数值分析第七章补充阅读

朱宇嘉

1 解非线性方程

1.1 一元方程求根

如果一元方程 $f(x)=0$ 不存在显式解，我们就需要使用迭代法去求根。但对于所有方程求根的迭代法，我们都要考虑以下几点：

1. 该方法**是否收敛**。
2. 该方法的**收敛速度**：我们往往用 p 阶收敛来刻画收敛速度。
3. 该方法**是否对初值敏感**：即在不同的初值下，该方法是否都能收敛，或是否都能收敛到同一个根。

1.2 非线性方程组

但是，在目前的应用领域中，大多数方程都是多元的，其形式为：

$$\begin{cases} f_1(x_1,x_2,...,x_n)=0 \\ f_2(x_1,x_2,...,x_n)=0 \\ \quad\quad ..... \\ f_n(x_1,x_2,...,x_n)=0 \end{cases}$$

我们也可以用迭代法求解非线性方程组。同样地，我们也可以用以上的三条标准来研究算法的好坏。

本章我们主要探讨以下内容。我们首先使用程序实现一元方程求根的各种算法。随后我们讨论弦截法的收敛速度。此外，由于牛顿法在实际的领域中有着广泛的应用，我们将在最后讨论一些牛顿法的替代算法和改进算法。

2 一元方程求根的程序实现

我们首先用编程语言来实现一元方程求根算法。需要实现的算法有：二分法，不动点迭代法，斯蒂芬森迭代法和牛顿法。

由于大家在期末实验中需要亲自动手编写各种一元方程求根算法，我在这里也只给出函数的接口形式，请大家自行完成函数的编写。

同样的，为了给大家一点启示，我会 R 语言和 Python 语言分别来实现一下二分法求根算法。

2.1 二分法

二分法求根的思路是：设 $f(x)$ 是定义在 $[a,b]$ 上的连续函数，若 $f(a)\cdot f(b)<0$，则可以按以下步骤求解：

- **Step 1**：取 $x_0=(a+b)/2$。若 $f(x_0)=0$，则输出 x_0。否则转到下一步。
- **Step 2**：若 $f(a)\cdot f(x_0)>0$，则 $a_1=x_0,b_1=b$，否则 $a_1=a,b_1=x_0$。
- **Step 3**：重复第二步，构造 $[a_1,b_1],[a_2,b_2],\ldots$，并计算 $x_k=(a_k+b_k)/2$。直至 $f(x_k)=0$ 停止迭代并输出 x_k。

我们可以用以下 R 语言代码来实现二分法。

```
Binary<-function(f,lo,hi,esp=1e-3){
  ## 输入: f 是方程, 我们希望求 f(x)=0 的根
  ## 输入: lo,hi 是 f 的求解范围
  ## 输入: esp 是求根误差, 默认值是 1e-3
  if(!is.function(f)) stop("f should be a function")
  if(lo>=hi) stop("interval range at least 0")
  if(f(lo)*f(hi)>=0) stop("need different sign at interval endpoint")
  if(esp<=0) stop("need positive esp")
  ## 先求 x
  iter<-0;x<-(lo+hi)/2
  ## 开始迭代
  while(abs(f(x))>esp){
    if(f(lo)*f(x)>0) lo<-x
    else hi<-x
    x<-(lo+hi)/2
    iter<-iter+1
  }
  ## 输出: x 求出的根
  ## 输出: iter: 迭代次数
```

```
  return(list(root=x,iter=iter))
}
```

我们用课件上的例 2 来验证一下。例 2 是求 $x^3 - x - 1 = 0$ 在 $[1.0, 1.5]$ 内的一个实根。

```
f<-function(x){
  return(x^3-x-1)
}
Binary(f,1,1.5,esp=1e-2)
```

```
## $root
## [1] 1.324219
##
## $iter
## [1] 6
```

