int64,1}:
 101
 103
 104
 109
 113
```

```
100 .- v1
```

```
5-element Array{Int64,1}:
 99
 97
 96
 91
 87
```

```
v1 .* 2
```

```
5-element Array{Int64,1}:
  2
  6
  8
 18
 26
```

```
v1 ./ 10
```

```
5-element Array{Float64,1}:
 0.1
 0.3
 0.4
 0.9
 1.3
```

```
v1 .^ 2
```

```
5-element Array{Int64,1}:
   1
   9
  16
  81
 169
```

B.4.6 向量与向量的四则运算

两个等长的向量之间作加点的四则运算，表示对应元素作相应的运算。如

```
v1 = [1, 3, 4, 9, 13]
v3 = [2, 5, 6, 7, 10]
v1 .+ v3
```

```
5-element Array{Int64,1}:
  3
  8
 10
 16
 23
```

```
v1 .- v3
```

```
5-element Array{Int64,1}:
 -1
 -2
 -2
  2
  3
```

```
v1 .* v3
```

```
5-element Array{Int64,1}:
   2
  15
  24
  63
 130
```

```
v1 ./ v3
```

```
5-element Array{Float64,1}:
 0.5
 0.6
 0.666667
 1.28571
 1.3
```

B.4.7 向量初始化

用 `zeros(n)` 可以生成元素类型为 Float64、元素值为 0、长度为 n 的向量，如

```
zeros(5)
```

```
5-element Array{Float64,1}:
 0.0
 0.0
 0.0
 0.0
 0.0
```

用 `Vector{Float64}(undef, n)` 可以生成元素类型为 Float64 的长度为 n 的向量，元素值未初始化，如

```
Vector{Float64}(undef, 3)
```

```
3-element Array{Float64,1}:
 1.114676523e-315
 1.92399178e-315
 1.5759804e-315
```

类似可以生成其它元素类型的向量，如

```
Vector{Int}(undef, 3)
```

```
3-element Array{Int64,1}:
 378739344
 378739376
 155525920
```

用这样的办法为向量分配存储空间后可以随后再填入元素值。

B.4.8 向量复制

定义一个向量后，此向量的变量名就“绑定”在某个内存地址上。修改变量元素实际是修改了变量名所绑定的地址中某个位置的值。如

```
v = [2, 3, 5, 7, 11]
v[3] = 0
v
```

```
5-element Array{Int64,1}:
  2
  3
  0
  7
 11
```

将向量赋给另外一个向量，并不能实现复制；两个向量实际上是绑定到了相同的内存地址。如

```
w = v
w
```

```
5-element Array{Int64,1}:
  2
  3
  0
  7
 11
```

```
v[1] = 97
w
```

```
5-element Array{Int64,1}:
 97
  3
  0
  7
 11
```

可以看出修改了 `v` 的元素值，`w` 的元素值也被修改了，因为这两个变量名指向的是相同的内存地址。为了制作一个向量的副本，不能使用直接赋值的方法，而是使用 `copy()` 函数。如

```
v = [2, 3, 5, 7, 11]
w = copy(v)
v[3] = 0
w
```

```
5-element Array{Int64,1}:
  2
```

```
  3
  5
  7
 11
```

v 被修改后，其副本 w 未受影响。另外，给 w 另外赋值，相当于解除了 w 原来的绑定，并将 w 绑定到了将 v 的内容复制到另一内存地址的副本地址上。

B.4.9 向量的循环遍历

可以用 for 循环对向量的每个元素遍历访问。格式如

```
for i in eachindex(v1)
  println("v1[", i, "] = ", v1[i])
end
```

```
v1[1] = 1
v1[2] = 3
v1[3] = 4
v1[4] = 9
v1[5] = 13
```

这里 i 是循环产生的向量下标。

B.4.10 向量的输出

设 v2 = [1.5, 3, 4, 9.12], 为了将其按显示格式保存到文件 “tmp1.txt” 中，可用如下代码：

```
using DelimitedFiles
writedlm("tmp1.txt", v2, ' ')
```

结果文件中每个数占一行。

B.4.11 向量的输入

假设文件“vecstore.txt”中包含如下的内容：

```
1.2 -5  3.6
7.8 9.12 4.11
```

可以用如下代码将文件中的数据读入到一个向量 v4 中：

```
using DelimitedFiles
v4 = readdlm("vecstore.txt")[:]; v4
```

B.5 矩阵

B.5.1 矩阵

前面讲的向量是一维数组，不区分行向量还是列向量。

矩阵是二维数组，有两个下标：行下标和列下标。

在方括号内两个同行的元素之间用空格分隔，两行之间用分号分隔，可以在程序中输入矩阵，如

```
A1 = [1 2 3; 4 5 6]
```

```
2×3 Array{Int64,2}:
 1  2  3
 4  5  6
```

也可以直接用换行符分隔不同行，例如

```
A1 = [1 2 3
      4 5 6]
```

B.5.2 矩阵下标

设 A 是矩阵，则 A[i,j] 表示 A 的第 i 行第 j 列元素，如:

```
A1[2,3]
```

```
6
```

给元素赋值可以在矩阵中修改元素值，如：

```
A1[2,3] = -6; A1
```

```
2×3 Array{Int64,2}:
 1  2   3
 4  5  -6
```

B.5.3 矩阵列和行

设 A 是矩阵，则 A[:, j] 表示 A 的第 j 列元素组成的向量（一维数组），如

```
A1[:, 2]
```

```
2-element Array{Int64,1}:
 2
 5
```

A[i, :] 表示 A 的第 i 行元素组成的向量（一维数组），如

```
A1[2, :]
```

```
3-element Array{Int64,1}:
  4
  5
 -6
```

取出后的列或者行不分行向量和列向量。

B.5.4 子矩阵

如果 `A` 是矩阵，`I` 和 `J` 是范围或者向量，则 `A[I,J]` 表示 `A` 的行号在 `I` 中的行与列号在 `J` 中的列交叉所得的子矩阵，如

```
A1[1:2, 2:3]
```

```
2×2 Array{Int64,2}:
 2   3
 5  -6
```

用冒号 ":" 作为行下标或列下标表示取该维的全部下标。

B.5.5 子矩阵赋值

可以给子矩阵赋值，赋值为一个标量使得对应元素都改为该标量值；如

```
A1[1:2, 2:3] = 0; A1
```

```
2×3 Array{Int64,2}:
 1  0  0
 4  0  0
```

给子矩阵赋值为一个同样大小的子矩阵给对应元素赋值，如

```
A1[1:2, 2:3] = [102 103; 202 203]; A1
```

```
2×3 Array{Int64,2}:
 1  102  103
 4  202  203
```

B.5.6 矩阵初始化

用 `zeros(m, n)` 可以生成元素类型为 Float64、元素值为 0 的 $m \times n$ 矩阵，如

```
zeros(2, 3)
```

```
2×3 Array{Float64,2}:
 0.0  0.0  0.0
 0.0  0.0  0.0
```

用 `Array{Float64}(undef, m, n)` 可以生成元素类型为 Float64 的 $m \times n$ 矩阵，元素值未初始化，如

```
Array{Float64}(undef, 2, 3)
```

```
2×3 Array{Float64,2}:
 3.90752e-316  6.78756e-316  5.18128e-316
 6.61297e-316  6.81373e-316  3.87102e-316
```

类似可以生成其它元素类型的矩阵，如

```
Array{Int}(undef, 2, 3)
```

```
2×3 Array{Int64,2}:
 386423696  78323216  78323216
  79156592  78323216  78331944
```

用这样的办法先为矩阵分配存储空间，然后可以再填入元素值。

B.5.7 矩阵元素遍历

对行下标和列下标分别循环，行下标变化最快，如

```
A1 = [1 2 3; 4 5 6]
```

```
2×3 Array{Int64,2}:
 1  2  3
 4  5  6
```

```
for j = 1:size(A1,2), i = 1:size(A1,1)
  println("A1[", i, ", ", j, "] = ", A1[i, j])
end
```

```
A1[1, 1] = 1
A1[2, 1] = 4
A1[1, 2] = 2
A1[2, 2] = 5
A1[1, 3] = 3
A1[2, 3] = 6
```

另一种办法是用类似于向量遍历的方法，如：

```
for i in eachindex(A1)
  println("A1[", i, "] = ", A1[i])
end
```

```
A1[1] = 1
A1[2] = 4
A1[3] = 2
A1[4] = 5
A1[5] = 3
A1[6] = 6
```

B.5.8 矩阵读入

设当前目录中文件 “vecstore.txt” 中包含如下内容：

```
1.2 -5  3.6
7.8 9.12 4.11
```

将文件中的内容每一行看作矩阵的一行，文件中保存了一个 2×3 矩阵。读入方法如下:

```
using DelimitedFiles
Ain = readdlm("vecstore.txt"); Ain
```

```
2×3 Array{Float64,2}:
 1.2  -5.0   3.6
 7.8   9.12  4.11
```

B.5.9 保存矩阵到文件中

考虑上面的 Ain 矩阵，为了将其按文本文件格式保存到 “tmp2.txt” 中，用如下程序：

```
using DelimitedFiles
writedlm("tmp2.txt", Ain, ' ')
```

B.5.10 矩阵与标量的四则运算

矩阵与一个标量之间用加点的四则运算符号进行运算，与向量和标量之间的运算类似，表示矩阵的每个元素和该变量的四则运算，结果仍为矩阵。如

```
A1 = [1 2 3; 4 5 6]
```

```
2×3 Array{Int64,2}:
 1  2  3
 4  5  6
```

```
A1 .+ 100
```

```
2×3 Array{Int64,2}:
 101  102  103
 104  105  106
```

```
100 .- A1
```

```
2×3 Array{Int64,2}:
 99  98  97
 96  95  94
```

```
A1 .* 2
```

```
2×3 Array{Int64,2}:
 2   4   6
 8  10  12
```

```
A1 ./ 10
```

```
2×3 Array{Float64,2}:
 0.1  0.2  0.3
 0.4  0.5  0.6
```

```
A1 .^ 2
```

```
2×3 Array{Int64,2}:
  1   4   9
 16  25  36
```

B.5.11 两个矩阵之间的四则运算

两个同样大小的矩阵之间用加点的四则运算符号进行运算，表示两个矩阵的对应元素的运算。如

```
A2 = A1 .* 100
```

```
2×3 Array{Int64,2}:
 100  200  300
 400  500  600
```

```
A1 .+ A2
```

```
2×3 Array{Int64,2}:
 101  202  303
 404  505  606
```

```
A2 .- A1
```

```
2×3 Array{Int64,2}:
  99  198  297
 396  495  594
```

```
A1 .* A2
```

```
2×3 Array{Int64,2}:
  100   400   900
 1600  2500  3600
```

```
A2 ./ A1
```

```
2×3 Array{Float64,2}:
 100.0  100.0  100.0
 100.0  100.0  100.0
```

B.5.12 矩阵乘法

用 `A * B` 表示矩阵乘法。如

```
A3 = [11 12; 21 22]
```

```
2×2 Array{Int64,2}:
 11  12
 21  22
```

```
A3 * A1
```

```
2×3 Array{Int64,2}:
  59   82  105
 109  152  195
```

一个矩阵与一个向量（一维数组）作矩阵乘法，向量自动变成列向量，如:

```
A3 * [1, -1]
```

```
2-element Array{Int64,1}:
 -1
 -1
```

注意结果是向量（一维数组），而不是 2×1 矩阵（二维数组）。

行向量可以直接表示成方括号内多个数值之间用空格分隔的格式，如

```
[1,  -1] * [1  -1]
```

```
2×2 Array{Int64,2}:
  1  -1
 -1   1
```

又如

```
[1  -1] * A3
```

```
1×2 Array{Int64,2}:
 -10  -10
```

注意结果是 1×2 矩阵，即行向量，而不是向量。向量是一维数组，行向量是二维数组。

从以上例子可以看出在矩阵运算中向量可以看成是列向量，矩阵乘法结果如果是列向量，也会表示成向量（一维数组）。

B.5.13 矩阵转置

用 A.’ 表示矩阵 A 的转置；用 A’ 表示矩阵 A 的共轭转置。如

```
A1
```

```
2×3 Array{Int64,2}:
 1  2  3
 4  5  6
```

```
A1.'
```

```
3×2 Array{Int64,2}:
 1  4
 2  5
 3  6
```

两个向量 x 和 y 的内积用 dot(x, y) 表示，而不要写成 x.’ * y。如

```
dot([1, -1], [2, 3])
```

B.5.14 矩阵求逆和解线性方程组

inv(A) 表示 A^{-1}。如

```
A4 = [1 3; 3 1]
```

```
2×2 Array{Int64,2}:
 1  3
 3  1
```

```
inv(A4)
```

```
2×2 Array{Float64,2}:
 -0.125   0.375
  0.375  -0.125
```

用 `A \ B` 表示 $A^{-1}B$，当 `B` 是向量或者列向量时，就是求解线性方程组 $Ax = B$ 中的 x。如

```
A4 \ [-2, 2]
```

```
2-element Array{Float64,1}:
  1.0
 -1.0
```

B.6 自定义函数

B.6.1 介绍

Julia 语言支持自定义函数。在计算机程序语言中，函数是代码模块化、代码复用的基础。将常用的计算或者操作写成函数以后，每次要进行这样的计算或者操作，只要简单地调用函数即可。将复杂的任务分解成一个个函数，可以使得任务流程更加明晰，任务的不同阶段之间减少互相干扰。另外，用自定义函数表示的算法才能充分利用 Julia 的即时编译优势。

B.6.2 自定义函数的单行格式

类似于 $f(x) = x^2 + 3x + 1$ 这样的简单函数，可以用一行代码写成

```
f(x) = x^2 + 3*x + 1
```

```
f (generic function with 1 method)
```

调用如

```
f(2)
```

```
11
```

```
f(1.1)
```

```
5.510000000000001
```

B.6.3 自定义函数的多行格式

需要用多行才能完成的计算或者操作，就需要写成多行的形式。格式如

```
function funcname(x, y, z)
  ...
end
```

其中 funcname 是函数名称，x, y, z 等是自变量名，... 是函数内的语句或表达式，函数以最后一个表达式为返回值（结果）。

例如，写一个函数，以向量的形式输入一个变量的样本值，计算样本标准差：

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

自定义函数如

```
function mysd(x)
  n = length(x)
  mx = 0.0
  for z in x
    mx += z
  end
  mx /= n
  s = 0.0
  for z in x
    s += (z - mx)^2
  end
  sqrt(s / (n-1))
end
```

```
mysd (generic function with 1 method)
```

调用如

```
mysd([1, 2, 3, 4, 5])
```

```
1.5811388300841898
```

事实上，上面的函数定义可以用已有的一些函数进行简化。比如，`sum(x)` 对向量 x 的元素求和，于是以上的函数可以写成：

```
function mysd_simple(x)
  n = length(x)
  mx = sum(x)/n
  sqrt( sum(x .- mx) / (n-1) )
end
```

```
mysd_simple (generic function with 1 method)
```

第二个版本只是利用了向量化和已有函数，在 Julia 中其运行效率并不比第一个版本更好，甚至于不如第一个版本。Julia 语言与 R、Matlab 等语言不同，显式的循环一般有更高的执行效率，向量化写法仅仅是可以使得程序比较简洁。

均值、标准差这些基本统计函数已经在 Julia 的 Statistics 标准库中有定义。

B.7 程序控制结构

B.7.1 复合表达式

用 `begin ... end` 可以将多行的多个表达式组合起来当作一个表达式，复合表达式的值是其中最后一个表达式的值。如

```
z = begin
  x = 1
  y = 2
  x + y
end
z
```

```
3
```

多个表达式也可以用分号分隔后写在圆括号中，作为一个复合表达式，如

```
z = (x = 1; y = 2; x + y)
z
```

```
3
```

B.7.2 比较运算

两个数值之间用如下的比较运算符进行比较：

```
==   !=   <   <=   >   >=
```

分别表示等于、不等于、小于、小于等于、大于、大于等于。要特别注意“等于”比较用两个等号表示。

比较的结果是 true(真值) 或者 false(假值)。结果类型为 Bool 类型（布尔型）。

如

```
1 == 1.0
```

```
true
```

```
2 != 2.0001
```

```
true
```

```
3.5 > -1
```

```
true
```

```
3.5 > -1.5
```

```
true
```

```
-3.5 < 1.2
```

```
true
```

```
-3.5 <= 1.2
```

```
true
```

```
-3.5 > 1.2
```

```
false
```

```
-3.5 >= 1.2
```

```
false
```

两个字符串之间也可以比较，比较时按字典序比较，两个字符的次序按照其 Unicode 编码值比较。如

```
"abc" == "ABC"
```

```
false
```

```
"ab" < "abc"
```

```
true
```

```
" 陕西省" != " 山西省"
```

```
true
```

B.7.3 逻辑运算

比较通常有变量参与。如

```
age = 35; sex="F"
age < 18
```

```
false
```

```
sex == "F"
```

```
true
```

有时需要构造复合的条件，如“年龄不足 18 岁且性别为女”，“年龄在 18 岁以上或者性别为男”等。

用 `&&` 表示要求两个条件同时成立，用 `||` 表示只要两个条件之一成立则结果为真，用`!cond` 表示 `cond` 的反面。如

