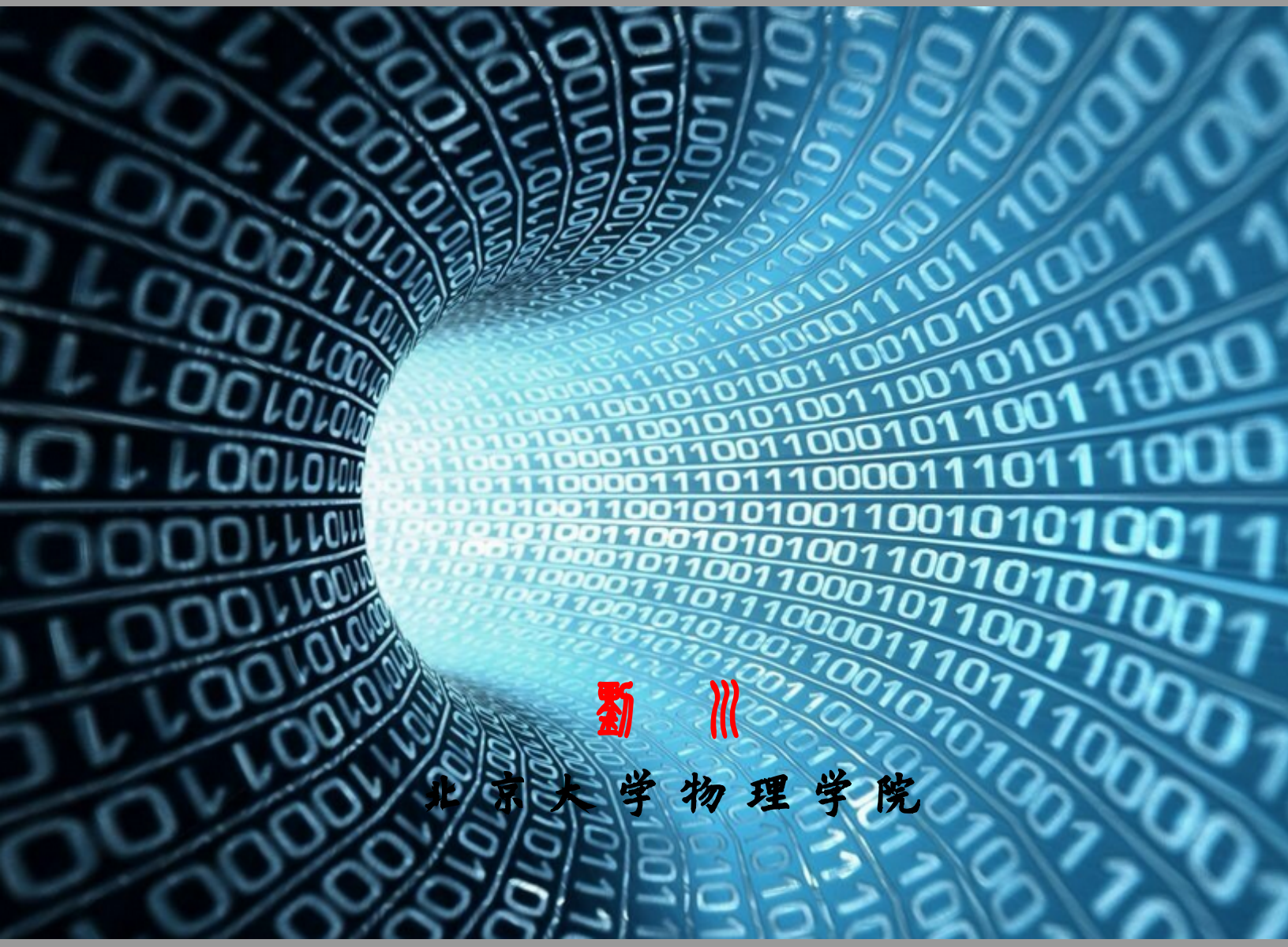
计 算 物 理 导 论



龚 昌

北 京 大 学 物 理 学 院

VERSION 0.98

开篇说明

2014 年深秋，我得知我将在下一个学期教授《计算物理》课程。这对我来说是个“不大不小”的挑战吧。说“不大”是因为“我老人家”毕竟也算是有经验的老教师了，况且计算物理所涉及的内容多数也是我平时研究中经常要遇到的，并不陌生。但是这个挑战也“不小”。主要是我以前从来没有讲授过这类以算法和应用为主的课程。一开始颇有些不知如何入手。

如果你在互联网上进行一番调研不难发现，国内外各个大学的计算物理（或者以其他名称出现但实际上与计算物理等价）课程是所有课程中差异性最大的一门课程。我们这个课程经过讨论后形成的共识是，计算物理 I 课程主要侧重于物理学各个领域都会遇到的、具有普遍性的数值问题。而一些更加具体的、仅仅涉及到某个物理学领域的问题则可以放在更加专门的计算物理 II 的课程中。问题确定了，但是如何解决这些问题也是个问题。计算物理当然是需要利用计算机进行求解的问题。这就涉及到人与机器的结合等多个问题：用什么机器，用什么语言或者用什么软件等等。这个问题实在是无法统一认识的问题。所以，我们最后这样来解决这个纠结，那就是在计算物理 I 中，我们更侧重于问题的基本解决的思路（也就是算法方面更侧重一些），至于说该问题的求解用什么语言则交给同学们自己选择。事实上，一些更加具体的、专门的问题可以留待计算物理 II 课程中由不同的教师来面对不同的学生进行个性化的解决。这个方针大致确定了以后，计算物理 I 课程的大致内容就可以框定了。

我们确定的内容主要包括：数值计算的基本知识；简单线性方程组的求解；数值的微分、积分；数值的内插、外推；非线性代数方程的求解与极值；本征值问题与矩阵对角化；随机数的产生与使用；数据的拟合及其统计描述；常微分方程的初值问题的求解；偏微分方程的边值问题的求解；快速傅里叶变换及其应用等等。

我觉得这些内容应当对大多数物理学院的同学来说足够完备了。当然，对于将来有志于潜心于某个具体的方向的研究生同学来说，也许针对性不够强，过于泛泛。但是这正是我们必须做出的取舍。就像前面说的，针对性更强的内容也许更适合于《计算物理 II》的课程而不是《计算物理 I》。

有些同学也许会说，这些内容目前可以用 MATLAB 很好地解决啊，我们为什么不直接开一个 MATLAB 课程呢？这个问题的答案本质在于，我们这个课程的主要目的并不是教会同学们如何使用某个具体的软件包，而是希望让大家了解数值计算的一些基本规则和方法。诚然，多数目前的商业软件包都提供了不错的用户界面，但是本课程的目的是让大家了解一下这些界面后面的一些东西。这样一来，在你以后面对完全不同的应用时，可以知道如何去求解一个全新的数值问题。也就是说，不仅仅会运用现成的软件提供的“黑匣子”，还能够大致了解这个“黑匣子”的工作原理。当然，这需要把握好一定的度。

我觉得我们这个课程的定位与 Numerical Recipes 基本一致，因此我强烈建议同学们以它为一本主要的参考书 [1]。这部书主要面对的读者是（美国一般大学的）高年级本科

生和低年级研究生。我感觉对世界一流大学的本科生应当可以基本无压力吧。当然，NR 的程序（如果你要使用的话）并不保证是最合适的。事实上，它的 Liscence 是相当“霸王条款”滴，尽管我天朝的读者是从来不看的。但是作为从讲述基本数值算法来说，这没有太多的影响。

如果你觉得这本书过于复杂了。你可以参考一本纯粹为美国本科生设计的关于数值计算的书 [6]。这是一部写得非常好的书。唯一的问题是，你们可能会觉得它过于简单了。事实上，为了能够让它能够面对更多的读者，作者有意规避了一些数值分析中遇到的数学概念。在获得巨大成功的同时，当然也有人不满意。比如，有些人希望能够在这个课程中引入合适的数学概念而不是回避。这方面的一个尝试是 Scott 的书。该书基本上保持在本科水平但注重了弥补 Burden & Faires 在数学概念介绍上的不足 [7]。

当然，同学们可以进一步参考更加倾向于研究生水平的教材，例如 [5]。属于同一系列（所谓的 Texts in Applied Mathematics，TAM 系列）的教材还有 [2]。如果具体到某个具体的题目，还可以参考更加专门的专著 [4]。由于时间仓促，这次只能将我讲授的部分准备出来。其他的部分将仅有一个条目 (以后可能还要变动)，等以后有机会再补上吧。所以这最初的版本号是 0.3。

在最后一段，一般总要对一些人表示致谢。如果读者觉得有些肉麻，请在您的眼睛瞟见我要致谢谁之前，赶快翻过这一页吧。我想我首先要感谢我在北大理论物理所的所有同仁，我还应当感谢选修这个课程的同学。特别是他们对于课程内容的反馈将成为我们不断改进本课程的重要依据。不管你们是真想学习这门课程还是被逼无奈地学习，你们辛苦了。在我的家人方面，我首先要感谢我的妻子韦丹，她对我的支持一直是我内心最大的感动。同时，也应该感谢我的儿子，他在我准备讲义的时候总是十分合作。

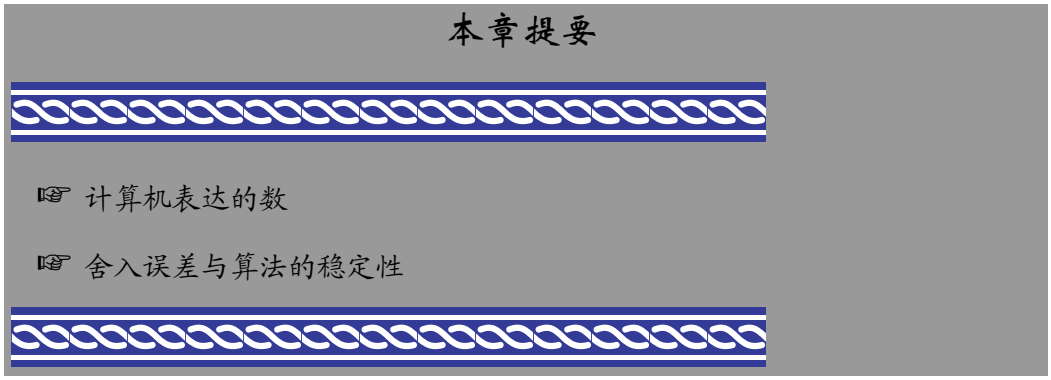
刘川，二零一四年初冬

¶2017 年秋，第一次完整地讲述计算物理课程。目前暂定版本号为 0.90。因为还有一些内容没有来得及整理到讲义之中，待随后输入后将版本号提升为 1.0 吧。

刘川，二零一七年秋

目 录

第一章　数值计算的基础



本章中我们将简要介绍数值计算的一些基础的概念。这包括计算机中可以表达的数、计算的舍入误差、算法的稳定性等。

1　计算机表达的数

¶ 由于计算机都只有有限的内存，因此它只能够表达有限多的数。这不仅仅包括实数，也包括整数。目前的经典计算机本质上都是通过记录二进制的数来记录所有的数的。也就是说，最基本的单元是所谓的位 (bit)，时髦的叫法是比特，它只有两个可能取值 0 和 1。每 8 个位构成一个字节 (byte)。

通常的的整数由 4 个 byte 构成，也就是说是 32 位。当然，如果嫌这个不够，也可以用 64 位 (也就是 8 个 byte)。例如在 C 语言之中，int 型的整数就是 4 个字节而所谓的 long 型的整数则是 8 个字节。[1] 对任何一个 4 字节 (32bit) 的非负整数都可以在二进制中表达为，

$$n = \sum_{i=0}^{31} b_i 2^i \,. \tag{1.1}$$

正整数的范围因此从 0 到 $2^{32}-1$。如果我们还希望表达负整数，我们只好牺牲一位来标记整数的正负，真正用于记录数字的位数减少到 31 位。整数的最大特点是，只要在其定

[1] 这里假设是标准的 64 位机器的编译器。

义域之内，机器对于整数的表达是严格的。而且整数之间的运算也是严格的 (只要结果不超出其定义域)。如果我们需要扩大整数的定义域，我们就需要启用更多位的整数。

对于实数，一般采用所谓的浮点数 (floating point number) 来表示。显然，由于实数构成不可数多的连续统，因此在计算机中能够表达的实数只能是其中的一个有限的部分。这就是浮点数系统。它保证了在任何一个实数的足够接近的邻域内总能找到一个浮点数来近似代表相应的实数。计算机中可以表达出来的那些数称为可表达的实数 (representable real numbers)，这个集合的大小取决于我们选择用多少位来表达一个实数。所有的那些不可表达的实数必须经过舍入 (rounding) 的操作，用它附近的一个可表达的实数来近似代替，相应产生的误差一般被称为舍入误差。特别地，我们可以定义一个在机器上能够表达的最小的正实数，它一般被称为机器精度 (machine precision)。它的定义可以取为：

$$\epsilon_M = \min\{g \in A | 1 \oplus g > 1\ , g > 0\}\ , \tag{1.2}$$

其中 A 表示计算机中所有可以表示的实数而 $\oplus$ 表示计算机中实现的“加法”(里面包含了对结果的舍入)。显然，机器精度的数值依赖于我们在一个实数上愿意投入多少内存。例如，对于通常的单精度的实数 (4 个字节) 的，$\epsilon_M \sim 10^{-7}$ 而对于双精度的实数 (8 个字节) 来说，$\epsilon_M \sim 10^{-16}$。

¶ 上面的描述是对于浮点数系统的一个比较粗略的概述。如果说得更仔细一些的话，我们选取一个正的自然数 $\beta \in \mathbb{N}$ 为底来表达的话 (通常 $\beta = 2$)，那么一般实数 x 在以 β 为底的系统中可以用下列整数部分和小数部分分别具有 $(n+1)$ 位和 m 位的小数表达，

$$x_\beta = (-)^s [x_n x_{n-1} \cdots x_1 x_0 . x_{-1} x_{-2} \cdots x_{-m}]\ , \quad x_n \neq 0\ , \tag{1.3}$$

其中 $s = 0, 1$ 称为符号位，用以标记该数是正还是负。这个表达称为实数 x(以 β 为底) 的位置表示 (positional representation)。实数的位置表示的另外一个等价写法是

$$x_\beta = (-)^s \left[\sum_{k=-m}^{n} x_k \beta^k\ . \right] \tag{1.4}$$

事实上，更为经济与方便的计数方法是将实数提出一个公共的因子，

$$x_\beta = (-1)^s \cdot (0.a_1 a_2 \cdots a_t) \cdot \beta^e = (-1)^s \cdot m \cdot \beta^{e-t}\ . \tag{1.5}$$

这个表示称为实数的浮点数表示 (floating point representation)，其中的 s 仍然是符号位，整数 $m = a_1 a_2 \cdots a_t$ 是一个 t 位的整数称为尾数 (mantissa)，显然 $0 \leq m \leq \beta^t - 1$；整数 e 则称为指数 (exponent)，它满足 $L \leq e \leq U$，一般选取 $L < 0$ 为一个负整数，$U > 0$ 为一个正整数。于是，对于一个占用 N 位的实数而言，计算机中存储时会让符号位占一位，有效数字 $a_i, i = 1, \cdots, t$ 占 t 位，指数 e 则占据剩下的 $N - t - 1$ 位。

常见的情形是单精度 (single precision) 的浮点数 (占据 32 位) 和双精度 (double precision) 的浮点数 (占据 64 位)，它们除了符号位占一位之外，分别具有 $t = 23$ 和 $t = 52$ 位来存储有效数字，而存储指数的位数则分别是 8 和 11 位。一般来说数字 0 会单独有一个特别的表示。因此，一个一般的浮点数系统 $\mathbb{F}$ 可以用它的基底 β、尾数的位数 t、指数 e 的上下限 L 和 U 标记为，

$$\mathbb{F}(\beta, t, L, U) = \{0\} \cup \left[x \in \mathbb{R} : x = (-)^s \beta^e \sum_{k=1}^{t} a_k \beta^{-k} \right] . \tag{1.6}$$

由于我们总是可以移动小数点并相应地改变指数 e 的数值，因此为了保证数字表示的唯一性，一般假设 $a_1 \neq 0$，否则我们可以向右移动小数点一位并减少 e 一个单位即可。类似的，一般假设 $m \geq \beta^{t-1}$。经过这样约定的浮点数表示称为正常化 (normalized) 的浮点数表达。例如，在没有正常化的以 10 为底的浮点数 $\mathbb{F}(10, 4, -1, 4)$ 系统中，实数 1 可以有下列四种表达，

$$0.1000 \cdot 10^1 \, , 0.0100 \cdot 10^2 \, , 0.0010 \cdot 10^3 \, , 0.0001 \cdot 10^4 \, ,$$

但是经过正常化后，只有第一个是符合要求的。容易证明，如果 $x \in \mathbb{F}$，那么我们必定有 $-x \in \mathbb{F}$。

容易发现，一个正常化表达的浮点数系统 $\mathbb{F}(\beta, t, L, U)$ 中的所有可表达的实数的绝对值一定介于 x_{min} 和 x_{max} 之间：

$$x_{\text{min}} \equiv \beta^{L-1} \leq |x| \leq \beta^U (1 - \beta^{-t}) \equiv x_{\text{max}} \, . \tag{1.7}$$

为了能够表达更多 (特别是绝对值更小) 的数，一般会将上述正常化后的浮点数系统与 $e = L$ 时的放弃要求 $a_1 \neq 0$ 的系统 (由于 $e = L$，因此这不会破坏数表示的唯一性) 合并。这可以产生在 $(-\beta^{L-1}, \beta^{L-1})$ 中的实数。这使得最小绝对值下降为 β^{L-t}。如果比这个还要小的数出现，机器会产生所谓的下溢 (underflow)。

浮点数的分布当然不是连续的。事实上也不是均匀的。假定有两个非零的浮点数 $x, y \in \mathbb{F}$，两者最近的距离一定介于 $\beta^{-1}\epsilon_M|x|$ 和 $\epsilon_M|x|$ 之间。其中的 $\epsilon_M = \beta^{1-t}$ 就是前面引入的机器精度 (machine epsilon)，它实际上是最小的满足 $1 + \epsilon_M > 1$ 的正实数。对于二进制来说，$\epsilon_M \sim 10^{-7}$，如果取 $t = 23$；$\epsilon_M \sim 10^{-16}$，如果取 $t = 52$。

如果我们考察相邻两个浮点数的相对误差，那么这是一个仅仅依赖于该数 mantissa 的函数，因此实际上是周期的。

$$\frac{\Delta x}{x} = \frac{(-)^s \beta^{e-t}}{(-)^s m(x) \beta^{e-t}} = \frac{1}{m(x)} \, , \tag{1.8}$$

其中 $m(x)$ 表示实数 x 的尾数部分 (一个整数)，由于 $\beta^{t-1} \leq m(x) < \beta^t - 1$，因此上述的相对误差会在 $[1/(\beta^t - 1), \beta^{1-t} = \epsilon_M]$ 之间震荡。

对于那些不在系统 $\mathbb{F}$ 中的实数怎么办呢？计算机必须能够处理它们。最常见的操作称为舍入 (rounding)。这个操作是从一个一般的实数 $x \in \mathbb{R}$ 到某个浮点数系统 $\mathbb{F}$ 的映射，

我们记为 $fl(\cdot)$。这个映射必须尽可能能够准确地反映原来的实数的大小，同时还不能改变实数的顺序。最简单的舍入方法就是利用大家熟悉的“四舍五入”原则。我们首先注意到任何的实数实际上都可以用公式 (1.5) 来表达出来，只要我们允许它的尾数可以有无穷多位。但是对一个给定的计算机浮点数系统，我们固定只能够保留 t 位的尾数。但是我们总可以考察某个实数的尾数的下一位，a_{t+1} 并且进行四舍五入的操作：如果它比 $\beta/2$ 要小，就直接舍掉；如果它大或等于 $\beta/2$，就将前一位加 1：

$$fl(x) = (-)^s(0.a_1a_2\cdots\tilde{a}_t)\cdot\beta^e\ , \quad \tilde{a}_t = \begin{cases} a_t & a_{t+1} < \beta/2 \\ a_t + 1 & a_{t+1} \geq \beta/2 \end{cases} \tag{1.9}$$

采用这样的处理之后，我们能够表达的实数范围大大扩大了。绝对值过大的实数仍然无法在计算机中表达。具体来说，如果 $|x| > x_{\text{max}}$，那么我们无法将其舍入到任何一个合理的可表达的实数。这时候计算机会产生一个溢出 (overflow)，程序的运行一般也会终止。类似的，如果某个实数的绝对值过小，$|x| < x_{\text{min}}$，这时候产生的一般称为下溢 (underflow)，此时往往系统会将其舍入为 0。

舍入造成的误差可以进行量化的估计。对于任意的 $x \in \mathbb{R}$，且满足 $x_{\text{min}} \leq |x| \leq x_{\text{max}}$，我们一定有，

$$fl(x) = x(1+\delta)\ , |\delta| \leq \frac{1}{2}\beta^{1-t} = \frac{1}{2}\epsilon_M\ . \tag{1.10}$$

因此我们有时候又称 $\epsilon_M/2$ 为舍入误差单位 (roundoff unit)。这意味着一个实数与它的舍入值之间的误差是完全在控制之中的：

$$E_{rel}(x) \equiv \frac{|x - fl(x)|}{|x|} \leq \frac{1}{2}\epsilon_M\ , E(x) \equiv |x - fl(x)| \leq \frac{1}{2}\beta^{e-t}\ . \tag{1.11}$$

从这个角度来说，浮点数的表示以及运算都是有误差的。浮点数比整数更为糟糕的是，即使是在其定义域内，浮点数也仅仅是某个实数的近似表示而不是严格表示，而且两个实数之间的运算的结果，即使它不超出定义域，通常也不是一个可表达的实数，需要将其经过舍入过程转换为一个可表达的实数。一般来说，由于舍入问题的存在，计算机中的基本运算比如，加减法、乘法等等，并不满足原先数学中的结合律、分配率等基本规律。例如，对于三个计算机中可表达的数 $a, b, c \in A$，在严格的数学中，$(a+b)+c = a+(b+c) = a+b+c$，但是在机器的运算中，$(a \oplus b) \oplus c$ 和 $a \oplus (b \oplus c)$ 一般是不相等的。但是，舍入方法的实现应当尽量确保在一次基本运算后造成的舍入误差仅仅在一两个机器精度的范围之内。即使这一点得到满足，由于计算机在复杂的算法中往往需要进行 N 多次的计算，因此我们必须考虑这种误差的传递和累计效应。这使得我们在数值计算中对算法的稳定性的要求提高了。

2 舍入误差与算法的稳定性

¶既然计算机中的实数都是利用不精确的浮点数来表达的，我们特别需要讨论算法的稳定性，或者说初始值中可能的误差，随着算法的运行是如何传递到最终的结果中去的。

我们首先来明确一下什么是算法。在计算数学中，从 n 个已知的数据 $x \in \mathbb{R}^n$ 出发，最终获得最终 m 个结果 $y \in \mathbb{R}^m$ 的一系列有限多次的操作之集合就构成了一个算法。为了获得一个具体问题的解可以有许多种算法。那么什么是一个好的算法呢。首先，一个好的算法首先必须是准确的 (当然，准确只是一个比较模糊的描述词语，具体的精度依赖于具体的问题)；也就是说，它可以给出问题的足够精确的近似解。其次，它必须是在计算资源方面比较快捷和有效的。当然正确性仍然是首要考虑。

既然每次计算机的操作往往都伴随着舍入误差，我们就需要分析这个误差是如何随着算法的发展而传递的。以一个简单的例子来说，假定我们需要计算三个实数 a, b, c 的和，即 $a+b+c$，我们有两个算法来进行这个计算，即 $(a+b)+c$ 和 $a+(b+c)$。这两种算法对应的误差可以分析如下。

对 $(a+b)+c$ 而言，我们有，

$$a \oplus b = (a+b)(1+\varepsilon_1)\ , (a \oplus b) \oplus c = [(a+b)(1+\varepsilon_1)+c](1+\varepsilon_2)\ ,$$

我们可以忽略掉二阶的无穷小量得到这个算法的结果为，

$$(a \oplus b) \oplus c = (a+b+c)\left[1+\frac{a+b}{a+b+c}\varepsilon_2+\varepsilon_2\right]\ . \tag{1.12}$$

显然，另外一种算法的误差估计就是将 a 与 c 互换，即，

$$(b \oplus c) \oplus a = (a+b+c)\left[1+\frac{c+b}{a+b+c}\varepsilon_2+\varepsilon_2\right]\ . \tag{1.13}$$

因此，两种算法有一个部分是相同的，即 ε_2 的部分；但是另外一个部分的放大因子分别为 $(a+b)/(a+b+c)$ 和 $(b+c)/(a+b+c)$。注意，这两个因子可以相差很远！事实上，两者之比为，

$$\left|\frac{a+b}{c+b}\right|$$

这意味着，如果某两个之和恰好特别小，比如两个数几乎正负相消 (比如 $a+b \simeq 0$)，那么先进行这两个数的加法的计算是更好的。事实上另外一种加法所产生的误差比首先进行几乎相消的两个数的加法要大很多倍。

相关的阅读

这一章我们简要介绍了数值计算的基础知识。

第二章　线性方程组的直接解法

本章提要

- ☞ 线性代数知识的回顾
- ☞ 高斯消元法
- ☞ *LU* 分解法
- ☞ Cholesky 分解法

本章我们将主要讨论求解线性方程组的数值方法中的比较“简单”的几类。线性方程组的求解在物理学的众多数值应用中都会遇到。因此，这是我们这个课程的一个基础。后面的许多章都会涉及。由于这是一个非常大的一个课题，本章中我们将仅仅涉及其中的一些基础的内容，主要包括：高斯消元法、*LU* 分解、Cholesky 分解等。更为复杂的内容 (比如奇异值分解等) 将放在后面的第 六章讨论。

3　线性代数知识的回顾

¶ 本节中我们需要回顾一下线性代数中最为基本的知识。这些知识对于我们理解本章以及后面各章中的基本算法是十分重要的。这些线性代数的知识大家在相关的数学课程中应当都多少接触过，但是由于我并不清楚有多少是大家接触过的，又有多少是没有接触过的，另外还可能虽然接触过，但由于时间久远已经忘记了的。总之，本节中我们将回顾一下这些知识。我们将仅仅回顾那些与我们数值计算密切相关的知识，这包括对矩阵的分类，以及一些重要的定理，尽管我们并不会去证明它们。另外，由于舍入误差的存在，我们往往需要对相关的算法的稳定性进行估计。这时候就需要对矢量以及矩阵的模进行讨论。这部分内容往往在通常的线性代数的课程中是没有的。我们在本节中也会介绍。由于我们后面 (参见第 六章) 还会讨论矩阵的对角化以及本征值的问题，因此有些线性代数的

知识 (特别是与本征值直接相关的知识) 将会在那里进行回顾。

首先是一些符号。一个数域 $\mathbb{K}$ 上面的 n 行 m 列的矩阵 A 我们一般记为：$A \in \mathbb{K}^{n\times m}$，其中数域 $\mathbb{K}$ 最为常见的情形是复数域 $\mathbb{C}$ 和实数域 $\mathbb{R}$。矩阵可以视为两个矢量空间 $\mathbb{K}^n$ 和 $\mathbb{K}^m$ 之间的一个线性映射。

3.1 矩阵的迹和行列式

当 $n = m$，我们称该矩阵为一个方阵。对于方阵我们可以定义它的迹 和 行列式。一个矩阵的迹就是该矩阵对角元之和：

这里要讨论一下方程的求解公式。

$$Tr(A) = \sum_{i=1}^{n} a_{ii} \,. \tag{2.1}$$

矩阵的行列式则在求解线性方程中具有重要的意义。它的原初定义为：

$$\det(A) = \sum_{\pi\in P} \mathrm{sign}(\pi) a_{1\pi_1} a_{2\pi_2} \cdots a_{n\pi_n} \,, \tag{2.2}$$

其中 π 表示 $(12\cdots n)$ 的一个排列而 $\mathrm{sign}(\pi)$ 则表示该排列的奇偶性，即经过奇数/偶数次对换可以恢复到原始排列。按照所谓的 Laplace 法则，矩阵的行列式还可以表达为，

$$\det(A) = \sum_{j=1}^{n} \Delta_{ij} a_{ij} \,, \tag{2.3}$$

其中 $i \in [1, n]$ 是任意一个行指标而 Δ_{ij} 是矩阵元 a_{ij} 的代数余子式。[1] 事实上，矩阵 A 的逆矩阵可以表达为

$$A^{-1} = \frac{1}{\det(A)} \Delta \,, \tag{2.4}$$

其中 Δ 是以 Δ_{ij} 为矩阵元的矩阵。由此我们看到，一个方阵的逆矩阵存在的充分必要条件是它的行列式不为零。

从上面的公式出发我们可以很容易获得求解线性方程 $Ax = b$ 的方法，

$$x_j = \Delta_j / \det(A) \,, \quad j = 1, \cdots, n \,, \tag{2.5}$$

其中 Δ_j 是将原矩阵中的第 j 列换为矢量 b 所得到的矩阵之行列式的值。这个法则一般称为 克莱默法则 (Cramer's rule)。

但是，上述行列式以、逆矩阵以及线性方程求解的公式并不能直接用于数值计算。因为按照这些公式，行列式的计算涉及到 $O(n!)$ 的计算量。这对于即使不太大的矩阵来说也过于庞大了。例如，即使对于大约 100 阶的矩阵来说 (或者说求解 100 个联立的线性方程组)，按照这些公式计算的话在我们有生之年都不太可能算出结果。为了获得数值上近似的解，我们需要更聪明的计算方法。

[1] 具体来说，$\Delta_{ij} = (-)^{i+j} \det(A_{ij})$，$A_{ij}$ 则是将原矩阵的第 i 行第 j 列消去后获得的矩阵。

3.2 矩阵的阶和核

矩阵 $A \in \mathbb{K}^{m\times n}$ 的秩 –记为 rank(A)–可以定义为从矩阵 A 中能够抽取的非奇异的子矩阵的最大的阶数。当 A 被视为 $\mathbb{K}^n \to \mathbb{K}^m$ 的线性映射时，我们可以定义其值域为：

$$\text{range}(A) = \{y \in \mathbb{K}^m : y = A \cdot x, x \in \mathbb{K}^n\} . \tag{2.6}$$

而矩阵的秩也可以定义为其值域空间的维数：rank$(A) = \dim(\text{range}(A))$。另一个重要的概念是矢量的线性相关。一个矩阵按照行的秩与其按照列的秩定义为线性无关的矢量的数目。严格来说，我们需要区分矩阵按照行的秩和按照列的秩。但是线性代数的基础知识告诉我们这两个是一致的。线性映射 A 的核定于为：

$$\ker(A) = \{x \in \mathbb{K}^n : A \cdot x = 0\} . \tag{2.7}$$

那么下列关系是成立的：

- rank$(A) = \text{rank}(A^T)$
- rank$(A) + \dim(\ker(A)) = n$

3.3 矢量与矩阵的模

¶ 我们讨论的矢量空间同时也是具有范数 (或者称为模) 的空间。 数学中可以对一般的模进行定义。一个矢量空间 V 上的模 $||\cdot||$ 一般来说可以定义为满足下列条件的非负函数：

1. 非负性：$||\mathbf{v}|| \geq 0, \forall \mathbf{v} \in V$ 且 $||\mathbf{v}|| = 0$ 当且仅当 $\mathbf{v} = 0$;
2. 均匀性：[2] $||\alpha\mathbf{v}|| = |\alpha| \cdot ||\mathbf{v}||; \forall \alpha \in \mathbb{K}, \forall \mathbf{v} \in V$;
3. 三角不等式：$||\mathbf{v} + \mathbf{w}|| \leq ||\mathbf{v}|| + ||\mathbf{w}||, \forall \mathbf{v}, \mathbf{w} \in V$.

一个常用的模是所谓的 p-模， 又称为 Hölder 模 (Hölder norm)，它由下式定义:

$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} , \forall \mathbf{x} \in V, 1 \leq p < \infty . \tag{2.8}$$

对于 p-模如果我们取极限 $p \to \infty$，就得到了无穷模，它实际上仅仅挑选出矢量 $\mathbf{x}$ 的分量中模最大的那个：

$$||\mathbf{x}||_\infty = \max_{1\leq i\leq n} |x_i| . \tag{2.9}$$

[2] 对于 $\mathbb{K} = \mathbb{R}$，$|\alpha|$ 表示其绝对值；对于 $\mathbb{K} = \mathbb{C}$，$|\alpha|$ 表示其模。

另外一个经常用到的是 $p=2$ 的情形。对于我们讨论的实空间和复空间来说，这个模称为相应空间的欧氏模 (Euclidean norm)：

$$||\mathbf{x}||_2 = (x,x)^{1/2} = (x^\dagger x)^{1/2} = \left(\sum_{i=1}^{n} |x_i|^2\right)^{1/2} . \tag{2.10}$$

既然矢量空间中模的定义可以有很多种，一个自然的问题是它们是否都等价呢？按照定义，矢量空间 V 上的两个模 $||\cdot||_p$ 和 $||\cdot||_q$ 被称为等价，如果存在两个正的常数 $c_{pg}>0$ 和 $C_{pg}>0$ 使得：

$$c_{pq}||\mathbf{x}||_q \le ||\mathbf{x}||_p \le C_{pq}||\mathbf{x}||_q \ , \forall \mathbf{x} \in V \ . \tag{2.11}$$

也就是说其中对于任意的矢量，其中一个模加在另外一个模的两个正常数倍数之间。相应的这些常数 c_{pq} 和 C_{pq} 被称为等价常数。可以证明上面给出的三种不同的 p-模都是等价的。等价的模的意义在于，当我们表述矢量空间中的极限的时候，例如 $\lim_{k\to\infty}\mathbf{x}^{(k)}=\mathbf{x}$，我们可以用相互等价的任何模函数来表征它。事实上，有限维矢量空间中的任何模都是等价的。

定义了矢量空间上的模之后就可以随之定义矩阵的模。对于 $\mathbb{K}^{m\times n}$ 上的矩阵，它的模 $||\cdot||$ 定义为：[3]

1. 非负性：$||A|| \ge 0, \forall A \in \mathbb{K}^{m\times n}$ 且 $||A||=0$ 当且仅当 $A=0$;
2. 均匀性：$||\alpha A|| = |\alpha|\cdot||A||; \forall \alpha \in \mathbb{K}, \forall A \in \mathbb{K}^{m\times n}$;
3. 三角不等式：$||A+B|| \le ||A|| + ||B||, \forall A, B \in \mathbb{K}^{m\times n}$.

如果矩阵的模和矢量的模满足

$$||A\mathbf{x}|| \le ||A||\cdot||\mathbf{x}|| \ , \forall \mathbf{x} \in \mathbb{K}^n, \ A \in \mathbb{K}^{m\times n} \ , \tag{2.12}$$

我们就称相应的矩阵模与矢量模兼容。另一方面，一个矩阵模 $||\cdot||$ 被称为服从乘法模 (sub-multiplicative norm) 如果它满足

$$||AB|| \le ||A||\cdot||B|| \ , \forall A \in \mathbb{K}^{n\times m}, \forall B \in \mathbb{K}^{m\times q}, \tag{2.13}$$

值得指出的是，并不是所有的矩阵模都是服从乘法的模。一个简单的例子是所谓的最大模，其定义为 $||A||_\Delta = \max(|a_{ij}|)$。读者可以验证它满足矩阵模的所有条件因而构成一个矩阵模。但是对于下面的矩阵：

$$A = B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} , \tag{2.14}$$

[3] 我们将使用同样的符号 $||\cdot||$ 来表示矩阵和矢量的模。这不会造成太大混淆因为它们的定义域是不同的。

我们可以很容易验证 $||AB||_\Delta = 2 > ||A||_\Delta ||B||_\Delta = 1$.

从一个矢量空间的模出发，我们可以定义一个矩阵模如下：

$$||A|| = \sup_{\mathbf{x}\neq 0} \frac{||A\mathbf{x}||}{||\mathbf{x}||} , \tag{2.15}$$

这称为诱导矩阵模或自然矩阵模。可以证明这个矩阵模是与诱导它的矢量模兼容的，同时也是服从乘法的。同时，对于由矢量的 p-模所诱导的矢量模，我们也会用同样的符号来标记，例如

$$||A||_p = \sup_{\mathbf{x}\neq 0} \frac{||A\mathbf{x}||_p}{||\mathbf{x}||_p} , \quad \forall \mathbf{x} \in V , \ \mathbf{x} \neq 0 . \tag{2.16}$$

3.4 一些特殊形状的矩阵

¶ 本小节我们罗列在数值计算中经常接触到的一些特殊的矩阵。

对角矩阵 (diagonal matrix) 是指仅仅对角元 a_{ii} 不为零的矩阵。通常意义下是指方阵，但是此定义也适用于长方阵。一个矩阵 $A \in \mathbb{K}^{m\times n}$，如果对 $i > j$ 就有 $a_{ij} = 0$，我们就称矩阵 A 为 上梯形矩阵。相应的，如果对 $i < j$ 就有 $a_{ij} = 0$，我们就称矩阵 A 为 下梯形矩阵。大家可以验证，如果 $m < n$ 的话，上梯形矩阵的非零矩阵元恰好构成一个梯形。对于 $m = n$ 的方阵而言，上/下梯形矩阵分别称为上/下三角矩阵。

上下三角矩阵的一些性质是容易验证的。它的行列式就是对角元的乘积。而且它的逆矩阵仍然维持原矩阵的上下三角的性质。如果上/下三角矩阵的对角元都等于 1，这样的上/下三角矩阵称为单位上/下三角矩阵。容易验证，两个单位上下三角矩阵的乘积仍然是单位上下三角矩阵。

三角矩阵的概念可以稍加推广到所谓的 带型矩阵 (banded matrices)。一般来说，对于 $A \in \mathbb{K}^{m\times n}$，我们称其具有 上带 p，如果对 $i > j + p$ 必定有 $a_{ij} = 0$；相应的，我们称其具有 下带 q 如果对于 $j > i + q$ 必定有 $a_{ij} = 0$。利用这个概念我们可以统一上面提及的几种矩阵。例如，对角矩阵是 $p = q = 0$ 的带型矩阵；下梯形矩阵是具有 $p = m - 1$,$q = 0$ 的带型矩阵；上梯形矩阵则是具有 $p = 0$,$q = n - 1$ 的带型矩阵。如果带型矩阵的 $p = q = 1$，则该带状矩阵称为 三对角矩阵 (tridiagonal matrix)。另外两种情形是 上双对角 ($p = 0, q = 1$) 和 下双对角 ($p = 1, q = 0$) 矩阵。另外一类我们后面会用到的矩阵是所谓的上/下 Hessenberg 矩阵。下 Hessenberg 矩阵具有 $p = m - 1, q = 1$ 而上 Hessenberg 矩阵则具有 $p = 1, q = n - 1$。

3.5 二次型与正定矩阵

另外一类重要的矩阵称为正定矩阵。它们与相应的二次型密切联系在一起。二次型起源于矢量空间中的标量积 (inner product) 运算。矢量空间 V 中的标量积可以视为 $V \times V$ 到 $\mathbb{K}$ 的一个映射 $(\cdot,\cdot)$，它满足：

1. 双线性：$(\alpha\mathbf{x}+\beta\mathbf{y},\mathbf{z}) = \alpha(\mathbf{x},\mathbf{z}) + \beta(\mathbf{y},\mathbf{z})$，$\forall \mathbf{x},\mathbf{y},\mathbf{z} \in V, \forall \alpha,\beta \in \mathbb{K}$;
2. 厄米性：$(\mathbf{x},\mathbf{y}) = (\mathbf{y},\mathbf{x})^*$,$\forall \mathbf{x},\mathbf{y} \in V$;
3. 正定性：$(\mathbf{x},\mathbf{x}) > 0$,$\forall \mathbf{x} \in V$ 除非 $\mathbf{x} = 0$.

对于空间 $\mathbb{C}^n$ 来说，我们可以内积为：[4]

$$(\mathbf{x},\mathbf{y}) = \mathbf{y}^\dagger \cdot \mathbf{x} = \sum_{i=1}^{n} y_i^* x_i \ . \tag{2.17}$$

这个标量积自然地生成前面提到的欧氏模 (也就是 $p=2$ 的 p-模)，

$$||\mathbf{x}||_2^2 \equiv (\mathbf{x},\mathbf{x}) = \sum_{i=1}^{n} |x_i|^2 \ . \tag{2.18}$$

对于上述的标量积，我们显然有

$$(A\mathbf{x},\mathbf{y}) = (\mathbf{x}, A^\dagger\mathbf{y}) \ , \tag{2.19}$$

其中 $A \in \mathbb{C}^{n\times n}$，$\mathbf{x},\mathbf{y} \in \mathbb{C}^n$。

如果对于 $\mathbb{C}^{n\times n}$(或 $\mathbb{R}^{n\times n}$) 中的矩阵 A 以及任意的非零矢量 $\mathbf{x} \in V$ 都有 $(A\mathbf{x},\mathbf{x})$ 是正的实数，我们就称矩阵 A 是正定的。如果 $>$ 换为 $\geq$ 且等号有可能成立，我们就称 A 是半正定的。一个实正定矩阵并不一定是对称的，但是只要它的对称部分是正定的就可以了。对于 $\mathbb{C}^{n\times n}$ 中的复矩阵 A，它是正定的条件要求 A 必定是厄米的 (从而其本证值均为实数) 并且具有正的本证值。这个结论的一个推论是，正定的复矩阵必定是不奇异的。

最后，让我们提及关于复厄米正定矩阵的下列性质。令 $A \in \mathbb{C}^{n\times n}$ 为厄米矩阵。那么它是正定矩阵当且仅当下列等价的条件之一获得满足：

1. $(A\mathbf{x},\mathbf{x}) > 0, \forall \mathbf{x} \neq 0, \mathbf{x} \in \mathbb{C}^n$;
2. A 的主子矩阵的本证值都是正的；
3. A 的主子矩阵的行列式都是正的 (又称 Sylvester 判据)；

[4]注意这个次序与我们物理学家在量子力学中习惯的 Dirac 符号的顺序恰好相反。在量子力学中我们通常定义 $(\mathbf{x},\mathbf{y}) = \mathbf{x}^\dagger \cdot \mathbf{y} = \langle \mathbf{x}|\mathbf{x}\rangle$。

4. 存在一个非奇异矩阵 $H \in \mathbb{R}^{n\times n}$ 使得 $A = H^\dagger H$.

正是这最后一个条件使得我们对于正定的厄米矩阵可以采用所谓的 Cholesky 分解，参见第 6节。事实上，非奇异的 H 不仅仅是存在的，我们还可以将其选为上三角矩阵 (从而 $H^\dagger$ 为下三角矩阵)。

4 高斯消元法

¶我们希望求解的线性系统可以统一表达为，

$$A \cdot x = b \ , \tag{2.20}$$

其中 $A \in \mathbb{C}^{n\times n}$ 是个一般的 $n \times n$ 的复矩阵，$b \in \mathbb{C}^n$ 是 $\mathbb{C}^n$ 中已知的复矢量，而 $x \in \mathbb{C}^n$ 则是待求的矢量。我们一般将假设 A 是不奇异的，否则方程一般无解。

数值上求解这类方程其实与我们在初中时学习的步骤基本一致。这实际上是一个古老的方法，称为高斯消元法 (Gauss Elimination Method, GEM)。我们首先写出这些方程，然后对它们进行一系列的所谓的初等变换 –即将某个方程乘以一定的数然后将其与另一个方程相加减 –其目的是消去某个变量前面的系数。通过这样的变换，将原先的方程化为如下形式的方程：

$$U \cdot x = c \ , U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & & u_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & u_{nn} \end{bmatrix} , \tag{2.21}$$

其中 U 是一个上三角矩阵。如果矩阵 A 是非奇异的，那么上三角矩阵 U 一定也是如此，这意味着所有的对角元 $u_{ii} \neq 0$。因此这个上三角系统可以运用所谓 反代 (back-substitution) 的方法来求解，即首先从最后一个方程解出 x_n，反代到倒数第二个方程解出 x_{n-1}；然后将这两者反代到倒数第三个方程解出 x_{n-2}，等等。用公式来表达就是，

$$x_i = \frac{c_i - \sum_{k=i+1}^{n} u_{ik} x_k}{u_{ii}} \ , \quad i = n, n-1, \cdots, 1 \ . \tag{2.22}$$

为了要将线性系统 (2.20) 化为上三角的形式 (2.21)，我们将原先的矩阵 A 和 b 合并为一个 $n \times (n+1)$ 的矩阵，它称为原来线性系统 (2.20) 的 增广矩阵 (augmented matrix)，我们将用 (A, b) 来标记它，

$$(A, b) = \begin{bmatrix} a_{11} & \cdots & a_{1n} & b_1 \\ \vdots & & \vdots & \vdots \\ a_{n1} & \cdots & a_{nn} & b_n \end{bmatrix} , \tag{2.23}$$

我们随后的线性变换都是直接对增广矩阵 (A,b) 来进行操作的。最终的目的是将其变换为上三角的形式 (2.21)。

我们首先去寻找一个 $a_{r1}\neq 0$。对所有的 $1\leq r\leq n$ 来说，这样的矩阵元总是存在的，否则矩阵 A 将是奇异的，与我们的初始假设矛盾。如果我们找到了这样的一个矩阵元，我们首先可以做的是将第 1 行与第 r 行进行互换。这可以通过一个交换矩阵来实现。假设通过交换行 1 和行 r 将 (A,b) 变为 $(\bar{A},\bar{b})$，那么这个变换可以表达为，

$$(\bar{A},\bar{b})=P^{(1r)}\cdot(A,b)\ , \tag{2.24}$$

其中 $n\times n$ 的置换矩阵 (permutation matrix)$P^{(1r)}$ 的矩阵元为，

$$\begin{cases}[P^{(1r)}]_{1r}=[P^{(1r)}]_{r1}=1\ ,[P^{(1r)}]_{11}=[P^{(1r)}]_{rr}=0\ ,\\ [P^{(1r)}]_{ij}=\delta_{ij}\ ,(ij)\neq(1r)\neq(r1)\ .\end{cases} \tag{2.25}$$

也就是说，除了第 1 和第 r 行、第 1 和第 r 列这四个矩阵元之外，其他的矩阵元与单位矩阵的相应矩阵元完全一样；而在这四个矩阵元的位置，它看起来就像 $\left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]$ 一样。大家很容易验证，这个矩阵是一个非奇异的矩阵并且它的逆矩阵就是它本身。当我们用它左乘上矩阵 (A,b) 时，其作用是交换该矩阵的第 1 行和第 r 行。经过这个变换，(A,b) 变换到了 $(\bar{A},\bar{b})$，用数学表达式来写就是：$(\bar{A},\bar{b})=P^{(1r)}\cdot(A,b)$。

另外一类初等变换的矩阵的相貌这样的，

$$G^{(1)}=\begin{bmatrix}1&0&\cdots&0\\-l_{21}&1&\cdots&0\\\vdots&\vdots&\ddots&\vdots\\-l_{n1}&0&\cdots&1\end{bmatrix}\ , \tag{2.26}$$

它是一个下三角矩阵，并且仅仅在一列 (具体到这个例子是第一列) 与单位矩阵不同，其他地方与单位矩阵相同。这样的矩阵在数学上称为 Frobinius 矩阵。它的作用是，只要我们适当地选取 l_{r1} 的数值，就可以将增广矩阵 $(\bar{A},\bar{b})$ 中第一列中除了第一个元素 $\bar{a}_{11}$ 之外的其他元素统统变换为零。这个具体的选择是 $l_{r1}=\bar{a}_{r1}/\bar{a}_{11}$，$r=2,\cdots,n$，其中我们假定了 $\bar{a}_{11}\neq 0$。容易证明，这个矩阵也是非奇异的，事实上它的逆矩阵也是一个 Frobinius 矩阵，只不过其中的 $-l_{r1}$ 都要换成 $+l_{r1}$。

因此，经过上述的两个变换，我们总可以将原先的最一般的增广矩阵 (A,b) 变换为如下的形式，

$$(A',b')=\begin{pmatrix}a'_{11}&a'_{12}&\cdots&a'_{1n}&b'_1\\0&a'_{22}&\cdots&a'_{2n}&b'_2\\\vdots&\vdots&\ddots&\vdots&\vdots\\0&a'_{n2}&\cdots&a'_{nn}&b'_n\end{pmatrix}\ ,(A',b')=G^{(1)}(\bar{A},\bar{b})=G^{(1)}P^{(1r)}(A,b)\ . \tag{2.27}$$

其中的第一个矩阵 $P^{(1r)}$ 的作用是，首先选择一个不为零的元素 a_{r1} 并且将它调到第一行，当然如果 a_{11} 本身就不等于零，这一步原则上也可以省略；第二个矩阵则将矩阵 $(\bar{A},\bar{b}) = P^{(1r)}(A,b)$ 的第一列从第二个元素开始往下的矩阵元全部清零。

经过上述两个步骤原先的增广矩阵 (A,b) 被约化了。具体来说，第一个待求的变量 x_1 仅仅出现在第一个方程之中，后面的 $(n-1)$ 个方程仅仅包含其余的 $(n-1)$ 个变量：$x_2,\cdots,x_n$。显然，我们可以对剩余的这 $(n-1)$ 个变量的增广矩阵 –也就是公式 (2.27) 中由绿色标记出来的矩阵 –继续利用上面提及的那两类变换，即将两行对掉 (注意，由于这时的对掉不牵涉第一行，因此不会破坏整个矩阵 (2.27) 的结构，也就是说，除了 a'_{11} 之外，第一列都是零) 以及将某行乘以一个数与另一行相加，我们将可以将其变为仅仅包含变量 $(x_3,\cdots,x_n)$ 的增广矩阵。这个过程可以一直迭代地做下去。最终的结果就是我们将增广矩阵变换为了我们希望的上三角的形式 (2.21) 并进一步利用反代的方法 (2.22) 获得线性系统的解。

前面提及的寻找的不为零的矩阵元 $\bar{a}_{11} = a_{r1}$ 有个专门的名称，叫做支点元 (pivot element)，或者简称支点 (pivot)。而这个过程称为支点遴选 (pivot selection)。虽然任意的不为零的元素都可以作为支点，但是直觉告诉我们它应当尽可能地远离零，因此一个自然的选择是

$$\bar{a}_{11} = \max_r |a_{r1}| \,. \tag{2.28}$$

的确，数学上可以证明这样的选择造成的误差比起随便胡乱选择的支点来说是比较小的。这个选择一般称为部分支点遴选 (partial pivoting)。与之相比，我们还可以进行所谓的完全支点遴选 (complete pivoting)。在完全支点遴选中，我们不仅仅局限于第一列，而是在所有矩阵元中挑选模最大的作为支点，即，

$$\bar{a}_{11} = \max_{r,s} |a_{rs}| \,. \tag{2.29}$$

随后我们将矩阵的第 1 行与第 r 行，第 1 列与第 s 列对换 (这相当于将原先的解 x 的第 1 个分量与第 s 个分量进行了一次对换)。[5] 这样一来，原先的增广矩阵 (A,b) 就变为了一个新的增广矩阵 $(\bar{A},\bar{b})$。然后可以进一步利用 $G^{(1)}$ 变换将其化为 (2.27) 的形式。我们可以将其简记为，

$$(A',b') = \left[\begin{array}{c|c|c} a'_{11} & a'^T & b'_1 \\ \hline 0 & \tilde{A} & \tilde{b} \end{array}\right] \,, \tag{2.30}$$

其中 a' 和 $\tilde{b}$ 都是具有 $(n-1)$ 个分量的矢量，[6] $\tilde{A}$ 则是一个 $(n-1)\times(n-1)$ 的复矩阵。下面的步骤就是对 $(n-1)$ 阶的增广矩阵 $(\tilde{A},\tilde{b})$ 继续重复上面的步骤就可以将其最终化为上三角形式。

[5] 如果愿意，这两个操作也可以用矩阵来表达：令 $P^{(1r)}\cdot(A,b) = (A",b")$，那么 $(\bar{A},\bar{b}) = (A"\cdot P^{(1s)},\bar{b})$。同时记住，$x_1$ 与 x_s 进行了对换。

[6] 其实还有左下角那个 0 也是，呵呵。

在第一步的支点遴选的过程中的置换矩阵 $P^{(1r)}$ 的上标 r 其实并不是必须的，它只是临时从各个行中计算出来的一个具有最大模的矩阵元的行指标而已。因此为了下面描述的方便，我们将这一步的置换矩阵记为 $P^{(1)}$，即隐去其上标 r。并且我们将约化后的矩阵记为 $(A^{(1)}, b^{(1)})$。原始的增广矩阵则给它一个上标 0，即 $(A^{(0)}, b^{(0)}) \equiv (A, b)$。这样一来第一步的约化可以表达为：

$$(A^{(1)}, b^{(1)}) = G^{(1)} P^{(1)} (A^{(0)}, b^{(0)}) \,. \tag{2.31}$$

利用这个记号，我们可以比较方便地将整个约化过程表述为，

$$\begin{cases} (A^{(j)}, b^{(j)}) = G^{(j)} P^{(j)} (A^{(j-1)}, b^{(j-1)}) \,, \quad j = 1, 2, \cdots, (n-1) \,. \\ (A^{(n-1)}, b^{(n-1)}) \equiv (U, c) \end{cases} \tag{2.32}$$

或者更为明确地写出，

$$(U, c) = G^{(n-1)} P^{(n-1)} G^{(n-2)} P^{(n-2)} \cdots G^{(1)} P^{(1)} (A, b) \,. \tag{2.33}$$

这个过程就是高斯消元法的完整过程。经过这个约化，原先的线性系统被成功约化为一个上三角线性系统。

¶ 高斯消元法并不是总能够一直进行下去的。在消元的过程之中，有可能会遇到支点为零的情形。这时候除非我们进行支点的重新选择，否则算法没有办法继续。因此，可以直接进行消元处理的矩阵比起一般的非奇异矩阵来说，需要更多的条件。关于一个矩阵可以直接进行高斯消元的充要条件，我们会在第 5节的定理 2.2中叙述。

下面我们列出进行部分支点遴选的高斯消元法的算法基本步骤。

Algorithm 1 部分支点遴选的高斯消元法

Require: 设 $A \in \mathbb{C}^{n \times n}$ 为一方阵。我们通过消元法化为上三角矩阵，然后利用反代法求解之。

【计算量】：大约 $2n^3/3$.

1: **for** $i = 1, \cdots, n$ **do**

2: 寻找一个不为零的支点元，令：

$$\bar{a}_{ii} = \max_{i \le r \le n} |a_{ri}| \,. \tag{2.34}$$

3: 将第 i 行与相应的第 r 行互换，这等价于左乘上一个置换矩阵 $P^{(i)}$；

4: 利用一般的 Frobinius 矩阵 (2.43)$G^{(i)}$ 左乘上上面得到的矩阵，从而将 $\bar{a}_{ii}$ 以下的矩阵元消为零；

5: **end for**

6: 这样一来矩阵已经化为上三角矩阵，可以进而利用反代法给出最后的解。

¶ 利用 GEM 进行消元并求解线性方程的计算量可以按照下列方法进行计算。消元的过程需要的计算量为 $2(n-1)n(n+1)/3 + n(n-1)$，随后求解两个三角线性系统还需

要 n^2 的计算量。因此整个过程需要计算量大约为 $(2n^3/3+2n^2)$，其领头阶的计算量为 $2n^3/3$。这对于量级为 $n\sim O(100)$ 的矩阵是没有任何问题的。

¶关于线性系统的稳定性与误差分析，我们需要引入矩阵的条件数的概念。给定一个非奇异的方阵 $A\in\mathbb{C}^{n\times n}$，它的条件数定义为，

$$K(A)=||A||\cdot||A^{-1}|| \,. \tag{2.35}$$

其中的模 $||\cdot||$ 可以是任意良好定义的模。例如，如果是我们前面定义的 $p-$ 模，我们会将相应矩阵的模记为 $K_p(A)$。一个矩阵的条件数一般来说依赖于模的选取。不过奇异矩阵的条件数总是趋于无穷大，而一个接近奇异的矩阵则具有非常大的条件数。虽然任何的矩阵模都可以用于条件数的定义，但是通常我们总是采取服从乘法的矩阵模。对于这类矩阵模我们有，

$$1=||AA^{-1}||\leq||A||\cdot||A^{-1}||=K(A) \,. \tag{2.36}$$

因此由服从乘法的矩阵模所定义的条件数总是大于 1 的。比较常用的是采用 p-模定义的矩阵模。这时候矩阵的条件数记为 $K_p(A)$。其中最为常用的是欧氏模定义的 $K_2(A)$。我们有，

$$K_2(A)=\frac{\sigma_1(A)}{\sigma_n(A)} \,, \tag{2.37}$$

其中 $\sigma_1(A)$ 和 $\sigma_n(A)$ 分别是矩阵 A 的最大和最小的奇异值(参见第 22节中奇异值分解的讨论)。如果局限于对称正定矩阵来说，我们有，

$$K_2(A)=\frac{\lambda_{\max}}{\lambda_{\min}} \,, \tag{2.38}$$

其中 $\lambda_{\max}$ 和 $\lambda_{\min}$ 分别是矩阵 A 的最大、最小本征值。

一般来说，如果考虑舍入误差等因素，我们的初始数据 –这包括矩阵 A 和方程的右边矢量 b–都是具有误差的。因此，我们真正求解出来解可以写成下面方程的形式，

$$(A+\delta A)(x+\delta x)=(b+\delta b) \,. \tag{2.39}$$

其中 δA，δb 衡量了初始数据的误差，而 δx 则显示了真实求出的解 $(x+\delta x)$ 对原始解 x 的偏差。那么我们有如下的结果，

定理 2.1 对于上面定义的线性方程我们有，

$$\frac{||\delta x||}{||x||}\leq\frac{K(A)}{1-K(A)||\delta A||/||A||}\left[\frac{||\delta b||}{||b||}+\frac{||\delta A||}{||A||}\right] \,. \tag{2.40}$$

特别值得注意的是，如果矩阵本身没有误差，即 $\delta A=0$，那么我们一定有 $||\delta x||/||x||\leq K(A)||\delta b||/||b||$。因此对一个大的条件数的矩阵来说，方程右边的相对误差会被放大。正因为如此，我们在求解线性方程组的过程之中，必须尽可能地保持矩阵的条件数不会变大。这也是为什么幺正变换 (相应的实矩阵为正交变换) 在矩阵的变换过程之中起了非常重要的意义。可以证明它们是保持一个矩阵的条件数的变换。因此在变换的过程之中不会将矩阵的条件数变差。

5 LU 分解

¶ 一个方阵 $A \in \mathbb{C}^{n\times n}$ 的 LU 分解是指将其分解为一个下三角和一个上三角矩阵的乘积：

$$A = LU\ , \tag{2.41}$$

其中 $L(U)$ 分别是下 (上) 三角矩阵。如果一个矩阵 A 的 LU 分解可以获得，那么求解它的线性方程可以转化为先后求解两个三角形矩阵的线性问题，而这个可以利用反代的方法解出。例如下面的两步走的方程的解 x 恰好就是 $Ax = b$ 的解，只要 $A = LU$：

$$y = Ux\ , Ly = b\ . \tag{2.42}$$

¶ 我们下面论证，前一节的高斯消元法恰好给出了矩阵 A 的一个 LU 分解。

按照上一节的讨论，利用一系列的置换矩阵与 Frobinius 矩阵的乘积，我们可以将 (A, b) 约化为上三角的形式 (U, c)。这些 Frobinius 矩阵的一般形式为，

$$G^{(j)} = \begin{pmatrix} 1 & \cdots & 0 & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \cdots & \cdots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & \cdots & 0 \\ \vdots & \cdots & \vdots & -l_{j+1,j} & 1 & \cdots & 0 \\ \vdots & \cdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & -l_{n,j} & 0 & \cdots & 1 \end{pmatrix}\ . \tag{2.43}$$

注意各个 $G^{(j)}$ 矩阵，如果我们遍历所有的 $j = 1, \cdots, n-1$，这些矩阵中非零的元素恰好填满一个 $n \times n$ 矩阵的左下三角区域，也就是主对角线的左下方的所有矩阵元。而且，由于 $G(j)$ 都是可逆的，因此我们实际上有，

$$U = G^{(n-1)}P^{(n-1)}G^{(n-2)}P^{(n-2)}\cdots G^{(1)}P^{(1)}A\ . \tag{2.44}$$

这个表达式实际上给出了矩阵 PA 的一个所谓的 LU 分解：[7]

$$PA = LU\ . \tag{2.45}$$

事实上任意方阵都可以分解为这种形式 (参见本页脚注中的论文中的 Corollary 3)，其中 $P = P^{(n-1)}\cdots P^{(1)}$ 是一系列置换矩阵的乘积。

利用 LU 分解可以解线性方程假定非奇异矩阵 A 可以分解一个下三角矩阵 L 与一个上三角矩阵 U 的乘积，即 $A = LU$，那么线性方程 $Ax = b$ 可以通过两个步骤来求解：

$$y = Ux\ , Ly = b\ . \tag{2.46}$$

[7] 如果我们定义矩阵 $M = G^{(n-1)}P^{(n-1)}\cdots G^{(1)}P^{(1)}$，那么可以证明，矩阵 $L \equiv PM^{-1}$ 一定仍然是下三角矩阵并且对角元都是 1。

其中每一步都是求解一个三角矩阵的线性方程组，这可以通过反代的方法给出。

于是一个关键的问题是，什么样的矩阵允许做这样的 LU 分解。显然，我们前面讨论的高斯消元法恰好给出了一个矩阵的 LU 分解，如果它存在的话。这个问题的关键实际上设计矩阵 A 以及它的各个子矩阵的秩。我们援引 arXiv 上面的一篇数学论文吧，这里就不赘述了。[8]

虽然对于任意矩阵的 LU 分解的定理有些复杂，但是对于一个实矩阵来说，它的判别标准还是比较简单的，这就是下面的定理。

定理 2.2 *对于任意的实矩阵 $A \in \mathbb{R}^{n\times n}$ 来说，它具有唯一的 LU 分解：$A = LU$，其中 L 是下三角矩阵且 $l_{ii} = 1$，$i = 1, \cdots, n$，U 为上三角矩阵的充要条件为 A 的所有主子矩阵 A_i，$i = 1, \cdots, (n-1)$ 都是非奇异的。*

特别值得注意的是，即使是一个奇异的 $n \times n$ 矩阵也可以有唯一的 LU 分解，只要它的各个主子矩阵一直到 $(n-1)$ 阶都是非奇异的即可。例如，对于奇异的矩阵 $A = \left[\begin{smallmatrix} 1 & 2 \\ 1 & 2 \end{smallmatrix}\right]$ 来说，我们可以获得它的标准的、唯一的 LU 分解：$A = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} 1 & 2 \\ 0 & 0 \end{smallmatrix}\right]$。对于不满足定理条件的矩阵来说，可能根本就没有 LU 分解的存在。例如大家可以验证，Pauli 矩阵 $\sigma_x = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ 就不存在任何的 LU 分解。

这个定理的证明可以利用对 i 的数学归纳法展开。其步骤非常类似于我们下面要讲述的 Cholesky 分解中的证明，我们这里不再赘述。

6 Cholesky 分解

¶ 正如前面提到的，对于正定的厄米矩阵 $A \in \mathbb{C}^{n\times n}$ 来说，我们可以找到一个矩阵 H 使得 $A = H^\dagger H$。事实上，我们可以要求矩阵 H 是上三角矩阵。这个分解一般称为 Cholesky 分解。一旦这样的 L 求得之后，求解线性方程 $A\mathbf{x} = \mathbf{b}$ 的问题就可以分为两步进行：$H\mathbf{x} = \mathbf{y}$，$H^\dagger \mathbf{y} = \mathbf{b}$，每一步都只涉及三角矩阵的线性系统。因此，对于求解正定厄米矩阵的线性方程问题就化为寻找矩阵的 Cholesky 分解问题。

按照我们前面提及的线性代数的结果，正定的厄米矩阵的所有主子矩阵也都是正定的。因此，寻找 Cholesky 分解可以按照数学归纳法的思路进行。对于 $n = 1$ 的一阶矩阵，问题的解是平庸的。令 A_i，$i = 1, \cdots, n$ 是原矩阵的第 i 阶的主子矩阵。它们显然也都是正定的厄米矩阵。假定我们已经找到了 $A_{i-1} \in \mathbb{C}^{(i-1)\times(i-1)}$ 的分解矩阵 H_{i-1}，即 $A_{i-1} = H_{i-1}^\dagger H_{i-1}$，我们试图来寻找 A_i 的分解矩阵 H_i。为此，我们将矩阵 A_i 表达为

$$A_i = \begin{bmatrix} A_{i-1} & \mathbf{v} \\ \mathbf{v}^\dagger & \alpha \end{bmatrix}, \tag{2.47}$$

其中 α 是一个正的实数，$\mathbf{v} \in \mathbb{C}^{i-1}$ 为一矢量。事实上我们有，$\mathbf{v} = (a_{1i}, a_{2i}, \cdots, a_{i-1,i})^T$。

[8] Pavel Okunev Charles R. Johnson, "Necessary and sufficient condition for existence of LU factorization of an arbitrary matrix", arXiv:math/0506382.

