基本优化方法（现学现卖）

李显求

VIPL Group

April 10, 2016

提纲

- 主要内容
  - 凸问题
  - 泰勒展开式
  - 梯度下降法
  - 牛顿迭代法
  - Levenberg–Marquardt 算法
  - 拟牛顿法
    - BFGS 算法
    - L-BFGS 算法
  - 次梯度算法
  - Bregman Projection Problem 及解法
  - Lagrange 对偶问题
  - 几个例子
- Q&A

符号编辑页 1（相当于草稿）

- 凸集合 $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S$, **then** $\boldsymbol{x} = \lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2 \in S$
- 凸函数 $f(\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2) = f(\boldsymbol{x}_2 + \lambda(\boldsymbol{x}_1 - \boldsymbol{x}_2)) < \lambda f(\boldsymbol{x}_1) + (1-\lambda) f(\boldsymbol{x}_2), \lambda \in (0,1)$
- Bregman Divergence:$D_F(\boldsymbol{p}, \boldsymbol{q}) = F(\boldsymbol{p}) - F(\boldsymbol{q}) - \langle \nabla F(q), \boldsymbol{p} - \boldsymbol{q} \rangle$
- f 是一个线性函数和一个常数的和，也就是 $f(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{b}$ 的形式，其中 $A \in R^{m \times n}, \boldsymbol{b} \in R^m$，那么 $f : R^n \to R^m$ 是仿射的
- 逐点上确界 $g(\boldsymbol{x}) = \sup\{f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \cdots\}$
- $\mathcal{L}(x, \lambda) = f_1(x) + \lambda f_0(x)$
- $s(x) = \frac{\exp(x)}{1+\exp(x)} = \frac{1}{1+\exp(-x)}, s'(x) = s(x)(1 - s(x))$
- $\min |x|, t_i = \frac{1}{i+1}$
- $\mathrm{dist}^2_{x' \in S}(x', x), S$

$$\begin{cases} \min f_1(x) \\ f_0(x) < 0 \end{cases} \tag{1}$$

符号编辑页 2（相当于草稿）

- $D_{ld}(A, A_0) = \mathrm{tr}(AA_0^{-1}) - \log\det(AA_0^{-1}) - n$
- $F(X) = -\log|\det(X)|, \nabla_X F(X) = -(X^T)^{-1}$

$$\begin{aligned} &\min_{A\succeq 0} D_{ld}(A, A_0) + \gamma D_{ld}(\mathrm{diag}(\xi), \mathrm{diag}(\xi_0)) \\ s.t \quad &\mathrm{tr}(A(x_i - x_j)(x_i - x_j)^T) \le \xi_{ij}, (i,j) \in S \\ &\mathrm{tr}(A(x_i - x_j)(x_i - x_j)^T) \ge \xi_{ij}, (i,j) \in D \end{aligned} \tag{2}$$

$$\begin{aligned} &\min_{A\succeq 0} D_{ld}(A_{k+1}, A_k) + \gamma D_{ld}(\mathrm{diag}(\xi_{k+1}), \mathrm{diag}(\xi_k)) \\ s.t \quad &\delta_{ij}\mathrm{tr}(A_{k+1}(x_i - x_j)(x_i - x_j)^T) \le \delta_{ij}\xi_{ij} \\ &\delta_{ij} = \begin{cases} 1, & if \ (i,j) \in S \\ -1, & if \ (i,j) \in D \end{cases} \end{aligned} \tag{3}$$

符号编辑页 3（相当于草稿）

- $D_{ld}(A, A_0) = \mathrm{tr}(AA_0^{-1}) - \log\det(AA_0^{-1}) - n$
- $F(X) = -\log|\det(X)|, \nabla_X F(X) = -(X^T)^{-1}$

$$
\begin{aligned}
\mathcal{L}(A,\beta) &= D_{ld}(A, A_k) + \gamma D_{ld}(\mathrm{diag}(\xi_{k+1}), \mathrm{diag}(\xi_k)) \\
&\quad + \alpha(\delta_{ij}\mathrm{tr}(A(\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^T) - \delta_{ij}\xi_{ij}) \\
\nabla_A \mathcal{L}(A,\beta) &= (A_k^{-1})^T - (A^{-1})^T + \alpha\delta_{ij}C_{ij}^T \\
A &= (A_k^{-1} + \alpha\delta_{ij}C_{ij})^{-1} \\
&= A_k - \alpha\delta_{ij}\frac{A_k(\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^T A_k}{1 + \alpha\delta_{ij}(\boldsymbol{x}_i - \boldsymbol{x}_j)A_k(\boldsymbol{x}_i - \boldsymbol{x}_j)}
\end{aligned} \tag{4}
$$

$$
\begin{aligned}
D\log\det(AB)(A)[H] &= D\log\det(AB)(AB)[D(AB)(A)[H]] \\
&= D\log\det(AB)(AB)[HB] \\
&= \left\langle \nabla_{AB}\log\det(AB), (HB)^T \right\rangle \\
&= \left\langle ((AB)^{-1})^T, (HB)^T \right\rangle \\
&= \mathrm{tr}((A^{-1})^T(B^{-1})^T B^T H^T)
\end{aligned} \tag{5}
$$