从输出中，我们可以看出，使用 6 次迭代，我们可以求出 1.3242 是拥有两位准确小数点的根。

下面我们用 Python 实现一下。

```
def Binary(f,lo,hi,esp=1e-3):
  ## 输入: f 是方程，我们希望求 f(x)=0 的根
  ## 输入: lo,hi 是 f 的求解范围
  ## 输入: esp 是求根误差，默认值是 1e-3
  assert callable(f), 'f should be a function'
  assert lo<=hi, 'interval range at least 0'
  assert f(lo)*f(hi)<0, 'need different sign at interval endpoint'
  assert esp>0, 'need positive esp'
  ## 先求 x
  it = 0
  x = (lo+hi)/2
  ## 开始迭代
  while abs(f(x))>esp:
    if f(lo)*f(x)>0: lo = x
    else: hi = x
    x = (lo+hi)/2
    it = it+1
  ## 输出: x 求出的根
  ## 输出: it: 迭代次数
  return x, it
```

同样的，我们可以验证函数的正确性。其输出结果和 R 语言应该一样。

```
def f(x):
  return x**3-x-1
root, it = Binary(f,1.0,1.5,esp=1e-2)
print("Root is",round(root,6),",iteration time is",it)
```

```
## Root is 1.324219 ,iteration time is 6
```

2.2 不动点迭代法

不动点迭代法的 R 语言接口如下：

```
FixedPoint<-function(phi,init,esp=1e-3){
  ## 输入: phi 是迭代函数
  ## 输入: init 是初始值
  ## 输入: esp 是求根误差, 默认值为 1e-3
  if(!is.function(phi)) stop("phi should be a function")
  if(esp<=0) stop("need positive esp")

  # 在这里完成函数编写

  ## 输出: root 是求得的根
  ## 输出: iter 是迭代次数
  ## 若算法不收敛, 使用 stop("not converge!") 语句
  return(list(root=root,iter=iter))
}
```

不动点迭代法的 Python 接口如下：

```
def FixedPoint(phi,init,esp=1e-3):
  ## 输入: phi 是迭代函数
  ## 输入: init 是初始值
  ## 输入: esp 是求根误差, 默认值为 1e-3
  assert callable(phi), 'phi should be a function'
  assert esp>0, 'need positive esp'

  # 在这里完成函数编写

  ## 输出: root 是求得的根
  ## 输出: it 是迭代次数
  ## 若算法不收敛, 使用 assert ***,'not converge!'语句
```

```
  return root, it
```

2.3 斯蒂芬森迭代法

斯蒂芬森迭代法的 R 语言接口如下：

```
Steffensen<-function(phi,init,esp=1e-3){
  ## 输入: phi 是未校正的迭代函数
  ## 输入: init 是初始值
  ## 输入: esp 是求根误差, 默认值为 1e-3
  if(!is.function(phi)) stop("phi should be a function")
  if(esp<=0) stop("need positive esp")

  # 在这里完成函数编写

  ## 输出: root 是求得的根
  ## 输出: iter 是迭代次数
  ## 若算法不收敛, 使用 stop("not converge!") 语句
  return(list(root=root,iter=iter))
}
```

斯蒂芬森迭代法的 Python 接口如下：

```
def Steffensen(phi,init,esp=1e-3):
  ## 输入: phi 是未校正的迭代函数
  ## 输入: init 是初始值
  ## 输入: esp 是求根误差, 默认值为 1e-3
  assert callable(phi), 'phi should be a function'
  assert esp>0, 'need positive esp'

  # 在这里完成函数编写

  ## 输出: root 是求得的根
  ## 输出: it 是迭代次数
  ## 若算法不收敛, 使用 assert ***,'not converge!'语句
  return root, it
```

2.4 牛顿法

牛顿法比较重要，建议大家认真仔细编写。

牛顿法的 R 语言接口如下：

```r
Newton<-function(f,init,method=c("normal","simplified"),down_hill=FALSE,esp=1e-3){
  ## 输入：f 是迭代函数，我们希望求 f(x)=0 的根
  ## 输入：init 是初始值
  ## 输入：method 是方法：normal 表示用一般的牛顿法，simplified 表示用简化的牛顿法
  ## 输入：down_hill 表示是否使用下山法
  ## 输入：esp 是求根误差，默认值为 1e-3
  if(!is.function(f)) stop("f should be a function")
  if(esp<=0) stop("need positive esp")

  # 在这里完成函数编写

  ## 输出：root 是求得的根
  ## 输出：iter 是迭代次数
  ## 若算法不收敛，使用 stop("not converge!") 语句
  return(list(root=root,iter=iter))
}
```