我们希望矩阵 A_i 具有的分解为：

$$A_i = H_i^\dagger H_i = \begin{bmatrix} H_{i-1}^\dagger & 0 \\ \mathbf{h}^\dagger & \beta \end{bmatrix} \cdot \begin{bmatrix} H_{i-1} & \mathbf{h} \\ 0^\dagger & \beta \end{bmatrix} \tag{2.48}$$

其中的 $\mathbf{h} \in \mathbb{C}^{i-1}$ 为一待定矢量而 β 为一个待定实数。将这个式子的右边乘出来我们就发现：

$$H_{i-1}^\dagger \cdot \mathbf{h} = \mathbf{v} \,, \tag{2.49}$$

同时 $\mathbf{h}^\dagger\mathbf{h} + \beta^2 = \alpha$。由于 $H_{i-1}^\dagger$，$\mathbf{v}$ 已知，并且 $H^\dagger$ 是一个下三角矩阵，我们当然可以轻易求解出矢量 $\mathbf{h}$。另一方面，$\beta = \sqrt{\alpha - \mathbf{h}^\dagger\mathbf{h}}$ 则给出了参数 β 的数值。

正定厄米矩阵的 Cholesky 分解可以通过下面的算法获得。

Algorithm 2 正定厄米矩阵的 Cholesky 分解

Require: 设 $A \in \mathbb{C}^{n\times n}$ 为一正定厄米矩阵。我们需要获得上三角矩阵 H 使得 $A = H^\dagger H$ 记下三角矩阵 $H^\dagger$ 的矩阵元为 $h_{i,j}$。首先令 $h_{11} = \sqrt{a_{11}}$，然后

【计算量】：大约 $n^3/3$ 的计算量 (比起 LU 分解要少一半左右).

1: **for** $i = 2, \cdots, n$ **do**

2: 计算 h_{ij} 其中 $j = 1, \cdots, (i-1)$

$$h_{ij} = \frac{1}{h_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} h_{ik} h_{jk} \right) \,, \quad j = 1, \cdots, (i-1) \,. \tag{2.50}$$

3: 计算 h_{ii}：

$$h_{ii} = \left[a_{ii} - \sum_{k=1}^{i-1} h_{ik}^2 \right]^{1/2} \,. \tag{2.51}$$

4: **end for**

Cholesky 分解的计算量大约是 $n^3/3$，这比通常的 LU 分解要节省大约一半。这主要来源于对于对称性的运用。从稳定性上来说，Cholesky 分解的稳定性极佳，只要矩阵确实是正定的厄米矩阵。比较常用的情形是运用在解决 χ^2 拟合过程中的线性方程的求解，那里出现的协方差矩阵注定是正定对称实矩阵，参见第 七章中第 28.4小节的讨论。

7 三对角矩阵线性方程组的求解

¶ 本节我们讨论一个三对角矩阵所给出的线性方程组的求解问题。这类问题会出现在不少的数值应用之中，例如后面会讨论到的利用三次样条函数进行拟合的问题 (参见第 10节)。

我们将问题中的三对角矩阵的 LU 分解写为如下的形式，

$$A=\begin{bmatrix} a_1 & c_1 & \cdots & 0 \\ b_2 & a_2 & \ddots & \\ & \ddots & & c_{n-1} \\ 0 & & b_n & a_n \end{bmatrix}=LU\ , \tag{2.52}$$

其中的矩阵 L 和 U 分别是下和上双对角矩阵：

$$L=\begin{bmatrix} 1 & & \cdots & 0 \\ \beta_2 & 1 & \ddots & \\ & \ddots & \ddots & \\ 0 & & \beta_n & 1 \end{bmatrix},U=\begin{bmatrix} \alpha_1 & c_1 & \cdots & 0 \\ & \alpha_2 & \ddots & \\ & & \ddots & c_{n-1} \\ 0 & & & \alpha_n \end{bmatrix}. \tag{2.53}$$

参数 α_i 以及 β_i 与原先的系数之间的关系有下式给出，

$$\alpha_1=a_1\ ,\quad \beta_i=\frac{b_i}{\alpha_{i-1}},\alpha_i=a_i-\beta_i c_{i-1},\ i=2,\cdots,n\ . \tag{2.54}$$

这个算法又称为 Thomas 算法 (Thomas algorithm)。经过这个分解之后，我们可以利用该分解求解方程 $Ax=b$。这个过程的计算量大约是 $8n-7$。具体来说，上述分解本身需要 $3(n-1)$，而求解两个三角系统的计算量为 $5n-4$。

Algorithm 3 三对角矩阵的 LU 分解 (Thomas 算法)

Require: 设 $A\in\mathbb{C}^{n\times n}$ 为已知三对角矩阵。令 $A=LU$，其中 L 和 U 由公式 (2.53) 给出。

本算法给出计算各系数 $\beta_2,\cdots,\beta_n$，$\alpha_1,\cdots,\alpha_n$ 的计算步骤

【计算量】：大约 $8n-7$.

1: 令 $\alpha_1=a_1$。

2: **for** $i=2,\cdots,n$ **do**

3:

$$\begin{cases} \beta_i=\dfrac{b_i}{\alpha_{i-1}}, \\ \alpha_i=a_i-\beta_i c_{i-1},\ . \end{cases} \tag{2.55}$$

4: **end for**

5: 最后可以进而利用反代法给出最后的解。

这个算法的稳定性也是不错的。如果我们假定非奇异的矩阵 A 有一个小的误差 δA，即令 $A+\delta A=\hat{L}\hat{U}$，那么我们可以证明

$$|\delta A|\le(4u+3u^2+u^3)|\hat{L}|\cdot|\hat{U}|\ , \tag{2.56}$$

其中符号 $|\cdot|$ 表示一个矩阵的特殊的模，及每个矩阵元的模的最大的。$u=\epsilon_M/2$ 则表示机器舍入误差单位。这个估计说明非奇异三对角矩阵的 LU 分解基本上是稳定的，只要各

个系数 β_i 以及 α_i 不会太大。这些系数变大的一种可能是其中某个 α_i 非常接近于零。因此，如果矩阵的确是非奇异的，那么各个 α_i 必定不能等于零。但是如果矩阵接近于奇异，那么有可能造成算法出现不稳定的情况。如果矩阵 $A \in \mathbb{R}^{n\times n}$ 是一个正定实对称矩阵，那么我们可以获得更好的估计，

$$|\delta A| \leq \frac{4u + 3u^2 + u^3}{1-u}|A| \,, \tag{2.57}$$

这是分解更为稳定。

相关的阅读

本章着重讨论了简单的线性方程组的求解过程。

第三章　　内插与函数的计算

本章提要

- ☞ 多项式内插
- ☞ 有理分式内插
- ☞ 样条函数内插
- ☞ 函数的近似与计算
- ☞ 函数的导数的计算

前一章中我们讨论了数值线性方程的解。本章中我们将涉及函数的内插或外推问题。这类问题广泛地出现在各类的实验科学中。

内插面对的基本问题大致如下：假定我们在一系列控制变量 (自变量)x 取值处 –例如我们在 $x = x_0, x_1, \cdots, x_n$ 这 $(n+1)$ 个两两不同的点处 –获得了相应的函数 $y = f(x)$ 的数值：$y_0, y_1, \cdots, y_n$，并且如果我们对函数 $y = f(x)$ 的宏观性状做某些合理的假定 (例如，它是什么类型的函数等等)，我们是否可以根据这些信息完全确定，或者近似地确定这个函数本身？如果能够“近似地”确定这个函数，那么我们就不再需要去测量其他 x 处 (即 $x \neq x_i, i = 0, \cdots, n$ 处) 的函数值，而可以直接“近似地”计算它。这种方法是有意义的是因为一方面在我们没有测量同时又希望了解的那些点处的实验可能是十分耗时，甚至是不可能实现的；另一方面，我们有时候需要在一定的定义域内获得相应物理量 Y 的一个简单的表达式，这个表达式可能可以用于关于这个函数的进一步计算。

不失一般性，我们可以假设已经测量的这些自变量的点是按照由小到大排列的。它们被称为内插问题中的支撑点 (support points) 或者节点 (nodes)。即：$x_0 < x_1 < \cdots < x_n$ 而整数 n 则称为支撑点的数目。我们要求内插的函数 $f(x)$ 必须满足：

$$f(x_i) = y_i\ ,\quad i = 0, 1, \cdots, n\ . \tag{3.1}$$

显然，支撑点数目越大，我们对函数的了解就越精细。如果我们希望计算的其他点 $x \in [x_0, x_n]$，这个问题称为内插问题；而如果 $x < x_0$ 或 $x > x_n$，这个问题称为外推问题。当然，笼统地说我们也可以统称其为内插问题。

外推则不太一样。这时候我们感兴趣的是函数在某个已知区间之外的点的函数值。而且，如果我们不能够对函数的形式进行任何的限制的话，外推可能得到完全疯狂的结果。

注意内插问题与我们后面会讨论的拟合问题是不太一样的。内插问题中我们假定在支撑点 x_i 处测得的函数值 y_i 是严格的。在拟合问题中，我们一般是需要考虑 y_i 的误差的。但是在内插问题中，我们认为在支撑点处的值是严格的。因此，在待定函数之中的参数的数目一般也与支撑点的数目相同。所不同的是函数的形式。函数形式的选择则依赖于我们对问题的物理理解。一般来说，如果我们认为待定的函数在我们研究的区间应当是无限光滑解析的，我们会选择诸如多项式这类的函数；反之，如果我们认为所研究的函数在该区间中可能有奇异性，我们则可以选择具有极点的分式函数。在下面的几节中我们依次来讨论这些内插方法。最后我们会讨论使用样条函数进行分段拟合的情况，这种情形是函数在整个区间中并没有一个统一的形式。样条函数特别是三次样条函数在计算机图像学中有非常广泛的应用。

本章中我们还将讨论函数的计算问题。这包括两种情形。一直情形是函数具有明确的解析表达式。这个表达式可能是由级数或者一个积分表达式给出。我们需要的是从数值上准确地计算这个函数值。另外一类是首先利用某种近似来描写目标函数，然后再计算相应的函数。这一类与内插问题有类似之处。因此函数的计算无疑是我们后面几章讨论函数的积分、方程求根以及函数极值问题的基础。

8 多项式内插

¶ 本节中我们讨论多项式的内插。考虑区间 $[x_0, x_n]$ 上面的 n 次多项式，

$$P_n(x) = a_0 + a_1 x + \cdots + a_n x^n \ . \tag{3.2}$$

它具有 $(n+1)$ 个参数：a_i，$i = 0, 1, \cdots, n$。我们试图利用这个多项式来解决区间 $[x_0, x_n]$ 上面的内插问题。也就是说，我们希望有，$P_n(x_i) = y_i, i = 0, \cdots, n$。这个问题的一般解是著名的拉格朗日多项式。

我们考虑如下形式的 n 次多项式：

$$L_j(x) = \prod_{0 \le m \le n m \ne j} \frac{x - x_m}{x_j - x_m} \ , j = 0, 1, \cdots, n \ . \tag{3.3}$$

按照构造，这个多项式满足

$$L_j(x_i) = \delta_{ij} \ , 0 \le i, j \le n \ . \tag{3.4}$$

于是如果我们令

$$P_n(x) \equiv \sum_{i=0}^{n} y_i L_i(x) \ , \tag{3.5}$$

我们可以直接验证它满足 $P_n(x_i) = y_i$，$i = 0, 1, \cdots, n$。这个公式一般称为拉格朗日内插公式 (Lagrange, 1795)。

¶另外一种多项式的内插方法是牛顿提出来的。它的特点是随着支撑点的增多，在利用已有的多项式的情形下，逐阶地提高多项式的阶数。

$$\begin{cases} N(x) = \sum_{i=0}^{n} a_i n_i(x) \ , \\ n_i(x) = \prod_{k=0}^{i-1} (x - x_k) \ , \quad n_0(x) \equiv 1 \ . \end{cases} \tag{3.6}$$

这个多项式的好处是，如果我们已经从 n 个支撑点的信息中获得了 $n_i(x), i = 0, 1, \cdots, n-1$，现在假设我们又增加了一个支撑点 (x_n, y_n)，我们需要的仅仅是计算 a_n 和 $n_n(x)$。显然，$n_n(x) = (x - x_n) n_{n-1}(x)$ 而系数 a_n 则可以通过求解下面的线性方程组来获得：

$$\begin{bmatrix} 1 & & \cdots & \cdots & 0 \\ 1 & x_1 - x_0 & & \cdots & \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) & & \vdots \\ \vdots & \vdots & & \ddots & \\ 1 & (x_n - x_0) & \cdots & \cdots & \prod_{i=0}^{n-1} (x_n - x_i) \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ \vdots \\ \vdots \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ \vdots \\ \vdots \\ y_n \end{bmatrix} \ . \tag{3.7}$$

这是一个下三角矩阵，因此可以利用前一章中讨论过的方法进行求解。同时，加入新的支撑点仅仅需要重新求解线性方程组的最后一行即可。

¶另外一种经常使用的多项式内插方法是所谓的 Neville 算法。我们下面就来介绍这种算法。假定我们的 $n+1$ 个节点为 (x_i, f_i)，$i = 0, 1, \cdots, n$。我们利用下面的递推关系定义一系列多项式：

$$\begin{aligned} P_i(x) &\equiv f_i \quad i = 0, 1, 2, \cdots, n \ . \\ P_{i_0 i_1 \cdots, i_k}(x) &= \frac{(x - x_{i_0}) P_{i_1 \cdots i_k}(x) - (x - x_{i_k}) P_{i_0 \cdots i_{k-1}}(x)}{x_{i_k} - x_{i_0}} \end{aligned} \tag{3.8}$$

根据这个定义并且利用数学归纳法不难证明，这个多项式满足

$$P_{i_0 i_1 \cdots, i_k}(x_{i_j}) = f_{i_j} \ , \quad j = 0, 1, \cdots, k \ . \tag{3.9}$$

因此它满足过各个节点的条件。具体的构造的时候比较有效的办法是列出一个三角形的

图来：

$$\begin{array}{c|c|cccc}
x_k & k=0 & 1 & 2 & 3 & \\
\hline
x_0 & P_0(x)\equiv f_0 & & & & \\
 & & P_{01}(x) & & & \\
x_1 & P_1(x)\equiv f_1 & & P_{012}(x) & & \\
 & & P_{12}(x) & & P_{0123}(x) & \\
x_2 & P_2(x)\equiv f_2 & & P_{123}(x) & \vdots & \\
 & & P_{23}(x) & \vdots & & \\
x_3 & P_3(x)\equiv f_3 & \vdots & & & \\
\vdots & \vdots & & & &
\end{array} \tag{3.10}$$

例如上面的图就显示了构造 $P_{0123}(x)$ 的过程：$k=0$ 的一列对应于原始的节点的函数值；$k=1$ 的一列中的各个函数可以通过它的左方一列中上下相邻的两个元素，根据迭代式 (3.8) 给出来，例如：

$$\begin{aligned}
P_{01}(x) &= \frac{(x-x_0)P_1(x)-(x-x_1)P_0(x))}{x_1-x_0}\,, \\
P_{12}(x) &= \frac{(x-x_1)P_2(x)-(x-x_2)P_1(x))}{x_2-x_1}\,, \\
P_{23}(x) &= \frac{(x-x_2)P_3(x)-(x-x_3)P_2(x))}{x_3-x_2}\,,\vdots
\end{aligned} \tag{3.11}$$

然后可以进一步获得 $k=2,3\cdots$ 的各个列。当我们需要增加一个节点的时候，我们只需要重新计算这个三角图的右下角的一个边上出现的数据。例如如果我们需要加上第五个节点 x_4，我们就只需要在图中的标有“$\vdots$”的地方填入相应的数据即可，三角图中心部分的数据不再需要再重新进行计算。一个常用的记号上的简化是，

$$T_{i+k,k}=P_{i,i+1,\cdots,i+k}(x)\,. \tag{3.12}$$

这样一来上面的三角图就变为，

$$\begin{array}{c|c|cccc}
x_0 & T_{00}=f_0 & & & & \\
 & & T_{11} & & & \\
x_1 & T_{10}=f_1 & & T_{22} & & \\
 & & T_{21} & & T_{33} & \\
x_2 & T_{20}=f_2 & & T_{32} & \vdots & \\
 & & T_{31} & \vdots & & \\
x_3 & T_{30}=f_3 & \vdots & & & \\
\vdots & \vdots & & & &
\end{array} \tag{3.13}$$

Algorithm 4 多项式内插的 Neville 算法

Require: 给定 $(n+1)$ 个节点信息：(x_j, f_j)，$j = 0, 1, \cdots, n$，Neville 算法给出一个 n 阶的多项式恰好过给定的 $(n+1)$ 个节点。

【计算量】：对于每个给定的 x 大约 $O(n^2)$.

1: 首先填充三角图的第一列：令 $T_{j0} = f_j$，$j = 0, 1, 2 \cdots, n$。

2: **for** $k = 1, \cdots, j$ **do**

3:

$$\begin{aligned} T_{j,k} &= \frac{(x - x_{j-k})T_{j,k-1} - (x - x_j)T_{j-1,k-1}}{x_j - x_{j-k}} , \\ &= T_{j,k-1} + \frac{T_{j,k-1} - T_{j-1,k-1}}{\frac{x-x_{j-k}}{x-x_j} - 1} . \end{aligned} \tag{3.14}$$

4: **end for**

5: 最后的 n 阶多项式多项式由 $T_{n,n}$ 给定。

在第 四章讨论数值积分时，我们会看到 Neville 算法的一个非常直接的应用，参见第 14节。

¶ 对于多项式内插而言，我们有如下的误差估计。假设我们需要描写的函数 $f(x)$ 在区间 $[x_0, x_n]$ 上至少具有 $(n+1)$ 阶的导数，那么对于任意的 $x \in [x_0, x_n]$，我们一定可以找到一个 $\xi \in [x_0, x_n]$ 使得，

$$f(x) - P_n(x) = \frac{\omega(x) f^{(n+1)}(\xi)}{(n+1)!} , \tag{3.15}$$

其中 $\omega(x) = (x - x_0) \cdots (x - x_n)$。也就是说，我们必定有

$$|f(x) - P_n(x)| \leq \frac{|x_n - x_0|^n}{(n+1)!} \max_{x_0 \leq \xi \leq x_n} |f^{(n+1)}(\xi)| \tag{3.16}$$

虽然我们有这个估计，但是它并不一定能够保证随着 n 的增加，误差就必定减小。

¶ 初看起来这个构造似乎已经完美解决了内插问题，至少是解决了利用进行多项式内插问题。但是实际上它是有一定的问题的。这个问题是当我们测量的支撑点的数目增加时，多项式的次数也随之增加。我们会发现，尽管在所有的支撑点处拉格朗日内插公式都满足 $P_n(x_i) = y_i$，但是在那些不是支撑点的地方，拉格朗日内插多项式可能与我们期望内插的函数 –这个函数的形式我们并不清楚 –可能相差很远。这个事实的另外一种体现是，当我们希望在已有的 $(n+1)$ 个支撑点之外再曾经哪怕一个支撑点，比如 (x_{n+1}, y_{n+1})，我们需要完全重新计算整个内插多项式的所有系数。也就是说在整个区间来看，$|P_n(x) - P_{n+1}(x)|$ 可以很大，尽管在所有的支撑点处它们的差别等于零。这个现象被称为 Runge 现象 (Runge's phenomenon)。

Runge 当年考虑的函数具体的形式为

$$f(x) = \frac{1}{1 + 25x^2} . \tag{3.17}$$

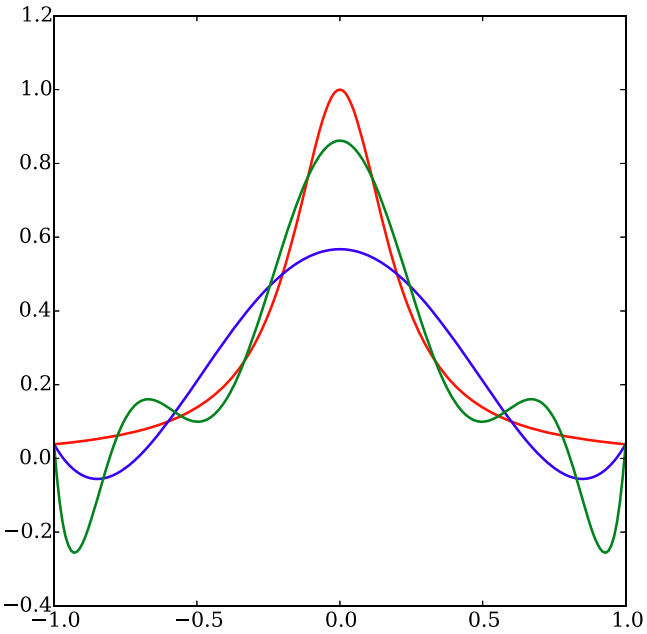


图 3.1: 利用拉格朗日多项式拟合时的 Runge 现象。图中显示的是所谓的 Runge 函数 $f(x) = 1/(1+25x^2)$ 在区间 $x \in [-1,+1]$ 之间的图像 (红线)。同时显示的是 $n = 5$ 阶 (蓝线) 和 $n = 9$ 阶 (绿线) 的拉格朗日拟合。本图取自 Wikipedia 网页:`https://en.wikipedia.org/wiki/Runge's_phenomenon`。

这个函数一般称为 Runge 函数。我们在图 3.1中画出了这个函数在 $x \in [-1,+1]$ 之间的图像 (红线)。同时显示的还有 $n = 5$ 阶 (蓝线) 和 $n = 9$ 阶 (绿线) 的拉格朗日多项式拟合。前者在 6 个支撑点处与 Runge 函数 (红线) 吻合；后者则在 10 个支撑点处与 Runge 函数吻合。但是只要不在这些支撑点处，随着内插阶数的升高，拉格朗日多项式体现出越来越严重的震荡特性。不在这些支撑点处，拉格朗日多项式严重地偏离原函数。这就是所谓的 Runge 现象。事实上可以证明

$$\lim_{n\to\infty} \left(\max_{-1\le x\le +1} |f(x) - P_n(x)| \right) = +\infty \,. \tag{3.18}$$

也就是说，随着内插阶数的升高，内插多项式与原函数的最大偏离的绝对值会发散。

在相当多的应用中，内插的目的恰恰是希望得到一定区间内没有测量过的点处的函数值。如果我们的内插函数在支撑点之外的地方可能严重偏离实际的函数，那这种偏差对这一类的应用是不可原谅的。因此我们希望克服这种弊端。具体到 Runge 函数本身，使用分式拟合就可以很好地解决这个问题。或者使用所谓的三次样条函数来进行内插。由于 Runge 函数本身就是一个分式的形式，如果使用正确的分式内插可以完全准确地确定这个函数；而如果我们使用三次样条函数来进行内插，我们也可以获得相当不错的结果。

9 有理分式内插

¶本节我们讨论利用有理分式进行内插。当函数在某个区间内的变化行为比较剧烈时，多项式内插会出现剧烈震荡的行为。这时候利用有理分式进行内插可能会更为合适一些。

考虑一个分式，

$$\Phi^{(m,n)}(x) = \frac{P_m(x)}{Q_n(x)} , \tag{3.19}$$

其中分子和分母分别是 m 阶和 n 阶的多项式。我们下面将假定这两个多项式是互素的，也就是说并不存在一个公共的多项式因子。

¶下面我们直接构造这样的有理函数。从 $i = 0, 1, \cdots$ 开始，我们用下面的次序来递推：

i	x_i	y_i	其他		
0	x_0	y_0			
1	x_1	y_1	$\phi(x_0, x_1)$		
2	x_2	y_2	$\phi(x_0, x_2)$	$\phi(x_0, x_1, x_2)$	
3	x_3	y_3	$\phi(x_0, x_2)$	$\phi(x_0, x_1, x_3)$	$\phi(x_0, x_1, x_2, x_3)$
$\vdots$					$\vdots$

(3.20)

其中的各个函数 ϕ 可以按照下式进行递推地定义：

$$\begin{aligned} \phi(x_i, x_j) &= \frac{x_i - x_j}{y_i - y_j} , \\ \phi(x_i, x_j, x_k) &= \frac{x_j - x_k}{\phi(x_i, x_j) - \phi(x_i, x_k)} , \\ &\vdots \\ \phi(x_i, \cdots, x_l, x_m, x_n) &= \frac{x_m - x_n}{\phi(x_i, \cdots, x_l, x_m) - \phi(x_i, \cdots, x_l, x_n)} . \end{aligned} \tag{3.21}$$

显然，为了获得具体的数而不是发散的结果，我们应当尽可能选择函数单调的一个区间进行内插的构造。由于我们总是假设各个 x_i 是不相同的，因此如果函数是单调的，那么上述各个构造的差值比 ϕ 就不会发散。一旦获得了这些系数，我们可以构造一个有理分式：

$$\Phi^{(n,n)}(x) = \frac{P_n(x)}{Q_n(x)} , \tag{3.22}$$

它是两个 n 次多项式的比。我们要求它能够经过 $(2n+1)$ 个点 (x_i, y_i)，$i = 0, 1, \cdots, 2n$。事实上这个结果可以写成一个连分数：

$$\Phi^{(n,n)}(x) = y_0 + \cfrac{x - x_0}{\phi(x_0, x_1) + \cfrac{x - x_1}{\phi(x_0, x_1, x_2) + \cfrac{x - x_2}{\ddots + \cfrac{x - x_{2n-1}}{\phi(x_0, \cdots, x_{2n})}}}} . \tag{3.23}$$

可以证明这个分式恰好通过给定的 $2n+1$ 个点，即满足 $\Phi^{(n,n)}(x_i)=y_i$，$i=0,1,\cdots,2n$。

作为一个例子，我们考虑前面曾经提及的 Runge 函数：

$$f(x)=\frac{1}{1+25x^2}\,. \tag{3.24}$$

我们选择其单调的区间，比如 $[0,1]$ 来进行内插。假定知道下列三个点的数值：$x^2=0,1/25,1$，相应的函数值分别为：1,1/2,1/26。按照上面的构造，按照 x^2 的有理分式为，

$$f(x)=\Phi^{(1,1)}(x^2)=1+\frac{x^2}{\phi(0,1/25)+\dfrac{x^2-1/25}{\phi(0,1/25,1)}} \tag{3.25}$$

其中的几个系数，

$$\begin{aligned}\phi(0,1/25)&=\frac{0-1/25}{1-1/2}=-\frac{2}{25}\,,\\ \phi(0,1)&=\frac{0-1}{1-1/26}=-\frac{25}{26}\,,\\ \phi(0,1/25,1)&=\frac{1/25-1}{-2/25+26/25}=-1\,.\end{aligned} \tag{3.26}$$

将这些数值带入前一式我们发现，我们竟然可以完全重构 $f(x)$。

另外一个例子，让我们考察 $f(x)=\cot(x)$ 的计算问题。

Remez 算法可以用来计算这种近似，而这种算法与所谓的 Chebyshev 多项式密切相关。

10 样条函数内插

¶ 这一节中，我们来讨论利用样条函数 (spline function) 来进行内插。

考虑一个区间 $[a,b]$ 的一个 n 段分割：$a=x_0<x_1\cdots x_{n-1}<x_n=b$。位于区间中间的那些点 x_i，$i=1,\cdots,n-1$ 称为节点 (knots)。我们考虑在该区间上的一个所谓三次样条函数 (cubic spline function) $S_\Delta:[a,b]\to\mathbb{R}$，它满足 $S_\Delta\in C^2[a,b]$ 并且在每个子区间 $[x_i,x_{i+1}]$ 上均为一个三次多项式并且它们在每个节点处函数值以及一二阶导数值都连续 (平滑衔接)。如果给定的一组函数值记为 $Y=\{y_0,y_1,\cdots,y_n\}$。我们把这样的三次样条差值函数记为 $S_\Delta(Y;\cdot)$，它满足 $S_\Delta(Y;x_i)=y_i$，$i=0,1,\cdots,n$。

上述信息还没有完全确定函数 $S_\Delta(Y;\cdot)$ 的性质。可以证明还有两个自由度可以调节。这两个自由度一般通过在两个端点处的边条件来加以限制。边条件大致可以分为下列三种：第一种是 Dirichlet 边条件；第二种是 Neumann 边条件；第三种是周期边条件。可以证明，上述三种方案的任何一种都唯一地确定了一个三次样条差值函数 $S_\Delta(Y;\cdot)$。样条函数被广泛的应用是因为它实际上并不像高阶的多项式内插那样会呈现出剧烈震荡的 Runge 现象。

我们现在考虑某个区间 $[a,b]$ 上的 $m>0$ 阶以下的所有阶导数连续且其 m 阶导数平方可积的函数构成的函数空间，将其记为 $\mathcal{K}^m[a,b]$。我们同时会用 $\mathcal{K}_p^m[a,b]$ 来标记所有的 m 阶以下导数都满足周期性的函数，即 $f^{(k)}(a)=f^{(k)}(b)$，$k=0,\cdots,m-1$。对于任意的函数 $f\in\mathcal{K}^2[a,b]$，我们可以定义一个函数的模：

$$||f||=\int_a^b dx|f''(x)|^2\,. \tag{3.27}$$

我们知道一个函数 $f(x)$ 的曲率 $R(x)=f''(x)(1+f'(x)^2)^{-3/2}$。如果在一个区间上 $f'(x)^2$ 比起 1 来说要小很多话，那么函数的曲率几乎就等于 $f''(x)$。因此，上面这个模从某种意义上是衡量了函数在一个区间上曲率模方的大小。可以证明，三次样条函数实际上是使得这个模最小的函数。具体来说，我们不加证明地引用下列定理，它一般称为最小模定理。从直观上说，它是“最光滑的”函数。

定理 3.1 实区间 $[a,b]$ 的任一 n 段分割 Δ：$a=x_0<x_1\cdots x_{n-1}<x_n=b$ 上的三次样条函数 $S_\Delta(Y;x_i)=y_i$，$i=0,1,\cdots,n$。对任给的 $f\in\mathcal{K}^2[a,b]$ 且 $f(x_i)=y_i$，我们一定有 $||f||\geq||S_\Delta(Y;\cdot)||$。其中的函数模由式 (3.27) 所定义。事实上可以证明，

$$||f-S_\Delta(Y;\cdot)||^2=||f||^2-||S_\Delta(Y;\cdot)||^2\geq 0\,, \tag{3.28}$$

只要函数 f 满足下列条件之一即可：

1. $S''_\Delta(Y;a)=S''_\Delta(Y;b)=0$;
2. $f\in\mathcal{K}_p^2[a,b]$，$S_\Delta(Y;\cdot)$ 是周期的;
3. $f'(a)=S'_\Delta(Y;a)$，$f'(b)=S'_\Delta(Y;b)$.

并且上面的任何一个条件都唯一地确定了样条函数 $S_\Delta(Y;\cdot)$。

¶ 下面我们具体讨论如何计算确定一个区间上的三次样条函数。我们会发现，它实际上对应于求解一个三对角矩阵的线性方程组。

首先，对于任意的一个 n 段分割 Δ：$a=x_0<x_1\cdots x_{n-1}<x_n=b$，我们首先引入下列记号:

$$h_{j+1}=x_{j+1}-x_j\,,j=1,\cdots,n\,, \tag{3.29}$$

它标志了每一段的长度。其次，我们注意到对于三次样条函数 $S_\Delta(Y;\cdot)$ 来说，它的二次导数 $S''_\Delta(Y;\cdot)$ 在每一段上面必定是一个一次函数。我们将其在各个节点处的数值称为三次样条函数的矩 (moments) 并记为，

$$M_j=S''_\Delta(Y;x_j)\,,j=0,1,\cdots,n-1,n\,. \tag{3.30}$$

对于每一小段 $x\in[x_j,x_{j+1}]$，$j=0,\cdots,n-1$ 上的线性函数 $S''_\Delta(Y;\cdot)$，我们可以验明，

$$S''_\Delta(Y;x)=M_j\frac{x_{j+1}-x}{h_{j+1}}+M_{j+1}\frac{x-x_j}{h_{j+1}}\,. \tag{3.31}$$

以此式为出发点我们可以积分一次获得，

$$S'_\Delta(Y;x) = -M_j\frac{(x_{j+1}-x)^2}{2h_{j+1}} + M_{j+1}\frac{(x-x_j)^2}{2h_{j+1}} + A_j\ , \tag{3.32}$$

进一步再积分一次得到，

$$S_\Delta(Y;x) = M_j\frac{(x_{j+1}-x)^3}{6h_{j+1}} + M_{j+1}\frac{(x-x_j)^3}{6h_{j+1}} + A_j(x-x_j) + B_j\ . \tag{3.33}$$

利用条件 $S_\Delta(Y;x_j)=y_j$，$S_\Delta(Y;x_{j+1})=y_{j+1}$ 我们就可以求出积分常数 A_j 和 B_j，其结果如下：

$$\begin{cases} B_j = y_j - M_j\dfrac{h_{j+1}^2}{6}\ , \\ A_j = \dfrac{y_{j+1}-y_j}{h_{j+1}} - \dfrac{h_{j+1}}{6}(M_{j+1}-M_j) \end{cases} \tag{3.34}$$

公式 (3.33) 和公式 (3.34) 告诉我们，一个任意 n 点分割 Δ 上的三次样条函数 $S_\Delta(Y;\cdot)$ 完全由其在各个节点 x_j 处的矩 M_j 所完全确定。于是，剩下的任务就是根据已知的条件确定样条函数在各节点处的矩。

我们现在还没有利用 $S'_\Delta(Y;x_j)$ 在各个节点处平滑衔接的条件。将公式 (3.34) 带入公式 (3.32) 并要求 x_j 两侧的导数平滑衔接 $S'_\Delta(Y;x_j+0)=S'_\Delta(Y;x_j-0)$，我们得到，

$$\frac{h_j}{6}M_{j-1} + \frac{h_j+h_{j+1}}{3}M_j + \frac{h_{j+1}}{6}M_{j+1} = \frac{y_{j+1}-y_j}{h_{j+1}} - \frac{y_j-y_{j-1}}{h_j}\ . \tag{3.35}$$

这个公式中的 $j=1,2,\cdots,n-1$ 给出了 $(n-1)$ 个方程。但是我们的未知数 M_j 共有 $(n+1)$ 个。我们仍然需要两个条件才可能完全确定所有的矩 M_j 进而确定三次样条函数。这两个额外的条件恰恰由我们定理 3.1中所提及的，函数在端点处的第一类、第二类或周期性条件给出。

公式 (3.35) 已经体现出典型的三对角矩阵所对应的线性方程组的形式。我们完全可以利用第 二章第 7节中介绍的方法进行数值求解。

11 函数的近似与计算

¶ 本节我们讨论函数的近似与计算问题。我们将分为几类的函数的计算。一类是函数具有明确的表达式的计算问题。另一类是用某种函数近似后进行的数值计算。

11.1 级数表达的函数的计算

很多函数是以级数的形式表达的 (比如 Bessel 函数、Legendre 函数等等)。很多函数实际上也是这样去计算的。需要指出的是，尽管某些函数的级数展开在数学上是收敛的，但是直接按照级数去计算往往并不是最合适的方法。典型的例子如

$$\sin x = \sum_{k=1}^{\infty}(-)^{k-1}\frac{x^k}{k!}\ . \tag{3.36}$$

我们知道这个级数对于任意的 $|x|$ 都是收敛的，但是如果 $|x| \geq 1$，那么直接这么计算需要求和的项数是比较多的。事实上，一种更为有效的做法是首先将 x 化到第一或第四象限：$x \in [-\pi/2, \pi/2]$，然后我们可以利用公式：$\sin x = 3\sin(x/3) - 4\sin^3(x/3)$ 将其化为计算较小的角度 $(x/3)$ 的正弦问题。当然，这个过程可以进一步迭代直到 $(x/3)$ 足够地小以至于一两项级数就足以满足精度要求为止。当然更为现代一些的算法是利用我们后面提及的 Chebyshev 近似来进行计算。事实上，多数的超越函数都可以利用 Chebyshev 近似的方法进行计算。

11.2 函数的 Chebyshev 近似及其计算

Chebyshev 多项式是个好东东。它可以用来近似很多的函数。事实上，它在数值计算中是如此有用以至于我们通常的很多函数，甚至是一些初等的函数，都可以利用 Chebyshev 进行展开和计算。

对于任意的 $x \in [-1, 1]$，我们可以定义 n 阶的第一类 Chebyshev 多项式 $T_n(x)$，它可以明确地写为，[1]

$$T_n(x) = \cos(n \arccos x) \ . \tag{3.37}$$

对于最低的几个多项式，我们很容易写出它的明确表达式：

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_2(x) &= 2x^2 - 1 \\ T_3(x) &= 4x^3 - 3x \\ &\cdots \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \quad n \geq 1 \ . \end{aligned} \tag{3.38}$$

Chebyshev 多项式满足一系列重要的性质。其中一个比较重要的是它的正交归一性，

$$\int_{-1}^{1} \frac{T_i(x)T_j(x)}{\sqrt{1-x^2}} dx = \begin{cases} 0 & i \neq j \\ \pi/2 & i = j \neq 0 \\ \pi & i = j = 0 \end{cases} \tag{3.39}$$

另一个比较重要的性质是其零点以及极值点的位置。$T_n(x)$ 在 $[-1, 1]$ 之中恰好有 n 个零点，它们的位置由下式给出：

$$x_k = \cos\left(\frac{\pi(k+1/2)}{n}\right) \ , \quad k = 0, 1, \cdots, n-1 \ . \tag{3.40}$$

而它的极值点共有 $(n+1)$ 个，其位置为，

$$x_k = \cos\left(\frac{\pi k}{n}\right) \ , \quad k = 0, 1, \cdots, n \ . \tag{3.41}$$

[1] 符号 T 是因为 Chebyshev 有时候又曾译为 Tchebycheff。

在这些极值点处，Chebyshev 多项式恰好等于 +1(极大值) 或 −1(极小值)。也就是说，Chebyshev 多项式的绝对值总是在 1 以下的。正是这个特性使得它成为近似任意函数的有力工具，因为它可以很好地控制误差。

最后一个重要的性质是分立版本的正交归一关系。令 x_k 是由式 (3.40) 给出的 $T_n(x)$ 的 n 个零点，那么对于 $i,j<n$，我们有，

$$\sum_{k=0}^{m-1} T_i(x_k)T_j(x_k) = \begin{cases} 0 & i\neq j \\ n/2 & i=j\neq 0 \\ n & i=j=0 \end{cases} \tag{3.42}$$

利用上面给出的正交归一关系我们可以证明，对于任意的一个定义在 $[-1,1]$ 上面的函数 $f(x)$，我们可以构建 N 个系数 c_m，$m=0,1,\cdots,N-1$ 如下，

$$c_m = \frac{2}{N}\sum_{k=0}^{N-1} f\left[\cos\left(\frac{\pi(k+1/2)}{N}\right)\right]\cos\left(\frac{\pi m(k+1/2)}{N}\right) . \tag{3.43}$$

然后我们可以用下面的式子来近似表达目标函数 $f(x)$：[2]

$$f(x) \approx \frac{c_0}{2} + \sum_{k=1}^{N-1} c_k T_k(x) . \tag{3.44}$$

事实上，利用前面的表达式可以很容易证明，恰好在 $T_N(x)$ 的零点处这个表达式是严格成立的 (如果不计舍入误差的话)。即使不在这些零点处，这个表达式也给出了函数 $f(x)$ 的一个非常好的近似。

我们运用 Chebyshev 最多的时候并不是利用很大的 N 的时候，而是往往利用不那么大的时候。由于 Chebyshev 具有模永远小于 1 的特性，因此它可以很好地控制误差。例如，对于给定区间 $[-1,1]$ 上的任意函数 $f(x)$ 和给定的阶 N，我们可以试图寻找一个阶数不超过 N 的多项式 $p_N(x)$，使得它与函数 $f(x)$ 的差的绝对值的最大值取极小，即 $\max_{x\in[-1,1]}|f(x)-p_N(x)|$ 取极小。这样得到的多项式 $p_N(x)$ 称为给区间上到指定阶 N 的最小偏离多项式 (minimax polynomial)。要求出一个函数的最小偏离多项式并不是很简单的事情，而且计算也颇费周章。但是函数的 Chebeshef 多项式展开 (3.44) 提供了一个非常好的近似。换句话说，它非常接近于相应的最小偏离多项式，而且其计算也方便很多。由于截断到 N 阶的近似式的下一阶是 $c_N T_N(x)$ 而 $|T_N(x)|\leq 1$，因此误差主要由 $|c_N|$ 的大小来控制。一般来说随 N 的增加 c_N 衰减得非常快 (如果函数足够光滑的话，它们随着 N 的增加是指数衰减的)，所以我们往往仅仅需要几阶就够了。这正是 Chebyshev 强大的地方。

[2]大家也许觉得这些展开很像 Fourier 展开，对吧。其实它们就是。或者说从数学上证明这些结论我们就是需要利用 Fourier 展开的定理。这当然也不奇怪，因为 Chebyshev 多项式的定义 (3.37) 本身就是利用三角函数给出的。顺便指出的是，下面的展开式还有另外一种常用的写法：$f(x)\approx -\frac{c_0}{2}+\sum_{k=0}^{N-1} c_k T_k(x)$。由于其中对于 k 的求和从 $k=0$ 开始，而 $T_0(x)\equiv 1$，因此这个表达方式与 (3.44) 是完全一致的。

最后我们指出，虽然获得了 Chebeshev 近似之后我们可以把函数表达成标准的多项式的形式 –也就是说 $f(x)=\sum_{i=0}^{N}a_i x^i$ 的形式 –但是我们一般不建议这样做，除非你知道如何处理舍入误差。换句话说，从数值计算的角度而言，以 Chebyshev 多项式为基远比以通常的 x 的幂次为基要好。例如我们刚刚提到的，如果函数足够光滑，它的 Chebyshev 展开系数一般是指数衰减的。但是如果改用 x 的幂次来展开一般不是。

¶ 函数的 Chebyshev 展开如何计算呢？我们可以利用所谓的 Clenshaw 提出的一个方法。这个方法专门用于计算

$$f(x)\approx S_N(x)=\sum_{k=0}^{N}c_k F_k(x)\ , \tag{3.45}$$

类型的展开，只要基函数 $F_k(x)$ 同时满足一定的递推关系。Chebyshev 多项式显然属于这一类，参见公式 (3.38)。

Clenshaw 算法假设展开的基函数满足一定的递推关系，

$$F_{k+1}(x)=\alpha_k(x)F_k(x)+\beta_k(x)F_{k-1}(x)\ . \tag{3.46}$$

这涵盖了一大类的展开：Legendre 展开，Chebyshev 展开等等。假定我们已经知道了系数 c_k，$k=0,\cdots,N$，我们的目的是计算函数的 N 阶近似式 $S_N(x)$。我们重新构造一系列新的函数 $b_k(x)$，首先令：

$$b_{N+1}(x)=b_{N+2}(x)=0\ . \tag{3.47}$$

然后利用下面的递推关系向下迭代，

$$b_k(x)=c_k+\alpha_k(x)b_{k+1}(x)+\beta_{k+1}(x)b_{k+2}(x)\ . \tag{3.48}$$

其中的 $\alpha_k(x)$ 和 $\beta_k(x)$ 来自基函数所满足的递推关系 (3.46)。持续迭代直到我们获得 $b_1(x)$ 和 $b_2(x)$。最后我们发现，

$$S_N(x)=c_0F_0(x)+b_1(x)F_1(x)+\beta_1(x)F_0(x)b_2(x)\ . \tag{3.49}$$

多数情形下这个迭代通常情况下是稳定的。

对于 Chebyshev 的特殊情形我们有 $\alpha_k(x)=2x$，$\beta_k(x)=-1$。因此 Clenshaw 迭代的公式为，

$$b_k(x)=c_k+2xb_{k+1}(x)-b_{k+2}(x)\ , \tag{3.50}$$

而最终由公式 (3.44) 给出的近似和为，[3]

$$S_N(x)=\frac{1}{2}c_0+xb_1(x)-b_2(x)\ . \tag{3.51}$$

[3]注意下面公式中 c_0 前面额外的系数 $1/2$。这仅仅是由于一般的 Clenshaw 求和公式 (3.45) 与 Chebyshev 求和式 (3.44) 之间定义的不同造成的。一般的 Clenshaw 求和公式 (3.49) 中并没有这个 $1/2$。

11.3 函数的 Padé 近似及其计算

¶一个函数 $f(x)$ 如果我们知道了它的 Taylar 展开的部分系数，我们还可以利用一个分式来近似地表示这个函数。如果我们令，

$$R(x) = \frac{\sum_{k=0}^{M} a_k x^k}{1 + \sum_{k=1}^{N} b_k x^k} \,. \tag{3.52}$$

我们期待它可以近似

$$f(x) = \sum_{k=0}^{\infty} c_k x^k \,. \tag{3.53}$$

我们要求，

$$R^{(k)}(x)|_{x=0} = f^{(k)}(x)|_{x=0} \,, \quad k = 0, 1, \cdots, M+N \tag{3.54}$$

并称 $R(x)$ 是函数 $f(x)$ 的一个 (M, N) 阶的 Padé 近似式。

假定所有的 Taylar 展开系数 c_k 已知，那么最为简单的计算 Padé 近似式的各个系数 a_k 和 b_k 的方法是令 $f(x) = R(x)$ 并将分母乘到等式的左边，然后比较两边 x 同幂次的系数。在假定 $M = N$ 的情形下，这给出如下的方程，

$$\begin{aligned} \sum_{m=1}^{N} b_m c_{N-m+k} &= -c_{N+k} \,, \quad k = 1, \cdots, N \,, \\ \sum_{m=0}^{k} b_m c_{k-m} &= a_k \,, \quad k = 1, \cdots, N \,, \end{aligned} \tag{3.55}$$

上式中的前一个方程可以视为 $(b_1, \cdots, b_N)$ 的一个线性方程，求解这个方程我们就可以获得系数 $(b_1, \cdots, b_N)$，再带入第二个方程就可以获得系数 $(a_1, \cdots, a_N)$。另一方面，我们必须有 $a_0 = c_0$。显然，要能够确定解出所有的系数，我们需要首先知道 c_k，$k = 0, 1, \cdots, 2N$ 这 $(2N+1)$ 个展开系数。

Padé 近似最经常的应用是为一个比较短的级数展开式“续命”。比如说我们在一个复杂的微扰计算中获得了某个函数的 Taylor 展开中的前 5 项，进行后续的展开实在是太过繁琐。但是有时候我们又希望将这个展开应用到比较大的 x 的区域。我们就可以利用 Padé 近似来达到这个“偷懒”的目的。一般来说，Padé 近似的优点和缺点是并存的。优点是，它可能会工作得很好，甚至出奇的好；当然，它也可能完全不行，甚至在相当多的情况下误导我们的判断。换句话说，它是一个不太靠谱的近似。但是它一般不会比你原先的 Taylor 展开式更差。从这个角度来说，如果你并不想继续计算下一阶的系数，那么你没有失去什么。当然，你也没有百分之百得到什么，因为 Padé 给出的结果并不可靠。

Numerical Recepies 给出了一个十分有趣的例子 [1]。假定经过复杂的解析计算 –比如说通过量子场论中一个复杂过程的大量四圈以下的费曼图的计算 –你获得了一个重要的物

理量 $f(x)$ 作为一个小的耦合参数 x 的微扰展开式：[4]

$$f(x) \approx 2 + \frac{1}{9}x + \frac{1}{81}x^2 - \frac{49}{8748}x^3 + \frac{175}{78732}x^4 + \cdots . \tag{3.56}$$

它实际上是你感兴趣的物理量 $f(x)$ 在 $x = 0$ 处的 Taylor 展开的前 5 项。在 x 比较小的时候 –比如说 $x \lesssim 1$ 的时候 –这个表达式被大量的实验证明是相当精确的。但是假定我们希望研究 $x \gtrsim 1$ 时的 $f(x)$ 的情况。当然，我们可以试图去计算下一阶的贡献。这意味着需要计算一系列的五圈图，而这注定是一个无法完成的任务，因为需要计算的费曼图的数目和计算量随着圈数的增加是指数增长的。事实上，上面的那个展开式恰好是下列函数的 Taylor 展开：

$$f(x) = [7 + (1 + x)^{4/3}]^{1/3} . \tag{3.57}$$

假如 x 的绝对值不是很大，那么上面的展开式 (3.56) 与严格的函数 (3.57) 本身的差别很小。事实上，如果你把这两个函数都画出来的话，[5] 你会发现大约从 $x \gtrsim 2$ 起两者的差别才变得明显。但是如果你并不知道函数的严格表达式 (3.57)，同时你有没有能力去计算下一阶的贡献，这时候你不妨试试 Padé 近似。仅仅使用已知的 5 个系数来构建相应的 Padé 近似式并与函数进行比较后我们会发现，即使到大约 $x \approx 10$，Padé 近似式与严格函数的差别都是可忽略的，而对于相应的 Taylor 展开，如果 x 超过 2 就已经相当明显地偏离了。在这个例子中，Padé 近似几乎就是一个奇迹。

12 数值微分的计算

¶本节我们讨论导数的计算问题。数值上计算函数的导数实际上主要从导数的定义出发。

$$f'(x) \simeq \frac{f(x+h) - f(x)}{h} , \quad h \to 0 . \tag{3.58}$$

也就是说，近似计算一个点 x 处的导数，我们需要计算两次函数。事实上，这并不是最佳的计算方法。对于比较光滑的函数来说，同样是计算 x 处的导数，我们宁愿计算，

$$f'(x) \simeq \frac{f(x+h) - f(x-h)}{2h} , \quad h \to 0 . \tag{3.59}$$

对于同样大小的 h，这个方法比起上面的拿衣服的方法要好很多。它的截断误差是正比于 h^2 而不是 h 的。另一方面，计算还有舍入误差。事实上，如果真的利用这种方法来计算导数，我们可以选择一些优化的间隔 h 以尽可能地减少舍入误差和截断误差。更加详细的讨论请参考 Numerical Recipes 的第 5.7 节中的讨论。

另外一种更好的方法是利用函数本身的 Chebyshev 近似式。正如我们上节所说，如果我们有，

$$f(x) \approx \sum_{k=0}^{N} c_k T_k(x) , \tag{3.60}$$

[4]这个例子与 Numerical Recipes 中的例子完全一致。当然，上面提及的量子场论的 4 圈图是我自己杜撰的，呵呵。这样使得读者会感觉得更真实一些。

[5]参见 Numerical Recipes [1] 中的 §5.12 中的图 5.12.1。

那么我们同样可以获得 $f'(x)$ 的 Chebyshev 展开式，

$$f'(x) \approx \sum_{k=0}^{N} c'_k T_k(x) \,, \tag{3.61}$$

其中的系数 c'_k 与原先的系数之间的关系为，

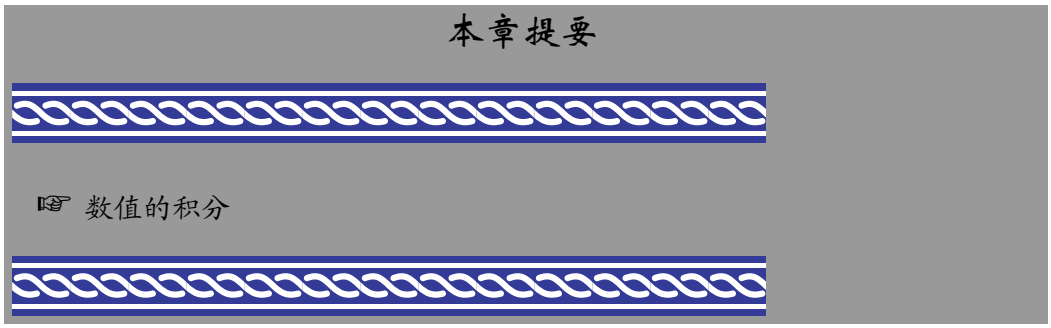
$$\begin{aligned} c'_N &= c'_{N-1} = 0 \,, \\ c'_{k-1} &= c'_{k+1} + 2kc_k \,, \quad (k = N-1, N-2, \cdots, 1) \end{aligned} \tag{3.62}$$

事实上，有了函数的 Chebyshev 近似式，除了可以计算它的导数 (微分) 之外，还可以计算其积分。这就获得了一个近似的积分表达式，而这是我们下一章中需要讨论的内容。

相关的阅读

这一章我们首先讨论了函数的各种内插问题。随后讨论了函数的近似问题，重点放在了函数的 Chebyshev 展开式。另外一个相关的内插问题是三角函数的内插问题。这实际上涉及著名的快速傅里叶变换的问题。由于这个问题的特殊性，我们将在后面单独用一章来讨论。

第四章　　数值积分



本章中我们讨论利用数值方法进行函数的积分。利用数值方法计算定积分是很多物理分支中必须面对的课题。这一章中我们将仅仅涉及普通的单元函数函数在一维的定积分。对于多元函数在多维空间的积分，一般采用 Monte Carlo 方法更为合适。

13　等间距的数值积分公式:Newton-Cortes

¶我们首先来考察数值积分的函数计算点是等间距的情形。假定我们希望计算积分，

$$I = \int_a^b f(x)dx \ . \tag{4.1}$$

通常的做法 –实际上也是积分的最初数学定义 –是将有限的区间 $[a,b]$ 等分为 N 份，每一份的长度为 $h = (b-a)/N$。我们令，

$$x_0 \equiv a, x_1 = x_0 + h, \cdots, x_N = x_0 + Nh = b \ , \tag{4.2}$$

并且计算各个点的函数值，

$$f(x_i) = f_i \ , \quad i = 0, 1, \cdots, N \ . \tag{4.3}$$

那么我们可以构造最基本的所谓梯形法则来近似函数的积分，

$$I \approx h \left[\frac{1}{2} f_0 + f_1 + \cdots + f_{N-1} + \frac{1}{2} f_N \right] \ , \tag{4.4}$$

更为普遍地讨论这类积分问题可以从我们前面讨论的多项式内插出发。我们认定 $a = x_0 < x_0 + h < \cdots < x_N = b$ 构成了区间 $[a, b]$ 的一个分割。我们知道我们可以利用 Lagrange 内插公式，

$$P_N(x) = \sum_{i=0}^{N} f_i L_i(x) \ , \tag{4.5}$$

这个多项式经过所有的 $N+1$ 个点 (x_i, f_i)。利用这个内插公式计算积分话，我们得到，

$$\int_a^b P_N(x)dx = \sum_{i=0}^{N} f_i \int_a^b L_i(x)dx \ . \tag{4.6}$$

现在我们令

$$x = a + ht \ , \ \ t \in [0, N] \ , \tag{4.7}$$

同时令

$$L_i(x) \equiv \phi_i(t) = \prod_{k=0, k\neq i}^{N} \frac{t-k}{i-k} \ . \tag{4.8}$$

