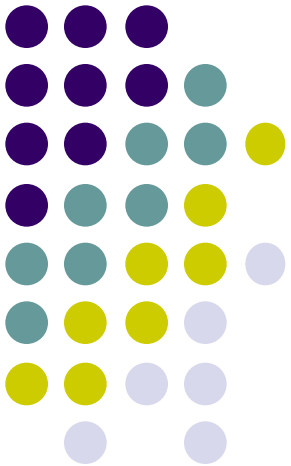
最优化方法 (IV)

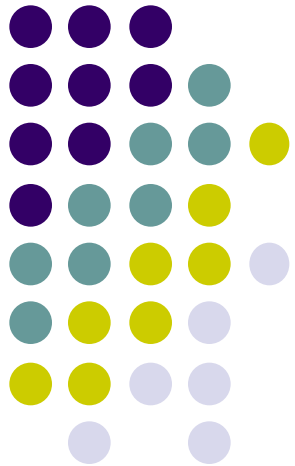
张宏鑫

2019-04-16

浙江大学计算机学院



1. 无约束非线性最优化



一. 无约束最优化问题

无约束最优化问题

$$\min \quad f(x)$$

$$s.t. \quad x \in R^n$$

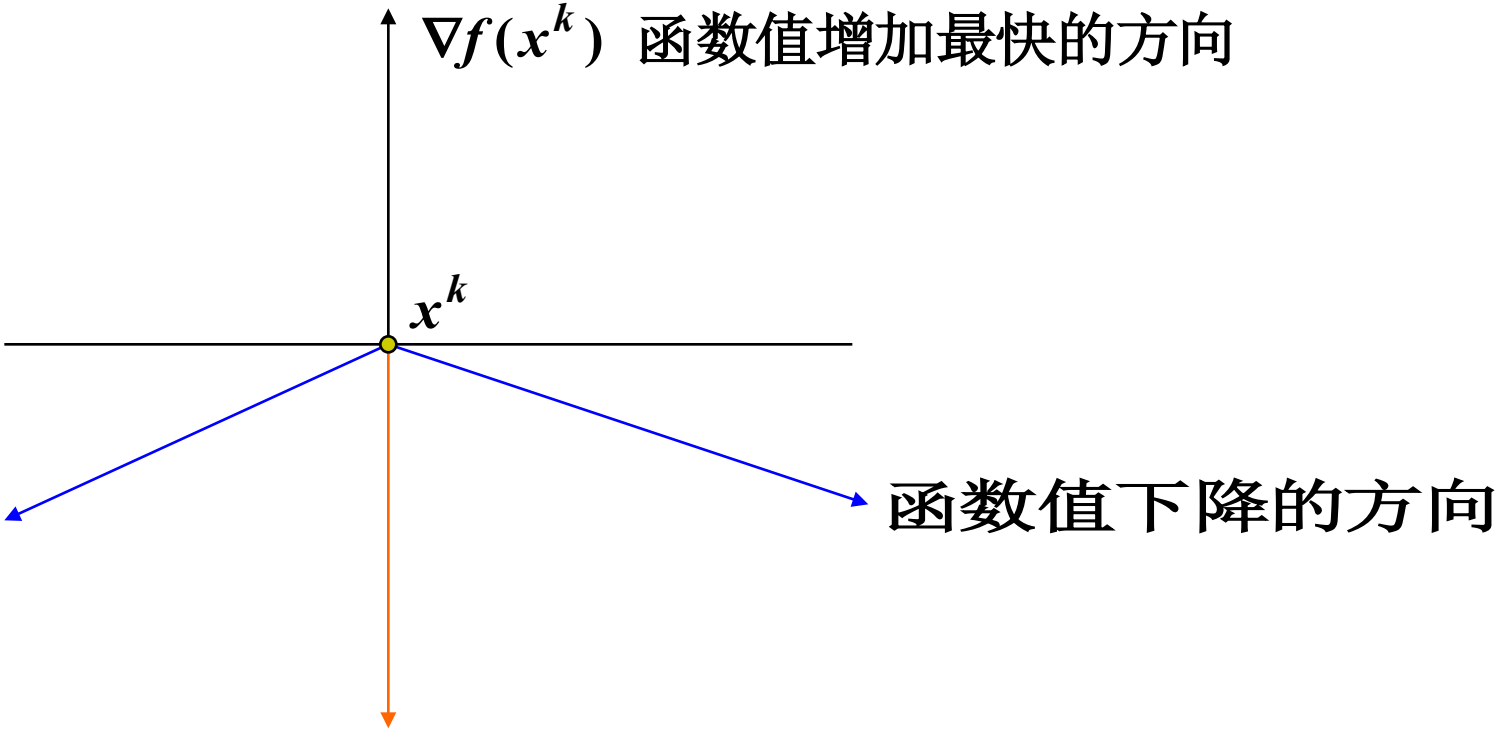
其中$f(x)$有一阶连续偏导数。

解析方法：利用函数的解析性质构造迭代公式使之收敛到最优解。

二. 梯度法（最速下降法）

迭代公式： $x^{k+1} = x^k + \lambda_k d^k$

如何选择下降最快的方向？



梯度法（最速下降法）：

1. 搜索方向：$d^k = -\nabla f(x^k)$，也称为最速下降方向；

2. 搜索步长：λ_k 取最优步长，即满足 $f(x^k + \lambda_k d^k) = \min_{\lambda} f(x^k + \lambda d^k)$。

梯度法算法步骤：

1. 给定初始点 $x^1 \in R^n$，允许误差 $\varepsilon > 0$，令 $k = 1$。

2. 计算搜索方向 $d^k = -\nabla f(x^k)$；

3. 若 $\| d^k \| \le \varepsilon$，则停止计算，$x^k$ 为所求极值点；否则，求最优步长 λ_k

   使得 $f(x^k + \lambda_k d^k) = \min_{\lambda} f(x^k + \lambda d^k)$。

4. 令 $x^{k+1} = x^k + \lambda_k d^k$，令 $k := k + 1$，转2。

例. 用最速下降法求解: $\min\ f(x)=x_1^2+3x_2^2$,

设初始点为$x^1=(2,1)^T$,

求迭代一次后的迭代点 x^2。

解: $\because\ \nabla f(x)=(2x_1,6x_2)^T$,

$\therefore\ d^1=-\nabla f(x^1)=(-4,-6)^T$.

$\therefore\ x^1+\lambda d^1=(2-4\lambda,1-6\lambda)^T$.

令 $\varphi(\lambda)=f(x^1+\lambda d^1)=(2-4\lambda)^2+3(1-6\lambda)^2$,

求解 $\min\limits_{\lambda}\varphi(\lambda)$

令 $\varphi'(\lambda)=-8(2-4\lambda)-36(1-6\lambda)=0\Rightarrow\lambda_1=\dfrac{13}{62}$