牛顿法的 Python 接口如下：

```python
def Newton(f,init,method='normal',down_hill=False,esp=1e-3):
  ## 输入：f 是迭代函数，我们希望求 f(x)=0 的根
  ## 输入：init 是初始值
  ## 输入：method 是方法：normal 表示用一般的牛顿法，simplified 表示用简化的牛顿法
  ## 输入：down_hill 表示是否使用下山法
  ## 输入：esp 是求根误差，默认值为 1e-3
  assert callable(f), 'f should be a function'
  assert method in ['normal','simplified'], 'method should be normal or simplified'
  assert esp>0, 'need positive esp'

  # 在这里完成函数编写

  ## 输出：root 是求得的根
  ## 输出：it 是迭代次数
  ## 若算法不收敛，使用 assert ***,'not converge!'语句
  return root, it
```

3 为什么弦截法超线性收敛？

牛顿法的迭代公式是：$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$。但在部分情况下，函数 $f(.)$ 可能过于复杂，导致求导十分困难。因此我们可以用弦截法来简化导数的计算。弦截法则用割线来近似切线，即令 $f'(x_k) \approx \frac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}$。因此弦截法有迭代公式：

$$x_{k+1} = x_k - \frac{f(x_k)}{f(x_k)-f(x_{k-1})}(x_k - x_{k-1})$$

牛顿法的收敛速度比较快，达到了二阶收敛。在将导数近似后，弦截法的收敛速度有所下降，其按阶 $p \approx 1.618$ 收敛，仍为超线性收敛。下面我们证明其收敛速度。

引理：设 $f \in C^2[a,b]$，$x, x_n, x_{n-1} \in [a,b]$，则存在 $\xi \in [a,b]$ 使得：

$$\frac{f(x)-f(x_n)}{x-x_n} - \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}} = \frac{f''(\xi)}{2}(x-x_{n-1})$$

Proof: 考虑 $f(p)$ 在 $p = x_n$ 处的泰勒展开：

$$f(p) = f(x_n) + f'(x_n)(p-x_n) + \frac{f''(\eta)}{2}(p-x_n)^2$$

将 $p = x, p = x_{n-1}$ 其代入，有：

$$f(x) = f(x_n) + f'(x_n)(x-x_n) + \frac{f''(\eta_1)}{2}(x-x_n)^2$$

$$f(x_{n-1}) = f(x_n) + f'(x_n)(x_{n-1}-x_n) + \frac{f''(\eta_2)}{2}(x_{n-1}-x_n)^2$$

将两式代入证明式的左边有：

$$\begin{aligned}\frac{f(x)-f(x_n)}{x-x_n} - \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}} &= f'(x_n) + \frac{f''(\eta_1)}{2}(x-x_n) - f'(x_n) - \frac{f''(\eta_2)}{2}(x_{n-1}-x_n) \\ &= \frac{f''(\eta_1)(x-x_n) + f''(\eta_2)(x_n-x_{n-1})}{2}\end{aligned}$$

又 $\frac{f''(\eta_1)(x-x_n)+f''(\eta_2)(x_n-x_{n-1})}{(x-x_n)+(x_n-x_{n-1})}$ 介于 $f''(\eta_1)$ 和 $f''(\eta_2)$ 之间。使用介值定理，存在 ξ，使得 $\frac{f''(\eta_1)(x-x_n)+f''(\eta_2)(x_n-x_{n-1})}{(x-x_n)+(x_n-x_{n-1})} = f''(\xi)$。

将其带回原式即有：

$$\frac{f(x)-f(x_n)}{x-x_n} - \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}} = \frac{f''(\xi)}{2}(x-x_{n-1})$$

得证。

事实上，我们在第二章学习过差商的知识，我们可以用差商快速证明这一引理。

Proof:

$$\frac{f(x)-f(x_n)}{x-x_n} - \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}} = f[x,x_n] - f[x_n,x_{n-1}] = f[x,x_n,x_{n-1}](x-x_{n-1}) = \frac{f''(\xi)}{2}(x-x_{n-1})$$

