前置知识

1. 牛顿法

作用：1. 求根 2.求极值

1. 求根

   目标: 求解 $f(y)=0$ 的根

   计算穿过初始点(x_0,f(x_0)) 并且斜率为 f'(x) 的直线与x轴的交点可得

   $$0=(x-x_0)f'(x_0)+f(x_0)$$

   迭代公式：

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

2. 求解一维无约束最小值

   目标: 求解 $min f(x), x\in R$ 的根

   牛顿法也可用来求解函数的极值。极值点是导数为0，可用牛顿法求导数的零点。

   $f(x+\Delta)$ 的二阶泰勒展开为

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

求解

$$\frac{\partial f(x+\Delta)}{\partial\Delta}=0$$

可得

$$f'(x)+f''(x)\Delta=0$$
$$\Delta=-\frac{f'(x_n)}{f''(x_n)}$$

迭代公式：

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

3. 求解高维无约束最小值

   高维情况下泰勒二阶展开为

$$f(\mathbf{x}+\boldsymbol{\Delta}) = f(\mathbf{x}) + \nabla f(\mathbf{x})\Delta + \frac{1}{2}\Delta^T H(f(\mathbf{x}))\Delta$$

   因此迭代公式：

$$x_{n+1} = x_n - [H(f(x_n))]^{-1}\nabla f(x)$$

优点

- 牛顿法是二阶收敛，比一般梯度下降法更快收敛，特别是当初始点距离目标足够靠近。

缺点

- 应用求极值的时候需要目标函数二次可微，而梯度下降法只需要可微
- 需要 $Hessian$ 矩阵正定，遇到 f 的极值点，或者初始点距离目标远时候可能无法收敛
- 每次迭代需要求 $Hessian$ 的逆矩阵，运算量非常大

2. 高斯–牛顿法

作用：降低牛顿法的计算量，提高计算效率

最小二乘法问题

对于向量函数 $\mathbf{f}: R^m \to R^n, m \geq n$

最小化 $||f(x)||$ 或者找到 $x^* = argmin_x\{F(x)\}$

这里 $F(x) = \frac{1}{2}\sum_{i=1}^m (f_i(x))^2 = \frac{1}{2}\mathbf{f}(\mathbf{x})^T\mathbf{f}(\mathbf{x})$

牛顿法推导

已知牛顿法迭代公式： $x_{n+1} = x_n - [H(f(x_n))]^{-1}\nabla f(x)$

$F(x)$ 的梯度

$$\nabla F(x)=\begin{bmatrix}\frac{\partial(\frac{1}{2}\sum_{i=1}^{m}(f_i(x))^2)}{\partial x_1}\\ \vdots\\ \frac{\partial(\frac{1}{2}\sum_{i=1}^{m}(f_i(x))^2)}{\partial x_n}\end{bmatrix}=\begin{bmatrix}\sum_{i=1}^{m}f_i\frac{\partial f_i}{\partial x_1}\\ \vdots\\ \sum_{i=1}^{m}f_i\frac{\partial f_i}{\partial x_n}\end{bmatrix}=\begin{bmatrix}\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n}\\ \vdots & \ddots & \vdots\\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n}\end{bmatrix}^T\begin{bmatrix}f_1\\ \vdots\\ f_m\end{bmatrix}$$

即 $\nabla F(x)=J_f^T f$

$Hessian$ 矩阵有

$$H_{jk}=\sum_{i=1}^{m}(\frac{\partial f_i}{\partial x_j}\frac{\partial f_i}{\partial x_k}+f_i\frac{\partial^2 f_i}{\partial x_j\partial x_k})$$

忽略二阶导数项有： $H_{jk}\approx\sum_{i=1}^{m}J_{ij}J_{ik}$

所以： $H\approx J_f^T J_f$

高斯–牛顿迭代公式： $x_{n+1}=x_n-[J_f^T J_f]^{-1}J_f^T f(x_n)\ s.t.|\frac{\partial f_i}{\partial x_j}\frac{\partial f_i}{\partial x_k}|\gg|f_i\frac{\partial^2 f_i}{\partial x_j\partial x_k}|$

优点

- J_f满秩，此时二次项 $|f_i\frac{\partial^2 f_i}{\partial x_j\partial x_k}|$ 可以忽略，高斯牛顿和牛顿法都会收敛
- 无需计算 $Hessian$ 矩阵

缺点

- 若 $|fi|$ 或 $|f_i\frac{\partial^2 f_i}{\partial x_j\partial x_k}|$ 比较大，会导致难以忽略该二次项，高斯牛顿法的收敛速度会很慢，甚至无法收敛。

Levenberg–Maquardt 算法

根据目标函数$F(x)$ 二阶近似得到：

$$F(x+h)\approx F(x)+\nabla F(x)h+\frac{1}{2}h^T H_F h\approx F(x)+J_f^T fh+\frac{1}{2}h^T J_f^T J_f h$$

我们定义下降方向为 $L(h)$

$$
L(h) \equiv F(x) + J_f^T f h + \frac{1}{2} h^T J_f^T J_f h
$$
$$
s.t. \quad h = x_{n+1} - x_n = \Delta
$$

高斯牛顿法迭代公式：

$$
x_{n+1} = x_n - [J_f^T J_f]^{-1} J_f^T f(x_n)
$$

LM迭代公式： $x_{n+1} = x_n - [J_f^T J_f + \mu I]^{-1} J_f^T f(x_n)$
or $[J_f^T J_f + \mu I] h = -J_f^T f(x_n)$

