计算物理讲义

冯旭

1　数值计算的基础

2　线性方程组的直接解法

3　内插与函数的计算

4　数值积分

5　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：$\{x^* \in E | f(x^*) = 0\}$，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 $E = F = R$。稍微推广一些的情形是 $E = F = R^n$，这时我们要求解的是一系列联立的非线性方程。

一般来说，求根问题，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，显然它必须使得待求的根满足：$x^* = \phi(x^*)$，即待求的根是迭代函数的不动点。

其实方程求根的问题也不见得就很简单。当一个问题拿到手以后，我们首先判断问题的敏感性。要分析敏感性，首先应假设问题中的数据如何扰动，一种易于分析的情况是将非线性方程写成

$$f(x) = y \tag{1}$$

的形式，然后讨论 y 在 0 值附近的扰动造成的问题敏感性。这个时候，求根问题变成了函数求值问题的反问题。如果函数值 $f(x)$ 对输入参数 x 在零点附近的变化很不敏感，则求根问题将很敏感；反之，若函数值对参数值很敏感，则求根问题不敏感。下面分析 y 发生扰动 Δy 引起的方程的根的扰动 Δx。由于当 $x = x^*$ 时，$y = 0$，我们使用绝对 (而不是相对) 条件数

$$\text{cond} = \left| \frac{\Delta x}{\Delta y} \right| \approx \frac{1}{|f'(x^*)|} \tag{2}$$

条件数的大小反映方程求根问题的敏感程度。若 $|f'(x^*)|$ 很小，则问题很敏感，是一个病态问题；反之，若 $|f'(x^*)|$ 很大，则问题不敏感。一种特殊情况是 $f'(x^*) = 0$，即 x^* 是重根，这个时候问题就会变得非常敏感，小扰动有可能会造成解的很大的误差。对于敏感的非线性方程求根问题，$f(x) \approx 0$ 并不意味着 x 很接近 x^*，所以这就涉及到迭代过程中的判停问题，就是我们什么时候让程序停下来。我们后面会讨论到这个问题。

5.1 对分法求根

首先考虑一个一维实函数 $f : R \to R$ 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 $x^* \in [a, b]$ 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 x^* 之间的差别: $\epsilon_i = x_i - x^*$，那么对于对分法我们一定有:

$$|x_{i+1} - x^*| \approx |x_i - x^*|/2 \tag{3}$$

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

$$\epsilon_{i+1} \sim c(\epsilon_i)^m \tag{4}$$

我们就称该算法是 m 幂次收敛的。其中 c 被称为收敛常数，m 被称为收敛阶数。c 一般是小于 1 的正数。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。

有一点要说明的是，由于我们采用的是浮点数系统，二分法运行结果的准确度不可能随迭代过程一直提高。二分法使得区间越来越小，当区间的两个端点已经是相邻的两个浮点数时，再执行二分法，区间也不会再改变。

5.2 不动点迭代法

由于二分法的计算效率不够高，我们下面介绍不动点迭代法。基本原理是，通过某种等价变换，将非线性方程改写为

$$f(x) = 0 \quad \Rightarrow \quad x = \phi(x) \tag{5}$$

给定初始值 x_0 后，可构造迭代计算公式

$$x_{k+1} = \phi(x_k), \quad k = 0, 1, \cdots \tag{6}$$

如果序列 $\{x_k\}$ 收敛，那么极限就是 $x = \phi(x)$，也就是 $f(x) = 0$ 的解。

不动点迭代法的算法给出如下

$$\begin{aligned}
&k := 0 \\
&\text{While } |f(x_k)| > \epsilon_1 \text{ or } |x_k - x_{k-1}| > \epsilon_2 \text{ do} \\
&\qquad x_{k+1} := \phi(x_k) \\
&\qquad k := k + 1; \\
&\text{End} \\
&x := x_k
\end{aligned} \tag{7}$$

其中 ϵ_1 和 ϵ_2 是用于判断迭代是否应该停止的两个阈值。关于这两个值如何选取，我们后面再讨论。

关于不动点迭代法，有个很重要的性质就是迭代的收敛性。我们首先讨论全局收敛的充分条件。有下面一个定理，给出了一个函数存在唯一不动点的充分条件。定理：设 $\phi(x) \in C[a,b]$，如果满足如下两个条件

- 对任意 $x \in [a,b]$，有 $a \le \phi(x) \le b$
- 存在正常数 $L \in (0,1)$，使对任意 $x_1, x_2 \in [a,b]$

$$|\phi(x_1) - \phi(x_2)| \le L|x_1 - x_2| \tag{8}$$

则 $\phi(x)$ 在 $[a,b]$ 上存在不动点，且不动点是唯一的。

我们先来理解一下定理中两个条件的含义。首先，采用不动点迭代法的计算公式为 $x_{k+1} = \phi(x_k)$, $k = 0, 1, 2, \cdots$，因此要使后续迭代计算合法，必须要求 $\phi(x)$ 的值在函数定义域内，条件 (1) 保证了这一点。其次，条件 (2) 表明，$\phi(x)$ 曲线上任意两点连线斜率的绝对值不会超过 L，当两点非常靠近时，它就是导数。因此，$\phi(x)$ 曲线上任意点的切线斜率的绝对值都小于 1。这个条件也称为 $L < 1$ 的李普希兹 (Lipschitz) 条件，L 为李普希兹系数。

我们首先来看不动点的存在性。如果 $\phi(a) = a$ 或者 $\phi(b) = b$，那么 a 或者 b 就是不动点。如果 $\phi(a) \ne a$ 且 $\phi(b) \ne b$, 那么我们有 $\phi(a) > a$, $\phi(b) < b$。令 $f(x) = \phi(x) - x$。我们有 $f(a) > 0$, $f(b) < 0$, 那么 $[a,b]$ 区间内必然存在着零点 x^*，使得 $f(x^*) = 0 \Rightarrow \phi(x^*) = x^*$。

我们下面来看不动点的唯一性。假设存在两个不动点 $x_1^* \ne x_2^*$。那么我们有

$$|x_1^* - x_2^*| = |\phi(x_1^*) - \phi(x_2^*)| \le L|x_1^* - x_2^*| < |x_1^* - x_2^*| \tag{9}$$

这显然产生矛盾，所以不动点是唯一的。

有了函数存在唯一不动点的充分条件，我们下面来看不动点迭代法收敛的充分条件。假设 $\phi(x)$ 函数已经满足了之前提到的两个条件，那么对于任意初值 $x_0 \in [a,b]$，由不动点迭代法得到的序列 $\{x_k\}$ 必然收敛到 $\phi(x)$ 的不动点 x^*，并且有误差估计

$$|x_k - x^*| \le \frac{L^k}{1-L}|x_1 - x_0| \tag{10}$$

这个其实很好理解。我们有

$$|x_k - x^*| = |\phi(x_{k-1}) - \phi(x^*)| \le L|x_{k-1} - x^*| \le \cdots \le L^k|x_0 - x^*| \tag{11}$$

当 $k \to \infty$ 时，我们有 $x_k = x^*$。另外，根据

$$|x_0 - x^*| = \lim_{k\to\infty} |x_0 - x^k| \le \lim_{k\to\infty} |x_0 - x_1| + |x_1 - x_2| + \cdots = \frac{1}{1-L}|x_0 - x_1|. \tag{12}$$

我们可以看到，不动点的收敛性其实并不依赖于初值 x_0 的选取，因此称为全局收敛。为了方便起见，我们也可以将前面定理中的条件 (2) 替换为：对任意 $x \in [a,b]$，有 $|\phi'(x)| \le L < 1$。