```
age < 18 && sex == "F"
```

```
false
```

```
age >= 18 || sex == "M"
```

```
true
```

B.7.4 短路与运算和分支

`&&` 是一种短路运算，表达式 `cond && expr` 仅当 `cond` 为 true 时才计算（运行）`expr`，所以这种写法经常用作程序分支的简写：条件 `cond` 为真时执行 `expr`，否则不执行。

比如，在计算 x 的平方根之前，先判断其非负：

```
x = -1.44
x < 0 && println(" 平方根计算：自变量定义域错误，x=", x)
```

```
平方根计算：自变量定义域错误，x=-1.44
```

B.7.5 短路或运算与分支

`||` 是一种短路或运算，表达式 `cond || expr` 仅当 cond 为 false 时才计算（运行）`expr`，所以这种写法经常作为程序分支的缩写：条件 cond 为假时才执行 `expr`，否则不执行。

比如，求平方根时当自变量不为负时才计算平方根：

```
x < 0 || (y = sqrt(x))
```

```
true
```

B.7.6 if–end 结构

可以用 `if cond ... end` 结构在条件 `cond` 成立时才执行某些语句，如

```
x = 1.44
if x >= 0
  y = sqrt(x)
  println("√", x, " = ", y)
end
```

```
√1.44 = 1.2
```

注意条件不需要用括号包围，结构以 end 语句结尾。

B.7.7 if–else–end 结构

if cond ... else ... end 结构当条件成立时执行第一个分支中的语句，当条件不成立时执行第二个分支中的语句。如

```
x = -1.44
if x >= 0
  y = sqrt(x)
  println("√", x, " = ", y)
else
  y = sqrt(-x)
  println("√", x, " = ", y, "i")
end
```

```
√-1.44 = 1.2i
```

B.7.8 if-elseif-else-end 结构

if cond1 ... elseif cond2 ... else ... end 可以有多个分支，有多个条件 cond1, cond2, ……，依次判断各个条件，那个条件成立就执行对应分支的语句，所有条件都不成立则执行 else 分支的语句。条件 cond2 隐含条件 cond1 不成立，cond3 隐含 cond1 和 cond2 都不成立，依此类推。例如

```
age = 35
if age < 18
  println(" 未成年")
elseif age < 60
  println(" 中青年")
elseif age < 100
  println(" 老年")
else
```

```
  println(" 老寿星！ ")
end
```

```
中青年
```

B.7.9 三元运算符

可以用 cond ? expr1 : expr2 表示比较简单的两分支选择，当 cond 成立时结果为 expr1 的结果，当 cond 不成立时结果为 expr2 的结果。如

```
x = -1.44
y = x >= 0 ? sqrt(x) : sqrt(-x)
```

```
1.2
```

B.7.10 for 循环

for 循环一般是沿着某个范围进行计算或处理，格式如下：

```
for loopvar = a:b
  expr1
  expr2
  ...
end
```

其中 loopvar 是自己命名的循环变量名，for 结构块内的语句（表达式）先对 loopvar=a 运行，再对 loopvar=a+1 运行, 最后对 loopvar=b 运行，然后结束。如

```
for i=1:3
  y = i^3
  println(i, "^3 = ", y)
end
```

```
1^3 = 1
2^3 = 8
3^3 = 27
```

范围也可以是 1:2:9, 0:0.1:1 这样的带有跨度（增量）的，可以是倒数的如 3:-1:1 表示 3, 2, 1。

B.7.11 对向量元素循环

用 in 关键字，可以使得循环变量遍历某个向量的元素，如：

```
x = [2, 3, 5, 7, 11]
for i in x
  y = i^3
  println(i, "^3 = ", y)
end
```

```
2^3 = 8
3^3 = 27
5^3 = 125
7^3 = 343
11^3 = 1331
```

B.7.12 向量元素按下标循环

设 x 是一个向量，上面的做法可以遍历 x 每个元素。有时还需要按照向量的下标遍历，这时使用 eachindex(x)，如

```
x = [2, 3, 5, 7, 11]
for i in eachindex(x)
  println("Prime No. ", i, " = ", x[i])
end
```

```
Prime No. 1 = 2
```

```
Prime No. 2 = 3
Prime No. 3 = 5
Prime No. 4 = 7
Prime No. 5 = 11
```

B.7.13 理解 (Comprehension)

对向量的循环，经常可以表达成一种称为 comprehension 的语法。例如，为了生成 1, 2, 3 的立方的向量，可以写成

```
xcube = [i^3 for i=1:3]
```

```
3-element Array{Int64,1}:
  1
  8
 27
```

对前 5 个素数作立方，可以写成

```
x = [2, 3, 5, 7, 11]
y = [z^3 for z in x]
```

```
5-element Array{Int64,1}:
    8
   27
  125
  343
 1331
```

B.7.14 两重 for 循环

for 循环可以嵌套，如

```
for i=1:9
  for j=1:i
    print(i, "×", j, " = ", i*j, "  ")
  end
  println()
end
```

```
1×1 = 1
2×1 = 2  2×2 = 4
3×1 = 3  3×2 = 6  3×3 = 9
4×1 = 4  4×2 = 8  4×3 = 12  4×4 = 16
5×1 = 5  5×2 = 10  5×3 = 15  5×4 = 20  5×5 = 25
6×1 = 6  6×2 = 12  6×3 = 18  6×4 = 24  6×5 = 30  6×6 = 36
7×1 = 7  7×2 = 14  7×3 = 21  7×4 = 28  7×5 = 35  7×6 = 42  7×7 = 49
8×1 = 8  8×2 = 16  8×3 = 24  8×4 = 32  8×5 = 40  8×6 = 48  8×7 = 56  8×8 = 64
9×1 = 9  9×2 = 18  9×3 = 27  9×4 = 36  9×5 = 45  9×6 = 54  9×7 = 63  9×8 = 72  9×9
```

这种两重循环可以简写为一个 for 语句，外层循环先写，内层循环后写，中间用逗号分隔。如

```
for i=1:9, j=1:i
  print(i, "×", j, " = ", i*j, "  ")
  j==i && println()
end
```

```
1×1 = 1
2×1 = 2  2×2 = 4
3×1 = 3  3×2 = 6  3×3 = 9
4×1 = 4  4×2 = 8  4×3 = 12  4×4 = 16
5×1 = 5  5×2 = 10  5×3 = 15  5×4 = 20  5×5 = 25
6×1 = 6  6×2 = 12  6×3 = 18  6×4 = 24  6×5 = 30  6×6 = 36
7×1 = 7  7×2 = 14  7×3 = 21  7×4 = 28  7×5 = 35  7×6 = 42  7×7 = 49
8×1 = 8  8×2 = 16  8×3 = 24  8×4 = 32  8×5 = 40  8×6 = 48  8×7 = 56  8×8 = 64
9×1 = 9  9×2 = 18  9×3 = 27  9×4 = 36  9×5 = 45  9×6 = 54  9×7 = 63  9×8 = 72  9×9
```

B.7.15 矩阵元素遍历

矩阵元素按照行列下标遍历，可以写成两重循环的形式。Julia 的矩阵是按列存储的，所以循环时先对第一列各个元素循环，再对第二列各个元素循环，……，按这样的次序遍历是比较有效的。如

```
A1 = [1 2 3; 4 5 6]
for j=1:3, i=1:2
  println("A1[", i, ", ", j, "] = ", A1[i,j])
end
```

```
A1[1, 1] = 1
A1[2, 1] = 4
A1[1, 2] = 2
A1[2, 2] = 5
A1[1, 3] = 3
A1[2, 3] = 6
```

for 结构的两重循环中写在前面的是外层循环, 循环变量变化较慢，写在后面的是内层循环，循环变量变化较快。上例中列下标 j 写在前面，行下标 i 写在后面，所以关于列的循环 j 是外层循环，关于行的循环 i 是内层循环，这样的矩阵元素遍历方式是按列次序遍历。

B.7.16 矩阵的 comprehension

在方括号内用两重的循环变量遍历可以定义矩阵。如

```
Ac = [i*100 + j for i=1:2, j=1:3]
```

```
2×3 Array{Int64,2}:
 101  102  103
 201  202  203
```

这里写在前面的循环变量 i 对应于行下标，写在后面的循环变量 j 对应于列下标。执行时行下标 i 在内层循环，列下标 j 在外层循环。

又如，如下程序返回矩阵对角化的结果：

```
[(i==j ? Ac[i,i] : 0) for i=1:2, j=1:3]
```

```
2×3 Array{Int64,2}:
 101    0  0
   0  202  0
```

B.7.17 while 循环

for 循环适用于对固定的元素或者下标的遍历，在预先未知具体循环次数时，需要使用当型循环循环或者直到型循环。

Julia 中用 `while cond ... end` 表示当型循环，在条件 cond 成立时执行结构内的语句，直到 cond 不成立时不再循环。

例如，要判断一个奇数 `x` 是否素数，从 3 开始逐个用奇数去除 `x`，直到除尽或者除数等于 `x` 为止：