于是我们就得到，

$$\int_a^b P_N(x)dx = h \sum_{i=0}^{N} f_i \alpha_i \ , \tag{4.9}$$

其中的权重 α_i 的定义为，

$$\alpha_i = \int_0^N \phi_i(t)dt \ . \tag{4.10}$$

作为一个例子我们取 $N = 2$，我们有，

$$\alpha_0 = \int_0^2 \frac{t-1}{0-1}\frac{t-2}{0-2}dt = \frac{1}{3} \ , \tag{4.11}$$

以及类似的计算给出 $\alpha_1 = 4/3$，$\alpha_2 = 1/3$。于是我们得到著名的 Simpson 规则 (或者称为 Simpson 公式)，

$$\int_a^b P_2(x)dx = \frac{h}{3}(f_0 + 4f_1 + f_2) \ . \tag{4.12}$$

对于任意的 N 点的公式，一般称为 Newton-Cortes 公式。它的形式为，

$$\int_a^b P_N(x)dx = h \sum_{i=0}^{N} f_i \alpha_i \ . \tag{4.13}$$

其中的权重 α_i 由式 (4.10) 所定义。由于它们源于有理系数的多项式在 $[0, N]$ 上的积分，因此它们必定都是有理数。通常将它们写为分数的形式。它们还满足一个约束条件，

$$\sum_{i=0}^{N} \alpha_i = N \ . \tag{4.14}$$

这可以在上面的 Newton-Cortes 公式中令 $f(x)\equiv 1$ (从而 $P_N(x)\equiv 1$ 并且 $f_i\equiv 1$) 获得。我们可以将其通分，令 $\alpha_i=\sigma_i/s$，其中的 σ_i 和 s 都是正整数，s 是各个 α_i 共同的分母，这样一来公式可以写成，

$$\int_a^b P_N(x)dx=\frac{b-a}{Ns}\sum_{i=0}^{N} f_i\sigma_i\ . \tag{4.15}$$

由于上面提及的对于 α_i 的约束，我们知道 $\sum_i(\sigma_i/Ns)=1$，这一点也可以从表 4.1中看到。

由多项式内插所造成的误差可以按照下式估计：

$$\int_a^b [P_N(x)-f(x)]dx=h^{p+1}\cdot K\cdot f^{(p)}(\xi)\ ,\quad \xi\in(a,b)\ , \tag{4.16}$$

其中的 p 和 K 仅仅依赖于 N，与被积函数 f 无关。

表 4.1: Newton-Cortes 积分表达式对于 $N=1,2,3,4,5,6$ 的情形。对于 $N>6$ 的更高阶数值积分公式，由于各个 σ_i 会正负相抵，因此并不能有效地改进计算效率。

N	σ_i	Ns	误差估计	名称
1	1 1	2	$h^3(1/12)f^{(2)}(\xi)$	梯形法则
2	1 4 1	6	$h^5(1/90)f^{(4)}(\xi)$	Simpson 规则
3	1 3 3 1	8	$h^5(3/80)f^{(4)}(\xi)$	3/8 规则
4	7 32 12 32 7	90	$h^7(8/945)f^{(6)}(\xi)$	Milne 规则
5	19 75 50 50 75 19	288	$h^7(275/12096)f^{(6)}(\xi)$	
6	41 216 27 272 27 216 41	840	$h^9(9/1400)f^{(8)}(\xi)$	Weddle 规则

对于 $N\le 6$ 的各个情况我们列在了表 4.1中。对于 $N>6$ 的积分公式，由于各个权重 α_i 会出现正负相消的情况，因此并不能有效地应用到数值积分之中。具体来说，对于 $N=1$ 我们获得的是所谓的梯形法则 (trapezoidal rule)。$N=2$ 的则是 Simpson 法则；$N=3$ 的称为 3/8 规则等等。

在真实计算数值积分时，我们往往并不是将上述规则直接运用到待积分的整个区间，而是首先将积分区间分为若干个小段，然后分别在各个小段上运用这些积分公式。换句话说，我们经常使用的是所谓的延展的积分规则，例如延展的梯形规则、延展的 Simpson 规则等等。

14 外推积分方法

¶前一节给出的积分公式中关于误差的估计实际上是一般的 Euler-Maclaurin 公式的特殊情形。下面我们给出一般的公式并且说明如何利用这个公式给出一系列利用外推的积分方法。

定理 4.1 对于任意正整数 m，假定函数 $g \in C^{2m+2}[0,1]$，那么我们有，

$$\int_0^1 g(t)dt = \frac{g(0)}{2} + \frac{g(1)}{2} + \sum_{k=1}^{m} \frac{B_{2k}}{(2k)!}\left[g^{(2k-1)}(0) - g^{(2k-1)}(1)\right] - \frac{B_{2m+2}}{(2m+2)!} g^{(2m+2)}(\xi)\,, \quad \xi \in (0,1)\,. \tag{4.17}$$

其中的 B_{2k} 是所谓的 Bernoulli 数，它的定义由下式给出：

$$\frac{z}{e^z - 1} = \sum_{n=0}^{\infty} B_n \frac{z^n}{n!}\,. \tag{4.18}$$

我们需要的是将上述公式复制 N 份之后的结果，

$$\int_0^N g(t)dt = \frac{g(0)}{2} + g(1) + \cdots + \frac{g(N)}{2} + \sum_{k=1}^{m} \frac{B_{2k}}{(2k)!}\left[g^{(2k-1)}(0) - g^{(2k-1)}(N)\right] - \frac{B_{2m+2}}{(2m+2)!} g^{(2m+2)}(\xi)\,, \quad \xi \in (0,N)\,. \tag{4.19}$$

这里我们假定 $g \in C^{(2m+2)}[0,N]$。将这个公式运用到任意的区间 $[a,b]$ 上的积分，如果我们假定中间函数计算的点是均匀分布的，从而积分的步长

$$h = \frac{b-a}{N}\,. \tag{4.20}$$

那么 Euler-Maclaurin 公式给出，

$$T(h) = \int_a^b f(t)dt + \sum_{k=1}^{m} \frac{B_{2k}h^{2k}}{(2k)!}\left[f^{(2k-1)}(0) - f^{(2k-1)}(N)\right] - \frac{B_{2m+2}h^{2m+2}}{(2m+2)!} f^{(2m+2)}(\xi)\,, \quad \xi \in (a,b)\,, \tag{4.21}$$

其中 $T(h)$ 代表梯形法则给出的近似式：

$$T(h) = h\left[\frac{f(a)}{2} + f(a+h) + \cdots + f(b-h) + \frac{f(b)}{2}\right]\,, \tag{4.22}$$

而我们假定 $f \in C^{(2m+2)}[a,b]$。需要指出的是，上面给出的这个公式一般并不是收敛的级数，而是一个渐近展开 (asymptotic expansion)。主要是因为 Bernoulli 数 B_{2k} 随着 k 的增大可以变得很大。[1] 虽然这个展开式不是收敛的级数，但是它已经足够用于发展外推积分算法了。

¶ 我们发现，作为步长 h 的函数，近似式 $T(h)$ 可以有如下的渐近展开，

$$T(h) = \tau_0 + \tau_1 h^2 + \cdots + \tau_m h^{2m} + \alpha_{m+1}(h)h^{2m+2}\,. \tag{4.23}$$

[1] 如果不相信，你可以试着利用 Mathematica 计算一下 B_{60}。事实上，$|B_{2k}| \sim 4\sqrt{\pi k}(k/(\pi e))^{2k}$，对于 $k \gg 1$。因此粗略来说，对于大的 k，我们有 $B_{2k} \sim k^{2k+1/2} \to \infty$。

其中按照我们的假设，如果 $f \in C^{(2m+2)}[a,b]$，那么系数 $\alpha_{m+1}(h)$ 在 h 足够小的时候是有界的并且它对于 h 的依赖会趋于零。这说明 $T(h)$ 可以展开成一个 h^2 的多项式，这提示我们可以对 $T(h)$ 进行多次计算，每次运用不同的 h，然后再进行多项式的内插/外推。我们需要的积分数值就由参数 τ_0 给出，

$$\tau_0 = \int_a^b dx f(x) \, . \tag{4.24}$$

一般来说，我们会选取一系列严格递增的整数系列：$1 = n_0 < n_1 < n_2 \cdots < n_m$ 并且令相应的积分步长为，

$$h_i = \frac{b-a}{n_i} \, . \tag{4.25}$$

我们看到 $h_0 = (b-a)$。一个经常使用的同时也是比较自然的序列是 $n_i = 2^i$。这是 Romberg 早先的选择。当然其他的选择也是可以的。然后我们令：

$$T_{i0} \equiv T(h_i) \, , \quad i = 0, 1, \cdots, m \, . \tag{4.26}$$

同时我们将内插的 m 阶多项式记为 $\tilde{T}_{mm}(h)$，它满足

$$\tilde{T}_{mm}(h_i) = T_{i0} = T(h_i) \, . \tag{4.27}$$

我们将用 $\tilde{T}_{mm}(0)$ 作为 τ_0 的一个近似值。于是我们可以利用第 8节中讨论过的 Neville 算法 4，只不过现在 $x_i = h_i^2$ 并且我们最后需要外推到 $x = h^2 = 0$。

对于 $1 \le k \le i \le m$，将 $\tilde{T}_{ik}(h)$ 是关于 h^2 的最高阶为 k 的内插多项式，并且满足：

$$\tilde{T}_{ik}(h_j) = T(h_j) \, , \quad j = i-k, i-k+1, \cdots, i \, , \tag{4.28}$$

并且我们将 $\tilde{T}_{ik}(0)$ 简记为 T_{ik}。我们有如下的 Neville 迭代，

$$T_{ik} = T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{h_{i-k}^2/h_i^2 - 1} \, . \tag{4.29}$$

这个式子可以从 T_{00} 开始一直往下迭代，直到最后获得 T_{mm}，也就是到 m 阶的积分近似值。这个可以安排成一个三角形的图：

$$\begin{array}{c|cccc}
h_0^2 & T(h_0) = T_{00} & & & \\
 & & T_{11} & & \\
h_1^2 & T(h_1) = T_{10} & & T_{22} & \\
 & & T_{21} & & T_{33} \\
h_2^2 & T(h_2) = T_{20} & & T_{32} & \vdots \\
 & & T_{31} & \vdots & \\
h_3^2 & T(h_3) = T_{30} & \vdots & & \\
\vdots & \vdots & & &
\end{array} \tag{4.30}$$

图中的第一列就是按照相应的步长计算的梯形近似的值 $T(h_i)$。往右的每一列中的元素都是由其左侧的上下两个相邻的元素按照迭代式 (4.29) 给出。最后的结果 (这里显示的是 T_{33}) 就是到相应阶的积分的近似值。

15 利用正交多项式的高斯积分法

¶ 正如我们前面提及的，如果一个函数可以用正交多项式来近似展开，我们还可以利用正交多项式进行积分。

前面我们已经介绍了一种最为常用的正交多项式，及所谓的 Chebyshev 多项式。事实上，还有其他的类型。我们这里做一个稍微一般一些的讨论。我们讨论型如

$$I = \int_a^b dx\omega(x)f(x) \,, \tag{4.31}$$

的积分，其中的函数 $\omega(x) \geq 0$ 是区间 $[a,b]$ 上一个非负的函数，我们称之为权重函数，它满足如下的条件：

- 非负性 $\omega(x) \geq 0$，$\forall x \in [a,b]$；
- 权重函数的各阶矩存在: $\mu_k = \int_a^b x^k\omega(x)dx$，$k = 0, 1, \cdots$；
- 对于任意的非负的多项式 $s(x)$ 来说，$\int_a^b s(x)\omega(x)dx = 0$ 必定给出 $s(x) = 0$。

我们下面将假设上面的三点对于权重函数都是成立的。在此前提下考虑积分 (4.31) 的计算问题。假定区间 $[a,b]$ 上定义了非负权重函数为 $\omega(x)$。我们定义在 $[a,b]$ 上最高阶系数为 1 的 j 阶多项式为 $\bar{\Pi}_j$，即 $\bar{\Pi}_j = \{p|p(x) = x^j + a_1x^{j-1} + \cdots + a_j\}$。同时，我们定义阶数不大于 n 的所有多项式的集合为 $\Pi_n = \{p|\text{degree}(p) \leq n\}$。

¶ 我们可以在区间 $[a,b]$ 上定义两个函数的内积：

$$(f,g) = \int_a^b \omega(x)f(x)g(x)dx \,, \tag{4.32}$$

显然由于 $\omega(x)$ 的非负性，我们有 $(f,f) = \int_a^b \omega(x)f(x)^2dx \geq 0$。因此，在这个内积下可以定义正交的概念。具体来说，如果 $(f,g) = 0$，我们就称两个函数 (在区间 $[a,b]$ 上) 正交。于是，在定义在 $[a,b]$ 上的各阶多项式中，我们可以构建一系列正交的多项式。这体现在如下的定理之中。

定理 4.2 假定区间 $[a,b]$ 上定义的非负权重函数为 $\omega(x)$。定义在 $[a,b]$ 上最高阶系数为 1 的 j 阶多项式为 $\bar{\Pi}_j$，即 $\bar{\Pi}_j = \{p|p(x) = x^j + a_1x^{j-1} + \cdots + a_j\}$。那么下面的结论成立：

1. 存在一系列的多项式 $p_n \in \bar{\Pi}_n$，$n = 0, 1, \cdots$，它们满足正交性关系：

$$(p_i, p_j) = 0 \,, \quad i \neq j \,. \tag{4.33}$$

2. 这些正交多项式 p_n 可以由下列递推关系给出：

$$\begin{aligned} p_{-1} \equiv 0 \,, \quad p_0(x) \equiv 1 \,, \\ p_{i+1}(x) = (x - \delta_{i+1})p_i(x) - \gamma_{i+1}^2 p_{i-1}(x) \,, \; i \geq 0 \,. \end{aligned} \tag{4.34}$$

其中的系数 γ_i^2 和 δ_i 由下式给出：

$$\begin{aligned}\delta_{i+1} &= (xp_i, p_i)/(p_i, p_i)\ , i = 0, 1, \cdots\ , \\ \gamma_{i+1}^2 &= (p_i, p_i)/(p_{i-1}, p_{i-1})\ , \gamma_1^2 = 0\ , \quad i = 1, 2, \cdots\end{aligned} \tag{4.35}$$

这里的正交化方法又称为 Gram-Schmidt 正交化方法。

3. 多项式 $p_n(x)$ 的 n 个根都是实数单根并且都位于 $[a, b]$ 之间。

4. 对于 n 个两两不同的宗量 t_i，$i = 0, 1, \cdots, n-1$，下列系数矩阵是非奇异的。

$$A = \begin{bmatrix} p_0(t_0) & \cdots & p_0(t_{n-1}) \\ \vdots & \ldots & \vdots \\ p_{n-1}(t_0) & \cdots & p_{n-1}(t_{n-1}) \end{bmatrix}\ . \tag{4.36}$$

根据这个定理，我们可以构造一个多项式

$$p(x) = \sum_{i=0}^{n-1} c_i p_i(x)\ , \tag{4.37}$$

并且要求 $p(t_i) = f_i$，$i = 0, 1, \cdots, (n-1)$，那么这个内插问题一定有唯一解 $c \in \mathbb{R}^n$。这个条件有时候又称为 Haar 条件。满足 Haar 条件的一系列多项式 p_n 被称为一个 Chebyshev 系统。显然，我们前面提及的 Chebyshev 多项式构成了一个 Chebyshev 系统。

下面我们进一步引入一个定理。

定理 4.3 令 $x_1, \cdots, x_n$ 为 $p_n(x)$ 的 n 个根。同时 $w_1, \cdots, w_n$ 为下列线性方程的解，

$$\sum_{i=1}^{n} p_k(x_i) w_i = \begin{cases} (p_0, p_0) & k = 0 \\ 0 & k = 1, 2, \cdots, n-1 \end{cases} \tag{4.38}$$

那么我们一定有 $w_i > 0$，$i = 1, \cdots, n$，并且对于阶数不大于 $(2n-1)$ 的任意多项式 $p(x) \in \Pi_{2n-1}$ 来说，我们一定有如下的关系：

$$\int_a^b \omega(x) p(x) dx = \sum_{i=1}^{n} w_i p(x_i)\ . \tag{4.39}$$

这些正的数 $w_i > 0$ 被称为相应多项式的权重因子。进一步，对于任意的 $f \in C^{2n}[a, b]$，我们一定可以找到一个 $\xi \in [a, b]$ 使得：

$$\int_a^b \omega(x) f(x) dx - \sum_{i=1}^{n} w_i f(x_i) = \frac{f^{(2n)}(\xi)}{(2n)!}(p_n, p_n)\ . \tag{4.40}$$

如果我们将前面递推关系 (4.34) 中的系数 δ_i，γ_i 等排成如下的 $n \times n$ 三对角矩阵，

$$J_n = \begin{bmatrix} \delta_1 & \gamma_2 & & & \\ \gamma_2 & \ddots & & & \\ & & \ddots & & \\ & & \ddots & \ddots & \gamma_n \\ & & & \gamma_n & \delta_n \end{bmatrix}\ , \tag{4.41}$$

那么 $p_n(x)$ 的根 x_i 恰好是 J_n 的本征值；如果假定相应的本征矢量记为 $v^{(i)} \in \mathbb{R}^n$ 并且按照如下方式归一，

$$v^{(i)T} v^{(i)} = (p_0, p_0) = \int_a^b \omega(x) dx \ , \tag{4.42}$$

那么权重因子 w_i 由下式给出：

$$w_i = \left(v_1^{(i)}\right)^2 \ , \quad i = 1, 2, \cdots, n \ . \tag{4.43}$$

高斯积分公式的神奇之处在于，对一个 n 点的高斯积分公式而言，如果函数是一个低于 $2n$ 阶的多项式，那么高斯积分公式是严格成立的，没有任何的误差 (除去计算机的舍入误差之外)，只要我们选择了正确的节点 x_i 和权重因子 w_i；即使对于不是多项式的任意的光滑函数 $f(x)$，公式 (4.40) 也给出了一个对误差估计。它与前面的 Newton-Cortes 公式的最大区别在于，节点并不一定是均匀分布的。

¶ 作为最为典型的正交多项式，如果我们选取区间 $[-1, +1]$ 和 $\omega(x) = 1$，我们就有 Legendre 多项式：

$$P_n(x) = \frac{n!}{(2n)!} \frac{d^n}{dx^n} \left[x^2 - 1\right]^n \tag{4.44}$$

这个定义与大家所熟悉的 Legendre 多项式之差一个常数因子。这个定义保证了多项式的最高级系数为 1. 这时的积分公式称为 Gauss-Legendre 积分公式 (Gauss-Legendre quadrature)。其他比较重要的特例还有：Jacobi 多项式和 Gauss-Jacobi 积分公式；Chebyshev 多项式和 Gauss-Chebyshev 积分公式；Hermite 多项式和 Gauss-Hermite 积分公式；Laguerre 多项式和 Gauss-Laguerre 积分公式等。这些多项式所对应的区间 $[a, b]$ 以及相应权函数 $\omega(x)$ 我们列在表 4.2中。

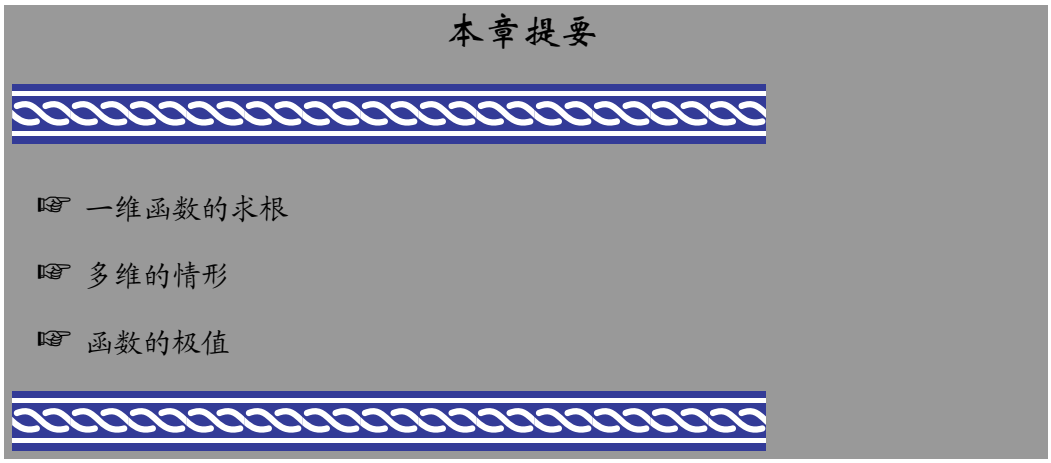
表 4.2: 一些经常使用的正交多项式。

区间 $[a, b]$	$\omega(x)$	多项式名称
$[-1, 1]$	1	Legendre 多项式 $P_n(x)$
$[-1, 1]$	$(1-x)^\alpha (1+x)^\beta$	Jacobi 多项式 $P_n^{(\alpha,\beta)}(x)$
$[-1, 1]$	$\frac{1}{\sqrt{1-x^2}}$	Chebyshev 多项式 $T_n(x)$
$[0, \infty)$	e^{-x}	Laguerre 多项式 $L_n(x)$
$(-\infty, +\infty)$	e^{-x^2}	Hermite 多项式 $H_n(x)$

相关的阅读

这是有点长的一章哦。

第五章　方程求根与函数求极值



本章中我们讨论经典的方程求根以及函数极值的数值计算方法。这两个问题往往联系在一起是因为如果我们需要寻找的函数是连续可微的函数，并且它的导数可以方便地计算，那么极值等价于函数的导数等于零。

我们可能需要求实函数 $f : E \to F$ 的零点：$\{\xi \in E | f(\xi) = 0\}$，其中 E 表示函数的定义域而 F 则表示它的值域。最为简单的情形是 $E = F = \mathbb{R}^1$。稍微推广一些的情形是 $E = F = \mathbb{R}^n$，这时我们要求解的是一系列联立的非线性方程。

一般来说，这类问题中的函数都不是线性函数，否则问题就化为前一章讨论的线性方程的求解问题了。所以，我们往往不可能有具体的“公式”来套用，而必须从某个猜测的出发点 x_0 出发，利用迭代的方法逐步逼近最终的解。一般来说，我们的迭代往往具有 $x_{i+1} = \Phi(x_i)$ 的形式，其中 $\Phi(x)$ 称为迭代函数，显然它必须选择得使得待求的根满足：$\xi = \Phi(\xi)$，即待求的根是迭代函数的不动点。

16　对分法求根

¶ 首先考虑一个一维实函数 $f : \mathbb{R} \to \mathbb{R}$ 的求根问题。如果我们知道该函数在某个区域 $[a, b]$ 内连续并且一定有根 $\xi \in [a, b]$ 存在，例如 $f(a)$ 与 $f(b)$ 异号，那么最为直接的求根方法就是对分法。这个算法的的好处是，它虽然不一定是最快的，但是它是最安全的，几乎

不会失败。因此，如果求根过程本身并不耗费太多的时间，那么这个方法是值得推荐的。这种求根的方法的另一个好处是，它仅仅需要计算函数的值，并不需要计算其导数的数值。这对于一些函数值已知，但是其导数值不易求出的情形是特别有用的。对分法还有一个好处就是它对于根的精度有绝对的控制。由于我们总是可以确信至少一个根存在于一个固定测度的线段之内，而这个线段的测度随着迭代的次数是指数减小的，因此所得到的根的精度是绝对有保障的。事实上如果考察迭代过程中尝试根 x_i 与真实根 ξ 之间的差别：$\epsilon_i \equiv x_i - \xi$，那么对于对分法我们一定有：

$$|x_{i+1} - \xi| \simeq |x_i - \xi|/2 \,. \tag{5.1}$$

在迭代算法中，这种收敛速度被称为线性收敛 (linearly convergent)。具体来说，如果

$$\epsilon_{i+1} \sim (\epsilon_i)^m \,, \tag{5.2}$$

我们就称该算法是 m 幂次收敛 的。线性收敛恰恰是 $m = 1$ 的情形。我们下面会看到一个平方收敛的算法 (即 $m = 2$ 的情形)。注意，误差 $|\epsilon_i|$ 的大小随着 i 的增加实际上是指数趋于零的。不过在算法上这一般被称为线性收敛。这个称呼符合其所需要的精度 (也就是有效数字的位数) 正好线性依赖于迭代的次数（也就是计算时间）。因此，对于线性收敛的算法来说，每提高一位精度，就意味着要多消耗一定量的计算时间。

下面的步骤就是标准的对分法求根的基本操作：

Algorithm 5 对分法求根

Require: 函数 $f(x)$ 在区间 $[a, b]$ 内连续且在两个端点异号，即：$f(a)f(b) < 0$，因此 $f(x)$ 在该区间内一定有根存在。

【计算量】：每次迭代，计算 $f(x)$ 一次。

1: **while** 目前根的精度还不满意 **do**
2: 取 $x_0 = (a + b)/2$ 并计算 $f(x_0)$；
3: 比较 $f(x_0)$ 与 $f(a)$ 或 $f(b)$，并用 x_0 替代与 $f(a)$ 或 $f(b)$ 同号的那个宗量。即：若 $f(x_0)f(a) > 0$，则令 $a \Rightarrow x_0$；若 $f(x_0)f(b) > 0$，则令 $b \Rightarrow x_0$；
4: **end while**
5: 输出 x_0 作为根。

17 Newton-Ralphson 方法及其推广

17.1 一维的情形

¶ 仍然考虑一个一维实函数 $f : \mathbb{R} \to \mathbb{R}$ 的求根问题。设该函数有一个根 ξ，即 $f(\xi) = 0$。我们假定 f 在所寻找的根 ξ 附近的邻域内足够光滑。于是，选择足够接近根 ξ 的另一个点 x_0 作为我们的出发点，我们有：

$$f(\xi) = 0 = f(x_0) + f'(x_0)(\xi - x_0) + \cdots \,. \tag{5.3}$$

如果我们忽略掉后面的高阶项，这等价于在所寻找的根附近对函数 f 运用线性近似，我们得到：

$$\xi \simeq x_0 - \frac{f(x_0)}{f'(x_0)} , \tag{5.4}$$

这实际上就是著名的 Newton-Ralphson 求根方法。它的具体的实现为：

Algorithm 6 Newton-Ralphson 方法求根

Require: 首先在需要寻找跟的附近找一个出发点 x_0；

【计算量】：每次迭代，计算 $f(x)$ 及其导数 $f'(x)$ 各一次。

1: **while** $f(x_i)$ 不足够接近于零 **do**

2: 令 $i = 0, 1, 2, \cdots$，进行下列迭代

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} , \quad i = 0, 1, 2, \cdots , \tag{5.5}$$

3: 计算 $f(x_{i+1})$，检查它是否足够小；

4: **end while**

5: 输出最终的 x_i 作为根。

注意，每次迭代中所求的近似根的改变量为,

$$\Delta x_i \equiv x_{i+1} - x_i = \frac{f(x_i)}{f'(x_i)} , \tag{5.6}$$

因此，取决于我们所要求的根的精度，我们会在某一步终止迭代并输出结果。我们看到，在线性的牛顿方法的每一步迭代中除了需要计算一次函数值 $f(x_i)$ 之外，还需要计算一次函数的导数的值 $f'(x_i)$。这是与对分法的最大区别。

当然，我们也可以在泰勒展开 (5.10) 中去更多的项，这样就对应于在 ξ 附近用更高级的多项式来近似函数。由于三次及以上的方程求根的公式本身就比较繁琐，因此这个方法很少推广到高于二次的情形。对于二次的情形，与迭代 (5.5) 对应的为，

$$x_{i+1} = x_i - \frac{f(x_i) \pm \sqrt{f'(x_i)^2 - 2f(x_i)f''(x_i)}}{f''(x_i)} , \quad i = 0, 1, 2, \cdots , \tag{5.7}$$

当然，与线性牛顿方法的迭代比较，这个算法还需要在每一步迭代中额外、再计算函数的二阶导数 $f''(x_i)$ 一次。

如果我们有函数的明显表达式并且其导数的计算也不复杂，一般来说 Newton-Ralphson 求根方法比对分法要快速。事实上，在真实根 ξ 的附近，可以证明在每次迭代后尝试根与真实根之间的差别是平方收敛 (quadratically convergent) 的，即如果我们令 $\epsilon_i \equiv x_i - \xi$，那么：

$$\epsilon_{i+1} \simeq -(\epsilon_i^2)\frac{f''(\xi)}{2f'(\xi)} , \tag{5.8}$$

也就是说，Newton-Ralphson 方法是一个平方收敛的算法。换句话说，在函数的根的附近，若在某个固定迭代步骤内牛顿迭代可以改善结果两位有效数字，那么传统的对分法在相同的迭代次数中则只能改进一位。但是，这个方法有一个潜在的弱点，那就是它可能会失败。其中最为典型的例子是初始选择的邻域中恰好包含了函数的极值点。在极值点附近函数的导数为零，因此迭代中每一步的修正 $|\Delta x_i|$ 非常大。在迭代中一旦某一步出现了这种情况，后续的迭代能够使得它再回到正轨的可能性很小。整个迭代因此很可能会失败。从这点来说，它远不如对分法稳定。因此，在应用牛顿法的时候，如果我们能够事先对函数的极值点有比较深入的了解将是十分有帮助的。

17.2 多维的推广

¶ 如果函数是多维的，例如 $E = F = \mathbb{R}^n$，那么方程 $f(x) = 0$ 实际上等价于 n 个联立的非线性方程：

$$\begin{cases} f_1(x) \equiv f_1(x^1, x^2, \cdots, x^n) = 0 \,, \\ \quad \vdots \\ f_n(x) \equiv f_n(x^1, x^2, \cdots, x^n) = 0 \,. \end{cases} \tag{5.9}$$

完全类似于一维时的公式 (5.10)，我们这时有：

$$f(\xi) = 0 = f(x_0) + Df(x_0) \cdot (\xi - x_0) + \cdots \,, \tag{5.10}$$

其中 $Df(x)$ 表示函数 $f : \mathbb{R}^n \to \mathbb{R}^n$ 的 Jacobi 矩阵：

$$[Df(x)]_{ij} = \frac{\partial f_i(x)}{\partial x^j} \,. \tag{5.11}$$

我们假定函数的 Jacobi 矩阵在 ξ 附近不奇异，那么我们可以得到 Newton-Ralphson 算法的 n 维推广如下：

$$x_{i+1} = x_i - [Df(x_i)]^{-1} \cdot f(x_i) \,, \quad i = 0, 1, 2, \cdots \,. \tag{5.12}$$

当然其停止的条件一般是要求 Δx_i 在 n 维空间的模 $||\Delta x_i||$ 足够小。

¶ 前面讨论的算法都是属于一步迭代算法。即我们总是从某一个出发点 x_0 开始，进行迭代：$x_{i+1} = \Phi(x_i)$。例如，对于上面讨论的多维空间的 Newton-Ralphson 方法，其迭代函数 $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ 为：

$$\Phi(x) = x - [Df(x)]^{-1} \cdot f(x) \,. \tag{5.13}$$

一般可以证明，只要映射 $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ 是一个收缩映射 (contraction)，它最终总是会收敛到固定点：$x = \Phi(x)$。在 Jacobian 非奇异的前提下，这等价于 $f(x) = 0$。我们这里就不去探讨它了。有兴趣的同学可以参考 [2] 的 §5.2 节。

18 其他的方法

18.1 割线法

¶这是一个介于线性收敛和二次收敛之间的迭代方法。前面提到，一般的对分法属于典型的线性收敛的迭代方法，它每次迭代仅仅需要计算一次函数的值；如果我们可以方便地计算函数的导数值，那么收敛更快的牛顿法会更好，只不过它在每次迭代中除了计算一次函数值之外，还需要计算一次导数值。如果函数的导数的计算非常耗时或者并不知道，我们可以利用前两次计算的函数值来近似其导数值：

$$f'(x_i) \simeq \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} , \tag{5.14}$$

这样一来，牛顿法的迭代可以化为：

$$x_{i+1} = x_i - \frac{(x_i - x_{i-1}) f(x_i)}{f(x_i) - f(x_{i-1})} , \quad i = 1, 2, \cdots , \tag{5.15}$$

这就是所谓的割线法。注意，这个方法必须从两个出发点：x_0 和 x_1 出发而不是一个，但是它不需要计算函数的导数。它的计算量是，平均每次迭代只需要再计算一次 (而不是两次！) 函数值。割线法在收敛性上也与牛顿法类似。其致命的缺点是有可能不收敛。特别是当初始的两个点 x_0，x_1 之间恰好包含了函数 $f(x)$ 的一个极值点的话，割线法很可能会失败。当然，这个缺点是可以被克服的。我们需要的只是将割线法与对分法的精神加以结合。这就产生了所谓的试位法 (regula falsi method)。[1]

如果割线法收敛，它的收敛速度也与牛顿法比较类似。如果我们令第 i 次迭代与真实的根之间的误差记为 $\epsilon_i = x_i - \xi$，我们可以得到如下的估计，

$$\epsilon_{i+1} \simeq \frac{f''(\xi)}{2f'(\xi)} \epsilon_i \epsilon_{i-1} . \tag{5.16}$$

我们看到这个误差并不是像牛顿法那样 $\epsilon_{i+1} \sim (\epsilon_i)^2$，因此，它的收敛速度会比 Newton-Ralphson 法要慢一些。如果我们将上述公式取一个对数，我们就发现 $\ln \epsilon_i$ 基本上构成一个类 Fibonacci 数列，因此我们得到：

$$\epsilon_{i+1} \simeq C(\epsilon_i)^F \quad , F = \frac{1+\sqrt{5}}{2} \approx 1.6180 \cdots , \tag{5.17}$$

其中 C 是一个常数。因此，割线法的收敛速度介于对分法和牛顿法之间。[2]

[1] 这个方法虽然初看起来不错，但是其稳定性其实还不如对分法，虽然它也一定会收敛。

[2] 虽然我们讲述割线法的方式是将其作为牛顿法的一个“差分近似”，但是割线法 (以及相关的试位法) 的历史其实远比牛顿法要古老。

Algorithm 7 割线法求根

Require: 待求的根附近找出两个出发点 x_0，x_1 并计算函数值 $f(x_0)$ 和 $f(x_1)$；

【计算量】：每次迭代，计算 $f(x)$ 一次。

1: **while** $f(x_{i+1})$ 不足够接近于零 **do**
2: 令 $i = 1, 2, \cdots$，进行迭代 (5.15)
3: 计算 $f(x_{i+1})$，检查它是否足够小；
4: **end while**
5: 输出最终的 x_i 作为根。

¶ 事实上，对于由迭代函数 Φ 所描写的迭代法求根步骤，如果 Φ 对应的算法的收敛速度为 m 次的，我们可以利用复合函数构成一个复合的迭代：

$$\tilde{\Phi}(\cdot) = \Phi(\Phi(\cdot)) \,. \tag{5.18}$$

由于 $\epsilon_n \sim (\epsilon_{n-1})^m \sim (\epsilon_{n-2})^{m^2}$，那么复合的迭代 $\tilde{\Phi}$ 所对应的收敛幂次将是 m^2 次的。

上面提到的这个结论可以应用于割线法。前面曾提及，每次割线法的迭代只需要额外计算一次函数值，而牛顿法则需要计算一次函数值外加加一次导数值。如果导数值的计算量与函数值大体相当的话，那么牛顿法实际上每次迭代的计算量比割线法要多一倍。为此我们可以利用两次复合的割线法，其计算量大体与一次的牛顿法相当，但是其收敛的速度的指数将大于 2，具体来说其幂次为 $F^2 \sim 2.6$。因此在导数值计算量与函数值相当的前提下，这个复合割线法比牛顿法的收敛速度还要快。[3]

18.2 Aitken-Δ^2 算法

¶ 这是一种加速原先迭代收敛性的方法。如果我们已经有了一个收敛于一维函数 f 的根 ξ 的一个迭代序列 $\{x_i, i = 1, 2, \cdots\}$:

$$\lim_{i\to\infty} x_i = \xi \,, \tag{5.19}$$

我们希望构建另一个更快地收敛于 ξ 的序列 $\{x'_i, i = 1, 2, \cdots\}$。

为此我们假定原先的序列在 i 足够大时候满足

$$x_{i+1} - \xi = k(x_i - \xi) \,, |k| < 1 \,, \tag{5.20}$$

也就是说，x_i 提供了一个一阶的算法。注意到待求的函数的根 ξ 是不依赖于 i 的常数。如果我们假定 k 也是如此，我们发现这两个常数实际上可以通过 x_{i+2}，x_{i+1} 和 x_i 完全确定：

$$\begin{cases} k = \dfrac{x_{i+2} - x_{i+1}}{x_{i+1} - x_i} \\ \xi = \dfrac{x_{i+2}x_i - x_{i+1}^2}{x_{i+2} - 2x_{i+1} + x_i} \end{cases} \tag{5.21}$$

[3]注意，所谓的复合割线法并不需要更改原先割线法的任何步骤，只需要每迭代两次 (而不是一次) 考察一下结果即可。因此所谓的复合割线法在算法实现上就是割线法。

利用一阶和二阶的差分算符：

$$\Delta x_i \equiv x_{i+1} - x_i \ , \quad \Delta^2 x_i = \Delta(\Delta x_i) = \Delta x_{i+1} - \Delta x_i = x_{i+2} - 2x_{i+1} + x_i \tag{5.22}$$

我们可以将 ξ 的公式写为：

$$\xi = x_i - \frac{(\Delta x_i)^2}{\Delta^2 x_i} \ . \tag{5.23}$$

为此，我们构建：

$$x_i' = x_i - \frac{(\Delta x_i)^2}{\Delta^2 x_i} = x_i - \frac{(x_{i+1} - x_i)^2}{x_{i+2} - 2x_{i+1} + x_i} \ , \tag{5.24}$$

这就是所谓的 Aitken-Δ^2 算法，它实际上是一种加速算法。[4] 可以证明的是，只要原先的算法是一阶的，即公式 (5.20) 成立，那么按照公式 (5.24) 构建的新序列 $\{x_i', i = 1, 2, \cdots\}$ 一定比原先的算法收敛要快，即我们有：

$$\lim_{i\to\infty} \frac{x_i' - \xi}{x_i - \xi} = 0 \ . \tag{5.25}$$

¶ 前面提到，所有的迭代一般可以用一个迭代函数 $\Phi(\cdot)$ 来实现。利用这个函数，我们有 $x_{i+1} = \Phi(x_i)$，而函数的根 ξ 是迭代函数的固定点：$\xi = \Phi(\xi)$。在这个形式下，Steffenson 建议将 Aitken-Δ^2 算法改为

$$y_i = \Phi(x_i) \ , z_i = \Phi(y_i) = \Phi(\Phi(x_i)) \ , x_{i+1} = x_i - \frac{(y_i - x_i)^2}{z_i - 2y_i + x_i} \ , \tag{5.26}$$

换句话说，这等价于一个新的迭代函数 $\Psi(\cdot)$，其形式为

$$\Psi(x) = \frac{x\Phi[\Phi(x)] - [\Phi(x)]^2}{\Phi[\Phi(x)] - 2\Phi(x) + x} \ , \tag{5.27}$$

而新的迭代为：$x_{i+1} = \Psi(x_i)$。这又被称为 Steffensen 算法。可以证明的是，如果原先的迭代 $\Phi(\cdot)$ 给出一个 $p > 1$ 阶的算法，那么新的迭代 $\Psi(\cdot)$ 一般给出一个 $2p - 1$ 阶的算法。而对于 $p = 1$ 的情形，只要 $\Phi'(\xi) \neq 1$，则新的迭代 Ψ 至少给出一个 2 阶的算法。虽然看起来似乎这个算法对于 $p > 1$ 的算法似乎改进更大，但是这其实是一个错觉。Steffenson 的改进方法最常用的的时候恰恰是 $p = 1$ 的情形。对于 $p > 1$，事实上用复合迭代 $\Phi(\Phi(\cdot))$ 往往比利用 $\Psi(\cdot)$ 还要更有优势一些。

最为常用的 Steffensen 算法利用：

$$x_{i+1} = x_i - \frac{f(x_i)}{g(x_i)} \ , \quad i = 0, 1, 2, \cdots \ , \tag{5.28}$$

其中函数的导数值用：

$$g(x_i) = \frac{f(x_i + f(x_i)) - f(x_i)}{f(x_i)} \ . \tag{5.29}$$

[4] Alexander Aitken, "On Bernoulli's numerical solution of algebraic equations", Proceedings of the Royal Society of Edinburgh (1926) 46 pp. 289–305.

或者将两式合并写为

$$x_{i+1} = x_i - \frac{[f(x_i)]^2}{f(x_i + f(x_i)) - f(x_i)}\ ,\quad i = 0, 1, 2, \cdots\ , \tag{5.30}$$

这个实现的好处是它完全不需要进行函数导数的计算，仅仅需要计算函数值，当然每一步迭代需要计算两次：$f(x_i)$ 和 $f(x_i + f(x_i))$。可以证明，如果它收敛的话，那么这个方法是平方收敛的。但是由于每次迭代需要计算两次函数值，因此它不见得就比割线法更有优越性，因为后者每一次迭代只需要一次函数计算。于是，正如我们前面讨论过的 (见第 51页的讨论)，我们可以轻易构建一个复合的割线法使得其收敛幂次为 $F^2 \sim 2.6$。

19 函数极小值的寻找

¶ 极小值的寻找有时候又称为“优化”问题 (optimization)，因为我们总是可以将需要优化的内容利用模型的方法集合在一个函数之中，而数值上寻找这个函数的极小值就对应于原先的优化问题。这类问题又大致可以分为有约束的和没有约束的两类。我们这里将仅仅讨论没有约束的情形，因为这类问题相对简单一些。

我们的讨论首先从最为简单的一维函数开始，然后拓展到多维空间的函数。其实上述每一种情形又可以分为仅仅知道函数值时的优化，以及既知道函数值又知道其导数值 (甚至是高阶的导数值) 时的优化。由于课时的原因，我们将仅仅选择几个例子，至于其余的情形，请大家参考相应的参考书。

19.1 一维函数的极小值

¶ 一维函数 $f : \mathbb{R} \to \mathbb{R}$ 的求极值问题与求根的方法类似，我们仍然可以利用计算函数值比较的方法，不断地缩小搜寻的范围，最后获得函数的近似的极小值点。

假设我们并不知道函数的导数（如果知道，问题就转化为求其导数的零点问题了），仅仅能够计算其函数值。我们预估该函数在某个区域内具有极值。为此我们在该区域内选择三个点：$x_0 < x_1 < x_3$ 并计算 $f(x_0)$，$f(x_1)$ 和 $f(x_3)$。这三个函数值必须呈现出所谓的“中间小、两头大”的趋势：$f(x_0) > f(x_1)$，$f(x_3) > f(x_1)$。这时我们基本可以肯定 $f(x)$ 在区间 (x_0, x_3) 之间存在一个极小值点，如果这个函数在该区间内没有什么奇异点话。对于一个一维的函数，首先大致确定极小值所在的区间实际上是蛮重要的一件事情，而且这对于一维函数一般是可以做到的。这个方法与前面讨论的对分法求根的思想其实非常接近。唯一的区别是，对分法的区间由两个坐标确定，而这里则是三个，因此在比较和更新的时候稍微复杂一些罢了。

如图 5.1所示，函数 $f(x)$ 在某个区间 (x_0, x_1, x_3) 上呈现出中间小两头大的现象，即：$f(x_1) < f(x_0)$，$f(x_1) < f(x_3)$。那么函数 $f(x)$ 在区间 (x_0, x_3) 中间必定至少存在一个极值点。图中还显示了一些相应的距离参数，如 a，b，c 等。我们将假定函数在区间中仅有一个极小值点。目前我们对于它的初步估计就是点 x_1。我们下一步就是希望寻找下一

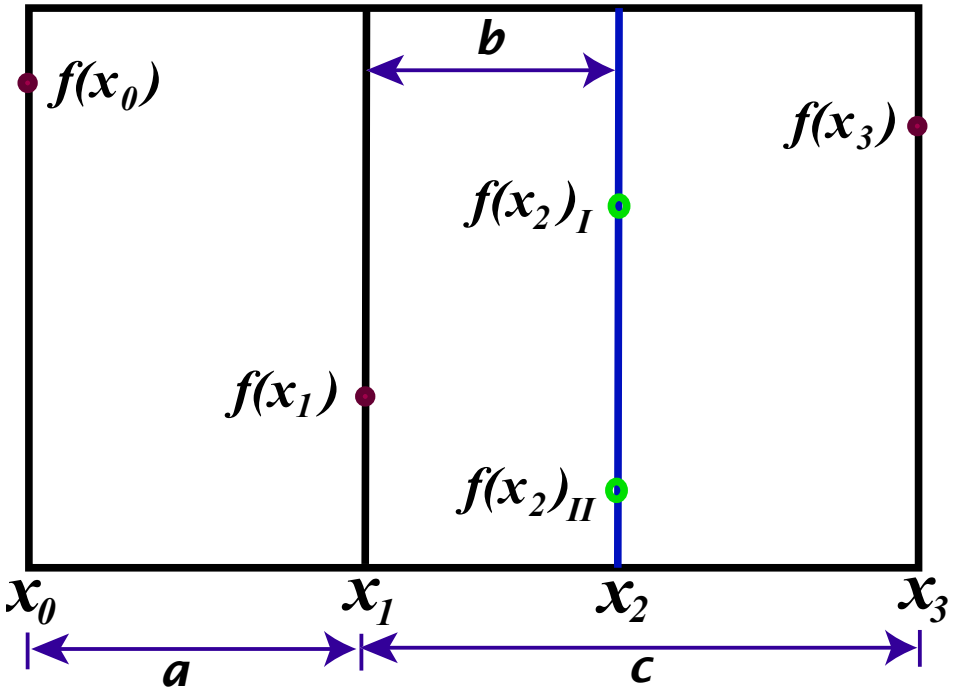


图 5.1: 函数 $f(x)$ 在区间 (x_0, x_1, x_3) 上呈现中间小，两头大的性状。我们知道它在该区间中一定有一个极小值。黄金分割搜寻法是试图寻找一个新的节点 x_2，并按照 $f(x_2)$ 与 $f(x_1)$ 大小的比较，分别用 (x_0, x_1, x_2)(标有下标 I 的情形) 或者 (x_1, x_2, x_3)(标有下标 II 的情形) 来替代原先的区间 (x_0, x_1, x_3)。

个更小的搜寻区间来替代目前的区间 (x_0, x_1, x_3)。新的、更小的搜寻区间必须仍然包含函数的极小值。随着我们不断地缩小搜寻区间的尺寸，我们对于函数极小值点的估计会越来越精确。

从原先的三个点 (x_0, x_1, x_3) 出发，我们希望寻找一个新的点 x_2，这个点可能在原先的中间点 x_1 的左侧，也可能在其右侧。图中显示的是恰好在其右侧的情形。然后我们可以计算函数 $f(x_2)$。我们现在将 $f(x_2)$ 与原先的中间点 x_1 处的函数值 $f(x_1)$ 进行比较。无非有两种可能：第一，$f(x_2) > f(x_1)$，即图中显示的较高的绿色的点，旁边标有 $f(x_2)_I$。显然这时函数在 (x_0, x_1, x_2) 上满足中间小、两头大的性质。我们可以用 (x_0, x_1, x_2) 替代原先的区间 (x_0, x_1, x_3)；第二，$f(x_2) < f(x_1)$，即图中显示的较低的绿色的点，旁边标有 $f(x_2)_{II}$。显然这时函数在 (x_1, x_2, x_3) 上满足中间小、两头大的性质。无论是哪一种情形，我们都得到了一个比原先区间更小的区间。显然，这个步骤可以重复迭代进行。

一个重要的问题是新的点 x_2 如何选择。我们假设区间 (x_0, x_1) 以及 (x_1, x_3) 的大小分别 a 和 c，新的点 x_2 相对于 x_1 的坐标为 b。注意，b 可以大于零（这对应于新的点 x_2 在

x_1 的右侧)；也可以小于零（这对应于新的点 x_2 在 x_1 的左侧)。图中显示的是在右侧的情形。由于我们无法判断在新的点处的函数值 $f(x_2)$ 是会大于还是小于旧的中间点的函数值 (也就是图中的情形 I 和情形 II)，我们的选择应当保持在这两种情形下，我们都能够得到有效的迭代算法。为此我们要求，在两种情形下，最后得到的新的区间的大小相等，即：

$$a+b=c\,,\Rightarrow b=c-a\,. \tag{5.31}$$

这意味着，如果 $c>a$ 则 $b>0$，反之如果 $c<a$ 则 $b<0$。因此，新的点一定位于原先两个区间 (x_0,x_1) 和 (x_1,x_3) 中 较大的 一个之中，这当然也是一个十分合理的结果。

要进一步确定最佳的 b 的数值我们需要再次利用迭代的理念。上述寻找步骤是需要计算机不断迭代的。因此，我们有理由要求一个好的迭代应当在利用同样的原则前提下可以在下一次中重复自己。这种尺度的“相似性”要求：

$$\frac{b}{c}=\frac{a}{a+c}\,, \tag{5.32}$$

将上述两个方程联立，我们发现比例 $\lambda_G\equiv c/(c+a)=a/c$ 满足：

$$\lambda_G^2+\lambda_G-1=0\,,\Rightarrow\ \lambda_G=\frac{-1+\sqrt{5}}{2}\approx 0.6180339\cdots\,. \tag{5.33}$$

这恰好是著名的 黄金分割 的比例。正因为如此，这种迭代又被称为 黄金分割迭代，或者 黄金分割搜寻。它是 J. Kiefer 在五十年代提出的。[5]

另一个重要的问题是何时停止上述黄金分割迭代。这当然依赖于我们对待求解问题的精度要求。为此我们假定区间已经足够小，以至于在函数的极小值点 $\bar{x}$ 附近我们可以利用泰勒展开：

$$f(x)\approx f(\bar{x})+\frac{1}{2}f''(\bar{x})(x-\bar{x})^2 \tag{5.34}$$

当上述近似中的第二项与第一项（也就是函数在极值点处的函数值）相比在机器允许的精度 ϵ 内可以忽略时，我们显然就没有必要再继续迭代了。也就是说如果

$$\frac{|x-\bar{x}|}{|\bar{x}|}\lesssim\sqrt{\epsilon}\cdot\sqrt{\frac{2|f(\bar{x})|}{\bar{x}^2f''(\bar{x})}}\,, \tag{5.35}$$

我们就可以停止迭代并将此时的中间点以及那里的函数值输出。需要注意的是上式中与机器精度有关的因子 $\sqrt{\epsilon}$，后面的那个因子对于通常的函数来说大概数量级为 1。我们知道对于单精度来说 $\epsilon\sim 10^{-7}$ 而对于双精度来说 $\epsilon\sim 10^{-16}$。这意味着 $\sqrt{\epsilon}$ 分别大约是 10^{-4}（单精度）和 10^{-8}（双精度)。超过机器精度的进一步的迭代显然是没有任何意义的。一个更为实用的停止条件是考察其相对误差。这时一个常用的停止条件是

$$|x_3-x_0|\le\sqrt{\epsilon}(|x_1|+|x_2|)\,, \tag{5.36}$$

[5] J. Kiefer, “Optimum sequential search and approximation methods under minimum regularity assumptions”, J. Soc. Indust. Appl. Math., 5(3), 1957, pp. 105-136.

其中对单精度来说 $\sqrt{\epsilon} \sim 10^{-4}$ 而对双精度而言 $\sqrt{\epsilon} \sim 10^{-8}$。对于黄金分割搜寻的收敛速度，可以证明每次迭代新的区间的测度会乘以一个因子 $\lambda_G \approx 0.618$。这点与对分法求根类似，只不过那里的因子是 $1/2$。所以按照我们的约定它基本上是线性收敛的。

下面的算法实现了上面的这些考虑。

Algorithm 8 Kiefer 黄金分割搜寻法

Require: 函数 $f(x)$ 在区间 (x_0, x_1, x_3) 上呈现中间小、两头大的行为。因此 $f(x)$ 在该区间内一定有极小值存在。

定义黄金分割比例常数 $\lambda_G = 0.61803399$ 和 $\bar{\lambda}_G \equiv 1 - \lambda_G = |b|/c$。

【计算量】：每次迭代计算函数值 $f(x)$ 一次；

1: 如果 $(x_3 - x_1) > (x_1 - x_0)$，新的点应当出现在点 x_1 的右侧。令 $x_2 = \lambda_G x_1 + \bar{\lambda}_G x_3$，并计算 $f(x_2)$；
2: **if** $f(x_2) > f(x_1)$ **then**
3: 做区间替换 $(x_0, x_1, x_3) \Rightarrow (x_0, x_1, x_2)$；
4: **else**
5: 做区间更新：$(x_0, x_1, x_3) \Rightarrow (x_1, x_2, x_3)$；
6: **end if**
7: 如果 $(x_3 - x_1) < (x_1 - x_0)$，新的点应当出现在点 x_1 的左侧。令 $x_2 = \bar{\lambda}_G x_0 + \lambda_G x_1$，并计算 $f(x_2)$；
8: **if** $f(x_2) > f(x_1)$ **then**
9: 做区间更新：$(x_0, x_1, x_3) \Rightarrow (x_2, x_1, x_3)$；
10: **else**
11: 做区间更新：$(x_0, x_1, x_3) \Rightarrow (x_0, x_2, x_1)$；
12: **end if**
13: 如果 x 的改变已经接近所预计的机器精度的极限 (5.36)，停止迭代并且输出 $(x_1, f(x_1))$ 或者 $(x_2, f(x_2))$ 这两对数据中函数数值较小的一对作为最终结果。否则回到上面的第 1 步或者第 7 步继续迭代。

19.2 多维函数的极值：单纯形方法

¶ 我们现在考虑函数 $f : \mathbb{R}^n \to \mathbb{R}$ 的求极小值问题。多维空间上的函数的极值计算方法大致可以分为两大类：一类是无需函数的导数 (或者梯度) 信息的方法；[6] 另一类则是需要函数导数信息的方法。我们本小节首先来讨论前者；后者我们可以统一称之为下降方法，它们一般都需要了解函数的导数值（甚至是二阶导数），我们将在下一小节介绍。

一种常用的无需计算函数的导数值的求极小方法是所谓的单纯形方法 (simplex

[6] 这可能有多种原因，可能是函数并没有具体的表达式以至于无法计算其导数，或者虽然有具体形式但是导数形式过于复杂等等。

method)。它由 Nelder 和 Mead 首先提出，因此又称为 Nelder-Mead 算法。[7] 我们下面会简单介绍一下这种方法。

单纯形方法可以看成是前一小节描述的一维中的线段法的多维推广。一个 n 维空间中的所谓单纯形 (simplex) 由 $n+1$ 个不在同一个超平面的上的点 $x^{(i)}: i=0,1,2,\cdots,n$ 以及它们之间的连线、三角形等等构成。这些点称为单纯形的顶点 (vertex)。我们这里要求单纯形的的确确在 n 维空间围成一块 n 维体积。在数学上这称为非退化 (non-degenerate) 的单纯形。一维的单纯形就是一个线段；二维的是一个有面积的三角形；三维的是个有体积的四面体等等。就像一个一维空间可以用线段来覆盖一样，一个 n 维空间可以用一系列的 n 维单纯形来覆盖，我们称之为单纯剖分。

给定一个 n 维空间的初始点 $x^{(0)}$，我们构建另外 n 个点：

$$x^{(i)} = x^{(0)} + h_i e_i \ , i=1,2,\cdots,n \ , \tag{5.37}$$

其中 $\{e_i : i=1,2,\cdots,n\}$ 是 n 维空间中的一组单位矢量基，其中 h_i 的选择需要根据具体问题中的尺度来确定。这 $n+1$ 个点构成了 n 维空间中的一个初始单纯形。注意，我们并不要求各个 e_i 构成正交基，只要它们线性无关即可，这保证了上述 $n+1$ 个点构成了 n 维空间的一个非退化的单纯形。我们的目的就是在初始单纯形的内部或者附近寻找函数的极小值。

为此，我们首先计算上述 $n+1$ 个点处的函数值：$f(x^{(i)})$。因为我们的目的是寻找函数的极小值，因此我们将这些点的函数值进行比较。请特别注意下列三个特殊的点：

- 最佳点：记为 $x^{(m)}$。它的函数值在上述 $n+1$ 个点中是最小的。因为我们要寻找极小值，所以它被冠以“最佳”的名号。与其对应的函数值则成为最佳值；
- 最差点：记为 $x^{(M)}$。它的函数值在上述 $n+1$ 个点中是最大的；与其相应的函数值称为最差值；
- 次差点：记为 $x^{(\mu)}$。它的函数中是上述 $n+1$ 个点中是第二大的。与其相应的函数值称为次差值。

所谓 M，m，μ 都是 0 到 n 之间的某个整数。除此之外，对于每一个点 $x^{(k)}$ 我们可以定义一个与它相对应 (或者说对偶) 的“中心点”，它实际上是所有其他点的代数平均 (或者说重心)：

$$x_c^{(k)} = \frac{1}{n} \sum_{j=0, j\neq k}^{n} x^{(j)} \ . \tag{5.38}$$

在这个算法中最为重要的是与最差点对偶的中心点 $x_c^{(M)}$。

[7] J. Nelder and R. Mead, “A simplex method for function minimization”, The Computer Journal 7, pp.308-313, 1965.