全局收敛性要求初始值 x_0 为定义域内任意值时，不动点迭代法都收敛，这常常是很难达到要求的。我们下面给出局部收敛性的概念。假设函数 $\phi(x)$ 存在不动点 x^*，若存在 x^* 的某个领域 D : $[x^* - \delta, x^* + \delta]$，对于初值 $x_0 \in D$，迭代法 $x_{k+1} = \phi(x_k)$ 产生的解序列 $\{x_k\}$ 收敛到 x^*，则称迭代

法局部收敛。这里，我们不难给出迭代法局部收敛的充分性条件：假设 x^* 是函数 $\phi(x)$ 的不动点，若 $\phi'(x)$ 在 x^* 的某个领域上连续，且 $|\phi'(x)| < 1$，则不动点迭代法 $x_{k+1} = \phi(x_k)$ 局部收敛。我们可以看出来，局部收敛的条件比较宽松，它只需要考察函数 $\phi(x)$ 在 x^* 这一点上是否满足要求。因此，不动点迭代法比较容易具有局部收敛性，而且对局部收敛性的判断也相对简单。这里我们说明一点，我们知道李普希兹系数 L 越小迭代收敛速度越快。对应于局部收敛，事实上 $|\phi'(x)|$ 越小，迭代收敛的速度就越快。

我们之前提到说二分法的收敛阶数是 1 阶收敛，那么不动点迭代法是几阶收敛呢？这和函数 $\phi(x)$ 的性质有关。若在所求根 x^* 的邻域上函数 $\phi(x)$ 的 p 阶导数连续，$p \geq 2$，则该迭代法在 x^* 的邻域上 p 阶收敛的充分必要条件是：$\phi'(x^*) = \phi''(x^*) = \cdots = \phi^{(p-1)}(x^*) = 0$，且 $\phi^{(p)}(x^*) \neq 0$。

这个事情其实很好理解。令 $\epsilon_k = x_k - x^*$。考察其收敛性

$$\begin{aligned} \epsilon_{k+1} &= x_{k+1} - x^* = \phi(x_k) - \phi(x^*) \\ &= \phi'(x^*)(x_k - x^*) + \frac{1}{2}\phi''(x^*)(x_k - x^*) + \cdots + \frac{1}{p!}\phi^{(p)}(\xi_k)(x_k - x^*)^p \\ &= \frac{1}{p!}\phi^{(p)}(\xi_k)(x_k - x^*)^p \end{aligned} \tag{13}$$

其中 ξ_k 为 x_k 和 x^* 之间的某个数。很明显这个时候迭代法 p 阶收敛，收敛常数为 $\left|\frac{1}{p!}\phi^{(p)}(x^*)\right|$。

5.3 Newton-Ralphson 方法及其推广

5.3.1 一维的情形

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

$$f(x^*) = 0 = f(x_0) + f'(x_0)(x^* - x_0) + \cdots \tag{14}$$

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：

$$x^* \approx x_0 - \frac{f(x_0)}{f'(x_0)} \tag{15}$$

这实际上就是著名的 Newton-Ralphson 求根方法。它实质上是一种不动点迭代法，函数 $\phi(x) = x - \frac{f(x)}{f'(x)}$，当然前提是函数 $f(x)$ 一阶导数不能为 0。我们后面会讨论导数为 0，也就是有重根时候的情形。Newton-Ralphson 方法的具体的实现为：

$k := 0$

While $|f(x_k)| > \epsilon_1$ or $|x_k - x_{k-1}| > \epsilon_2$ do

$\quad x_{k+1} := x_k - \dfrac{f(x_k)}{f'(x_k)}$

$\quad k := k + 1$

End

$x := x_k$ (16)

我们看到，在牛顿法的每一步迭代中除了需要计算一次函数值 $f(x_k)$ 之外，还需要计算一次函数的导数的值 $f'(x_k)$。这是与对分法的最大区别。

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 Newton- Ralphson 方法是平方收敛，这是因为

$$\phi'(x)\big|_{x=x^*} = 1 - \frac{f'(x)}{f'(x)} + f(x)\frac{f''(x)}{f'(x)^2}\big|_{x=x^*} = 0. \tag{17}$$

当然这个结论的前提是 $f'(x^*) \neq 0$, 也就是说方程只有单根。如果方程具有重根, 那么情况更为复杂。我们考虑 x^* 是方程 $f(x) = 0$ 的 m 重根, 这实际意味着 $f^{(j)}(x^*) = 0, j = 0, 1, \cdots, m-1$, 而 $f^{(m)}(x^*) \neq 0$。我们下面对迭代的收敛性进行分析

$$\begin{aligned} \epsilon_{k+1} &= x_{k+1} - x^* = \epsilon_k - \frac{f(x_k)}{f'(x_k)} \\ \Rightarrow \quad \frac{\epsilon_{k+1}}{\epsilon_k} &= 1 - \frac{f(x_k)}{f'(x_k)(x_k - x^*)} = 1 - \frac{f(x_k) - f(x^*)}{f'(x_k)(x_k - x^*)} \end{aligned} \tag{18}$$

做 Taylor 展开以后，我们发现

$$\frac{\epsilon_{k+1}}{\epsilon_k} \approx 1 - \frac{1}{m} \tag{19}$$

这表明，在有重根的情况下，Newton-Ralphson 方法为局部线性收敛，收敛常数为 $1 - \frac{1}{m}$。一般来讲这个收敛性是比较差的。

另外，Newton-Ralphson 求根方法还有一个潜在的弱点，就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 $\Delta x_k = x_{k+1} - x_k = -\frac{f(x_k)}{f'(x_k)}$ 可能很大。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。

下面我们介绍一下不动点迭代法的判停准则。主要有以下三种

- 残差判据，即要求 $|f(x_k)| \leq \epsilon_1$
- 误差判据，即要求 $|x_{k+1} - x_k| \leq \epsilon_2$
- 相对误差判据，即要求 $|x_{k+1} - x_k| \leq \epsilon_3 |x_{k+1}|$

残差判据有一个缺陷，就是当问题比较敏感时，$|f(x_k)|$ 很小并不意味着 x_k 很接近 x^*。而两种误差判据也有一个缺陷，就是当近似解序列 $\{x_k\}$ 收敛很慢时，$|x_{k+1} - x_k|$ 很小并不能说明 $|x_{k+1} - x^*|$ 很小。因此，实际应用时，往往将这三种判据组合起来使用，有时也需要根据问题特点和经验额外设置条件。

5.3.2 多维的推广

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

$$\begin{cases} f_1(x) = f_1(x_1, \cdots, x_n) = 0 \\ \cdots \\ f_n(x) = f_n(x_1, \cdots, x_n) = 0 \end{cases} \tag{20}$$

类似于一维时的公式，我们这时有: $f(x^*) = 0 = f(x_0) + Df(x_0) \cdot (x^* - x_0) + \cdots$ ，其中 $Df(x)$ 表示函数 $f : R^n \to R^n$ 的 Jacobi 矩阵：

$$[Df(x)]_{ij} = \frac{\partial f_i(x)}{\partial x_j} \tag{21}$$

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

$$x_{i+1} = x_i - [Df(x_i)]^{-1} \cdot f(x_i), \quad i = 0, 1, 2, \cdots \tag{22}$$

当然其停止的条件一般是要求 $\Delta x_i = x_{i+1} - x_i$ 在 n 维空间的模 $||\Delta x_i||$ 足够小。

5.4 其他的方法

5.4.1 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:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \tag{23}$$

这样一来，牛顿法的迭代可以化为:

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})} \tag{24}$$

这就是所谓的割线法。在迭代关系 $x_{i+1} = \phi(x_i, x_{i-1})$ 中，函数 ϕ 含有两个参数。所以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。如果我们令第 i 次迭代与真实的根之间的误差记为 $\epsilon_i = x_i - x^*$，我们可以得到如下的估计,

$$\begin{aligned} \epsilon_{i+1} &= x_{i+1} - x^* = x_i - x^* - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})} \\ &= \frac{f(x_i)(x_{i-1} - x^*) - f(x_{i-1})(x_i - x^*)}{f(x_i) - f(x_{i-1})} \\ &= \epsilon_i \epsilon_{i-1} \frac{\frac{f(x_i)}{x_i - x^*} - \frac{f(x_{i-1})}{x_{i-1} - x^*}}{f(x_i) - f(x_{i-1})} \end{aligned} \tag{25}$$

对于分母的函数来讲，我们有 $f(x_i) - f(x_{i-1}) = f'(\xi_1)(x_i - x_{i-1})$。对于分子的函数来讲，我们有 $g(x_i) - g(x_{i-1}) = g'(\xi_2)(x_i - x_{i-1})$，其中 $g(x) = \frac{f(x) - f(x^*)}{x - x^*}$，$g'(x) \approx f''(x)/2$，所以我们有

$$\epsilon_{i+1} \approx \frac{f''(\xi)}{2f'(\xi)} \epsilon_i \epsilon_{i-1} \tag{26}$$

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 $\ln \epsilon_{i+1} \approx \ln \epsilon_i + \ln \epsilon_{i-1}$，我们知道这个数列的特征方程是 $x^2 - x - 1 = 0$。用特征方程的两个解 $x_{1,2} = \frac{1 \pm \sqrt{5}}{2}$ 可以构造数列的通项 $\ln \epsilon_i = x_1^i + x_2^i$。因此我们得到:

$$\lim_{i \to \infty} \frac{\ln \epsilon_{i+1}}{\ln \epsilon_i} = \frac{1 + \sqrt{5}}{2} \quad \Rightarrow \quad \epsilon_{i+1} \sim c(\epsilon_i)^F, \quad F = \frac{1 + \sqrt{5}}{2} \tag{27}$$

因此，割线法的收敛速度介于对分法和牛顿法之间。

割线法的一个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。

事实上，对于由迭代函数 ϕ 所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:

$$\tilde{\phi}(\cdot) = \phi(\phi(\cdot)) \tag{28}$$

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下，这个复合割线法比牛顿法的收敛速度还要快。

5.5 函数极小值的寻找

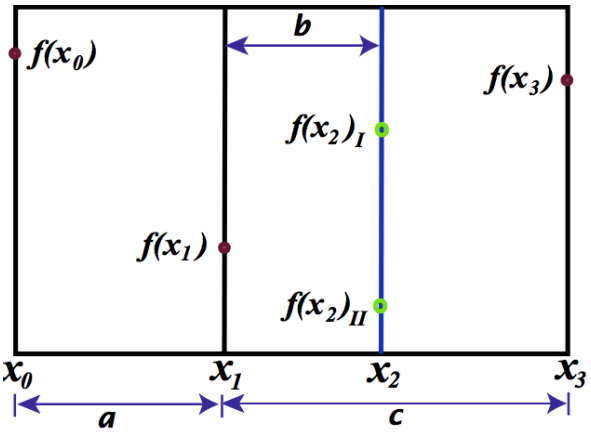
极小值的寻找有时候又称为“优化”问题 (optimization)，因为我们总是可以将需要优化的内容利用模型的方法集合在一个函数之中，而数值上寻找这个函数的极小值就对应于原先的优化问题。这类问题又大致可以分为有约束的和没有约束的两类。我们这里将仅仅讨论没有约束的情形，因为这类问题相对简单一些。

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。

5.5.1 一维函数的极小值

一维函数 $f\colon R \to R$ 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的话。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。不失一般性，我们不妨假设 $c > a$。这个时候，如果 x_2 在 x_1 的左侧，那么当 $f(x_2) > f(x_1)$ 时，极小值出现的范围为 $c + b$。因为 $c > a$，$b + c$ 还是一个比较大的范围，因此迭代的效率是比较低的。因此我们希望让 x_2 出现在 x_1 的右侧。这个时候，对于 $f(x_2) > f(x_1)$，极小值出现的范围为 $a + b$；对于 $f(x_2) < f(x_1)$，极小值出现的范围为 c。我们在设计算法的时候，总要考虑最坏的情况。为了使得迭代的效率最高，我们要让最坏的情况尽可能不那么坏，也就是说我们要让 $\max\{a + b, c\}$ 尽量的小，这个时候最好的选择是 $a + b = c$。

按照这个规则，如果 a 和 c 给定，那么 b 就定了，然后我们可以把迭代一直做下去。这里其实还有个问题，就是如何分配 a 和 c，使得算法是最优化的。很显然 $c \gg a$ 的时候，迭代的效率是很低的。我们通过第一步迭代把区间 $c + a$ 变成 c，然后再通过第二步迭代变到 $b = c - a$，基本上区间变化很小。如果 $c \approx a$ 的时候，尽管第一步的迭代 $c + a \to c$ 的效率比较高，但从第二步开始起，由于我们有 $a \gg b$，因此后续迭代的效率还是很低。我们要尽可能避免出现 $c \gg a$ 或者 $a \gg b$ 的情形。一种比较理想的情况是

$$\frac{b}{a} = \frac{a}{c}, \tag{29}$$

这种尺度的“相似性”的好处是计算机在不断迭代的过程中重复自己,算法非常稳定,永远不会出现 $c \gg a$ 或者 $a \gg b$ 的情形。我们把 $b = c - a$ 代入等式，马上得到

$$\lambda_G = \frac{a}{c} = \frac{-1 + \sqrt{5}}{2} \tag{30}$$

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。对于黄金分割迭代的收敛速度，可以证明每次迭代新的区间的长度会乘以一个因子 λ_G。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。

我们下面要考虑的一个问题是何时停止上述黄金分割迭代。这当然依赖于我们对需要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 x^* 附近我们可以利用泰勒展开:

$$f(x) \approx f(x^*) + \frac{1}{2}f''(x^*)(x-x^*)^2 \tag{31}$$

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

$$\left|\frac{f''(x^*)(x-x^*)^2}{2f(x^*)}\right| < \epsilon \quad \Rightarrow \quad \frac{|x-x^*|}{|x^*|} \approx \sqrt{\epsilon}\cdot\sqrt{\frac{2|f(x^*)|}{x^{*2}f''(x^*)}} \tag{32}$$

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。

5.5.2 多维函数的极值: 单纯形方法

我们现在考虑函数 f: $R^n \to R$ 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; 另一类则是需要函数导数信息的方法。我们首先来讨论前者; 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，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值方法是所谓的单纯形方法 (simplex method)。它由 Nelder 和 Mead 首先出，因此又称为 Nelder-Mead 算法。需要说明的，Nelder-Mead 单纯形方法与 20 世纪十大算法里的 Simplex 算法同一个名字，但它并不是十大算法。(十大算法里的 Simplex 算法是用来解线性规划问题的)。尽管 Nelder-Mead 算法没有位列十大算法，但它的原始的那篇文章的被引用将近 2 万次，也是一种非常重要的算法。我们下面来简单介绍一下这种方法。

单纯形方法可以看成是前面我们讨论过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。所谓超平面，对于二维空间来讲，就是一条直线；对于三维空间来讲，就是空间中的平面。对于更高维度空间，我们就可以类似地定义超平面。我们把单纯形的 $n+1$ 个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。单纯形更简单直观的描述就是 n 维中的 $n+1$ 个顶点的凸包，是一个多胞体：比如说直线上的一个线段，平面上的一个三角形，三维空间中的一个四面体等等。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 $n+1$ 个点，计算每个点处的函数值。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。在比较的时候我们可以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。

- 最佳点: 记为 x_0。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_n。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_{n-1}。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。

我们以最佳点 x_0 为起点，可以定义另外 n 个点的位置:

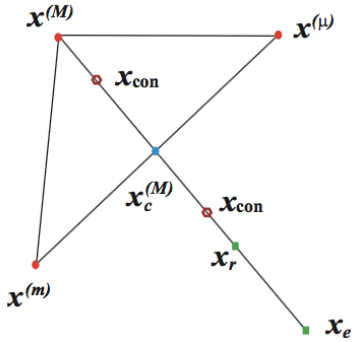
$$x_i = x_0 + h_i e_i, \quad i = 1, 2, \cdots, n \tag{33}$$

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 表征了我们所构造的空间点阵的尺度大小。我们希望这 $n+1$ 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。

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):

$$x_{k;c} = \frac{1}{n} \sum_{j=0, j\neq k}^{n} x_j \tag{34}$$

在这个算法中最为重要的是与最差点对偶的中心点 $x_{n;c}$，而且我们在整个算法里面也只需要计算这个与最差点对偶的中心点，所以我们简单记为 x_c 即可。



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。我们画出了 $n = 2$ 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:

- **反射步骤**：计算 $n+1$ 个点的函数值，如果它与 $n+1$ 点函数值的平均值之间的偏差

$$\Delta^2 = \frac{1}{n} \sum_{i=0}^{n} (f(x_i) - \bar{f})^2 < \epsilon \tag{35}$$

其中 $\epsilon > 0$ 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

$$\bar{f} = \frac{1}{n+1} \sum_{i=0}^{n} f(x_i) \tag{36}$$

否则，将最差点 x_n 相对于与它对偶的中心点 x_c 进行反射

$$x_r = (1+\alpha) x_c - \alpha x_n \tag{37}$$

其中 $\alpha > 0$ 是一个可调整的反射系数，事实上 $\alpha = 1$ 时，x_c 就是 x_n 和 x_r 的中点。这个操作在几何上实际上是将点 x_n 的相对于其他点的重心位置进行了反射。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。

- 如果函数值介于最佳值和次差值之间，$f(x_0) \le f(x_r) < f(x_{n-1})$，则用 x_r 替代最差顶点 x_n 并回到上一步继续，也就是说用我们得到的新的单纯形进行反射步骤。可能对于大多数的情况，新得到的函数 $f(x_r)$ 不是特别大，又不是特别小，那么它会位于最佳值和次差值之间。把最差顶点替换了以后，我们离找最小值的任务又进了一步。

- **扩展步骤**。极端情况下，如果函数值比最佳值还小，即 $f(x_r) < f(x_0)$，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，

$$x_e = \beta x_r + (1-\beta)x_c \tag{38}$$

  其中 $\beta > 1$ 是一个可调的扩展参数。比方当 $\beta = 2$ 时，实际上 x_r 是 x_e 和 x_c 的中点，我们可以看到，相比 x_r，x_e 的位置离 x_c 更远，为的就是要把区域往远离 $x_c^{(n)}$ 的方向扩展，以期能把极小值包围到单纯形里边。注意这个关系可以等价地写为: $x_e - x_c = \beta(x_r - x_c)$。也就是说，点 x_e 一定在点 x_c 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 $f(x_e) < f(x_0)$，也就是说 $f(x_e)$ 比最佳值还要更小，则以 x_e 替代点 x_n;
  - 如果 $f(x_e) \ge f(x_0)$，那就是说我们扩展的有些过头了，因为我们做扩展的前提是 $f(x_r) < f(x_0)$，这个时候我们有 $f(x_r) < f(x_0) \le f(x_e)$，所以我们直接舍弃 x_e 点，用 x_r 替代点 x_n;

  当扩展步骤完成之后，我们得到新的单纯形，然后我们回到反射的步骤，继续进行反射操作，试图进一步把单纯性的范围缩小。

- **收缩步骤**。另一个极端情况是，$f(x_r)$ 的函数值比次差值还要大，也就是 $f(x_r) \ge f(x_{n-1})$，那么极小值应当在点 x_r 的内侧 (靠向 x_c 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 $f(x_r) < f(x_n)$，极小值点应在 x_r 与 x_c 之间，因此我们选外收缩点:

$$x_{con} = \gamma x_r + (1-\gamma)x_c, \quad \gamma \in (0,1) \tag{39}$$

  - 如果 $f(x_r) \ge f(x_n)$，极小值点应在 x_n 与 x_c 之间，因此我们选内收缩点:

$$x_{con} = \gamma x_n + (1-\gamma)x_c, \quad \gamma \in (0,1) \tag{40}$$

  现在计算函数值 $f(x_{con})$ 并做如下的比较

  - 如果 $f(x_{con}) < f(x_n)$ 且 $f(x_{con}) < f(x_r)$，则用点 x_{con} 替换点 x_n，回到反射步骤继续;
  - 当然还有一种情况，就是 $f(x_{con}) \ge f(x_n)$ 或者 $f(x_{con}) > f(x_r)$，这种情况一般比较少出现。它的出现说明，你通过向内收缩，得到的点还要比 $f(x_r)$ 或者最差点的情况更差。我们可以想像，如果我们已经在极小值的邻域内做单纯形搜索，那么，这种情况是不会出现的。向内收缩变得更差，意味着我们原来构造单纯形的尺度 h_i 太大了，而我们后面所有的操作，都是以尺度 h_i 为单位乘上某个因子，比方说 α, β, γ 来进行的，h_i 大导致每步迭代跨越的

尺度也大，以致于收敛性不好。所以如果出现向内收缩变得更差的情况，我们就需要重新选择 h_i 的尺寸。我们保持 x_0 不变，然后把所有的 x_i 变换为

$$x_i = x_0 + \sigma h_i e_i, \quad \sigma \in (0,1) \tag{41}$$

重新产生新的 n 个点 x_k, $k = 1, 2, \cdots, n$ 以后，再将 $n+1$ 个点进行排序，然后重复以上操作。

我们在整个单纯形搜索里面有 4 个参数，与反射步骤有关的 α，与扩展有关的 β，与收缩有关的 γ，以及与尺度变化有关的 σ。如果没有特殊情况，这 4 个参数的选取分别是

$$\alpha = 1, \quad \beta = 2, \quad \gamma = 1/2, \quad \sigma = 1/2. \tag{42}$$

需要提一下的是，作为单纯形方法来讲，初始的单纯形很重要。事实上，如果一开始构造的单纯形太小，很容易导致我们一直在进行局部的搜索，而无法高效率地找到全局的极小值。因此，采用单纯形方法，尤其在构造初始单纯形的时候需要我们对问题本身有一定的了解。

5.6 多维函数的极值: 下降方法

下面我们会介绍共轭梯度算法，也就是 CG(conjugate gradient) 算法。这个算法首先是由 M. Hestenes 和 E. Stiefel 提出来的, 属于 Krylov 子空间迭代法的一种。人们把这个算法和之前 C. Lanczos 提出的构建 Krylov 子空间的方法合并在一起，称为 **Krylov 子空间迭代法**。它是 20 世纪十大算法中的一种。在介绍共轭梯度法之前，我们首先介绍最速下降法，它是共轭梯度法的基础。

5.6.1 最速下降法

在大多数情况下, CG 算法是用来解线性方程组的, 特别是运用在大型稀疏矩阵的线性方程组的迭代求解中。那为什么我们要把这个算法放在多维函数求极值的章节里面呢。我们下面来解释一下。我们考虑线性方程组

$$A\vec{x} = \vec{b} \tag{43}$$

的求解问题。假设 A 是 n 阶实对称正定矩阵。求解线性方程组的其中一种思路是将它转化为在 n 维向量空间求函数最小值的问题，这个函数是下面的 n 元二次函数

$$\varphi(x) = \frac{1}{2}\vec{x}^T A\vec{x} - \vec{b}^T\vec{x} \tag{44}$$

这个最小值点必定满足条件 $\frac{\partial\varphi(x)}{\partial x_i} = 0$, $(i = 1, 2, \cdots, n)$，不难证明，这样得到的 n 个方程就是线性方程组

$$A\vec{x} - \vec{b} = 0 \tag{45}$$

这说明方程组的解就是 n 元二次函数 $\varphi(x)$ 的极小点。还可以证明，当 A 是对称正定的时候，这个极小点就是 $\varphi(x)$ 唯一的最小值点。这种将对称正定线性方程组的求解问题转化为多元二次函数最小值问题的方法称为变分原理。