得证。

定理：假设 $f(x)$ 在根 x^* 的领域 $\Delta : |x - x^*| \leq \delta$ 内具有二阶导数，且对任意 $x \in \Delta$ 有 $f'(x) \neq 0$，又初值 $x_0, x_1 \in \Delta$，那么当 Δ 充分小时，两点弦截法超线性收敛，且按阶 $p = \frac{1+\sqrt{5}}{2} \approx 1.618$ 收敛到 x^*。

Proof: 记 $f'_D(x_n) = \frac{f(x_n)-f(x_{n-1})}{x_n - x_{n-1}}$。将 $f(x)$ 在 x_n 处展开成以下式子：

$$f(x) = f(x_n) + f'_D(x_n)(x - x_n) + R_n$$

将 x^* 代入，则有 $0 = f(x^*) = f(x_n) + f'_D(x_n)(x^* - x_n) + R_n$，所以：

$$R_n = f(x^*) - f(x_n) - f'_D(x_n)(x^* - x_n) = [\frac{f(x^*) - f(x_n)}{x^* - x_n} - \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}](x^* - x_n)$$

应用引理，我们有 $R_n = \frac{f''(\xi)}{2}(x^* - x_{n-1})(x^* - x_n)$，代回即有：

$$0 = f(x^*) = f(x_n) + f'_D(x_n)(x^* - x_n) + \frac{f''(\xi)}{2}(x^* - x_{n-1})(x^* - x_n)$$

除以 $f'_D(x_n)$，我们有：

$$x_n - x^* - \frac{f(x_n)}{f'_D(x_n)} = \frac{f''(\xi)}{2f'_D(x_n)}(x^* - x_{n-1})(x^* - x_n)$$

注意到 $x_n - x^* - \frac{f(x_n)}{f'_D(x_n)} = x_{n+1}$，对等号两侧取绝对值有：

$$|x_{n+1} - x^*| = |\frac{f''(\xi)}{2f'_D(x_n)}||x^* - x_{n-1}||x^* - x_n|$$

$$\Rightarrow e_{n+1} = |\frac{f''(\xi)}{2f'_D(x_n)}|e_n e_{n-1}$$

$$\Rightarrow \lim_{n\to\infty} \frac{e_{n+1}}{e_n e_{n-1}} = |\frac{f''(\xi)}{2f'_D(x_n)}| = M < \infty$$

所以，弦截法超线性收敛。下面我们来说明为什么 $p \approx 1.618$，不过我们这里只给出一个不严格的说明。设弦截法的收敛阶数为 p，则 $\lim_{n\to\infty} \frac{e_{n+1}}{e_n^p} = \lambda < \infty$。所以：

$$\begin{aligned}
\lambda^p &= \lambda \cdot \lambda^{p-1} \\
&= \lim_{n\to\infty} \frac{e_{n+1}}{e_n^p} \lim_{n\to\infty} \frac{e_n^{p-1}}{e_{n-1}^{p(p-1)}} \\
&= \lim_{n\to\infty} \frac{e_{n+1}}{e_n e_{n-1}^{p(p-1)}} < \infty
\end{aligned}$$

由 $\lim_{n\to\infty} \frac{e_{n+1}}{e_n e_{n-1}}$ 收敛，我们知道 $p(p-1) \geq 1$，因此 $p \geq \frac{1+\sqrt{5}}{2}$。即弦截法收敛阶至少为 $\frac{1+\sqrt{5}}{2}$。

4 解非线性方程组和优化的关系

4.1 一个简单的例子

我们考虑一个简单的线性回归的例子：$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} + \epsilon_i, i = 1, 2, ..., n$。从回归分析的课程中，我们知道 β 的估计是：

$$\hat{\beta} = \min_{\beta_0, \beta_1, ..., \beta_p} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - ... - \beta_p x_{ip})^2$$

因此，回归分析问题实际上就是一个优化问题。其优化对象是函数 $f(\beta_0, \beta_1, ..., \beta_p) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - ... - \beta_p x_{ip})^2$，我们希望最小化这个函数，即 $\min_{\beta_0, \beta_1, ..., \beta_p} f(\beta_0, \beta_1, ..., \beta_p)$。