```
x = 1333333
i = 3
while i < x && x % i != 0
    i += 2
end
if i == x
    println(x, "is prime.")
else
    println(x, "=", i, " X ", x ÷ i)
end
```

```
1333333=23 X 57971
```

B.7.18 直到型循环与 break 语句

当型循环每次进入循环之前判断循环条件是否成立，成立才进入循环。

直到型循环每次先进入循环，在循环末尾判断循环退出条件是否满足，满足退出条件时就不再循环。Julia 语言没有提供专门的直到型循环语法，可以用如下的方法制作直到型循环：

```
while true
  expr1
  expr2
  ...
  cond && break
end
```

其中 `cond` 是循环退出条件。`break` 语句表示退出一重循环。

例如，用泰勒展开近似计算自然对数 $\log(1+x)$：

$$\log(1+x) = x + \sum_{k=2}^{\infty} (-1)^{k-1} \frac{x^k}{k}$$

实际计算时不可能计算无穷次，所以指定一个精度如 `eps=0.0001`, 当计算的通项小于此精度时停止计算。程序用直到型型循环写成：

```
eps = 0.0001
x = 1.0
y = x; xk = x; sgn = 1; k = 1
while true
  k += 1; sgn *= -1; xk *= x
  item = xk / k
  y += sgn*item
  item < eps && break
end
println("eps = ", eps, " log(1+", x, ") = ", y,
  " Iterations: ", k)
```

```
eps = 0.0001 log(1+1.0) = 0.6931971730609582 Iterations: 10001
```

附录 C

Maxima 介绍 (*)

C.1 Maxima 初步认识

C.1.1 介绍

计算机代数软件可以进行多项式化简、分解因式、分式化简、通分、积分、微分等公式推导。这种运算又称为“符号计算”，与“数值计算”相对。著名的计算机代数系统有 Maple、Mathematica、Maxima。Maxima 是一个免费的计算机代数软件，可安装在各种不同的操作系统中。具有计算机代数系统必须的各种功能。用 Lisp 语言编制，是一个有悠久历史的软件，开发历史可追溯至 1968 年到 1982 年之间在 MIT 开发的 DOE Macsyma, 后来转由 Texas 大学 William F. Schelter 教授维护，在 1998 年转为 GNU GPL 版权协议。

在 MS Windows 系统中，可以下载安装 wxMaxima 软件 (http://http://andrejv.github.io/wxmaxima/index.html)。wxMaxima 运行界面是一个窗口形式，在窗口中逐行输入命令，命令以分号结尾，用 Shift+RETURN 键运行每行命令。命令用 $ 代替分号作为结尾则不显示命令的结果。

在命令行界面，可以用% 访问上一个公式的输出；用%i1, %o1 等访问历史的输入与输出。用`%th(k)` 访问向上数第 k 个结果。用冒号定义一个变量。

用`?` 命令查询一个命令，如

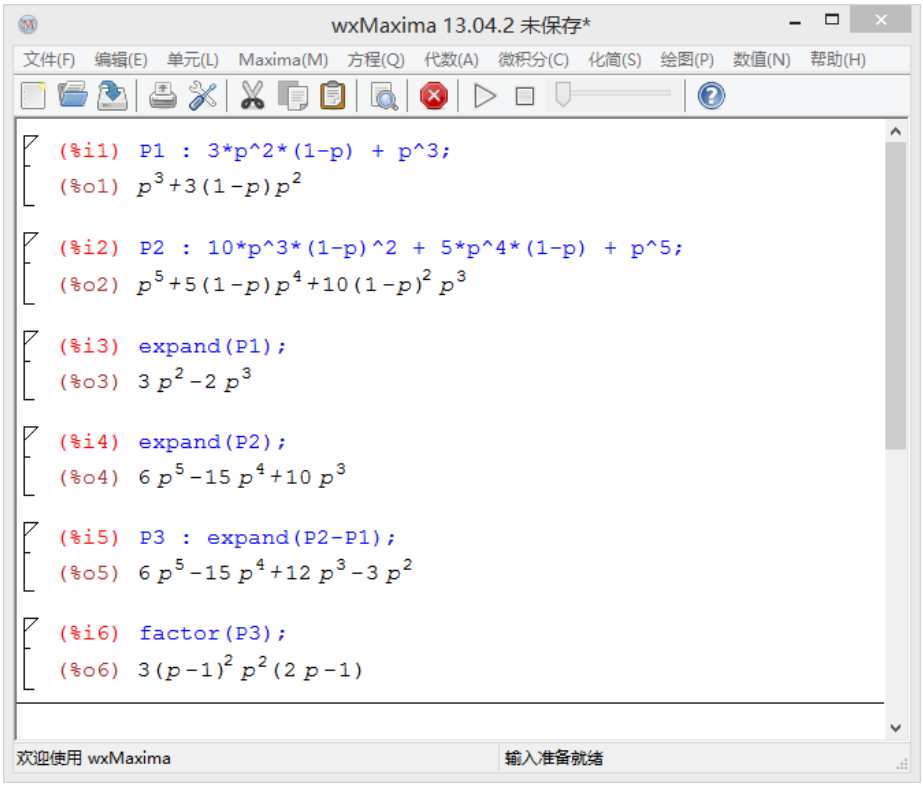


图 C.1: wsMaxima 运行界面示例

```
? diff
 0: diff  (Functions and Variables for Differentiation)
 1: diff <1>  (Functions and Variables for Differentiation)
 2: diff <2>  (Functions and Variables for itensor)
```

这时可以输入数字后按 Shift+RETURN 查询某个匹配的项。

C.1.2 四则运算

四则运算符号用 `+ - * /`, 乘方用 `^` 表示。Maxima 中不带小数点的四则计算结果保存为有理式，不作近似。如

```
12/5 + 6*(2^3 + 1/11);
```

结果为

$$\frac{2802}{55}$$

为了求以上结果的近似值，用 `numer` 函数开关，或 `float()` 函数，如

```
12/5 + 6*(2^3 + 1/11), numer;
```

或

```
float(%);
```

结果为 50.94545454545455。这样，Maxima 可以当作一个高级计算器使用。

C.1.3 数值类型

Maxima 的数值类型有整数、有理数、普通浮点数、高精度的浮点数。普通浮点数如 1.2, 1.2E-3。高精度浮点数如 1.2B0, 1.2B-3, 有效位数用变量 `fpprec` 控制。函数 `float(x)` 把 `x` 转为浮点型，函数 `bfloat(x)` 将 `x` 转为高精度浮点型。有高精度浮点数参加运算的算式结果为高精度浮点数；没有高精度浮点数但有浮点数算式结果为浮点数。

用`%e`、`%pi` 表示 e 和 π, 用`%i` 表示虚数单位。`inf`, `minf`, 分别表示 $+\infty$, $-\infty$, `infinity` 表示复数无穷大。如

```
cos(%pi/6)
```

结果为

$$\frac{\sqrt{3}}{2}$$

C.1.4 复数

用%i 表示虚数单位，如此可以输入复数，如

```
z1 : 5 + 3*%i;
```

用 realpart() 和 imagpart() 返回复数的实部和虚部。用 conjugate() 返回复数的复共轭。用 abs() 求复数模，用 carg() 求辐角。

rectform 把复数表为直角坐标形式，如

```
rectform(z1);
```

结果显示为

$$3\,i + 5$$

polarform 把复数表为直角坐标形式，如

```
polarform(z1);
```

显示为

$$\sqrt{34}\,e^{i\,\mathrm{atan}\left(\frac{3}{5}\right)}$$

C.1.5 列表 (list)

用方括号把逗号分隔的数值组合在一起作为一个列表，如

```
[2,3,4] + [7,8,9]
```

结果为

```
[9,11,13]
```

C.1.6 变量

用冒号定义变量，类似于其它语言的 =，如

```
a: 1;
b: 2;
```

定义变量 a 值为 1, b 值为 2。用 kill() 删除变量，如

```
kill(a, b)
```

values 列表保存了现有的用户自定义变量名。

C.2 多项式

C.2.1 多项式合并

考虑如下的一个推导问题。甲、乙比赛，每局比赛甲赢的概率 $p \in (0.5, 1)$。问：三局两胜还是五局三胜对甲有利？三局两胜中甲赢的概率为

$$\begin{aligned} &P(\text{三局两胜中甲赢}) = P(\text{甲赢 2 局}) + P(\text{甲赢 3 局}) \\ =&C_3^2 p^2(1-p) + p^3 = 3p^2 - 2p^3, \end{aligned}$$

五局三胜中甲赢的概率为

$$\begin{aligned} &P(\text{五局三胜中甲赢}) \\ =&P(\text{甲赢 3 局}) + P(\text{甲赢 4 局}) + P(\text{甲赢 5 局}) \\ =&C_5^3 p^3(1-p)^2 + C_5^4 p^4(1-p) + p^5 = 10p^3 - 15p^4 + 6p^5, \end{aligned}$$

在 wxMaxima 中用如下两个命令定义了这两个概率:

```
P1 : 3*p^2*(1-p) + p^3;
P2 : 10*p^3*(1-p)^2 + 5*p^4*(1-p) + p^5;
```

这里 P1 和 P2 是两个以 p 为变量的符号表达式，用来表示 p 的一元多项式。用 expand() 函数可以把多项式合并同类项化简:

```
expand(P1);
expand(P2);
```

如下命令计算两个多项式的差并合并同类项:

```
P3 : expand(P2-P1);
```

结果得到两个概率的差为

$$6p^5 - 15p^4 + 12p^3 - 3p^2$$

C.2.2 多项式因式分解

以上的两种赛制甲获胜概率之差简单地化为

$$3p^2(2p^3 - 5p^2 + 4p - 1),$$

为判断其是否在 $p \in (0.5, 1)$ 总为正值，可以分解因式。Maxima 用 `factor()` 函数分解因式，如:

```
factor(P3);
```

结果为

$$3\,(p-1)^2\,p^2\,(2\,p-1)\,.$$

在 $p \in (0.5, 1)$ 为正值。

C.2.3 推导结果导出

wxMaxima 的结果显示为数学公式形式，选中结果，可以选 “编辑—复制为 LaTeX” 菜单，把结果导出到 LaTeX 文件中。也可以把公式复制为纯文本、图片。

C.3 分式化简

考虑如下分式:

```
R1 : (x+1)^2/(x-1) + 1/(x+1);
```

结果为

$$\frac{(x+1)^2}{x-1}+\frac{1}{x+1}$$

`expand()` 函数可以分别展开分子分母中的多项式，但不能通分合并:

```
expand(R1);
```

结果为

$$\frac{1}{x+1}+\frac{x^2}{x-1}+\frac{2\,x}{x-1}+\frac{1}{x-1}$$

函数 `ratexpand()` 可以把分式的分子分母中多项式合并同类项，然后通分, 结果表示为分子每个不同幂次单独一个分式的加法：

```
ratexpand(R1);
```

结果为

$$\frac{x^3}{x^2-1}+\frac{3\,x^2}{x^2-1}+\frac{4\,x}{x^2-1}$$

函数 `ratsimp()` 把分式之和通分合并变成一个分式：

```
ratsimp(R1);
```

结果为

$$\frac{x^3+3\,x^2+4\,x}{x^2-1}$$

`factor()` 函数可以对分式的分子分母分别进行因式分解，如

```
factor(%);
```

结果为

$$\frac{x\,\left(x^2+3\,x+4\right)}{(x-1)\,(x+1)}$$

C.4 三角函数化简

Maxima 用如下函数进行三角函数化简:

- `trigexpand` 和差化积;
- `trigreduce` 积化和差;
- `trigsimp` 利用 $\sin^2 x + \cos^2 x = 1$ 之类化简;
- `trigrat` 把三角有理分式化为 sin 和 cos 的线性函数。

C.4.1 和差化积

考虑如下三角函数有理式:

```
T1 : sin(2*x)/cos(x) + cos(2*x);
```

结果显示为

$$\frac{\sin(2\,x)}{\cos(x)} + \cos(2\,x)$$

`trigexpand` 把 $\sin(2x)$ 展开为 $2\sin x\cos x$, 把 $\cos(2x)$ 展开为 $\cos^2 x - \sin^2 x$:

```
T2 : trigexpand(T1);
```

结果显示为

$$-\sin(x)^2 + 2\sin(x) + \cos(x)^2$$

上述结果中同时有 $\cos^2 x$ 和 $\sin^2 x$，用 `trigsimp` 化简

```
trigsimp(%);
```

结果为

$$2\sin(x) + 2\cos(x)^2 - 1$$

C.4.2 积化和差

如下程序中

```
T3 : trigreduce(T2);
```

$\sin^2 x$, $\cos^2 x$ 被化为一次式, 结果显示为

$$\frac{\cos(2\,x)+1}{2}+\frac{\cos(2\,x)}{2}+2\sin(x)-\frac{1}{2}$$

用 `expand` 简单地合并同类项:

```
expand(%);
```

结果为

$$\cos(2\,x)+2\sin(x)$$

用 `trigsimp()` 或 `trigrat()` 也可以合并同类项。

C.4.3 三角函数有理式化简

对 T1

$$\frac{\sin(2\,x)}{\cos(x)}+\cos(2\,x)$$

这样的三角函数有理分式，用 `ratsimp()` 可以直接化简为三角函数线性式, 或分子、分母都是三角函数线性项的有理式:

```
trigrat(T1);
```

结果为

$$\cos(2\,x)+2\sin(x)$$

```
ratsimp(T1);
```

结果为

$$\frac{\sin(2x)+\cos(x)\cos(2x)}{\cos(x)}$$

又如，

```
T4 : sin(x)^2 / cos(x) + cos(2*x);
```

为

$$\cos\left(2\,x\right)+\frac{\sin\left(x\right)^{2}}{\cos\left(x\right)}$$

用 `trigrat` 作用:

```
trigrat(%);
```

结果为

$$\frac{\cos\left(3\,x\right)-\cos\left(2\,x\right)+\cos\left(x\right)+1}{2\cos\left(x\right)}$$

又可以用 `trigexpand()` 和 `trigsimp()` 作用:

```
trigexpand(%);
```

结果为

$$\frac{-3\cos\left(x\right)\,\sin\left(x\right)^{2}+\sin\left(x\right)^{2}+\cos\left(x\right)^{3}-\cos\left(x\right)^{2}+\cos\left(x\right)+1}{2\cos\left(x\right)}$$

```
trigsimp(%);
```

结果为

$$\frac{2\cos\left(x\right)^{3}-\cos\left(x\right)^{2}-\cos\left(x\right)+1}{\cos\left(x\right)}$$

C.5 函数和微积分

C.5.1 常用初等函数

```
sqrt(), abs(), max(), min(), sign()
exp(), log()
sin(), cos(), tan(), cot(), sec(), csc()
asin(), acos(), atan(), acot(), asec(), acsc()
sinh(), cosh(), tanh(), coth(), sech(), csch()
asinh(), acosh(), atanh(), acoth(), asech(), acsch()
```

C.5.2 组合函数

用 $x!$ 计算阶乘，如

```
5!;
```

结果为 120。用 `binomial`(n, k) 计算 C_n^k, 如

```
binomial(5,2);
```

结果为 10。

C.5.3 自定义变量和自定义函数

用冒号定义变量，可以是数值，也可以是代数式，如:

```
g : 9.8;
P1 : 3*p^2*(1-p) + p^3;
```

用:= 定义函数，如

```
f(x,y) := 1/2*g*x^2*y;
```

结果显示为

$$\mathrm{f}(x,y) := \frac{1}{2}\, g\, x^2\, y$$

调用如

```
f(2,10);
```

结果为 196.0。

C.5.4 代换

带有未知数的式子可以用逗号格式把未知数代换为数值或其他代数式，如

```
P1 : 3*p^2*(1-p) + p^3;
P1, p=0.5;
P1, p=1;
```

后面两式的结果分别为 0.5 和 1。

用 $\frac{1}{2}+a$ 替换 p，如

```
expand(P1), p=1/2 + a;
```

结果为

$$-2\,a^3+\frac{3\,a}{2}+\frac{1}{2}$$

C.5.5 级数与连乘

用 `sum` 函数表示级数和，如

```
sum(i^2, i, 1, 10);
```

表示 $\sum_{i=1}^{10} i^2$, 结果为 385。而

```
sum(i^2, i, 1, n);
```

显示为

$$\sum_{i=1}^{10} i^2$$

用 `simpsum()` 进行级数化简，如

```
%, simpsum;
```

结果为

$$\frac{2\,n^3+3\,n^2+n}{6}$$

$\prod_{i=m}^{n} i^2$ 可以表示为 `product(i^2, i, m, n)`。用 `simpproduct` 化简连乘积，如

```
product(i, i, 1, n), simpproduct;
```

结果为

$$n!$$

C.5.6 微分

`diff(f(x), x, n)` 求 $\frac{d^n f(x)}{dx^n}$。如

```
diff(sin(x)*exp(2*x), x, 1);
```

结果为

$$2\,e^{2\,x}\sin\left(x\right)+e^{2\,x}\cos\left(x\right)$$

又

```
diff(sin(x)*exp(2*x), x, 2);
```

结果为

$$3\,e^{2\,x}\sin\left(x\right)+4\,e^{2\,x}\cos\left(x\right)$$

C.5.7 不定积分

`integrate(f(x), x)` 计算不定积分。如

```
integrate(x / (1+x)^2, x);
```

结果为

$$\log\left(x+1\right)+\frac{1}{x+1}$$

C.5.8 定积分

`integrate(f(x), x, a, b)` 计算定积分 $\int_a^b f(x)\,dx$。如