现在的问题是如何有效地寻找多元二次函数的最小值, 我们这里先介绍最速下降法的基本思路。先给定一个初始向量 $\vec{x}_0$，假设沿方向 $\vec{p}_0$ 搜索下一个点

$$\vec{x}_1 = \vec{x}_0 + \alpha_0\vec{p}_0 \tag{46}$$

使得 $\vec{x}_1$ 为这个方向上的最小点。然后从 $\vec{x}_1$ 出发，选定一个搜索方向 $\vec{p}_1$，沿直线 $\vec{x}=\vec{x}_1+\alpha\vec{p}_1$ 再跨一步，即找到 α_1 使得 $\varphi(\vec{x}_1+\alpha_1\vec{p}_1)$ 最小。按照相同的方法做下去，可以得到一系列的向量 $\vec{x}_0,\vec{x}_1,\cdots$，它们逐渐逼近 $\varphi(x)$ 在全空间的最小值点，也就是方程组的解。这个搜索过程是一个迭代计算过程，其关键问题是确定搜索方向 $\vec{p}_k$ 和搜索步长 α_k, $(k=0,1,2,\cdots,)$.

我们先分析如何去定搜索步长。假设从 $\vec{x}_k$ 出发，已选定搜索方向为 $\vec{p}_k$，令

$$f(\alpha)=\varphi(\vec{x}_k+\alpha\vec{p}_k) \tag{47}$$

则搜索步长 α_k 是使一元函数 $f(\alpha)$ 取最小值的 α 值。把 $f(\alpha)$ 写得更具体一点

$$\begin{aligned} f(\alpha) &= \frac{1}{2}\left(\vec{x}_k+\alpha\vec{p}_k\right)^T A\left(\vec{x}_k+\alpha\vec{p}_k\right)-\vec{b}^T(\vec{x}_k+\alpha\vec{p}_k) \\ &= \frac{1}{2}\alpha^2\vec{p}_k^T A\vec{p}_k-\alpha\vec{r}_k^T\vec{p}_k+\varphi(\vec{x}_k) \end{aligned} \tag{48}$$

其中 $\vec{r}_k=\vec{b}-A\vec{x}_k$ 是方程组的残差。函数 $f(\alpha)$ 为简单的一元二次函数，且矩阵 A 对称正定保证了 $\vec{p}_k^T A\vec{p}_k>0$。所以 $f(\alpha)$ 有唯一的最小值，对应的 α 值为

$$\alpha=\frac{\vec{r}_k^T\vec{p}_k}{\vec{p}_k^T A\vec{p}_k} \tag{49}$$

对应不同的方法来讲，确定搜索步长这一步基本是一致的。但是确定搜索方向 $\vec{p}_k$ 的策略却可以各有不同。

一个最直接的确定搜索方向 $\vec{p}_k$ 的方法是最速下降法。也就是把 $\vec{p}_k$ 取为让多元函数 $\varphi(x)$ 增加最快的方向也就是其梯度方向的反方向。一个多元函数的梯度方向为 $D\varphi(x)$。按照这个思路

$$\vec{p}_k=-D\varphi(\vec{x}_k)=-(A\vec{x}-\vec{b})|_{\vec{x}=\vec{x}_k}=\vec{b}-A\vec{x}_k=\vec{r}_k \tag{50}$$

如果我们画出 $\varphi(x)$ 的等值线,那么 $\vec{p}_k=\vec{r}_k$ 应该是等值线的法线方向。另外,我们还可以证明,$\vec{p}_k\perp\vec{r}_{k+1}$, 也就是说下一步搜索的方向和上一步搜索的方向是垂直的。我们来看一下，首先 $\vec{p}_k=\vec{r}_k$，我们有

$$\alpha_k=\frac{r_k^T r_k}{r_k^T A r_k},\quad \vec{x}_{k+1}=\vec{x}_k+\alpha_k\vec{r}_k \tag{51}$$

从 $\vec{x}_{k+1}$ 开始，下一步搜索的方向是

$$\begin{aligned} \vec{r}_{k+1} &= \vec{b}-A\vec{x}_{k+1}=\vec{b}-A\vec{x}_k-\alpha_k A\vec{r}_k=\vec{r}_k-\alpha_k A\vec{r}_k \\ &\Rightarrow \quad \vec{r}_k^T\vec{r}_{k+1}=0 \end{aligned} \tag{52}$$

所以我们证明每次的搜索方向和上一次的搜索方向都是垂直的。这一点也比较好理解。如果不是垂直的，那就说明在上一次搜索的时候还需要在搜索的方向走得更远或者更近一点才能达到局域的最小值,这和我们的搜索判定是矛盾的。

关于最速下降的算法，给出来也很简单

$$\begin{aligned} &\vec{r}:=\vec{b}-A\vec{x} \\ &\text{While } ||\vec{r}||>\epsilon \text{ do} \\ &\qquad \alpha:=\frac{r^T r}{r^T A r} \\ &\qquad \vec{x}:=\vec{x}+\alpha\vec{r} \\ &\qquad \vec{r}:=\vec{r}-\alpha A\vec{r} \\ &\text{End} \end{aligned} \tag{53}$$

在这个算法中，每次迭代的主要计算是做矩阵 A 与向量 $\vec{r}$ 的乘法，如果 A 为稀疏矩阵的话，那么乘法只需要遍历矩阵所有非零元素即可，而且算法不改变原始的矩阵 A，所以这种方法对于大型稀疏矩阵是比较有效的。而且 A 是对称正定矩阵保证了多元二次函数 $\varphi(x)$ 是凸函数，一定存在唯一的最小值点，也就是说，当矩阵 A 是对称正定的时候，算法一定是收敛的。

最快下降算法的一个缺点是，负梯度的方向只是从局部来看是最佳搜索方向，而且每一步都是在该选定方向上使函数值达到最小，但它并没有在全局上使得 $\varphi(x)$ 最小化。从我们的搜索路径上来看是沿着一个多重的前后垂直的折线在行进。这样的效果是需要很多小碎步才能接近极小值，而且搜索方向会被不停的重复，因此实际上是效率很低的。

5.6.2 共轭梯度法

下面我们介绍共轭梯度法，它是对最速下降法的一种优化。给定初始向量 $\vec{x}_0$，第一步仍选负梯度方向为搜索方向，即 $\vec{p}_0 = \vec{r}_0$，于是有

$$\alpha_0 = \frac{\vec{r}_0^T \vec{r}_0}{\vec{p}_0^T A \vec{p}_0}, \quad \vec{x}_1 = \vec{x}_0 + \alpha_0 \vec{p}_0, \quad \vec{r}_1 = b - A\vec{x}_1 \tag{54}$$

从第二步开始起，我们做一下变化。最速下降法对应于在迭代的第 k 步，沿着 $\vec{r}_k$ 进行优化：

$$x_{k+1} : \min_\lambda \varphi(x_k + \lambda r_k), \quad r_k = b - Ax_k \tag{55}$$