符号编辑页 3（相当于草稿）

- $$\begin{aligned}
\nabla_{A_{k+1}} D_{ld}(A_{k+1}, A_0) &= \nabla_{A_k} D_{ld}(A_k, A_0) + \alpha\delta_{ij} C_{ij} \\
(A_0^{-1})^T - (A_{k+1}^{-1})^T &= (A_0^{-1})^T - (A_k^{-1})^T + \alpha\delta_{ij} C_{ij} \\
A_{k+1} &= (A_k^{-1} - \alpha\delta_{ij} C_{ij})^{-1} \\
&= A_k + \alpha\delta_{ij} \frac{A_k(\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^T A_k}{1 - \alpha\delta_{ij}(\boldsymbol{x}_i - \boldsymbol{x}_j) A_k (\boldsymbol{x}_i - \boldsymbol{x}_j)}
\end{aligned} \tag{6}$$

从泰勒展开说起

定义

设 $f(x)$ 是实数域上的无穷可微的函数，那么它在任意一点 x_0 泰勒展开式表示为[a]:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots \tag{7}$$

同样的对于无穷可微向量函数 $f(\boldsymbol{x})$ 以及初始点 $\boldsymbol{x_0}$ 有[b]:

$$f(\boldsymbol{x}) = f(\boldsymbol{x_0}) + \left(\frac{\partial f(\boldsymbol{x_0})}{\partial \boldsymbol{x}}\right)^T (\boldsymbol{x} - \boldsymbol{x_0}) + \frac{1}{2!}(\boldsymbol{x} - \boldsymbol{x_0})^T \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x_0}) + \cdots \tag{8}$$

[a] 这里只讨论优化问题故省略了关于收敛域的讨论；所以只需要知道它是 x_0 周围的一个小领域就可以了，优化时只要步长不太大一般不会违反这个准则

[b] 其中 $\boldsymbol{H}$ 矩阵就是通常意义的 Hessian 矩阵

梯度下降优化方法

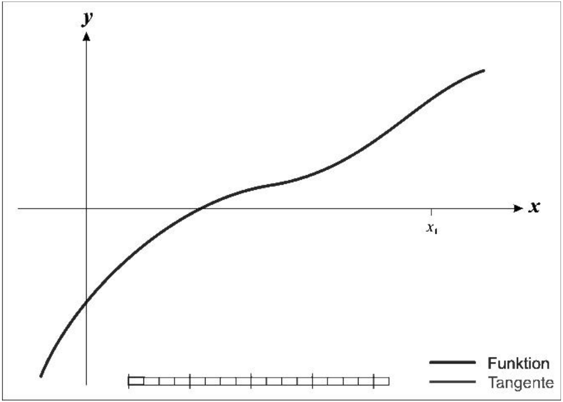
首先，梯度下降涉及两个参数：方向 $\boldsymbol{g}$ 和步长 α，为了方便起见这里不妨假设 $\|\boldsymbol{g}\| = 1$（虽然在实际中我们并不一定做归一化）。
最优化的目的就是使得 $f(\boldsymbol{x_0} + \alpha \boldsymbol{g})$ 最小，于是利用一阶泰勒展开我们得到：

$$f(\boldsymbol{x_0} + \alpha \boldsymbol{g}) \approx f(\boldsymbol{x_0}) + \alpha \left(\frac{\partial f(\boldsymbol{x_0})}{\partial \boldsymbol{x}} \right)^T \boldsymbol{g} \tag{9}$$

上式右边当 $\boldsymbol{g} = -\left(\frac{\partial f(\boldsymbol{x_0})}{\partial \boldsymbol{x}}\right) / \|\frac{\partial f(\boldsymbol{x_0})}{\partial \boldsymbol{x}}\|$ 的时候是 $f(\boldsymbol{x_0} + \alpha \boldsymbol{g})$ 较 $f(\boldsymbol{x_0})$ 下降最快的方向，所以梯度下法也叫最速下降法。
至于步长 α 的选择则是一个一维的优化问题，这个问题比较简单（二分法，0.618 法等都可以解决），当然也可以是预先定义的。
对于凸函数由于有：$f(\boldsymbol{x}) \geq f(\boldsymbol{x_0}) + \left(\frac{\partial f(\boldsymbol{x_0})}{\partial \boldsymbol{x}}\right)^T (\boldsymbol{x} - \boldsymbol{x_0})$，所以梯度下降算法可以保证目标函数值是不增加的

牛顿迭代法（1/3）

下面的动画显示了牛顿法在一维时的情况，这也相当于阐述了牛顿法的几何意义



在下一页中将使用数学来解释这种过程了，也就是牛顿法（牛顿迭代）的数学形式。

类似于梯度下降算法，首先给出 $f(x)$ 的二阶泰勒展开式[1]

$$f(\boldsymbol{x}) = f(\boldsymbol{x_0}) + \left(\frac{\partial f(\boldsymbol{x_0})}{\partial \boldsymbol{x}}\right)^T (\boldsymbol{x} - \boldsymbol{x_0}) + \frac{1}{2!}(\boldsymbol{x} - \boldsymbol{x_0})^T \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x_0}) + \cdots$$

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\partial f(\boldsymbol{x_0})}{\partial \boldsymbol{x}} + \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x_0}) + \cdots \tag{10}$$

不难发现的是过 $\boldsymbol{x_0}$ 点 $\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}$ 的" 切平面" 的就是：

$$\boldsymbol{y} = \frac{\partial f(\boldsymbol{x_0})}{\partial \boldsymbol{x}} + \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x_0}) \tag{11}$$