```
integrate(x*sin(x)*exp(-x), x, 0, %pi);
```

结果为

$$\frac{(\pi+1)\ e^{-\pi}}{2}+\frac{1}{2}$$

又如

```
integrate(exp(-1/2*x^2), x, minf, inf);
```

结果为 $\sqrt{2}\sqrt{\pi}$。

C.6 方程

C.6.1 一元方程

用 `solve(eqn, x)` 求解关于未知数 x 的一元方程，如

```
eq1 : a*x^2 + b*x + c = 0;
```

显示为

$$a\,x^2+b\,x+c=0$$

求解是符号运算:

```
solve(eq1, x);
```

结果为

$$[x=-\frac{\sqrt{b^2-4\,a\,c}+b}{2\,a},x=\frac{\sqrt{b^2-4\,a\,c}-b}{2\,a}]$$

以上一般解代入具体参数计算，如

```
%, a=1, b=2, c=1;
```

结果为

$$[x=-1,x=-1]$$

又如，以下的三次方程 (注意可以只给出方程左端)

```
solve(x^3 + 2*x^2 + 3*x + 4, x);
```

结果为

$$\left[x=-\frac{5\left(\frac{\sqrt{3}\,i}{2}-\frac{1}{2}\right)}{9\left(\frac{5\sqrt{2}}{3^{\frac{3}{2}}}-\frac{35}{27}\right)^{\frac{1}{3}}}+\left(\frac{5\sqrt{2}}{3^{\frac{3}{2}}}-\frac{35}{27}\right)^{\frac{1}{3}}\left(-\frac{\sqrt{3}\,i}{2}-\frac{1}{2}\right)-\frac{2}{3},\right.$$

$$x=\left(\frac{5\sqrt{2}}{3^{\frac{3}{2}}}-\frac{35}{27}\right)^{\frac{1}{3}}\left(\frac{\sqrt{3}\,i}{2}-\frac{1}{2}\right)-\frac{5\left(-\frac{\sqrt{3}\,i}{2}-\frac{1}{2}\right)}{9\left(\frac{5\sqrt{2}}{3^{\frac{3}{2}}}-\frac{35}{27}\right)^{\frac{1}{3}}}-\frac{2}{3},$$

$$\left.x=\left(\frac{5\sqrt{2}}{3^{\frac{3}{2}}}-\frac{35}{27}\right)^{\frac{1}{3}}-\frac{5}{9\left(\frac{5\sqrt{2}}{3^{\frac{3}{2}}}-\frac{35}{27}\right)^{\frac{1}{3}}}-\frac{2}{3}\right]$$

设函数

$$f(x)=ax^4+x+1$$

其中 $a>0$，求 $f(x)$ 的最小值点和最小值。在 Maxima 中计算：

```
f: a*x^4 + x + 1;
diff(%, x);
```

$$4*a*x^3+1$$

```
solve(%, [x]);
```

$$[x=-\frac{\sqrt{3}i-1}{24^{\frac{1}{3}}\,a^{\frac{1}{3}}},x=\frac{\sqrt{3}i+1}{24^{\frac{1}{3}}\,a^{\frac{1}{3}}},x=-\frac{1}{4^{\frac{1}{3}}\,a^{\frac{1}{3}}}]$$

求解导数等于零的方程得到三个根，但仅有第三个是实根：

```
rhs(%[3]);
```

$$-\frac{1}{4^{\frac{1}{3}}\,a^{\frac{1}{3}}}$$

这就是最小值点。最小值为：

```
f, x=%;
```

$$-\frac{1}{4^{\frac{1}{3}}\,a^{\frac{1}{3}}}+\frac{1}{4^{\frac{4}{3}}\,a^{\frac{1}{3}}}+1$$

即最小值为

$$1-\frac{3}{4^{4/3}a^{1/3}}$$

C.6.2 二元方程组

二元方程求解也使用 `solve`，其中方程和未知数都用方括号表示的列表表示。如

```
eq1 : y + 2*c = 0;
eq2 : 2*x*y - c*y = 5;
```

对应于

$$y+2\,c=0$$

$$2\,x\,y-c\,y=5$$

求解如

```
solve([eq1, eq2], [x, y]);
```

结果为

$$[[x = \frac{2\,c^2 - 5}{4\,c}, y = -2\,c]]$$

结果表示为解的列表，每个解又是包含 x, y 两部分的列表。

一个二元二次方程组的例子:

```
solve([ (x-1)^2 + y^2 = 4, x*y=1], [x, y]);
```

结果为

$$\begin{aligned}
&[[x = 0.51535959522949, y = 1.94039270687237],\\
&\ [x = 0.3167481152931\,i - 0.74342140598106,\\
&\quad y = -0.48506249405943\,i - 1.138462468793731],\\
&\ [x = -0.3167481152931\,i - 0.74342140598106,\\
&\quad y = 0.48506249405943\,i - 1.138462468793731],\\
&\ [x = 2.971483220309511, y = 0.3365322722219]]
\end{aligned}$$

一次方程组可以用 `linsolve()` 求解，如

```
linsolve ([3*x + 4*y = 7, 2*x + a*y = 13], [x, y]);
```

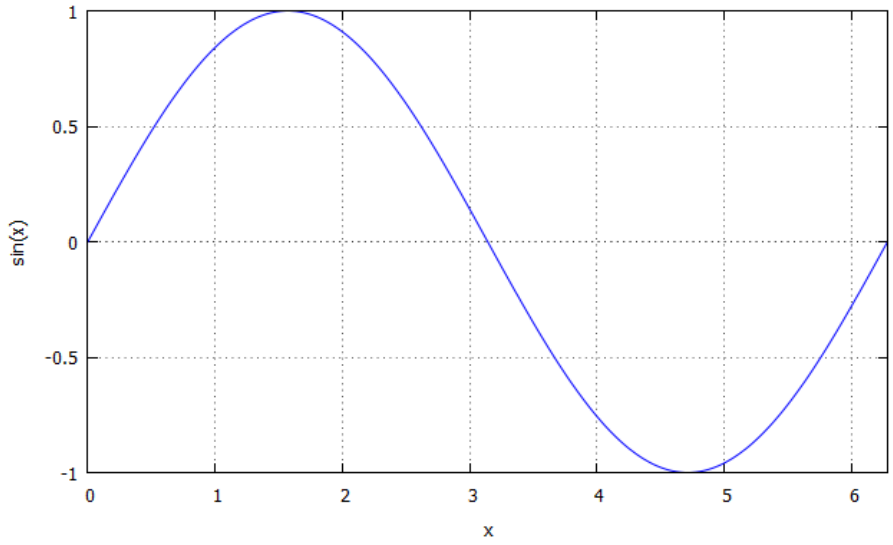
$$[x = \frac{7a - 52}{3a - 8}, y = \frac{25}{3a - 8}]$$

C.7 图形

C.7.1 曲线图

函数的曲线图使用 `plot2d()` 函数，用如

```
plot2d(sin(x), [x, 0, 2*%pi]);
```



可以指定纵坐标范围，如

```
plot2d(sin(x), [x, 0, 2*%pi],
  [y, -1.2, 1.2]);
```

C.7.2 图形文件

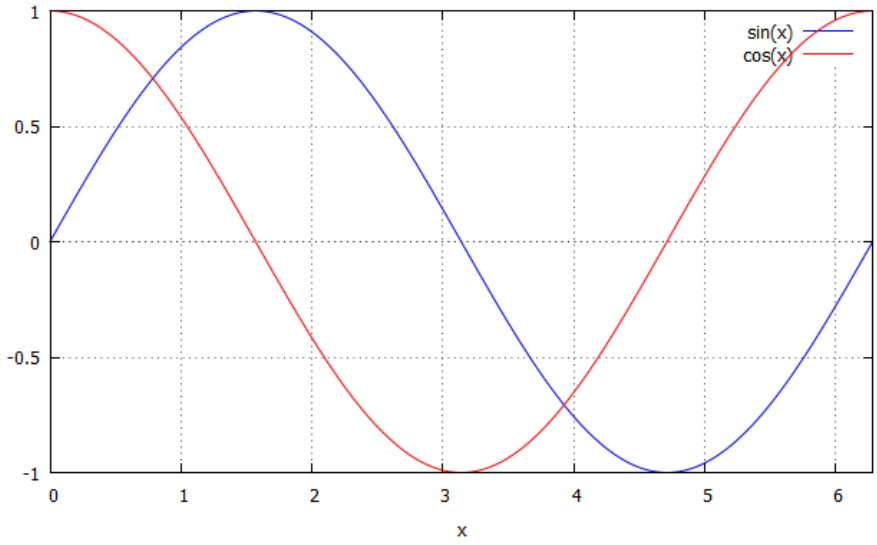
作图结果可以复制到 Windows 剪贴板，然后粘贴到 Word 中，或粘贴到画图程序中再保存为图形文件。也可以直接保存为图形文件，如