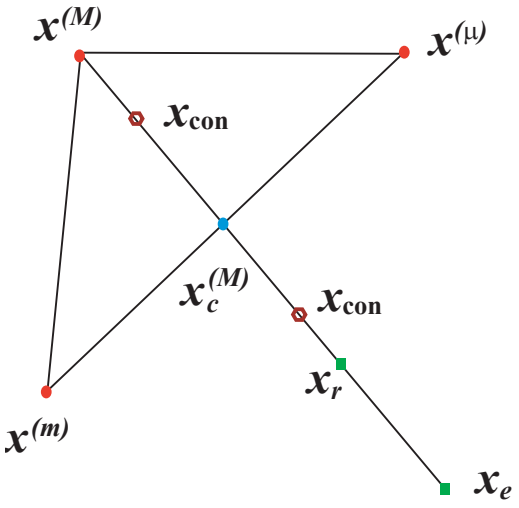


图 5.2: Nelder-Mead 算法 (Simplex algorithm) 的示意图。点 $x^{(m)}$，$x^{(\mu)}$，$x^{(M)}$ 分别为构成的单纯形 (二维就是三角形) 的顶点中的最佳、次差、最差点。点 $x_c^{(M)}$ 是与最差点对应的中心点；点 x_r 则是最差点相对于它的中心点的反射点；点 x_e 为扩展点而点 x_{con} 为收缩点。

我们下面要做的其实就是在通过一系列步骤，要么在初始的单纯形内部寻找函数的极小值，要么将初始的单纯形“更新”成一个新的单纯形。这些步骤主要包括反射 (reflection)、扩展 (expansion) 和收缩 (contraction) 等。读者可以参考图 5.2，其中画出了 $n = 2$ 维的情形，请特别注意下面的描述中提及的各个点。下面我们来具体描述这些步骤：

1. 对给定的 $n+1$ 个点构成的单纯形，确定点 $x^{(M)}$，$x^{(m)}$，$x^{(\mu)}$；

2. **【反射步骤】**：计算 $n+1$ 个点的中心处的函数值，如果它与 $n+1$ 点函数值的平均值之间的偏差

$$\Delta^2 = \frac{1}{n}\sum_{i=0}^{n}\left(f(x^{(i)}) - \bar{f}\right)^2 < \varepsilon\,, \tag{5.39}$$

其中 $\varepsilon > 0$ 为某个选定的正数，停止迭代并输出结果。上式中的的 $\bar{f}$ 的定义为

$$\bar{f} = \frac{1}{n+1}\sum_{i=0}^{n} f(x^{(i)})\,. \tag{5.40}$$

否则，将最差点 $x^{(M)}$ 相对于与它对偶的中心点 $x_c^{(M)}$ 进行反射

$$x_r = (1+\alpha)x_c^{(M)} - \alpha x^{(M)} \tag{5.41}$$

其中 $\alpha > 0$ 是一个可调整的反射系数。这个操作在几何上实际上是向点 $x^{(M)}$ 的相对于其他点的相反方向进行了反射。下面我们会根据函数值 $f(x_r)$ 的大小而进行不同的进一步操作。

3. 如果函数值介于最佳值和次差值之间，$f(x^{(m)}) < f(x_r) < f(x^{(\mu)})$，则用 x_r 替代最差顶点 $x^{(M)}$ 并回到上一步继续；

4. **【扩展步骤】**：如果函数值比最佳值还小，即 $f(x_r) < f(x^{(m)})$，说明 x_r 是一个新的最佳值。这意味着函数的极小值很可能不在原先的 $n+1$ 个点所确定的单纯性内部，而可能在其外部。为此，我们定义一个新的扩展点 x_e，

$$x_e = \beta x_r + (1-\beta)x_c^{(M)} , \tag{5.42}$$

其中 $\beta > 1$(例如，$\beta = 2$) 是一个可调的扩展参数。注意这个关系可以等价地写为：$x_e - x_c^{(M)} = \beta(x_r - x_c^{(M)})$。也就是说，点 x_e 一定在点 $x_c^{(M)}$ 到点 x_r 的延长线上 (距离放大到 β 倍)。现在计算函数值 $f(x_e)$，然后我们有两种选择：

- 如果 $f(x_e) < f(x^{(m)})$，则以 x_e 替代点 $x^{(M)}$;
- 如果 $f(x_e) \geq f(x^{(m)})$，则以 x_r 替代点 $x^{(M)}$;

回到第二步继续。

5. **【收缩步骤】**：如果 $f(x_r) > f(x^{(\mu)})$，那么极小值应当在点 x_r 的内侧 (靠向 $x_c^{(M)}$ 的一侧)，我们不用再向外寻找了。而是试图收缩已有的这些点以便找到更精确的极小值点。

- 如果 $f(x_r) < f(x_c^{(M)})$，极小值点应在 x_r 与 $x_c^{(M)}$ 之间，因此我们选收缩点：

$$x_{\text{con}} = \gamma x_r + (1-\gamma)x_c^{(M)} , \quad \gamma \in (0,1) , \tag{5.43}$$

- 如果 $f(x_r) \geq f(x_c^{(M)})$，极小值点应在 $x^{(M)}$ 与 $x_c^{(M)}$ 之间，因此我们选收缩点：

$$x_{\text{con}} = \gamma x^{(M)} + (1-\gamma)x_c^{(M)} , \quad \gamma \in (0,1) , \tag{5.44}$$

现在计算函数值 $f(x_{\text{con}})$ 并做如下的比较；

- 如果 $f(x_{\text{con}}) < f(x^{(M)})$ 且 $f(x_{\text{con}}) < f(x_r)$，将点 $x^{(M)}$ 替换为点 x_{con}，回到第二步继续；
- 如果 $f(x_{\text{con}}) \geq f(x^{(M)})$ 或者 $f(x_{\text{con}}) > f(x_r)$，利用公式 (5.37) 重新产生新的 n 个点 $x^{(k)}, k = 1, 2, \cdots, n$，将其中的各个 h_k 减半并从第一步开始。

在具体的操作中，为了逻辑上的简洁往往会在第一步计算完函数值后进行一次快速排序操作，这样就可以将各个点按照其函数值的大小排好次序。以便于下面的各项操作。

19.3 多维函数的极值：下降方法

¶ 如果我们能够获得函数的导数的信息，那么我们可以更好地寻找其极小值。本小节中我们假定 $f \in C^2(\mathbb{R}^n)$，因此它在点 $x \in \mathbb{R}^n$ 处的梯度 (gradient) 记为，

$$g(x) \equiv Df(x) = \left(\frac{\partial f}{\partial x^1}, \cdots, \frac{\partial f}{\partial x^n} \right)^T . \tag{5.45}$$

函数沿着某个方向 s 的导数则记为，

$$\frac{\partial f}{\partial s}(x) = \lim_{\alpha \to 0} \frac{f(x+\alpha s) - f(x)}{\alpha} = [Df(x)]^T \cdot s . \tag{5.46}$$

类似的，我们将函数的二阶导数，即所谓的 Hessian 矩阵，记为 $H(x)$，它的矩阵元为：

$$H(x)_{ij} = \frac{\partial^2 f(x)}{\partial x^i \partial x^j} . \tag{5.47}$$

¶ 在所有的下降方法中，依赖于我们对函数各阶导数了解情况，我们可以采取不同的迭代方法。下面我们将着重讨论的例子是著名的共轭梯度法。这个方法假设我们了解函数的一阶导数值。如果我们同时还可以获得函数更高阶的导数值，还存在收敛更快 (迭代次数更少) 的算法，不过在每一次迭代中需要的计算量也相应更大一些。

与求根问题不同的是，求极小值除了最初的出发点 $x_0 \in \mathbb{R}^n$ 之外，还需要有一个所谓的搜寻方向 (search direction)，$s_i \in \mathbb{R}^n$。因此，它的迭代方法一般为：

$$x_{i+1} = x_i - \lambda_i s_i , \tag{5.48}$$

其中 s_i 就是所谓的搜寻方向。一旦搜寻方向确定，我们可以令

$$\phi_i(\lambda) := f(x_i - \lambda s_i) , \tag{5.49}$$

并对 λ 取极小值 (也就是在 s_i 的方向上寻找一个一维函数的极小值)：$f(x_{i+1}) \simeq \min_\lambda \phi_i(\lambda)$。不同的算法会选择不同的搜寻方向 s_i，但是一般来说，我们总是要求在 $\lambda = 0$ 的地方，s_i 与函数的下降方向的夹角是锐角：

$$\phi_i'(0) = -g^T(x_i) s_i < 0 , \tag{5.50}$$

其中 $g(x)$ 是函数 $f(x)$ 在 x 处的梯度：

$$g^T(x) \equiv Df(x) = \left(\frac{\partial f}{\partial x^1}, \cdots, \frac{\partial f}{\partial x^n} \right) . \tag{5.51}$$

一个最初的想法是，我们每次都沿着函数的负梯度的方向走一小步难道不是最安全合理的吗？答案是，并非如此！永远严格沿着函数的负梯度方向搜索被称为“最陡下降法”。

这个方法实际上在数值上并不有效，因此其实已经不用了，尽管它可能源自大数学家 Cauchy。[8] 如果我们能够在无消耗的情形下进行无穷小的行走，那么这种方法当然是合适的。但是，我们往往只能够走有限大的步长。并且，在每一步中的计算量往往是恒定的 (如果选定某个固定的算法)，因此最终的总计算量依赖于我们要走多少步才可以到达最终的极小值点。如果函数的二阶导数 (其 Hessian 矩阵) 可能在某些方向比较奇葩，例如相较于其他方向而言具有特别大的数值。[9] 在这样复杂的地形中搜寻函数的极小值的时候，如果按照最陡下降法来做就会沿着一个多重的曲曲折折的折线行进。这样的效果是需要很多小碎步才能接近极小值，因此实际上是效率很低的。

另外一种方法是首先随机地选择一个方向 s。现在我们首先沿着这个方向将函数极小化。我们到达一个极小值点 ξ。按照定义，在这点函数的梯度一定没有沿着 s 分量，即

$$s^T Df(\xi) = 0 \,. \tag{5.52}$$

下一步我们再寻找另外一个方向 t，它必须不与 s 重合 (这不废话嘛！)。事实上，如何选择 t 是值得讨论的事情。需要注意的一点是，如果我们不希望破坏我们在 s 方向已经取得的成果，我们必须保证函数的梯度与 s 垂直。

假定我们的初始点距离极小值点不太远。我们将坐标的原点选择在初始点上。那么在原点附近函数 $f(x)$ 的 Taylor 展开为：

$$f(x) = f(0) - b^T x + \frac{1}{2} x^T A x + \cdots \,, \tag{5.53}$$

其中 $b = -Df(0)$ 是函数在原点的负梯度，A 是函数在原点的 Hessian 矩阵。不失一般性，我们将假定 $f(0) = 0$。在原点附近的任意一个点 x 处的梯度为：

$$Df(x) = Ax - b \,. \tag{5.54}$$

需要注意的是，在我们迭代搜寻的过程中，函数的梯度是不断改变的。显然，由于 b 是一个常矢量，因此迭代过程中的改变为：

$$\delta(Df(x)) = A(\delta x) \,, \tag{5.55}$$

因此，如果我们首先在多维空间中沿着某个方向 s 搜寻到函数的极小值了，现在我们要从该点出发，沿着一个新的方向 t 搜寻。我们当然不希望沿着新的方向的搜寻“破坏”我们之前沿着 s 方向搜寻已经达成的成果。因此我们必定要求搜寻过程中函数梯度的改变仍然与旧的方向垂直。即要求：

$$0 = s^T \delta(Df(x)) \Rightarrow s^T A t = 0 \,. \tag{5.56}$$

也就是说，并不是新方向与旧方向垂直，而是在 A 的作用下垂直。这样的方向 t 就称为原先方向的 共轭方向 (conjugate direction)

• • • •

[8] 注意，在 Cauchy 的那个年代还没有计算机。所以他本人并没有实践过这个算法。

[9] 设想一个非常狭长的山谷。在垂直于山谷方向的二级导数远远大于沿着山谷走向的二阶导数。

共轭梯度算法 (conjugate gradient algorithm) 就是一个建立在共轭方向上的一种下降方法。这种方法首先是 M. Hestenes, E. Stiefel, C. Lanczos 等人在五十年代提出的，求解线性方程组的方法。这个方法往往运用在大型稀疏矩阵的线性方程组的迭代求解中。后来 Fletcher，Reeves，Polak 等人将其发展到了无约束条件下求非线性函数的极小值问题。这也是获得二十世纪 top 10 算法提名的著名算法。[10] 它的主要方法是假定函数在极值点附近具有标准的二次形式，即公式 (5.53) 中的 $\cdots$ 是可以忽略的。由于要是极小值点，因此 Hessian 矩阵 A 必定是对称正定的实矩阵。

¶ 无论是最陡下降法还是共轭梯度法实际上都可以看成是在一个子空间中寻找一个二次函数的极小值的问题。这个二次函数就是

$$\Phi(x) = -b^T x + \frac{1}{2} x^T A x \ , \tag{5.57}$$

其中的 A 是一个正定对称实矩阵。如果我们忽略优化的函数 $f(x)$ 高于二阶的展开，这就是我们希望优化的目标函数 (5.53)。我们已经知道这个函数的梯度 $Df = D\Phi = Ax - b \equiv -r$。最陡下降方法对应于在迭代的第 k 步，沿着 r_k 进行优化：

$$x_{k+1} : \min_{\lambda} \Phi(x_k + \lambda r_k) \ , \quad r_k = b - A x_k \ , \tag{5.58}$$

这相当于在 r_k 的方向实行一个一维的优化。

共轭梯度方法则是引入了另外一个搜寻方向 p_k，它一般不同于 r_k。经过迭代，一系列的 p_k 实际上与一系列的 r_k 张成了同样的子空间。这个子空间可以看成是下列的一系列矢量所张成：

$$\mathcal{K}_{k+1}(A; r_0) = \text{span}\{r_0, A r_0, \cdots, A^k r_0\} \ , \tag{5.59}$$

这称为矢量 r_0 和矩阵 A 所生成的 $k+1$ 维 Krylov 子空间。在共轭梯度算法中，这个子空间等价于由各个 r_k 所构成的子空间。因此，在迭代的第 k 步，我们实际上是在 Krylov 子空间中进行优化：

$$x_{k+1} : \min_{\lambda_0, \cdots, \lambda_k} \Phi(x_k + \lambda_0 r_0 + \cdots + \lambda_k r_k) \ , \quad r_j = b - A x_j \ , \ j = 0, \cdots, k \ . \tag{5.60}$$

由于各个 r_j 是线性无关的，而一个 n 维空间最多有 n 个线性无关的向量，因此假定不考虑舍入误差的话，共轭梯度法可以保证最多迭代 n 次一定可以收敛于严格解。

共轭梯度法非常广泛地应用是利用迭代法求解线性方程，特别是求解大型稀疏矩阵的线性方程。这类应用最为常见的情形是偏微分方程的边值问题 (参见第 十 章的第节)。我们在后面会更加详细地处理。这里只不过是利用了一个二次型求极小 (具体说就是公式 (5.53) 所给出的二次函数) 与一个解线性方程：$Df \simeq Ax - b = 0$ 的等价性，将该算法进行了阐述。当然，这个前提是假定我们要求极小的函数在搜寻的点 (假定就在极小值点附近) 附近可以很好地用二次型 (5.53) 近似表达。初看起来我们必须计算函数在原点的的二阶 Hessian 矩阵 A。但是如果二次型近似 (5.53) 足够好，其实我们可以仅仅知道函数的

[10] 参见：http://www.siam.org/pdf/news/637.pdf。

梯度就可以了，因为对于一个标准的二次函数，它的梯度实际上同时也包含了其 Hessian 的信息。

下面的算法实现了基本的共轭梯度法，其中我们假定了 Hessian 矩阵 A 可以很容易地计算出来。

Algorithm 9 共轭梯度方法求极小值 (假定 Hessian 矩阵 A 已知)

Require: 函数在原点附近可以用二次型近似描述；

【计算量】：每一步迭代一次额外的 Ap_k 计算 (或者沿 p_k 方向求函数极小)

1: 初始设定：$k=0$，$x_0=0$，$p_0=r_0=-Df(x_0)$；

2: **repeat**

3: 计算：

$$\alpha_k=\frac{r_k^Tr_k}{p_k^TAp_k}=\frac{p_k^Tr_k}{p_k^TAp_k}\ , \tag{5.61}$$

4: 更新 x_k 和 r_k：

$$\begin{cases} x_{k+1}=x_k+\alpha_kp_k\ , \\ r_{k+1}=r_k-\alpha_kAp_k\ , \end{cases} \tag{5.62}$$

5: **if** $||r_{k+1}||$ 足够小 **then**

6: 停止迭代并输出 x_k 和 $f(x_k)$。

7: **end if**

8: 计算 β_k 并更新 p_k，

$$\begin{cases} \beta_k=\dfrac{r_{k+1}^Tr_{k+1}}{r_k^Tr_k}\ , \\ p_{k+1}=r_{k+1}+\beta_kp_k\ , \end{cases} \tag{5.63}$$

9: $k=k+1$；

10: **until** k 已经达到某个最大容许的次数。

11: **return** x_k 和 $f(x_k)$;

对于一个严格的二次型函数，CG 算法最多需要 n 次迭代一定可以收敛到其极小值 (假定没有 roundoff 误差的话)。这些良好的性质我们总结在如下的定理之中 (关于证明，请参考 [2] 的 §8.7 节)。

定理 5.1 对于一个 $n\times n$ 的正定对称实矩阵 $A\in\mathbb{R}^{n\times n}$。假定函数可以用一个二次型表达：$f(x)=\frac{1}{2}x^TAx-b^Tx$，其中 $b\in\mathbb{R}^n$。对于任意的初始点 $x_0\in\mathbb{R}^n$ 来说，上面的共轭梯度迭代过程中一定存在一个正整数 $l\le n$，使得 $p_l=0$，因此进一步的迭代停止 (x_l,r_l 不再发生变化)。之前所产生的各个矢量满足如下的性质：

1. x_l 恰好满足线性方程：$Ax_l=b$。因此，这个方法至多迭代 n 次一定可以获得方程 $Ax=b$ 的解；

2. 对于 $0 \le j < i \le l$ 来说，$r_i^T p_j = 0$;

3. 对于 $i \le l$ 来说，$r_i^T r_i = r_i^T p_i$;

4. 对于 $0 \le j < i \le l$ 来说，$p_j^T A p_i = 0$；对于 $j < l$ 来说，$p_j^T A p_j > 0$;

5. 对于 $0 \le j < i \le l$ 来说，$r_j^T r_i = 0$；对于 $j < l$ 来说，$r_j^T r_j > 0$;

6. 对于 $j \le l$ 来说，$r_j = b - A x_j$.

利用这个定理，也很容易估计出 CG 算法的收敛速度。我们假定 A 是正定实对称矩阵，于是我们可以定义一个矢量的模如下：

$$||y||_A = y^T A y \ , \quad y \in \mathbb{R}^n \ . \tag{5.64}$$

显然对于任意的非零矢量 y 来说我们都有 $||y||_A > 0$。我们记第 k 步迭代的解 x_k 与真实的解 x 之间的差为 $e_k = x_k - x$，那么对于 e_k 来说我们有如下的估计：

$$\frac{||e_k||_A}{||e_0||_A} \le \left[T_k \left(\frac{c+1}{c-1} \right) \right]^{-1} \le 2 \left(\frac{\sqrt{c}-1}{\sqrt{c}+1} \right)^k \ , \tag{5.65}$$

其中 $c = \lambda_{\max}/\lambda_{\min}$ 是矩阵 A 的条件数，$T_k(z)$ 表示 k 阶的 Chebyshev 多项式。因此，对于大条件数的矩阵来说，CG 的收敛速度是比较慢的，但是它的稳定性还是不错的，只要矩阵 A 的确是正定实对称矩阵。

上面的这个算法假定了函数在原点的 Hessian 矩阵 A 可以很容易地计算出来。于是上述算法中的 Ap_k 这类因子的计算就可以直接完成。如果这一点不成立，只要我们仍然能够很好的计算函数的导数值，并且假设函数可以很好地用二次型近似，那么 Hessian 的信息 –具体来说就是上述迭代中所有的 Ap_k 的因子 –也可以通过求一维方向上的极小值获得。例如，我们从 x_k 出发，沿着 p_k 方向求一个一维函数 $f(x_k + \lambda p_k)$ 的极小值：$\min_\lambda f(x_k + \lambda p_k)$，可以证明这个解恰好就是 $\lambda = \alpha_k$，而 α_k 就由公式 (5.61) 给出。随后的公式 (5.62) 中的第一式给出了 x_k 的更新，而第二式则等价于 $r_{k+1} = -Df(x_{k+1})$。

上述实现方式是 Fletcher-Reeves 原始的 CG 算法。后来 Polak 和 Ribbere 建议将 β_k 的表达式稍加修改：

$$\beta_k = \frac{(r_{k+1} - r_k)^T r_{k+1}}{(r_k)^T r_k} \ , \tag{5.66}$$

如果函数严格是二次型，那么根据前面定理中给出的正交关系，不难看出这个定义与原始的公式 (5.63) 完全一致。但是有证据表明，如果函数并不严格是二次型，这种新的 β_k 的取法会更有利一些。

¶ 正如我们上面指出的，共轭梯度法实际上只需要在每一点计算函数的梯度而不需要明确计算函数的 Hessian。如果我们所面对的求极小值问题中可以比较轻易地计算出函数的 Hessian 矩阵，还存在着其他收敛更快的算法，最为典型的就是类牛顿法。关于这些算法的计算大家可以参考 Numerical Recipes 中的相关讨论，我们这里就不再深入了。

相关的阅读

这一章我们简单介绍了计算函数零点以及求函数极值的数值方法。这类问题一般处理的函数都是非线性函数，因此往往需要利用迭代法来进行数值求解。

第六章　本征值和本征矢量数值求解

本章提要

- ☞ 线性代数知识的回顾
- ☞ QR 算法
- ☞ 矩阵的赝逆矩阵与奇异值分解
- ☞ 对称矩阵的本征值及本征矢

这一章中，我们将讨论数值线性代数中的进一步的问题，特别是本征值问题。我们也将对大型稀疏矩阵的线性代数问题进行简单的介绍。这类矩阵大量出现在偏微分方程的数值求解中。

20　本征值问题的一般描述

20.1　基本数学基础的回顾

¶ 让我们首先回忆一下关于本征值的一些数学定义和一些重要的结果。对于任意一个 $n \times n$ 的（可能是复的）矩阵 $A \in \mathbb{C}^{n\times n}$，如果存在一个非零的矢量 $x \in \mathbb{C}^n$ 和一个 $\lambda \in \mathbb{C}$ 满足

$$Ax = \lambda x \,, \tag{6.1}$$

我们就称 λ 为矩阵 A 的一个本征值，x 则称为对应于这个本征值的（右）本征矢量。矩阵 A 的所有本征值的集合称为矩阵 A 的谱，记为 $\sigma(A)$。[1] 我们会称 (λ, x) 为矩阵 A 的一

[1] 类似的如果有非零的矢量 y 满足 $y^\dagger A = \lambda y^\dagger$，$y$ 就称为 A 的左本征矢量。而如果不提左右，默认指的是右本征矢。

个本征对(eigenpair)。本章中主要的目的就是介绍如何利用数值方法求出一个给定矩阵的部分或者全部的本征对。另一个重要的概念是矩阵 A 的谱半径，它被定义为 A 的本征值（可能是复的）中的最大的模：

$$\rho(A) = \max_{\lambda\in\sigma(A)} |\lambda| \,. \tag{6.2}$$

由于 λ 是 A 的本征值等价于 $\bar{\lambda}$ 是 $A^\dagger$ 的本征值，因此结合谱半径的定义容易验明 $\rho(A) = \rho(A^\dagger)$。

矩阵的谱半径与它的模之间有着密切的联系。一个矩阵 $A \in \mathbb{C}^{n\times n}$ 的模可以通过多种方式定义 (在满足一定公理的前提下)。例如，我们可以定义

$$||A|| = \sup_{x\neq 0} \frac{||Ax||}{||x||} \,, \tag{6.3}$$

它被称为由矢量模 $||\cdot||$ 诱导的矩阵模，其中 $||\cdot||$ 是任何一个具有良好定义的、矢量空间 $\mathbb{C}^n$ 中的 (矢量) 模。比较常用的是所谓的 p-模，即：

$$||x||_p = \left[\sum_{i=1}^{n} |x_i|^p\right]^{1/p} \,, \tag{6.4}$$

而最为常用的是 $p = 2$ 的欧氏模。对于相应定义的矩阵模，我们也会在它的右下方加上一个脚标，例如 $p = 2$ 的矩阵模记为 $||\cdot||_2$。另一个常用的矩阵模是所谓的 Frobenius 模 (或 Hilbert-Schmidt 模)：

$$||A||_F = \sqrt{\sum_{i,j=1}^{n} |a_{ij}|^2} \,. \tag{6.5}$$

它实际上是与欧氏模兼容的，即：$||Ax||_2 \leq ||A||_F \cdot ||x||_2$。可以证明，一个矩阵 $A \in \mathbb{C}^{n\times n}$ 的欧氏模与 $A^\dagger A$ 和 $AA^\dagger$ 的谱半径以及 A 的最大奇异值 (参见后面的第 22节的讨论) 联系在一起：

$$||A||_2 = \sqrt{\rho(A^\dagger A)} = \sqrt{\rho(AA^\dagger)} = \sigma_1(A) \,. \tag{6.6}$$

因此对于厄米矩阵我们有 $||A||_2 = \rho(A)$ 而对于幺正矩阵来说我们有 $||A||_2 = 1$。

对于矩阵的数值计算中非常重要的是它的条件数 (condition number)。它的定义为：

$$\kappa(A) = ||A|| \cdot ||A^{-1}|| \,. \tag{6.7}$$

其中我们假定矩阵非奇异。一个奇异矩阵的条件数被约定为无穷。如果我们用矩阵的 p-模来定义条件数，我们相应地也会加上一个下标，例如 $\kappa_2(A) = ||A||_2||A^{-1}||_2$。利用公式 (6.6) 我们可以得到：

$$\kappa_2(A) = \sigma_1(A)/\sigma_n(A) \,, \tag{6.8}$$

其中 $\sigma_1(A)$，$\sigma_n(A)$ 分别为 A 的最大和最小奇异值。

线性代数的知识告诉我们，要求出一个矩阵的本征值，我们只需要令其特征多项式为零即可：

$$\chi_A(\lambda) \triangleq (-)^n \det(A-\lambda I) = (\lambda-\lambda_1)^{\sigma_1}\cdots(\lambda-\lambda_k)^{\sigma_k} = 0\,, \tag{6.9}$$

其中 $\chi_A(\lambda)$ 为 A 的特征多项式，其中各个本征值 $\lambda_1,\cdots,\lambda_k$ 被认为是不同的，它们相应的重数 $\sigma_1,\cdots,\sigma_k$ 被称为相应本征值的代数多重度 (algebraic multiplicity)。另外，下列事实也是大家所熟知的：

$$\det(A) = \prod_{i=1}^{n}\lambda_i\,,\quad Tr(A) = \sum_{i=1}^{n}\lambda_i\,. \tag{6.10}$$

如果一个矩阵的行列式为零，我们称这个矩阵是奇异 (singular) 的。按照上式它至少有一个本征值为零。事实上对 $A\in\mathbb{C}^{n\times n}$，下列事实是完全等价的：

- A 是非奇异的；
- $\det(A)\neq 0$;
- A 的核仅包含零矢量，即：$\ker(A)=\{0\}$;[2]
- A 是满秩的，即：$\text{rank}(A)=n$;
- A 具有线性独立的行和线性独立的列.

对某个 $\lambda\in\sigma(A)$，由所有相应的本征矢以及零矢量一起构成了一个矢量空间，称为本征值 λ 的本征空间。这个空间可以认为是线性变换矩阵 $(A-\lambda I)$ 的核 (kernel)：$\ker(A-\lambda I)$。如果我们用 $\text{rank}(A)$ 来表示一个矩阵 A 的秩 (rank) 的话，与一个本征值 λ 相应的本征空间的维数：

$$\dim[\ker(A-\lambda I)] = n - \text{rank}(A-\lambda I)\,, \tag{6.11}$$

被称为该本征值的几何多重度 (geometric multiplicity)。可以证明的是，对任何本征值而言，其几何多重度一定不大于代数多重度。几何多重度严格小于代数多重度的本征值被称为亏损本征值，如果矩阵 A 的本征值中至少有一个是亏损的，我们就称矩阵 A 是有亏损矩阵。[3]

[2] 一个变换矩阵 A 的核的定义为：$\ker(A)=\{x\in\mathbb{C}^n : Ax=0\}$。

[3] “亏损”是我在百度找到的翻译，相应的英文是 defective。俺不清楚这个翻译是否在数学中是标准的，反正乃们知道我说的是啥就行了。另一个相关但不同的概念是所谓的减次矩阵 (derogatory matrix)。

20.2 矩阵的本征结构与对角化

¶ 为了求出矩阵的本征值，我们往往需要对矩阵进行相似变换。对于 $A \in \mathbb{C}^{n\times n}$ 和另一个和它同样大小的、非奇异的矩阵 C 可以构建一个新的变换后的矩阵 $\tilde{A} \equiv C^{-1}AC$，这称为原矩阵 A 的一个相似变换 (similar transformation)。这两个矩阵，A 和 $\tilde{A}$，被称为是相似的 因为两者具有完全一致的谱。 事实上如果 (λ, x) 是 A 的一个本征对，那么容易验证 $(\lambda, C^{-1}x)$ 必定是 $\tilde{A}$ 的本征对。反之亦然。如果我们进行变换的矩阵 C 是幺正的，这时我们就称 A 与 $\tilde{A}$ 是幺正相似 的。幺正相似的矩阵在物理上往往是完全等价的，不同的只是选取了不同的表象而已。

关于一个任意矩阵的重要的结构性定理是所谓的 Schur 分解定理 (Schur decomposition theorem)。我们这里不加证明的引用。

定理 6.1 任给 $A \in \mathbb{C}^{n\times n}$，一定存在一个幺正矩阵 U 使得

$$U^{-1}AU \equiv U^{\dagger}AU = \begin{bmatrix} \lambda_1 & \tilde{a}_{12} & \cdots & \tilde{a}_{1n} \\ 0 & \lambda_2 & & \tilde{a}_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & \lambda_n \end{bmatrix}. \tag{6.12}$$

其中的对角元 $\lambda_i, i = 1, 2, \cdots, n$ 为 A 的本征值 (不一定不同)。

Schur 分解定理告诉我们，一个任意的 复矩阵一定可以利用某个 (不一定是唯一的) 幺正变换 U 化为一个上三角矩阵，变换后的矩阵之对角元为原矩阵的本征值。这个形式的上三角矩阵称为原先矩阵 A 的舒尔形式。需要着重指出的是，虽然任何复矩阵都可以利用幺正矩阵化为上三角矩阵，但这并不意味着任何的矩阵都是可对角化 (diagonalizable) 的，后者要求矩阵 $A \in \mathbb{C}^{n\times n}$ 必须与一个对角矩阵 (而不仅仅是上三角矩阵) 相似。一个一般的 $A \in \mathbb{C}^{n\times n}$ 并不一定是可对角化的。由于本征值 λ 可以由相应矩阵的特征多项式为零给出，按照代数学基本定理，本征值在复数域内总是存在 n 个根的 (可能重复)。因此，Schur 定理中的对角元总是存在的 (并且原则上可以数值求出)；但是对不可对角化的矩阵来说，我们并不能找到一个非奇异的相似变换 C 使得 $C^{-1}AC$ 是完全对角的。这种情形出现在矩阵的某个本征值是亏损的时候，此时我们无法找到 n 个线性独立的本征矢来张成全空间 $\mathbb{C}^n$。这时的矩阵 A 一定是不可对角化的。

原始的 Schur 分解定理 6.1中并没有对复矩阵 A 做任何的限制。如果我们要求它再具有一些额外的性质，比如对于厄米性，或者其他在幺正变换下保持不变的性质，我们很容易发现 Schur 定理的下列重要推论：

推论 6.1 给定任意一个厄米矩阵 $A \in \mathbb{C}^{n\times n}$，$A^{\dagger} = A$，一定存在一个幺正变换 U 将其对角化: $U^{-1}AU = \mathrm{Diag}(\lambda_1, \cdots, \lambda_n)$，其中所有的本征值 λ_i 都是实数，并且相应的幺正矩阵 U 的列向量就是相应的本征矢量，即 $U = [x_1, x_2, \cdots, x_n]$ 满足 $Ax_i = \lambda_i x_i$，而且我们可

以选择 x_i 使得它们两两正交并且张成整个 $\mathbb{C}^n$ 空间。[4]

显然，厄米性在幺正变换下保持不变。因此，对一个厄米矩阵来说，我们可以将其变换为上三角矩阵的同时也将其变换为了下三角矩阵。一个同时为上三角和下三角的矩阵一定就是对角矩阵，即矩阵已经被对角化了。在数学上，厄米矩阵的一个推广是所谓的正规矩阵(normal matrices)。一个矩阵 $A \in \mathbb{C}^{n\times n}$ 如果满足 $AA^\dagger = A^\dagger A$ 就称为正规矩阵。正规矩阵就是和自身的厄米共轭对易的矩阵。厄米矩阵当然是正规矩阵，因为任何矩阵都和自己对易。另一个在物理上经常用到的正规矩阵就是幺正矩阵，因为这时候 $AA^\dagger = A^\dagger A = I$。正规矩阵几乎满足推论 6.1中关于厄米矩阵完全一样的性质，即它也属于可对角化的矩阵，唯一的区别是它的本征值 λ_i 不再保证一定是实数。另一点值得提及的是，对正规矩阵而言（当然也包括厄米矩阵），Schur 定理中提到的幺正变换 U 实际上几乎是唯一的。[5]

那么哪些矩阵是不可对角化的呢？最为典型的例子是所谓的 Jordan 标准矩阵。一个 $\nu\times\nu$ 阶的 Jordan 标准矩阵的典型形式 (假定 $\nu \geq 1$) 为：

$$J_\nu(\lambda) \equiv \begin{bmatrix} \lambda & 1 & \cdots & & 0 \\ 0 & \lambda & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \\ & & & \lambda & 1 \\ 0 & \cdots & & 0 & \lambda \end{bmatrix}_{\nu\times\nu} . \tag{6.13}$$

大家可以轻易验证，这个矩阵的本征值为 λ，这一般是一个多重根 (假定 $\nu \geq 2$)，其代数多重度为 ν。但是如果你试图去寻找它相应的本征矢的话，你会发现它仅有一个线性独立的本征矢，即：$x = (1, 0, \cdots, 0)^T$。因此，这是一个亏损矩阵，是不可对角化的。上面的 Jordan 标准矩阵虽然看起来十分特殊，但是线性代数中的一个重要定理告诉我们，任何一个 $n\times n$ 的复矩阵都可以通过一个非奇异的相似变换化为由若干个形如上述 Jordan 块矩阵所构成的块对角矩阵：

定理 6.2 对任意的 $A \in \mathbb{C}^{n\times n}$，总可以找到一个非奇异的矩阵 $X \in \mathbb{C}^{n\times n}$ 使得

$$X^{-1}AX = J = \mathrm{Diag}(J_{\nu_1}(\lambda_1), \cdots, J_{\nu_k}(\lambda_k)) , \tag{6.14}$$

其中 $J_\nu(\lambda)$ 是公式 (6.13) 中定义的 Jordan 块矩阵 (若 $\nu \geq 2$)，$\lambda_1, \cdots, \lambda_k$ 是矩阵的本征值 (可能相同)；而如果 $\nu = 1$，我们约定 $J_1(\lambda) = \lambda$。

对这个定理的证明有兴趣的同学可以参考相关的线性代数教材。这个定理告诉我们，每个矩阵都可以通过相似变换，变换为由 Jordan 块矩阵构成的标准形式。它称为该矩阵的 Jordan 标准型。其中的每一个块矩阵 $J_\nu(\lambda)$ 称为原来矩阵的一个 Jordan 块。虽然我们

[4] 这其实是大家在量子力学中熟悉的结论。现在大家可以理解为什么在量子力学中，我们要求物理量对应于一个厄米矩阵了。

[5] 当然你可以将不同的本征矢量重新排列一下，这个不计算在内的话。

一般并不会真正去计算一个矩阵的 Jordan 标准型，但是它的存在性和结构特点仍然对我们后面的本征值以及本征矢的计算有着重要的理论指导意义。[6]

关于矩阵的约当标准型的重要结论我们有如下的额外说明：

- 对每一个给定的本征值 λ，可以有不止一个 Jordan 块。也就是说对一个给定的本征值，可以出现多个与之相应的 Jordan 块，并且每个块的大小还可以不同。但正如我们前面刚刚提到的，对每个 Jordan 块 $J_\nu(\lambda)$ 来说，对应于其相应本征值 (即 λ)，有且只有一个线性独立的本征矢。因此，一个矩阵的 Jordan 块的总数就是能够找到的线性独立的本征矢的总数。

- 一个矩阵要是可对角化的充要条件就是它的 Jordan 标准型中的所有的 Jordan 块都是 1×1 的 Jordan 块矩阵。由此我们看到，只要矩阵的 Jordan 标准型中有某个 Jordan 块的尺寸是超过 1 的，它一定是不可对角化的，而这恰恰与亏损矩阵的定义是一致的。因此，矩阵的非亏损性与它的可对角化性是完全等价的概念。

- 我们前面看到，一个正规矩阵 (重要的例子包括厄米矩阵、幺正矩阵等等) 是可对角化的。另一个可对角矩阵的例子是所有本征值 (无论是实的还是复的) 都不相同的矩阵一定也是可以对角化的，因为它的每个 Jordan 块都是 1×1 的。事实上，如果所有的本征值都不同，其相应的本征矢必定是线性无关的从而一定张成全空间。

简而言之，一个矩阵是否可以对角化，直接与该矩阵的 Jordan 标准型的形式挂钩。

¶ 由于矩阵的亏损性、减次性等概念与矩阵的本征值以及其对角化关系十分密切，下面我们稍微更详细地来阐明一下这些概念之间的联系。上面已经提及，矩阵的亏损性直接与矩阵的 Jordan 标准型的形式有关。另一个相关的概念 –矩阵的减次性 –则是与矩阵的所谓最小多项式 (minimal polynomial) 相关联的。按照定义，一个方阵 A 的最小多项式 $\mu_A(\lambda)$ 是一个关于 λ 的多项式，其 λ 的最高幂次的系数为 1，并且是满足 $\mu_A(A) \equiv 0$ 的幂次最低的多项式。大家可以回忆一下线性代数中的所谓 Cayley–Hamilton 定理 (Cayley–Hamilton theorem)，它告诉我们矩阵的特征多项式满足 $\chi_A(A) = 0$。最小多项式则被定义为满足这个的最低幂次的多项式。因此，矩阵的特征多项式必定包含最小多项式为因子。一个矩阵 A 的特征多项式可以与最小多项式完全相同。但如果两者不相同，那么特征多项式一定比最小多项式的幂次要高。这时候我们称矩阵 A 是减次的。

要求出矩阵的最小多项式，我们不加证明地引用一个数学的结论：对于矩阵 $(A - \lambda I)$ 而言，我们可以在它的左右各乘以一个非奇异的矩阵，这两个矩阵 (比方说 U 和 V) 一般来说是不相同的同时也没有什么关联，这个操作可以将矩阵 $(A - \lambda I)$ 化为下列的对角型

[6] 一个矩阵的 Jordan 标准型的数值计算是十分不稳定的。一个典型的例子是 $A = \left[\begin{smallmatrix} 1 & 1 \\ \epsilon & 1 \end{smallmatrix}\right]$。若 $\epsilon \neq 0$，则 A 的 Jordan 标准型为 $\left[\begin{smallmatrix} 1+\sqrt{\epsilon} & 0 \\ 0 & 1-\sqrt{\epsilon} \end{smallmatrix}\right]$；而如果 $\epsilon \equiv 0$, 那么上式中的 $A = \left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right]$ 已经是 Jordan 标准型了。正因为如此，并没有非常稳定的数值方法来完成这个任务。数值上更稳定的是计算矩阵的 Schur 标准型。我们下面介绍的 QR 算法就是如此。

的矩阵，它称为矩阵 $(A-\lambda I)$ 的 Smith 标准型 (Smith normal form)：

$$U(A-\lambda I)V=\begin{bmatrix} E_1(\lambda) & & & & \\ & \ddots & & & \\ & & E_i(\lambda) & & \\ & & & \ddots & \\ & & & & E_n(\lambda) \end{bmatrix} . \tag{6.15}$$

其中各对角元 $E_i(\lambda)$ 满足如下的性质：它们都是 λ 的最高次系数为 1 的多项式，并且每个 $E_i(\lambda)$ 必定可以除尽 $E_{i+1}(\lambda)$，$i=1,\cdots,n-1$。也就是说，每一个多项式 $E_i(\lambda)$ 都是后一个 $E_{i+1}(\lambda)$ 的因子。这些 $E_i(\lambda)$ 称为矩阵 $(A-\lambda I)$ 的 不变因子 (invariant factors)。这些不变因子可以进行因式分解：[7]

$$E_i(\lambda)=(\lambda-\alpha_1)^{e_{i1}}\cdots(\lambda-\alpha_k)^{e_{ik}} , \tag{6.16}$$

其中各个幂次 e_{ij}，$i=1,\cdots,n, j=1,\cdots,k$ 都是零或者正整数。各个因子 $(\lambda-\alpha_j)^{e_{ij}}$ 中非平庸的因子，即 $e_{ij}>0$ 的因子被称为矩阵 A 的 基本因子 (elementary divisor)。由于前面提及的两个变换矩阵 U 和 V 都是非奇异的，因此当 $\lambda=\alpha_j, j=1,\cdots,k$ 时 $\det(A-\lambda I)=0$，即各个不同的 α_j 其实就是原来矩阵 A 的本征值。按照定义，$E_i(\lambda)$ 是 $E_{i+1}(\lambda)$ 的一个因子，所以我们必定有：$e_{ij}\le e_{i+1,j}$，$i=1,\cdots,n-1$，$j=1,\cdots,k$。

利用这个记号，矩阵 A 的特征多项式为，

$$\chi_A(\lambda)\equiv(-1)^n\det(A-\lambda I)=\prod_{i=1}^{n}E_i(\lambda)=\prod_{i=1}^{n}\prod_{j=1}^{k}(\lambda-\alpha_j)^{e_{ij}} . \tag{6.17}$$

而矩阵 A 的最小多项式则为，

$$\mu_A(\lambda)=\prod_{j=1}^{k}(\lambda-\alpha_j)^{e_{nj}} . \tag{6.18}$$

也就是说，最小多项式是对应于每个本征值 α_j 的最高幂次的因子 $(\lambda-\alpha_j)^{e_{nj}}$ 的连乘积。我们在线性代数中曾经证明过一个所谓的 Cayley–Hamilton 定理，

我们也可以从矩阵的 Jordan 标准型来考察这个问题。矩阵 A 的约当标准型中的第 i 个约当块阶数记为 e_i，其中 $i=1,\cdots,t$ 标记矩阵 A 的所有 Jordan 块，当然我们有 $1\le e_i\le n$。由于约当标准型的行列式中必须包含所有的基本因子，因此阶数为 e_i 的约当块对特征多项式贡献的因子为 $(\lambda-\lambda_i)^{e_i}$，从而 A 的特征多项式一定可以写为：

$$\chi_A(\lambda)=\prod_{i=1}^{t}(\lambda-\lambda_i)^{e_i} , \quad \sum_{i=1}^{t}e_i=n . \tag{6.19}$$

[7]在下面这个表达式中，我们约定各个 α_j，$j=1,\cdots,k$，$k\le n$ 是不同的。否则我们可以将两个因子合并为一个因子并将其相应的幂次相加。

但是需要注意的是，这里面的各个 λ_i 并不一定是不相同的。它们完全可以是相同的本征值，只不过出现在不同的 Jordan 块之中罢了。换句话说，上式中的各个 λ_i 就是前面公式 (6.17) 中的各个 α_j，只不过那里我们要求各个 α_j 必定是不同的而这里各个 λ_i 则可以相同。也正因为如此，这里的各个 e_i 一般来说也不一定等同于前面讨论的代数多重度 σ_i。

有了上面的这些基本概念的澄清，我们现在可以比较明确地来表述矩阵的亏损性以及减次性了。

- 如果矩阵 A 的第 i 个约当块的阶数 $e_i > 1$(即约当块的尺寸至少是 2×2) 这个本征值就称为亏损的 (defective)；矩阵 A 中只要有任何一个本征值是亏损的，矩阵 A 就被称为亏损的，反之该矩阵就被称为不亏损的 (non-defective)。这个名称的来源是，正如我们上面多次提到的，对应于一个亏损的本征值来说，我们仅能找到一个本征矢量。因此，如果与该本征值对应的约当块的阶数 $e_i > 1$，那么我们显然还缺少 $(e_i - 1)$ 个线性无关的矢量以张成相应的 e_i 维的矢量空间。

- 如果矩阵 A 的任何两个不同的约当块中包含了同一个本征值 λ_i，我们就称相应的本征值是减次的 (derogatory)；只要矩阵 A 有一个本征值是减次的，该矩阵就被称为减次的。反之则称为非减次的 (non-derogatory)。对减次矩阵而言，其最小多项式 $\mu_A(\lambda)$ 一定比其特征多项式 $\chi_A(\lambda)$ 的幂次要低且后者包含前者作为因子。两者都满足：$\mu_A(A) = \chi_A(A) = 0$。

在下面的例子中，我们将分别给出这上述两个概念 (亏损与减次) 相互交叉的四种可能情形：非亏损、非减次；非亏损减次；亏损非减次；亏损且减次的矩阵的实例。我们将以 4×4 矩阵为例，$\lambda_1, \lambda_2, \lambda_3, \lambda_4$ 则代表四个不同的本征值。我们将分别给出这些矩阵的基本因子 $E_i(\lambda)$、最小多项式 $\mu_A(\lambda)$ 以及特征多项式 $\chi_A(\lambda)$。

- **非亏损且非减次**

$$A = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \lambda_3 & \\ & & & \lambda_4 \end{bmatrix} . \tag{6.20}$$

这个矩阵的各个不变因子以及最小多项式、特征多项式为：

$$\begin{aligned} E_1 = E_2 = E_3 = 1 \, , E_4 = (\lambda - \lambda_1)(\lambda - \lambda_2)(\lambda - \lambda_3)(\lambda - \lambda_4) \, , \\ \mu_A(\lambda) = \chi_A(\lambda) = E_4(\lambda) \, . \end{aligned} \tag{6.21}$$

矩阵具有 4 个不相等的本征值，相应于每个本征值有一个本征矢。显然可以选为 $(1,0,0,0)^T$，$(0,1,0,0)^T$，$(0,0,1,0)^T$ 和 $(0,0,0,1)^T$，它们张成了整个 4 维空间。

- **非亏损但减次**

$$A=\begin{bmatrix}\lambda_1 & & & \\ & \lambda_1 & & \\ & & \lambda_3 & \\ & & & \lambda_4\end{bmatrix}. \tag{6.22}$$

这个矩阵的各个不变因子以及最小多项式、特征多项式为：

$$\begin{aligned} &E_1=E_2=1\,,E_3=\lambda-\lambda_1\,,E_4=(\lambda-\lambda_1)(\lambda-\lambda_3)(\lambda-\lambda_4)\,,\\ &\mu_A(\lambda)=(\lambda-\lambda_1)(\lambda-\lambda_3)(\lambda-\lambda_4)\,,\\ &\chi_A(\lambda)=(\lambda-\lambda_1)^2(\lambda-\lambda_3)(\lambda-\lambda_4)\,. \end{aligned} \tag{6.23}$$

矩阵具有 2 个不相等的本征值 λ_3 和 λ_4，同时还有一个二重简并的本征值 λ_1。但是相应于每个本征值，包括简并的本征值在内，都有有足够多的线性独立的本征矢以张成相应的本征空间。大家容易验证，$(1,0,0,0)^T$，$(0,1,0,0)^T$，$(0,0,1,0)^T$ 和 $(0,0,0,1)^T$，它们仍然张成了整个 4 维空间。其中前两个张成了对应于二重简并本征值 λ_1 的二维本征空间。但是，由于简并造成了矩阵的减次，即最小多项式比特征多项式的幂次要低，少了一个因子 $(\lambda-\lambda_1)$。

- **亏损但非减次**

$$A=\begin{bmatrix}\lambda_1 & 1 & & \\ & \lambda_1 & & \\ & & \lambda_3 & \\ & & & \lambda_4\end{bmatrix}. \tag{6.24}$$

这个矩阵的各个不变因子以及最小多项式、特征多项式为：

$$\begin{aligned} &E_1=E_2=E_3=1\,,E_4=(\lambda-\lambda_1)^2(\lambda-\lambda_3)(\lambda-\lambda_4)\,,\\ &\mu_A(\lambda)=\chi_A(\lambda)=(\lambda-\lambda_1)^2(\lambda-\lambda_3)(\lambda-\lambda_4)\,. \end{aligned} \tag{6.25}$$

矩阵具有 2 个不相等的本征值 λ_3 和 λ_4，同时还有一个二重简并的本征值 λ_1。但是与 λ_1 相应的约当块是大于 1 阶的。因此，对应于这个本征值仅有一个线性独立的本征矢：$(1,0,0,0)^T$。另外两个本征值各自有各自的本征矢：$(0,0,1,0)^T$ 和 $(0,0,0,1)^T$。但是整个矩阵仅有三个线性独立的本征矢，无法张成整个 4 维空间。因此矩阵是亏损的，但是不减次。

- **亏损且减次**

$$A=\begin{bmatrix}\lambda_1 & 1 & & \\ & \lambda_1 & & \\ & & \lambda_1 & \\ & & & \lambda_4\end{bmatrix}. \tag{6.26}$$

这个矩阵的各个不变因子以及最小多项式、特征多项式为：

$$\begin{aligned} & E_1 = E_2 = 1\,, E_3 = \lambda - \lambda_1\,, E_4 = (\lambda-\lambda_1)^2(\lambda-\lambda_4)\,, \\ & \mu_A(\lambda) = E_4(\lambda) = (\lambda-\lambda_1)^2(\lambda-\lambda_4)\,, \\ & \chi_A(\lambda) = (\lambda-\lambda_1)^3(\lambda-\lambda_4)\,, \end{aligned} \tag{6.27}$$

矩阵具有 2 个不同的本征值 λ_1 和 λ_4，前者是三重简并的。但是与 λ_1 相应的约当块由一个 1 阶和一个 2 阶构成。因此，对应于这个本征值的二阶的约当块将仅有一个线性独立的本征矢：$(1,0,0,0)^T$；另一个一阶的约当块的本征矢可以取为 $(0,0,1,0)^T$；最后 λ_4 的本征矢可以取为 $(0,0,0,1)^T$。但是整个矩阵仍然仅有三个线性独立的本征矢，无法张成整个 4 维空间。因此矩阵既是亏损的，又是减次的。

20.3 本征值的分布

¶ 有的时候我们需要对矩阵的本征值在复平面的分布做一个估计。这时就可以用到 Gershgorin(或者 Geršgorin) 的一系列定理。其中最为重要的是

定理 6.3 对任意的 $A \in \mathbb{C}^{n\times n}$，它的谱记为 $\sigma(A)$。我们构建复平面的 n 个圆 (盘):

$$\mathcal{R}_i = \left\{ z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{j=1, j\neq i}^{n} |a_{ij}| \right\}\,, \tag{6.28}$$

其中 $i = 1, 2, \cdots, n$。它们称为 Gershgorin 圆 (盘)。那么矩阵 A 的本征值一定落在这些圆盘的并集之中:

$$\sigma(A) \subseteq \mathcal{S}_{\mathcal{R}} = \bigcup_{i=1}^{n} \mathcal{R}_i\,. \tag{6.29}$$

这个称为 Gershgorin 圆盘定理。对其证明有兴趣的同学可以参考 [5] 的 §5.1 节或者 [6] 的 §9.1 节。这个定理对于实对称矩阵 (或者厄米矩阵) 特别有效，因为它们的本征值一定都位于实轴上。因此，这个定理可以告诉我们矩阵的本征值所在的区间。我们在后面 (见第 23.2小节) 讨论 Sturm 序列的时候会用到这个定理。

20.4 推广的本征值问题

¶ 一种推广的本征值问题是需要求解下列问题。给定两个矩阵 $A, B \in \mathbb{C}^{n\times n}$，寻找一个复数 λ 以及一个非零矢量 x 使得下式得到满足：

$$Ax = \lambda B x\,. \tag{6.30}$$

这时我们称 (λ, x) 是推广的本征值问题的一个本征对。这时可以引入所谓的矩阵束 (matrix pencil) 的概念。我们把矩阵 $L(\lambda) = (A - \lambda B)$ 视为 λ 的在 $\mathbb{C}^{n\times n}$ 空间取值的函数，并将其记为 (A, B)。矩阵束的本征值问题也可以用下面要讨论的 QR 算法的推广版本来处理。我们这里就不再涉及了。有兴趣的同学可以参考 [5] 的 §5.9 节。

21 QR 算法

¶我们本节将着重介绍著名的 QR 算法。这是目前各界应用的最为广泛的算法之一。一般来说，如果待求的矩阵不是特别巨大 ($n \lesssim 3000$)，那么 QR 算法实际上是最为合适的算法。在本世纪初，人们总结了 20 世纪诞生的最为有用的 10 个算法，[8] QR 算法就被选中，位列第六 (其实排名是按照算法诞生的时间排列的，所以并不是说前面的算法就更重要)。

我们将首先计算原始的 QR 算法。我们会看到，对于一个一般的矩阵，这个算法的迭代中需要的计算量是蛮大的。因此，有必要在迭代之前首先将矩阵化为简单的形式。常用的方法包括 Householder 约化和 Givens 变换。然后我们会讨论 Hessenberg-QR 算法。最后简单介绍一下加速其收敛的 shifted QR 算法 (这实际上是很多线性代数包中运用的方法)。为了简单起见，我们将仅仅讨论实数矩阵的 QR 算法。关于更为复矩阵的 QR 算法，有需要的同学可以参考其他的资料。[9]

21.1 原始的 QR 算法

¶如前所述，我们这里只考虑 $n \times n$ 实矩阵的 QR 算法。令 $A \in \mathbb{R}^{n\times n}$ 为任意一个 $n \times n$ 的实矩阵。$Q^{(0)} \in \mathbb{R}^{n\times n}$ 为一实正交矩阵。我们令矩阵 $T^{(0)} = (Q^{(0)})^T A Q^{(0)}$ 为经过 $Q^{(0)}$ 变换后的矩阵。标准 QR 迭代基本上由下列步骤组成：

Algorithm 10 标准的 QR 迭代

Require: $T^{(0)} = (Q^{(0)})^T A Q^{(0)}$。若 $Q^{(0)} = \mathbb{1}_{n\times n}$，则 $T^{(0)} = A$。

【计算量】：每次迭代 $O(n^3)$ 计算.

1: **for** $k = 1, 2, \cdots$ **do**

2: $T^{(k-1)}$ 矩阵的 QR 分解：即确定正交矩阵 $Q^{(k)}$ 和一个上三角矩阵 $R^{(k)}$ 使得两者的乘积等于 $T^{(k-1)}$，

$$Q^{(k)} R^{(k)} = T^{(k-1)} , \tag{6.31}$$

3: 令：$T^{(k)} = R^{(k)} Q^{(k)}$。

4: **end for**

[8] 关于这 10 个算法具体内容，有兴趣的同学可以参见：http://www.siam.org/pdf/news/637.pdf。其中一个其实与 QR 算法相关，就是 Householder 约化，我们下面也会涉及。

[9] 关于 QR 算法的发明人的故事，可以参考：G. Golub and F. Uhlig, "The QR algorithm: 50 years later–its genisis by John Francis and Vera Kublanovskaya, and subsequent developments", IMA Journal of Numerical Analysis (2009) 29, 467-485.