[1] 需要注意的是：牛顿法计算的是零点（或者说方程的根，因此，其实牛顿法是通过极值点处的梯度为零将两者联系起来

在公式 $\boldsymbol{y} = \frac{\partial f(\boldsymbol{x_0})}{\partial \boldsymbol{x}} + \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x_0})$ 中，令 $\boldsymbol{y} = \boldsymbol{0}$ 可以得到：

$$\boldsymbol{x} = \boldsymbol{x_0} - \boldsymbol{H}^{-1}\frac{\partial f(\boldsymbol{x_0})}{\partial \boldsymbol{x}}$$
$$or \tag{12}$$
$$\boldsymbol{x_{k+1}} = \boldsymbol{x_k} - \boldsymbol{H}^{-1}\frac{\partial f(\boldsymbol{x_k})}{\partial \boldsymbol{x}}$$

在最后，这里在举一个例子用来说明牛顿法的应用（求一个实数 a 的 n 次方根）：

- 首先写出方程：$f(x) = x^n - a$，则 $f(x)$ 的零点即为所求。
- 根据牛顿迭代法得到更新公式：$x_{k+1} = x_k - \frac{x_k^n - a}{n x_k^{n-1}}$。
- 最后由于算法是二阶的所以很快就会收敛了。

Levenberg-Marquardt 算法

假设 f 是一个从 $\mathbb{R}^m \to \mathbb{R}^n$ 的非线性映射，也就是说 $\boldsymbol{x} \in \mathbb{R}^m$，经过 f 变换 $\boldsymbol{y} = f(\boldsymbol{w}|\boldsymbol{x})$ 有 $\boldsymbol{y} \in \mathbb{R}^n$。
Levenberg-Marquardt 算法的目的就是希望任意给定一个 $\boldsymbol{y}$ 以及合理的初始值 $\boldsymbol{w_0}$，能找到一个 $\boldsymbol{w}^*$，使得 $\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}$ 尽量小（局部极小），其中 $\boldsymbol{\epsilon} = f(\boldsymbol{w}^*|\boldsymbol{x}) - \boldsymbol{y}$
接着还是请出 $f(\boldsymbol{w}|\boldsymbol{x})$ 的泰勒展开 (一阶)：$f(\boldsymbol{w}+\boldsymbol{g}) \approx f(\boldsymbol{w}|\boldsymbol{x}) + \boldsymbol{J}\boldsymbol{g}$ 对于第 $k+1$ 次迭代:

$$\begin{aligned}\min \|\boldsymbol{\epsilon_{k+1}}\|^2 &= \min \|\boldsymbol{y} - f(\boldsymbol{w}+\boldsymbol{g}_k)\| \\ &\approx \min \|\boldsymbol{y} - f(\boldsymbol{w}|\boldsymbol{x}) - \boldsymbol{J}\boldsymbol{g_k}\|^2 \\ &= \min \|\boldsymbol{\epsilon_k} - \boldsymbol{J}\boldsymbol{g_k}\|\end{aligned} \tag{13}$$

该最小二乘问题有解析解：$(\boldsymbol{J^T}\boldsymbol{J})\boldsymbol{g_k} = \boldsymbol{J}^T\boldsymbol{\epsilon_k}$
对该式稍作修改可以得到（类似于岭回归）：$(\mu\boldsymbol{I} + \boldsymbol{J^T}\boldsymbol{J})\boldsymbol{g_k} = \boldsymbol{J}^T\boldsymbol{\epsilon_k}$
最后再说一句：当 μ 较大的时候上式接近梯度下降，反之接近牛顿法

BFGS 算法（1/2）

BFGS(**B**royden,**F**letcher,**G**oldfarb,**S**hanno)；本节内容参考：
`http://blog.csdn.net/itplus/article/details/21896453`
牛顿法是二阶收敛的，速度上是非常理想的；但是 Hessian 矩阵非常难以计算，在实际计算中往往得不偿失甚至计算不出来；**拟牛顿法**应运而生，其核心就是如何估计 Hessian 矩阵（或其逆）。
另一方面为了判断估计的好坏，于是有了**拟牛顿条件**，首先回顾牛顿法中的公式：

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \approx \frac{\partial f(\boldsymbol{x_0})}{\partial \boldsymbol{x}} + \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x_0})$$

令 $x = x_{k+1}, x_0 = x_k$ 并令 $\boldsymbol{g_{k+1}} = \frac{\partial f(\boldsymbol{x_{k+1}})}{\partial \boldsymbol{x}}, \boldsymbol{g_k} = \frac{\partial f(\boldsymbol{x_k})}{\partial \boldsymbol{x}}$ 带入上式得到**拟牛顿条件**：

$$\boldsymbol{g_{k+1}} - \boldsymbol{g_k} \approx \boldsymbol{H}_k(\boldsymbol{x_{k+1}} - \boldsymbol{x_k})$$

最后 $\boldsymbol{s_k} \triangleq \boldsymbol{x_{k+1}} - \boldsymbol{x_k}, \boldsymbol{y_k} \triangleq \boldsymbol{g_{k+1}} - \boldsymbol{g_k}$ 带入得到：

$$\boldsymbol{s_k} \approx \boldsymbol{H}^{-1}\boldsymbol{y_k} \; or \; \boldsymbol{y_k} \approx \boldsymbol{H}\boldsymbol{s_k}$$

BFGS 算法（2/2）

与 DFP[2]不同 BFGS 算法估计的是 Hessian 矩阵而前者估计的是 Hessian 矩阵的逆

BFGS: 利用迭代式 $\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \Delta\boldsymbol{B}_k$ 估计 $\boldsymbol{H}_k$:$\boldsymbol{B}_{k+1} \approx \boldsymbol{H}_k$，其主要有以下几步：