这相当于在 $\vec{r}_k$ 的方向实行一个一维的搜索。

共轭梯度法的想法是，搜索方向不仅仅考虑 $\vec{r}_k$，而是在过点 $\vec{x}_k$ 由向量 $\vec{r}_k$ 和 $\vec{p}_{k-1}$ 所张成的平面 $\{\vec{x} = \vec{x}_k + \xi\vec{r}_k + \eta\vec{p}_{k-1}\}$ 内寻找函数 $\varphi(x)$ 的最小值，记为

$$\begin{aligned} f(\xi,\eta) &= \varphi(\vec{x}_k + \xi\vec{r}_k + \eta\vec{p}_{k-1}) \\ &= \frac{1}{2}(\vec{x}_k + \xi\vec{r}_k + \eta\vec{p}_{k-1})^T A(\vec{x}_k + \xi\vec{r}_k + \eta\vec{p}_{k-1}) - b^T(\vec{x}_k + \xi\vec{r}_k + \eta\vec{p}_{k-1}) \end{aligned} \tag{56}$$

直接计算可得到两个方程

$$\begin{aligned} \frac{\partial f}{\partial \xi} &= \xi\vec{r}_k^T A\vec{r}_k + \eta\vec{r}_k^T A\vec{p}_{k-1} - \vec{r}_k^T\vec{r}_k = 0 \\ \frac{\partial f}{\partial \eta} &= \xi\vec{r}_k^T A\vec{p}_{k-1} + \eta\vec{p}_{k-1}^T A\vec{p}_{k-1} = 0 \end{aligned} \tag{57}$$

其中第二个式子用到了 $\vec{r}_k^T\vec{p}_{k-1} = 0$ 的结论，因为 $\vec{p}_{k-1}$ 是搜索方向，$\vec{r}_k$ 是残差，这两个向量必然垂直，所以 $\vec{r}_k^T\vec{p}_{k-1} = 0$ 这个结论在共轭梯度法里面也是成立的。假设要求的极小值为 $\tilde{x}$，那么可以令

$$\tilde{x} = x_k + \tilde{\xi}\vec{r}_k + \tilde{\eta}\vec{p}_{k-1} \equiv \vec{x}_{k+1} \tag{58}$$

其中 $\tilde{\xi}$ 和 $\tilde{\eta}$ 满足 $\frac{\partial f}{\partial \xi} = 0$ 和 $\frac{\partial f}{\partial \eta} = 0$ 的方程组。一旦 $\tilde{x}$ 定了以后，新的搜索方向也就定下来了

$$\vec{p}_k \equiv \frac{1}{\tilde{\xi}}(\tilde{x} - \vec{x}_k) = \vec{r}_k + \frac{\tilde{\eta}}{\tilde{\xi}}\vec{p}_{k-1} \tag{59}$$

令 $\beta_{k-1} = \frac{\tilde{\eta}}{\tilde{\xi}}$，由 $\frac{\partial f}{\partial \eta} = 0$ 的方程得到

$$\beta_{k-1} = -\frac{\vec{r}_k^T A\vec{p}_{k-1}}{\vec{p}_{k-1}^T A\vec{p}_{k-1}} \tag{60}$$

那么 $\vec{p}_k$ 就可以直接写成

$$\boxed{\vec{p}_k = \vec{r}_k + \beta_{k-1}\vec{p}_{k-1}} \tag{61}$$

另外，搜索步长是由 $\tilde{\xi}$ 给出的，我们可以令 $\alpha_k = \tilde{\xi}$，根据方程 (71) 和 $\frac{\partial f}{\partial \xi} = 0$，我们有

$$\alpha_k = \frac{\vec{r}_k^T \vec{p}_k}{\vec{p}_k^T A \vec{p}_k} \tag{62}$$

(这里有用到 $\vec{p}_k$ 和 $\vec{p}_{k-1}$ 关于 A 共轭正交的性质，后面我们会讨论到。)

下一个迭代的解为

$$\boxed{\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{p}_k} \tag{63}$$

而新的残差定义为

$$\boxed{\vec{r}_{k+1} = \vec{b} - A\vec{x}_{k+1} = \vec{r}_k - \alpha_k A\vec{p}_k} \tag{64}$$

目前，我们得到了共轭梯度算法三个向量 $\vec{x}_k$, $\vec{r}_k$ 和 $\vec{p}_k$ 的迭代关系式，这三个向量分别被称为近似解向量、残差向量和搜索方向向量。有了它们的迭代关系，我们可以给出共轭梯度法的算法伪码

$$\begin{aligned}
&\vec{r}_0 = \vec{b} - A\vec{x}_0, \quad \vec{p}_0 = \vec{r}_0, \quad k = 0 \\
&\text{While } ||\vec{r}_k|| > \epsilon \text{ do} \\
&\qquad \alpha_k = (\vec{r}_k^T \vec{p}_k)/(\vec{p}_k^T A\vec{p}_k) \quad \text{计算搜索步长} \\
&\qquad \vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{p}_k \quad \text{更新解} \\
&\qquad \vec{r}_{k+1} = \vec{r}_k - \alpha_k A\vec{p}_k \quad \text{计算新残差向量} \\
&\qquad \beta_k = -(\vec{r}_{k+1}^T A\vec{p}_k)/(\vec{p}_k^T A\vec{p}_k) \\
&\qquad \vec{p}_{k+1} = \vec{r}_{k+1} + \beta_k \vec{p}_k \quad \text{计算新的搜索方向} \\
&\qquad k = k + 1 \\
&\text{End}
\end{aligned} \tag{65}$$

我们来看一下共轭梯度法好在什么地方，首先它在一个平面上，而不在一条线上进行最小值搜索，那么它比最速下降法来得优化是很显然的。但如果仅仅如此，那么共轭梯度法也只是改进了一点点。事实上，共轭梯度法的好处正在于“共轭”两字。

我们先来看这样一件事情，不管我们搜索方向 p 如何选取，我们称 x_k 是相对于方向 p 的最优化的点，如果它满足

$$\varphi(x_k) \le \varphi(x_k + \lambda p), \quad \forall \lambda \in R \tag{66}$$

假设搜索方向 p 的选择不只一种，它构成了一个矢量空间 V，那么我们称 x_k 是相对于矢量空间 V 的最优化的点，如果它满足上面这个式子。我们看一下最优化的点有哪些性质。首先，在最优化的点处，$\varphi(x_k + \lambda p)$ 相对 λ 的导数为 0

$$\frac{\partial \varphi(x_k + \lambda p)}{\partial \lambda}\Big|_{\lambda=0} = p^T(Ax_k - b) = -p^T r_k = 0 \tag{67}$$

也就是说，由 x_k 所给出的残差向量 r_k 正交于所有的矢量 p，只要 $p \in V$。

如果按照最速下降法，我们从 x_k 出发得到 $x_{k+1} = x_k + \alpha_k r_k$，$\alpha_k$ 的选取使得 $\varphi(x_{k+1})$ 最小。也就是说 x_{k+1} 是相对于方向 $p_k = r_k$ 是最优化的点，也就是说 $r_{k+1} \perp p_k$。但我们可以证明 x_{k+2} 却不