$\therefore\ x^2=x^1+\lambda_1 d^1=(\dfrac{36}{31},\dfrac{-8}{31})^T$

收敛性

性质. 设 $f(x)$ 有一阶连续偏导数，若 步长 λ_k 满足

$$f(x^k+\lambda_k d^k)=\min_{\lambda} f(x^k+\lambda d^k)$$

则有 $\nabla f(x^k+\lambda_k d^k)^T d^k=0$。

证明：令 $\varphi(\lambda)=f(x^k+\lambda d^k)$，所以

$$\varphi'(\lambda)=\nabla f(x^k+\lambda d^k)^T d^k .$$

$$\because\ f(x^k+\lambda_k d^k)=\min_{\lambda} f(x^k+\lambda d^k)$$

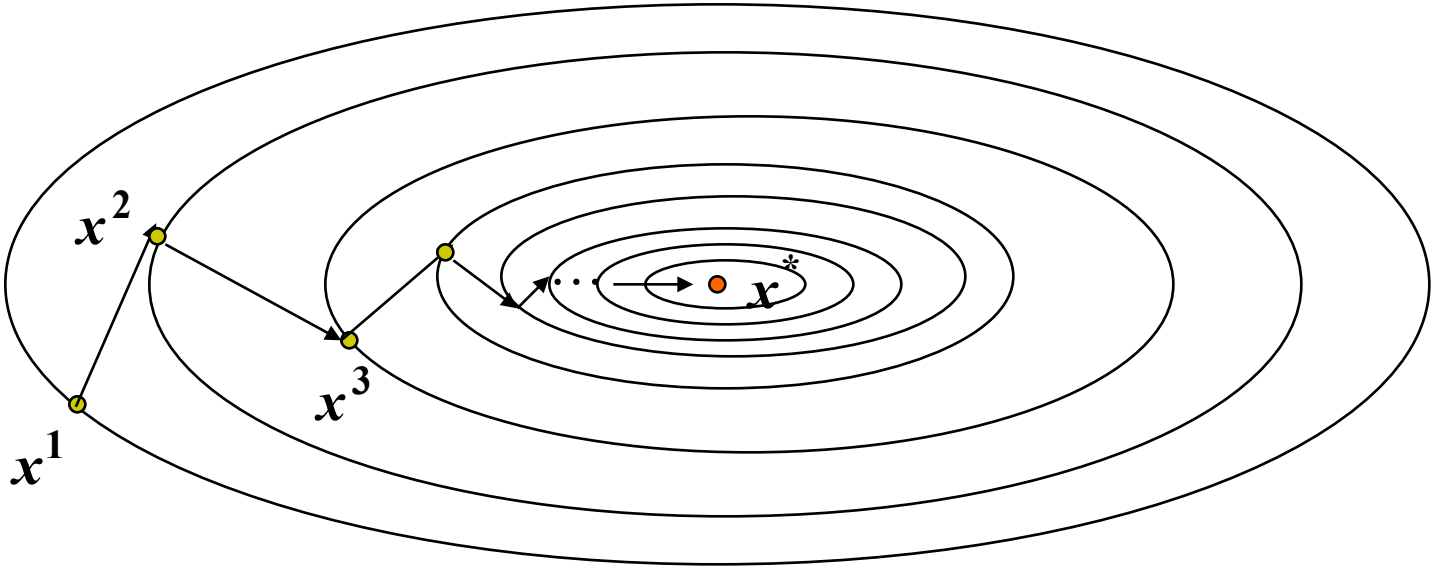
$$\therefore\ \varphi'(\lambda_k)=\nabla f(x^k+\lambda_k d^k)^T d^k=0 .$$

注：因为梯度法的搜索方向 $d^{k+1}=-\nabla f(x^k+\lambda_k d^k)$，所以

$(d^{k+1})^T d^k=0\Rightarrow d^{k+1}\perp d^k$。

锯齿现象

在极小点附近，目标函数可以用二次函数近似，其等值面近似椭球面。



最速下降方向反映了目标函数的一种局部性质
它只是局部目标函数值下降最快的方向
最速下降法是线性收敛的算法

三. 共轭梯度法

1. 共轭方向和共轭方向法

定义 设 A 是 $n\times n$ 的对称正定矩阵，对于 R^n中的两个非零向量 d^1 和 d^2，

若有 $d^{1T}Ad^2=0$，则称 d^1和d^2关于A共轭。

设 $d^1, d^2,\cdots,d^k$ 是 R^n 中一组非零向量，如果 它们两两关于 A 共轭，即 $d^{iT}Ad^j=0, i\neq j, i,j=1,2,\cdots,k$。