- 假设 $\Delta\boldsymbol{B}_k = \alpha\boldsymbol{u}\boldsymbol{u}^T + \beta\boldsymbol{v}\boldsymbol{v}^T$，其中 $\alpha, \beta, \boldsymbol{u}, \boldsymbol{v}$ 待定
- 带入拟牛顿条件:$\boldsymbol{y_k} \approx \boldsymbol{B}_k\boldsymbol{s}_k + (\alpha\boldsymbol{u}\boldsymbol{u}^T + \beta\boldsymbol{v}\boldsymbol{v}^T)\boldsymbol{s_k} = \boldsymbol{B}_k\boldsymbol{s}_k + (\alpha\boldsymbol{u}^T\boldsymbol{s_k})\boldsymbol{u} + (\beta\boldsymbol{v}^T\boldsymbol{s_k})\boldsymbol{v}$。
- 令 $(\alpha\boldsymbol{u}^T\boldsymbol{s_k}) = 1, (\beta\boldsymbol{v}^T\boldsymbol{s_k}) = -1, \boldsymbol{u} = \boldsymbol{y}_k, \boldsymbol{v} = \boldsymbol{B}_k\boldsymbol{s}_k$。
- 解得 $\alpha = \frac{1}{\boldsymbol{y}_k^T\boldsymbol{s}_k}, \beta = \frac{1}{\boldsymbol{s}_k^T\boldsymbol{B}_k\boldsymbol{s}_k}$，于是 $\Delta\boldsymbol{B}_k = \frac{\boldsymbol{y}_k\boldsymbol{y}_k^T}{\boldsymbol{y}_k^T\boldsymbol{s}_k} - \frac{\boldsymbol{B}_k\boldsymbol{s}_k\boldsymbol{s}_k^T\boldsymbol{B}_k}{\boldsymbol{s}_k^T\boldsymbol{B}_k\boldsymbol{s}_k}$。
- 更新公式 $\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \frac{\boldsymbol{y}_k\boldsymbol{y}_k^T}{\boldsymbol{y}_k^T\boldsymbol{s}_k} - \frac{\boldsymbol{B}_k\boldsymbol{s}_k\boldsymbol{s}_k^T\boldsymbol{B}_k}{\boldsymbol{s}_k^T\boldsymbol{B}_k\boldsymbol{s}_k}$。

最后利用 Sherman-Morrison 公式[3]可以方便的计算 $\boldsymbol{B}_{k+1}^{-1}$：

$$\boldsymbol{B}_{k+1}^{-1} = \left(\boldsymbol{I} - \frac{\boldsymbol{s_k}\boldsymbol{y_k^T}}{\boldsymbol{y_k^T}\boldsymbol{s_k}}\right)\boldsymbol{B_k^{-1}}\left(\boldsymbol{I} - \frac{\boldsymbol{y_k}\boldsymbol{s_k^T}}{\boldsymbol{y_k^T}\boldsymbol{s_k}}\right) + \frac{\boldsymbol{s_k}\boldsymbol{s_k^T}}{\boldsymbol{y_k^T}\boldsymbol{s_k}}$$

[2] http://blog.csdn.net/itplus/article/details/21896981

Sherman-Morrison 公式

定义

设 $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ 为非奇异方阵，$\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$，若 $1 + \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u} \neq 0$，则有：

$$(\boldsymbol{A} + \boldsymbol{u}\boldsymbol{v}^{\boldsymbol{T}})^{-1} = \boldsymbol{A}^{-1} - \frac{\boldsymbol{A}^{-1}\boldsymbol{u}\boldsymbol{v}^{\boldsymbol{T}}\boldsymbol{A}^{-1}}{1 + \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u}}$$

借此空间，show 一下 BFGS 算法的作者：

Broyden, Fletcher, Goldfarb, Shanno

L-BFGS 算法（1/3）

L-BFGS 是 Limit-Storage/Limit-Memory BFGS 的缩写，其设计的目的就是为了解决 BFGS 的内存问题（因为 B_k 在数据量较大时候其存储量是非常巨大的），解决的思路就是从下式出发:

$$\boldsymbol{B}_{k+1}^{-1} = \left(\boldsymbol{I} - \frac{\boldsymbol{s_k y_k^T}}{\boldsymbol{y_k^T s_k}}\right) \boldsymbol{B}_{\boldsymbol{k}}^{-1} \left(\boldsymbol{I} - \frac{\boldsymbol{y_k s_k^T}}{\boldsymbol{y_k^T s_k}}\right) + \frac{\boldsymbol{s_k s_k^T}}{\boldsymbol{y_k^T s_k}}$$

- 记 $\rho_k = \frac{1}{\boldsymbol{y_k^T s_k}}, \boldsymbol{V}_k = \boldsymbol{I} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k$；
- 于是有：$\boldsymbol{B}_{k+1}^{-1} = \boldsymbol{V}_k^T \boldsymbol{B}_k \boldsymbol{V}_k + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T$；递推得到:

$$\begin{aligned}\boldsymbol{B}_{k+1}^{-1} &= \left(\Pi_{i=0}^{k} \boldsymbol{V}_i^T\right) \boldsymbol{B}_0 \left(\Pi_{i=0}^{k} \boldsymbol{V}_i\right) \\ &+ \sum_{j=1}^{k} \left(\left(\Pi_{i=j}^{k} \boldsymbol{V}_i^T\right) \left(\boldsymbol{\rho}_{j-1} \boldsymbol{s}_{j-1} \boldsymbol{s}_{j-1}^T\right) \left(\Pi_{i=j}^{k} \boldsymbol{V}_i\right)\right) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T\end{aligned}$$