注意，在第 k 步 QR 迭代之后，我们有：

$$\begin{aligned} T^{(k)} &= \left(Q^{(k)}\right)^T T^{(k-1)} Q^{(k)} \\ &= \left(Q^{(0)} \cdots Q^{(k)}\right)^T A \left(Q^{(0)} \cdots Q^{(k)}\right) . \end{aligned} \tag{6.32}$$

这个迭代保证了在每一步的新的矩阵都与原先的矩阵只差一个正交变换。因此新的矩阵的条件数一定不坏于原先的矩阵。这一点对于算法的稳定性是非常重要的。

QR 算法的初衷是利用正交变换将 $A \in \mathbb{R}^{n\times n}$ 约化为一个上三角矩阵。不幸的是，这一点并不是总能做到的。因为一个任意的实矩阵有可能有成对的复本征值，但是实的正交矩阵变换后的矩阵元一定仍然保持是实的。如果上述变换总能成立就意味着变换后的三角矩阵的对角元 –也就是它的本征值 – 一定都是实数，与一般实矩阵可以有复本征值矛盾。但是下列定理说明我们起码可以尽可能地将它约化到三角矩阵：

定理 6.4 对于任意的 $A \in \mathbb{R}^{n\times n}$，总是存在一个正交矩阵 $Q \in \mathbb{R}^{n\times n}$ 使得

$$Q^T A Q = \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1m} \\ 0 & R_{22} & \cdots & R_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{mm} \end{bmatrix} , \tag{6.33}$$

其中位于对角线上的块矩阵 R_{ii} 要么就是一个实数要么就是一个 2×2 的、具有一对复共轭根的实矩阵。其中的正交矩阵 Q 可以通过上述的 QR 迭代获得，即:

$$Q = \lim_{k\to\infty} \left[Q^{(0)} Q^{(1)} \cdots Q^{(k)}\right] , \tag{6.34}$$

其中的 $Q^{(k)}$ 就是 QR 迭代算法 10中第 k 步进行 QR 分解 (6.31) 时的正交矩阵 $Q^{(k)}$。

本定理中给出的形式 (6.33) 称为实矩阵 $A \in \mathbb{R}^{n\times n}$ 的 实舒尔分解 (real Schur decomposition) 或者 实舒尔形式 (real Schur form)。需要提请大家注意的是，对一个实矩阵 $A \in \mathbb{R}^{n\times n}$ 来说，它的实舒尔形式与前面定理 6.1提及的矩阵的舒尔形式是有区别的。舒尔形式是 完全上三角 的形式，其对角元就是原矩阵的本征值；实舒尔形式则仅仅是块上三角矩阵，它可以允许 2×2 块矩阵 R_{ii} 的存在，而这些 2×2 的块矩阵包含了原先矩阵的复共轭本征值。

与公式 (6.32) 比较我们发现，它其实就是 QR 迭代中矩阵 $T^{(k)}$ 在 $k\to\infty$ 时的结果。因此，要获得一个实矩阵的实舒尔分解，我们只需要持续不断地进行 QR 迭代即可。我们下面要讲述的其实就是如何有效地进行 QR 迭代，其中最重要的步骤是 QR 分解 (6.31)。一旦获得了矩阵 $A \in \mathbb{R}^{n\times n}$ 的实舒尔分解，我们可以比较容易地获得它的所有本征值并且可以比较简单地计算其本征矢量（如果需要的话）。

上述定理说明，只要我们不断进行 QR 迭代，最后矩阵就会被约化为准三角形式 (6.33)，但是并没有告诉我们收敛的速度有多快。下面的这个定理部分地回答了这个问题。

定理 6.5 给定 $A \in \mathbb{R}^{n\times n}$ 并假设它的本征值的排序为

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n| \,, \tag{6.35}$$

那么我们有

$$\lim_{k\to\infty} T^{(k)} = \begin{bmatrix} \lambda_1 & t_{12} & \cdots & t_{1n} \\ 0 & \lambda_2 & t_{23} & \cdots \\ \vdots & \vdots & \ddots & \ddots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} , \tag{6.36}$$

其中的非对角元的收敛速度为:

$$|t_{i,j}^{(k)}| \sim \mathcal{O}\left(\left| \frac{\lambda_i}{\lambda_j} \right|^k \right) \,, \quad i > j, \;\; k \to \infty \,. \tag{6.37}$$

这个定理告诉我们，如果矩阵的所有本征值的模都不相等，那么经过 QR 迭代，我们不仅仅会获得矩阵的实舒尔形式，我们甚至可以直接获得其舒尔形式 –即所有的本征值。同时我们也看到，如果矩阵具有两个模十分接近的本征值，那么相应的非对角元可以收敛得很慢。当矩阵具有完全等模的本征值时，QR 迭代甚至可能不收敛。

¶ 读者也许会好奇为什么我们需要无穷的迭代解法。也就是说，是否可以找到一个求一般矩阵的本征值的“直接解法”呢？很不幸的是，这一点原则上是不可能的。原因就在于一个矩阵的本征值本质上是通过解其相应的特征多项式的根获得的，尽管数值上我们并不一定真的这样实行。假设存在一个有限步骤的所谓“直接算法”，即通过有限多次的初等代数运算我们就可以获得一个一般的矩阵的本征值，这意味着会存在一个任意阶多项式求根的公式，而这是与著名的 Abel-Ruffini 定理 矛盾的。该定理告诉我们，对于一般的 $n \geq 5$ 的多项式，并不存在一般的利用初等代数的求根公式。因此，一个矩阵求本征值的问题一般来说需要无穷次的迭代来近似地获得 (当然，非常特殊的矩阵除外)。这就是为什么矩阵本征值的计算与相应的线性方程组的求解是不同的：后者存在着直接的解法 (例如我们前面提及的高斯消元法)，当然也可以寻求近似的迭代解法 (这往往出现在矩阵本身是特别大的稀疏矩阵的情形)，但前者一般来说只能够通过迭代方法来近似求解。

¶ 最为基本的 QR 算法可以从 $Q^{(0)} = \mathbb{1}_{n\times n}$ 开始。在这种情形下 $T^{(0)} = A$ 就是原始的矩阵。一个矩阵的 QR 分解 –即公式 (6.31)–可以利用标准的 Gram-Schmidt 正交化 方法获得，其计算量大约为 $O(n^3)$，因此这个算法在每一次迭代的基本计算量也是 $O(n^3)$。[10] 由于总的计算量还要在乘以一个迭代的次数 N_{iter}，而 N_{iter} 并不直接与 n 有关，而是联系到矩阵的本征值的分布。因此，有必要尽可能减少每次 QR 迭代中的计算量。这就是下面我们要讨论的 Hessenberg-QR 算法。所谓的 下/上 Hessenberg 矩阵 是具有下列形式的 $m \times n$

[10] 对于 $A \in \mathbb{R}^{n\times n}$，我们可以从 A 的各个线性独立的列矢量出发，构造一系列正交归一的矢量基，这就是标准的 Gram-Schimidt 正交化方法。它的数值稳定性没有我们下面讨论的 Householder 方法好。

矩阵：[11]

$$H=\begin{bmatrix} h_{11} & h_{12} & & 0 \\ h_{21} & h_{22} & \ddots & \\ \vdots & & \ddots & h_{m-1,n} \\ h_{m1} & \cdots & \cdots & h_{mn} \end{bmatrix}, \quad H=\begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1n} \\ h_{21} & h_{22} & & h_{2n} \\ & \ddots & \ddots & \vdots \\ 0 & & h_{m,n-1} & h_{mn} \end{bmatrix}. \tag{6.38}$$

也就是说上/下 Hessenberg 矩阵就是下/上副对角线以下/上的矩阵元全为零的矩阵。具体来说，如果矩阵 A 是一个上 Hessenberg 矩阵，那么每次 QR 迭代的计算量可以从 $O(n^3)$ 减少到 $O(n^2)$。最为重要的一点是，QR 算法 10中给出的 QR 迭代会保持矩阵的上 Hessenberg 特性。换句话说，如果某个 $T^{(k_0)}$ 是一个上 Hessenberg 矩阵，那么以后的 $T^{(k>k_0)}$ 也一定会保持上 Hessenberg 矩阵的形式。当然，作为代价我们必须首先将一个一般的矩阵 A 化为上 Hessenberg 的形式。我们会看到，这大约需要 $O(n^3)$ 计算量。但是其优点是，以后的每一次 QR 迭代中，我们只需要付出 $O(n^2)$ 的计算量。所以，如果需要进行 QR 迭代的次数非常大 –这往往出现在矩阵的本征值的模非常接近的时候 –在开始 QR 迭代之前做这样预处理是划算的。

21.2 Householder 约化

¶ 考虑 $v\in\mathbb{R}^n$ 中的一个矢量。构造如下的矩阵：

$$P=\mathbb{1}_{n\times n}-2vv^T/||v||_2^2\,, \tag{6.39}$$

其中 $||v||_2^2\equiv v^Tv$ 是矢量 v 的欧氏模方。当我们将这个矩阵乘以 $x\in\mathbb{R}^n$ 时，即 $y=Px$，它的作用将原来的矢量 x 相对于以 v 为法线方向的超平面 –我们记为 $\pi(v)$–进行镜面反射。或者说，x 中与 v 垂直的分量不变，但是与 v 平行的部分反了一个符号 (镜面反射的特点)。这个变换一般称为 Householder 变换；相应的 v 称为 Householder 矢量，P 称为与 v 相对应的 Householder 矩阵。注意，Householder 矩阵并不改变矢量的长度，因此实际上是一个正交矩阵。另一点值得指出的是，当我们需要计算一个 Householder 矩阵乘以一个矢量 (例如 Px) 的时候，我们并不需要首先将 Householder 矩阵存在内存中，然后计算矩阵乘以矢量，我们需要的只是两个矢量的内积 v^Tx，v^Tv 等等。因此仅仅是一个 $O(n)$ 量级的计算而不是通常的 $O(n^2)$ 的计算。[12]

对于任意的 $x\in\mathbb{R}^n$，我们取

$$v=x\pm||x||_2e_m\,, \tag{6.40}$$

其中 e_m 是沿着第 m 个方向中的单位矢量。现在我们按照公式 (6.39) 构造相应的 Householder 矩阵。我们发现矢量 Px 将仅仅包含第 m 个分量，其余分量都是零：

$$Px=[0,\cdots,0,\pm||x||_2,0,\cdots,0]^T\,. \tag{6.41}$$

[11] 我们下面的讨论中基本上仅仅涉及方阵，因此 $m=n$。当然这里给出的是一般的定义。

[12] Householder 变换是 Alston Scott Householder 在 Oak Ridge National Lab 工作期间 (1958 年) 引入的。

如果我们希望对矢量 x 的前 k 个分量保持不变，将从第 $k+2$ 个以及以后的所有分量都设为零。由于 Householder 为正交矩阵不改变矢量的模，因此新的矢量的第 $k+1$ 个分量一定大小为 $x^{(n-k)}$，其中 $x^{(n-k)}$ 为矢量 x 的后 $n-k$ 个分量构成的矢量。对于这个操作我们可以用如下的 Householder 矩阵，

$$P_{(k)} = \begin{bmatrix} \mathbb{1}_k & 0 \\ 0 & R_{n-k} \end{bmatrix} , \; R_{n-k} = \mathbb{1}_{n-k} - 2\frac{w^{(k)}(w^{(k)})^T}{||w^{(k)}||_2^2} , \tag{6.42}$$

其中 $w^{(k)} \in \mathbb{R}^{n-k}$ 为 $\mathbb{R}^{n-k}$ 中的一个矢量。按照公式 (6.40)，我们应当取

$$w^{(k)} = x^{(n-k)} \pm ||x^{(n-k)}||_2 e_1^{(n-k)} , \tag{6.43}$$

其中 $e_1^{(n-k)}$ 是 $\mathbb{R}^{n-k}$ 中的第一个单位矢量。这时我们可以发现，如果令 $y = P_{(k)}x$，那么我们有

$$\begin{cases} y_i = x_i \quad i = 1, 2, \cdots, k \\ y_{k+1} = \pm||x^{(n-k)}||_2 , \\ y_i = 0 \quad i = k+2, \cdots, n \end{cases} \tag{6.44}$$

¶ 利用 Householder 矩阵我们可以将任意的 $n \times n$ 矩阵化为上 Hessenberg 形式。这个步骤一般需要 $O(n^3)$ 的计算操作。[13] 我们下面来说明，利用我们前面定义的矩阵 (6.42) 可以实现这一点。对于任意的 $A \in \mathbb{R}^{n\times n}$，我们可以利用一连串矩阵：$P_{(1)} \cdots P_{(n-2)}$ 试图将矩阵化为上 Hessenberg 形式。事实上，令 $A^{(0)} \equiv A$，对任意的 $k \geq 1$，我们有

$$A^{(k)} = P_{(k)}^T A^{(k-1)} P_{(k)} = \left(P_{(k)} \cdots P_{(1)}\right)^T A \left(P_{(k)} \cdots P_{(1)}\right) \tag{6.45}$$

或者等价地写为 $A^{(k)} = Q_{(k)}^T A^{(k)} Q_{(k)}$，其中正交矩阵 $Q_{(k)} = P_{(k)} \cdots P_{(1)}$ 为一系列 Householder 矩阵的乘积。基本上经过第一次变换：$A^{(1)} = P_{(1)}^T A^{(0)} P_{(1)}$，原先矩阵的第一列中 a_{21} 以下的数都会被变换为零；经过第二个变换，在保持第一列的结构的同时，将矩阵的第二列的 a_{32} 以下的矩阵元都变换为零，等等。利用 Householder 矩阵将矩阵变换为 Hessenberg 形式的操作一般称为 Hessenberg-Householder 约化 (Hessenberg-Householder reduction)。下式以 4×4 矩阵为例，演示了这种约化的过程：

$$\begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} \xrightarrow{P_{(1)}} \begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \end{bmatrix} \xrightarrow{P_{(2)}} \begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \\ 0 & 0 & \bullet & \bullet \end{bmatrix} . \tag{6.46}$$

¶ 值得提及的一点是，如果原先的矩阵 A 是一个对称矩阵，那么 Householder 变换后其对称性仍然保持。因此，在经过 Householder 约化之后，一个对称矩阵一定能够化为一个对称的、三对角矩阵。[14]

[13] 对于一个任意的 $A \in \mathbb{C}^{m\times n}$，基本上在第 k 次迭代的时候，你需要完成的主要计算量为 $4(m-k)(n-k)$。因此，总的计算量大约为 $\sum_{k=1}^{n} 4(m-k)(n-k) \approx 2mn^2 - 2n^3/3$。对于方阵 $m = n$，因此 Householder 约化的计算代价为 $4n^3/3 \approx O(n^3)$。

[14] 所谓三对角矩阵 (tridiagonal matrix) 是指其非零矩阵元都位于对角线和两个副对角线上。

21.3 Hessenberg-QR 算法

¶ 前一小节我们说明了如何利用 Hessenberg-Householder 约化将任意的矩阵 $A \in \mathbb{R}^{n\times n}$ 化为上 Hessenberg 形式。本小节中我们来讨论如何对一个上 Hessenberg 矩阵进行有效的 QR 分解。然后我们就可以构建标准的 Hessenberg-QR 算法了。

这时我们可以利用所谓的 Givens 变换矩阵 (又称为 Givens 转动矩阵)，这其实就是二维转动的推广。对于一对不相等的整数指标 (i,k)，我们定义矩阵 $G(i,k,\theta)$，其 (i_1,i_2) 矩阵元如下:

$$G(i,k,\theta)_{i_1,i_2} = \begin{cases} \cos\theta & i_1 = i_2 = i, \text{or } i_1 = i_2 = k \\ 1 & i_1 = i_2 \neq i \neq k \\ \sin\theta & i_1 = i, i_2 = k \\ -\sin\theta & i_1 = k, i_2 = i \\ 0 & \text{other cases} \end{cases} \tag{6.47}$$

容易验证，如果我们进行定义 $y = G(i,k,\theta)^T x$，那么矢量 y 的各个分量为，

$$y_j = \begin{cases} x_j & j \neq i,k, \\ x_i\cos\theta - x_k\sin\theta & j = i, \\ x_i\sin\theta + x_k\cos\theta & j = k. \end{cases} \tag{6.48}$$

换句话说，Givens 矩阵 $G(i,k,\theta)$ 实际上是在第 i 个坐标和第 k 个坐标构成的平面里面进行了一个转动，转角为 θ。[15] 如果矢量 x 的第 i 和第 k 分量分别为 x_i，x_k，我们可以选取 θ，使得 y_i 或者 y_k 为零。

例如，一个在第一、第二轴的平面内转动的 Givens 矩阵的形式为：

$$G(1,2,\theta) = \begin{bmatrix} \hat{G}(\theta) & 0 \\ 0 & \mathbb{1}_{(n-2)\times(n-2)} \end{bmatrix}, \tag{6.49}$$

其中的 2×2 矩阵 $\hat{G}(\theta)$ 为

$$\hat{G}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}. \tag{6.50}$$

现在我们考虑一个任意的上 Hessenberg 矩阵 $H^{(0)}$：

$$H^{(0)} = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \cdots & \bullet \\ \bullet & \bullet & \bullet & \bullet & \cdots & \bullet \\ 0 & \bullet & \bullet & \bullet & \cdots & \bullet \\ 0 & \vdots & \bullet & \cdots & \cdots & \vdots \\ 0 & \vdots & \cdots & \cdots & \cdots & \bullet \\ 0 & \cdots & \cdots & \cdots & \cdots & \bullet \end{bmatrix} \tag{6.51}$$

我们现在希望将它进行 QR 分解。这可以通过以下步骤来实现：

[15] Givens 转动是美国数学家兼计算机学家 Wallace Givens 在上世纪五十年代在 Argonne National Lab 工作期间引入的。

1. 选择一个形式如 (6.49) 的 Givens 转动矩阵 $G(1,2,\theta_1)^{(1)} \equiv G_1^{(1)}$ 作用于 $H^{(0)}$，使得 $\left(G_1^{(1)}\right)^T H^{(0)}$ 的第 $(2,1)$ 元素为零。我们可以调整参数 θ_1 使得原先的 $H^{(0)}$ 中左上角的 2×2 矩阵 –图中用红色标出的部分 –的左下角矩阵元为零。这样左乘之后我们得到：

$$\left(G_1^{(1)}\right)^T H^{(0)} = \begin{bmatrix} \times & \times & \times & \times & \cdots & \times \\ 0 & \times & \times & \times & \cdots & \times \\ 0 & \bullet & \bullet & \bullet & \cdots & \bullet \\ 0 & \vdots & \bullet & \cdots & \cdots & \vdots \\ 0 & \vdots & \cdots & \cdots & \cdots & \bullet \\ 0 & \cdots & \cdots & \cdots & \cdots & \bullet \end{bmatrix} \tag{6.52}$$

其中矩阵的前两行是与原先矩阵比较发生了变化的部分，同时我们用 $\times$ 表示它是与原先的 $\bullet$ 不同的 (一般来说非零的) 矩阵元。

2. 选择第二个 Givens 转动矩阵 $G(2,3,\theta_2)^{(1)} \equiv G_2^{(1)}$ 作用于上部得到的矩阵之上，只不过它的作用的目的是上面矩阵中蓝色的 2×2 的块。这样的矩阵的形式为：

$$G(2,3,\theta_2) = G_2^{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \hat{G}(\theta_2) & 0 \\ 0 & 0 & \mathbb{1}_{(n-3)\times(n-3)} \end{bmatrix}, \tag{6.53}$$

我们可以调节 θ_2 使得原矩阵的 $(3,2)$ 矩阵元 –也就是蓝色 2×2 块矩阵的左下角矩阵元为零。这时我们得到的矩阵看起来是如下的模样：

$$\left(G_2^{(1)}\right)^T \left(G_1^{(1)}\right)^T H^{(0)} = \begin{bmatrix} \times & \times & \times & \times & \cdots & \times \\ 0 & \blacksquare & \blacksquare & \blacksquare & \cdots & \blacksquare \\ 0 & 0 & \blacksquare & \blacksquare & \cdots & \blacksquare \\ 0 & \vdots & \bullet & \cdots & \cdots & \vdots \\ 0 & \vdots & \cdots & \cdots & \cdots & \bullet \\ 0 & \cdots & \cdots & \cdots & \cdots & \bullet \end{bmatrix} \tag{6.54}$$

3. 继续上述步骤，我们用 $n-1$ 个 Givens 转动获得矩阵 $\left(G_{n-1}^{(1)}\right)^T \cdots \left(G_1^{(1)}\right)^T H^{(0)}$，我们调节每个 Givens 矩阵的转角使得最终的矩阵变为上三角矩阵：

$$(Q^{(1)})^T H^{(0)} \equiv \left(G_{n-1}^{(1)}\right)^T \cdots \left(G_1^{(1)}\right)^T H^{(0)} = R^{(1)} . \tag{6.55}$$

其中 $R^{(1)}$ 是一个上三角矩阵。

4. 我们可以完成另一半的变换，即定义：

$$H^{(1)} \equiv R^{(1)} Q^{(1)} = R^{(1)} \left(G_1^{(1)} \cdots G_{n-1}^{(1)}\right) . \tag{6.56}$$

这就完成了一次 QR 迭代。大家容易验证，矩阵 $H^{(1)}$ 与原先的 $H^{(0)}$ 一样，仍然具有上 Hessenberg 形式。[16]

5. 将上述步骤里面所有矩阵的上标中的 (1) 和 (0) 都换成 (k) 和 $(k-1)$，对任意的 $k \geq 1$，我们可以重复上述步骤继续 QR 迭代：

$$(Q^{(k)})^T H^{(k-1)} = \left(G_{n-1}^{(k)}\right)^T \cdots \left(G_1^{(k)}\right)^T H^{(k-1)} = R^{(k)} \,, \tag{6.57}$$

这大致需要 $O(3n^2)$ 的计算量而另一半的计算

$$H^{(k)} = R^{(k)} Q^{(k)} = R^{(k)} \left(G_1^{(k)} \cdots G_{n-1}^{(k)}\right) \,, \tag{6.58}$$

也需要 $O(3n^2)$ 的计算量。因此，以 $O(6n^2)$ 的计算量为代价，我们可以完成一个上 Hessenberg 矩阵的一次 QR 迭代。

现在我们可以将上述步骤与前面的算法 10 结合，就可以完成对任意一个矩阵 $A \in \mathbb{R}^{n\times n}$ 的 QR 迭代算法：首先我们将矩阵 A 化为上 Hessenberg 形式 (代价 $O(n^3)$)；然后我们对上 Hessenberg 形式的矩阵进行 QR 分解 (代价 $O(n^2)$) 从而完成基本的 QR 迭代。这就形成了标准的 Hessenberg-QR 算法：

Algorithm 11 Hessenberg-QR 算法

Require: 对任意的 $A \in \mathbb{R}^{n\times n}$，首先将其化为上 Hessenberg 形式，然后进行 QR 迭代

【计算量】：每次迭代 $O(6n^2)$ 计算量

1: 利用一系列 $(n-2$ 个)Householder 约化，将 A 化为上 Hessenberg 形式：$T^{(0)} = Q^T A Q$，其中 $Q = P_{(n-2)} \cdots P_{(1)}$。计算代价为 $O(n^3)$。

2: **for** $k = 1, 2, \cdots$ **do**

3: 利用一系列 $(n-1$ 个)Givens 变换矩阵 $G_{1\leq j\leq n-1}^{(k)}$，对 $T^{(k-1)}$ 矩阵进行 QR 分解：

$$\left(G_{n-1}^{(k)}\right)^T \cdots \left(G_1^{(k)}\right)^T T^{(k-1)} = R^{(k)} \,, \tag{6.59}$$

其中 $R^{(k)}$ 为上三角矩阵 (计算量 $3n^2$)。

4: 令：$T^{(k)} = R^{(k)} Q^{(k)}$(计算量 $3n^2$)。

5: **end for**

¶ 如果仅仅需要对一个矩阵进行 QR 分解，即将 $A = QR$ 的两个因子 Q 和 R 求出的话，原则上我们可以利用 Householder，或者利用 Givens 转动，或者利用所谓的 Gram-Schmidt 正交化。但是如果我们需要的是求矩阵的本征值，那么我们需要的是对原

[16] 非常类似于上面的讨论，现在我们右乘以 $G_1^{(1)}$ 的时候，上三角矩阵 $R^{(1)}$ 的矩阵元中只有最左边的两列会发生改变；当再右乘以 $G_2^{(1)}$ 的时候，相应矩阵的从左边数的第二、第三列会发生变化，等等。这样一来最终的 $H^{(1)}$ 又成为了上 Hessenberg 矩阵。

来的矩阵进行相似变换，这时候仅仅用有限次的迭代就无法完成了。例如，我们可以不停地将合适的 Householder 矩阵左乘上原先的矩阵，这样是可以将其化为上三角形式的矩阵 R 的，但是如果我们要求是相似变换，那么我们还需要在矩阵的右边乘以 Householder 矩阵。这样一来我们发现，最多我们只能够将它相似变换为一个上 Hessenberg 形式而不可能是上三角形式。

这就回溯到我们是需要求解方程 $Ax = b$ 还是要求矩阵 A 的本征谱。对前者而言，我们只需要对矩阵 A 进行 QR 解。事实上，如果我们知道 $A = QR$，那么方程 $Ax = b$ 即等价为 $QRx = b$，两边乘以 Q^T 我们得到 $Rx = Q^T b$。这是一个三角矩阵的求解问题，可以通过前面提及的反代法求出解。

要对 A 进行 QR 分解我们可以对矩阵 A 不停地左乘以适当的 Householder 矩阵即可。注意到

$$P \cdot A = P \cdot [a_1^{(0)}, a_2^{(0)}, \cdots, a_n^{(0)}] = [Pa_1^{(0)}, Pa_2^{(0)}, \cdots, Pa_n^{(0)}] \,, \tag{6.60}$$

其中 $a_i^{(0)}$，$i = 1, \cdots, n$ 为矩阵 A 的各个列所对应的矢量。因此我们可以首先选取第一个 Householder 矩阵 P_1 使得，

$$P_1 a_1^{(0)} = k e_1 \,, \tag{6.61}$$

其中 $k = \pm ||a_1^{(0)}||_2$，$e_1 = (1, 0, \cdots, 0)^T$ 是 $\mathbb{R}^n$ 中的第一个单位矢量。这样一来，矩阵 $P_1 A = [a_1^{(1)}, a_2^{(1)}, \cdots, a_n^{(1)}]$ 的各个列看起来是这样的样子：$a_1^{(1)} = (\pm ||a_1^{(0)}||, 0, 0, \cdots, 0)^T$，而其余的各个列则没有什么特别的。换句话说，通过左乘上一个 Householder 矩阵，我们将所得到的新的矩阵的第一列元素从第二行一直到第 n 行都设置为零了。我们可以将矩阵写为分块的形式：

$$P_1 A = \begin{bmatrix} \pm ||a_1^{(0)}|| & \alpha^T \\ 0 & \bar{A} \end{bmatrix} \,. \tag{6.62}$$

其中 α 是一个一般的 $n-1$ 个分量的矢量而 $\bar{A} \in \mathbb{R}^{(n-1)\times(n-1)}$ 则是一个一般的矩阵。

第二步中我们寻求如下形式的矩阵：

$$P_2 = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_2 \end{bmatrix} \,, \tag{6.63}$$

其中 $\tilde{P}_2 \in \mathbb{R}^{(n-1)\times(n-1)}$ 是一个恰当的 $(n-1)\times(n-1)$ Householder 矩阵。于是我们有，

$$P_2 P_1 A = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_2 \end{bmatrix} \begin{bmatrix} \pm ||a_1^{(0)}|| & \alpha^T \\ 0 & \bar{A} \end{bmatrix} = \begin{bmatrix} \pm ||a_1^{(0)}|| & \alpha^T \\ 0 & \tilde{P}_2 \bar{A} \end{bmatrix} \,. \tag{6.64}$$

于是化为上三角矩阵的问题化为阶数减少一的同样问题。我们要做的是根据矩阵 $\bar{A}$ 的各个列适当地选择 $(n-1)\times(n-1)$ 的 Householder 矩阵 $\tilde{P}_2$，它可以将 $\bar{A}$ 的 (11) 元素一下的各个元素设为零。最终，通过类似的 $n-1$ 个恰当的 Householder 矩阵，我们有

$$P_{n-1} \cdots P_2 P_1 A \equiv Q^T A = R \,, \tag{6.65}$$

其中 R 是一个上三角矩阵。由于各个 P_i 都是 Householder 矩阵从而是正交矩阵，因此它们的乘积仍然是一个正交矩阵，记为 Q^T。于是 $A = QR$ 就完成了矩阵 A 的 QR 分解。这个方法所耗费的计算量大约是 $(2/3)n^3$。

¶ 作为一个例子，考虑下面具体的 5×5 并且已经是上 Hessenberg 形式的实矩阵：

$$A = \begin{bmatrix} 3 & 17 & -37 & 18 & -40 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} . \tag{6.66}$$

已知它的本征值为：-4，$\pm i$，2 和 5。我们将这个矩阵送入算法 11并迭代 40 次后得到：

$$T^{(40)} = \begin{bmatrix} 4.9997 & 18.9739 & -34.2570 & 32.8760 & -28.4604 \\ 0 & -3.9997 & 6.7693 & -6.4968 & 5.6216 \\ 0 & 0 & 2 & -1.4557 & 1.1562 \\ 0 & 0 & 0 & 0.3129 & -0.8709 \\ 0 & 0 & 0 & 1.2607 & -0.3129 \end{bmatrix} . \tag{6.67}$$

我们发现，这个矩阵基本上已经收敛到了原矩阵的实舒尔形式。前三个对角元对应于原矩阵的三个实本征值，最后的一个 2×2 的对角块的本征值也近似为 $\pm i$。注意，按照定理 6.5，这一点并不是一定有保证的。在这里之所以达成了是因为该矩阵只有一对复共轭的根并且其他几个实根的模都不相等。如果存在两个或多个模相等的实根，那么 QR 迭代并不收敛到矩阵的实舒尔形式。例如，对于

$$A = \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix} . \tag{6.68}$$

已知它的本征值为：65，± 21.28，± 13.13。在将其化为上 Hessenberg 矩阵并迭代 40 次后我们得到的是

$$T^{(40)} = \begin{bmatrix} 65 & 0 & 0 & 0 & 0 \\ 0 & 14.67 & 14.24 & 4.48 & -3.44 \\ 0 & 16.67 & -14.67 & -1.22 & 2.04 \\ 0 & 0 & 0 & -13.03 & -0.76 \\ 0 & 0 & 0 & -3.32 & 13.03 \end{bmatrix} . \tag{6.69}$$

我们看到它并没有趋于完全的上三角矩阵，而是趋于块上三角矩阵。这个形式其实也并不是定理 6.4所预言的矩阵 A 的标准实舒尔形式，而是它的一个变种。[17] 最上面的 1×1 的

[17]定理 6.4说的 2×2 的实矩阵仅仅出现在矩阵出现一对复共轭的根的情形下。这里的根都是实根，只不过它们具有相同的模。于是 QR 迭代不收敛。

块恰好包含了本征值 65，另外两个 2×2 的块矩阵则分别包含了 ± 21.28 和 ± 13.13。没有收敛的原因就在于它具有模相同的本征值。要确保 QR 迭代收敛到矩阵的实舒尔形式需要用到我们下小节要讨论的 shift 技术。

21.4 Shifted QR

¶前面讨论的标准的 QR 算法对多数实矩阵都可以很好的工作。但是，按照前面的定理 6.5，如果矩阵有若干个非常接近的实本征值，或者有成对的复本征值时可能会收敛得很慢。一个通常的做法是利用 shift 的技术。它实际上等价于将原先的矩阵加上一个正比于单位矩阵的矩阵。

我们将前面的标准 QR 算法 10更改为如下的、具有单一 shift 的 QR 算法：

Algorithm 12 单一 shift 的 QR 迭代

Require: $T^{(0)}=(Q^{(0)})^T AQ^{(0)}$ 已经具有 Hessenberg 形式。

1: **for** $k=1,2,\cdots$ **do**

2: $T^{(k-1)}$ 矩阵的 shifted-QR 分解：

$$Q^{(k)}R^{(k)}=T^{(k-1)}-\mu\mathbb{1}_{n\times n}\,, \tag{6.70}$$

其中 $\mu\in\mathbb{R}$ 称为该算法的 shift。

3: 令：$T^{(k)}=R^{(k)}Q^{(k)}+\mu\mathbb{1}_{n\times n}$。

4: **end for**

前面提到，对于具有模非常接近的本征值的时候，QR 迭代很可能收敛得很慢甚至根本不收敛于矩阵的实舒尔形式。为了克服这一点，我们可以参用上述具有单个 shift 的 QR 迭代。按照前面的定理 6.5，我们应当选择 $\mu\in\mathbb{R}$ 使得

$$|\lambda_1-\mu|\geq|\lambda_2-\mu|\geq\cdots\geq|\lambda_n-\mu|\,. \tag{6.71}$$

这样一来变换过程中第 k 次迭代时的非对角元 $t^{(k)}_{j,j-1}$ 大致会按照 $|(\lambda_j-\mu)/(\lambda_{j-1}-\mu)|^k$ 的方式趋于零。因此，一个比较自然的选择是令：

$$|\lambda_n-\mu|<|\lambda_i-\mu|\,,\quad i=1,\cdots,n-1\,, \tag{6.72}$$

那么矩阵元 $t^{(k)}_{n,n-1}$ 会快速地趋于零。在实际的应用中，人们一般会取

$$\mu=t^{(k)}_{n,n}\,. \tag{6.73}$$

在这个选择下往往非对角元 $t^{(k)}_{n,n-1}$ 会快速趋于零。在具体操作中，我们还可以监控 $t^{(k)}_{n,n-1}$ 的大小。如果一旦

$$|t^{(k)}_{n,n-1}|\leq\varepsilon\left(|t^{(k)}_{n-1,n-1}|+|t^{(k)}_{n,n}|\right)\,,k\geq 0 \tag{6.74}$$

我们就可以直接将 $t^{(k)}_{n,n-1}$ 设为零，其中 ε 为相应机器的精度。这时的对角元 $t^{(k)}_{n,n}$ 其实就近似地给出原来矩阵的一个本征值。

利用上面提到的这个算法我们可以将前面提到的矩阵 (6.68) 成功地化为其实舒尔形式：

$$T^{(40)} = \begin{bmatrix} 65 & 0 & 0 & 0 & 0 \\ 0 & -21.278 & 2.59 & -0.045 & -4.296 \\ 0 & 0 & -13.126 & -4.029 & -13.079 \\ 0 & 0 & 0 & 21.278 & -2.620 \\ 0 & 0 & 0 & 0 & 13.126 \end{bmatrix} . \tag{6.75}$$

事实上，仅仅需要大约十几次迭代就可以得到这个结果。

¶ 最后我们指出，对于仅仅有实数本征值的实矩阵来说，一般利用 single shift 技术就可以比较好地解决 QR 迭代的收敛问题。我们一般总是可以获得矩阵的实舒尔形式 –也就是定理 6.4所预言的块上三角形式。但是要获得矩阵的舒尔形式 – 也就是定理 6.1所给出的标准的上三角形式 (从而可以直接从对角元读出其本征值) –还需要所谓的 double shift 的算法。事实上，J. Francis 有一个非常优美的所谓 Implicit-QR 算法 (又被称为 Francis 算法)，它可以等效地引入一对复共轭的 shift，但是仍然保持在实数域内进行运算。 关于这个算法的介绍我们这里就不再深入了。有兴趣的同学可以参考 [2] 中的 §6.6.5 小节或者参考 D.S. Watkins 的文章。[18] 特别值得提及的是，你如果使用 MATLAB 求一个实矩阵的本征值，非常可能它实际上用的就是这个算法。

21.5 由矩阵的实舒尔形式计算其本征矢

¶ 假设通过前面讲述的 QR 算法我们已经得到了矩阵的实舒尔形式，$Q^T A Q = T$，那么 T 的对角元就是原先矩阵的本征值 (对于出现 2×2 包含复共轭根的情况除外)。假定对于某个本征值 λ，我们希望求解其本征矢 x。容易验明，若 $Ax = \lambda x$，则令 $y = Q^T x$，我们一定有 $Ty = \lambda y$。因此，为了求出 x 我们可以直接先求 y，然后再以 Q 左乘之即可。我们令 $\lambda = t_{kk} \in \mathbb{R}$ 为 A 的某个单一本征值。那么矩阵 T 的形式一定为：

$$T = \begin{bmatrix} T_{11} & v & T_{13} \\ 0 & \lambda & w^T \\ 0 & 0 & T_{33} \end{bmatrix} , \tag{6.76}$$

其中 $T_{11} \in \mathbb{R}^{(k-1)\times(k-1)}$，$T_{33} \in \mathbb{R}^{(n-k)\times(n-k)}$ 是两个上三角矩阵；$v \in \mathbb{R}^{k-1}$，$w \in \mathbb{R}^{n-k}$ 为两个矢量。按照假定，λ 是一个单一的本征值，因此它一定不会出现在 T_{11} 或者 T_{33} 的谱中。这意味着矩阵 $(T_{11} - \lambda \mathbb{1}_{(k-1)\times(k-1)})$ 和 $[T_{33} - \lambda \mathbb{1}_{(n-k)\times(n-k)}]$ 都是非奇异的上三角矩阵。我们假定 $y = (y^T_{k-1}, y', y^T_{n-k})^T$，其中 $y_{k-1} \in \mathbb{C}^{k-1}$，$y_{n-k} \in \mathbb{C}^{n-k}$，于是本征方程

[18] David S. Watkins "Francis' s Algorithm", American Mathematical Monthly 118, 05, 387 (2011).

$(T-\lambda\mathbb{1})y=0$ 可以明确地写为：

$$\begin{cases}[T_{11}-\lambda\mathbb{1}_{(k-1)\times(k-1)}]y_{k-1}+yv+T_{13}y_{n-k}=0\\ w^T y_{n-k}=0\\ [T_{33}-\lambda\mathbb{1}_{(n-k)\times(n-k)}]y_{n-k}=0\end{cases} \tag{6.77}$$

由于 $[T_{11}-\lambda\mathbb{1}_{(k-1)\times(k-1)}]$ 和 $[T_{33}-\lambda\mathbb{1}_{(n-k)\times(n-k)}]$ 均是非奇异的，因此上述方程可以“解出”(不失一般性，我们可以令 y’ $=1$) 如下的解：

$$y=\begin{pmatrix}-[T_{11}-\lambda\mathbb{1}_{(k-1)\times(k-1)}]^{-1}v\\ 1\\ 0\end{pmatrix}. \tag{6.78}$$

最终，原来矩阵 A 的本征矢 x 可以由 $x=Qy$ 给出。

22 赝逆与奇异值分解

22.1 奇异值分解的理论基础

¶奇异值分解是一项应用十分广泛的计算技术。首先让我们回忆这个线性代数中十分重要的定理。

定理 6.6 对于一个 $m\times n$ 的复矩阵 $A\in\mathbb{C}^{m\times n}$，一定存在两个幺正矩阵 $U\in\mathbb{C}^{m\times m}$ 和 $V\in\mathbb{C}^{n\times n}$，它们能够将 A “对角化”：[19]

$$U^\dagger AV=\Sigma=\mathrm{diag}(\sigma_1,\cdots,\sigma_p)\in\mathbb{C}^{m\times n}\ ,\quad p=\min(m,n)\ . \tag{6.79}$$

其中 $\sigma_1\geq\sigma_2\geq\cdots\geq\sigma_p\geq 0$ 称为矩阵 A 的 奇异值 (singular values) 而上式则称为矩阵 A 的 奇异值分解 (singular value decomposition, SVD)。

矩阵的那些非零的奇异值实际上是矩阵 $A^\dagger A$ 的本征值的根号：

$$\sigma_i(A)=\sqrt{\lambda_i(A^\dagger A)}\ . \tag{6.80}$$

由于 $A^\dagger A$ 以及 $AA^\dagger$ 都是厄米的，因此矩阵 U 的列被称为矩阵 A 的 左奇异矢量；而矩阵 V 的列则称为 A 的 右奇异矢量。特别地，若 A 的奇异值满足

$$\sigma_1\geq\cdots\geq\sigma_r>\sigma_{r+1}=\cdots=\sigma_p=0\ , \tag{6.81}$$

[19]对一个长方形的矩阵，其矩阵元如果只有当行列的指标相等时才不为零就称为对角的矩阵。从“几何上”看，非零的对角元并不位于长方形的对角线上。

那么 A 的秩为 r，同时它的核 ker(A) 由 V 的列矢量 $\{v_{r+1},\cdots,v_n\}$ 张成；而 A 的域 $range(A)$ 则由 U 的列矢量 $\{u_1,\cdots,u_r\}$ 张成。

一个矩阵的 SVD 实际上包含了该矩阵的重要信息。虽然上述定理中的两个幺正矩阵并不是唯一的，但是它们实际上几乎是唯一的。所不唯一的一点就是你可以将这些幺正矩阵的不同的列交换一下而已。

上述定理中的结论还可以倒过来写：

$$A = U\Sigma V^\dagger \,. \tag{6.82}$$

我们可以定义一个对角矩阵 Σ_k，它的第 k 个对角元就等于 σ_k，其余的所有矩阵元都恒等于零。于是 $\Sigma \equiv \sum_{k=1}^p \Sigma_k$。同时我们将矩阵 U 按照其列矢量写为 $U=(U_1,\cdots,U_m)$，$V^\dagger$ 写为行矢量，我们就得到：

$$A = \sum_{k=1}^{p} \sigma_k U_k \otimes (V_k)^\dagger \,. \tag{6.83}$$

这个公式是矩阵奇异值分解的另外一种写法。我们下面会看到，它对于讨论的图像的压缩具有重要意义。另外一种写法是将矩阵 A 的特定的矩阵元写出来，

$$A_{ij} = \sum_{k=1}^{p} \sigma_k (u_i)_k (v_j)_k^* \,. \tag{6.84}$$

其中 u_i 表示矩阵 U 的第 i 行 (注意是行而不是列) 的矢量，而 v_j 则表示矩阵 V 的第 j 行的矢量。所以矩阵元 A_{ij} 有点像是两个单位矢量，一个是 U 的第 i 行，另一个是 V 的第 j 行，之间的某种用奇异值进行加权后的内积。由于 U 和 V 的行都是模为 1 的矢量 (因为它们是幺正矩阵)，因此一个矩阵中的“主要部分”就由那些较大的奇异值贡献，奇异值恒等于零的将完全没有贡献，而那些虽然不是零但是非常接近于零的奇异值及其左右奇异矢量的贡献也是比较小的。按照约定，如果我们已经将它们排好次序，那么上式中的前 k 项实际上就构成了矩阵 A 的一个“近似”：

$$A^{(k)} = \sum_{j=1}^{k} \sigma_j U_j \otimes (V_j)^\dagger \,. \tag{6.85}$$

这是一个秩为 k 的矩阵。如果我们给矩阵一个模 (norm) 的定义，比如：

$$||A||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2} \,, \tag{6.86}$$

这个模称为 Frobenius 模或者 Hilbert-Schmidt 模，它的另外一种写法是 $\sqrt{Tr(A^\dagger A)}$。可以证明，在所有的秩为 k 的矩阵中，公式 (6.85) 给出的是 A 的一个最佳的近似，即 $||A^{(k)}-A||_F$ 是最小的。

¶ 与一个矩阵的奇异值分解密切相关的概念就是一个任意矩阵 (甚至不是方阵) 的 赝逆，又称为 Moore-Penrose 赝逆 (Moore-Penrose pseudo-inverse)。对于一个 $A\in\mathbb{C}^{m\times n}$，它的赝逆 $A^+\in\mathbb{C}^{n\times m}$ 满足如下的四个性质：

1. $AA^+A = A$;
2. $A^+AA^+ = A^+$;
3. $(AA^+)^\dagger = AA^+$;
4. $(AA^+)^\dagger = AA^+$

注意上述定义对任意的一个矩阵都成立。如果 A 是满秩的，那么 A^+ 可以利用“传统的”表达式给出。具体来说，

- 如果 A 是列满秩，即 $A^\dagger A$ 是可逆的，我们有：$A^+ = (A^\dagger A)^{-1}A^\dagger$，并且 $A^+A = I$。
- 如果 A 是行满秩，即 $AA^\dagger$ 是可逆的，我们有：$A^+ = A^\dagger(AA^\dagger)^{-1}$，并且 $AA^+ = I$.

22.2 奇异值分解的应用举例

¶奇异值分解的强大之处在于，任何复矩阵都可以这样进行分解！正因为如此，它被广泛地应用在各个领域，其中除了我们熟悉的科技的领域，还包括一些非科技的领域。我们下面举几个例子给大家一定感觉。

- 最为直接的应用是求最小二乘法的解。这个古老的问题最早是高斯首先研究的。这个问题我们在数据分析以及拟合部分还会遇到。具体来说，如果我们需要求解线性系统 $Ax = b$，其中 $A \in \mathbb{R}^{m\times n}$，$x \in \mathbb{R}^n, b \in \mathbb{R}^m$，那么如果 $m > n$ 一般来说是不会有解的 (方程个数大于自变量个数)，我们需要求得是使得 $||Ax - b||_2$ 最小的解 x，这就是所谓的最小平方问题，它广泛地出现在参数拟合之中。可以证明的是，赝逆恰好提供了这个问题的一个完美的解。具体来说，对于任意的 $x \in \mathbb{C}^n$，$A \in \mathbb{C}^{m\times n}$，如果我们令 $z \equiv A^+b$，那么我们有：

$$||Ax - b||_2 \geq ||Az - b||_2 \,, \tag{6.87}$$

其中 $||\cdot||_2$ 表示 $\mathbb{C}^m$ 中的欧氏模。其中的等号成立的条件为：

$$x = A^+b + (\mathbb{1}_{m\times m} - A^+A)w \,, \tag{6.88}$$

其中 $w \in \mathbb{C}^m$ 为任意矢量。如果 A 的列满秩，那么 $(\mathbb{1}_{m\times m} - A^+A) \equiv 0$。

- 图像压缩：一张图片 –无论是黑白的、灰度的、还是彩色的、在计算机中实际上是储存为一个矩阵的。粗略来说，矩阵的每个矩阵元可以对应于图片中的一个像素。这个矩阵可以进行 SVD，因此它的信息很大程度上体现在它的奇异值及其相应的左右奇异矢量中。一种图像压缩的思想就是将一个很大的图像矩阵进行 SVD，然后仅仅保持它的奇异值中比较大的部分 (包括相应的左右奇异矢量)，将其余的奇异值都人为地设为零。再将其带入奇异值分解。我们就得到了一个比原先的图像需要存储空间要小的图像。这种压缩当然会失去一些信息。但是在我们并不需要高质量画质的情况下就完全足够了。它的好处是可以极大地减小原先所需的存储。

- 类似的，这方面的另一个应用是对人脸的图像识别门禁问题。假定我们在数据库中存贮了一系列获准进入门禁的人脸的图像。每个人脸的图像比如说由 $m \times n = M$ 个像素构成。我们将其记为一个矢量 $f_i \in \mathbb{R}^M$，其中 $i = 1, 2, \cdots, N$，标志了第 i 号获准进入门禁的人脸图像。我们可以构建一个平均的人脸 (average face) 如下：

$$\bar{f} = \frac{1}{N} \sum_{i=1}^{N} f_i \, . \tag{6.89}$$

然后我们再构建每个人的图像对于这个平均人脸的偏离，记为 $\delta f_i \in \mathbb{R}^M$。以这些 δf_i 为列，可以构成一个 $M \times N$ 的矩阵：

$$A = [\delta f_1, \delta f_2, \cdots, \delta f_N] \, , \tag{6.90}$$

我们现在可以对这个矩阵进行奇异值分解。[20]

$$A = \sum_{i=1}^{K} \sigma_i u_i \otimes (v_i)^T \, , \quad K = \min(M, N) \, . \tag{6.91}$$

由于各个左奇异矢 $[u_1, \cdots]$ 构成了空间 $\mathbb{R}^M$ 中的正交归一的矢量基，我们可以选取一个合适的正整数 p 并存储小于它的各个左奇异矢 $u_{i \le p}$。我们可以进一步定义一个任意的待测人脸图像 $f \in \mathbb{R}^M$ 在各个左奇异矢 $[u_1, \cdots, u_p]$ 张成的空间中的坐标，

$$x = [u_1, \cdots, u_p]^T \cdot \delta f \, . \tag{6.92}$$

而原先的被允许进入门禁的各个人脸的坐标则分别为 x_i：

$$x_i = [u_1, \cdots, u_p]^T \cdot \delta f_i \, , \tag{6.93}$$

其中 $\delta f_i = f_i - \bar{f}$。我们现在需要的就是判别这个坐标 x 是否与某个 x_i 足够接近。

现在我们选取某个正定的小的实数 $\epsilon_0 > 0$。如果 $\min_i ||x - x_i||_2 \le \epsilon_0$，我们就认为目前待测的人脸 f 就是 f_i，其中 i 就是使得 $||x - x_i||_i$ 最小的那个 i，同时允许其进入门禁；而如果对所有的 i 来说，$||x - x_i|| > \epsilon_0$，我们就认为目前待测的人脸 f 不属于任何被允许进入的人并拒绝其申请。显然，参数 ϵ_0 的选取是需要稍微调节一下的。一方面它必须足够的小，起码能够有效地区分已经被允许进入门禁的不同的人。也就是说，它只少应当满足 $\epsilon_0 \le \min_{i \ne j} ||x_i - x_j||_2$。另一方面 ϵ_0 也不能够太小，因为同一个人的图像与存储的图像之间仍然有细微的差别。我们应该保证被允许进入门禁的人在通常的情况下都可以顺利通过门禁系统。

[20] 用每个人对于一个平均的人脸的偏离来构成这个矩阵是因为可以滤掉一般的人脸中一部分共同的特性 (比如都是一个鼻子一个嘴两个眼睛等等)，如果不减除掉这部分特性，它们在图像上有可能会体现出较大的奇异值从而占用我们需要存储的资源。而滤掉以后再进行奇异值分解，如果我们仅仅储存部分较大的奇异值，则更能够体现不同人脸之间的差异性。

- 推荐系统：这是目前网络营销中最为常用的方法了。当你在某个网站购买了某项物品之后，该网站及其合作商们会根据你的购买向你推荐一系列其他的商品。这样的一个系统就是所谓的推荐系统 (recommender system)。这个问题中 SVD 也可以起到至关重要的作用。

  基本上商家需要构建一个矩阵，它的行列分别由顾客的 ID–c 以及商品的 ID–p 构成。推荐系统的运行依赖于各个顾客 c 对各种商品 p 的评价。这个矩阵包含了我们推荐系统分析的基础。我们姑且称之为 推荐矩阵 。当然，我们都做过顾客，我们一般是懒得评价每种商品的，即使是我们购买过的商品。而如果我们没有购买过某个产品就更不会对该产品进行评价了。一般来说，如果一个顾客对某个产品没有评价，这个位置的矩阵元就被赋值为零。因此这个矩阵实际上是一个很大的稀疏矩阵。

  为了明确起见，下面我们考虑一个简化的模型。我们仍然将矩阵的行和列由指标 c(customer) 和 p(product) 来标记，其中 $c = 1, 2 \cdots, N_c, p = 1, 2, \cdots, N_p$，其中 N_c 和 N_p 分别为总得注册用户人数、和总的产品数目。同时假定这个矩阵的矩阵元只有 0(这是初始的缺省设置) 或者是 1 两个值。具体来说，如果某个用户 c 对于某个产品 p 没有购买过或者没有评价过，我们就设 $a_{cp} = 0$，如果某个用户 c 购买过产品 p 或者给过评价我们就将这个值设为 1。这当然是对具体的情况的一个简化。具体的购买过程中，用户可能购买了一个产品之后对它的评价可以分为几个不同的数值。我们暂时先不考虑这些复杂的情况。在这个简化了的模型中，如果一个用户没有购买过任何产品，它对应于推荐矩阵中一行全是零的零矢量；同样，如果某个产品 p 没有任何顾客购买过，它对应于一个全是零的列矢量。这两类零矢量原则上可以从我们的推荐矩阵中完全删除因为它们对于后续的分析完全没有任何作用。

  假定我们想向用户推荐他未曾购买的商品，我改如何操作呢？这里面包括两方面的考虑：对于那些已经购买过很多商品的老客户来说，通过分析他的购买习惯，我们可以向 TA 推荐与他以往购买过的类似的产品。比较困难的是向那些没怎么购买过商品的用户，该做什么样的推荐呢？这个可以通过下列的考虑来进行。首先按照我们的假设，我们已经剔除了完全没有购买过商品的用户。因此任何一个用户 c，TA 至少购买过一个商品 p。我们可以简单地向他推荐所有产品中最热销的产品；我们可以考察那些购买过和他相同样产品的其他用户，看看他们都购买了什么产品，并推荐他们买的最多的一种或几种商品；我们可以从整个推荐矩阵的统计平均性质出发，向他推荐一种或几种产品。综合考虑以上各种方法的一个简便的计算方法就是利用 SVD。按照公式 (6.84) 我们有，

$$a_{cp} = \sum_{k=1}^{P} \sigma_k (u_c)_k (v_p)_k = 0 \,, \tag{6.94}$$

  因为我们假定了用户 c 并没有购买过商品 p。但是这是对于“所有的”奇异值的贡献的结果。如果我们仅仅考虑其中一小部分最重要的奇异值的贡献，意味着我们仅仅求和到某个 $K < P$，或者等价地说，构建型如公式 (6.85) 的一个矩阵 A 的近似表达，那么一般来说 a_{cp} 并不一定等于零，尽管该用户 c 的确没有购买该商品 p。这些

在严格公式中为零但在近似式中非零的贡献就来源于上面我们提及的各个方面的其他用户的“贡献”。这些数值可以作为向各个用户进行推荐的参考。例如，对于

$$A=\begin{bmatrix}0&1&1&1&0&0&1&0&0\\1&0&1&0&0&1&0&0&0\\0&0&0&0&1&0&0&0&1\\0&1&0&1&0&0&0&1&0\\1&0&0&0&1&0&0&0&0\\0&0&1&0&0&0&0&0&1\\1&0&0&0&0&0&0&0&0\end{bmatrix}. \tag{6.95}$$

我们进行 SVD 后发现，如果按照奇异值的大小排列并且仅仅选取其中 4 个最大的奇异值的贡献，那么近似的矩阵为，

$$A^{(4)}=\begin{bmatrix}0.00&1.00&1.01&1.00&-0.17&0.17&0.63&0.37&0.17\\1.12&0.00&1.02&0.00&-0.02&0.69&0.17&-0.17&-0.03\\0.06&-0.02&0.08&-0.02&0.97&-0.18&-0.05&0.03&0.99\\0.00&1.00&-0.01&1.00&0.18&-0.17&0.37&0.63&-0.17\\1.02&0.02&-0.09&0.02&0.86&0.16&-0.12&0.14&0.16\\-0.11&0.01&0.94&0.01&0.18&0.16&0.22&-0.21&0.88\\0.78&-0.02&0.08&-0.02&0.24&0.26&-0.05&0.03&-0.18\end{bmatrix}. \tag{6.96}$$

考察这个近似的矩阵与原先的矩阵 (6.95) 的差别我们发现，对于每一个用户 c 来说，原先为 1 的那些数值仍然是比较大的。原先为零的那些数值现在并不严格等于零了。话句话说，虽然一个用户 c 并没有选择商品 p，但是如果我们近似地考虑其他用户以及产品的“影响”，他仍然有选择该产品的潜质。因此，我们的推荐的政策可以是：选择每个用户的行中除了他已经购买过的产品之外的赋值最高的那个产品进行推荐。例如，对于上面的 7 个用户，我们将分别推荐产品的编号分别为：8,7,3,8,9,7,6。所以，SVD 可以帮助我们完成所谓的 机器学习 (machine learning) 的步骤，其出发点当然是“训练”它的初始矩阵 (6.95)，所以我们称之为 训练矩阵 (training matrix)。

我们说这些在训练矩阵中为零但在近似式中不为零的矩阵元是考虑了其他用户及产品的综合影响不是没有原因的。它是基于如下的考虑：类似的用户倾向于购买类似的产品；很多人购买的产品也有更大的可能被新的用户购买。记住矩阵 U 和 V 都是正交矩阵。它们的列分别起到了在用户空间以及产品空间的矢量基矢的作用。比如，在用户空间中我们可以定义两个用户的类似度的概念。就是两个用户在各个 u_i 的坐标架中的坐标矢量的内积 (方向余弦)。这个类似度可以在 $[-1,+1]$ 之间。如果两个用户的类似度很高，他们就很可能购买类似的产品。同样的分析也适用于 v_k 张成的矢量空间。由于奇异值的大小体现了不同用户、产品交织的空间中的权重，因此如果我们仅仅选取若干个较大的奇异值的贡献，我们相当于只考虑了整个矩阵的平均的统计性质而部分地忽略了具体用户的需求。当我们将截断 K 调到最大可能

时，它当然完全恢复了具体到一个用户对一个特定产品的情况。换句话说，通过调节参数 K，我们就可以在整体性和个性之间自由地游走。

22.3 奇异值分解的具体算法

¶一个矩阵的奇异值分解的计算一般运用 Golub-Kahan-Reinsch 算法。[21] 不失一般性，我们假定 $A \in \mathbb{R}^{m\times n}$ 且 $m \geq n$，否则我们就计算 A^T 的奇异值分解。奇异值分解分为以下步骤

1. 首先，矩阵 A 先被变换为如下形式

$$U^T A V = \begin{pmatrix} B \\ 0 \end{pmatrix} . \tag{6.97}$$

其中 U 和 V 是两个正交矩阵，[22] $B \in \mathbb{R}^{n\times n}$ 是一个上双对角 (upper bidiagonal) 矩阵。[23] 矩阵 U 和 V 由以下步骤，通过 $n+m-3$ 个 Householder 矩阵 $U_1, \cdots, U_{m-1}$ 和 $V_1, \cdots, V_{n-2}$ 依次产生。首先我们将 $(U_1)^T$ 左乘到 A 上：$A^{(1)} = (U_1)^T A$，使 $A^{(1)}$ 的第一列的第二个矩阵元以下都为零；然后将 V_1 右乘到 $A^{(1)}$ 上得到 $A^{(2)} = A^{(1)} V_1$，使得 $A^{(2)}$ 的第一行的第三个矩阵元右边的全部为零，同时不破坏第一列的那些已经化为零的矩阵元

$$A^{(1)} = U_1^T A = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \end{bmatrix} \Rightarrow A^{(2)} = A^{(1)} V_1 = \begin{bmatrix} \bullet & \bullet & 0 & 0 \\ 0 & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \end{bmatrix} \tag{6.98}$$

2. 第二步，我们可以利用 QR 迭代将上双对角矩阵 B 对角化，即我们会获得两个正交矩阵 W 和 Z 使得

$$W^T B Z = \Sigma = \mathrm{diag}(\sigma_1, \cdots, \sigma_n) , \tag{6.99}$$

其中 $\sigma_i, i = 1, \cdots, n$ 就是矩阵 A 的奇异值。于是矩阵 A 的奇异值分解可以表达为，

$$\mathcal{U}^T A \mathcal{V} = \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} , \tag{6.100}$$

其中的矩阵 $\mathcal{U} = U\mathrm{diag}(W, \mathbb{1}_{(m-n)\times(m-n)})$，$\mathcal{V} = VZ$。

[21] 该算法最初由 G. Golub 和 W. Kahan 于 1965 年发表，后来 G. Golub 和 C. Reinsch 在 1970 年进行了完善。

[22] 我们这里假定矩阵 A 是实的矩阵。对于复矩阵，U 和 V 则替换为相应的幺正矩阵。上式中的 U^T 也应当换为 $U^\dagger$。

[23] 所谓上双对角矩阵是指除了对角元 (对角线上的矩阵元) 和上副对角元 (与对角线平行并紧邻其右上方的对角线上的矩阵元) 不为零，其他矩阵元全等于零的矩阵。类似的，可以定义下双对角矩阵。

Golub-Kahan-Reinsch 算法具有良好的稳定性。这个方法计算奇异值分解的计算量为：$2m^2n+4mn^2+(9/2)n^3$，而如果仅仅需要奇异值的话，计算量可以减小到 $2mn^2-(2/3)n^3$。我们看到，矩阵的奇异值分解总体来说还是一个相当耗时的过程。

23 对称矩阵的算法

23.1 Jacobi 算法

¶ 如果矩阵是实对称的方矩阵，那么我们可以利用下面的一些算法来计算其本征值和本征矢量。最为直接的方法就是 Jabobi 算法，它直接利用 Givens 转动将矩阵的非对角元变换为零。

设 $A^{(0)} \equiv A \in \mathbb{R}^{n\times n}$ 为一实对称矩阵，对一对不相等的指标 p 和 q 满足 $1 \le p < q \le n$，我们将 Givens 矩阵 $G(p,q,\theta) = G_{pq}$ 作用于 $A^{(k-1)}$ 得到新的 $A^{(k)}$：

$$A^{(k)} = (G_{pq})^T A^{(k-1)} G_{pq} \,, \tag{6.101}$$

我们可以选择 θ 使得其矩阵元满足

$$a_{ij}^{(k)} = 0 \,, \text{如果：} (i,j) = (p,q) \,, \tag{6.102}$$

由于 Givens 矩阵的结构，$A^{(k)}$ 的上述关系等价于

$$\begin{bmatrix} a_{pp}^{(k)} & a_{pq}^{(k)} \\ a_{pq}^{(k)} & a_{qq}^{(k)} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a_{pp}^{(k-1)} & a_{pq}^{(k-1)} \\ a_{pq}^{(k-1)} & a_{qq}^{(k-1)} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \tag{6.103}$$

显然，如果 $a_{pq}^{(k-1)} = 0$，那么我们可以选择 $c = 1$，$s = 0$，这时公式 (6.102) 自动成立；如果 $a_{pq}^{(k-1)} \neq 0$，我们发现条件 (6.102) 成立的前提是 $t \equiv s/c$ 满足一个二次方程：

$$t^2 + 2\eta t - 1 = 0 \,, \quad \eta = \frac{a_{qq}^{(k-1)} - a_{qq}^{(k-1)}}{2a_{pq}^{(k-1)}} \,. \tag{6.104}$$

这个方程有两个根：$\eta_1 = 1/(\eta + \sqrt{1+\eta^2})$ 和 $\eta_2 = -1/(-\eta + \sqrt{1+\eta^2})$。如果 $\eta \ge 0$，我们就取第一个根 η_1，否则就取第二个根。然后我命令

$$c = \frac{1}{\sqrt{1+\eta^2}} \,, \quad s = ct \,. \tag{6.105}$$

这样就可以使得条件 (6.102) 成立。因此，对于任意一对非对角指标 (p,q) 我们都可以进行上述操作。

为了考察 Jacobi 迭代收敛的速度我们定义矩阵的一个特征函数

$$\Psi(M) = \left(\sum_{i,j=1;i\neq j}^{n} m_{ij}^2 \right)^{1/2} = \left(||M||_F^2 - \sum_{i=1}^{n} m_{ii}^2 \right)^{1/2} \,, \tag{6.106}$$

其中 $||\cdot||_F$ 表示矩阵的 Frobinius 模。前面的 Jacobi 迭代给出：

$$\left(\Psi(A^{(k)})\right)^2 = \left(\Psi(A^{(k-1)})\right)^2 - 2\left(a_{pq}^{(k-1)}\right)^2 \leq \left(\Psi(A^{(k-1)})\right)^2 . \tag{6.107}$$

其中的等号成立的条件是 $a_{pq}^{(k-1)} = 0$。因此，在进行 Jacobi 迭代时，最有效的是从其模最大的非对角元开始。但是实际上也可以就对每一个非对角元都做一遍。

Jacobi 算法仅仅适用于实对称矩阵以及它的复推广厄米矩阵。而且它的计算代价还是比较大的，基本上是 $O(n^3)$。但是它有一个优点就是超级稳定。原因就在于上面讨论的严格成立的估计 (6.107)。因此它特别适合于体量比较小的 (典型的 $n \lesssim O(100)$) 实对称矩阵或者复厄米矩阵。如果我们不仅仅要求本征值还要求本征矢，这往往会增加大约 50% 的计算量。

23.2 Sturm 序列

¶ 所谓的 Sturm 序列适用于计算一个实的三对角对称矩阵的本征值问题。前面我们看到，如果矩阵是实对称矩阵，通过 Householder 变换，我们可以将其化为一个三对角对称矩阵 T。这类矩阵的本征值的计算也可以利用 Givens 变换。

我们假设三对角矩阵 T 的对角元为 d_i，$i = 1, 2, \cdots, n$；两个副对角线上的元素为 b_i，$i = 1, 2, \cdots, n-1$。不失一般性我们假设所有的 $b_i \neq 0$，否则经过重新排列问题可以化为一个更小的三对角矩阵的本征值问题。一个重要的观察是，这样一个三对角矩阵的特征多项式可以通过迭代的方法给出。令 $p_0(x) \equiv 1$ 和 $p_1(x) = d_1 - x$，

$$p_i(x) = (d_i - x)p_{i-1}(x) - b_{i-1}^2 p_{i-2}(x) \ , \quad i = 2, \cdots, n \ . \tag{6.108}$$

容易验证 $p_n(x)$ 恰好就是 T 的特征多项式 $\det(T - x\mathbb{1}_{n\times n})$。不仅如此，事实上 $p_i(x) = \det(T_i - x\mathbb{1}_{i\times i})$ 是 T_i 的特征多项式，其中 T_i 就是由矩阵 T 的前 i 行和 i 列构成的矩阵 (也称为 主子矩阵)。公式 (6.108) 给出的多项式序列称为 Sturm 序列 。事实上，大家应当已经看出来了，这些个多项式就是我们在第 15节中提及的正交多项式。读者通过比较公式 (6.108) 和第 15节中的公式 (4.34) 就可以看出。因此那里讨论的正交多项式的性质同样适用于这里的 Sturm 序列。

Sturm 序列满足一系列重要的性质。例如 $p_i(x)$ 的根 (也就是 T_i 的本征值) 与 $p_{i-1}(x)$ 的根一定会交错地出现等等。另外一个是我们很容易确定小于某个数值的实根的个数，这就是下面的结论：

定理 6.7 对于上述的 Sturm 序列 $\{p_i(x) : i = 0, 1, \cdots, n\}$，我们选取任意的 $\mu \in \mathbb{R}$ 并且构造如下的序列：

$$S_\mu = \{p_0(\mu), p_1(\mu), \cdots, p_n(\mu)\} \ , \tag{6.109}$$

那么上述实数序列中从前往后数的过程中，其数值变号的次数 $s(\mu)$ 就是 $p_n(x) = 0$ 所具有的严格小于 μ 的实根的数目。这里我们约定：如果 $p_i(\mu) = 0$，我们认为它是与前面的邻居 $p_{i-1}(\mu)$ 不同号的 (即算做一次变号)。

作为一个例子，考虑 4×4 的三对角矩阵其中 $d_i=2$，$b_i=-1$。这个矩阵的四个本征值由小到大依次是 $\{0.38,1.38,2.62.3.62\}$。如果我们计算 $\mu=3$ 时各个多项式的值我们有：

$$\{p_0(3),p_1(3),p_3(3),p_3(3),p_4(3)\}=\{1,-1,0,1,-1\}\,, \tag{6.110}$$

从左到右数过去一共变号三次 (按照定理中的约定，其中的 0 也算一次)，因此 $p_4(x)=0$ 的根中有 3 个小于 3。与上面提及的矩阵的本征值比对我们发现的确是如此。

由于所有对称矩阵 (或者厄米矩阵) 的根都是实根，因此如果我们仅仅希望寻找它的本征值的话，我们完全可以利用前一章中所提及的各种求根的方法，其中最为稳定的当属对分法。当然，这需要首先选定一个寻找的区间，这时我们可以利用第 20.3小节中的 Gershgorin 圆盘定理 。例如，对于实对称矩阵来说，我们发现搜寻其实根区间的下限 α 和上限 β 可以分别取为，

$$\alpha=\min_{1\leq i\leq n}\left[d_i-(|b_{i-1}|+|b_i|)\right]\,,\quad \beta=\max_{1\leq i\leq n}\left[d_i+(|b_{i-1}|+|b_i|)\right]\,, \tag{6.111}$$

其中我们约定了 $b_0=b_n=0$。更加详细的讨论大家可以参考 [5] 中的 §5.10.2.