更一般的优化问题也具有如此形式：$\min_{x \in R^n} f(x)$。

对于这样一个优化问题，我们普遍的做法求 $f(x)$ 的一阶导数，并令其一阶导数等于 0，随后解一个方程组。因此：

$$\min_{x \in R^n} f(x) \iff \begin{cases} \frac{\partial f(x)}{\partial x_1} = 0 \\ \frac{\partial f(x)}{\partial x_2} = 0 \\ \quad ... \\ \frac{\partial f(x)}{\partial x_n} = 0 \end{cases}$$

即解一个（非线性）方程组问题和优化问题实际上是同一个问题。因此很多经典的优化算法都和解非线性方程组的求解算法有着千丝万缕的关系。下面我们来着重讨论优化问题，因为优化问题在很多统计或者机器学习的领域中有着更为广阔的应用。

当然，如果我们的问题像线性回归一样简单（可以直接写出显式解），那就直接万事大吉了。但是，万事大吉的情况显然不会很多。大多数情况下，优化问题往往是没有显式解的，我们还是需要通过一些迭代方法来求解该问题。目前有两种非常具有典型性的问题：

1. 需要优化的函数十分复杂，但输入维数不是很高。
2. 需要优化的函数不复杂，但输入的维数相当高。

我们下面主要讨论第 2 类问题（深度学习的优化就是这类问题的典型代表）。我们目前需要做以下优化问题：$\min_{x \in R^n} f(x)$，其中 n 非常大，但 $f(.)$ 形式较为简单，求各阶导数也十分容易。

下面我们来主要来说两种算法：牛顿法和梯度下降法。

4.2 牛顿法与梯度下降法

我们很容易想到：$\min_{x \in R^n} f(x) \iff \frac{\partial f(x)}{\partial x} = 0$。因此，我们可以用方程求根的牛顿法来求解优化问题。

记得分向量 $U(x) = \frac{\partial f(x)}{\partial x}$，黑塞矩阵 $H(x) = [\frac{\partial^2 f(x)}{\partial x_i \partial x_j}]_{n \times n}$。

使用牛顿法（有时候也被称作 Newton-Raphson 方法），即有迭代公式：$x_{k+1} = x_k - H^{-1}(x_k)U(x_k)$。因此，牛顿法的步骤可如下表示：

算法：牛顿法

输入：目标函数 $f(x)$，精度要求 ϵ

Step 1：计算 $f(x)$ 的一阶导数 $U(x)$ 和二阶导数 $H(x)$。

Step 2：取初始点 x_0。

Step 3：计算 $U_k = U(x_k), H_k = H(x_k)$

Step 4：若 $||U_k|| < \epsilon$，则停止计算，得到近似解 $x^* = x_k$，否则计算 $x_{k+1} = x_k - H_k^{-1}U_k$ 并转 Step 3。

牛顿法的优点是二阶收敛，收敛速度较快。但是其缺点在高维时也十分明显，由于牛顿法需要算黑塞矩阵的逆矩阵，对于十分大的 n，求逆矩阵的运算量是十分巨大的。因此，在深度学习等超高维的优化问题中，我们往往不适用牛顿法进行求解。我们使用的是梯度下降法及其变形。

梯度下降法也是求解如下问题：$\min\limits_{x \in R^n} f(x)$。与牛顿法不同的是，梯度下降法只用到目标函数的一阶导数信息，而不使用目标函数的二阶导数信息。我们知道，多元函数的梯度方向是函数值上升最快的方向，因此梯度的反方向就是函数值下降最快的方向。梯度下降法的思想就基于此，每次函数都朝着函数下降最快的方向迈进。这样，我们有理由相信，在最后我们可以达到函数的最小值点。

不过，我们还需解决一个问题，就是每次朝着函数下降最快的方向迈进多少。事实上，我们有很多种选择方法。其中一种方法是贪心算法，即每一次要迈向这个方向上的最低点；还有一种方法是根据问题的种类，人为设定一个固定的迈步大小，我们在深度学习中常常使用这一方法，并将人为设定的迈步大小称为学习率。因此，梯度下降法的迭代公式为：$x_{k+1} = x_k - \lambda_k U(x_k)$，其中 λ_k 就是迈步大小。使用贪心算法时有：$\lambda_k = \overset{argmin}{\lambda} f(x_k + \lambda U(x_k))$，设置学习率时则有：$\lambda_k = \lambda$。梯度下降法的步骤如下：