则称这组方向是关于A共轭的，也称它们是一组A共轭方向。

注：如果A是单位矩阵，则

$$d^{1T}\cdot I\cdot d^2=0\Rightarrow d^{1T}\cdot d^2=0$$

$$\Rightarrow d^1\perp d^2$$

共轭是正交的推广。

定理 1. 设 A是 n阶对称正定矩阵，$d^1, d^2, \cdots, d^k$ 是 k 个 A共轭的非零向量，则这个向量组线性无关。

证明　设存在实数 $\alpha_1, \alpha_2, \cdots, \alpha_k$，使得

$$\sum_{i=1}^{k} \alpha_i d^i = 0,$$

上式两边同时左乘 $d^{jT} A$，则有

$$\sum_{i=1}^{k} \alpha_i d^{jT} A d^i = 0,$$

因为 $d^1, d^2, \cdots, d^k$ 是 k 个 A共轭的向量，所以上式 可化简为

$$\alpha_j d^{jT} A d^j = 0.$$

因为 $d^j \neq 0$, 而 A是正定矩阵，所以 $d^{jT} A d^j > 0$，

所以　　$\alpha_j = 0,\ j = 1, 2, \cdots, k$。

因此 $d^1, d^2, \cdots, d^k$ 线性无关。

几何意义

设有二次函数

$$f(x)=\frac{1}{2}(x-\bar{x})^T A(x-\bar{x})$$

其中 A 是 $n\times n$ 对称正定矩阵，$\bar{x}$ 是一个定点。

则函数 $f(x)$ 的等值面 $\frac{1}{2}(x-\bar{x})^T A(x-\bar{x})=c$

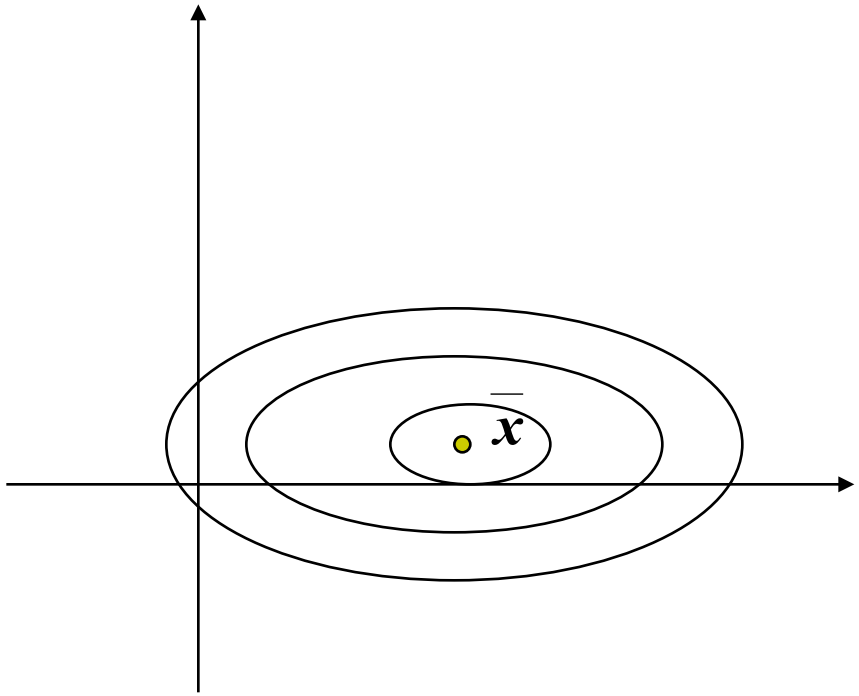
是以 $\bar{x}$ 为中心的椭球面。

由于 $\nabla f(\bar{x})=A(\bar{x}-\bar{x})=0,$

而 $\nabla^2 f(\bar{x})=A,$

因为A 正定，所以$\nabla^2 f(\bar{x})=A>0,$

因此 $\bar{x}$ 是 $f(x)$ 的极小点。



设 $x^{(0)}$ 是在某个等值面上的一点，$d^{(1)}$ 是 R^n 中的一个方向，

$x^{(0)}$ 沿着 $d^{(1)}$ 以最优步长搜索得到点 $x^{(1)}$。

则 $d^{(1)}$ 是点 $x^{(1)}$ 所在等值面的切向量。

该等值面在点 $x^{(1)}$ 处的法向量为

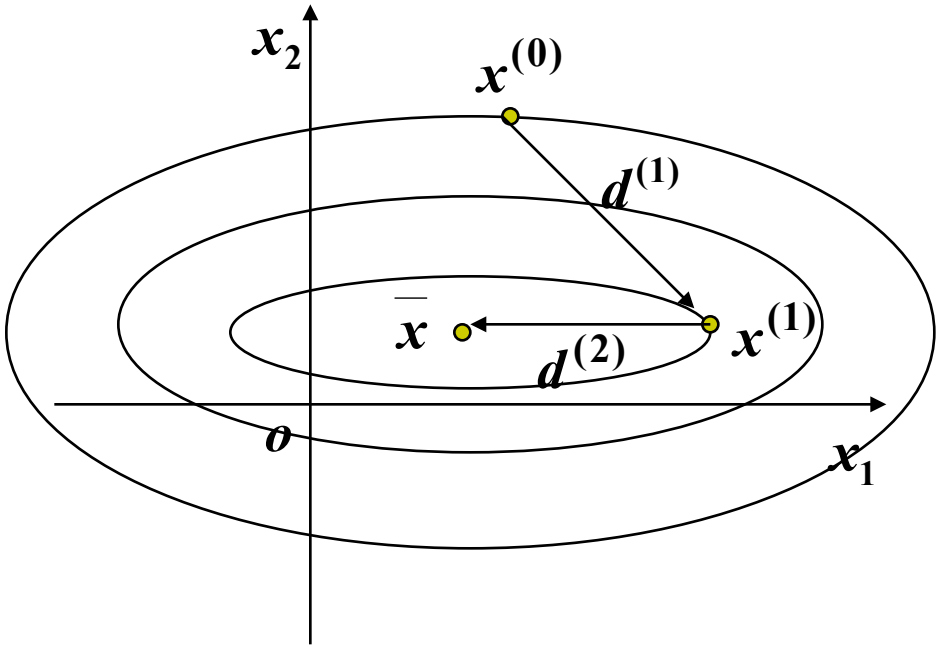
$$\nabla f(x^{(1)}) = A(x^{(1)} - \bar{x}).$$

则 $d^{(1)}$ 与 $\nabla f(x^{(1)})$ 正交，

即 $d^{(1)T}\nabla f(x^{(1)}) = 0,$

令 $d^{(2)} = \bar{x} - x^{(1)},$

所以 $d^{(1)T} A d^{(2)} = 0,$



即等值面上一点处的切 向量与由这一点指向极 小点的向量关于 A 共轭。

定理 2. 设有函数 $f(x)=\frac{1}{2}x^TAx+b^Tx+c$，

其中 A 是 n 阶对称正定矩阵。$d^{(1)},d^{(2)},\cdots,d^{(k)}$ 是一组 A 共轭向量。

以任意的 $x^{(1)}\in R^n$ 为初始点，依次沿 $d^{(1)},d^{(2)},\cdots,d^{(k)}$ 进行搜索，

得到点 $x^{(2)},x^{(3)},\cdots,x^{(k+1)}$，则 $x^{(k+1)}$ 是函数 $f(x)$ 在 $x^{(1)}+B_k$ 上的

极小点，其中

$$B_k=\{x\mid x=\sum_{i=1}^{k}\lambda_i d^{(i)},\lambda_i\in R\}$$

是由 $d^{(1)},d^{(2)},\cdots,d^{(k)}$ 生成的子空间。特别地，当 $k=n$ 时，$x^{(n+1)}$ 是

$f(x)$ 在 R^n 上的唯一极小点。

推论 在上述定理条件下，必有

$$\nabla f(x^{(k+1)})^T d^{(i)}=0,\quad i=1,2,\cdots,k。$$

共轭方向法

对于极小化问题

$$\min \quad f(x)=\frac{1}{2}x^{T}Ax+b^{T}x+c,$$

其中A是正定矩阵，称下述算法为共轭方向法：

(1) 取定一组 A 共轭方向 $d^{(1)},d^{(2)},\cdots,d^{(n)}$；

(2) 任取初始点 $x^{(1)}$，依次按照下式由 $x^{(k)}$ 确定点 $x^{(k+1)}$，

$$\begin{cases} x^{(k+1)}=x^{(k)}+\lambda_k d^{(k)} \\ f(x^{(k)}+\lambda_k d^{(k)})=\min\limits_{\lambda} f(x^{(k)}+\lambda d^{(k)}) \end{cases}$$

直到某个 $x^{(k)}$ 满足 $\nabla f(x^{(k)})=0$。

注 由定理2可知，利用共轭方向法求解上述极小化问题，至多经过 n 次迭代必可得到最优解。

如何选取一组共轭方向？

2. 共轭梯度法

Fletcher – Reeves 共轭梯度法：

$$\min \quad f(x)=\frac{1}{2}x^{T}Ax+b^{T}x+c$$

其中 $x\in R^{n}$，A是对称正定矩阵，$b\in R^{n}$，c是常数。

基本思想：将共轭性和最速下降方向相结合，利用已知迭代点处的梯度方向构造一组共轭方向，并沿此方向进行搜索，求出函数的极小点。

以下分析算法的具体步骤。

(1) 任取初始点 $x^{(1)}$，第一个搜索方向取为 $d^{(1)} = -\nabla f(x^{(1)})$；

(2) 设已求得点 $x^{(k+1)}$，若$\nabla f(x^{(k+1)}) \neq 0$，令 $g_{k+1} = \nabla f(x^{(k+1)})$，

则下一个搜索方向 $d^{(k+1)}$按如下方式确定：

令 $d^{(k+1)} = -g_{k+1} + \beta_k d^{(k)}$ (1)

如何确定 β_k?

要求$d^{(k+1)}$和$d^{(k)}$关于A共轭。

则在（1）式两边同时左乘 $d^{(k)^T} A$，得

$$0 = d^{(k)^T} A d^{(k+1)} = -d^{(k)^T} A g_{k+1} + \beta_k d^{(k)^T} A d^{(k)}$$

解得 $\beta_k = \dfrac{d^{(k)^T} A g_{k+1}}{d^{(k)^T} A d^{(k)}}$ (2)

(3) 搜索步长的确定：

已知迭代点 $x^{(k)}$ 和搜索方向 $d^{(k)}$，利用一维搜索确定最优步长 λ_k，

即求解 $\min\limits_{\lambda} \quad f(x^{(k)}+\lambda d^{(k)})$。

记 $\varphi(\lambda)=f(x^{(k)}+\lambda d^{(k)})$,

令 $\varphi'(\lambda)=\nabla f(x^{(k)}+\lambda d^{(k)})^T d^{(k)}=0$,

即有 $[A(x^{(k)}+\lambda d^{(k)})+b]^T d^{(k)}=0$,

令 $g_k=\nabla f(x^{(k)})=Ax^{(k)}+b$，则有

$$[g_k+\lambda A d^{(k)}]^T d^{(k)}=0,$$

解得 $$\lambda_k=-\frac{g_k^T d^{(k)}}{d^{(k)^T} A d^{(k)}} \qquad (3)$$

定理3 对于正定二次函数 $f(x)=\frac{1}{2}x^TAx+b^Tx+c$，FR算法在 $m\le n$ 次一维搜索后即终止，并且对所有的 $i(1\le i\le m)$，下列关系成立

(1) $d^{(i)^T}Ad^{(j)}=0,\ j=1,2,\cdots,i-1;$

(2) $g_i{}^Tg_j=0,\ j=1,2,\cdots,i-1;$

(3) $g_i^Td^{(i)}=-g_i^Tg_i$。

注

(1) 由定理3可知搜索方向 $d^{(1)},d^{(2)},\cdots,d^{(m)}$ 是 A 共轭的。

(2) 算法中第一个搜索方向必须取负梯度方向，否则构造的搜索方向不能保证共轭性。

(3) 由定理3的（3）可知，$g_i^Td^{(i)}=-g_i^Tg_i=-\|g_i\|^2<0$，所以 $d^{(i)}$ 是迭代点 $x^{(i)}$ 处的下降方向。

(4) 由定理3，*FR*算法中β_i的计算公式可以简化。

$$\beta_i = \frac{d^{(i)T} A g_{i+1}}{d^{(i)T} A d^{(i)}} = \frac{g_{i+1}^T A d^{(i)}}{d^{(i)T} A d^{(i)}}$$

$$= \frac{g_{i+1}^T A[(x^{(i+1)} - x^{(i)})/\lambda_i]}{d^{(i)T} A[(x^{(i+1)} - x^{(i)})/\lambda_i]}$$

$$\because \quad g_i = \nabla f(x^{(i)}) = A x^{(i)} + b.$$

$$\therefore \quad \beta_i = \frac{g_{i+1}^T (g_{i+1} - g_i)}{d^{(i)T} (g_{i+1} - g_i)} = \frac{\| g_{i+1} \|^2}{- d^{(i)T} g_i}$$

$$= \frac{\| g_{i+1} \|^2}{\| g_i \|^2} \qquad (4)$$

***FR*算法步骤：**

1. 任取初始点 $x^{(1)}$，精度要求 ε，令 $k=1$。

2. 令 $g_1=\nabla f(x^{(1)})$，若 $\|g_1\|<\varepsilon$，停止，$x^{(1)}$ 为所求极小点；

否则，令 $d^{(1)}=-g_1$，利用公式（3）计算 λ_1，令 $x^{(2)}=x^{(1)}+\lambda_1 d^{(1)}$。

3. 令 $g_{k+1}=\nabla f(x^{(k+1)})$，若 $\|g_{k+1}\|<\varepsilon$，停止，$x^{(k+1)}$ 为所求极小点；