L-BFGS 算法（2/3）

接上一页内容：

- 计算 $\boldsymbol{B}_{k+1}$ 需要 $\{\boldsymbol{s}_i, \boldsymbol{y}_i\}_{k=0}^{k}$，这很耗空间，当 $k+1>m$ 考虑 m 阶截断：

$$
\begin{aligned}
\boldsymbol{B}_{k+1}^{-1} &= \left(\Pi_{i=k-m+1}^{k} \boldsymbol{V}_i^T\right) \boldsymbol{B}_0 \left(\Pi_{i=k-m+1}^{k} \boldsymbol{V}_i\right) \\
&\quad + \sum_{j=k-m+2}^{k} \left(\left(\Pi_{i=j}^{k} \boldsymbol{V}_i^T\right) (\boldsymbol{\rho}_{j-1} \boldsymbol{s}_{j-1} \boldsymbol{s}_{j-1}^T) \left(\Pi_{i=j}^{k} \boldsymbol{V}_i\right)\right) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T
\end{aligned}
$$

- 通常 $m<10$；至此，L-BFGS 算法的理论部分就基本完成了，下一节介绍其算法流程。

L-BFGS 算法（3/3）：算法流程[4]

算法 ($B_k \cdot \mathbf{g}_k$ 的快速算法)

Step 1 初始化.

$$\delta = \begin{cases} 0, & 若\ k \le m \\ k-m, & 若\ k > m \end{cases};\quad L = \begin{cases} k, & 若\ k \le m \\ m, & 若\ k > m \end{cases};\quad \mathbf{q}_L = \mathbf{g}_k.$$

Step 2 后向循环.

FOR $i = L-1, L-2, \cdots, 1, 0$ **DO**

{

$j = i + \delta;$

$\alpha_i = \rho_j \mathbf{s}_j^T \mathbf{q}_{i+1};$ // α_i 需要存下来，前向循环要用！

$\mathbf{q}_i = \mathbf{q}_{i+1} - \alpha_i \mathbf{y}_j.$

}

Step 3 前向循环.

$\mathbf{r}_0 = B_0 \cdot \mathbf{q}_0;$

FOR $i = 0, 1, \cdots, L-2, L-1$ **DO**

{

$j = i + \delta;$

$\beta_j = \rho_j \mathbf{y}_j^T \mathbf{r}_i;$

$\mathbf{r}_{i+1} = \mathbf{r}_i + (\alpha_i - \beta_i)\mathbf{s}_j.$

}

最后算出的 $\mathbf{r}_L$ 即为 $H_k \cdot \mathbf{g}_k$ 的值.

[4] Jorge Nocedal.updating quasi-newton matrices with limited storage. Mathematics of Computation.1980.

次梯度算法（1/3）

关于次梯度这里：`http://www.hanlongfei.com/convex/2015/10/02/cmu-10725-subgradidient/`做了比较详细的介绍（这里开始就完全是现学现卖了）；首先，来回顾一下凸函数的一阶判别条件[5]

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T(\boldsymbol{y}-\boldsymbol{x})$$

两个条件：1）$f(x)$ 可微，2）上式在**整个定义域**都满足
点 x 的**次梯度**指在函数 f 上的点 x 满足以下条件的 $\boldsymbol{g} \in \mathbb{R}^n$：

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{g}^T(\boldsymbol{y}-\boldsymbol{x})$$

几点注意：

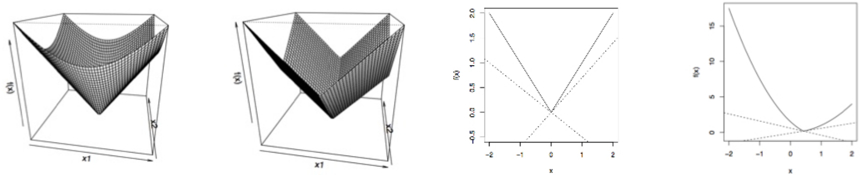
- $f(x)$ 可以不是凸函数也不要求它是可微的
- 该定义是在一个点（x）上定义的[6]
- 次梯度可以有多个（一个集合），可以只有有一个，也可以是空集

[5] 顺带说一下它的二阶判别条件就是，其 Hessian 阵是半正定的（如果存在的话）

[6] 个人认为应该还有一定的范围限制，如在 x 的领域内

次梯度算法（2/3）

几个需要计算次梯度的例子（各个图的函数形式位于图篇的底部）：



$\| x \|_2$　　$\| x \|_1$　　$| x |$　　$max\{f_1(x), f_2(x)\}$

有了次梯度接下来定义次微分（记为 $\partial f(\boldsymbol{x})$）的概念：

$$\partial f(\boldsymbol{x}) = \{\boldsymbol{g} \in \mathbb{R}^n : \boldsymbol{g} \ is \ a \ subgradient \ of \ f \ at \ \boldsymbol{x}\}$$

在第 k 次迭代时有了次梯度 $\boldsymbol{g}_k \in \partial f(\boldsymbol{x})$ 就可以类似于梯度下降的算法定于次梯度下降算法（其中 t_k 表示步长）：

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - t_k \boldsymbol{g}_k, k = 1, 2, 3, \ldots$$

