高斯牛顿法与 LM 算法极值求解

一、一元函数二阶泰勒展开

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2!}f''(x_0)(x - x_0)^2$$

令x看成$x + \Delta$，则有

$$f(x + \Delta) \approx f(x) + f'(x)\Delta + \frac{1}{2!}f''(x)\Delta^2$$

我们要要求$f(x)$函数最小值时 x的值，即

$$x = \arg\min f(x)$$

利用数学定理的方法则是先求$f'(x) = 0$时x的值，然后判断$f''(x) \leq 0$，若是，则所得x的值对应的$f(x)$最小。

二、高斯牛顿法

现有函数$f(x) = x^2$，求出函数$f(x)$最小值时x的值。即

$$x = \arg\min f(x)$$

$$f'(x) = 2x$$

令x_n为高斯牛顿法第n次的迭代解，有如下迭代公式：

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

以上x是一维的，但实际中遇到的都是多维的，这时可以用雅克比、海赛矩阵之类的来表示高维求导函数。

雅可比矩阵：

$$J_f = \begin{bmatrix} \frac{\partial f}{\partial x_0} & \cdots & \frac{\partial f}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_0} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix}$$

海赛矩阵：

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial {x_0}^2} & \frac{\partial^2 f}{\partial x_0 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_0 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_0} & \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial {x_n}^2} \end{bmatrix}$$

所以高维牛顿法解最优化问题又可写成：

$$X_{n+1} = X_n - H_f(X_n)^{-1} \nabla f(X_n)$$

在这里，梯度代替了低维情况中的一阶导，海赛矩阵代替了二阶导，求逆代替了除法。

例：不妨设目标函数为

$$s(x) = \sum_{i=0}^{n} f^2(x_i)$$

所以梯度向量在方向上的分量：

$$g_j = 2 \sum_{i=0}^{n} f(x_i) \cdot \frac{\partial f(x_i)}{\partial x_j}$$

海赛矩阵的元素则直接在梯度向量的基础上求导：

$$H_{jk} = 2 \sum_{i=0}^{n} \left(\frac{\partial f(x_i)}{\partial x_j} \cdot \frac{\partial f(x_i)}{\partial x_k} + f(x_i) \cdot \frac{\partial f(x_i)}{\partial x_j \partial x_k} \right)$$

高斯牛顿法的一个技巧是，将二次偏导省略，于是：

$$H_{jk} \approx \sum_{i=0}^{n} J_{ij} J_{ik}$$

其中J_{ij}为雅可比矩阵中底i行j列元素。

改写成矩阵形式有：

$$g = 2{J_j}^T \cdot f$$

$$H \approx 2{J_f}^T J_f$$

代入牛顿法高维迭代方程的基本形式，得到高斯牛顿法迭代方程：

$$X^{s+1} = X^s - \left({J_f}^T J_f \right)^{-1} {J_f}^T f$$

三、Levenberg-Marquardt 算法

莱文贝格－马夸特方法（Levenberg－Marquardt algorithm）能提供数非线性最小化（局部最小）的数值解。此算法能借由执行时修改参数达到结合高

斯-牛顿算法以及梯度下降法的优点，并对两者之不足作改善（比如高斯-牛顿算法之反矩阵不存在或是初始值离局部极小值太远[1]。

与高斯牛顿法相比，LM 算法的迭代公式如下：

$$X^{s+1} = X^s - \left(J_f{}^T J_f + \lambda I\right)^{-1} J_f{}^T f$$

其中λ为可变参数，I为单位矩阵。

- 如果下降太快，则使用较小的λ，使步长缩短，这样更接近高斯牛顿法；
- 如果下降太慢，则使用较大的λ，使步长增大，这样更接近梯度下降法。

四、求函数最小值

Question 1: 求函数$f(x) = x^2$ 的最小值时x的取值

$$J_f = 2x$$

$$H_f = 2$$

则 LM 算法的更新公式为：

$$x_{n+1} = x_n - \left(J_f{}^T J_f + \lambda I\right)^{-1} J_f{}^T f = x_n - [(2x_n)^T 2x_n + \lambda I]^{-1} \cdot (2x_n)^T \cdot {x_n}^2$$