否则，令 $d^{(k+1)}=-g_{k+1}+\beta_k d^{(k)}$，其中 β_k 用公式（4）计算。

令 $k:=k+1$。

4. 利用公式（3）计算 λ_k，令 $x^{(k+1)}=x^{(k)}+\lambda_k d^{(k)}$，转3。

例 用*FR*算法求解下述问题：

$$\min \quad f(x)=2x_1^2+x_2^2$$

初始点取为$x^{(1)}=(2,2)^T$。

解：$\because \quad f(x)=\frac{1}{2}(x_1,x_2)\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}, \quad \therefore \quad A=\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}.$

$$\nabla f(x)=(4x_1,2x_2)^T.$$

第1次迭代：

令 $d^{(1)}=-g_1=(-8,-4)^T,$

而 $\lambda_1=-\frac{g_1^T d^{(1)}}{d^{(1)T}Ad^{(1)}}$

$$=-\frac{(8,4)\begin{bmatrix}-8\\ -4\end{bmatrix}}{(-8,-4)\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}\begin{bmatrix}-8\\ -4\end{bmatrix}}=\frac{5}{18}$$

所以 $x^{(2)} = x^{(1)} + \lambda_1 d^{(1)}$

$$= (2,2)^T + \frac{5}{18}(-8,-4)^T = (\frac{-2}{9},\frac{8}{9})^T$$

第2次迭代:

$\because \quad g_2 = (\frac{-8}{9},\frac{16}{9})^T.$

$$\therefore \quad \beta_1 = \frac{\| g_2 \|^2}{\| g_1 \|^2} = \frac{(\frac{-8}{9})^2 + (\frac{16}{9})^2}{8^2 + 4^2} = \frac{4}{81}.$$

$\therefore \quad d^{(2)} = -g_2 + \beta_1 d^{(1)}$

$$= (\frac{8}{9},\frac{-16}{9})^T + \frac{4}{81}(-8,-4)^T$$

$$= \frac{40}{81}(1,-4)^T$$

$$\because \quad \lambda_2 = -\frac{g_2^T d^{(2)}}{d^{(2)T} A d^{(2)}}$$

$$= -\frac{\frac{40}{81}\left(\frac{-8}{9}, \frac{16}{9}\right)\begin{bmatrix} 1 \\ -4 \end{bmatrix}}{\left(\frac{40}{81}\right)^2 (1,-4)\begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ -4 \end{bmatrix}} = \frac{9}{20}$$

$$\therefore \quad x^{(3)} = x^{(2)} + \lambda_2 d^{(2)}$$

$$= \left(\frac{-2}{9}, \frac{8}{9}\right)^T + \frac{9}{20} \times \frac{40}{81}(1,-4)^T$$

$$= (0,0)^T$$

$$\because \quad g_3 = (0,0)^T$$

$\therefore$ $x^{(3)}$即为所求极小点。

3. 用于一般函数的共轭梯度法

$$\min \quad f(x)$$

$$s.t. \quad x \in R^n$$

对用于正定二次函数的共轭梯度法进行修改：

(1) 第一个搜索方向仍取最速下降方向，即 $d^{(1)} = -\nabla f(x^{(1)})$。

其它搜索方向按下式计算：

$$d^{(i+1)} = -\nabla f(x^{(i+1)}) + \beta_i d^{(i)},$$

其中 $\beta_i = \dfrac{\| \nabla f(x^{(i+1)}) \|^2}{\| \nabla f(x^{(i)}) \|^2}$。

(2) 搜索步长 λ_i 不能利用公式（3）计算，需由一维搜索确定。

(3) 算法在有限步迭代后不一定能满足停止条件，此时可采取如下措施：

以n次迭代为一轮，每次完成一轮搜索后，如果还没有求得极小点，则以上一轮的最后一个迭代点作为新的初始点，取最速下降方向作为第一个搜索方向，开始下一轮搜索。

注 在共轭梯度法中，也可采用其它形式的公式计算β_i，如

$$\beta_i = \frac{g_{i+1}^T(g_{i+1} - g_i)}{g_i^T g_i}$$ （*PRP*共轭梯度法）。

Numerical Optimization: Understanding L-BFGS

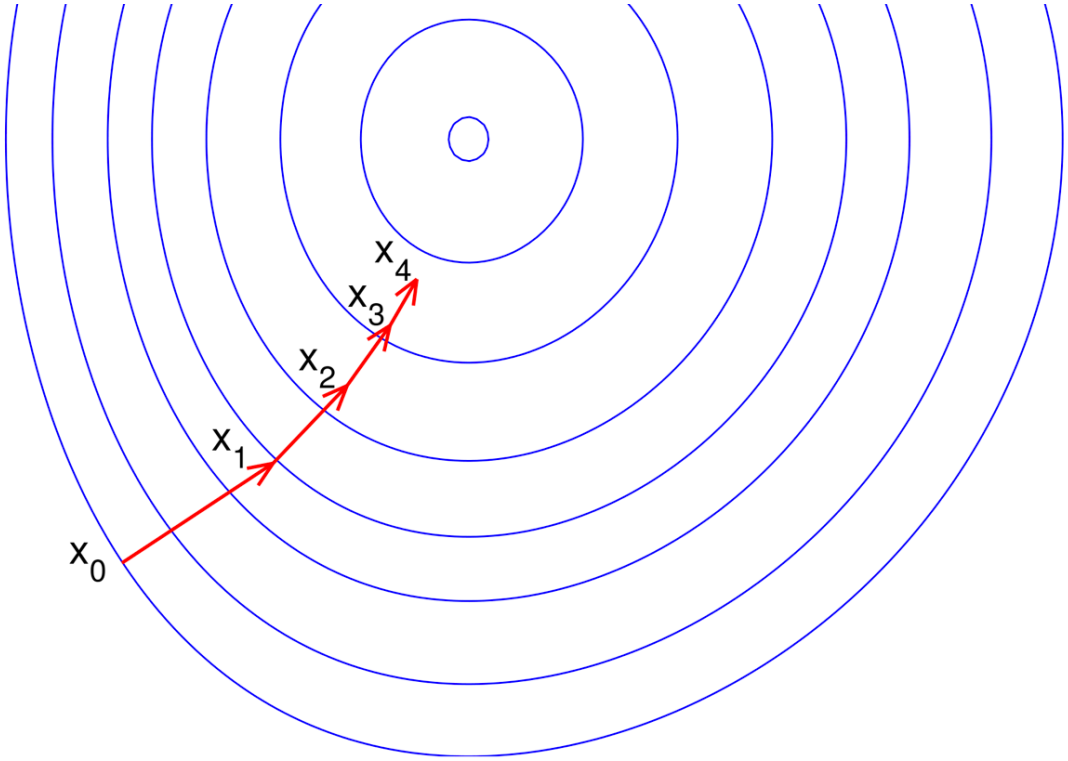
- http://aria42.com/blog/2014/12/understanding-lbfgs

Numerical optimization is at the core of much of machine learning. Once you've defined your model and have a dataset ready, estimating the parameters of your model typically boils down to minimizing some multivariate function $f(x)$, where the input x is in some high-dimensional space and corresponds to model parameters. In other words, if you solve:

$$x^* = \arg\min_x f(x)$$

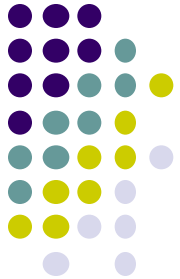
then x^* is the 'best' choice for model parameters according to how you've set your objective.[1]

Newton's method



Two thing: gradient and Hessian

Newton's method



Most numerical optimization procedures are iterative algorithms which consider a sequence of 'guesses' x_n which ultimately converge to x^* the true global minimizer of f. Suppose, we have an estimate x_n and we want our next estimate x_{n+1} to have the property that $f(x_{n+1}) < f(x_n)$.