次梯度算法（3/3）

关于次梯度算法，有几点需要注意：

- 如果目标函数是凸的那么它能保证算法是收敛的，但是次梯度算法并不能保证目标函数每次都是递减的（或者说是单调下降的），为此会有

$$f(\boldsymbol{x}_{best}^{(k)}) = \min_{i=0,1,\ldots,k} \{f(\boldsymbol{x}_i)\} = \min\{f(\boldsymbol{x}_k), f(\boldsymbol{x})_{k-1}\}$$

- 当次梯度方向不唯一的时候理论上是选取任意一个都可以的
- 与梯度下降不同次梯度下降的步长是预先设定好的且满足:

$$\sum_{i=0}^{\infty} t_i = \infty \ and \ \sum_{i=0}^{\infty} t_i^2 < \infty \tag{14}$$

关于公式14最后再说两句：1）公式前半部分保证了搜索范围；2）后半部分保证了步长减小（收敛）

Bregman Projection[7](1/2)

首先，来回顾一下 Bregman Divergence:

$$D_F(p,q) = F(p) - F(q) - \langle p - q, \nabla F(q) \rangle$$

其中 $F(x)$ 是实值可微的凸函数：$F(x) : \Omega \to R$，而所谓 Projection 就是指的是将原始空间中的数据（例如下面的 W_1）投影到某个子空间（集合）的过程。
其物理意义就是原数据到这个子空间（集合）的距离最短；所以 Bregman Projection 描述的是类似于如下形式的问题：

$$\begin{aligned} \boldsymbol{W}^* &= \min_{W} D_F(\boldsymbol{W}, \boldsymbol{W}_1) \\ s.t\ \boldsymbol{W} &= \boldsymbol{W}^T, \mathrm{tr}(\boldsymbol{W}) = 1 \\ &\mathrm{tr}(\boldsymbol{W}\boldsymbol{C}_j) < 0, for\ j = 1, 2, ..., n \end{aligned}$$

[7] Tsuda, K., Raetsch, G., & Warmuth, M. K. (2005). Matrix exponentiated gradient updates of online learning and bregman projection. Journal of Machine Learning Research, 6.

Bregman Projection[8](2/2)

然而全部约束一起考虑太难了，因此 Bregman 说了：

It is well known that the Bregman projection into the intersection of convex regions can be solved by sequential projections to each region (Bregman, 1967; Censor and Lent,1981)

于是前述问题可以改成每次只考虑一个约束条件的形式（通常是不满足的那个），循环迭代：

$$
\begin{aligned}
\boldsymbol{W}^* &= \min_{W} D_F(\boldsymbol{W}, \boldsymbol{W}_1) \\
s.t\ \boldsymbol{W} &= \boldsymbol{W}^T, \mathrm{tr}(\boldsymbol{W}) = 1 \\
&\mathrm{tr}(\boldsymbol{W}\boldsymbol{C}_j)
\end{aligned}
$$

[8] Tsuda, K., Raetsch, G., & Warmuth, M. K. (2005). Matrix exponentiated gradient updates of online learning and bregman projection. Journal of Machine Learning Research, 6.

Lagrange 对偶问题 (1/3)

前面介绍的内容都是目标函数已有后的优化方法，这一部分将会介绍如何将一个一般的有约束的问题转化为 Lagrange 对偶问题。
原问题：

$$\begin{aligned} & \min f_0(\boldsymbol{x}) \\ s.t \quad & f_i(\boldsymbol{x}) \leq 0, i = 1, 2, ..., m \\ & h_i(\boldsymbol{x}) = 0, i = 1, 2, ..., p \end{aligned} \tag{15}$$

Lagrange 函数:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{v}) = f_0(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{x}) + \sum_{i=1}^{p} v_i h_i(\boldsymbol{x}) \tag{16}$$

Lagrange 对偶函数:

$$g(\boldsymbol{\lambda}, \boldsymbol{v}) = \inf_{x} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{v}) = \inf_{x} \left(f_0(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{x}) + \sum_{i=1}^{p} v_i h_i(\boldsymbol{x}) \right) \tag{17}$$

Lagrange 对偶问题 (2/3)

对偶问题17对任意的 $\boldsymbol{\lambda} \geq \mathbf{0}$ 以及 $\boldsymbol{v}$ 都是原问题15的下界，此外对于 $\boldsymbol{\lambda} < \mathbf{0}$ 的情形这将导致 $g(\boldsymbol{\lambda}, \boldsymbol{v}) = -\infty$ 并没有实际意义。
既然 $g(\boldsymbol{\lambda}, \boldsymbol{v})$ 对于任意的 $\boldsymbol{\lambda} \geq \mathbf{0}$ 以及 $\boldsymbol{v}$ 是原问题15的下界，那么什么样的 $\boldsymbol{\lambda}, \boldsymbol{v}$ 才是好的，对偶问题考虑：

$$\max g(\boldsymbol{\lambda}, \boldsymbol{v}), s.t\ \boldsymbol{\lambda} \geq \mathbf{0} \tag{18}$$

所以实际上的 Lagranged 对偶问题优化的是：

$$\max_{\boldsymbol{\lambda}, \boldsymbol{v}} \left(\min_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{v}) \right)$$

关于 SVM 的博客:
`http://blog.csdn.net/v_july_v/article/details/7624837`
中把 SVM 作为一个实例对 Lagrange 对偶问题进行了解说

Lagrange 对偶问题 (3/3)