```
plot2d(sin(x), [x, 0, 2*%pi],
  [gnuplot_term, png],
  [gnuplot_out_file, "test-save.png"]);
```

C.7.3 多条曲线

多条曲线只要指定多个函数，如

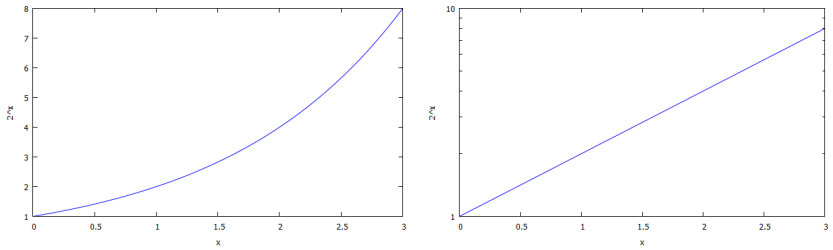
```
plot2d([sin(x), cos(x)], [x, 0, 2*%pi]);
```



C.7.4 对数坐标轴

`plot2d` 中加选项 `[logy]` 使得 y 轴为对数轴，加选项 `[logx, logy]` 使得 x, y 轴都为对数轴，如

```
plot2d(2^x, [x, 0, 3]);
plot2d(2^x, [x, 0, 3], [logy]);
```



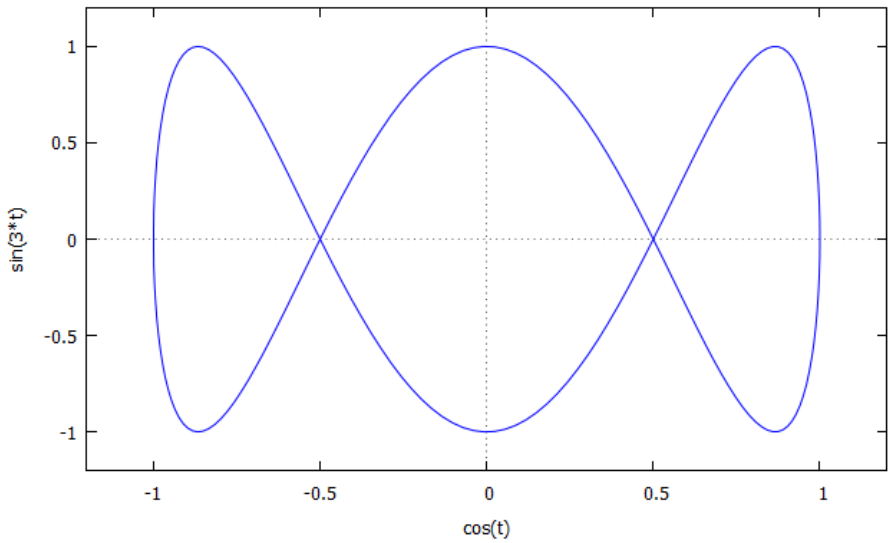
C.7.5 参数方程作图

在 `plot2d` 中把要绘图的函数用 `[parametrix, y(t), x(t), [t, tmin, tmax]]` 表示，可以进行参数方程作图。例如，如下参数方程表示的曲线

$$\begin{cases} x(t) = cos(t), \\ y(t) = sin(3t), \end{cases} \quad t \in [0, 2\pi]$$

可以作图如下:

```
plot2d([parametric, cos(t), sin(3*t),
  [t, 0, 2*%pi]],
  [x, -1.2, 1.2], [y, -1.2, 1.2],
  [nticks, 200]);
```



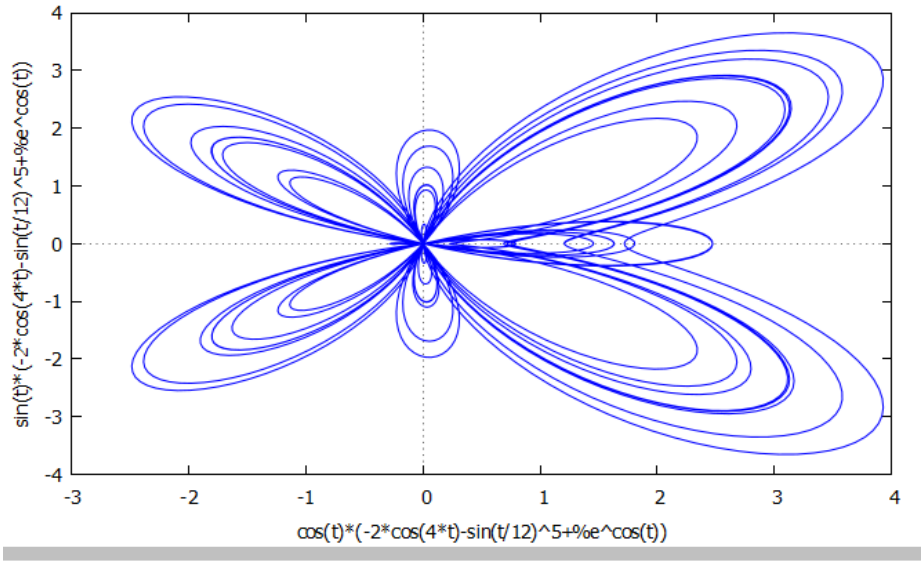
极坐标参数曲线可以化为直角坐标后参数作图，如

$$z(t) = \left[e^{\cos t} - 2\cos 4t - \sin^5 \frac{t}{12}\right] e^{it}, \ t \in [-8\pi, 8\pi],$$

程序如

```
r : (exp(cos(t)) - 2*cos(4*t) - sin(t/12)^5);
```

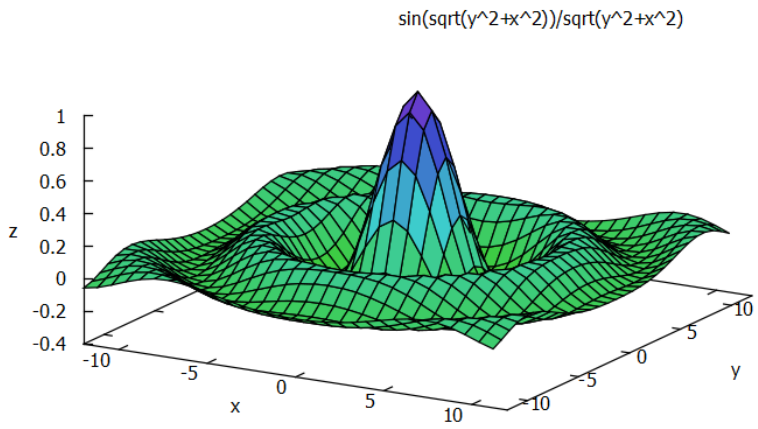
```
plot2d([parametric, r*sin(t), r*cos(t),
  [t, -8*%pi, 8*%pi]],
  [nticks, 2000]);
```



C.7.6 三维曲面图

三维图用 `plot3d()`。如

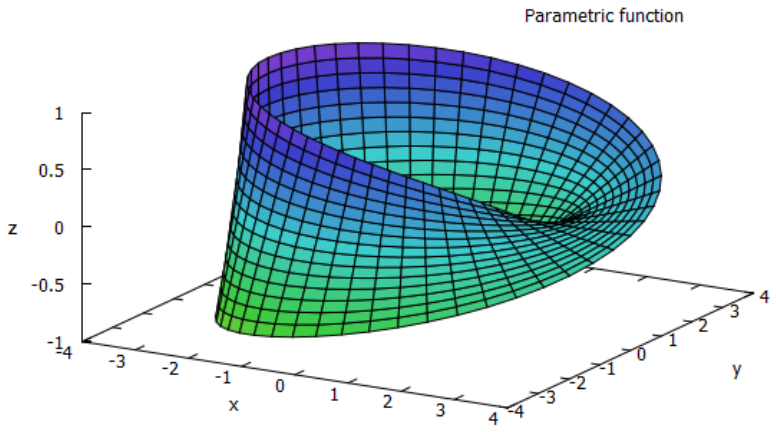
```
plot3d(sin(sqrt(x^2+y^2))/sqrt(x^2+y^2),
    [x, -12, 12], [y, -12, 12])$
```



三维参数曲面图}

三维图用 plot3d 和参数方程，有两个参数。如