Newton's method is centered around a quadratic approximation of f for points near x_n. Assuming that f is twice-differentiable, we can use a quadratic approximation of f for points 'near' a fixed point x using a Taylor expansion:

$$f(x + \Delta x) \approx f(x) + \Delta x^T \nabla f(x) + \frac{1}{2} \Delta x^T \left(\nabla^2 f(x) \right) \Delta x$$

where $\nabla f(x)$ and $\nabla^2 f(x)$ are the gradient and Hessian of f at the point x_n. This approximation holds in the limit as $||\Delta x|| \to 0$. This is a generalization of the single-dimensional Taylor polynomial expansion you might remember from Calculus.

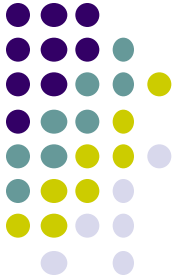
Newton's method

In order to simplify much of the notation, we're going to think of our iterative algorithm of producing a sequence of such quadratic approximations h_n. Without loss of generality, we can write $x_{n+1} = x_n + \Delta x$ and re-write the above equation,

$$h_n(\Delta x) = f(x_n) + \Delta x^T \mathbf{g}_n + \frac{1}{2}\Delta x^T \mathbf{H}_n \Delta x$$

where $\mathbf{g}_n$ and $\mathbf{H}_n$ represent the gradient and Hessian of f at x_n.

Newton's method



We want to choose Δx to minimize this local quadratic approximation of f at x_n. Differentiating with respect to Δx above yields:

$$\frac{\partial h_n(\Delta x)}{\partial \Delta x} = \mathbf{g}_n + \mathbf{H}_n \Delta x$$

Recall that any Δx which yields $\frac{\partial h_n(\Delta x)}{\partial \Delta x} = 0$ is a local extrema of $h_n(\cdot)$. If we assume that $\mathbf{H}_n$ is [postive definite] (psd) then we know this Δx is also a global minimum for $h_n(\cdot)$. Solving for Δx:[2]

$$\Delta x = -\mathbf{H}_n^{-1}\mathbf{g}_n$$

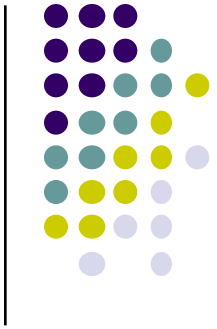
This suggests $\mathbf{H}_n^{-1}\mathbf{g}_n$ as a good direction to move x_n towards. In practice, we set $x_{n+1} = x_n - \alpha(\mathbf{H}_n^{-1}\mathbf{g}_n)$ for a value of α where $f(x_{n+1})$ is 'sufficiently' smaller than $f(x_n)$.

Iterative Algorithm

$$
\begin{aligned}
&\textbf{NewtonRaphson}(f, x_0):\\
&\quad \text{For } n = 0, 1, \ldots \text{ (until converged)}:\\
&\qquad \text{Compute } \mathbf{g}_n \text{ and } \mathbf{H}_n^{-1} \text{ for } x_n\\
&\qquad d = \mathbf{H}_n^{-1}\mathbf{g}_n\\
&\qquad \alpha = \min_{\alpha \geq 0} f(x_n - \alpha d)\\
&\qquad x_{n+1} \leftarrow x_n - \alpha d
\end{aligned}
$$

The computation of the α step-size can use any number of line search algorithms. The simplest of these is backtracking line search, where you simply try smaller and smaller values of α until the function value is 'small enough'.

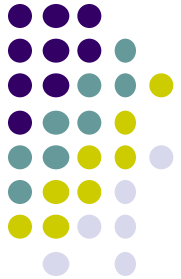
Huge Hessians



The central issue with $\mathbf{NewtonRaphson}$ is that we need to be able to compute the inverse Hessian matrix.[3] Note that for ML applications, the dimensionality of the input to f typically corresponds to model parameters. It's not unusual to have hundreds of millions of parameters or in some vision applications even billions of parameters. For these reasons, computing the hessian or its inverse is often impractical. For many functions, the hessian may not even be analytically computable, let along representable.

Because of these reasons, $\mathbf{NewtonRaphson}$ is rarely used in practice to optimize functions corresponding to large problems. Luckily, the above algorithm can still work even if $\mathbf{H}_n^{-1}$ doesn't correspond to the exact inverse hessian at x_n, but is instead a good approximation.

Quasi-Newton（拟牛顿法）



Suppose that instead of requiring $\mathbf{H}_n^{-1}$ be the inverse hessian at x_n, we think of it as an approximation of this information. We can generalize $\mathbf{NewtonRaphson}$ to take a $\mathrm{QuasiUpdate}$ policy which is responsible for producing a sequence of $\mathbf{H}_n^{-1}$.

$$
\begin{aligned}
&\mathbf{QuasiNewton}(f, x_0, \mathbf{H}_0^{-1}, \mathrm{QuasiUpdate}) : \\
&\quad \text{For } n = 0, 1, \ldots \text{ (until converged)} : \\
&\qquad // \text{ Compute search direction and step-size} \\
&\qquad d = \mathbf{H}_n^{-1} \mathbf{g}_n \\
&\qquad \alpha \leftarrow \min_{\alpha \geq 0} f(x_n - \alpha d) \\
&\qquad x_{n+1} \leftarrow x_n - \alpha d \\
&\qquad // \text{ Store the input and gradient deltas} \\
&\qquad \mathbf{g}_{n+1} \leftarrow \nabla f(x_{n+1}) \\
&\qquad s_{n+1} \leftarrow x_{n+1} - x_n \\
&\qquad y_{n+1} \leftarrow \mathbf{g}_{n+1} - \mathbf{g}_n \\
&\qquad // \text{ Update inverse hessian} \\
&\qquad \mathbf{H}_{n+1}^{-1} \leftarrow \mathrm{QuasiUpdate}(\mathbf{H}_n^{-1}, s_{n+1}, y_{n+1})
\end{aligned}
$$

Quasi-Newton

$$
\begin{aligned}
&\textbf{QuasiNewton}(f, x_0, \mathbf{H}_0^{-1}, \text{QuasiUpdate}) : \\
&\quad \text{For } n = 0, 1, \ldots \text{ (until converged)} : \\
&\qquad // \text{ Compute search direction and step-size} \\
&\qquad d = \mathbf{H}_n^{-1} \mathbf{g}_n \\
&\qquad \alpha \leftarrow \min_{\alpha \geq 0} f(x_n - \alpha d) \\
&\qquad x_{n+1} \leftarrow x_n - \alpha d \\
&\qquad // \text{ Store the input and gradient deltas} \\
&\qquad \mathbf{g}_{n+1} \leftarrow \nabla f(x_{n+1}) \\
&\qquad s_{n+1} \leftarrow x_{n+1} - x_n \\
&\qquad y_{n+1} \leftarrow \mathbf{g}_{n+1} - \mathbf{g}_n \\
&\qquad // \text{ Update inverse hessian} \\
&\qquad \mathbf{H}_{n+1}^{-1} \leftarrow \text{QuasiUpdate}(\mathbf{H}_n^{-1}, s_{n+1}, y_{n+1})
\end{aligned}
$$

We've assumed that QuasiUpdate only requires the former inverse hessian estimate as well tas the input and gradient differences (s_n and y_n respectively). Note that if QuasiUpdate just returns $\nabla^2 f(x_{n+1})$, we recover exact NewtonRaphson.

Behave like a Hessian

What form should $\mathrm{QuasiUpdate}$ take? Well, if we have $\mathrm{QuasiUpdate}$ always return the identity matrix (ignoring its inputs), then this corresponds to simple gradient descent, since the search direction is always ∇f_n. While this actually yields a valid procedure which will converge to x^* for convex f, intuitively this choice of $\mathrm{QuasiUpdate}$ isn't attempting to capture second-order information about f.