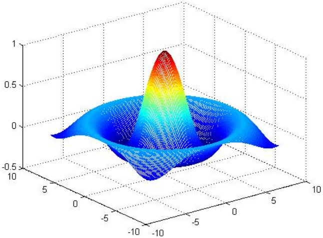
当原问题是凸问题的时候，满足 **KKT** 条件[9]的点也是对偶问题的解。
KKT 条件:

$$\begin{cases} \nabla f_0(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i \nabla f_i(\boldsymbol{x}) + \sum_{i=1}^{p} v_i \nabla h_i(\boldsymbol{x}) = \mathbf{0} \\ f_i(\boldsymbol{x}) \leq 0, i = 1, 2, ..., m \\ h_i(\boldsymbol{x}) = 0, i = 1, 2, ..., p \\ \lambda_i \geq 0, i = 1, 2, ..., m \\ \lambda_i f_i(\boldsymbol{x}) = 0, i = 1, 2, ..., m \end{cases}$$

[9] KKT：Karush-Kuhn-Tucker

几个例子——之一 (1/4)

首先，第一个例子来介绍一下 Mexico 草帽：



函数的形式：$f(x,y) = \frac{\sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$

几个例子——之一 (2/4)

接上一页：

- 函数的梯度形式：

$$\partial f(x,y) = \begin{cases} \cos(\sqrt{x^2+y^2})\dfrac{x}{x^2+y^2} - \sin(\sqrt{x^2+y^2})\dfrac{x}{\sqrt{x^2+y^2}^3} \\ \cos(\sqrt{x^2+y^2})\dfrac{y}{x^2+y^2} - \sin(\sqrt{x^2+y^2})\dfrac{y}{\sqrt{x^2+y^2}^3} \end{cases}$$

- Hessian 矩阵：

$$H = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix} \quad where \quad \begin{cases} a_{11} = -\sin(\sqrt{x^2+y^2})\dfrac{x^2(x^2+y^2)+y^2-2x^2}{\sqrt{x^2+y^2}^5} \\ \qquad + \cos(\sqrt{x^2+y^2})\dfrac{(y^2-2x^2)}{\sqrt{x^2+y^2}^4} \\ a_{12} = -\sin(\sqrt{x^2+y^2})\dfrac{xy(x^2+y^2)-3xy}{\sqrt{x^2+y^2}^5} \\ \qquad - \cos(\sqrt{x^2+y^2})\dfrac{3xy}{\sqrt{x^2+y^2}^4} \\ a_{22} = -\sin(\sqrt{x^2+y^2})\dfrac{y^2(x^2+y^2)+x^2-2y^2}{\sqrt{x^2+y^2}^5} \\ \qquad + \cos(\sqrt{x^2+y^2})\dfrac{(x^2-2y^2)}{\sqrt{x^2+y^2}^4} \end{cases}$$

几个例子——之一 (3/4)

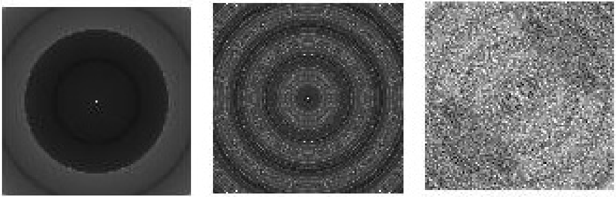
首先注意到的是：即便是如此简单的一个函数函数其 Hessian 矩阵也是难以计算的。
下面就来看一下墨西哥草帽上的优化问题的一些结果：

Mexico_hat(range=[-10,10,-10,10],step=0.2,scale=101×101)					
算法	总时间（单位：s）	maxiter	average_iter	存储	备注
grad	8.450431	500	93.97	$O(n)$	###
newton	3.720508	500	5.43	$O(n^2)$	###
bfgs	14.776714	500	42.27	$O(n^2)$	###

几个例子——之一 (4/4)

再来看几张图（从左到右：grad->newton->bfgs）：

把图片稍微锐化一下就是下面的样子了：



这些图片代表了在 [-10,10,-10,10] 的范围内以 0.2 为步长遍历所有点，并把它作为初始值赋给算法，然后各个点的灰度值代表了迭代次数大小

几个例子——之二 (1/2)

这次做的是回归问题：$f(x) = p_1 e^{p_2 x}$，采样 $x = -1 : 0.02 : 1$，获得观测值 $yn = f(x)$，其中真实参数 $(p_1, p_2) = (0.5, 0.1)$，目标函数：

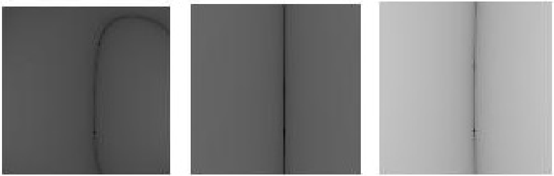
$$\min \|\epsilon\epsilon^T\|, \epsilon = \hat{y} - yn$$