23.3 稀疏矩阵的本征值问题：Lanczos 方法

¶ 如果我们希望计算的矩阵是一个庞大的稀疏矩阵，那么前面所列举的方法并不能有效地进行操作。这时，如果我们仅仅需要求解矩阵的部分而不是全部本征对，那么我们可以利用 Krylov 空间迭代的方法。我们这里仅仅考虑实空间的实矩阵，推广到复空间和复矩阵往往只需要简单的符号改动而已。

考虑任意的一个矢量 $v\in\mathbb{R}^n$ 以及一个实对称矩阵 $A\in\mathbb{R}^{n\times n}$。我们考虑矢量空间：

$$K_m(A;v)=\text{span}\{v,Av,\cdots,A^{m-1}v\}\,, \tag{6.112}$$

我们称之为矢量 v 和矩阵 A 的 m 阶的 Krylov 子空间 。

对于一个 $n\times n$ 阶的稀疏矩阵 A，由于它的非零矩阵元只有 $O(n)$ 个，因此对于给定的矢量 v，计算矢量 Av 仅仅需要 $O(n)$ 的浮点数计算量而不是稠密矩阵的 $O(n^2)$。同时储存 Av 也仅仅需要 $O(n)$ 的内存。这就是为什么我们对于大型稀疏矩阵需要利用迭代的方法，已经为何我们对它的 Krylov 子空间感兴趣。迭代的方法求解 $Ax=b$ 的过程相当于在与 A 相关的 Krylov 子空间中寻找方程的近似解。因此在这类计算中我们一般假设 $m\ll n$。

一个重要的问题是张成 Krylov 子空间的这些矢量是否是线性无关的？这个问题之所以重要是因为我们希望在 Krylov 子空间中寻找线性方程的近似解。所以这个子空间是否完备就很关键了。事实上，著名的 Cayley-Hamilton 定理告诉我们，如果矩阵 A 的特征多项式为 p，那么我们一定有：$p(A)=0$。也就是说，$K_m(A;v)$ 最多可能达到的维数就是 n。[24] 在实际的操作中，随着 m 的增加，我们得到的矢量 A^mv 往往变得线性相关，或者

[24] 事实上根据我们第 20.2小节的讨论，如果矩阵 A 是减次的，我们知道 Krylov 子空间 $K_m(A;v)$ 的维数只能够达到矩阵 A 的最小多项式的幂次。

至少是近似地线性相关。其中的原因在于，如果我们将矩阵 A 的最大模的本征值记为 λ_1，相应的本征矢为 x_1，那么当 m 逐渐增加的时候，$A^m v$ 中原先投影在 x_1 上的分量（相对于其他分量而言）会越来越被放大。事实上，这是一个计算矩阵最大模本征值 (以及相应的本征矢量) 的不错的方法，称为幂次方法 (power method)。也就是说，对于任意的矢量 $v = q^{(0)}$，如果我们定义单位矢量：

$$q^{(k)} = \frac{A^k q^{(0)}}{||A^k q^{(0)}||_2} , \tag{6.113}$$

那么这个 $q^{(k)}$ 将逐渐仅仅包含沿 x_1 方向的分量。[25] 我们还可以构建所谓的 Rayleigh 比 (Rayleigh quotient):

$$\nu^{(k)} = q^{(k)\dagger} A q^{(k)} , \tag{6.114}$$

我们一定有：$\lim_{k\to\infty} \nu^{(k)} = \lambda_1$。

¶ 对于矩阵 A 的 m 阶 Krylov 子空间 $K_m(A; v)$，我们可以在其中构建一组正交完备的基矢。这个做法十分类似于标准的 Gram-Schmidt 正交归一化方法。我们首先假定 m 个矢量是线性无关的。我们可以进行下面的 Lanczos 迭代：

Algorithm 13 Lanczos 迭代算法

Require: $A \in \mathbb{R}^{n\times n}$ 且 $A^T = A$；令 $v_0 = 0$，$v_1 = v$；相应的 m 阶 Krylov 子空间为 $K_m(A; v)$。设 $\beta_1 = 0$.

1: **for** $k = 1, 2, \cdots, m-1$ **do**

2:

$$w_k = A v_k , \tag{6.115}$$

3: 计算内积 $\alpha_k = w_k^T v_k$ 并且更新 w_k:

$$w_k \leftarrow w_k - \alpha_k v_k - \beta_k v_{k-1} . \tag{6.116}$$

4: 归一化 w_k：

$$\beta_{k+1} = ||w_k||_2 , \quad v_{k+1} = \frac{w_k}{||w_k||_2} . \tag{6.117}$$

5: **end for**

6: 令 $w_m = A v_m$，以及 $\alpha_m = w_m^T \cdot v_m$。

经过上述的 Lanczos 迭代，我们获得了一组 $K_m(A; v)$ 中的正交归一的基矢，它们构

[25] 当然，还有一系列额外的要求，比如一般要求矩阵 A 是可对角化的；初始矢量 $q^{(0)}$ 在 x_1 上的投影不为零等等。

成了一个长方正交矩阵 (因为一般来说 $m \ll n$):

$$V_m = (v_1, v_2, \cdots, v_m) \ . \tag{6.118}$$

同时我们还可以得到一个三对角矩阵:

$$T_{mm} = \begin{bmatrix} \alpha_1 & \beta_2 & 0 & & & 0 \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ 0 & \beta_3 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & \beta_{m-1} & \\ & & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ 0 & & & & \beta_m & \alpha_m \end{bmatrix} \ . \tag{6.119}$$

它可以看成是变换矩阵 $V_m \in \mathbb{R}^{n\times m}$ 变换以后的原矩阵 A 在 Krylov 子空间中的投影:

$$T_{mm} = V_m^T A V_m \ . \tag{6.120}$$

由于 T_{mm} 是一个比原矩阵 A 小很多的矩阵，因此处理它的本征值问题要简单的多。而且由于 T_{mm} 的三对角形式，我们可以运用前一小节提及的 Sturm 序列求解。需要注意的是，一般的 Lanczos 算法当 m 变得比较大的时候往往会遇到稳定性的问题。也就是说，原先做好的正交归一化步骤有可能由于舍入误差的影响而不再成立。这时可能在需要重新进行 Gram-Schmidt 的操作。

最后我们提一下，如果矩阵不是对称矩阵而是一般的实矩阵，也存在一个与 Lanczos 迭代类似的算法，称为 Arnoldi 迭代 。其步骤也与 Lanczos 非常类似。只不过由于矩阵不具有对称性，因此最终的矩阵 T_{mm} 也一般不是三对角矩阵，而仅仅是上 Hessenberg 矩阵，其本征对可以利用我们前面讨论过的 QR 算法处理。进一步的讨论参见第 40节中的讨论。

相关的阅读

这一章我们讨论了矩阵的本征值问题。由于涉及的问题实在太多，因此难免有所疏漏。只好请同学们阅读相关的参考文献了。

第七章　随机数、数据的处理与拟合

本章提要

- ☞ 伪随机数的产生与基本应用
- ☞ 数据的统计描述
- ☞ 误差分析与再抽样方法
- ☞ 曲线的拟合

这一章中我们将讨论两个相互关联的课题：伪随机数的产生和科学数据的分析和拟合。伪随机数是计算机诞生之后派生出来的一个新物种。在计算机诞生之前，我们只会产生真正的随机数：它们都是由真实的物理过程产生的 (比如扔骰子)。伪随机数大量地应用于日常生活中。从各种电脑游戏到臭名昭著的帝都机动车摇号系统，[1] 都是伪随机数应用的典型案例。在科学计算中，伪随机数应用最多的方法是所谓的 Monte Carlo 模拟，这个题目本身就可以写一本相当厚的书了，所以这里我们就不过多涉及了。我们将主要介绍如何产生最简单的几种分布的伪随机数。对 Monte Carlo 方法感兴趣的同学可以参考相关的专著。

本章中要讨论的第二类问题涉及科学数据的处理，这又包括数据的分析和拟合两个重要方法。原则上所有实证科学都会面对实验数据（或计算数据）的处理问题。如何“科学地”处理这些数据是一切自然科学必须认真且仔细处理的重要问题。这个问题背后的数学基础是数理统计和概率论。虽然我们这个课程并不会用到过多的数学内容，但是对此有一定的基础认识还是有必要的。因此，我们将首先讨论数据的统计描述问题。随后我们将

[1] 我们实际上应当感谢计算机。如果不是它能够轻松快速地产生伪随机数。我们可以想象一下，如果我们只能用传统的物理过程 –比如扔硬币 –来处理这二百多万申请人的摇号问题，那该如何是好？姑且还不说还要对长期摇不到号人的增加可能性等等这些复杂的设置上。但是对于计算机来说，这些都是非常简单的事情。

进一步讨论如何利用这些数据。[2]

虽然今天误差分析已经成为所有实证科学的基本要求，但是这一点并不是从一开始就是这样的。事实上，即使是在上世纪的二三十年代，误差分析甚至还并不是物理学工作中所必须的。最为典型的例子就是 1925-1929 年间 Dayton C. Miller 所做的对地球相对于以太漂移速度的测定了。

凡是学习过狭义相对论的同学都知道著名的迈克尔逊 -莫雷实验 (Michealson-Morley Experiment)。一般认为，这个实验对狭义相对论的建立起到了重要的作用。[3] 这个实验被又称为史上最著名的“失败的实验”。迈克尔逊 -莫雷实验是在 1887 年。随后，人们不断地试图改进这个实验。其中把这个装置做到极致的就是莫雷的助手 Dayton Clarance Miller 了。他最后制造出的干涉仪的臂长达到了 30 多米。在当时大家都认为最好在山上进行这个实验后，Miller 也是蛮拼的。他亲自雇人将如此笨重的实验器材用马队驮到了加州 Mt. Wilson 的山顶，那里是他认为测定地球相对于以太漂移的理想场所。他的文章 “The Ether-Drift Experiment and the Determination of the Absolute Motion of the Earth”，最终于 1933 年发表在 Review of Modern Physics 上面。[4] 如果大家还不知道这是个什么杂志的话，可以去查查。总之是个很牛的杂志啦。他的结果是地球相对于以太有大约 $10 \sim 11 \pm 0.33 km/s$ 的速度。这在当时还是十分有影响的结果。

当然我们知道，如果狭义相对论是正确的，Miller 的结果一定是错误的。是他什么地方测错了吗？其实不尽然。Miller 是一个非常仔细的人。当时他试图排除了各种可能的系统误差。但就是没有太注意其数据的处理。直到最近 (2006 年)，Thomas J. Roberts 利用 Miller 大约 75 年前采集的大量（大约 5200000 个测量数据！天啊，真是不可思议，他老兄眼睛不累吗？）原始数据，对其进行了“现代化”的误差分析，修正了当年 Miller 对数据处理的一些问题。在修正了这些问题之后，发现 Miller 声称测定的地球相对于以太的漂移速度，在考虑了真实的误差之后，实际上是与零结果兼容的。[5] 事实上，Roberts 在 90% 的置信水平上得到地球相对于以太的漂移速度 $v_e < 6 km/s$。大家也许会问，Miller 难道不知道如何分析误差吗。答案是当时对实验数据进行误差分析并不是完全标准的程序。在 Miller 那个年代，尽管有不少实验也分析误差，事实上 Miller 也给出了数据的误差–只不过这个误差实际上被大大地低估了。无论如何，在那个年代误差分析还远没有成为物理学实验的标准。通过这个例子我们看到，进行系统的、科学地误差分析是一切自然科学的基础。因此，对于一个科学数据，不要仅仅看它的中心值，还应该看它的误差。千万不要以为误差不重要，误差有时候会直接影响一些非常重要的物理结果。这就是为什么我

[2] 大家知道误差可以大致分为统计误差和系统误差。我们这里涉及的基本上都是前者，它也是所有测量中都存在的。统计误差的数学基础是概率论。系统误差往往与具体的实验（或者数值计算）的细节有关，它并不像统计误差那样具有普遍性。

[3] 当然，也有观点认为它其实没有那么决定性。在这个问题上，爱因斯坦本人的观点有些矛盾。

[4] 参见 Dayton C. Miller, “The Ether-Drift Experiment and the Determination of the Absolute Motion of the Earth”, Rev.Mod.Phys, 5, p203-242(1933).

[5] 参见 Thomas J. Roberts, “An Explanation of Dayton Miller’s Anomalous “Ether Drift” Result”, arXiv:physics/0608238.

们有时候会极端地说：一个没有误差的实验数据根本就不配称为科学数据。

其实我们日常生活中的很多例子也能说明正确的误差分析的重要性。例如，最近国家工商局与淘宝闹得不可开交。工商总局在抽查了淘宝上的商品后发现了赝品，而且赝品率还不低。淘宝则质疑工商总局抽样不科学。的确，某些商品的抽样仅仅抽了一个样本，一旦这个产品不幸是赝品，那么它的赝品率就是百分之百。也就是说，原则上除了公布其赝品率之外，还需要公布这个赝品率数据的误差是多少。从这个角度来说，工商总局公布的结果无疑是有瑕疵的。当然，最后他们之间的矛盾随着工商总局领导张茅与马云的一次“亲切友好”的会见而冰释前嫌。[6] 另一个经常见诸媒体的是关于某个公众事件的网络调查。除了公布对某个公众事件的支持率和反对率之外，还应当公布你这个调查取样的大小–这将直接影响结果的统计误差；同时还应公布样本是如何获取 –这将影响结果的系统误差。[7] 所有这一切都将直接影响最终调查结果的可信度。

正如我们前面提到的，本章中我们将侧重于数据处理方面常见的两类应用。第一类最为直接的运用就是数据的误差分析。当我们可以直接测量我们感兴趣的物理量的时候，科学的误差分析最终将给我们一个待测物理量的中心值以及相应的误差。我们同时也会说明如何在数学上理解中心值和误差。另一类十分常用的应用是数据的拟合。这往往出现在某些物理量并不是实验室直接可以测量的情况下。例如，我们感兴趣的物理量 x 可能并不能（或不易）直接测量。我们能够直接测量的是另一些物理量 y。而且我们知道 y 以某种函数形式依赖于 x。这时我们就需要用到所谓的数据拟合。在具体讨论这两类应用之前，我们将首先简单回顾一下随机变量、样本等的基本知识。

24 伪随机数的产生及其应用

¶相信大家都或多或少玩过一些计算机上面的游戏吧。在许多的游戏过程中，游戏的结果是依赖于游戏者的 RP 的。也就是说，并不是每次的结果都是完全相同的，而是有一定的随机性。[8] 其中的原因就在于游戏虽然是从其存档中读取一系列固定的参数，但是游戏的进程中还会与计算机系统的随机数发生相互作用。这就导致了计算机从相同的游戏存档出发，结果并不相同。这在某种程度上增加了游戏的乐趣。当然，游戏者可以利用这一点，通过不停地存档 (Save)/读档 (Load)，尝试通过游戏中某个困难的关卡。这又被游戏者戏称为“S/L 大法”。

计算机上面产生的随机数严格来说称为伪随机数，因为它们并不是真正意义上的随机的，而只是尽可能地随机。要给出伪随机数的一个严格的定义并不是十分容易的。我们将从比较实际的角度出发，来讨论计算机的随机数是如何产生的，以及我们如何探测

[6]参见：http://news.xinhuanet.com/2015-01/31/c_127442652.htm.

[7]例如针对北京实行的机动车限行政策。如果做民意调查时取样的人群是经常开车出行的人和不经常开车出行的人（甚至是没有机动车的人）得到的结果必定是完全不同的。仅仅采信其中任何一部分人的民意其实都是有失偏颇的。这就是我们所说的采样造成的系统误差。

[8]事实上不确定性是游戏性中不可或缺的一个侧面。

它的随机性。计算机上产生伪随机数的程序又被称为伪随机数发生器，有时候又简称为随机数发生器 (Random Number Generator, RNG)。一般来说，影响伪随机数发生器有两个因素：第一是计算速度；第二是所产生的随机数的质量。针对不同的应用，有时候我们必须在上述两个因素中做出取舍和平衡。所谓速度比较好理解，就是平均产生出一个随机数需要的计算数目 –这等效于计算时间；而所谓质量则依赖于我们对伪随机数性能的要求。针对其随机性我们可以采取一系列的测试，如果一个随机数发生器能够通过所有我们能够想象的测试，我们就说它的质量是好的，至少好于那些不能够通过类似测试的随机数发生器。

24.1 线性同余产生器

¶ 前面提到过，计算机产生的实际上不是严格的随机数，而是所谓的伪随机数。它总是通过确定的算法从一个或几个已知的整数 –它们一般被称为种子 –出发，以确定的方式获得另一个或几个貌似随机的整数。由于这个方式是完全确定的（除非将来我们用所谓的量子计算机），因此容易想象我们获得的新的随机整数必定与种子之间存在关联。当然，一个好的随机数发生器应当尽可能降低这种关联。我们后面（参见 § 24.3节）会再次回到这个问题。

¶ 一个最为常用的随机数发生器是利用线性同余产生器（linear congruential generator, LCG）：

$$I_{n+1} = aI_n + c \pmod m , \tag{7.1}$$

其中所有的数都是整数。具体来说，m 称为该算法的模 (modulus)；$a > 0$ 称为乘子 (multiplier)；$c \geq 0$ 则称为增量 (increment)，第一个输入的数，比如 I_0 称为该随机数发生器的种子 (seed)。由于这个算法是通过同余算法生成新的随机整数，因此产生的所有整数一定位于 $[0, m)$ 之间。换句话说，它的周期不可能大于模 m。一旦某个 I_n 取了它的前任曾经取过的值，随后的随机整数一定重复此前的序列。一个随机数发生器发生重复的最小间隔称为该随机数发生器的周期。显然，要使得随机数发生器尽可能的随机，我们需要其周期尽可能地大。显然，上述线性同余发生器的周期一定不大于 m。可以证明，一个线性同余发生器达到最大可能周期的充要条件为：

- c 与 m 互素；
- $a = 1 \pmod p$，其中 p 是 m 的任意一个质因子；
- $a = 1 \pmod 4$，若 m 可被 4 整除。

这些条件又被称为 Hull-Dobell 定理。[9] 由于在计算机中采用二进制，因此比较方便的选

[9] T.E. Hull and A.R. Dobell, "Random Number Generators", SIAM Review, Vol.4, No. 3(1962), 230-254.

择是将 m 选为 2 的幂次，因此我们必须选择 c 为奇数同时 $a = 1 \pmod 4$。[10] 如果我们的选择不满足上述定理的要求，那么随机数的周期会比其模 m 更短。因此，对于严肃的科学计算来说，随机数产生器的参数选择不是一个随意的问题。

24.2 一些常用分布的产生

¶前面我们讨论了（伪）随机整数的产生。但在实际的应用中，我们往往需要按照各种分布的随机实数。本小节中我们就来讨论这个问题。

1. 在 $[0,1)$ 之间均匀分布的随机实数。
   这是后面所有讨论的分布的基础。其实如果我们满足于线性同余发生器，那么最为简单的办法就是选择 m 为机器上最大的可表示的整数。对于用 4 个 Byte 表达的正整数来说，它一共有 $4 \times 8 = 32$ 个 bit，因此可以选 $m = 2^{32}$。于是，只要计算 I_n/m 我们自然就得到

2. 正态分布的实数。
   一个非常经常用到的分布就是正态分布（又称高斯分布），这可以通过下面的方法产生。如果 x_1, x_2 是按照均匀分布的两个随机数，我们令：

$$\begin{cases} y_1 = \sqrt{-2\ln x_1}\cos(2\pi x_2) \\ y_2 = \sqrt{-2\ln x_1}\sin(2\pi x_2) \end{cases}, \tag{7.2}$$

   以及它的逆关系

$$\begin{cases} x_1 = e^{-(1/2)(y_1^2+y_2^2)} \\ x_2 = \dfrac{1}{2\pi}\tan^{-1}\dfrac{y_2}{y_1} \end{cases}, \tag{7.3}$$

   我们可以容易地计算出从变量 (x_1, x_2) 变换到变量 (y_1, y_2) 之间的雅克比行列式：

$$\frac{\partial(x_1,x_2)}{\partial(y_1,y_2)} = \left[\frac{1}{\sqrt{2\pi}}e^{-y_1^2/2}\right]\left[\frac{1}{\sqrt{2\pi}}e^{-y_2^2/2}\right]. \tag{7.4}$$

   即如果 (x_1, x_2) 满足二维单位区间的均匀分布，则按照公式 (7.2) 产生的 (y_1, y_2) 必定遵从高斯分布。如果随机变量 y 的分布函数为：

$$p(y) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-(y-\mu)^2/(2\sigma^2)}, \tag{7.5}$$

   我们将其记为 $y \sim \mathcal{N}(\mu, \sigma^2)$。我们看到，$y_1$ 和 y_2 都服从标准的正态分布，前提是 (x_1, x_2) 的确是服从二维的均匀分布的。这个看似 trivial 的假设其实并不那么 trivial，参见下一小节的讨论。

[10] 这样做的另一个好处是，在二进制中一个 2 的幂次的整数只有一个位为 1，其余位都为零。因此在进行同余运算时，我们可以仅仅考虑一个 bit 上的操作即可。

24.3 线性同余发生器的问题

¶ 前面的讨论假设我们并不需要统计性能十分优异的随机数产生器。但是，在一些大型的 Monte Carlo 计算中，人们往往需要具有特别良好统计行为的随机数产生器。这时前面讨论的线性同余发生器就显得不够了。

¶ 线性同余发生器从一个随机整数出发，完全确定地产生出下一个整数。可以想象，这两个数之间一定存在着关联。换句话说，如果我们真是随机的产生三个整数，I_1，I_2，I_3 的话，它们都在 $[0, m)$ 之间，如果我们画出坐标 $(I_1, I_2, I_3), (I_4, I_5, I_6), \cdots$ 在三维的分布，它们应当随机地、均匀地分布在一个边长为 m，间隔为 1、并且加上了周期边条件的一个三维正方晶格上。但是实际上这些点将分布于三维晶格的某些特定的晶面（也就是比三维要低的一个维度）上。为了符号上的方便，我们可以将其除以 LCG 的模，这样一来点

$$p_1 = (I_1, I_2, I_3)/m \,, \tag{7.6}$$

就位于一个三维的单位立方体之内。这个立方体实际上是一个加了周期边条件的晶格 (也就是说，实际上是一个 torus)，其晶格常数为 $1/m$。我们可以考虑一般的点：$p_i = (I_i, I_{i+1}, I_{i+2})/m$，它也是位于上述单位立方体内。G. Marsaglia 在六十年代末首先意识到，这些点并不会充满三维单位立方体中的所有格点，而是会位于一系列相互平行的二维晶面上。[11] 因此，这个现象有时候又被称为 Marsaglia 效应。事实上，如果我们将 D 个相邻的随机数看成是 D 维空间中的一个点的话，那么 Marsaglia 证明了，这些点并不会随机地充满 D 维空间中的单位立方晶格，而是会落在 $D-1$ 维的一系列平行的超晶面上面。这些晶面的数目最多只可能是 $(D!m)^{1/D}$。由于单位立方晶格在 D 维的每个方向上有 m 个格点，因此如果

$$m \gg (D!m)^{1/D} \,, \tag{7.7}$$

说明我们的随机数发生器具有较强的 Marsaglia 效应，在需要连续多个随机数时可能会有问题；而如果 $m \sim (D!m)^{1/D}$，则说明随机数发生器表现良好，Marsaglia 效应不明显。需要注意的是，Marsaglia 效应依赖于维数 D。

在随机数发生器使用的早期，人们并没有意识到这个问题的严重性。但是当你一次需要多个随机数时，Marsaglia 效应就可能会显现。一个著名的例子是早年 IBM 机器上的臭名昭著的随机数发生器 RANDU。这个随机数发生器是一个典型的、但是糟糕的（按照 Knuth 的说法）随机数发生器。它的参数是：$m = 2^{31}$，$a = 2^{16} + 3 = 65539$，$c = 0$。大家很容易发现，这些参数并不满足我们前面提到的 Hull-Dobell 定理，因此它的周期一定比 m 要短。事实上，它往往只有 $m/4$。但是这并不是 RANDU 的最大问题。它的更为糟糕的问题是在三维以上有严重的 Marsaglia 效应。[12] 例如，如果我们将相邻的三个随机数（每个分量归一到区间 $(0, 1)$ 内）作为坐标画在前面提到的单位立方体内，我们发现它们的确分布在一系列 (15 个) 的晶面上面。这些晶面显示在图 7.1中。显然 $15 \ll m = 2^{31}$，因此 RANDU 在三维显示出严重的 Marsaglia 效应。事实上，RANDU 现在已经不再使用

[11] G. Marsaglia, "Random numbers fall mainly in the planes", Proc. Natl. Acad. Sci. **61(1)**, 25-28 (1968).

[12] 如果我们考察二维的情形，其实 RANDU 还是可以的。

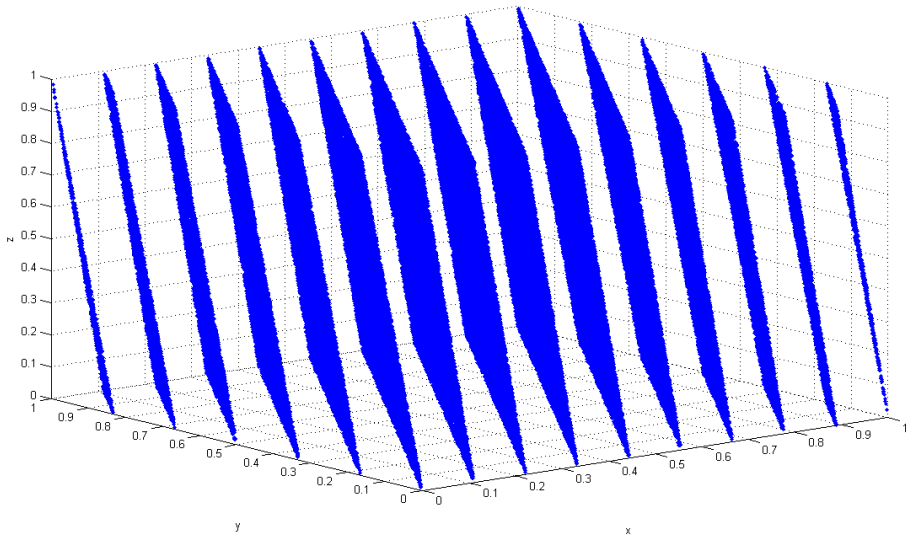


图 7.1: 臭名昭著的随机数发生器 RANDU 的 Marsaglia 效应。图中显示了 100,000 个三维点在单位立方体中的分布。它们集中分布在 15 个平面上。注意，你必须选择合适的角度才能看清这种结构。如果角度不合适（比如正好与这些平面垂直），它们看起来还是很“随机”的。本图取自 Wikipedia 网页:`http://en.wikipedia.org/wiki/Spectral_test`。

了。不过回顾这些历史还是有助于现在的我们理解随机数发生器选择的重要性，特别是当你需要做严肃的、需要大量随机数的科学计算（最典型的就是 Monte Carlo 模拟）的时候。

为了衡量一个随机数发生器的优劣，人们设计了随机数发生器的谱测试 (spectral test)。前面提到，尽管随机数发生器可以通过多个测试，但是谱测试是比较严格的一个。它可以比较有效地衡量一个随机数发生器的有效性。有关这点，请有兴趣的同学参考 D. Knuth 的大作，我们这里就不再深入了。[13]

¶同学们也许觉得像 RANDU 这样的梦魇应当只是出现在计算机科学发展的初期，在比较现代的时候是不会重演的。其实未必。有兴趣的同学可以参考下面两篇文章。[14] 在这两篇文章中，作者们利用一系列在九十年代时还在运用的随机数发生器结合 Monte Carlo 算法，计算了二维正方晶格上 Ising 模型在临界点附近的能量密度和比热容。大家都知道，二维正方晶格上的 Ising 模型是一个严格可解的统计模型，即它的配分函数是可以解析地计算出来的（所谓的 Onsager 解）。于是，我们可以计算在临界点附近的能量密度和热容。

[13] D.E. Knuth, “The Art of Computer Programming volume 2: Semi-numerical algorithms”, 2nd Ed. Addison-Wesley, p.89

[14] A.M. Ferrenberg, D.P. Landau and Y.J. Wong, “Monte Carlo Simulations: Hidden Error from “Good” Random Number Generators”, Phys.Rev.Lett. 69, 3382 (1992); P.D. Coddington, “Analysis of Random Number Generators Using Monte Carlo Simulation”, Int.J.Mod.Phys. C5 547 (1994).

这个解析的结果可以与 Monte Carlo 数值模拟的结果进行比较，从而提供了一个对 Monte Carlo 计算相当严苛的检验。作者们发现，一些被认为很不错的随机数发生器其实都存在潜在的风险。也就是说，Monte Carlo 计算出来的结果与应当呈现的解析结果之间呈现出显著的偏离 (当然是在统计意义上)。请注意，二维 Ising 模型是一个十分特殊的模型，这里我们知道其解析结果从而可以进行比较。但是对绝大多数 Monte Carlo 计算处理的问题来说，我们并不知道严格的结果是什么。甚至在很多情况下，Monte Carlo 即将“定义”所谓的严格结果。[15] 考虑到这一点，这种偏离还是十分“惊悚”的。这就是为什么 M. Lüscher 专门为大型的 Monte Carlo 计算提出了基于 Marsaglie-Zaman 随机数发生器之上的一种新的随机数发生器，称为 RANLUX(意为 Luxury Random Number Generator)。[16] 更多的信息我们这里就不再深入了。

¶最后让我们总结以下关于随机数发生器的选择问题。这严重地依赖于你的具体应用。你需要大量的随机数序列吗？如果答案是否定的，那么基本上你可以用任何的 RNG，甚至是我们前面提到的臭名昭著的 RANDU；如果你的应用的确需要大量的随机数，但是对其质量并不特别苛求（例如，你正在设计的某款电脑游戏中要用到的随机数），那么其实你应当尽可能用运算快捷的 RNG，虽然这类 RNG 的质量往往好不到哪里去，但是也许你的游戏玩家也许恰恰会感谢你呢；如果你不仅仅需要大量的随机数，而且还需要高质量，那么你的困难来了：你必须在这两者之间寻找一个平衡点，因为一般来说高质量的 RNG 总会更慢一些。目前笔者所能想到的需要大量的、高质量的随机数的情况仅仅限于大型的、科学方面的 Monte Carlo 计算。因此从这个角度来说，如果你的应用不是这一类，你基本上不必太介意本节所讨论的这些问题。

25 数据的统计描述

25.1 随机变量的一些基本知识

¶从数学上讲，我们物理上所称的数据可以看成是某个随机变量所取的值。一个随机变量（random variable, or stochastic variable）X 可以看成是从某个概率空间 $(\Omega, \mathcal{F}, P)$ 到某个值域 (一般是实数域 $\mathbb{R}$，但也可以是更为复杂的) 的一个可测函数。[17] 说的通俗一

[15] 这一点绝不是耸人听闻。典型的例子是统计物理中研究了多年的所谓“钢球气体模型”。自然界中是没有严格的钢球气体的。这个模型仅仅存在于计算机中。因此，对于钢球气体的数值模拟实际上就定义了它的统计性质。

[16] 参考网页 http://luscher.web.cern.ch/luscher/ranlux/，以及文章：M. Lüscher, “A portable high-quality random number generator for lattice field theory simulations”, Comp.Phys.Comm. 79 100 (1994).

[17] 可测函数的数学定义如下：它是两个可测空间之间的一个映射：$f : (\Omega_1, \Sigma_1) \to (\Omega_2, \Sigma_2)$，使得对每个 $E \in \Sigma_2$，它的原像一定也属于 Σ_1。即：$f^{-1}(E) := \{x \in \Omega_1 | f(x) \in E\} \in \Sigma_1 \ , \forall E \in \Sigma_2$。注意，函数的可测性不仅仅依赖于其定义域和值域，还依赖于其上定义的 σ-代数。如果我们不强调这点的时候，也可以直接写为：$f : \Omega_1 \to \Omega_2$。特别是当值域是实数域的时候，我们默认 $(\Omega_2, \Sigma_2) = (\mathbb{R}, \mathcal{B}) := \mathbb{R}$，其中 $\mathcal{B}$ 是实数域上的 Borel 代数。

点儿，就是一个以一定的概率取各个可能值的函数。

之所以这样来描写我们获得的实验数据是因为我们知道实验数据并不总是一样的。即使是测量同一个经典的物理量，我们都会得到不太一样的结果。这时我们大致可以区分两类物理量，一类称为经典的物理量，另一类就是量子的物理量。对经典的物理量，我们相信它具有完全确定的数值，至少在我们测量它的那个时间附近，它的“固有值”是完全确定的。但是由于我们的测量总是具有各种不可控的因素，这就造成了我们每次测量出来的结果稍有不同。但是，经验告诉我们，所有的这些结果都会集中分布在其固有值附近。换句话说，这时的随机性并不是待测客体所天生具有的，而是由于我们主观的测量所引入的。但是如果我们测量的是一个量子的客体，那么它本质上就是随机的。我们的测量只是按照其量子的概率分布，使得待测的客体按照量子力学所预言的概率塌缩到一系列固定的本征态而已。

一个取值在实数域的随机变量 X 可以由其 (非负的) 概率分布函数（probability distribution function, PDF）$f(x)$ 来描写。该随机变量处于区间 $[a,b]$ 的概率由：

$$\text{Prob}[a \le X \le b] = \int_a^b f(x)dx \,, \tag{7.8}$$

其中的积分应当在 Lebesgue 意义下来理解。分立的情形也可以用这种语言来描写只要我们允许其中的概率分布函数可以取象 δ-函数这样的广义函数即可。累计概率分布函数（cumulative distribution function）实际上是 pdf 的积分：

$$F(x) = \text{Prob}[-\infty < X \le x] = \int_{-\infty}^{x} f(t)dt \,. \tag{7.9}$$

由于 pdf 是非负的，因此 $F(x)$ 必定是单调不减的。并且它自然地满足归一化条件：$F(+\infty) = 1$。

一个随机变量 X 的期望值 (expectation value) 的定义是：

$$\langle X\rangle := E[X] = \int_{-\infty}^{\infty} xf(x)dx \,, \tag{7.10}$$

其中 $E[\cdot]$ 表示取期望值的运算。期望值的另一个名称是平均值 (mean)，它实际上是 X 的一阶矩。与期望值相对应的还有一个称为随机变量的中位数 (median) 的概念。中位数 m 的定义是：

$$F(m) = 1/2 \,, \tag{7.11}$$

其中 $F(x)$ 是随机变量 X 的累计概率分布函数。也就是说，小于 m 和大于 m 的概率各占 $1/2$。一个随机变量的标准偏差 (standard deviation) 的平方被称为方差 (variance)，记为 σ^2，它被定义为

$$\sigma^2[X] = Var(X) = E\left[(X - \langle X\rangle)^2\right] = E[X^2] - (E[X])^2 \,. \tag{7.12}$$

也就是说，是该随机变量的二极矩再减去其一极矩的平方。类似地，我们还可以定义更高的极矩。例如，对于三极矩，我们一般称之为偏斜度(skewness)，其定义为

$$\gamma_1 = E\left[\left(\frac{X-\langle X\rangle}{\sigma}\right)^3\right], \tag{7.13}$$

其中 σ 是 X 的标准偏差。类似地四极矩被称为峰起度(kurtosis):

$$\gamma_2 = \frac{1}{\sigma^4}E[(X-\langle X\rangle)^4] - 3\,. \tag{7.14}$$

显然，当一个分布函数给定后，它的任意极矩也就给定了；但是反过来不一定。要确定一个分布函数原则上需要所有极矩的数值，然后可以利用所谓的 Mellin 变换获得分布函数。

前面的讨论仅仅涉及到一个随机变量。事实上我们很多物理问题中会涉及到多个随机变量。不同的随机变量之间有可能并不是完全不相关的。描写两个随机变量 X，Y 之间统计相关性的特征函数是两者的协方差(covariance):

$$Cov(X,Y) := E\left[(X-\langle X\rangle)(Y-\langle Y\rangle)\right]\,. \tag{7.15}$$

如果 $Cov(X,Y)=0$，我们就称随机变量 X，Y 统计非相关(uncorrelated)；反之则称两者统计相关。这里需要澄清的一点是，统计独立性与统计相关性是两个不完全相同的概念。两个随机变量的相关性依赖于它们的协方差。而统计独立性，如果用分布函数来描写，意味着总的分布函数是两个随机变量各自分布函数的乘积。事实上，统计独立性一定导致统计非相关性，[18] 但反之未必成立。

25.2 样本性质的描述

¶ 我们物理学中测量的那些数据并不是随机变量本身，而是随机变量的一个“实现”。测量出来的这些数据一般被称为一个样本(sample)，而测量可以看成是我们从随机变量所对应的概率分布中取样(sample)的过程。我们的目的就是希望通过这些测量（或者等价地说，取样）来了解我们希望了解的随机变量 X 的概率分布。

物理测量的结果其实在多数情况下并不是那么“随机”的，至少在经典物理量的测量是如此。例如，考虑我们用经典的天平来测量某个经典的宏观物体（比如一个苹果）的质量。那么尽管测量的数据有可能有些变化，但大体上总是在某个数值附近，例如 0.288kg。如果我们立刻再测一次，它可能会变为 0.282kg，但是不太会变成 0.145kg，除非某人在测量之前咬了一口。因此，如果我们将苹果的质量 X 考虑为一个随机变量的话，它的概率分布函数一定是在 0.288kg 附近的概率比较大。离开这个数值越远，其概率分布函数就越

[18] 因为若 X,Y 统计独立，则：$Cov(X,Y) = E\left[(X-\langle X\rangle)\right]E[(Y-\langle Y\rangle)] = 0$，因此一定非相关。

小（尽管可能并不严格为零）。对于这种情形，经验告诉我们，要获得关于这个苹果的质量更加准确的信息，我们需要的是多次的进行测量（或者说）并平均。

假定我们进行了 n 次独立的测量，分别获得了数据为 $\{x_1, x_2, \cdots, x_N\}$，这就是一组样本（sample），它可以看成是每次测量的随机变量 X_i 的结果。

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{n} x_i \ , \tag{7.16}$$

被称为这一组数据的样本平均值。类似地，样本方差 (sample variance) 的定义为：

$$s^2 := \frac{1}{N-1}\sum_{i}^{N}(x_i - \bar{x})^2 \ . \tag{7.17}$$

请特别注意其中的 $1/(N-1)$ 的因子，我们下面会解释这是为什么。

样本平均有什么用？答案是，它提供了原先随机变量期望值的一个无偏的估计。要看清这点并不复杂。我们定义如下的随机变量：

$$\bar{X} := \frac{1}{N}\sum_{i=1}^{N} X_i \ . \tag{7.18}$$

于是，样本平均值 $\bar{x}$ 可以看成是随机变量 $\bar{X}$ 的一个“实现”。假定我们每次测量时的随机变量 X_i 的期望值是相同的，即 $E[X_i] = \mu$, $\forall i$，那么我们立刻得知 $E[\bar{X}] = \mu$。读者也许要问，既然每个 i 我们都有 $E[X_i] = \mu$，那么我们测量一次不就行了吗？经验告诉你，多测量可以减小你的测量误差。我们来看看是否如此。为此我们假定在测量的过程中每个随机变量 X_i 的期望值和方差都是不变的：

$$E[X_i] = \mu \ , E[(X_i - \mu)^2] = \sigma^2 \ , \ \ \forall i \ . \tag{7.19}$$

那么对于公式 (7.18) 中定义的随机变量 $\bar{X}$ 来说，我们可以计算它的方差：

$$Var(\bar{X}) = \frac{1}{N^2}\left(\sum_{i=1}^{N} Var(X_i) + \sum_{i\neq j} Cov(X_i, X_j)\right) = \frac{\sigma^2}{N} \ . \tag{7.20}$$

其中我们利用了各个 X_i 不相关的事实。我们看到，虽然 $\bar{X}$ 的实现 -也就是 $\bar{x}$-与任何一次测量的结果给出同样的期望值，但是它对应的方差却比原先一次测量的小了 N 倍。所以，经过多次测量我们就可以获得待测物理量更加精确的信息。

那么我们可以构建如下的随机变量：

$$S^2 = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - \bar{X})^2 \ , \tag{7.21}$$

其中 $\bar{X} = (1/N)\sum_{i=1}^{N} X_i$ 是我们公式 (7.18) 中定义的随机变量。那么我们前面定义的样本方差 (7.17) 可以看成是这个随机变量 S^2 的一个实现。我们现在来计算 $E[S^2]$。

$$\begin{aligned} E[S^2] &= E\left[\frac{1}{N-1}\sum_{i=1}^{N} X_i^2 - \frac{N}{N-1}\bar{X}^2\right] , \\ &= \frac{1}{N-1}\left(NE[X_i^2] - NE[\bar{X}^2]\right) , \\ &= \frac{N}{N-1}\left(Var(X_i) + E[X_i]^2 - Var(\bar{X}) - E[\bar{X}]^2\right) \\ &= \frac{N}{N-1}\left(\frac{N-1}{N}Var(X_i)\right) = Var(X_i) . \end{aligned} \tag{7.22}$$

因此 S^2 的方差与原先待测物理量的方差相同从而 s^2 给出了待测物理量方差的一个无偏估计。如果我们的定义中不是用 $1/(N-1)$ 而是简单地用 $1/N$，那么相应的估计是有偏的。这一点在历史上是 F.W. Bessel 首先意识到的。[19] 这就揭示了为什么我们总是在实验中利用

$$\Delta x = \sqrt{\frac{1}{N(N-1)}\sum_{i=1}^{N}(x_i - \bar{x})^2} , \tag{7.23}$$

来标记待测物理量的误差。

25.3 样本统计参数的数值计算问题

¶ 一般来说样本平均值的计算没有什么特别需要注意的。对于样本方差的计算，则可能需要注意舍入误差带来的影响。按照样本方差的表达式：

$$\begin{aligned} s^2 &= \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2 \\ &= \frac{1}{N-1}\sum_{i=1}^{N} x_i^2 - \frac{N}{N-1}\bar{x}^2 . \end{aligned} \tag{7.24}$$

虽然上式中的两行在代数上是完全相同的，但是一般来说第二行更容易受到舍入误差的影响。因此更加安全的计算方法是按照第一行来进行数值计算。当然，所有这些一般都仅仅会在样本数目很大的情况下出现。如果样本的数目 N 仅仅是几十或者几百，怎么做都不会有太大影响。但是如果 $N \sim O(10^7)$ 或更多，那么求和就需要有些额外的考虑了。

在一个求和中如何控制舍入误差的影响？一般来说有以下几种方法：第一个最直接的方法就是提高浮点数的精度；一般的求和是利用单精度进行的，如果觉得可能有问题，可以将其换成双精度进行。第二种方法就是利用所谓的 补偿求和 的方法。

• • • •

[19] 这里假定随机变量 X 的严格的期望值 $\mu = \langle X \rangle$ 并不已知。在现实应用中一般总是如此。

例如我们需要计算求和：

$$S_n = \sum_{i=1}^{n} x_i \ , \tag{7.25}$$

其中 $n \gg 1$，例如 $n \sim O(10^7)$ 或更大。一般来说，每一次加法计算机都会进行舍入。舍入过程一定造成误差 ε。舍入误差对于单精度来说大概是 10^{-7} 而对双精度来说则大约为 10^{-16}。对于一个一般的 n 项求和，如果每次加法造成的舍入误差为 ε，那么通常 S_n 的舍入误差最大可能大约是 $O(n\varepsilon)$。也就是说，对于单精度数的求和，当 $n \sim 10^6 - 10^7$，通常的求和造成的舍入误差已经相当可观。当然，如果我们运用双精度求和，那么可以允许的项数大约为 $n \sim 10^{16}$。[20] 一个通常的加法的累加程序大体上如下：

1. 初始设置，$s = 0$;
2. 从 $i = 1$ 开始累加：$s = s + x_i$;
3. 回到上一步，直到所有数据累加完毕，输出 s。

当 n 很大时，这种通常的加法往往会造成明显的误差。其主要原因是，通常求和的中间项 s 会随着求和项数的增加而慢慢变大，而需要加上的 x_i 的低位信息很有可能会被舍入 (roundoff)。这就是典型的一个大数加一个小数时造成的对小数的误差。也就是说，如果在求和的中间过程中承载数据的变量 s 没有足够多的位数，那么机器自然就会舍弃那些它认为“不重要”的位数。因此，一个最为直接的解决方法就是让变量 s 的位数足够地多。比如原先是单精度的，现在用双精度；如果原先已经是双精度，现在可以改用四精度（当然这时可能需要重新定义加法），等等。当然，有的时候我们必须采用某种固定的位数。例如，如果我们由于某种原因必须使得 s 是单精度的。可是我们又需要进行大量的数的求和，怎么办呢？这时可以利用所谓 补偿求和 (又称为 Kahan 求和) 的方法。仅仅是稍微修改了一下上述程序，试图在求和中间的每一步都保留待加入的数 x_i 的低位的信息。因此，一个版本的算法如下：

1. 初始设置，$s = 0$，$c = 0$;
2. 从 $i = 1$ 开始累加：
   (a) $ss = x_i - c$；初始时 $c = 0$，因此最初 $ss = x_1$。在下一轮求和中，$(-c)$ 实际上包含了前一轮待求和的数据的低位信息。
   (b) $t = s + ss$；在一个漫长的求和过程中，s 会较大而 ss 较小。因此，直接的相加会造成 ss 的低位信息被 roundoff 而丢失。

[20] 需要注意的是，由于通常的计算机都会采用类型的递归，实际上仅仅需要被累加的数采用双精度即可。也就是说，下面通常的求和程序中，仅仅需要让 s 取为双精度即可，x_i 仍然可以是单精度。当机器执行：$s = s + x_i$ 的语句时，x_i 的低位信息自动会按照双精度的标准，而不是单精度的标准加到 s 中去。

(c) $c=(t-s)-ss$；在求和过程中，$(t-s)$ 包含了 s 的高位；$(-c)$ 则保留了 ss 的低位信息；

(d) $s=t$；回到步骤 (a)。这时补偿 $(-c)$ 会被加入到求和之中。直到所有数据累加完毕，输出 s。

可以证明的是，这种补偿求和的方法基本上可以使得 S_n 的误差达到 ε 的量级，也就是说，它完全不依赖于求和的项数，舍入误差基本上就是每个待加达到数的原始精度。这当然是非常理想的。如果还希望进一步提高精度，则只能够通过提高每个数据 x_i 的精度来实现了。

26 统计相关数据的误差分析

¶ 前面的讨论中各个测量都被假定是互不相关的。事实上，不同的测量之间往往有一些相关性。这时就需要考虑统计相关数据的分析问题了。样本中不同次的测量可以用不同的“时间”t 来标记，也就是说，我们前一节中的随机变量现在我们将它记为 X_t，其中 t 可以取一系列等间隔的分立值。为了简单起见，我们取 $t=0,1,2,\cdots$。当然，t 也可以取连续的实数值，这时它真的可以代表时间。一个依赖于时间 t 的随机变量称为一个**随机过程** X_t。我们将假定随机过程是所谓二阶平稳的，即 $\mu=E[X_t]$ 和 $\sigma^2=Var(X_t)$ 都不依赖于时间。对二阶平稳的过程，我们定义它的**自关联函数** (autocorrelation function)：

$$A(\tau)=\frac{1}{\sigma^2}E[(X_t-\mu)(X_{t+\tau}-\mu)]\,. \tag{7.26}$$

如果 $A(\tau)=\delta_{\tau,0}$，我们就说 X_t 不存在自相关，否则就称其存在自相关。自相关又称为**序列相关** (serial correlation)。按照定义，自关联函数满足：$A(0)=1$，$A(\tau)=A(-\tau)$。

自相关实际上反映了不同时间的测量之间的某种“记忆”。一般来说，自关联函数是随着时间间隔指数衰减的：

$$A(\tau)\sim e^{-|\tau|/\tau_a}\,, \tag{7.27}$$

其中参数 τ_a 称为该随机过程的**自关联时间**。另外一种标度自相关的量是所谓的**积分自关联时间**，它的定义是：

$$\tau_{int}=1+2\sum_{t=1}^{\infty}A(\tau)\,. \tag{7.28}$$

我们下面会看到，这个数会影响我们的误差分析。

自关联在实验中实际上是普遍存在的。尽管我们认为我们尽可能地做到每次测量都是“独立的”，没有受到前面测量的影响，但是那只是理想的情况。在另一种情形下这种

自关联变得更为普遍，那就是利用 Markov 过程产生的数据，这广泛地出现在格点场论以及统计物理的 Monte Carlo 计算中。

¶ 如果我们的测量具有统计的相关性，那么样本的平均值仍然是该物理量期望值的一个无偏估计：

$$E[\bar{X}] := E\left[\frac{1}{N}\sum_{t=0}^{N-1} X_t\right] = \mu \,, \tag{7.29}$$

但是 $\bar{X}$ 的方差的计算则因为关联而有所不同。按照公式 (7.20)，我们有：

$$Var(\bar{X}) = \frac{1}{N^2}\left(\sum_{t=0}^{N-1} Var(X_i) + \sum_{t\neq t'} Cov(X_t, X_{t'})\right)$$

但是 $Cov(X_t, X_{t'}) = \sigma^2 A(|t-t'|) \neq 0$，因此我们得到：

$$Var(\bar{X}) = \frac{\sigma^2}{N}\left[1 + 2\sum_{t=1}^{N-1}\left(1 - \frac{t}{N}\right)A(t)\right] \,. \tag{7.30}$$

注意到在 $N \to \infty$ 的情形下，上式中方括号内的量正是前面定义的积分自关联时间 τ_{int}。因此我们得到：

$$Var(\bar{X}) = \frac{\sigma^2}{N} \cdot \tau_{int} \,. \tag{7.31}$$

也就是说，实际的等效的独立测量数目并不是 N，而是 N/τ_{int}。粗略地说，在有相关的测量过程中，大约每 τ_{int} 次测量才是一次统计独立的测量。

注意，一般来说积分自关联函数的计算依赖于所有的测量结果。因此要估计出它的方差是相当困难的。事实上按照的定义 (7.28)，如果我们将它作为 τ_{int} 的一个估计，可以证明它的方差趋于无穷大：$Var(\tau_{int}) \to \infty$。因此，真正计算 τ_{int} 一般采用

$$\bar{\tau}_{int}(t) = 1 + 2\sum_{t'=1}^{t} A(t') \,, \tag{7.32}$$

然后在 $\bar{\tau}_{int}(t) - t$ 图中寻找一个平台区间。

¶ 对于已经确定具有自相关的测量结果，我们一般需要进行 Blocking 的处理。这样才能够更加准确地估计其误差。为此，我们将原先的随机变量 X_t 记为 $X_t^{(0)}$。我们随后构建如下的一系列随机变量：

$$X_t^{(n+1)} = \frac{1}{B}\sum_{j=1}^{B} X^n_{B(t-1)+j} \,. \tag{7.33}$$

说的通俗一点，就是将原先的测量变量相邻的 B 个进行一个代数平均，称为第 $n+1$ 次 blocking 的变量。对应于原先的 N 次测量我们有 N 个随机变量，经过一次 Blocking 之后，我们就只有 N/B 个随机变量了。如果再 Block 一次，就只有 N/B^2 个，等等。这个过程可以迭代地进行直到最后只有一个变量（不失一般性，我们假定 N 恰好是 B 的幂次）。

容易证明，对一个二阶平稳的随机过程而言，Blocking 过程并不改变期望值。因为我们总是有：

$$\bar{X^{(1)}} \equiv \frac{1}{N/B}\sum_{t=1}^{N/B} X_t^{(1)} = \frac{1}{N}\sum_{t'=1}^{N} X_{t'}^{(0)} \ , \tag{7.34}$$

而每一个 $E[X_t^{(0)}] = \mu$，因此我们必定有：$E[\bar{X}^{(0)}] = E[\bar{X}^{(0)}] = \mu$。

虽然任何的 Blocking 的大小 B 都是可以接受的，但是最为常用的是取 $B = 2$。我们下面来考察对于一个有自关联的随机过程，它的各次测量中的方差如何变化。为此我们必须对过程的自关联函数做一定的假设。为了简单起见，我们假定相邻的两次测量之间的关联满足：

$$Cov\left(X_t^{(0)}, X_{t+1}^{(0)}\right) = \sigma^2 e^{-1/\tau_a} \ , \tag{7.35}$$

也就是说，由指数自关联时间 τ_a（一个常数）控制。于是，在一次 Blocking 之后，$X_t^{(1)}$ 的方差为：

$$Var(X_1^{(1)}) = Var\left(\frac{1}{2}(X_1^{(0)} + X_2^{(0)})\right) = \sigma^2\left(\frac{1+e^{-1/\tau_a}}{2}\right) \ . \tag{7.36}$$

由于 $0 < e^{-1/\tau_a} < 1$，因此 Blocking 之后的变量的方差比原先没有 Blocking 的要减小了一个因子。类似地，我们可以考虑在一次 Blocking 之后，相邻的两次测量之间的协方差：

$$Cov\left(X_1^{(1)}, X_2^{(1)}\right) = \sigma^2 e^{-1/\tau_a}\left(\frac{1+e^{-1/\tau_a}}{2}\right)^2 \ . \tag{7.37}$$

因此，无论是协方差还是每一个的方差，每次 Blocking 之后都会下降一个因子。但是这个因子并不显著，特别是在 $\tau_a \gg 1$ 的极限下。例如，一次 Blocking 仅仅是使得 X_t 的方差稍微下降了一点点：

$$Var(X_1^{(1)}) \simeq \sigma^2\left(1 - \frac{1}{2\tau_a}\right) \ . \tag{7.38}$$

但是要记住，经过一次 Blocking，总的测量次数减少了一半：$N \Rightarrow N/2$。因此，在在 $N \gg \tau_a \gg 1$ 的极限下，每次 Blocking 大致会使得误差上升一个 $\sqrt{2}$ 的因子。当然这种上升不会永远持续下去。一般来说，经过 n 次 Blocking 之后，误差的上升会大致饱和，这个 n 大致满足：$2^n \sim \tau_a$。这时再进一步的 Blocking 不会对最终的误差产生任何影响，因为这个时候（以该时刻的间隔而不是最初的间隔为单位！）有 $\tau_a \sim 0$，因此每次 Blocking 使得新的变量的方差正好下降一个因子 2，与测量次数的减少的因子 2 正好相消。这个时候给出的误差大约是：$\sqrt{2\tau_a\sigma^2/N}$，比所谓的天真误差 $\sqrt{\sigma^2/N}$ 正好大一个 $\sqrt{2\tau_a}$ 的因子。利用这个结果，可以给出 τ_a 的一个近似的估计。

27 重抽样方法：Jackknife 与 Bootstrap

¶ 前面我们讨论了在具体测量一个感兴趣的物理量时的一系列问题：如何估计其平均值，如果估计其误差；包括在测量中没有自相关时和有自相关时分别应当如何处理。但是

我们还没有考虑过同时测量两个（或者多个）物理量时的情形。比如我们同时感兴趣 X 和 Y 这两个物理量，情况又会如何呢？这里面一个重要的关系是这两个随机变量是否统计相关，也就是说两者的协方差，

$$Cov(X,Y) = E[(X - \langle X\rangle)(Y - \langle Y\rangle)] \,, \tag{7.39}$$

是否为零。如何利用我们的测量数据来估计两者之间的统计相关性呢？我们这节就来探讨这个问题。显然，我们需要同时测量这两个物理量，X 和 Y，一共测量了 N 次，分别得到数据 $(x_1,y_1),\cdots,(x_N,y_N)$。那么前面的讨论告诉我们，它们各自的样本平均值：

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \,, \quad \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i \,, \tag{7.40}$$

分别构成了 X 和 Y 的期望值的无偏估计。我们如何来估计两者之间的协方差呢？我们尝试如下的定义：

$$\rho(X,Y) = corr(X,Y) = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} \,, \tag{7.41}$$

它称为两个随机变量的关联系数 (correlation coefficient)。可以证明它绝对值一定不大于 1。对于多次的测量，我们可以用

$$\rho(X,Y) = \frac{\sum_{i=1}^{N}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i-\bar{x})^2 \sum_{j=1}^{N}(y_j-\bar{y})^2}} \,. \tag{7.42}$$

来对 $\rho(X,Y)$ 进行估计，这当然没有问题。我们的问题是，$\rho(X,Y)$ 的误差又该如何估计呢？显然这不是一个简单的问题，因为 $\rho(X,Y)$ 的计算依赖于所有的测量。我们虽然可以利用所有的测量值来给出它的估计值，但是并不能直接给出这个估计值的误差。这时我们就需要所谓的重抽样方法了 (resampling methods)。

上面提到的仅仅是重抽样方法的一方面的应用。另外一类常见的应用是我们虽然可以直接测量某个物理量 X，但是我们真正感兴趣的是它的期望值 $\langle X\rangle$ 一个（一般是非线性的）函数 $f(\langle X\rangle)$ 的情况。当然，对于 X 的每次测量的值 x_i，我们总是可以计算 $f(x_i) \equiv f_i$。但是我们知道：

$$f(\langle X\rangle) \neq \langle f(X)\rangle \simeq \frac{1}{N}\sum_{i=1}^{N} f_i \,. \tag{7.43}$$

最关键的是如果我们还希望估计 f 的误差，那么问题就比较复杂了。特别是如果函数 f 在所考虑的区间内正好剧烈变化的时候，通常的误差传递公式 $\delta f \simeq f'(\langle X\rangle)\delta X$ 是无法直接运用的。事实上，当你正确地估计了 X 的误差 δX 之后，你往往会发现 $f(\langle X\rangle \pm \delta X)$ 的数值关于其中心值 $f(\langle X\rangle)$ 并不对称。

下面要讨论的两种重抽样方法可以帮助我们处理上面提到的困难。我们下面将假定，所有的测量都是统计无关的。这纯粹是为了讨论方便而已。如果数据本身并非如此，我们总可以在估计了其自关联时间之后，对初级的数据进行适当的 Blocking 的操作，使得新的一组数据等价于统计无关的测量。

27.1 Jackknife 方法

¶ 所谓的 Jackknife 方法就是将 N 次测量的值中的某一部分 –例如相邻的 m 个 –剔除，其余的 $(N-m)$ 个测量进行平均来获得“新的测量”。剔除的测量的个数依赖于原先测量的统计相关度，或者说它的自关联时间。如果我们近似认为所有的测量是统计无关的，那么原则上可以仅仅剔除一个，即取 $m=1$。如果不是，那么一般我们应当首先对所有的原始测量进行 Blocking，粗略来说可以取 $m \simeq \tau_a$。

下面就是一般 Jackknife 方法的步骤：

1. 利用所有的 N 个数据计算感兴趣的物理量的平均值 $\bar{\rho}$；
2. 将所有数据 (x_i, y_i) 按照时间的顺序，等分为大小为 m 的块 (blocks)，因此总的块数为 $M = N/m$。我们必须使得 $m \gg \tau_a$，其中 τ_a 是该测量序列的自相关时间；如果我们确认数据几乎没有自关联，我们可以取 $m=1$；
3. 我们可以利用分别剔除第 i 块的剩余的 $M-1$ 个块中的数据计算任何我们感兴趣的物理量。这个结果记为 $\bar{\rho}_{(i)}$；
4. 物理量 ρ 的误差可以用下式来估计：

$$\delta\rho = \sqrt{\frac{M-1}{M}\sum_{i}^{M}(\bar{\rho}_{(i)} - \bar{\rho})^2}\,. \tag{7.44}$$

当然，依赖于被剔除的数据的不同，$\rho_{(i)}$ 随着 i 会有所涨落。但是，由于它几乎是利用所有的数据计算出来的结果，经验告诉我们它的涨落一定比原始数据的涨落要小。

类似地，如果我们需要计算某个变量 X 的期望值的函数 $f(\langle X\rangle)$，以及它的误差，我们可以利用下式来计算：

$$\delta f = \sqrt{\frac{M-1}{M}\sum_{i=1}^{M}(f_{(i)} - \langle f\rangle)^2}\,, \tag{7.45}$$

其中 $f_{(i)} = f(x_{(i)})$ 而 $\langle f\rangle = \frac{1}{M}\sum_{i=1}^{M} f_{(i)}$.