Let's think about our choice of $\mathbf{H}_n$ as an approximation for f near x_n:

$$h_n(d) = f(x_n) + d^T \mathbf{g}_n + \frac{1}{2} d^T \mathbf{H}_n d$$

Secant Condition

A good property for $h_n(d)$ is that its gradient agrees with f at x_n and x_{n-1}. In other words, we'd like to ensure:

$$\nabla h_n(x_n) = \mathbf{g}_n$$
$$\nabla h_n(x_{n-1}) = \mathbf{g}_{n-1}$$

Using both of the equations above:

$$\nabla h_n(x_n) - \nabla h_n(x_{n-1}) = \mathbf{g}_n - \mathbf{g}_{n-1}$$

Secant Condition

Using both of the equations above:

$$\nabla h_n(x_n) - \nabla h_n(x_{n-1}) = \mathbf{g}_n - \mathbf{g}_{n-1}$$

Using the gradient of $h_{n+1}(\cdot)$ and canceling terms we get

$$\mathbf{H}_n(x_n - x_{n-1}) = (\mathbf{g}_n - \mathbf{g}_{n-1})$$

This yields the so-called "secant conditions" which ensures that $\mathbf{H}_{n+1}$ behaves like the Hessian at least for the diference $(x_n - x_{n-1})$. Assuming $\mathbf{H}_n$ is invertible (which is true if it is psd), then multiplying both sides by $\mathbf{H}_n^{-1}$ yields

$$\mathbf{H}_n^{-1}\mathbf{y}_n = \mathbf{s}_n$$

where $\mathbf{y}_{n+1}$ is the difference in gradients and $\mathbf{s}_{n+1}$ is the difference in inputs.

Symmetric

Recall that the a hessian represents the matrix of 2nd order partial derivatives: $\mathbf{H}^{(i,j)} = \partial f / \partial x_i \partial x_j$. The hessian is symmetric since the order of differentiation doesn't matter.

The BFGS Update

Intuitively, we want $\mathbf{H}_n$ to satisfy the two conditions above:

- Secant condition holds for $\mathbf{s}_n$ and $\mathbf{y}_n$
- $\mathbf{H}_n$ is symmetric

Given the two conditions above, we'd like to take the most conservative change relative to $\mathbf{H}_{n-1}$. This is reminiscent of the MIRA update, where we have conditions on any good solution but all other things equal, want the 'smallest' change.

$$\begin{aligned} \min_{\mathbf{H}^{-1}} \quad & \|\mathbf{H}^{-1} - \mathbf{H}_{n-1}^{-1}\|^2 \\ \text{s.t.} \quad & \mathbf{H}^{-1}\mathbf{y}_n = \mathbf{s}_n \\ & \mathbf{H}^{-1} \text{ is symmetric} \end{aligned}$$

The norm used here $\|\cdot\|$ is the weighted frobenius norm.[4] The solution to this optimization problem is given by

$$\mathbf{H}_{n+1}^{-1} = (I - \rho_n y_n s_n^T)\mathbf{H}_n^{-1}(I - \rho_n s_n y_n^T) + \rho_n s_n s_n^T \qquad \text{where } \rho_n = (y_n^T s_n)^{-1}.$$

对称秩一校正(SR1校正)

希望有$H_{k+1} = H_k + E_k$, 其中E_k为一个低阶矩阵。在秩一校正情形下，

$E_k = uv^T$.

根据拟牛顿条件有,$\left(H_k + uv^T\right)y_k = s_k \Rightarrow \left(v^T y_k\right)u = s_k - H_k y_k$, 故

u必在方向$s_k - H_k y_k$上。

取$u = \dfrac{1}{v^T y_k}\left(s_k - H_k y_k\right)$, 因为$Hesse$为对称阵，故取$v = \left(s_k - H_k y_k\right)$。

那么$H_{k+1} = H_k + \dfrac{\left(s_k - H_k y_k\right)\left(s_k - H_k y_k\right)^T}{\left(s_k - H_k y_k\right)^T y_k}$, 称为对称秩一校正($SR1$校正).

性质

1。对于二次函数，不需要精确一维搜索，具有n步终止性质，$H_n = G^{-1}$

2。不能保持正定性，仅当$\left(s_k - H_k y_k\right)^T y_k > 0$时。但这个条件往往很难满足。这使得$SR1$校正在应用中受到限制。

DFP校正

设秩二校正为$H_{k+1} = H_k + auu^T + bvv^T$.

若要拟牛顿条件$\left(H_k + auu^T + bvv^T\right)y_k = s_k$成立,

对于u,v一个明显的取法为$u = s_k,\ \ v = H_k y_k$。

同时令：$au^T y_k = 1, bv^T y_k = -1,$可知(*)恒成立。

那么$a = \dfrac{1}{s_k^T y_k}, b = -\dfrac{1}{y_k^T H_k y_k}$

那么$H_{k+1} = H_k + \dfrac{s_k^T s_k}{s_k^T y_k} - \dfrac{H_k y_k y_k^T H_k^T}{y_k^T H_k y_k},$称为*DFP*校正.

性质

1。对于二次函数，精确一维搜索，具有n步终止性质，$H_n = G^{-1}$

2。对于二次函数，$H_0 = I$时，产生共轭方向与共轭梯度。

3。校正保持正定性，下降性质成立。

4。具有超线性收敛。

5。采用精确线性搜索时，对于凸函数，总体收敛。

The BFGS Update

$$H_{i+1} = H_i + \frac{s_i s_i^T}{s_i^T q_i} - \frac{H_i q_i q_i^T H_i}{q_i^T H_i q_i}$$

$$+\left[\nabla f(X_{i+1}) - \nabla f(X_i)\right]^T H_i \left[\nabla f(X_{i+1}) - \nabla f(X_i)\right] w w^T \quad \cdots[14]$$

where

$$w = \frac{s_i}{s_i^T q_i} - \frac{H_i q_i}{q_i^T H_i q_i}$$

codelast.com

$$= H_i + \frac{X_{i+1} - X_i}{(X_{i+1} - X_i)^T \left[\nabla f(X_{i+1}) - \nabla f(X_i)\right]}$$

$$- \frac{H_i \left[\nabla f(X_{i+1}) - \nabla f(X_i)\right]}{\left[\nabla f(X_{i+1}) - \nabla f(X_i)\right]^T H_i \left[\nabla f(X_{i+1}) - \nabla f(X_i)\right]} \quad \cdots[15]$$

codelast.com

http://www.codelast.com/?p=2780#more-2780

Broyden, Fletcher, Goldfarb, Shanno

The BFGS Update

- $\mathbf{H}_{n+1}^{-1}$ is positive definite (psd) when $\mathbf{H}_{n}^{-1}$ is. Assuming our initial guess of $\mathbf{H}_0$ is psd, it follows by induction each inverse Hessian estimate is as well. Since we can choose any $\mathbf{H}_0^{-1}$ we want, including the I matrix, this is easy to ensure.

- The above also specifies a recurrence relationship between $\mathbf{H}_{n+1}^{-1}$ and $\mathbf{H}_{n}^{-1}$. We only need the history of s_n and y_n to re-construct $\mathbf{H}_{n}^{-1}$.

The last point is significant since it will yield a procedural algorithm for computing $\mathbf{H}_n^{-1} d$, for a direction d, without ever forming the $\mathbf{H}_n^{-1}$ matrix. Repeatedly applying the recurrence above we have

$$
\begin{aligned}
&\textbf{BFGSMultiply}(\mathbf{H}_0^{-1}, \{s_k\}, \{y_k\}, d) : \\
&\quad r \leftarrow d \\
&\quad // \text{ Compute right product} \\
&\quad \text{for } i = n, \ldots, 1 : \\
&\qquad \alpha_i \leftarrow \rho_i s_i^T r \\
&\qquad r \leftarrow r - \alpha_i y_i \\
&\quad // \text{ Compute center} \\
&\quad r \leftarrow \mathbf{H}_0^{-1} r \\
&\quad // \text{ Compute left product} \\
&\quad \text{for } i = 1, \ldots, n : \\
&\qquad \beta \leftarrow \rho_i y_i^T r \\
&\qquad r \leftarrow r + (\alpha_{n-i+1} - \beta) s_i \\
&\quad \text{return } r
\end{aligned}
$$

Since the only use for $\mathbf{H}_n^{-1}$ is via the product $\mathbf{H}_n^{-1} \mathbf{g}_n$, we only need the above procedure to use the BFGS approximation in $\mathrm{QuasiNewton}$.