这次直接上结果：

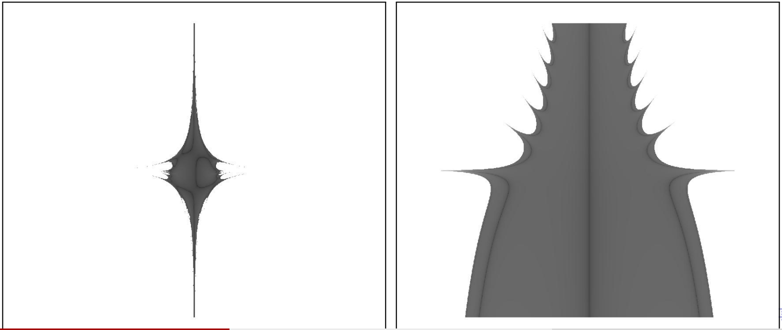
fit(range=[-1,1,-1,1],step=0.02,scale=101×101)					
算法	总时间（单位：s）	maxiter	average_iter	存储	备注
grad	90.776934	178	156.14	$O(ns)$	###
levmar	160.395254	384	351.70	$O(ns)$	$\mu = 1$
levmar	91.96887	217	200.64	$O(ns)$	$\mu = 0.1$
levmar	84.676262	200	185.18	$O(ns)$	$\mu = 0.01$
levmar	83.078666	198	182.28	$O(ns)$	$\mu = 0$
fit(range=[-10,10,-10,10],step=0.02,scale=1001×1001)					
grad	14965.815	500	332.04	$O(ns)$	###
levmar	31156.32704	500	486.58	$O(ns)$	$\mu = 0$

几个例子——之二 (2/2)

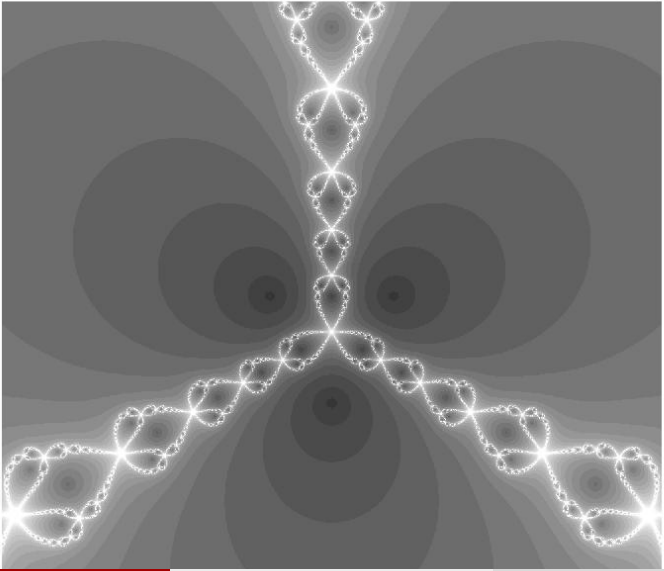
最后还是看几个图（从左到右：grad，levmar($\mu = 0.01$)，levmar($\mu = 1$)）：



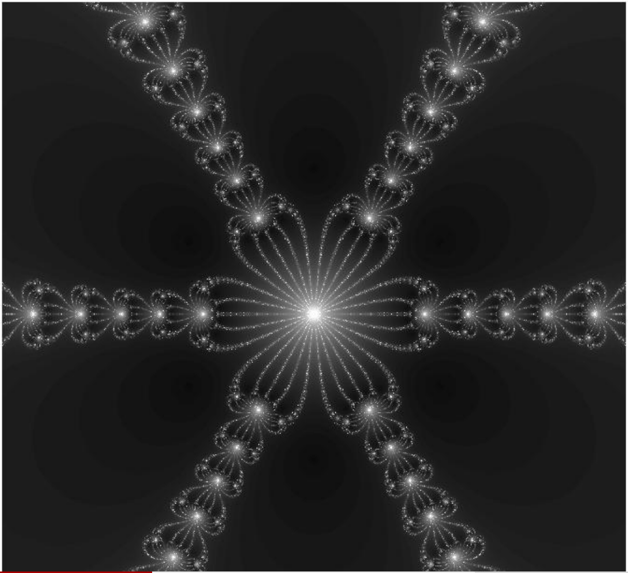
更大范围内的情况 (range=[-10,10,-10,10]，左到右：grad,levmar($\mu = 0$))：



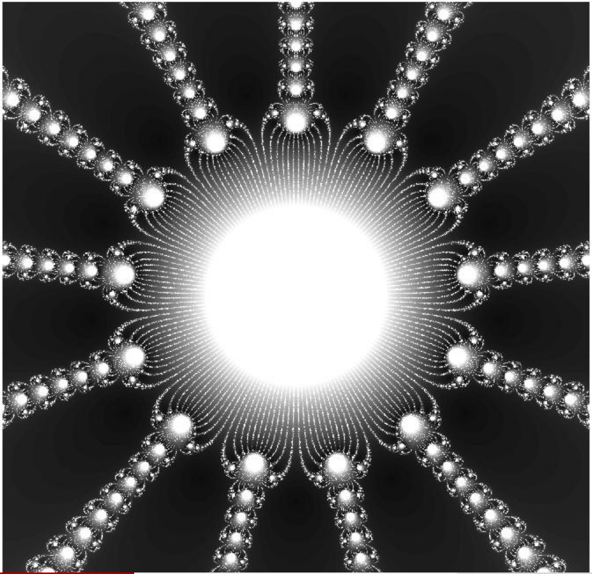
几个例子——之 Newton 法与多项式 $(x^3 - 0.01)$



几个例子——之 Newton 法与多项式 $(x^6 - 0.01)$



几个例子——之 Newton 法与多项式 $(x^{13} - 0.01)$



下集预告

- 本集遗漏的内容
  - 共轭梯度算法
  - EM 算法
- 一维搜索（线搜），以下内容全凭记忆想到，欢迎指正和补充
  - 二分查找
  - 0.618 法
  - BFGS 一维的情况
  - 其它（暂时没想好）

Q&A 由于是现学现卖，我尽力而为

谢谢！