27.2 Bootstrap 方法

¶ 另外一种重抽样方法称为 Bootstrap. 它的做法是这样的。

1. 将所有数据 (x_i, y_i) 按照时间的顺序，等分为大小为 m 的块 (blocks)，因此总的块数为 $M = N/m$。我们必须使得 $m \gg \tau_a$，其中 τ_a 是该测量序列的自相关时间；如果我们确认数据几乎没有自关联，我们可以取 $m=1$；
2. 从上述的 M 个块中随机地取出 M 块来，不避讳重复。将这 M 块的数据称为 Bootstrap 数据并储存之。重复这个过程直到我们获得 $N_B >> 1$ 套这样的数据；

3. 对上述获得的 N_B 套的数据（每套数据包含 M 块）中的每一套，计算感兴趣的物理量，比如说 ρ，对第 i 套数据的计算结果记为 ρ_i^*，其中 $i = 1, 2, \cdots, N_B$；

4. 根据所有 ρ_i^* 的分布情况（例如可以画出其直方图）来确定物理量 ρ 的期望值

$$\langle \rho \rangle \simeq \rho_0 = \frac{1}{N_B} \sum_{i=1}^{N_B} \rho_i^* \ , \tag{7.46}$$

以及误差。对误差来说，可以利用

$$\frac{\text{Number of } \rho_i^* < a}{N_B} = 0.16 \ , \frac{\text{Number of } \rho_i^* > b}{N_B} = 0.16 \ , \tag{7.47}$$

来确定两个实数 a 和 b。这时 ρ 位于区间 (a, b) 内的概率为 68%，即所谓的一个 σ 区间。因此我们可以写为

$$\rho = \rho_0 {}^{+(b-\rho_0)}_{-(\rho_0-a)} \ . \tag{7.48}$$

或者当误差的不对称性不明显时，我们也可以用下式估计误差：

$$\delta\rho = \frac{b-a}{2} \ . \tag{7.49}$$

5. 另外一种常见的误差估计为：

$$\delta\rho = \sqrt{\frac{1}{N_B - 1} \sum_{i=1}^{N_B} (\rho_i^* - \rho_0)^2} \ . \tag{7.50}$$

注意，Bootstrop 更加类似于重抽样。我们可以从实际的 M 次独立测量中，随机地、允许重复地重新抽取 M 个样本，构成新的一组样本。我们重新抽样的次数 N_B 越多越好，典型的数值为 $N_B \sim 10^3$。

28 χ^2 拟合

28.1 拟合问题的描述

¶前面讨论的误差处理过程都是针对可以直接测量的物理量。但是在自然科学中有很多情形下我们感兴趣的物理量并不能直接进行测量，或者不易进行直接测量。这个时候我们就需要借助于另一个十分常用的数据处理手段，这就是曲线的拟合。

现代物理学的很多时候都需要进行数据的拟合。最为典型的例子是我们目前关于我们的宇宙基本构成的所谓标准宇宙学模型了。目前最为流行的是所谓的 Λ-CDM 模型（Λ-Cold Dark Matter）模型。[21] 这个模型认为目前我们宇宙中各种物质的比例大约是：暗

[21] 字母 Λ 在宇宙学中一般代表宇宙学常数，暗能量的另外一种称呼。

能量占 73%，冷暗物质占 22%，其他可见物质及中微子占 5%。这些数据显然不是可以直接测量的。它们实际上都来源于对目前多方面宇宙学的观测数据的拟合。

¶ 一般来说，假定我们感兴趣的物理量记为 $x \in \mathbb{R}^n$，它们并不能（或者不易）直接进行测量。我们可以直接测量的是另外一个物理量 $y \in \mathbb{R}$，它以特定的函数形式依赖于 x：

$$y = f(z; x) \,, \tag{7.51}$$

这个函数关系一般称为一个模型。它可能来源于一种猜测，或者来源于某种理论考虑。模型中的参数 z 标记另外一些可以控制实验测量的参数，我们称之为控制参数。这些控制参数可能是一个或几个独立的实数。然后，我们改变这些控制参数并对 y 进行 m 次的测量。即对 $z = z_1, z_2, \cdots, z_m$ 我们最终测到了 y 的数值：$y_1, y_2, \cdots, y_m$。我们假定，通过重复测量或者其他的方法，我们也正确地估计了这些测量值的误差，分别记为：$\Delta y_1, \Delta y_2, \cdots, \Delta y_m$。数据的拟合过程最主要希望回答以下两个方面的问题：

1. 给定上述的测量结果和特定的模型 f，我们能够获取多少关于 x 的信息?
2. 如果我们猜测的函数关系为 f，我们如何能够通过上述测量和拟合过程来确定实验数据是否支持我们的猜测?

其中第一个问题侧重于给定一个模型，如何通过拟合确定该模型中的参数；而第二个问题则侧重于如何判别一个模型是否可以获得实验数据的支持。

如果我们的测量是严格的（没有误差），同时函数关系也是严格成立的，那么我们可以对上述 m 次实验的结果列出下列联立方程：

$$\begin{cases} y_1 = f(z_1; x) \,, \\ y_2 = f(z_2; x) \,, \\ \quad \vdots = \vdots \\ y_m = f(z_m; x) \,. \end{cases} \tag{7.52}$$

一般来说，如果 $m < n$，我们无法完全确定 $x \in \mathbb{R}^n$；如果 $m = n$，我们可以“解出”一个确定的 $x \in \mathbb{R}^n$；如果 $m > n$，除非上述某些方程完全等价，否则该联立方程无解。但是，考虑到所有的测量都存在误差。因此，上述的各个 y_i 实际上并不是确定的数值。它只是某个随机变量 Y_i 的一个取值而已。事实上，我们假定与它们相应的误差为 Δy_i。因此，即使是在 $m > n$ 的前提下，x 也存在概率意义下的解。因此我们的理论模型 (7.51) 也不是关于严格数值的模型，而应当在期望值的意义下理解：$E[Y] = f(z; x)$。总之，拟合问题不同于插值问题。

¶ 我们假定有 m 次独立的测量，测量的值为 y_i，$i = 1, 2, \cdots, m$。我们进一步假定各个 y_i 遵从相同的概率分布：$p(y_i|x)$，其中 $x \in \mathbb{R}^n$ 是待定的 n 个参数。我们可以构建 m

个独立随机变量的联合概率分布：

$$\mathcal{L}(x|y_1,\cdots,y_m)=\prod_{i=1}^{m}p(y_i|x)\ , \tag{7.53}$$

它称为这个参数估计问题的似然函数 (likelihood function)，它的对数则称为对数似然函数 (log-likelihood function)：

$$\hat{l}=\frac{1}{m}\ln\mathcal{L}(x|y_1,\cdots,y_m)\ , \tag{7.54}$$

对于参数 x 的所谓的最大似然估计 (maximum likelihood estimator) 为，[22]

$$\hat{x}=\arg\max_x \hat{l}(x|y_1,\cdots,y_m)\ , \tag{7.55}$$

它给出了最大可能出现的情况下，相应的参数 $x\in\mathbb{R}^n$ 的一个估计值。

一个最为简单而常见的例子是假定各个 y_i 是高斯分布的。这时我们有

$$p(y|\mu,\sigma^2)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(y-\mu)^2}{2\sigma^2}}\ , \tag{7.56}$$

其中 μ 和 σ^2 是高斯分布的期望值和方差：$E[y]=\mu$，$E[(y-\mu)^2]=\sigma^2$，它们是我们希望估计的参数。那么最大似然估计给出，

$$\hat{\mu}=\frac{1}{m}\sum_{i=1}^{m}y_i\ ,\quad \hat{\sigma}^2=\frac{1}{m}\sum_{i=1}^{m}(y_i-\hat{\mu})^2\ . \tag{7.57}$$

这基本上就是我们经常使用的公式，其中第一个给出了期望值的一个估计，它是一个无偏估计，即 $E[\hat{\mu}]=\mu$；但对于第二个实际上是一个有偏估计，因为我们知道 $E[\hat{\sigma}^2]=(1-1/m)\sigma^2$。

28.2 最大似然估计与最小 χ^2 估计

¶我们现在对各次的测量做如下的假定：

1. 各个 y_i 的测量统计独立；
2. 我们的理论模型在参数 x 取其“正确值” x_0 时在期望值意义下给出正确的函数值，即：$E[y_i]=f(z_i;x_0)$，$i=1,2,\cdots,m$;
3. 对于每个 y_i 的误差估计 Δy_i 是正确的

[22]对于似然函数本身取极大值还是它的对数是等价的，因为对数函数是一个单调函数。

这时我们可以认为随机变量

$$\xi_i \equiv \frac{y_i - f(z_i; x_0)}{\Delta y_i} \sim \mathcal{N}(0,1) \,, \tag{7.58}$$

按照期望值为 0、方差为 1 的正态分布 $\mathcal{N}(0,1)$。这个问题的似然函数为 $\mathcal{L} \propto \exp(-\chi^2/2)$，其中

$$\chi^2 = \sum_{i=1}^{m} \left(\frac{y_i - f(z_i; x)}{\Delta y_i} \right)^2 \,. \tag{7.59}$$

于是在这种情形下，最大似然估计等价于**最小 χ^2 估计**。

对于 d 个正态分布的随机变量 ξ_i，$i = 1, 2, \cdots, d$，如果我们它们的平方和：

$$\chi^2 = \sum_{i=1}^{d} \xi_i^2 \,, \tag{7.60}$$

那么变量 χ^2 所遵从的概率分布被称为自由度为 d 的**χ^2 分布**，其概率分布密度为：

$$f(\chi^2|d) = \frac{1}{2^{d/2}\Gamma(d/2)} \left(\chi^2\right)^{d/2-1} e^{-\chi^2/2} \,. \tag{7.61}$$

与这个概率分布相关的是它的累计积分函数。

$$Q(\chi^2|d) \equiv 1 - P(\chi^2|d) = \frac{1}{\Gamma(d/2)} \int_{\chi^2/2}^{\infty} e^{-t} t^{d/2-1} dt \,. \tag{7.62}$$

$Q(\chi_0^2|d)$ 给出了某个遵从 d 个自由度的 χ^2 分布的随机变量 χ^2 的数值大于或等于一个给定的 χ_0^2 的概率。相应的，$P(\chi_0^2|d)$ 则给出了同样分布的随机变量 χ^2 的数值**不大于**给定 χ_0^2 的概率。这些函数对于我们分析一个拟合的质量是十分重要的。一般来说，函数 $Q(\chi^2|d)$ 被称为**拟合质量** (quality of fit) 而函数 $P(\chi^2|d) = 1 - Q(\chi^2|d)$ 则被称为相应拟合的**p-值** (p-value)。

由于拟合问题中我们有 n 个待定的参数 $x \in \mathbb{R}^n$，因此如果 $m < n$ 我们往往可以通过调节参数 x 以获得 χ^2 函数最小可能的值：0。这时这个问题其实根本不是一个典型的拟合问题。拟合问题往往出现在 $m > n$ 的情形下。这时公式 (7.59) 中的 χ^2 并不是 m 个独立的正态分布随机变量的平方和，真正独立的个数是所谓的自由度数目：

$$d := m - n \,. \tag{7.63}$$

因此，对于拟合问题来说，其 χ^2 遵从自由度数目为 $d = m - n$ 的 χ^2 分布。这时我们一般来说无法通过调节待定参数 x 使得 χ^2 取为零，而是通过将其取极小值 χ^2_{min} 来获得对参数 x 的最似然估计。

由于 χ^2 分布是 d 个正态分布量的平方和，因此粗略来说我们应当有：

$$\chi^2/d \sim 1 \,, \tag{7.64}$$

也就是说，作为一个粗略的估计 (所谓的 “rule of thumb”)，每个自由度大约贡献数量级为 1 的 χ^2 值是应当是可以接受的。如果需要更为细致的判据，那么我们就需要了解函数 $Q(\chi^2|d)$ 的数值了。我们在后面会再次回到这个问题。

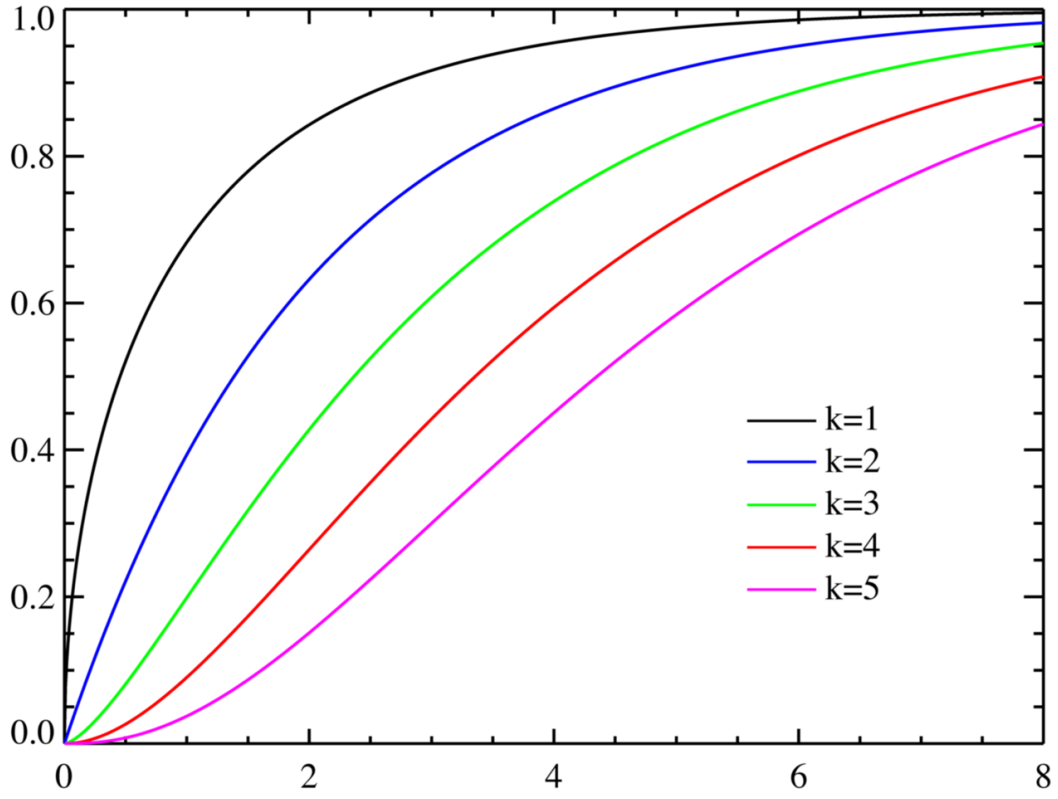


图 7.2: 图中显示了函数 $Q(\chi^2|k)$ 和 $P(\chi^2|k) = 1 - Q(\chi^2|k)$ (就是将纵坐标从上往下读!) 在 $k = 1, 2, 3, 4, 5$ 时的大致行为。

28.3 统计独立的 χ^2 拟合

¶我们首先讨论最为简单的情形，即所谓线性最小 χ^2 拟合的问题。这时我们的测试函数 $f(z;x)$ 关于待定参数 x 是线性的。如果我们用记号：$f_i(x) \equiv f(z_i;x)$，那么这个假设意味着存在一个 $m \times n$ 的矩阵 $A \in \mathbb{R}^{m\times n}$ 使得

$$\frac{f_i(x)}{\Delta y_i} = \sum_{j=1}^{n} A_{ij}x_j \ , \quad i = 1, 2, \cdots, m \ . \tag{7.65}$$

这个矩阵 A 称为该拟合问题的设计矩阵 (design matrix)。按照前面的介绍，这时的 χ^2 函数可以表达为

$$\chi^2 = (\tilde{y} - Ax)^T(\tilde{y} - Ax) \equiv ||\tilde{y} - Ax||^2 \ , \tag{7.66}$$

其中 $\tilde{y} \equiv y_i/\Delta y_i$ 而 $||\cdot||$ 表示 $\mathbb{R}^m$ 中的标准的欧氏模。线性拟合问题就是对于给定的 A 以及 $\tilde{y}$，求出使得 χ^2 最小的参数 x。

需要特别指出的是，上述的线性是指关于待拟合参数 x 的线性，而不是关于控制参数 z 的线性。换句话说，函数 $f(z;x)$ 关于 z 完全是任意的，只需要它关于 x 是线性的就

够了。

线性最小 χ^2 拟合问题的解一定存在并且满足正规方程：

$$(A^T A)x = A^T \tilde{y} \,, \tag{7.67}$$

可以证明，如果矩阵 A 的各个列在 $\mathbb{R}^m$ 中线性无关，那么矩阵 $A^T A$ 一定正定且非奇异，此时正规方程一定存在一个唯一的解：

$$x = (A^T A)^{-1} A^T \tilde{y} \,. \tag{7.68}$$

矩阵 $A^T A$ 实际上有非常直观的物理含义。记住 $\tilde{y}$ 是满足正态分布的随机变量，即 $E[\tilde{y}] = \mu$，$\mathrm{Cov}(\tilde{y}, \tilde{y}^T) = \mathbb{1}_{m\times m}$。因此，如果我们将解 $x = (A^T A)^{-1} A^T \tilde{y}$ 也视为随机变量，我们有 $E[x] = (A^T A)^{-1} A^T \mu$，而它的协方差矩阵为：

$$E[(x - E[x])(x - E[x])^T] = (A^T A)^{-1} \,. \tag{7.69}$$

因此矩阵 $(A^T A)^{-1}$ 就是拟合出的参数的协方差矩阵。

拟合问题中的设计矩阵的行为决定了一个拟合问题的难易程度。一般来说，如果 A 的各个列线性无关，那么 $(A^T A)$ 远离奇异，这时我们只需要用普通的线性代数的方法求解正规方程 (7.67) 即可。常用的方法包括：Gauss-Jordan 消元法、LU 分解法、Cholesky 分解等。而如果 $(A^T A)$ 很接近于奇异，这时候比较安全的做法是利用设计矩阵的奇异值分解：

$$A = UWV^\dagger \,, \tag{7.70}$$

其中 U 和 V 分别是 $m\times m$ 和 $n\times n$ 幺正矩阵，而 $m\times n$ 的对角矩阵 W 则包含了 A 的奇异值。这个时候，人们可以定义一个矩阵 A 的赝逆 (pseudo-inverse)，又称为 Moore-Penrose 赝逆：

$$A^+ = VW^+ U^\dagger \,. \tag{7.71}$$

其中对角矩阵 W 的赝逆 W^+ 的定义就是将其非零的对角元取倒数然后再转置。利用赝逆可以将线性最小 χ^2 拟合的解写为：

$$x = A^+ \tilde{y} \,. \tag{7.72}$$

关于奇异值分解的原理的稍微详细一些的讨论，大家可以参考本讲义的第 22节。奇异值分解的计算虽然可能更耗费时间，但是它一般会更稳定。

28.4 统计相关的 χ^2 拟合

¶ 如果测量直接有统计关联，那么相应的拟合需要重新考虑。一般来说，这时我们需要构建的 χ^2 函数大体上是如下的形式：

$$\chi^2 = \sum_{i,j=1}^{m} (y_i - f_i(x)) C_{ij}^{-1} (y_j - f_j(x)) \,, \tag{7.73}$$

其中的 $m \times m$ 矩阵 C_{ij} 就是变量 y 的协方差矩阵：$Cov(y_i, y_j) = C_{ij}$。它可以利用已有的数据进行估计。这个矩阵一定是正定对称的矩阵，因此它存在逆 C_{ij}^{-1}。一旦求得了协方差矩阵，我们就可以构造拟合问题的 χ^2 并求出问题的解。事实上我们定义矩阵 C 的逆根号：$C^{-1/2}$。这样一来上面的 χ^2 实际上可以写为

$$\chi^2 = (y - f(x))^T C^{-1/2} C^{-1/2} (y - f(x)) = (\tilde{y} - \tilde{A}x)^T (\tilde{y} - \tilde{A}x) \tag{7.74}$$

其中 $\tilde{y} = C^{-1/2} y$，$\tilde{A} = C^{-1/2} A$，这样这个问题就化为上小节讨论的统计独立情况的拟合问题。因此唯一需要解决的是估计出协方差矩阵 C。

协方差矩阵实际上需要多次的实验来进行确定。前面提到的测量值 y_i 及其误差 Δy_i 实际上也是通过多次实验测定的。为此我们假定在第 l 次实验中，我们测定了 y_i，$i = 1, 2, \cdots, m$，它的数值是 $y_i^{(l)}$，其中 $l = 1, 2, \cdots, N$ 标志我们实验的次数，而 $i = 1, 2, \cdots, m$ 则标记是在测量 y_i 这个物理量 (相应的控制参量为 z_i，我们假定在 N 次实验中这个保持不变)。我们进一步假定不同的 i 之间是统计相关的，但是不同次测量之间，也就是不同的 l 之间是统计独立的。[23] 于是，y_i 和 y_j 之间的协方差可以利用下式来估计，

$$Cov(y_i, y_j) \simeq \frac{1}{N} \sum_{l=1}^{N} (y_i^{(l)} - \bar{y}_i)(y_j^{(l)} - \bar{y}_j) \,, \tag{7.75}$$

其中 $\bar{y}_i = (1/N) \sum_{l=1}^{N} y_i^{(l)}$ 是各个 y_i 的在 N 次测量中的样本平均值。

公式 (7.75) 中构建的协方差矩阵是个正定、对称实矩阵。因此必定可以对角化并且其本征值均为正实数。当然，这并不意味着该矩阵不能接近奇异。在真实的应用中，很不幸这看上去的小概率事件经常发生。这时候我们第 6节中讨论的 Cholesky 分解或第 22节中涉及的奇异值分解很可能会是有用的。无论如何，在确定了协方差矩阵 C 并搞定 $C^{-1/2}$ 之后，我们就可以按照前面的方法，对公式 (7.73) 和公式 (7.74) 中定义的 χ^2 进行拟合操作了。

28.5 非线性的 χ^2 拟合

¶前面的讨论涉及到的都是所谓线性 χ^2 拟合，它由一个设计矩阵 A(如果有统计关联，还要加上协方差矩阵 C) 刻画。但是，在许多应用中也需要用到非线性拟合。这时一般就必须利用迭代的方法来求 χ^2 的极小值了。

这里求函数极小值的方法虽然原则上与前面讨论的非线性函数极小值的方法没有本质的不同。但是目前我们有一个优势：一个一般的多元函数求极小值问题并不一定能够计算函数的梯度，它的 Hessian 矩阵等等信息。但是这里，由于我们待拟合的函数 $f(z; x)$ 的形式是我们选定的，因此 χ^2 对于参数 x 的各阶导数是可以直接求出来的。你不会变态到猜测描写一堆数据的函数竟是一个你根本不知道的、很奇葩的函数，对吧？因此，在这类

[23] 如果这点不满足，我们总是可以将数据进行适当的 blocking 操作，使得剩下的数据块满足这个要求。参加第 26节中的讨论。

问题中我们一般会假定函数的导数以及二阶导数 (Hessian 矩阵) 的信息是可以比较轻易计算出来的。目前比较常用的是所谓的 Levenberg-Marquardt 方法。由于课时的原因，我们这里就不再深入讨论了，有需要的同学可以参考 [1] 的 §15.5 节中的讨论。

28.6 χ^2 拟合结果的统计诠释与置信水平

¶ 前面提到了 χ^2 中的函数 $Q(\chi^2|d)$ 可以用来刻画一个拟合的质量。也就是说，即使我们的函数 $f(z;x)$ 是完全正确的，我们通过 m 次测量 y_i(以及相应的误差 Δy_i)，然后对 n 个待定参数 x 进行最小 χ^2 拟合，结果获得一个不小于 $\chi^2_{\min}$ 的概率由 $Q(\chi^2_{\min}|(m-n))$ 给出。这个概率大概到什么程度被认为是可信的呢？这个当然是个仁者见仁、智者见智的事情。但是一般的“常识”还是有的。

- 首先，$\chi^2_{\min}$ 的大/小，并一定能够否定/肯定我们的理论公式：$y = f(z;x)$，因为过大的 $\chi^2_{\min}$ 除了可能由理论模型不正确引起，也可能由过小估计了误差 Δy_i 引起。类似地，小的 $\chi^2_{\min}$，例如满足我们所说的 $\chi^2_{\min}/d \sim 1$，也可能是由于我们过高地估计了 Δy_i 引起。因此，只有在所有的误差都正确估计之后，再来谈 $\chi^2_{\min}$ 的大小才是合适的。另一个值得一提的是统计相关性。例如，在应当进行统计相关数据的拟合时错误地运用了统计独立数据的拟合，就很可能错误地估计 χ^2 的数值从而造成不必要的误解。

- 假设上面提到的这些问题都已经避免了。那么一般的主流观点认为：

  - $Q \gtrsim 0.1$：一般被认为是可信的；
  - $Q < 0.1$ 但 $Q \gtrsim 0.001$：一般认为还是可以接受的，特别是有一些迹象表明误差可能被低估了话；
  - $Q < 0.001$：一般被认为是不可信的，除非你有其他的理由；

¶ 利用最小 χ^2 拟合不仅仅可以告诉我们最佳的参数 x，还可以用来设置置信水平。我们假定 $\chi^2(x)$ 在其极小值 x_0 附近可以进行展开，利用 $x = x_0 + \delta x$：

$$\Delta\chi^2_{\min} = \chi^2(x) - \chi^2_{\min} = \delta x^T (A^T A)\delta x \,. \tag{7.76}$$

于是我们发现在 $\mathbb{R}^n$ 中，等 χ^2 的点构成了以最佳点 x_0 为中心的一个椭球面。利用这些椭球面我们可以设置所谓的置信水平。值得注意的是，如果我们仅仅考虑线性拟合，那么上式实际上是严格的。也就是说等 χ^2 的曲面一定是一个椭球面，其各个轴的大小和取向完全由 x 的协方差矩阵 $(A^T A)^{-1}$ 确定。如果考虑到非线性拟合，那么上式只是在极小值点附近的一个邻域内成立。无论如何，在 $\mathbb{R}^n$ 中，δx 遵从一个 n-维的正态分布：

$$P(\delta x)d^n x \propto \exp\left(-\frac{1}{2}\delta x^T (A^T A)\delta x\right) d^n x \,. \tag{7.77}$$

表 7.1: $\Delta\chi^2$ 的数值在不同自由度数目 d 的情况下与概率置信水平的对应关系。

	d					
p (%)	1	2	3	4	5	6
68.3 (1σ)	1.00	2.30	3.53	4.72	5.89	7.04
95.4 (2σ)	4.00	6.17	8.02	9.70	11.3	12.8
99.7 (3σ)	9.00	11.8	14.2	16.3	18.2	20.1

人们往往用一个自由度情形下的概率分布来类比定义置信水平。例如，对于一个自由度的情形，$\Delta\chi^2 = 1, 4, 9$ 的时候分别对应于 68.3%，95.4% 和 99.7% 的概率。[24] 人们通常称它们为 1σ，2σ 和 3σ 的置信水平。需要引起大家注意的是，$\Delta\chi^2$ 的数值与固定的概率之间的关系是依赖于自由度数目的，虽然 1σ，2σ 和 3σ 的定义是在 $d = 1$ 情形下制定的。如果我们仍然沿用几个 σ 与概率大小的对应关系，那么在 $d \neq 1$ 的情况下，$\Delta\chi^2$ 的数值也将不同。在表 7.1中我们列出了不同自由度数目 d 的情况下，相应的 $\Delta\chi^2$ 与不同置信水平的对照表。例如，在 $d = 3$ 的情况下，如果要达到 1σ(即 68.3%) 的置信水平，我们要求 $\Delta\chi^2 = 3.53$，2σ 对应于 $\Delta\chi^2 = 8.02$ 等等。[25]

有时候我们虽然有 n 个拟合参数，但是我们最后关心的只是其中的 $\nu < n$ 个参数的置信水平，对于其他的 $n - \nu$ 个参数的取值并不关心。也就是说，如果我们首先将 ν 个参数固定，对其他的 $n - \nu$ 个参数求极小，这时得到的 χ^2 极小值记为 χ^2_ν。显然，这个值比起对所有参数求极小后获得的 $\chi^2_{\min}$ 要大一些。两者的差我们记为 $\Delta\chi^2_\nu = \chi^2_\nu - \chi^2_{\min}$。可以证明，$\Delta\chi^2_\nu$ 遵从自由度数目为 ν 的 χ^2 分布。因此，我们在设置这 ν 个参数的置信水平的时候，应当按照 ν 个自由度的 χ^2 分布的置信水平来进行。图 7.3中显示了这样的一个例子。假设我们一共有两个参数 ($n = 2$)，但是我们仅仅关系其中一个参数 (具体来说是 x_2) 的置信水平 ($\nu = 1$)，对于另一个参数 (x_1) 的取值我们并不关心。图中的原点对应于在两个参数空间中使得 χ^2 取得极小值 $\chi^2_{\min}$ 的点。稍大的两个椭圆对应于 $d = 2$ 时的 1σ 和 2σ 等置信水平线。假设我们关心 x_2 的 1σ 置信水平线。这应当选择 $\Delta\chi^2 = 1$ 的椭圆 (砖红色的椭圆)，再向 x_2 轴进行投影得到。类似地，如果我们要寻找 x_2 的 2σ 的置信水平，我们应当画出 $\Delta\chi^2 = 4$ 的椭圆 (图 7.3中并没有画出来，如果要画出来的话，它应当在两个稍

[24] 其实就是一个标准高斯分布 $\mathcal{N}(0, \sigma)$ 的情形下位于 $[-\sigma, +\sigma]$，$[-2\sigma, +2\sigma]$ 和 $[-3\sigma, +3\sigma]$ 之间所包含的概率。因此才有 1σ，2σ 和 3σ 的名称。

[25] 这里顺便提一下在粒子物理中的一个约定。当一个实验组声称发现一个新粒子的时候 (例如 2012 年声称的 Higgs 粒子)，一般要求置信水平达到 5σ 以上。如果只有 3σ 但没有达到 5σ 则往往只能称看到了迹象 (hint)。类似的约定也出现在发现偏离标准模型的信号上。如果偏离没有超出 3σ，基本上都没有人会认真对待。这种现象我称之为 σ 贬值 (σ devaluation) 现象。其主要的原因在于：我们这里仅仅讨论了统计误差，并没有讨论系统误差。在粒子物理实验中，困难的往往在于系统误差的估计。即使我们很确信一个实验中的误差是由统计误差导致的，我们还需要假设各个量是高斯分布的。但真实的物理量并不一定满足这一点。总之，考虑到所有这些因素以及结合历史上的经验和教训告诉我们，3σ 是一个合理的门槛。

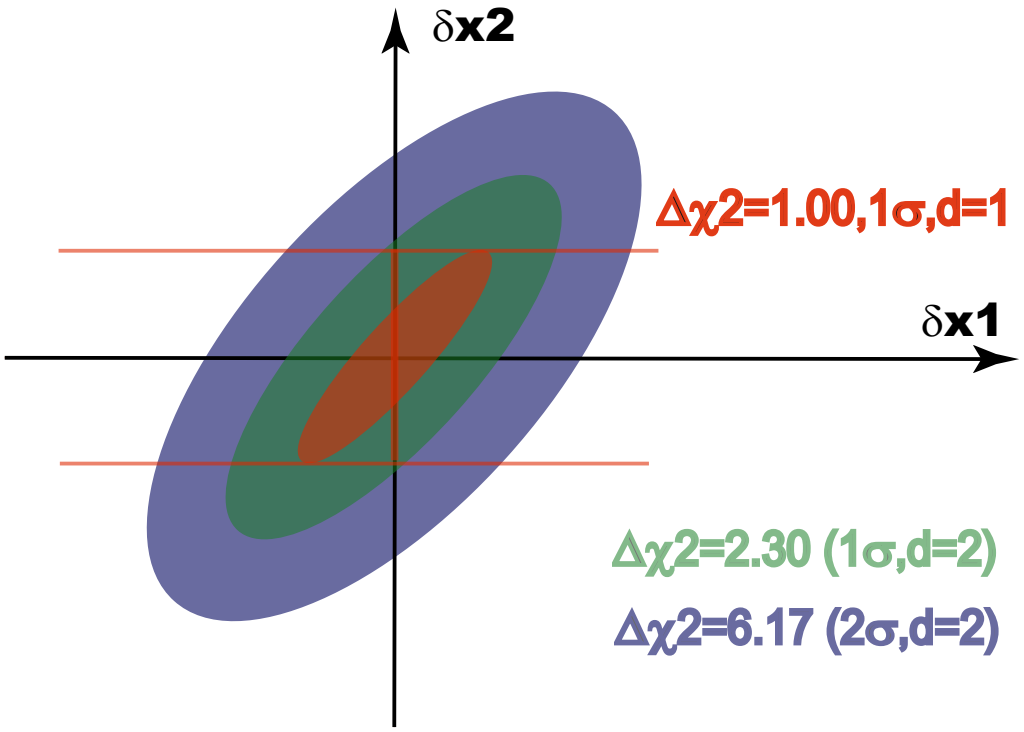


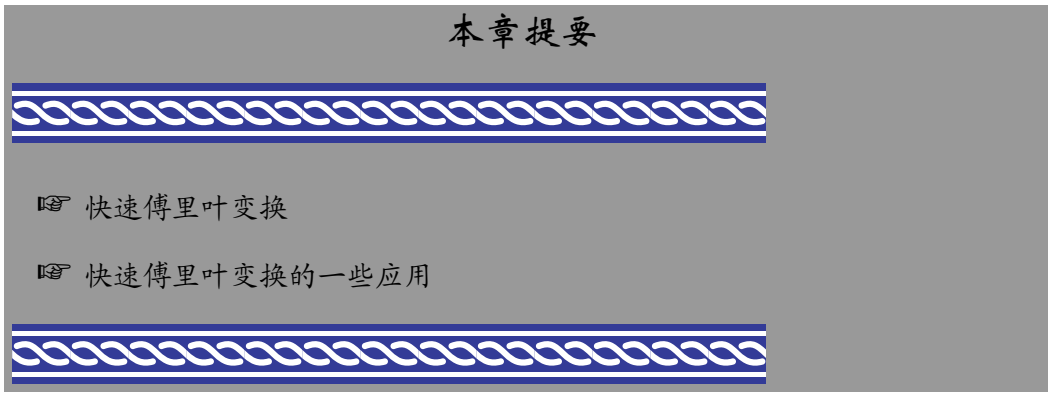
图 7.3: 部分参数置信水平设定的一个例子。我们一共有两个参数 ($n = 2$)，但是我们仅仅关系其中一个参数 (具体来说是 x_2) 的置信水平 ($\nu = 1$)，对于另一个参数 (x_1) 的取值我们并不关心。原点对应于两个参数空间中使得 χ^2 取得极小值 $\chi^2_{\rm min}$ 的点。稍大的两个椭圆对应于 $d = 2$ 时的 1σ 和 2σ 等置信水平线。我们如果只关心 x_2 的 1σ 置信水平线，则应当选择 $\Delta\chi^2 = 1$ 的椭圆 (砖红色的椭圆)，再向 x_2 轴进行投影得到。

大的椭圆之间) 然后再向 x_2 轴投影得到。

相关的阅读

这一章中我们着重讨论了数据处理中的一系列问题。我想所有学习科学的同学都会或多或少地遇到数据处理方面的问题，特别是对于学习物理学的同学而言，这类问题还特别经常性地、大规模地出现。因此我希望这章的讨论对于大家有所帮助。再学习完这一章后大家可以回忆一下，你们在普物实验中是否按照这里讨论的方法处理了你们的数据呢?

第八章　　快速傅里叶变换方法



本章中我们讨论应用十分广泛的快速傅里叶变换。傅里叶变换的运用是非常广泛的，基本上涵盖了整个科学和工程领域。连续的傅里叶变换一般可以视为分立版本的傅里叶变换的极限。因此我们将主要讨论分立的傅里叶变换。尽管分立的傅里叶变换可以适用于任意的维度，我们的讨论将局限于一维的情形。任意有限维度的分立傅里叶变换在适当地标记其点阵之后都可以化为一维的情形。所以对一维的讨论并不失去一般性。

对于一个周期长度为 N 的数据 $\{f(t) : t = 0, 1, 2, \cdots (N-1)\}$，这个可以是某个一维晶格上的物理量或者其他周期函数在一个周期内的测量值。我们已经将测量的间隔取为 1，而其周期为 N。这个分立数据的傅里叶变换定义为，

$$F(k) = \sum_{t=0}^{N-1} e^{2\pi itk/N} f(t) \ , \quad k = 0, 1, \cdots, N-1 \ , \tag{8.1}$$

其中我们运用了约定：F 代表相应的 f 的傅里叶变换。[1] 它的逆变换为，

$$f(t) = \frac{1}{N} \sum_{k=0}^{N-1} e^{-2\pi itk/N} F(k) \ , \quad t = 0, 1, \cdots, N-1 \ . \tag{8.2}$$

如果将原始的数据 $\{f(t) : t = 0, 1, 2, \cdots (N-1)\}$ 视为长度为 N 的一个矢量，那么傅里叶变换相当于是这个矢量空间的一个线性变换 (事实上是一个幺正变换)。由于变换矩阵本身是一个满阵，因此这个计算量的数量级是 $O(N^2)$。对于非常庞大的 N，这个计算量还是非常惊人的。我们下面要介绍的快速傅里叶变换将能够将计算量减少到 $O(N \ln N)$。

[1] 另外一种常用的约定方式是在相应的字母上方加上一个弯。因此我们这里的约定等同于：$F(k) = \tilde{f}(k)$。

29 快速傅里叶变换

¶快速傅里叶变换的基本原理其实很简单，基本上就是分而治之 (divide and conquer, D&C) 的方法。为了方便，我们将 N 个数据的相因子记为，

$$W_N \equiv e^{2\pi i/N} , \tag{8.3}$$

它实际上是 1 在复平面的 N 次方根。那么我们可以将傅里叶变换 (8.1) 写为，

$$F(k) = \sum_{t=0}^{N-1} (W_N)^{tk} f(t) . \tag{8.4}$$

我们可以进一步将求和中的奇数和偶数的 t 分开，

$$\begin{aligned} F(k) &= \sum_{j=0}^{N/2-1} (W_N)^{2jk} f(2j) + \sum_{j=0}^{N/2-1} (W_N)^{(2j+1)k} f(2j+1) \\ &= \sum_{j=0}^{N/2-1} (W_{N/2})^{jk} f(2j) + (W_N)^k \sum_{j=0}^{N/2-1} (W_{N/2})^{jk} f(2j+1) \\ &= F^e(k) + (W_N)^k F^o(k) . \end{aligned} \tag{8.5}$$

在这个式子中，我们将一个长度为 N 的傅里叶变换改写为两个长度为 $N/2$ 的傅里叶变换的线性组合。这个公式又被称为 Danielson-Lanczos 引理 (Danielson-Lanscos lemma)。它的最大作用在于，这种改写可以继续迭代地进行下去。换句话说，我们可以进一步将 $F^e(k)$ 和 $F^o(k)$ 的计算进一步分解为 $F^{ee}(k)$，$F^{eo}(k)$ 以及 $F^{oe}(k)$，$F^{oo}(k)$ 的计算，它们对应于长度为 $N/4$ 的傅里叶变换。这四个量又可以进一步表达为另外 8 个两的傅里叶变换的计算，只不过长度变为 $N/8$，如此等等。虽然对于任意的 N 都可以进行快速傅里叶变换，但是显然最为方便的是假设 $N = 2^m$，这时上述 Danielson-Lanscos 分解可以一直分解到最后。我们最后需要计算的是长度为 0 的傅里叶变换，即压根不需要变换。换句话说，如果 $N = 2^m$，我们一定有

$$F^{eo\cdots oe}(k) = f(n) \quad \text{对某个 } n , \tag{8.6}$$

我们需要做的，就是找出具体的 n 与一串 $(eo\cdots oe)$ 之间的对应关系。

这个对应关系是这样的：将特定的一串字符 $(eo\cdots oe)$ 的顺序完全反过来，也就是从右向左写出它们，然后令 $e = 0$，$o = 1$，这一串 0 和 1 就恰好对应于整数 n 的二进制表达。正因为如此，这个次序又称为位翻转序 (bit-reversed order)。因此，快速傅里叶变换中一个重要的步骤就是将输入的数据首先按照位翻转序排好。比如对于长度为 8 的数据，原先的排列顺序是

$$000\ 001\ 010\ 011\ 100\ 101\ 110\ 111 \tag{8.7}$$

所谓的位翻转序是将其排为，

$$000\ 100\ 010\ 110\ 001\ 101\ 011\ 111 \tag{8.8}$$

即将原先的序列中的第 2 个和第 5 个，第 4 个和第 7 个对调一下即可。

下面是关于 FFT 的几点说明

- 前面我们假定了 N 恰好是 2 的幂次。事实上这是最为简单的情形。如果 N 不是 2 的幂次的话，一种简单的处理方法是将 N 扩展到下一个 2 的幂次然后利用 0 将其他的数据填充起来。虽然看起来这样仿佛挺浪费的，但是 FFT 为你赢得优势往往远好于做普通的傅里叶变换。当然，更为精细的做法是对 N 进行质因数分解，例如对于 $N = 2^8 \cdot 3^4$，这就需要处理基底为 3 的傅里叶变换。相应的程序 (还包括基底为 5 的) 如果需要在网上也是可以找到的。

- 前面的讨论中我们假定一直迭代到长度为 1 的傅里叶变换。这一点并不是必须的。有一类 FFT 程序是迭代到长度是 4 的傅里叶变换。这主要是得益于 W_4 的特殊性：它要么是直接将结果乘以一个可能的符号，要么就是将实部虚部互换后再乘以一个可能的符号。因此，完全可以很快地计算。利用这点，往往可以比通常一直迭代到最后要快 20% 左右。

- 傅里叶变换一般来说是一个复的变换。如果我们变换的是一个实的函数，那么相应的变换中具有一定的对称性可以利用。一个常用的办法是将两个待变换的实函数集合成一个复函数，FFT 这个复函数，然后同时获得两个函数的傅里叶变换。例如，如果我们需要做两个实函数 $f(t)$ 和 $g(t)$ 的长度为 N 的傅里叶变换，我们可以构造复函数

$$h(t) = f(t) + ig(t) \,, \quad t = 0, 1, \cdots, N-1 \,, \tag{8.9}$$

然后直接计算复函数 $h(t)$ 的傅里叶变换：

$$H(k) = \sum_{t=0}^{N-1} e^{2\pi ikt/N} (f(t) + ig(t)) \,, \tag{8.10}$$

由于 $f(t)$ 和 $g(t)$ 是实的，我们很容易发现下列关系成立，

$$H(k) + H(N-k)^* = 2F(k) \,, \quad H(k) - H(N-k)^* = 2iG(k) \,, \tag{8.11}$$

其中 $F(k)$ 和 $G(k)$ 分别是实函数 $f(t)$ 和 $g(t)$ 的傅里叶变换。

如果我们不需要计算两个实函数的傅里叶变换，只想计算一个函数的傅里叶变换，也可以将其分为奇数偶数的顺序分别放入一个长度为 $N/2$ 的复函数，然后对复函数进行长度为 $N/2$ 的傅里叶变换。具体来说就是令，

$$h(t) = f(2t) + if(2t+1) \,, t = 0, 1, \cdots, N/2 - 1 \,. \tag{8.12}$$

其傅里叶变换为 $H(k) = F^e(k) + iF^o(k)$，其中

$$\begin{aligned} F^e(k) &= \sum_{t=0}^{N/2-1} f(2t) e^{2\pi itk/(N/2)} \,, \\ F^o(k) &= \sum_{t=0}^{N/2-1} f(2t+1) e^{2\pi itk/(N/2)} \,, \end{aligned} \tag{8.13}$$

而原先实函数 $f(t)$ 的傅里叶变换可以从下面的公式获得：

$$F(k)=\frac{1}{2}(H(k)+H(N/2-k)^*)-\frac{i}{2}(H(k)-H(N/2-k)^*)e^{2\pi ik/N}\ , \qquad (8.14)$$

其中 $k=0,1,\cdots,N-1$.

¶ 前面的讨论我们假定所有的计算都是在一个独立的计算机上面完成的。另外一个大型计算中经常遇到的是采用并行计算的情形。这时候由于不同的节点之间必须进行通信，因此计算量之外还必须考虑通信的影响。这就是我们下面简单讨论的并行计算中的 FFT 的应用问题。假定我们需要进行傅里叶变换的矢量长度 N 很大，以至于我们希望将它分散在不同的并行节点上进行计算。假定 $N=m\times M$ 其中我们假定 m 和 M 都是 2 的幂次 (从而 N 也是)。于是我们待变换的数据可以记为，

$$f(Jm+j)\ ,\quad 0\le j<m\ ,\ 0\le J<M\ . \qquad (8.15)$$

这些数据是存放在不同的节点上的，例如 M 个节点上，每个节点上面有 m 个数据。当然也可以是存放在 m 个节点上，每个节点有 M 个数据。这完全视具体情况而定。同样，它的傅里叶变换也可以记为 $F(kM+K)$，其中的 k 和 K 与 m 和 M 类似，是频率空间的指标：

$$\begin{aligned}F(kM+K)&=\sum_{j,J}e^{2\pi i(kM+K)(Jm+j)/(mM)}f(Jm+j)\\&=\sum_j\left\{e^{2\pi ijk/m}\left[e^{2\pi ijK/(mM)}\left(\sum_J e^{2\pi iJK/M}f(Jm+j)\right)\right]\right\}\end{aligned} \qquad (8.16)$$

这个公式基本上指出了如何在并行机上计算 FFT 的步骤，只要我们从上面公式的最里层逐渐往外进行操作就可以了。

- 对于每个给定的 j，构造一个长度为 M 的矢量 $f(Jm+j)$，$0\le J<M$。
- 对每个这样的矢量进行长度为 M 的 FFT 操作。这样指标 J 就变为了频率空间的指标 K；
- 对每个分量乘以相因子 $\exp[2\pi ijK/(mM)]$；
- 将数据排为一个由 j 标记的矢量，$0\le j<m$；
- 对该矢量进行 FFT，于是指标 j 就变为了频率空间的指标 k；
- 原来矢量的傅里叶变换就在 $F(kM+K)$ 之中

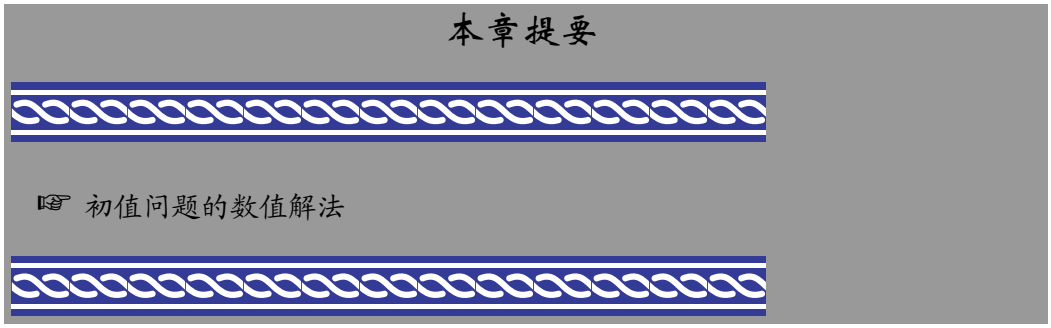
注意在上述 6 个步骤之中，取决于原先数据的存储方式，其中第一、第四和第六个步骤中可能会涉及到节点之间的通讯步骤。它们被统称为换序操作 (transpose operation)。换序操作的速度依赖于所使用的计算机的通信的带宽等因素，很难一概而论。但通常的并行机上

面通讯往往会占据相当的时长，因此我们应当尽可能地减少节点之间的通讯。容易验证，仅仅从计算量来看，这个 FFT 的计算量仍然是 $N\ln M + N\ln m = N\ln N$。

相关的阅读

这一章我们简单介绍了快速傅里叶变换及其应用。

第九章　初值问题的数值解法



本章我们讨论常微分方程的初值问题的数值解法。原则上常微分方程也可以求解边值问题。这个我们后面再讨论。

任何有限阶的常微分方程总可以化为下列的一般的常微分方程组：

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(t;\mathbf{y})\ ,\mathbf{y}(t_0) = \mathbf{y}_0\ . \tag{9.1}$$

其中我们待求的函数记为 $\mathbf{y}(t) \in \mathbb{R}^n$ 为 n 维实空间的函数，$\mathbf{f}(\cdot,\cdot) \in \mathbb{R}^n$ 则是一个已知的函数。我们还要求函数 $\mathbf{y}(t)$ 并且满足相应的初始条件：$\mathbf{y}(t_0) = \mathbf{y}_0$ 而 $\mathbf{y}_0 \in \mathbb{R}^n$ 为 n 维空间任意一个矢量。我们知道在相当一般的条件下，这个初值问题的解存在并且唯一。我们一般将假设这些条件是成立的并进而讨论如何利用数值的方法来求解这个方程。

虽然初值问题可以在任意的 $\mathbb{R}^n$ 中来讨论，我们将主要局限在 $n = 1$ 的简单情形。这时候一般的初值问题可以表达为求出某个实数区间 I 上的函数 $y \in C^1(I)$ 使其满足，

$$\begin{cases} \dfrac{dy}{dt} = f(t;y(t))\ , \quad t \in I\ , \\ y(t_0) = y_0\ , \end{cases} \tag{9.2}$$

其中 $f(t,y)$ 是条状区域 $S = I \times (-\infty,+\infty)$ 上的已知连续函数。这个问题通常被称为 Cauchy 问题。如果 f 不显含 t，我们就称该微分方程是自治的 (autonomous)。与 Cauchy 问题之微分方程等价地是其积分方程，

$$y(t) = y_0 + \int_{t_0}^{t} f(\tau, y(\tau))d\tau\ . \tag{9.3}$$

作为数值解法我们必须考虑其稳定性。为此我们也需要考虑微扰后的 Cauchy 问题：

$$\begin{cases} \dfrac{dz}{dt} = f(t; z(t)) + \delta(t) \ , \quad t \in I \ , \\ z(t_0) = y_0 + \delta_0 \ , \end{cases} \tag{9.4}$$

其中我们将初始条件 y_0 和中间的函数 f 都做了微扰。我们的解是否稳定关键就看如果初始条件和函数的微扰都足够小，微扰后的解 $z(t)$ 是否足够接近 $y(t)$。为此，我们引入所谓的 Liapunov 稳定性的定义。如果对于任意给定的 $\epsilon > 0$，我们选取

$$|\delta_0| < \epsilon \ , |\delta(t)| < \epsilon \ , \forall t \in I \ , \tag{9.5}$$

如果存在一个与 ϵ 无关的常数 C 使得

$$|z(t) - y(t)| \leq C\epsilon \ . \ \forall t \in I \ , \tag{9.6}$$

我们就称原先的解 $y(t)$ 是 Liapunov 稳定的，或者称具有 Liapunov 稳定性。

30 各种单步算法的描述

¶ 我们首先介绍各种求解常微分方程的初值问题是在一步之中的算法。随后，这样的一步可以简单地重复多步，或者如果问题需要，可以调整步长再继续。但无论如何第一步总是需要的。

一个数值积分常微分方程的近似数值方法中，如果获得 y_{n+1} 的步骤仅仅依赖于前一步的 y_n，那么这种算法就被称为一个单步算法。否则，就称为多步算法。

最为简单的算法称为 Euler 方法，它从一个简单的数值 t_0 开始，直接计算该处的导数值并前行，令 $t_n = t_0 + nh$，其中 $h \in \mathbb{R}$ 是积分的步长，那么我们有，

$$y_{n+1} = y_n + hf(t_n, y_n) \ . \tag{9.7}$$

这个算法称为向前 Euler 方法，它的的精度只有 $O(h)$，实际上并不值得推荐。与此类似，我们还可以构建所谓的向后 Euler 方法：

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \ . \tag{9.8}$$

这个式子与前一个的区别在于，等号的右边也包含 y_{n+1}。因此，当给定了 y_n 以及 t_{n+1} 之后，原则上我们需要从上面的方程 (其中包含可能非线性的函数 f) 之中解出 y_{n+1} 的数值来。这个方法被称为隐式方法。与此相对应，如果 y_{n+1} 的计算只需要 $y_{m\leq n}$ 的数值便可以直接计算出来，那么该方法就称为显式方法。从这个意义上说，上面的向前 Euler 方法是一个显式的单步算法，而向后的 Euler 方法则是一个隐式的单步算法。

例如，一个比它好很多的方法是所谓的中点方法或者二阶 Runge-Kutta 方法：

$$\begin{cases} k_1 = hf(t_n, y_n) \\ k_2 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\ y_{n+1} = y_n + k_2 + O(h^3) \end{cases} \tag{9.9}$$

二阶的 Runge-Kutta 方法的每一步中需要计算两次微分方程右边的函数 $f(t, y)$。一般来说，这个函数的计算往往是计算之中最为耗时的。因此我们需要尽可能地减少其计算次数。二阶的 Runge-Kutta 方法优于前面提及的 Euler 方法是说，虽然前者每步需要计算两次而后者仅仅需要计算一次函数 $f(t, y)$，在同样精度的前提下，二阶的 Runge-Kutta 仍然比 Euler 要更为有效。

高级的 Runge-Kutta 方法也是经常被推荐的。其中人们最为推崇的是所谓的四阶 Runge-Kutta 方法。

$$\begin{cases} k_1 = hf(t_n, y_n) \\ k_2 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\ k_3 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2) \\ k_4 = hf(t_n + h, y_n + k_3) \\ y_{n+1} = y_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 + O(h^5) \end{cases} \tag{9.10}$$

31 误差估计与步长的调整

¶ 本节中我们讨论有限步长造成的误差以及如何调整步长。我们会看到这会给我们很多的自由度。

假定我们从某个 t_n 出发，利用单步方法将微分方程积分到 $t_n + 2h$。我们可以有两种方法：一种方法是直接利用步长为 $2h$ 的 Runge-Kutta，另一种方法是利用步长为 h 的 Runge-Kutta 两次。我们将第一个方法得到的近似解记为 y_1，第二个记为 y_2，如果我们将问题的严格解记为 $y(t_n + 2h)$，那么我们有，

$$\begin{aligned} y(t_n + 2h) &= y_1 + (2h)^5\phi + O(h^6) \\ y(t_n + 2h) &= y_2 + 2(h)^5\phi + O(h^6) \end{aligned} \tag{9.11}$$

其中对于小的 h，系数 ϕ 基本上在步长范围内是个常数。显然，两种方法的差值：

$$\Delta = y_2 - y_1 \ , \tag{9.12}$$

可以作为衡量截断误差的一个重要指标。如果忽略掉 h^6 的项，我们发现

$$\Delta \simeq 30h^5\phi + O(h^6) \ . \tag{9.13}$$

而严格的解可以利用下面的公式进行估计：

$$y(t_n+2h)=y_2+\frac{\Delta}{15}+O(h^6)\ . \tag{9.14}$$

一般的做法是，如果 $\Delta/15$ 小于某个给定的误差（但不显著小于），我们就继续走下一步；否则我们可以将步长减小一半在计算这个误差直至其满足要求。反之，如果误差小于要求误差的 $1/32$，我们也可以将步长加倍。这样就实现了步长的自动调节。

四阶的 Runge-Kutta 方法是比较常用的单步算法。一个原因就是对于一个 M 阶的单步算法而言，如果 $M>4$，那么一般来说我们需要计算方程右边的函数的次数会高于 M。只有对 $M=4$ 而言，四阶 Runge-Kutta 恰好需要计算 4 次函数值。但是，如果我们考虑到误差估计以及步长调整，那么常规的 4 阶 Runge-Kutta 算法就不一定是最优的了(尽管通常也是不错的)。例如，Fehlberg 就提出了一种内嵌 Runge-Kutta 方法 (embedded Runge-Kutta)，[1] 它本身是一个 5 阶的单步算法，这个算法中每一步需要计算函数值 6 次。但是它的好处是，在计算的这六个函数值之中，除了实现 5 阶的单步算法之外，它们的另外一种组合可以提供一个 4 阶的单步算法。换句话说，一个 5 阶的算法内部自动内嵌了一个 4 阶的算法。利用这两种单步算法对函数 $y(t+h)$ 进行计算，两者的差别就可以用来进行误差估计和步长调整。

一个 5 阶的 Runge-Kutta 方法大致如下，

$$\begin{cases} k_1=hf(t_n,y_n) \\ k_2=hf(t_n+c_2h,y_n+a_{21}k_1) \\ \quad\cdots \\ k_6=hf(t_n+c_6h,y_n+a_{61}k_1+\cdots+a_{65}k_5) \\ y_{n+1}=y_n+b_1k_1+b_2k_2+b_3k_3+b_4k_4+b_5k_5+b_6k_6+O(h^6) \end{cases} \tag{9.15}$$

相应的内嵌的 4 阶算法的估计是，

$$y^*_{n+1}=y_n+b^*_1k_1+b^*_2k_2+b^*_3k_3+b^*_4k_4+b^*_5k_5+b^*_6k_6+O(h^5)\ . \tag{9.16}$$

于是两者的差别为，

$$\Delta\equiv y_{n+1}-y^*_{n+1}=\sum_{i=1}^{6}(b_i-b^*_i)k_i\ . \tag{9.17}$$

上述算法的方案首先由 Fehlberg 提出，他还给出了具体的各个系数 (即 c_i，a_{ij}，b_i，b^*_i) 的取值。Numerical Recipes 推荐的是 Dormand-Prince 的改进版本。[2] 表 9.1中给出了相应的各个系数的取值。

[1] Fehlberg, E. 1964, "New high-order Runge-Kutta formulas with stepsize control for systems of first- and second-order differential equations", Z. Angew. Math. Mech., **44**, 17-29.