由于是一维的x，因此I为1×1的单位矩阵，化简为：

$$x_{n+1} = x_n - \frac{2{x_n}^3}{4{x_n}^2 + \lambda}$$

取初值$x_0 = 10$，初始$\lambda = 1$，对上式进行 10 次迭代：

$iter$	x	f	λ	梯度∇
0	10	100	1	4.987531172
1	5.012468828	25.12484375	1	2.481542297
2	2.530926531	6.405589104	1	1.217929409
3	1.312997122	1.723961442	1	0.573353755
4	0.739643366	0.547072309	1	0.253827915
5	0.485815452	0.236016653	1	0.117959473
6	0.367855978	0.135318021	1	0.064592804
7	0.303263174	0.091968553	1	0.040779591
8	0.262483583	0.068897632	1	0.028354706
9	0.234128877	0.054816331	1	0.021052163
10	0.213076714	0.045401686	1	0.016374385

实际上，在算法运行时为了减小迭代次数，让算法尽快收敛，λ是可变的。变化规则如下：

while:

$J_f{}^T J_f + \lambda I$为非正定的时候，$\lambda \coloneqq 4\lambda$

计算步长 $\left(J_f{}^T J_f + \lambda I\right)^{-1} J_f{}^T f$，

if $0 < \left(J_f{}^T J_f + \lambda I\right)^{-1} J_f{}^T f < 0.25$：

在下一步迭代时增大步长，$\lambda \coloneqq 4\lambda$

if $0.25 \leq \left(J_f{}^T J_f + \lambda I\right)^{-1} J_f{}^T f \leq 0.75$：

步长合适，在下一步迭代时λ不变，$\lambda \coloneqq \lambda$

if $\left(J_f{}^T J_f + \lambda I\right)^{-1} J_f{}^T f > 0.75$：

在下一步迭代时缩短步长以避免越过最优点，$\lambda \coloneqq \lambda/2$

除此外，还需要设定算法停止的规则：

if $\left(J_f{}^T J_f + \lambda I\right)^{-1} J_f{}^T f < 0$：

break，算法停止，函数值是向着上升而非下降的趋势变化了（与最优化的目标相反），说明已经越过最优点了。

if $|x_{n+1} - x_n| < error$：

break，已经达到相应的误差精度，算法收敛，没必要继续跌倒。其中 error 为自定义参数。

Question 2: 求函数$f(x,y)=(1-x)^2+100(y-x^2)^2$ 的最小值时x,y的取值

雅可比矩阵：

$$J_f=\begin{bmatrix}\frac{\partial f}{\partial x} & \frac{\partial f}{\partial y}\end{bmatrix}=[2(x-1)-400x(y-x^2) \qquad 200(y-x^2)\,]$$

海赛矩阵：

$$H_f=\begin{bmatrix}\frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial y\partial x}\\ \frac{\partial^2 f}{\partial x\partial y} & \frac{\partial^2 f}{\partial y^2}\end{bmatrix}\approx {J_f}^T J_f$$

$$=\begin{bmatrix}[2(x-1)-400x(y-x^2)]^2 & [2(x-1)-400x(y-x^2)]\cdot 200(y-x^2)\\ [2(x-1)-400x(y-x^2)]\cdot 200(y-x^2) & [200(y-x^2)]^2\end{bmatrix}$$

迭代公式为：

$$[x_{n+1},y_{n+1}]=[x_n,y_n]-\left({J_f}^T J_f+\lambda I\right)^{-1}{J_f}^T f$$

取初值$x_0=5$，$y_0=5$，初始$\lambda=1$，对上式进行 10 次迭代：

$iter$	x	y	f
0	5	5	40016
1	4.009699092	5.099010305	12062.19206
2	3.335389268	5.183066412	3535.899476
3	2.899393254	5.248387015	1000.96361
4	2.6342536	5.294062941	273.3486603
5	2.482324615	5.322845872	72.60442607
6	2.39885261	5.33959953	19.17051773
7	2.352999249	5.349090187	5.346804986
8	2.324414316	5.355072611	1.982837507
9	2.284015946	5.363274105	3.796248366
10	2.311551756	5.357128285	1.739368996

理论上$f(x,y)$最小值时，$(x,y)=(1,1)$。从上面迭代结果只是刚开始的迭代时，y值不降反增，但随着迭代次数的不断增大，y也会逐渐降低并收敛到 1。

Note：LM 算法是在回归中参数拟合的常用方法之一，其目标函数一半为误差损失最小：

$$\underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{m} \left(y_i - f(x_i;\theta)\right)^2$$

其中，θ为需要拟合到参数；

m为样本数量；

y_i为第 i个样本的期望输出；

x_i为第 i个样本的输入；

$f(x_i;\theta)$为模型函数。

参考文献

[1]https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm

[2]http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3215/pdf/imm3215.pdf