算法：梯度下降法

输入：目标函数 $f(x)$，精度要求 ϵ

Step 1：计算 $f(x)$ 的一阶导数 $U(x)$。

Step 2：取初始点 x_0。

Step 3：计算 $U_k = U(x_k)$，并按照设置计算 λ_k

Step 4：若 $||U_k|| < \epsilon$，则停止计算，得到近似解 $x^* = x_k$，否则计算 $x_{k+1} = x_k - \lambda_k U_k$ 并转 Step 3。

梯度下降法是一种线性收敛的方法，但由于其在大规模问题中不需要求逆，减少了计算量，因此在深度学习等领域中更为常用。

关于梯度下降法和牛顿法的比较可见下表：

	梯度下降法	牛顿法
收敛速度	一阶/线性收敛	二阶收敛
是否需要二阶导数	否	是
是否需要额外设定参数	需要设置学习率	否
是否能收敛到全局最小值	凸函数时可以，其余不能保证	不能

4.3 随机梯度下降法

前面我们看到，在高维的情况下，梯度下降法虽然收敛速度慢，但是由于较少的计算量，仍然可以较快的得到优化问题的解。但是，现如今的问题不仅仅是数据的维数高，数据的容量也非常大。因此，在实际领域中，为了解决数据容量过大所带来的计算缓慢的问题，我们往往用随机梯度下降法（**S**tochastic **G**radient **D**escent, SGD）来代替梯度下降法。SGD 也是目前深度学习最主要的训练和优化方法。

回到回归分析的优化目标函数：

$$f(\beta)=\sum_{i=1}^{n}(y_i-\beta_0-\beta_1x_{i1}-...-\beta_px_{ip})^2:=\sum_{i=1}^{n}f_i(\beta)$$

因此，当 n 很大时，我们每次在计算梯度时，都需要将 n 个梯度（$\frac{\partial f_i(\beta)}{\partial \beta},i=1,2,...,n$）相加，较为费时。因此，我们往往从 n 个样本取出一定数量的样本（如 16、32、64 个）构成一个批次（batch）。用这个批次的梯度来近似整个数据集的梯度。

但是由于每次我们是从数据集中随机抽出一定数量的样本，因此此时的梯度方向的反方向不一定是函数下降的最快方向了，我们在优化时会用一些技巧来加速优化的速度，其中常用的方法有：

- SGD+Momentum
- Adagrad 和 RMSProp
- Adam

这些方法都是深度学习中常用的优化方法，具体内容可点击**这里**。

5 拟牛顿法

前面，我们看到，由于牛顿法中求逆过程的时间复杂度较高。我们使用了线性收敛的梯度下降法来代替牛顿算法来处理高维问题。事实上，这是一个收敛速度和计算复杂度的 trade-off。我们能不能想出一个介于两个方法之间的方法，使该方法既有着接近牛顿法的收敛速度，又有着接近梯度下降法的计算复杂度呢？我们的思想很简单，我们仍然用牛顿法的迭代公式进行优化，但我们在计算逆矩阵时需要进行简化运算，减少复杂度。这样的方法至少有：

- **Fisher-scoring 算法**：大家熟悉的 logistic 回归就使用了此方法来代替牛顿法。其思想是用**黑塞矩阵的期望**来代替黑塞矩阵来进行计算。大部分广义线性模型都使用这一方法来加速牛顿法，因为在广义线性模型下，黑塞矩阵的期望是有显式表达式的，十分容易计算。
- **拟牛顿法（Quasi-Newton Method）**：我们使用一个 n 阶矩阵 $G_k = G(x_k)$ 来近似代替 $H_k^{-1} = H^{-1}(x_k)$。

下面我们主要来讨论拟牛顿法。我们首先查看牛顿法中黑塞矩阵 H_k 满足的条件，使用中值定理，我们可得：

$$U(x_{k+1}) - U(x_k) = H(x_k + \delta_x)(x_{k+1} - x_k) \approx H_k(x_{k+1} - x_k)$$