L-BFGS:
BFGS on a memory budget

The BFGS quasi-newton approximation has the benefit of not requiring us to be able to analytically compute the Hessian of a function. However, we still must maintain a history of the s_n and y_n vectors for each iteration. Since one of the core-concerns of the $\mathbf{NewtonRaphson}$ algorithm were the memory requirements associated with maintaining an Hessian, the BFGS Quasi-Newton algorithm doesn't address that since our memory use can grow without bound.

The L-BFGS algorithm, named for *limited* BFGS, simply truncates the $\mathbf{BFGSMultiply}$ update to use the last m input differences and gradient differences. This means, we only need to store $s_n, s_{n-1}, \ldots, s_{n-m-1}$ and $y_n, y_{n-1}, \ldots, y_{n-m-1}$ to compute the update. The center product can still use any symmetric psd matrix $\mathbf{H}_0^{-1}$, which can also depend on any $\{s_k\}$ or $\{y_k\}$.

L-BFGS:
BFGS on a memory budget

Two loops approach $y_k = g_{k+1} - g_k \qquad \rho_k = \dfrac{1}{y_k^{\mathrm{T}} s_k}$

$q = g_k$

For $i = k-1, k-2, \ldots, k-m$

$\alpha_i = \rho_i s_i^{\mathrm{T}} q$

$q = q - \alpha_i y_i$

$H_k = y_{k-1}^{\mathrm{T}} s_{k-1} / y_{k-1}^{\mathrm{T}} y_{k-1}$

$z = H_k q$

For $i = k-m, k-m+1, \ldots, k-1$

$\beta_i = \rho_i y_i^{\mathrm{T}} z$

$z = z + s_i(\alpha_i - \beta_i)$

Stop with $H_k g_k = z$

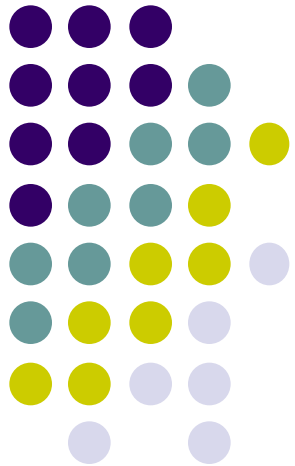
http://en.wikipedia.org/wiki/Limited-memory_BFGS

L-BFGS variants

There are lots of variants of L-BFGS which get used in practice. For non-differentiable functions, there is an othant-wise varient which is suitable for training L_1 regularized loss.

One of the main reasons to *not* use L-BFGS is in very large data-settings where an online approach can converge faster. There are in fact online variants of L-BFGS, but to my knowledge, none have consistently out-performed SGD variants (including AdaGrad or AdaDelta) for sufficiently large data sets.

2. 约束非线性最优化



约束优化最优性条件

约束最优化问题通常写为

$$\min f(\mathbf{x})$$

$$\text{s.t. } c_i(\mathbf{x})=0, i\in E=\{1,\ldots,m_e\},$$

$$c_i(\mathbf{x})\geq 0, i\in I=\{m_e+1,\ldots,m\}$$

在$\mathbf{x}^*$处的非积极约束

设$\mathbf{x}^*$为一个局部极小点，若不等式约束i_0有，$c_{i0}(\mathbf{x}^*)>0$，则可将第i_0个约束去掉，且$\mathbf{x}^*$仍然是去掉第i_0个约束条件的问题的局部极小点。称约束c_{i0}在$\mathbf{x}^*$处是非积极的。

定义：$I(\boldsymbol{x}) = \{\, i \mid c_i(\boldsymbol{x}) \leq 0,\ i \in I\}; A(\boldsymbol{x}) = E \cup I(\boldsymbol{x})$ 为$\boldsymbol{x}$点处的积极集合。

一阶最优性条件

Kuhn - Tucker必要条件：

若x^*是问题P的一个局部极小点，如果$\nabla c_i(x^*)(i \in E \cup I(x^*))$线性无关，则必存在$\lambda_i^*(i=1,\ldots,m)$，使得

$$\nabla f(x^*) = \sum_{i=1}^{m} \lambda_i^* \nabla c_i(x^*), \qquad (*)$$

$$\lambda_i^* \geq 0, \quad \lambda_i^* c_i(x^*) = 0, i \in I. \qquad (**)$$

满足上述两式的点称为K - T点。与该定理联系密切的是Lagrange函数：

$$L(x,\lambda) = f(x) - \lambda^T c(x).$$

则(*)条件等价于$\nabla_x L = 0$。λ称为Lagrange乘子。

二阶必要条件

定义：设x^*是$K-T$点，λ^*称为相应的$Lagrange$乘子,若存在序列$\{d_k\}$和$\{\delta_k>0\}$使得

$$x^*+\delta_k d_k \in X$$

$$\sum_{i=1}^{m}\lambda_i^* c_i\left(x^*+\delta_k d_k\right)=0, i\in I.$$

且有$d_k \to d$，$\delta_k \to 0$，则称d为x^*处的序列零约束方向。在x^*处的所有序列零约束方向的集合记为$S\left(x^*,\lambda^*\right)$.

二阶必要性条件：

设x^*为局部极小点，λ^*称为相应的$Lagrange$乘子，则必有

$d^T\nabla_{xx}^2 L\left(x^*,\lambda^*\right)d\geq 0, \forall d\in S\left(x^*,\lambda^*\right)$. 其中$L(x,\lambda)$为$Lagrange$函数。

稍加强可得充分性条件：

设x^*为$K-T$点，λ^*称为相应的$Lagrange$乘子，若

$d^T\nabla_{xx}^2 L\left(x^*,\lambda^*\right)d>0, \forall 0\neq d\in S\left(x^*,\lambda^*\right)$, 则$x^*$为局部严格极小点。

可行方向法：基本想法

- 可行方向法即要去每次迭代产生的点$\boldsymbol{x}_k$都是约束优化问题的可行点。
- 关键在于每一步寻找可行下降方向：

$$\mathbf{d}^T\nabla f\left(\mathbf{x}_k\right)<0,\mathbf{d}\in FD(\mathbf{x}_k,X).$$

可行方向：设$\bar{x}\in X,0\neq d\in R^n$,如存在$\delta>0$使得$\bar{x}+td\in X$,则称$d$为$\bar{x}$处的可行方向。$X$在$\bar{x}$处的所有可行方向集合记为$FD(\bar{x},X)$。

变量消去法

考虑等式约束问题

$$\min f(\mathbf{x}),$$
$$s.t.\ \ c(\mathbf{x}) = 0$$

设有变量分解$\mathbf{x} = \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{bmatrix}$,其中$x_B \in R^m$，$x_N \in R^{n-m}$.

则$c(x) = 0$可改写为

$$c\left(x_B, x_N\right) = 0. \qquad (1)$$

假定可以从(1)解出$x_B = \phi\left(x_N\right)$，则原问题等价于

$$\min_{x_N \in R^{n-m}} f\left(x_B, x_N\right) = f\left(\phi\left(x_N\right), x_N\right) = \tilde{f}\left(x_N\right).$$

称$\tilde{g}\left(x_N\right) = \nabla_{x_N} \tilde{f}\left(x_N\right)$为既约梯度。

变量消去法

不难验证

$$\tilde{g}\left(x_N\right)=\frac{\partial f\left(x_B,x_N\right)}{\partial x_N}+\frac{\partial x_B^T}{\partial x_N}\frac{\partial f\left(x_B,x_N\right)}{\partial x_B},$$

从(1)可得

$$\frac{\partial x_B^T}{\partial x_N}\frac{\partial c\left(x_B,x_N\right)^T}{\partial x_B}+\frac{\partial c\left(x_B,x_N\right)^T}{\partial x_N}=0.$$

假设$\frac{\partial c\left(x_B,x_N\right)^T}{\partial x_B}$非奇异，可得到

$$\tilde{g}\left(x_N\right)=\frac{\partial f\left(x_B,x_N\right)}{\partial x_N}-\frac{\partial c\left(x_B,x_N\right)^T}{\partial x_N}\left(\frac{\partial c\left(x_B,x_N\right)^T}{\partial x_B}\right)^{-1}\frac{\partial f\left(x_B,x_N\right)}{\partial x_B}$$