作用：结合了高斯牛顿法与梯度下降法的特点，引入阻尼因子来调节算法特性。

因子作用：

- 当 $\mu > 0$ 时保证系数矩阵正定，从而确保迭代的下降方向
- 当 μ 很大时，退化为梯度下降法： $x_{n+1} = x_n - \frac{1}{\mu} J_f^T f(x_n)$
- 当 μ 很小时，退化为高斯牛顿法： $x_{n+1} = x_n - [J_f^T J_f]^{-1} J_f^T f(x_n)$

μ的计算

- 初始取值： μ 的初始值μ_0 与 $J(x_0)^T J(x_0)$ 矩阵的元素个数有关：

$$
\mu_0 = \tau * max_i \{a_{ii}^{(0)}\}
$$

- 更新： 由系数 ϱ 来控制，这里：

$$
\varrho = \frac{F(x) - F(x+h)}{L(0) - L(h)}
$$

分子的目标函数在步长 h 下的实际变化，分母为目标函数二阶近似的变化：

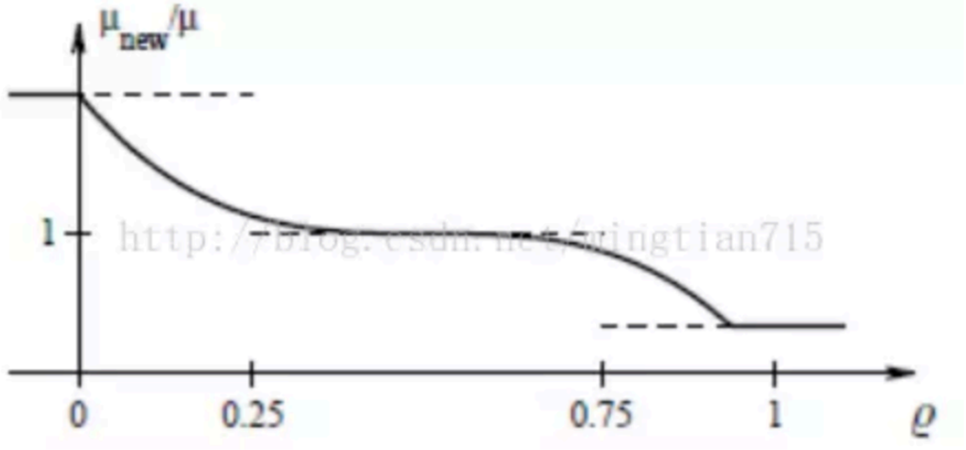
$$
L(0) - L(h) = (F(x)) - (F(x) + h^T J^T f + \frac{1}{2} h^T J^T J h) = -h^T J^T f - \frac{1}{2} h^T J^T J h
$$

可以看出

- ϱ 越大，表明 L 对 F 的效果越好，可以缩小 μ 以使得 LM 算法接近高斯牛顿法

- ϱ 越小，表明 L 对 F 的效果越差，所以增大 μ 以使得 LM 算法接近梯度下降法并减少步长 h

$$if\, \varrho > 0\, \mu = \mu * max\{\frac{1}{3}, 1-(2\varrho-1)^3\}$$
$$else\, \mu = \mu * v; v = 2$$



u 和 p 关系曲线图

LM算法流程图

Algorithm 3.16. Levenberg–Marquardt method

begin
$k := 0; \quad \nu := 2; \quad \mathbf{x} := \mathbf{x}_0$
$\mathbf{A} := \mathbf{J}(\mathbf{x})^\top \mathbf{J}(\mathbf{x}); \quad \mathbf{g} := \mathbf{J}(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$
$found := (\|\mathbf{g}\|_\infty \le \varepsilon_1); \quad \mu := \tau * \max\{a_{ii}\}$
while (**not** *found*) **and** $(k < k_{\max})$
$k := k+1; \quad$ Solve $(\mathbf{A} + \mu \mathbf{I})\mathbf{h}_{lm} = -\mathbf{g}$
if $\|\mathbf{h}_{lm}\| \le \varepsilon_2(\|\mathbf{x}\| + \varepsilon_2)$
found := **true**
else
$\mathbf{x}_{new} := \mathbf{x} + \mathbf{h}_{lm}$

$\varrho := (F(\mathbf{x}) - F(\mathbf{x}_{new}))/(L(\mathbf{0}) - L(\mathbf{h}_{lm}))$
if $\varrho > 0$ {step acceptable}
$\mathbf{x} := \mathbf{x}_{new}$
$\mathbf{A} := \mathbf{J}(\mathbf{x})^\top \mathbf{J}(\mathbf{x}); \quad \mathbf{g} := \mathbf{J}(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$
$found := (\|\mathbf{g}\|_\infty \le \varepsilon_1)$
$\mu := \mu * \max\{\frac{1}{3}, 1 - (2\varrho - 1)^3\}; \quad \nu := 2$
else
$\mu := \mu * \nu; \quad \nu := 2 * \nu$
end

LM算法流程图

Reference

- boksic 非线性优化整理-1.牛顿法 http://blog.csdn.net/boksic/article/details/79130509
- boksic 非线性优化整理-2.高斯牛顿法 https://blog.csdn.net/boksic/article/details/79055298
- boksic 非线性优化整理-3.Levenberg-Marquardt法(LM法)https://blog.csdn.net/boksic/article/details/79177055#
- Timmy_Y 训练数据常用算法之Levenberg–Marquardt（LM）https://blog.csdn.net/mingtian715/article/details/53579379
- Miroslav Balda 的Methods for non-linear least square problems http://download.csdn.net/detail/mingtian715/9708842