[2] Dormand, J.R. and Prince, P.J. 1980,"A Family of Embedded Runge-Kutta Formulae", Journal of Computational and Applied Mathematics, **6**, 19-26.

表 9.1: Dormand-Prince 方法：一种内嵌的 5 阶 Runge-Kutta 单步积分法。具体公式 (9.15) 和式 (9.16) 中的各个系数由下表给出。

i	c_i	a_{ij}						b_i	b_i^*
1								$\frac{35}{384}$	$\frac{5179}{57600}$
2	$\frac{1}{5}$	$\frac{1}{5}$						0	0
3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					$\frac{500}{1113}$	$\frac{7571}{16695}$
4	$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				$\frac{125}{192}$	$\frac{393}{640}$
5	$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			$-\frac{2187}{6784}$	$-\frac{92097}{339200}$
6	1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		$\frac{11}{84}$	$\frac{187}{2100}$
7	1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0	$\frac{1}{40}$
j		1	2	3	4	5	6		

仔细观察表 9.1你会发现，其中的指标 i 一直给到了 $i=7$，但是我们原始的公式 (9.15) 和公式 (9.16) 中的各个系数仅仅给到了 $i=6$。这个与所谓的首尾相同技巧 (first-same-as-last, FSAL) 有关。事实上，我们可以将公式 (9.15) 和公式 (9.16) 一直计算到包括 $i=7$ 的项，这相当于多计算了 $f(t_n+h, y_{n+1})$ 的函数值。但是这个函数值在数值积分的下一步反正是需要计算的。因此，平均到每一步我们仍然只需要计算 6 次函数值而不是 7 次。大家也许注意到了，表中 $i=7$ 的一行中的 a_{ij} 部分恰好与 b_i 的一列的内容完全相同 (见表中用浅色标出的两部分)，这正是 FSAL trick 所要求的。

既然误差近似地由 $|\Delta$ 给出，在程序的设计过程中我们可以要求，

$$|\Delta| = |y_{n+1} - y_{n+1}^*| \leq E_{tol} \ , \tag{9.18}$$

其中的 E_{tol} 是我们期望承受的误差 (tolerance)。这个误差当然可以是绝对的误差，或者是相对的误差。为了方便我们可以令，

$$E_{tol} = E_{abs} + \max(|y_n|, |y_{n+1}|) \cdot E_{rel} \ . \tag{9.19}$$

其中 E_{abs} 和 E_{rel} 是两个参数。如果我们要求绝对误差，我们可以令 $E_{rel}=0$，$E_{abs}=\epsilon$；反之，如果我们希望用相对误差，则可以令 $E_{abs}=0$，$E_{rel}=\epsilon$。这样可以比较方便地控制程序。

上面的讨论假定是只有一个微分方程。如果是一个微分方程组，误差 Δ 和 E_{tol} 也会推广为相应的高维空间的矢量。显然上面的绝对值需要替换为相应的高维空间中的模。尽管不同的模都可以被采用，但是最为常用的是无穷模和欧氏模。前者对应于选取最大偏离的误差而后者则是所有分量误差的一个方均根。以欧氏模为例，对于具有 N 个方程的微分方程组，我们可以定义，

$$\bar{E} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\Delta_i}{E_i}\right)^2} \ , \tag{9.20}$$

其中 Δ_i 和 E_i 是第 i 个分量的偏差和误差控制。有效且合理的微分方程解的条件是 $\bar{E} \leq 1$，其中的 N 个 E_i 由 $2N$ 个可调参数控制。我们首先选取一个初始的步长并估计上述误差 $\bar{E}$。如果 $\bar{E} \lesssim 1$，我们就接受它并且继续下一步；如果 $\bar{E} > 1$，我们就拒绝它并进而缩小步长。同样的，如果步长已经足够小以至于 $\bar{E} \ll 1$，我们则可以增加步长以提高效率。由于误差基本上正比于 h^5，所以如果两个不同步长 h_0 和 h_1 所对应的误差分别为 $\bar{E}_0$ 和 $\bar{E}_1$，那么我们显然有，

$$\frac{h_0}{h_1} = \left(\frac{\bar{E}_0}{\bar{E}_1}\right)^{1/5} . \tag{9.21}$$

这个式子告诉了我们在需要的时候如何调整步长。

32 多步方法的介绍

¶ 本节中我们将简要介绍一下所谓的多步方法。我们的介绍将比较简略。因为近年来多步算法的适用范围越来越少，它实际上仅仅出现在非常个例的应用中。更多的细节同学们可以参考 [2] 的 §7.2.6-7.2.13 等节。

首先澄清一点的是，这里谈及的多步方法并不是指将单步方法重复多次。在求解 ODE 初值问题中的所谓 多步方法 (multi-step method) 是指对初值问题，

$$\dot{y} = f(t, y(t)) \, , y(t_0) = y_0 \, , \tag{9.22}$$

来说，我们将自变量按照等间距地增加：$t_k = t_0 + kh$，我们在构造严格解 $y(t_{j+r})$ 的近似解 η_{j+r} 的时候，从之前的 r 个近似解 η_k，$k = j, j+1, \cdots, j+r-1$ 的信息出发，其中 $r \geq 2$。如果 $r = 1$，这就是前面介绍的单步方法。显然，要使得一个 r 步的多步算法能够进行，我们首先需要获得前 r 步的近似解：$\eta_0, \eta_1, \cdots, \eta_{r-1}$，这些近似解必须利用其它的方法来获得，比如说前面介绍的任何一种单步方法。下面我们将假设多步方法 (具体来说，是一个 r 步方法) 中的前 r 个近似值已经获得。

多数的多步方法都可以通过与原先的初值问题等价的积分方程来获得。事实上我们有，

$$y(t_{p+k}) - y(t_{p-q}) = \int_{t_{p-q}}^{t_{p+k}} dt f(t, y(t)) \, . \tag{9.23}$$

然后我们可以将上式中的被积函数利用一个多项式进行内插。例如，我们要求所用的多项式的阶数不超过 q，记为 $P_q(t)$；同时我们要求 $P_q(t_i) = f(t_i, y(t_i))$，$i = p, p-1, \cdots, p-q$，这样我们就可以获得如下的公式：

$$\eta_{p+k} = \eta_{p-j} + h \sum_{i=0}^{q} \beta_{qi} f(t_{p-i}, \eta_{p-i}) \, , \tag{9.24}$$

其中我们已经将严格的解 y 换成了它的相应的近似式 η。选择不同的整数 k，j，q，我们就可以获得不同的多步方法。[3] 例如，取 $k = 1$，$j = 0$ 并选取不同的整数 q，

[3] 如果选择 $k = 0$，$j = 1$，并且选择 $q = 0, 1, \cdots$ 我们得到的方法实际上是一个隐式方法 (implicit method)，即这时候新的近似式会出现在方程的两边。这个方法称为 Adams-Moulton 方法

$q=0,1,2,\cdots$，我们就得到了 Adams-Bashforth 方法 (Adams-Bashforth methods)，

$$\eta_{p+1}=\eta_p+h[\beta_{q0}f_p+\beta_{q1}f_{p-1}+\cdots+\beta_{qq}f_{p-q}]\,, \tag{9.25}$$

这里我们运用的简化的符号 $f_i\equiv f(t_i,\eta_i)$，其中的各个系数 β 为，

$$\beta_{qi}=\int_0^1\prod_{l=0,l\neq i}^{q}\frac{l+s}{l-i}ds\,,\quad i=0,1,\cdots,q\,. \tag{9.26}$$

显然这个 r 步方法中 $r=q+1$。

如果选取 $k=1$，$j=1$，我们就得到了所谓的 Nyström 方法 (Nyström method):

$$\eta_{p+1}=\eta_{p-1}+h[\beta_{q0}f_p+\beta_{q1}f_{p-1}+\cdots+\beta_{qq}f_{p-q}]\,, \tag{9.27}$$

其中的系数为 (注意积分限与前面的不同)，

$$\beta_{qi}=\int_{-1}^1\prod_{l=0,l\neq i}^{q}\frac{l+s}{l-i}ds\,,\quad i=0,1,\cdots,q\,. \tag{9.28}$$

如果令 $q=0$，我们就得到最简单的 中点法则 (midpoint rule):

$$\eta_{p+1}=\eta_{p-1}+2hf_p\,. \tag{9.29}$$

33 外推积分法的运用

¶另外一种十分有效的方法是运用类似于第 14节中的外推积分方法。这种方法基于如下的考虑：如果将我们希望得到的解看做关于步长 h 的一个函数，这个函数一般会存在一个关于 h 幂次的渐近展开式。如果我们知道了这一点，我们可以在几个不同的步长来计算同样的微分方程的解然后进行外推，外推到 $h\to 0$ 得到的结果就是我们所希望的。这种方法由 Bulirsch 和 Stoer 首先倡导，因此又称为 Bulirsch-Stoer 方法。一般来说，Bulirsch-Stoer 方法特别适用于方程的右边的函数都是十分光滑，并且我们对于解的精度有比较高的要求的时候。否则的话，前面介绍的一般的 Runge-Kutta 及其改进版本就已经足够了。

假定我们需要求解的初值问题由下式定义：

$$\dot{y}(t)=f(t,y)\,,y(t_0)=y_0\,. \tag{9.30}$$

我们试图将方程从 t_0 积分到 $\bar{t}=t_0+H$，其中 H 是一个不一定很小的步长。我们一般会将整个的 H 分为 n 份，即 $h=H/n$，其中 $n>0$ 是一个正的自然数。我们按照下式定义一个函数值 $S(\bar{t};h)$：

$$\begin{aligned}
\eta_0&=y_0\\
\eta_1&=\eta_0+hf(t_0,\eta_0)\,,t_1=t_0+h\,,\\
\eta_{j+1}&=\eta_{j-1}+2hf(t_j,\eta_j),t_{j+1}=t_j+h,\ ,j=1,2,\cdots,n-1\\
S(\bar{t};h)&=\tfrac{1}{2}[\eta_n+\eta_{n-1}+hf(t_n,\eta_n)]\,.
\end{aligned} \tag{9.31}$$

为此，我们首先选取一个比较大胆的步长 H 并且尝试将方程从某个 t_0 积分到 $\bar{t}=t_0+H$。这个步长一般来说过于大了。因此，我们会将它分为若干份：

$$n\in F=\{n_0,n_1,\cdots\}\ ,\quad n_0<n_1<n_2\cdots\ , \tag{9.32}$$

并且取 $h_j=H/n_j$ 进行积分。Bulirsch 和 Stoer 原先建议取：

$$n=2,4,6,8,12,16,24,\cdots\ ,\quad n_j=2n_{j-2},j\geq 3\ . \tag{9.33}$$

后来的实践证明取连续的偶数序列其实就挺好，即：$n_j=2(j+1),j\geq 0$。我们将对不同的 $h_j=H/n_j$ 进行积分。如果我们将相应的数值 $S(\bar{t};h_j)$ 的内插 k 阶多项式记为 $\tilde{T}_{ik}(h)$，

$$\tilde{T}_{ik}(h)=a_0+a_1h^2+\cdots+a_kh^{2k}\ , \tag{9.34}$$

那么我们有，

$$\tilde{T}_{ik}(h_j)=S(\bar{t};h_j)\ ,j=i,i-1,\cdots,i-k\ . \tag{9.35}$$

正如我们前面讨论过的，我们需要的是 $h=0$ 处的值。记

$$T_{ik}=\tilde{T}_{ik}(0)\ . \tag{9.36}$$

我们有

$$\lim_{i\to\infty}T_{ik}=y(\bar{t})\ ,k=0,1,\cdots \tag{9.37}$$

事实上 T_{ik} 与严格的解 $y(\bar{t})$ 之间的差别如下，

$$T_{ik}=y(\bar{t})+(-1)^kh_i^2h_{i-1}^2\cdots h_{i-k}^2[\tilde{u}_{k+1}(\bar{t})+\tilde{v}_{k+1}(\bar{t})]\ , \tag{9.38}$$

其中的 $\tilde{u}_{k+1}(\bar{t})$ 和 $\tilde{v}_{k+1}(\bar{t})$ 是两个特定的解析函数。

如果引进外推积分法中 Neville 记号，将近似解记为 $T_{k0}=S(\bar{t};h_k)$。我们可以获得类似的 Neville 三角形图：

$$\begin{array}{l|llll}
S(\bar{t};h_0) & T_{00} & & & \\
 & & T_{11} & & \\
S(\bar{t};h_1) & T_{10} & & T_{22} & \\
 & & T_{21} & & T_{33} \\
S(\bar{t};h_2) & T_{20} & & T_{32} & \vdots \\
 & & T_{31} & \vdots & \\
S(\bar{t};h_3) & T_{30} & \vdots & & \\
\vdots & \vdots & & &
\end{array} \tag{9.39}$$

其中的各个 $T_{k,j}$ 由下列递推关系给出 (参见第 14节中的公式 (4.29))：

$$T_{k,j+1}=T_{kj}+\frac{T_{kj}-T_{k-1,j}}{(n_k/n_{k-j})^2-1}\quad j=0,1,\cdots,k-1\ . \tag{9.40}$$

解的差别，例如在单分量的情形下就是 $|T_{kk} - T_{k,k-1}|$ 就体现了利用不同步长的误差。当这个差值足够小的时候，我们就可以停止迭代了。

¶ 一般来说，对于一个常微分方程的初值问题我们经常采取的方法主要包括两类，一类是比较普适的 Runge-Kutta 或者其各类推广；另一类就是这里介绍的 Bulirsch-Stoer 方法。后者特别适用于对于解的精度要求极高并且方程的右边具有良好的解析行为的情形。另外还有一类，即所谓的多步方法。多步方法其实往往不如前面两类方法那么实用，以目前的情况来看，它仅仅存活在比较特殊的情形中。[4]

34 硬的常微分方程组

¶ 前面提及的方法都属于显式方法 (explicit methods)。它们对于所谓硬的微分方程组 (stiff ODEs) 是无能为力的。这时我们需要运用所谓隐式方法 (implicit methods)。隐式方法意味着在每一步我们需要迭代求解 y_{n+1} 而不是直接计算。因此一般每一步需要更多的函数计算。具体的次数依赖于迭代的效率。

考虑一个简单的线性方程组，

$$\dot{y} = -A \cdot y \,, \tag{9.41}$$

其中 $y \in \mathbb{R}^N$ 是 N 个分量的矢量，$A \in \mathbb{R}^{N\times N}$ 则是一个常数矩阵。该方程具有标准的解析解法。我们将仅仅讨论矩阵 A 可以被对角化的情况，尽管总体的结论是类似的。如果 A 可以被对角化，这时存在一个相似变换 T 将 A 变为对角矩阵，

$$TCT^{-1} = \text{Diag}(\lambda_0, \cdots, \lambda_{n-1}) \,, \tag{9.42}$$

其中各个 λ_i 是矩阵 A 的本征值。于是令 $z = T^{-1} \cdot y$，我们可以写出方程的一般解，

$$y = T\text{Diag}(e^{-\lambda_0 t}, \cdots, e^{-\lambda_{n-1} t}) \cdot T^{-1} \cdot y_0 \,. \tag{9.43}$$

通常我们感兴趣的是所谓的稳定解，即假定

$$\text{Re}(\lambda_i) > 0 \quad i = 0, \cdots, n-1 \,. \tag{9.44}$$

如果我们利用数值的方法来求解方程 (9.41)，我们发现迭代的方程为，

$$y_{n+1} = (1 - Ah) \cdot y_n \quad y_n = (1 - Ah)^n \cdot y_0 \,. \tag{9.45}$$

如果初始的 y_0 可以用 A 的本征矢量 ξ_i 进行展开，

$$y_0 = \sum_{i=0}^{N-1} \alpha_i \xi_i \,, \tag{9.46}$$

[4] 一个典型例子是方程的右边计算起来特别复杂，这时候多步方法也许是有优势的。

那么我们有，

$$y_n = \sum_{i=0}^{N-1} \alpha_i (1 - h\lambda_i)^n \cdot \xi_i \ . \tag{9.47}$$

要获得稳定解的条件为，

$$|1 - h\lambda_i| < 1 \ . \quad i = 0, \cdots, N-1 \ . \tag{9.48}$$

这意味着如果所有的本征值都是正的实数的话，那么步长 h 的限制为，

$$h < \frac{2}{\lambda_{\max}} \ , \tag{9.49}$$

其中 $\lambda_{\max}$ 是 (都是正的) 本征值中最大的一个。所以，如果矩阵 A 恰好有一个特别大的本征值，那么步长就需要非常地小。

通常如果仅仅有一个方程，那么方程仅包含一个时间尺度。这时候问题其实并不是特别大的限制。因为这个不等式仅仅告诉你需要选取合适的单位而已。真正的困难发生在当有两个相差非常悬殊的本征值的时候。这意味着，如果我们按照其中一个选取积分的步长，对于另一个尺度而言就不可能是合适的，从而形成硬的微分方程组。

对付硬的微分方程组的一个办法是采用隐式求解。利如对于一步的欧拉法，我们将迭代修改为，

$$y_{n+1} = y_n + h\dot{y}_{n+1} \ . \tag{9.50}$$

这里的等式右边的 $\dot{y}$ 选择在新的点 y_{n+1} 而不是旧的点 y_n 进行计算。这时候的迭代给出，

$$y_{n+1} = (1 + Ah)^{-1} \cdot y_n \ . \tag{9.51}$$

即在每一步迭代中，我们需要求解一个矩阵的逆。相应的稳定性条件则修改为，

$$\frac{1}{|1 + h\lambda_i|} < 1 \quad i = 0, \cdots, N-1 \tag{9.52}$$

这对于全是正本征值的矩阵来说总是满足的，无论使用多么大的步长 h。但是其代价是必须在每一步迭代时候求解矩阵 $(1 + Ah)$ 的逆。

如果我们求解的方程不是常矩阵线性微分方程组而是具有形式，

$$\dot{y} = f(y) \ , \tag{9.53}$$

其中 f 是一 $\mathbb{R}^N$ 到自身的映射，相应的迭代方程可以写为，

$$y_{n+1} = y_n + hf(y_{n+1}) \ , \tag{9.54}$$

这一般会表现为一组非线性方程，需要迭代地求解。如果我们试图将其线性化得到，

$$y_{n+1} = y_n + h\left[f(y_n) + \left.\frac{\partial f}{\partial y}\right|_{y_n} \cdot (y_{n+1} - y_n) \right] \ , \tag{9.55}$$

其中的 $(\partial f/\partial y)$ 是函数 f 的 Jacobi 矩阵。于是我们得到，

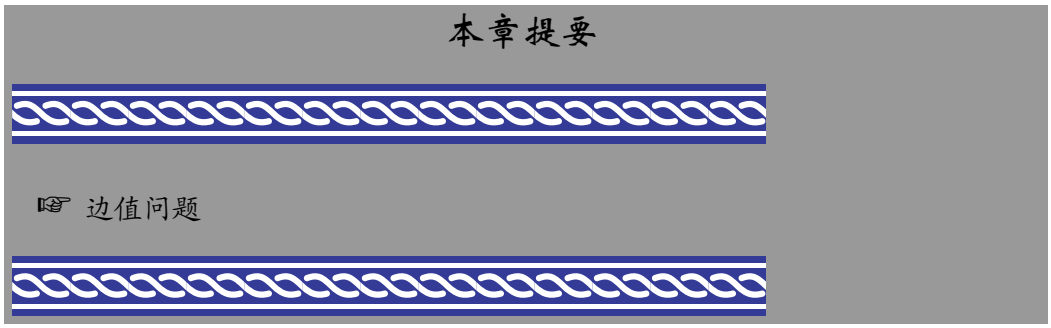
$$y_{n+1} = y_n + h\left[1 - h\frac{\partial f}{\partial y}\right]^{-1} \cdot f(y_n)\ . \tag{9.56}$$

利用线性化近似求解一般的非线性方程的方法一般称为半隐式方法 (semi-implicit method)。上面这个解法就称为半隐式欧拉方法 (semi-implicit Euler method)。

相关的阅读

这一章我们简单介绍了常微分方程的初值问题的解法

第十章　偏微分方程的数值解法



本章中我们讨论偏微分方程的初值或边值问题的解法。这当然是一个非常庞大的题目。它可以包括各种类型的偏微分方程：一阶的，二阶的或更高阶的；线性的或非线性的等等。我们这里仅仅选择多数物理学分支中遇到最为频繁的一个类型 –二阶线性偏微分方程的边值问题 –来展开我们的讨论。这类应用中，我们需要将连续的问题分立化。无论是运用简单的格点分立化还是更为复杂的有限元分立化，最后需要求解的数值问题都是一个大型稀疏矩阵的线性代数问题。

与前面讨论的线性代数问题不同的是，这类问题由于矩阵的庞大，因此不能直接利用前面的方法进行求解，而是需要借助于迭代的方法。

35　边值问题的描述

¶在物理问题中经常遇到偏微分方程的边值问题。一个典型的问题是 Poisson 方程的求解问题。考虑二维单位正方形上的方程，

$$\nabla^2\Phi(x) = \rho(x)\,, \quad 0 \le x_1 \le 1\,, 0 \le x_2 \le 1\,, \tag{10.1}$$

其中的函数 $\rho(x)$ 是一个已知的函数。如果 x 在边界上时，函数的值是已知的，这称为 Dirichlet 边条件，或者又称为第一类边条件。这类问题在静电边值问题中会经常会遇到。

它的求解可以将相应的区域进行分立化。一个简单的分立化方法是将每一维均匀地分为 N 份，引入格距 $a \equiv \frac{1}{N}$，这样相应的坐标为，

$$x_1 = i\cdot a,\ 0 \le i \le N\,, \quad x_2 = j\cdot a,\ 0 \le j \le N\,. \tag{10.2}$$

于是区域的内部点满足 $x=(i,j):0<i<N,0<j<N$，而四个边界则满足 $i=0,N$ 或者 $j=0,N$。标准的差分给出拉普拉斯算子的近似表达为，

$$\left(\nabla^2\right)_{x,y}=\sum_{i=1}^{2}\frac{1}{a^2}\left[\delta_{x+\hat{i},y}+\delta_{x-\hat{i},y}-2\delta_{x,y}\right]\ , \tag{10.3}$$

其中的 $x\pm\hat{i}$ 表示从格点 x 出发，沿着 i 的正/负方向走一格所对应的格点。于是对于内部的点，方程化为，

$$\sum_{i=1}^{2}\left[\delta_{x+\hat{i},y}+\delta_{x-\hat{i},y}-2\delta_{x,y}\right]\Phi_y=a^2\rho_x\ . \tag{10.4}$$

而对于边界上的点，$\Phi(x)$ 或者它的导数是一个已知的函数。将所有的已知的信息都移到等式的右边，我们发现上述方程化为一个线性方程，

$$A\cdot\Phi=b\ , \tag{10.5}$$

其中 Φ 代表了内部点处的所有未知的函数值，而矢量 b 则表示了所有已知的量的组合 (包括已知的电荷密度和边界的信息等等)。矩阵 A 则是一个大的稀疏矩阵，它的阶数就是未知的 Φ 的个数，即 $(N-1)^2\times(N-1)^2$ 阶的矩阵。

36 初值问题：扩散方程的数值解

¶偏微分方程同样也会出现初值问题。其中典型的例子是扩散方程以及量子力学中的 Schrödinger 方程。一个一维空间中的扩散方程为，

$$\frac{\partial u}{\partial t}=\frac{\partial}{\partial x}D\frac{\partial u}{\partial x}\ , \tag{10.6}$$

其中 D 称为扩散系数它可以是坐标的函数。我们感兴趣的是，给定 $t=0$ 时刻的 $u(t,x)$ 以及可能的边条件，如何求解随后时间的 $u(t,x)$。

将函数在时间和空间方向分别以 Δt 和 Δx 为间距分立化，我们就得到下面的分立化版本的扩散方程，

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}=D\left[\frac{u_{j+1}^n+u_{j-1}^n-2u_j^n}{(\Delta x)^2}\right]\ . \tag{10.7}$$

这个公式中我们将分立化版本的待求函数 $u(n\Delta t,j\Delta x)$ 写成了 u_j^n。其中的时间方向的分立化是准确到 Δt 一阶的，而空间方向的分立化是准确到 Δx 的二阶。这个分立化方法经常被称为 (forward time centered space)FTCS 模式。这是一个完全显式的形式。在迭代中，它的解的一般行为是，

$$u_j^n\sim\xi^n e^{ikj\Delta x}\ . \tag{10.8}$$

显然，要使得迭代稳定，我们必须要求放大因子 ξ 满足 $|\xi|<1$。扩散方程中的放大因子很容易发现为，

$$\xi=1-\frac{4D\Delta t}{(\Delta x)^2}\sin^2\left(\frac{k\Delta x}{2}\right)\ , \tag{10.9}$$

因此迭代的稳定性要求，

$$\frac{2D\Delta t}{(\Delta x)^2} \leq 1 \,. \tag{10.10}$$

物理上说，它要求扩散的距离模方 $2D\Delta t$ 要小于空间的格距模方。

仅仅从稳定性的角度来说，我们可以采用另外一种分立化的方法：

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = D\left[\frac{u_{j+1}^{n+1} + u_{j-1}^{n+1} - 2u_j^{n+1}}{(\Delta x)^2}\right] \,. \tag{10.11}$$

这个式子与前面的完全一样，唯一的区别是等式的右边我们采用了 u^{n+1} 而不是 u^n。这个式子看起来使得等式两边都含有 u^{n+1} 因此可以称为完全隐式 (fully implicit) 的算法。但是对于线性方程来说，我们完全可以重新组合后得到，

$$-\alpha u_{j-1}^{n+1} + (1+2\alpha)u_j^{n+1} - \alpha u_{j+1}^{n+1} = u_j^n \,, \tag{10.12}$$

其中的参数 α 由下式给出，

$$\alpha = \frac{D\Delta t}{(\Delta x)^2} \,. \tag{10.13}$$

我们看到，如果我们形式上取 $\Delta t \to \infty$，上式实际上会趋于拉普拉斯方程的解。因此我们预计这个算法实际上对于任意的 Δt 都是稳定的。的确，它的放大因子为，

$$\xi = \frac{1}{1 + 4\alpha \sin^2\left(\frac{k\Delta x}{2}\right)} \,, \tag{10.14}$$

它总是满足 $|\xi| < 1$。这个方法一般称为 Crank-Nicolson 方法。

¶另一类含时的扩散类方程是量子力学中的 Schrödinger 方程。

$$i\frac{\partial \psi}{\partial t} = -\frac{\partial^2 \psi}{\partial x^2} + V(x)\psi \,. \tag{10.15}$$

假定我们给定了初始的 $\psi(x, t=0)$ 以及已知的势函数 $V(x)$，我们需要求解方程给出 $\psi(x,t)$。利用前面的隐式分立化的办法，我们得到

$$i\left[\frac{\psi_j^{n+1} - \psi_j^n}{\Delta t}\right] = -\left[\frac{\psi_{j+1}^{n+1} + \psi_{j-1}^{n+1} - 2\psi_j^{n+1}}{(\Delta x)^2}\right] + V_j \psi_j^{n+1} \,. \tag{10.16}$$

这个分立化的放大因子为，

$$\xi = \frac{1}{1 + i\left[\frac{4\Delta t}{(\Delta x)^2}\sin^2\left(\frac{k\Delta x}{2}\right) + V_j \Delta t\right]} \tag{10.17}$$

这个因子对于任意的 Δt 都保持稳定。但是这个方法有一个问题。它的时间演化并不是幺正的。也就是说，无法保持概率守恒。正确的做法是运用下面的近似式，

$$e^{-iH\Delta t} \simeq \frac{1 - \frac{1}{2}iH\Delta t}{1 + \frac{1}{2}iH\Delta t} \,, \tag{10.18}$$

这个表达式的好处是对于任意的 Δt 都是幺正的，只要算符 H 是厄米的即可。因此分立化版本应当取为，

$$(1+\tfrac{1}{2}iH\Delta t)\psi_j^{n+1}=(1-\tfrac{1}{2}iH\Delta t)\psi_j^n\ . \tag{10.19}$$

然后将算符 H 中的偏微分算符用标准的方法分立化即可。这个方法不仅是稳定的，而且是幺正的，并且在时空都是二阶精确的。这样一来我们就不必随着时间的演化不停地重新归一化波函数了。

37 迭代解法

¶ 边值问题的解也可以视为初值问题解的稳定解。例如，如果我们希望求解椭圆形偏微分方程 $\mathcal{L}\cdot u=\rho$，其中 $\mathcal{L}$ 是一个椭圆形二阶算子，ρ 是已知函数，而 u 则是待求的方程的解。我们可以考察含时的方程，

$$\frac{\partial u}{\partial t}=\mathcal{L}\cdot u-\rho\ . \tag{10.20}$$

如果我们解这个含时的偏微分方程并构造一个序列使得它趋于上述方程的稳定解，那么这个序列就可以视为求解方程 $\mathcal{L}\cdot u=\rho$ 的一个迭代解法。以 Poisson 方程为例，我们求解

$$\frac{\partial u}{\partial t}=\nabla^2 u-\rho\ . \tag{10.21}$$

于是利用标准的 FTCS 分立化方法得到，

$$u_x^{n+1}=u_x^n+\frac{\Delta t}{\Delta^2}\sum_{i=1}^{2}\left(u_{x+\hat{i}}^n+u_{x-\hat{i}}^n-2u_x^n\right)-\rho_x\Delta t\ , \tag{10.22}$$

其中 Δt 是时间方向的间隔，Δ 则是二维空间方向的格距。对于二维空间来说，稳定性要求我们有

$$\frac{\Delta t}{\Delta^2}\le\frac{1}{4}\ . \tag{10.23}$$

如果我们就选取最大可能的 Δt，我们得到的算法为，

$$u_x^{n+1}=\frac{1}{4}\left(u_{x+\hat{1}}^n+u_{x-\hat{1}}^n+u_{x+\hat{2}}^n+u_{x-\hat{2}}^n\right)-\frac{\Delta^2}{4}\rho_x\ . \tag{10.24}$$

这就是著名的 Jacobi 算法。它是很多近代迭代算法的前身，尽管本身由于收敛速度太慢已经不会在正式的计算中被采用了。

¶ 下面我们讨论更为一般的迭代解法。我们已经看到，所有的偏微分方程的解法中都涉及求解一个大型非奇异的稀疏矩阵 A 的线性方程：

$$Au=b\ , \tag{10.25}$$

其中 $A \in \mathbb{C}^{N\times N}$ 为 $N \times N$ 的非奇异稀疏复矩阵，$b \in \mathbb{C}^N$ 为已知的复矢量，而 $u = A^{-1}b \in \mathbb{C}^N$ 为待求的解。为此我们寻找一个尽可能与 A 接近，但容易求解的非奇异矩阵 B 并写下等价的方程，

$$Bu + (A - B)u = b\,, \tag{10.26}$$

并且构造如下的迭代，

$$Bu^{i+1} + (A - B)u^i = b\,. \tag{10.27}$$

或者等价地写为，

$$u^{i+1} = (1 - B^{-1}A)u^i + B^{-1}b\,. \tag{10.28}$$

那么如果矩阵 B 可以比较容易地求解，而且 $(1 - B^{-1}A)\cdot u$ 可以容易地计算，我们就可以从一个初始的尝试解 u^0 出发去迭代地逼近严格解 $u = A^{-1}b$。这类迭代解法是 Wittmeyer 在 1936 年最先考虑的，因此称为 Wittmeyer 迭代。我们可以引入 $(N + 1) \times (N + 1)$ 的 Wittmeyer 矩阵，

$$W = \begin{bmatrix} 1 & 0 \\ B^{-1}b & (1 - B^{-1}A) \end{bmatrix}\,, \tag{10.29}$$

那么上面的 Wittmeyer 迭代可以等价地写为，

$$\begin{bmatrix} 1 \\ u^{i+1} \end{bmatrix} = W \cdot \begin{bmatrix} 1 \\ u^i \end{bmatrix}\,. \tag{10.30}$$

显然 Wittmeyer 矩阵总是有一个左本征矢 $(1, 0)$，其本征值为 $\lambda_0 \equiv 1$：

$$(1, 0) \cdot W = \lambda_0(1, 0) \tag{10.31}$$

因此很容易证明这个迭代收敛的要求是矩阵 $(1 - B^{-1}A)$ 的所有本征值 $\lambda_i, i = 1, \cdots, N$ 的模均小于 1：

$$|\lambda_i| < 1 \quad i = 1, 2, \cdots, N\,. \tag{10.32}$$

事实上可以证明，Wittmeyer 迭代收敛的充要条件是 $\rho(1 - B^{-1}A) < 1$，其中 $\rho(\cdot)$ 表示矩阵的谱半径 (见第 20节的第 20.1小节)。

前面提及的关于 Poisson 方程的 Jacobi 算法可以看成是一般的 Wittmeyer 迭代解法的一个特例。一般来说，在求解大型稀疏矩阵的线性方程 $Au = b$ 的过程中，我们可以将矩阵 A 做如下的标准分解：将它的对角部分记为 D，它的下三角和上三角部分分别记做 $-E$ 和 $-F$。即我们定义，

$$A = D - E - F\,. \tag{10.33}$$

我们还进一步假定 A 的所有对角元 $a_{ii} \neq 0$。这时候矩阵 D 的逆矩阵是非常容易求出的。事实上它仍然是一个对角矩阵并且对角元就是 $1/a_{ii}$。于是我们引入下列符号：

$$L = D^{-1}E\,, \quad U = D^{-1}F\,, \quad J = L + U\,, H = (1 - L)^{-1}U\,. \tag{10.34}$$

注意，矩阵 L 和 U 仍然分别具有下三角和上三角的性质。利用上面这些定义，我们可以发现所谓的 Jacobi 方法实际上相当于在 Wittmeyer 迭代中取

$$B = D\ , (1 - B^{-1}A) = J\ . \tag{10.35}$$

而需要求解的方程 (10.27) 则化为，

$$a_{jj}u_j^{i+1} + \sum_{k\neq j} a_{jk}u_k^i = b_j \quad j = 1, 2, \cdots, n\ , i = 0, 1, \cdots\ . \tag{10.36}$$

另外一种迭代的方法与上面类似，称为 Gauss-Seidel 迭代方法。它对应于取

$$B = D - E\ , (1 - B^{-1}A) = (1 - L)^{-1}U = H\ . \tag{10.37}$$

与其对应的迭代每步中要求解的方程 (10.27) 化为一下三角线性系统，

$$\sum_{k<j} a_{jk}u_k^{i+1} + a_{jj}u_j^{i+1} + \sum_{k>j} a_{jk}u_k^i = b_j\ , j = 1, 2, \cdots, n\ , i = 0, 1, \cdots, \ . \tag{10.38}$$

由于矩阵 A 是个稀疏矩阵，因此求解方程 (10.36) 和 (10.38) 所需要的计算量都是 $O(n)$ 而不是 $O(n^2)$。

¶ 下面简要讨论一下 Wittmeyer 迭代解法的收敛性。前面已经提及，这类迭代算法收敛的充要条件是 $\rho(1 - B^{-1}A) < 1$。事实上，我们可以选取特殊的模，

$$||A||_\infty = \max_i(\sum_k |a_{ik}|)\ , \tag{10.39}$$

这称为矩阵的最大模 (maximum norm)，对于最大模来说 Wittmeyer 迭代收敛的充分条件可以表述为 $||(1 - B^{-1}A)||_\infty < 1$。更为精细的分析需要考察第 i 次迭代 $x^{(i)}$ 与方程的严格解 x 之间的误差。为此我们定义

$$e_i \equiv x^{(i)} - x\ , \tag{10.40}$$

其中 x 是 $Ax = b$ 的解。可以证明，当迭代次数增加时，迭代误差相对应初始的误差基本上按照 $\rho(1 - B^{-1}A)$ 的幂次衰减：

$$\frac{||e_i||}{||e_0||} \simeq [\rho(1 - B^{-1}A)]^i\ , \tag{10.41}$$

其中 $||\cdot||$ 是任何一个矢量模。因此，$\rho(1 - B^{-1}A) < 1$，迭代才能收敛；同时 $\rho(1 - B^{-1}A) < 1$ 越小，迭代收敛得越快。

那么对于什么样的矩阵来说迭代能够产生小于 1 的 $\rho(1 - B^{-1}A) < 1$ 呢？对于前面介绍的 Jacobi 算法，我们有如下的定理：

定理 10.1 给定一个矩阵 $A \in \mathbb{C}^{n\times n}$，如果它满足强行对角主导条件，

$$|a_{ii}| > \sum_{k\neq i} |a_{ik}| \quad \forall i = 1, \cdots, n\ , \tag{10.42}$$

那么 Jacobi 迭代是收敛的，即 $\rho(J)<1$。类似的，如果矩阵满足所谓的强列对角主导条件，

$$|a_{kk}|>\sum_{i\neq k}|a_{ik}| \quad \forall k=1,\cdots,n\ , \tag{10.43}$$

Jacobi 迭代也是收敛的，即 $\rho(J)<1$。

¶对于多数物理应用而言，下列的所谓 Stein-Rosenberg 定理更具有实用性。

定理 10.2 给定一个矩阵 $A\in\mathbb{R}^{n\times n}$ 及其标准分解 (10.33), (10.34)，如果矩阵 $J=L+U$ 是非负矩阵，[1] 那么下列相互不兼容的四种情形中有且仅有一种会实现:

- $\rho(H)=\rho(J)=0$;
- $0<\rho(H)<\rho(J)<1$;
- $\rho(H)=\rho(J)=1$;
- $\rho(H)>\rho(J)>1$.

这意味着 Gauss-Seidel 算法和 Jacobi 算法，基本上要么同时收敛，要么同时发散。如果它们收敛的话，Gauss-Seidel 要更快一些。

这个定理之所以对于多数的物理中遇到的矩阵非常重要是因为如果我们将物理中经常遇到的二阶椭圆形偏微分算符分立化后，总是得到对角元为正，非对角元为负的稀疏矩阵。可以证明这类矩阵都满足本定理所描述的条件，即与其对应的矩阵 J 是非负矩阵。因此，这个定理说明我们物理学中常用的偏微分方程中出现的矩阵，如果要求解其边值问题话，利用 Gauss-Seidel 一般总是收敛的而且快于 Jacobi 算法。

38 驰豫算法

¶迭代算法中经常用到的一类算法称为驰豫算法 (relaxation algorithm)。本节中我们就简要介绍一下这类算法。前一节的讨论告诉我们，广义的 Wittmeyer 迭代需要寻找一个矩阵 B，它必须足够接近于 A 同时应当尽量使得 $\rho(1-B^{-1}A)$ 足够小。在驰豫算法中，我们考虑下面的矩阵 $B(\omega)$，

$$B(\omega)=\frac{1}{\omega}D(1-\omega L)\ , \tag{10.44}$$

其中 (关于矩阵 D，L 等的定义) 我们沿用了上一节关于矩阵 A 分解的记号，见公式 (10.33) 和 (10.34)。这里的 ω 是一个可调参数。我们后面会调节它使得 $\rho(1-B^{-1}(\omega)A)$ 尽可能地小。如果 $\omega=1$，则回到前面的 Gauss-Seidel 算法。为了记号上的方便，我们定义

$$H(\omega)=1-B(\omega)^{-1}A\ . \tag{10.45}$$

[1] 一个实矩阵被称为 非负矩阵 (nonnegative matrix) 如果它的所有矩阵元都是非负实数。注意，非负矩阵与正定或半正定矩阵并不是一个概念！

这样一来迭代收敛的充要条件就是 $\rho(H(\omega)) < 1$。

为了讨论何种矩阵在做了标准分解 (10.33), (10.34) 和 (10.45) 之后满足 $\rho(H(\omega)) < 1$，R. Varga 引进了所谓 自洽排序矩阵 (consistently ordered matrix) 的概念。为了不至于涉及过多的数学细节，我们这里仅仅指出：所有通过有限差分所得到的物理学中常用的算符基本上都对应于自洽排序的矩阵。[2] 关于 $\rho(H(\omega)$，我们有如下的性质：

定理 10.3 对于任意的矩阵 A，我们按照公式 (10.33), (10.34) 和 (10.45) 进行分解，那么下列性质成立：

1. $\rho(H(\omega)) \geq |\omega - 1|$，因此迭代收敛的必要条件是 $\omega \in (0, 2)$;

2. 对于正定矩阵 A，我们有 $\rho(H(\omega)) < 1$，$\forall \omega \in (0, 2)$；因此，对于所有的正定矩阵而言，Gauss-Seidel 算法 (对应于 $\omega = 1$) 一定收敛;

3. 对于矩阵 A 而言，如果相应的 L 和 U 是非负矩阵，那么只有过驰豫能够给出比 Gauss-Seidel 更快的收敛速度;

4. 如果矩阵 A 不可约并且相应的矩阵 J 为非负矩阵，[3] 进一步假定 Jacobi 算法收敛，即 $\rho(J) < 1$，那么 $\rho(H(\omega))$ 在区间 $(0, \tilde{\omega}]$ 上是关于 ω 的减函数，其中 $\tilde{\omega} \geq 1$;

5. 对于所谓自洽排序矩阵 A 而言，相应的 Gauss-Seidel 算法和 Jacobi 算法的谱半径满足：$\rho(H) = [\rho(J)]^2$；因此 (如果如果收敛) 根据误差估计 (10.41)，达到同样的精度来说，Gauss-Seidel 只需要 Jacobi 一半的迭代次数;

6. 对于所谓自洽排序矩阵 (consistently ordered matrix) 而言，如果矩阵 J 的本征值都是实数并且 Jacobi 方法是收敛的即：$\rho(J) = \rho_J < 1$，那么驰豫算法相应的最佳驰豫参数为，

$$\omega_b = \frac{2}{1 - \sqrt{1 - \rho_J^2}} \,. \tag{10.46}$$

因此最佳的驰豫参数对应于过驰豫 (overrelaxation)，即 $1 < \omega < 2$;

7. 对应于上述最佳的驰豫参数，过驰豫算法当然也是收敛的并且其谱半径满足，

$$\rho(H(\omega_b)) = \omega_b - 1 = \left(\frac{\rho_J}{1 + \sqrt{1 - \rho_J^2}} \right)^2 \,. \tag{10.47}$$

[2] 关于自洽排序矩阵的确切数学定义参考 [2] 的第 §8.3 节或 [4] 的 §4.1 节。

[3] 如果不存在一个置换矩阵 P 将矩阵 A 化为上三角块矩阵的形式 $P^T A P = \left[\begin{smallmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{smallmatrix} \right]$，我们就称矩阵 A 是不可约的；否则就称之为可约的。

39 例子

¶作为一个例子，考虑正方晶格上二维的 Poisson 方程的求解问题。我们将分别比较上节讨论过的几种算法的有效性。我们考虑的问题是，

$$-\nabla^2 u(x,y) = f(x,y)\ , \tag{10.48}$$

求解的区域为二维平面上第一象限的单位正方形，即 $\Omega = \{(x,y) : 0 \le x \le 1; 0 \le y \le 1\}$。在边界上 $u(x,y)$ 满足 Dirichlet 边条件 (即恒等于零)。现在我们将其分为 $(N+1)$ 份，为此定义格距

$$h = \frac{1}{N+1}\ , \quad N \ge 1\ . \tag{10.49}$$

我们记 $u(x,y) = u_{ij}$，其中 $x = ih, y = jh$。相应的，我们会将 $f(x,y)$ 记为 f_{ij}。于是我们可以将拉普拉斯算子的作用近似为，

$$-\nabla^2 u \simeq \frac{1}{h^2}\left[4u_{ij} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}\right] \equiv \frac{1}{h^2}(-\hat{\nabla})^2\ . \tag{10.50}$$

于是待求的方程为，

$$(-\hat{\nabla})^2 \cdot u = b\ , \tag{10.51}$$

其中 $b = h^2 f_{ij}$。如果我们将矩阵 $(-\hat{\nabla})^2$ 视为我们迭代解法中的矩阵 A，即 $A = (-\hat{\nabla})^2$。容易验证它满足前面的所有条件 (自洽排序等)，并且相应的 Jacobi 迭代的矩阵为，

$$J = I - \frac{1}{4}A\ , \tag{10.52}$$

其中 I 表示单位矩阵。矩阵 J 的本征对恰好可以完全求出来，其本征函数 $z_{ij}^{(l,k)}$ 以及相应的本征值 $\mu^{(l,k)}$ 由一对整数 (l,k) 来刻画：

$$z_{ij}^{(l,k)} = \sin\left[\frac{l\pi i}{N+1}\right]\sin\left[\frac{k\pi j}{N+1}\right]\ , \quad 1 \le l,k \le N\ . \tag{10.53}$$

显然这个本征函数满足 Dirichlet 边条件。与其相应的本征值为，

$$\mu^{(l,k)} = \frac{1}{2}\left[\cos\frac{l\pi}{N+1} + \cos\frac{k\pi}{N+1}\right]\ , \quad 1 \le l,k \le N\ . \tag{10.54}$$

因此我们可以轻易求出矩阵 J 的谱半径，

$$\rho(J) = \max_{k,l}(|\mu^{(l,k)}|) = \cos\frac{\pi}{N+1}\ . \tag{10.55}$$

我们发现这个数是一个小于 1 的正数，因此 Jacobi 算法是收敛的。

按照前面的讨论也很容易获得 Gauss-Seidel 算法的谱半径，

$$\rho(H) = \rho(J)^2 = \cos^2\frac{\pi}{N+1}\ . \tag{10.56}$$

因此 Gauss-Seidel 算法也是收敛的并且比 Jacobi 要快一倍。

我们发现当 $N \to \infty$ 时，$\rho(J)$ 和 $\rho(H)$ 都会趋于 1。因此当 N 很大时，两种方法的收敛速度都会变慢。相应的优化 SOR 方法给出的谱半径则为，

$$\rho(H(\omega_b)) = \frac{\cos^2 \frac{\pi}{N+1}}{\left(1+\sin\frac{\pi}{N+1}\right)^2} . \tag{10.57}$$

显然优化的 SOR 仍然是收敛的并且当 N 很大时收敛的速度也会变慢。为了比较这三种算法在大的 N 时的速度，我们定义一个参数 κ 如下，

$$\kappa = \frac{\ln(\rho(H(\omega_b)))}{\ln(\rho(J))} , \tag{10.58}$$

它给出了优化 SOR 相对于 Jacobi 算法的快慢 (对于 Gauss-Seidel 来说，$\kappa = 2$) 之比。对于足够大的 N 来说，带入上述公式我们发现这个比值依赖于 N，结果为，

$$\kappa = \kappa(N) \simeq \frac{4}{\pi}(N+1) . \tag{10.59}$$

因此，越是大的 N，优化 SOR 相对于 Jacobi 和 Gauss-Seidel 越是具有优势。事实上，如果要将误差减少一个量级的话，Jacobi，Gauss-Seidel 和优化 SOR 这三种算法分别需要的迭代次数为，

$$\begin{aligned} R_J &= -\frac{\ln 10}{\ln \rho(J)} \simeq 0.467(N+1)^2 \\ R_{GS} &= \frac{1}{2}R_J \simeq 0.234(N+1)^2 \\ R_{SOR} &= -\frac{\ln 10}{\ln \rho(H(\omega_b))} \simeq 0.367(N+1) \end{aligned} \tag{10.60}$$

显然，这个结果说明，如果能够利用优化的 SOR 方法，我们显然应当应用它。

需要指出的一点是，上述关于三种算法迭代次数的比较式 (10.60) 只是对于多次迭代之后的各算法的渐进值之间的比较。这并不意味着最佳的 ω_b 的值在迭代的一开始也是合适的。一般来说，任何的迭代算法都大致与误差减少一个量级的迭代次数等量级的迭代之后，才能真正过渡到其相应的渐进估计值。也就是说，对 SOR 算法，至少需要 $O(N)$ 的迭代次数之后其渐进值才能实现。因此如果迭代一开始就令 $\omega = \omega_b$，有可能反而使得迭代在开始的阶段很慢。一个有效的应对这个问题的办法是加入所谓的 Chebyshef 加速 (Chebyshef acceleration) 过程。我们在每次迭代的中间加入一系列的半整数的迭代，其间我们依次改变参数 ω 的数值。最初是 Gauss-Seidel，然后逐渐过渡到最佳的 SOR：

$$\begin{aligned} \omega^{(0)} &= 1 \\ \omega^{(1/2)} &= 1/(1-\rho(J)^2/2) \\ \omega^{(n+1/2)} &= 1/(1-\rho(J)^2\omega^{(n)}/4) , \quad n = 1/2, 1, 3/2, \cdots , \\ \omega^{(\infty)} &\to \omega_b \end{aligned} \tag{10.61}$$

Chebyshef 加速的好处是，可以证明它能够保证误差在每一步迭代中一定是减少的，并且在足够多迭代之后后，它可以达到最佳的 SOR 算法的效果。因此它往往能够实现最少的总的迭代次数，从而是一个很难拒绝的诱惑。

40 非自伴矩阵的迭代算法

¶ 前面 (参见第 23.1小节) 我们讨论过的最陡下降法和共轭梯度算法也是求解偏微分方程的边值问题中经常用到的。共轭梯度法主要用于处理正定对称矩阵的问题。另外，我们还会遇到矩阵不是对称矩阵的情形，这时候可以利用所谓的推广的最小剩余法 (generalized minimal residual, GMRES) [4] 或者双共轭梯度算法 (Bi-conjugate gradient, BiCG)。本节中我们对这两类算法做一个简单的介绍。

¶ GMRES 算法起源于 最小剩余算法 (Minimal Residual, MR 或 MINRES)。[5] 所以我们首先从最原始的最小剩余法开始。它原则上既可以处理对称矩阵也可以处理非对称矩阵。考虑 $A \in \mathbb{C}^{n\times n}$ 为一个任意的 (一般来说不是厄米的) 非奇异的复矩阵。对于稀疏的矩阵 A 以及已知的复矢量 $b \in \mathbb{C}^n$，如果我们希望迭代地求解线性方程，

$$Ax = b\,. \tag{10.62}$$

迭代可以这样进行：随意选取一个任意的初始值 (比如 $x_0 = 0$)，计算剩余矢量 $r_0 = b - Ax = b$。然后我们进行如下的迭代 $i = 1, 2, \cdots$：

$$\begin{aligned} x_{i+1} &= x_i + \alpha_i r_i\,, \\ r_{i+1} &= r_i - \alpha A r_i\,, \end{aligned} \tag{10.63}$$

其中我们选择 α 使得在每一步的 r_{i+1} 的欧氏模方最小。很容易发现这给出的参数 α_i 的选择是，

$$\alpha_i = \frac{(Ar_i)^\dagger r_i}{(Ar_i)^\dagger (Ar_i)}\,. \tag{10.64}$$

因此，原始的最小剩余算法相当于是在当前剩余矢量 r_i 的方向上求下一步剩余矢量的方向极小值。显然，我们可以将寻找极小值的方向进一步扩充，例如在所有剩余矢量的 Krylov 子空间 $K_m(A; r_0)$ 中去寻找极小值。这就是产生了所谓的推广的最小剩余法。其代价是我们必须在迭代的过程中储存更多的矢量：$\{r, Ar, \cdots, A^{m-1}r\}$，其中 m 是一个可以选择的正整数。由于处理的稀疏矩阵的维度一般都非常巨大，所以我们一般会选取 $m \ll n$。

具体实施的过程中要面临的一个问题是，虽然各个矢量 $\{r, Ar, \cdots, A^{m-1}r\}$ 一般是线性无关的，但是它们并不是正交归一的基。这对于计算并不是非常方便。比较方便的做法

[4] Y. Saad and M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (1986), pp. 856-869.

[5] C.C. Paige and M.A. Saunders, Solution of sparse indefinite systems of linear equations, SIAM J. Numer. Anal., 12 (1975), pp. 617-624.

是从它们出发构造出一组正交归一的基矢。这个过程一般通过所谓的 Arnoldi 迭代 来实现。如果相应的矩阵 A 是一个厄米矩阵，那么 Arnoldi 迭代就等同于我们在第23.3小节中讨论的 Lanczos 迭代 。

如前所述，Arnoldi 迭代也是试图构建一组正交归一的一系列矢量：$q_1, q_2, \cdots$，它们称为 Arnoldi 矢量，对于任意 m 个 Arnoldi 矢量 $(q_1, \cdots, q_m)$ 来说，它们都张成了 Krylov 子空间 $K_m(A; r)$。一般我们选取 $m \ll n$。

Algorithm 14 Arnoldi 迭代

Require: $A \in \mathbb{C}^{n\times n}$；令 $q_1 = r_0/||r_0||_2 \in \mathbb{C}^n$ 且 $||q_1||_2 = 1$；相应的 m 阶 Krylov 子空间为 $K_m(A; r_0)$。

1: **for** $k = 2, \cdots, m$ **do**

2:

$$q_k \leftarrow A q_{k-1} \ , \tag{10.65}$$

3: **for** $j = 1, \cdots, (k-1)$ **do**

4:

$$\begin{aligned} h_{j,k-1} &\leftarrow q_j^\dagger q_k \ , \\ q_k &\leftarrow q_k - h_{j,k-1} q_j \ , \end{aligned} \tag{10.66}$$

5: **end for**

6:

$$\begin{aligned} h_{k,k+1} &\leftarrow ||q_k||_2 \ , \\ q_k &\leftarrow \frac{q_k}{h_{k,k-1}} \ . \end{aligned} \tag{10.67}$$

7: **end for**

内部的循环将每一个矢量 q_k 中沿着各个之前的矢量 $q_1, q_2, \cdots, q_{k-1}$ 的分量都投影掉了，这保证每一个 q_k 都与前面的各个 $q_{j<k}$ 的矢量正交。最后我们再将它归一化。经过迭代到 $k = m$，我们获得了一组正交归一的矢量基：$(q_1, q_2, \cdots, q_m)$，它们张成了 Krylov 子空间 $K_m(A; r_0)$。因此，Arnoldi 迭代的产物是这一系列正交归一的矢量 q_i 以及各个系数 $h_{j,k}$。显然，这些系数构成一个 $m \times m$ 的上 Hessenberg 矩阵 H_m：$(H_m)_{jk} = h_{j,k}$。如果原先的矩阵是厄米矩阵 (或者在实矩阵情形下，对称矩阵)，Hessenberg 矩阵就变为是三对角矩阵。事实上，如果我们将各个归一的矢量 $(q_1, \cdots, q_m)$ 排成一个 $n \times m$ 的矩阵 Q_m，我们有，

$$H_m = Q_m^\dagger A Q_m \ , \tag{10.68}$$

因此矩阵 H_m 可以视为原先的更大的矩阵 $A \in \mathbb{C}^{n\times n}$ 的在子空间 $K_m(A; r_0)$ 上的一个等效投影。如果我们已经获得了各个系数的矩阵 H_m，那么我们可以定义一个新的 $(m+1) \times m$

的上 Hessenberg 矩阵 $\tilde{H}_m$，它是在原先的矩阵 H_m 的最下方再加上一行：

$$\tilde{H}_m = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \cdots & h_{1,m} \\ h_{2,1} & h_{2,2} & h_{2,3} & \cdots & h_{2,m} \\ 0 & h_{3,2} & h_{3,3} & \cdots & h_{3,m} \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & 0 & h_{m,m-1} & h_{m,m} \\ 0 & \cdots & \cdots & 0 & h_{m+1,m} \end{bmatrix} . \tag{10.69}$$

这个矩阵与下一阶的迭代的关系为，

$$AQ_m = Q_{m+1}\tilde{H}_m . \tag{10.70}$$

利用上述的 Arnoldi 迭代，我们可以构建所谓的推广的最小剩余算法。

GMRES 试图在 Krylov 子空间 $K_m(A; r_0)$ 中寻找方程 $Ax = b$ 的近似解。其做法是要求其第 m 次迭代的解 x_m 能够使得剩余矢量 $r_n = b - Ax_n$ 的模最小。为此我们令，

$$x_m = Q_m y_m , \tag{10.71}$$

其中 Q_m 是前面 Arnoldi 迭代获得的 $n \times m$ 矩阵，$y_m \in \mathbb{C}^m$ 相当于矢量 x_m 在 $K_m(A; r_0)$ 中的展开系数。于是求 r_m 的极小值意味着，

$$||Ax_m - b|| = ||AQ_m y_m - b|| = ||Q_{m+1}\tilde{H}_m y_m - b|| = ||\tilde{H}_m y_m - Q_{m+1}^{\dagger} b|| = ||\tilde{H}_m y_m - \beta e_1|| , \tag{10.72}$$

其中 $e_1 = (1, 0, \cdots, 0)^T \in \mathbb{C}^m$ 是 $\mathbb{C}^m$ 中第一个基矢，参数 $\beta = ||r_0|| = ||b - Ax_0||$ 是初始的剩余的模。上面的推导的最后一步我们利用了事实：第一个矢量总是选择归一化的初始剩余的，即：$r_0 = q_1 \cdot ||r_0||$。因此我们必定有：

$$r_0 = (||r_0||)Q_{m+1}e_1 . \tag{10.73}$$

因此，在第 m 次迭代中，GMRES 首先进行 Arnoldi 迭代构造矩阵 Q_m，H_m，$\tilde{H}_m$ 和 Q_{m+1}，随后需要求解一个 $m+1$ 维的最小平方和问题：

$$||r_m|| = \min_{y_m} ||\tilde{H}_m y_m - \beta e_1|| . \quad x_m = Q_m y_m . \tag{10.74}$$

这样获得的 x_m 就是线性方程 $Ax = b$ 在 Krylov 子空间 $K_m(A; r_0)$ 中的近似解。一般来说，如果到某一个 m 我们认为其剩余的模 $||r_m||$ 已经足够小，我们就停止迭代并返回 x_m 作为近似解。

GMRES 具有良好的稳定性。对于多数的矩阵都可以相当好地工作。它的一个问题是会耗费比较多的内存，这发生在 m 很大的时候。如果需要很大的 m 才能收敛，那么储存这些矢量 –或者等价的说矩阵 Q_m–需要相当可观的内存。虽然原始的矩阵 A 是稀疏的，但是矩阵 Q_m 并不是。因此如果 m 很大的话，这个还是会超出我们内存的承受能力的。

原则上讲，如果没有舍入误差，GMRES 最多迭代 n 次一定会收敛到严格解。当然，我们一般都是希望它能够在比较小的 m 的时候已经足够接近严格解了。

¶另外一种迭代方法是试图将 GMRES 的长迭代变为类似共轭梯度法 (CG) 那样的短迭代。对于非厄米的矩阵来说，这类算法的一个典型是所谓的双共轭梯度算法 (Bi-conjugate gradient, Bi-CG)。但是这个算法的稳定性不太好。为此人们后来又提出了稳定双共轭梯度算法 (Bi-conjugate gradient stabilized, Bi-CGStab)。有兴趣的读者可以参见相关的文献，我们这里就不再深入了。

相关的阅读

这一章我们简单介绍了偏微分方程的边值问题的数值解法。

参考书

[1] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery *Numerical Recipes: The art of scientific computing*, 3rd Ed., Cambridge University Press, 2007.

[2] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, 2nd Ed., (Texts in Applied Mathematics, TAM 12), Springer-Verlag, 1998.

[3] D. Arnold, *A Concise Introduction to Numerical Analysis*, lecture notes, 2001.

[4] R. S. Varga, *Matrix Iterative Analysis*, 2nd ed., Springer-Verlag, 2009.

[5] A. Quarteroni, R. Sacco and F. Saleri, *Numerical Mathematics*, (Texts in Applied Mathematics, TAM 37), Springer-Verlag, 2000.

[6] R. L. Burden and J. D. Faires, *Numerical Analysis*, 9th ed., Brooks/Cole, Cengage Learning, 2011.

[7] L. R. Scott, *Numerical Analysis*, Princeton University Press, 2011.

索 引