```
plot3d([cos(x)*(3 + y*cos(x/2)),
        sin(x)*(3 + y*cos(x/2)),
        y*sin(x/2)],
        [x, -%pi, %pi], [y, -1, 1],
        ['grid, 50, 15])$
```


Parametric function
z
x
y
1
0.5
0
-0.5
-1
-4
-3
-2
-1
0
1
2
3
4

附录 D

理论证明补充 (*)

D.1 对立变量的补充证明

定理 D.1. 设 $f(x_1,\dots,x_n)$ 和 $g(x_1,\dots,x_n)$ 是关于每个自变量单调不减的函数，随机变量 $X_1,\dots,X_n$ 相互独立，记 $X=(X_1,\dots,X_n)$，则下面的不等式当其中期望存在有限时成立：

$$E[f(X)g(X)] \geq E[f(X)]\,E[g(X)]. \tag{D.1}$$

证明：用数学归纳法。当 $n=1$ 时，对任意实数 x,y 有

$$[f(x)-f(y)][g(x)-g(y)] \geq 0.$$

于是对任意随机变量 X,Y 有

$$[f(X)-f(Y)][g(X)-g(Y)] \geq 0.$$

于是

$$E\left\{[f(X)-f(Y)][g(X)-g(Y)]\right\} \geq 0.$$

当期望存在有限时有

$$E[f(X)g(X)]+E[f(Y)g(Y)] \geq E[f(X)g(Y)]+E[f(Y)g(X)].$$

设 X,Y 独立同分布，且涉及的期望存在有限，则

$$\begin{aligned}E[f(X)g(X)] =&E[f(Y)g(Y)],\\ E[f(X)g(Y)] =&E[f(Y)g(X)] = E[f(X)]E[g(X)],\end{aligned}$$

从而有

$$E[f(X)g(X)] \geq E[f(X)]E[g(X)].$$

设定理结论对 $n-1$ 个自变量的情形成立, 设 $f(x_1,\dots,x_n)$ 和 $g(x_1,\dots,x_n)$ 是关于每个自变量单调不减的函数，则

$$\begin{aligned}&E[f(X)g(X)|X_n = x]\\ =&E[f(X_1,\dots,X_{n-1},x)g(X_1,\dots,X_{n-1},x)|X_n = x]\\ =&E[f(X_1,\dots,X_{n-1},x)g(X_1,\dots,X_{n-1},x)] \quad (\text{由}X_n\text{与}X_1,\dots,X_n\text{的独立性})\\ \geq&E[f(X_1,\dots,X_{n-1},x)]E[g(X_1,\dots,X_{n-1},x)]\\ =&E[f(X_1,\dots,X_{n-1},x)|X_n = x]E[g(X_1,\dots,X_{n-1},x)|X_n = x]\end{aligned}$$

所以有

$$E[f(X)g(X)|X_n] \geq E[f(X)|X_n]E[g(X)|X_n].$$

两边取期望得

$$E[f(X)g(X)] \geq E\left\{E[f(X)|X_n]E[g(X)|X_n]\right\}.$$

注意到 $E[f(X)|X_n]$ 和 $E[g(X)|X_n]$ 分别是关于 X_n 的一元函数，且关于 X_n 是增函数，由 $n=1$ 时已证明的结论可知

$$\begin{aligned}&E\left\{E[f(X)|X_n]E[g(X)|X_n]\right\}\\ \geq&E\left\{E[f(X)|X_n]\right\}E\left\{E[g(X)|X_n]\right\}\\ =&E[f(X)]E[g(X)].\end{aligned}$$

证毕。

※※※※※

定理 D.2. 设 $h(x_1,\dots,x_n)$ 是关于每个自变量分别单调的函数，$U_1,\dots,U_n$ 独立同 $U(0,1)$ 分布，则当 $h(U_1,\dots,U_n)$ 二阶矩有限时

$$Cov[h(U_1,\dots,U_n),h(1-U_1,\dots,1-U_n)] \leq 0.$$

证明：因为自变量次序调整不影响结论，所以不妨设 h 关于前 r 个分量分别是单调增的，关于后 $n-r$ 个分量分别是单调减的。令

$$\begin{aligned} f(x_1,\dots,x_n) =& h(x_1,\dots,x_r,1-x_{r+1},\dots,1-x_n),\\ g(x_1,\dots,x_n) =& -h(1-x_1,\dots,1-x_r,x_{r+1},\dots,x_n), \end{aligned}$$

则 $f(x_1,\dots,x_n)$ 和 $g(x_1,\dots,x_n)$ 是关于每个自变量单调不减的函数，设 $V_1,\dots,V_n$ 为独立同 U(0,1) 分布随机变量，由定理D.1可知

$$\begin{aligned} &\mathrm{Cov}[f(V_1,\dots,V_n),g(V_1,\dots,V_n)] && \text{(D.2)}\\ =&-\mathrm{Cov}[h(V_1,\dots,V_r,1-V_{r+1},\dots,1-V_n), && \text{(D.3)}\\ &\qquad h(1-V_1,\dots,1-V_r,V_{r+1},\dots,V_n)] \geq 0, && \text{(D.4)} \end{aligned}$$

设 $U_1,\dots,U_n$ 独立同 U(0,1) 分布，令 $(V_1,\dots,V_n) = (U_1,\dots,U_r,1-U_{r+1},\dots,1-U_n)$，则 $V_1,\dots,V_n$ 独立同 U(0,1) 分布，从而由(D.4)式可知

$$\begin{aligned} &\mathrm{Cov}[h(U_1,\dots,U_r,U_{r+1},\dots,U_n,\\ &\qquad h(1-U_1,\dots,1-U_r,1-U_{r+1},\dots,1-U_n)] \leq 0. \end{aligned}$$

证毕。

※※※※※

参考文献

R. A. Becker and J. M. Chambers. *S: An Interactive Environment for Data Analysis and Graphics*. Wadsworth Advanced Books Program, Belmont CA, 1984.

Richard A. Becker, John M. Chambers, and Allan Reeve Wilks. *The New S Language*. Chapman and Hall, New York, 1988.

J. M. Chambers and T. Hastie. *Statistical Models in S*. Chapman and Hall, New York, 1992.

Mary Kathryn Cowles. *Applied Bayesian Statistics With R and OpenBUGS Examples*. Springer, 2013.

A. P. Dempster, N. M. Lairdk, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

Anderson. E. and ten others. Lapack users' guide, 1999. URL http://www.netlib.org/lapack/lug/lapack_lug.html.

K.-T. Fang and Y. Wang. *Number-Theoretic Methods in Statistics*. Chapman & Hall, 1994.

James E. Gentle. *Elements of Computational Statistics*. Springer Science + Business Media, Inc., 2002.

James E. Gentle. *Matrix Algebra: Theory, Computations, and Applications in Statistics*. Springer, 2007.

James E. Gentle. *Computational Statistics*. Springer, 2009.

Paul Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer, 2004.

R. J. Hyndman and Y. Fan. Sample quantiles in statistical packages. *American Statistician*, 50:361–365, 1996.

IMSL Inc. Imsl quality mathematical and statistical fortran subroutines for ibm personal computers. *IEEE Micro*, 5(2):97, 1985.

Tim A. Kochenderfer, Mykel J. and Wheeler. *Algorithms for Optimization*. MIT Press. URL https://algorithmsbook.com/optimization/.

Kenneth Lange. *Optimization*. Springer, 2013.

Christiane Lemieux. *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer, 2009.

Jun S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer, 2001.

D. Lunn, A. Thomas, N. Best, and D. Spiegelhalter. Winbugs — a bayesian modelling framework: concepts, structure, and extensibility. *Statistics and Computing*, 10:325–337, 2000.

D. Lunn, D. Spiegelhalter, A. Thomas, and N. Best. The bugs project: Evolution, critique and future directions (with discussion). *Statistics in Medicine*, 28:3049–3082, 2009.

P. L' Ecuyer. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 47:195–164, 1999. URL http://www.iro.umontreal.ca/lecuyer/myftp/papers/combmrg2.ps.

M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.

John F. Monahan. *Numerical Methods of Statistics*. Cambridge University Press, 2001.

J. Nelder and R. Mead. A simplex method for function minimization. *Comput. J.*, 1965.

SAS Institute Inc. *Base SAS 9.2 Procedures Guide: Statistical Procedures*. Cary, NC: SAS Institute Inc., third edition edition, 2010.

G. W. Stewart. *Introduction to Matrix Computations*. Academic Press, 1973.

B. A. Wichmann and I. D. Hill. Algorithm as 183: An efficient and portable pseudo-random number generator. *Applied Statistics*, 31:188–190. Remarks: 34, 198 and 35, 89, 1982.

关治 and 陆金甫. 数值分析基础. 高等教育出版社, 1998.

徐成贤, 陈志平, and 李乃成. 近代优化方法. 科学出版社, 2002.

茆诗松, 王静龙, and 濮晓龙. 高等数理统计. 高等教育出版社, 第二版 edition, 2006.

钱敏平, 龚光鲁, 陈大岳, and 章复熹. 应用随机过程. 高等教育出版社, 2011.

陈家鼎、孙山泽、李东风、刘力平. 数理统计学讲义. 高等教育出版社, 第 2 版 edition, 2006.

高惠璇. 统计计算. 北京大学出版社, 1995.

高立. 数值最优化方法. 北京大学出版社, 2014.