是相对于 r_k 最优化的点，也就是说 $r_{k+2} \perp r_k(p_k)$ 不成立。这个证明留待同学们自己去验证以下。我们很自然得会问，是否存在一种迭代方式，

$$x_{k+1} = x_k + q \tag{68}$$

使得 x_k 相对于矢量空间 V 中的任意方向 p 是最优化的，而后续的 x_{k+1} 也是相对于 p 是最优化的。如果这样的迭代方式存在，那么我们不难看出，我们后续搜索得到的点，依然是关于之前搜索过的 V 空间的最优化的点，所以我们没有做任何重复的搜索。我们可以问这样一个问题：如果这样的迭代，也就是这样的方向 q 存在，那么它应该满足什么样的性质？这里我们不妨假设 x_k 已经是关于向量 p 最优化的点，那么我们有残差向量 $r_k \perp p$。我们希望在迭代的过程中 x_{k+1} 也是相对于 p 是最优化的点，那么 $r_{k+1} \perp p$。于是我们马上得到关系式

$$0 = p^T r_{k+1} = p^T(b - Ax_{k+1}) = p^T(r_k - Aq) = -p^T Aq \tag{69}$$

我们把中间包含矩阵 A 的正交关系 $p^T Aq = 0$ 称为向量 p 和 q 关于矩阵 A 共轭正交。这也是共轭梯度法的名称来源。我们需要找到一个搜索方向 q，使得 q 与之前 V 空间里的所有向量 p 都 A-共轭正交。这样我们 x_k 和后续的迭代 x_{k+1} 都是相对于空间 V 最优化的点。

事实上，这件事情是可以做到的，而做法就是我们之前提到的在 r_k 和 p_{k-1} 构成的平面上去寻找最小值 x_{k+1}。我们得到的一个新的搜索方向为

$$p_k = r_k + \beta_{k-1} p_{k-1}, \quad \beta_{k-1} = -\frac{r_k^T A p_{k-1}}{p_{k-1}^T A p_{k-1}} \tag{70}$$

我们下面证明这个迭代具有如下性质：

- 搜索方向 p_k 与所有的 p_j $(j < k)$ 都 A-共轭正交，或者说，向量序列 $p_0, p_1, \cdots, p_{k-1}, p_k$ 两两 A-共轭正交
- 残差向量 r_{k+1} 与所有的 p_j $(j < k+1)$ 正交
- 残差向量 r_{k+1} 与所有的 r_j $(j < k+1)$ 正交，或者说，向量序列 $r_0, r_1, \cdots, r_k, r_{k+1}$ 两两正交

我们用数学归纳法来证明这三条。首先假设向量 $p_0, p_1, \cdots, p_{k-1}$ 两两 A-共轭正交；r_k 与所有的 p_j $(j < k)$ 正交；向量序列 $r_0, r_1, \cdots, r_k$, 两两正交。那么，我们有，当 $j < k-1$ 时

$$p_j^T A p_k = p_j^T A(r_k + \beta_{k-1} p_{k-1}) = p_j^T A r_k = \frac{1}{\alpha_j}(r_j^T - r_{j+1}^T) r_k = 0 \tag{71}$$

当 $j = k-1$ 时，利用 β_{k-1} 的定义，得到

$$p_{k-1}^T A p_k = p_{k-1}^T A r_k + \beta_{k-1} p_{k-1}^T A p_{k-1} = 0 \tag{72}$$

第一条性质得以证明。

另外，我们还有，当 $j < k$ 时

$$p_j^T r_{k+1} = p_j^T(r_k - \alpha_k A p_k) = p_j^T r_k = 0 \tag{73}$$

当 $j = k$ 时

$$\begin{aligned} \vec{p}_k^T \vec{r}_{k+1} &= \left(\vec{r}_k^T + \beta_{k-1}\vec{p}_{k-1}^T\right)(\vec{r}_k - \alpha_k A\vec{p}_k) = \vec{r}_k^T \vec{r}_k - \alpha_k \vec{r}_k^T A\vec{p}_k \\ &= \vec{r}_k^T(\vec{p}_k - \beta_{k-1}\vec{p}_{k-1}) - \alpha_k \vec{r}_k^T A\vec{p}_k = \vec{r}_k^T \vec{p}_k - \alpha_k \vec{r}_k^T A\vec{p}_k = 0 \end{aligned} \tag{74}$$

于是第二条性质得以证明。

最后，我们还有，当 $j < k$ 时

$$r_{k+1}^T r_j = (r_k^T - \alpha_k p_k^T A) r_j = -\alpha_k p_k^T A r_j = -\alpha_k p_k^T A(p_j - \beta_{j-1} p_{j-1}) = 0 \tag{75}$$

当 $j = k$ 时，有 (第一个等式右边需要代入 α_k 的值。)

$$\vec{r}_{k+1}^T \vec{r}_k = (\vec{r}_k^T - \alpha_k \vec{p}_k^T A)\vec{r}_k = \vec{r}_k^T \vec{r}_k - \vec{r}_k^T \vec{p}_k = \vec{r}_k^T(-\beta_{k-1}\vec{p}_{k-1}) = 0 \tag{76}$$

所以第三条性质得以证明。

我们通过这三条性质的证明，可以看到，共轭梯度法实现了寻找全局最小值的功能。我们来看一下搜索方向 $\vec{x}_k$ 的迭代过程

$$\vec{x}_1 = \vec{x}_0 + \alpha_0 \vec{p}_0 = \vec{x}_0 + \alpha_0 \vec{r}_0 \tag{77}$$

所以 $\vec{x}_1$ 实际上是位于通过 $\vec{x}_0$ 点的以向量 $\vec{r}_0$ 为基的子空间里面。接着往下迭代

$$\vec{x}_2 = \vec{x}_1 + \alpha_1 \vec{p}_1 = \vec{x}_1 + \alpha_1(\vec{r}_1 + \beta_0 \vec{p}_0) = \vec{x}_1 + \alpha_1\beta_0\vec{r}_0 + \alpha_1(\vec{r}_0 - \alpha_0 A\vec{r}_0) \tag{78}$$

所以 $\vec{x}_2$ 是位于通过 $\vec{x}_0$ 点的以向量 $\{\vec{r}_0, A\vec{r}_0\}$ 为基的子空间里面。事实上，我们可以得到 $\vec{x}_{k+1}$ 是位于通过 $\vec{x}_0$ 点的以向量 $\{\vec{r}_0, A\vec{r}_0, \cdots, A^k\vec{r}_0\}$ 为基的子空间里面。可以记为

$$\vec{x}_{k+1} \in \vec{x}_0 + \text{span}\{\vec{r}_0, A\vec{r}_0, \cdots, A^k\vec{r}_0\} \tag{79}$$

其中 $\text{span}\{\vec{r}_0, A\vec{r}_0, \cdots, A^k\vec{r}_0\}$ 称为 Krylov 子空间，简写为 $\mathcal{K}_{k+1}(A, \vec{r}_0)$。

共轭梯度法产生的近似解向量 $\vec{x}_{k+1}$ 不光位于 Krylov 子空间中，而且它还有一系列非常好的性质。

- $\vec{x}_{k+1}$ 不但在向量 $\vec{r}_k$ 和 $\vec{p}_{k-1}$ 所张成的平面上使 $\varphi(x)$ 最小，而且在超平面 $\vec{x}_0 + \mathcal{K}_{k+1}(A, \vec{r}_0)$ 也使得 $\varphi(x)$ 最小。
- 搜索方向向量满足 A 共轭正交关系 $\vec{p}_i^T A \vec{p}_j = 0,\ i \neq j$
- 搜索方向向量与残差向量相互正交 $\vec{p}_j^T \vec{r}_k = 0,\ 0 \leq j < k$
- 残差向量相互正交 $\vec{r}_i^T \vec{r}_j = 0,\ i \neq j$
- 在超平面 $\vec{x}_0 + \mathcal{K}_k(A, \vec{r}_0)$ 上的所有点中，近似解 $\vec{x}_k$ 的范数误差最小

$$||\vec{x}_k - \vec{x}^*||_A = \min_{\vec{x} \in \vec{x}_0 + \mathcal{K}_k(A, \vec{r}_0)} ||\vec{x} - \vec{x}^*||_A \tag{80}$$

这里 $\vec{x}^*$ 表示方程 $A\vec{x} = \vec{b}$ 的准确解，而 $||\cdot||_A$ 是由矩阵 A 定义的向量范数 $||\vec{x}||_A = \sqrt{\vec{x}^T A \vec{x}}$