记 $y_k = U(x_{k+1}) - U(x_k), \delta_k = x_{k+1} - x_k$，则:

$$y_k = H_k\delta_k \iff H_k^{-1}y_k = \delta_k$$

以上两式被称为**拟牛顿条件**。拟牛顿法用 G_k 来近似代替 H_k^{-1}，要求 G_k 满足同样的条件，即 $G_k y_k = \delta_k$。每次迭代过程中，我们只需更新 G_k 即可，往往我们令 $G_{k+1} = G_k + \Delta G_k$。选择 G_k 的方法有很多，下面介绍几种常用的算法。

5.1 DFP 算法 (Davidon-Fletcher-Powell algo.)

DFP 算法通过两个附加项来更新矩阵 G_k:

$$G_{k+1} = G_k + P_k + Q_k$$

其中，P_k, Q_k 是待定矩阵。此时我们在两边同时乘上 y_k，我们有：

$$G_{k+1}y_k = G_k y_k + P_k y_k + Q_k y_k$$

为使 G_{k+1} 满足拟牛顿条件，我们可令：

$$P_k y_k = \delta_k, Q_k y_k = -G_k y_k$$

事实上，这样的 P_k, Q_k 存在很多，我们可以取：

$$P_k = \frac{\delta_k \delta_k^T}{\delta_k y_k}$$

$$Q_k = -\frac{G_k y_k y_k^T G_k}{y_k^T G_k y_k}$$

这样我们就可以得到 DFP 算法的迭代公式：

$$G_{k+1} = G_k + \frac{\delta_k \delta_k^T}{\delta_k y_k} - \frac{G_k y_k y_k^T G_k}{y_k^T G_k y_k}$$

$$x_{k+1} = x_k - G_k U_k$$

可以证明，如果初始矩阵 G_0 是正定的，那么迭代过程中，每一个 G_k 都是正定的，这可以保证每次 DFP 算法都是搜索一个下降的方向。

5.2 BFGS 算法 (Broyden-Fletcher-Goldfarb-Shanno algo.)

BFGS 是目前最为流行的拟牛顿算法。

DFP 算法使用 G_k 来近似代替 H_k^{-1}，而 BFGS 算法则是用 B_k 来近似代替黑塞矩阵 H_k。此时，B_k 应该满足 $B_k\delta_k = y_k$。同样，我们可对 B_{k+1} 进行分解：

$$B_{k+1} = B_k + P_k + Q_k$$

$$B_{k+1}\delta_k = B_k\delta_k + P_k\delta_k + Q_k\delta_k$$

因此，考虑：

$$P_k\delta_k = y_k, Q_k\delta_k = -B_k\delta_k$$

我们可以取适当的 P_k, Q_k 得到迭代公式：

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T\delta_k} - \frac{B_k\delta_k\delta_k^T B_k}{\delta_k B_k\delta_k}$$

$$x_{k+1} = x_k - B_k^{-1}U_k$$

注意，此时 B_k^{-1} 有显式迭代公式：

$$B_{k+1}^{-1} = (I - \frac{\delta_k y_k^T}{y_k^T\delta_k})B_k^{-1}(I - \frac{y_k\delta_k^T}{y_k^T\delta_k}) + \frac{\delta_k\delta_k^T}{y_k^T\delta_k}$$

同样可以证明，如果初始矩阵 B_0 是正定的，那么迭代过程中，每一个 B_k 都是正定的，这可以保证每次 DFP 算法都是搜索一个下降的方向。

5.3 Broyden 类算法

Broyden 类综合了 DFP 算法和 BFGS 算法。记 $G^{DFP} = G_k, G^{BFGS} = B_k^{-1}$，Broyden 类使用了两者的线性组合：

$$G_{k+1} = \alpha G^{DFP} + (1-\alpha)G^{BFGS}, 0 \le \alpha \le 1$$

此时的 G_k 仍然是正定的。

6 参考

1. Locietta. 02R03 弦截法. 知乎.
2. 李航. 统计学习方法（第 2 版）. 附录 A& 附录 B. 清华大学出版社.
3. Quasi-Newton method. Wikipedia.