从第一条性质我们可以看出，共轭梯度法迭代可以看成是在 Krylov 子空间寻找最优化的向量。应当指出，还有其他的通过搜索 Krylov 子空间得到线性方程组近似解的方法，所有这些方法都可以统称为 Krylov 子空间迭代法。另外，对于 n 个方程构成的方程组来讲，它的解肯定可以在一个 n 维空间中找到。如果我们扩展 Krylov 子空间到 n 维：$\mathcal{K}_n(A, \vec{r}_0) = \text{span}\{\vec{r}_0, A\vec{r}_0, \cdots, A^{n-1}\vec{r}_0\}$，那么这个时候搜索到的最优化的点 x_n，就是准确解 (如果我们不考虑数值误差的话)。

我们通过前面得到的正交化关系，还可以进一步简化共轭梯度法的迭代计算

$$\begin{aligned}\alpha_k &= \frac{\vec{r}_k^T\vec{p}_k}{\vec{p}_k^T A\vec{p}_k} \quad \Rightarrow \quad \alpha_k = \frac{r_k^T r_k}{p_k^T A p_k} \\ \beta_k &= -\frac{\vec{r}_{k+1}^T A\vec{p}_k}{\vec{p}_k^T A\vec{p}_k} \quad \Rightarrow \quad \beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}\end{aligned} \tag{81}$$

其中第一个 α_k 的变换式子很显然。对于 β_k 来讲，其实也不复杂

$$\beta_k = -\frac{\vec{r}_{k+1}^T A\vec{p}_k}{\vec{p}_k^T A\vec{p}_k} = -\frac{1}{\alpha_k}\frac{\vec{r}_{k+1}^T(r_k - r_{k+1})}{\vec{p}_k^T A\vec{p}_k} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \tag{82}$$

最后一步利用了 α_k 的表达式。

有了这两个简化以后的表达式，我们可以给出更实用的共轭梯度法算法描述

$\vec{r} = \vec{b} - A\vec{x}, \quad \vec{p} = \vec{r}$

While $||\vec{r}|| > \epsilon$ do

$\alpha = (\vec{r}^T\vec{r})/(\vec{p}^T A\vec{p})$ 计算搜索步长

$\vec{x} = \vec{x} + \alpha\vec{p}$ 更新解

$\tilde{r} = r$ 保存上一个残差向量

$r = \vec{r} - \alpha A\vec{p}$ 计算新残差向量

$\beta = (\vec{r}^T\vec{r})/(\tilde{r}^T\tilde{r})$

$\vec{p} = \vec{r} + \beta\vec{p}$ 计算新的搜索方向

End (83)

从这个算法可以看出来，共轭梯度法的每次迭代只需要做一次矩阵向量乘法 (Ap)，以及两次向量内积 $p^T Ap$ 和 $r^T r$。从存储上看，共轭梯度法也是非常节省的。

我们下面来看一下 CG 算法的收敛速度。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：

$$\frac{||e_k||_A}{||e_0||_A} \leq 2\left(\frac{\sqrt{c}-1}{\sqrt{c}+1}\right)^k \tag{84}$$

其中 $c = \lambda_{max}/\lambda_{min}$ 是矩阵 A 的条件数。当矩阵是比较良态的时候 $c \approx 1$，收敛会很迅速；但当 A 非常病态的时候 $c \gg 1$，收敛则可能非常慢，甚至不收敛。其实这个事情也很好理解。CG 算法是一种 Krylov 子空间迭代法。在我们构造 Krylov 子空间的时候，它的基矢是由 $r_0, Ar_0, \cdots, A^k r_0$ 来给出。我们可以把 r_0 用 A 的本征矢量 v_i 来分解 $r_0 = \sum_{i=1}^n a_i v_i$。对于 $A^k r_0$ 来讲

$$A^k r_0 = \sum_{i=1}^n a_i \lambda_i^k v_i \tag{85}$$

我们可以看到，随着 k 增加，$A^k r_0$ 在最大的本征矢上的投影会越变越大，因此，当 k 比较大的时候 $A^k r_0$ 往往变得线性相关，或者至少是近似地线性相关。这个时候扩大 Krylov 子空间并不能帮我们有效地搜索最小值。而且当条件数越大，$A^k r_0$ 的线性相关度就会越高，情况也就变得越糟糕。

对于共轭梯度法来讲，虽然，共轭梯度法经过 n 次迭代，必然可以找到全局最小值，但实际上随着迭代步数的增多，舍入误差会变得很严重，我们基本上不会真的通过 n 次迭代直接求方程组的解。我

们一般面临的无非两种情况，一种是条件数较小，这个时候，我们通过 $k \ll n$ 次迭代就已经可以得到大型方程组的解了。还有一种是条件数很大，这个时候我们一般要引进所谓的预条件技术。它的基本做法是用矩阵 M^{-1} 去乘 A，然后求解

$$M^{-1}Ax = M^{-1}b \tag{86}$$

其中 M 是使方程组 $Mz = y$ 易于求解的矩阵，如果它的逆近似于 A 的逆，则 $M^{-1}A$ 就会变得比较良态，从而提高共轭梯度法的收敛速度。这么讲好像比较抽象，我们可以想象一种比较特殊的情况，就是矩阵 A 的本征值和它的对角元相差得不是很大，那么我们就可以把 M 取成 A 的对角元所构成的矩阵。它既很容易求逆，又比较接近 A。一般把这种预条件称为 Jacobi 预条件。事实上预条件计算的研究是一个非常重要的课题，有兴趣的同学可以自己找一些文献来看。从技术上来讲，为了保持系数矩阵的正定性和对称性，我们还应该对 M 做分解 $M = LL^T$，然后对 $L^{-1}A(L^{-1})^T$ 应用共轭梯度法。我们在算法里面可以应用一些技巧，使得迭代过程中只出现 M 而不出现 L。下面给出预条件共轭梯度法的算法描述

$$\begin{aligned}
&\vec{r} = \vec{b} - A\vec{x}, \quad \vec{p} = M^{-1}\vec{r}, \quad z = p \quad \{z = M^{-1}r\} \\
&\text{While } ||\vec{r}|| > \epsilon \text{ do} \\
&\qquad \alpha = (\vec{r}^T\vec{z})/(\vec{p}^T A\vec{p}) \quad \text{计算搜索步长} \\
&\qquad \vec{x} = \vec{x} + \alpha\vec{p} \quad \text{更新解} \\
&\qquad \delta = r^T z \quad \text{保存上一个残差向量的有关计算结果} \\
&\qquad r = \vec{r} - \alpha A\vec{p} \quad \text{计算新残差向量} \\
&\qquad z = M^{-1}r \\
&\qquad \beta = (\vec{r}^T\vec{z})/\delta \\
&\qquad \vec{p} = \vec{z} + \beta\vec{p} \quad \text{计算新的搜索方向} \\
&\text{End}
\end{aligned} \tag{87}$$

预条件技术的主要作用就是为了减小矩阵的条件数。事实上，我们之前课上讲过的一个例子，也是要减小条件数的。我们求得矩阵 A 的前 m 个本征值和本征矢量，可以构造投影算符 P_0 和 $1 - P_0$，其中 P_0 投影出小本征矢量所构成的空间，$1 - P_0$ 投影出大本征矢量所构成的空间。然后我们把解方程组分成两步来完成

$$A\vec{x}_H = (1 - P_0)\vec{b}, \quad A\vec{x}_L = P_0\vec{b} \tag{88}$$

解 x_H 时，因为与小本征值有关区域已经给投影出去了，所以条件数大大减小，而求 x_L，则可以利用我们已经得到的本征值和本征矢量直接求得。因此把这种做法和 CG 算法相结合，也能通过比较少的迭代次数，求出方程组的解。