变量消去法

令$\lambda = \left(\frac{\partial c\left(x_B, x_N\right)^T}{\partial x_B} \right)^{-1} \frac{\partial f\left(x_B, x_N\right)}{\partial x_B}$,则发现可以将既约梯度写成$L\arg range$函数在既约空间上的梯度

$$\tilde{g}\left(x_N\right) = \frac{\partial}{\partial x_N}\left[f\left(x\right) - \lambda^T c\left(x\right)\right]$$

或者说，有

$$\nabla_x L\left(x, \lambda\right) = \begin{bmatrix} 0 \\ \tilde{g}\left(x_N\right) \end{bmatrix}$$

变量消去法

故既约梯度可看作*Lagrange*函数之梯度的非零部分。

利用既约梯度，可以构造无约束优化问题的线搜索方向。

例如可取最速下降方向

$$\tilde{d}_k = -\tilde{g}\left(\left(x_N\right)_k\right),$$

或拟牛顿方向

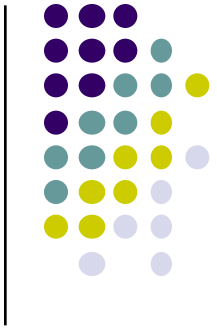
$$\tilde{d}_k = -H_k\tilde{g}\left(\left(x_N\right)_k\right)。$$

在无约束问题上作线性搜索，等价于对原目标函数 $f(x)$在曲线

$$c\left(x_B,\left(x_N\right)_k + \alpha\tilde{d}_k\right) = 0 \qquad (2)$$

上作曲线搜索。因为$\phi(x)$的解析表达式并不知道，故作一维搜索时，每个试探步长$\alpha > 0$都要用(2)来求解x_B.

变量消去法



1。给出可行点$x_1, \varepsilon \geq 0, k = 1$

2。计算$\dfrac{\partial c\left(x_k\right)^T}{\partial x} = \begin{bmatrix} A_B \\ A_N \end{bmatrix}$，其中划分使得$A_B$非奇异；

计算$\lambda = \left(\dfrac{\partial c\left(x_B, x_N\right)^T}{\partial x_B} \right)^{-1} \dfrac{\partial f\left(x_B, x_N\right)}{\partial x_B}$及$\tilde{g}\left(\left(x_N\right)_k\right) = \dfrac{\partial}{\partial x_N}\left[f(x) - \lambda^T c(x)\right]$。

3。如果$\left\|\tilde{g}_k\right\| \leq \varepsilon$，则停止；否则利用某种方式产生下降方向$\tilde{d}_k$,即使得

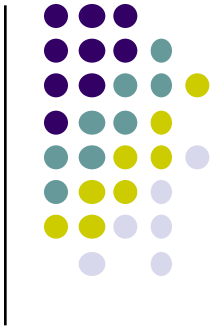
$\tilde{d}_k \tilde{g}_k < 0;$

4。$\min\limits_{\alpha \geq 0} f\left(\phi\left(\left(x_N\right)_k + \alpha \tilde{d}_k\right), \left(x_N\right)_k + \alpha \tilde{d}_k\right)$进行线性搜索给出$\alpha_k > 0$,令

$x_{k+1} = \left(\phi\left(\left(x_N\right)_k + \alpha_k \tilde{d}_k\right), \left(x_N\right)_k + \alpha_k \tilde{d}_k\right), k = k + 1,$

转2步。

广义消去法



考虑更一般的形式。任意非奇异矩阵$S \in R^{n\times n}$，以及变量替换

$x = Sw$

对w进行变量分离$w = \begin{bmatrix} w_B \\ w_N \end{bmatrix}$。利用约束条件$c(x) = 0$进行变量消去

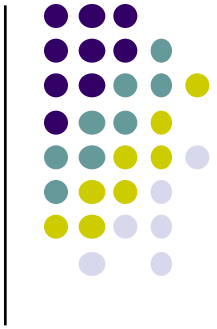
得到$w_B = \phi(w_N)$。于是原问题等价于

$$\min_{w_N \in R^{n-m}} f(S_B w_B + S_N w_N) = \tilde{f}(w_N)$$

可直接计算得到$\nabla_{w_n} \tilde{f}(w_N) = \tilde{g}((x_N)_k) = (S_k)_N \left[\nabla f(x) - \nabla c(x)^T \lambda\right]$，

其中λ满足$(S_k)_B \left[\nabla f(x) - \nabla c(x)^T \lambda\right] = 0$。

广义消去法



1。给出可行点$x_1, \varepsilon \geq 0, k=1$

2。以某种方式构造一非奇异矩阵S_k,且有划分$S_k=\left[(S_k)_B,(S_k)_N\right]$,使得$(S_k)_B^T \dfrac{\partial c(x_k)^T}{\partial x}$非奇异;

根据$(S_k)_B\left[\nabla f(x)-\nabla c(x)^T\lambda\right]=0$计算$\lambda$，以及计算$\tilde{g}\left((x_N)_k\right)=(S_k)_N\left[\nabla f(x)-\nabla c(x)^T\lambda\right]$。

3。如果$\|\tilde{g}_k\| \leq \varepsilon$，则停止；否则利用某种方式产生下降方向$\tilde{d}_k$,即使得

$$\tilde{d}_k\tilde{g}_k<0;$$

4。$\min\limits_{\alpha\geq 0} f\left((S_k)_B\phi\left((w_k)_N+\alpha\tilde{d}_k\right),(S_k)_B\left((x_k)_N+\alpha\tilde{d}_k\right)\right)$进行线性搜索给出$\alpha_k>0$,令

$$x_{k+1}=\left((S_k)_B\phi\left((w_k)_N+\alpha_k\tilde{d}_k\right),(S_k)_B\left((x_k)_N+\alpha_k\tilde{d}_k\right)\right), k=k+1,$$

转2步。

若$S_k=I$，广义消去法就是变量消去法。

投影梯度法

广义消去法每次迭代的变量增量$x_{k+1}-x_k$由两部分组成，

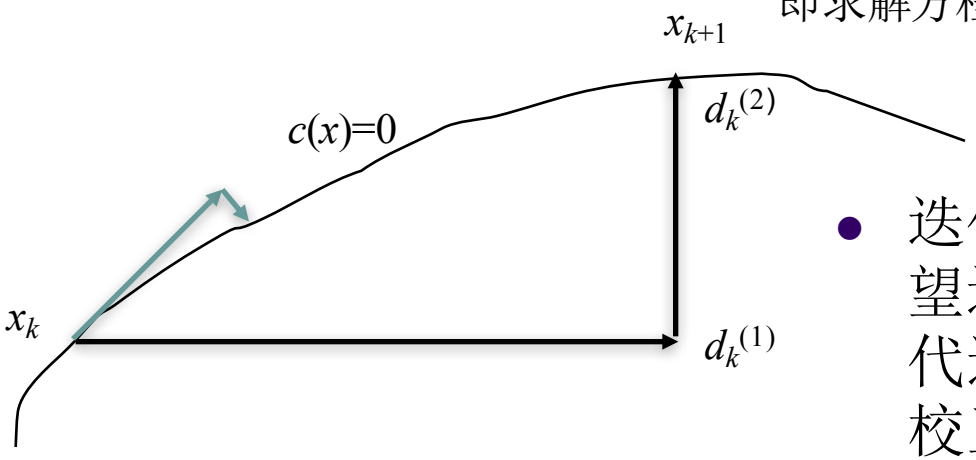
$x_{k+1}=x_k+d_k^{(1)}+d_k^{(2)}$,其中

$d_k^{(1)}=\alpha_k(S_k)_N\tilde{d}_k, d_k^{(2)}=(S_k)_B\left[\varphi\left((w_k)_N+\alpha_k\tilde{d}_k\right)-(w_k)_B\right]$

迭代过程是先得到$d_k^{(1)}$，再在$d_k^{(2)}$方向上迭代得到正确的步长。

在$d_k^{(2)}$方向上迭代的过程就是利用$c(x)=0$求解w_B的过程。

即求解方程$c\left((S_k)_B w_B+(S_k)_N\left[(w_k)_N+\alpha\tilde{d}_k\right]\right)=0$



- 迭代方式存在不合理之处。本来希望迭代点都在可行域上，但具体迭代过程却是先远离可行域，然后再校正回可行域中。
- 希望离开程度尽可能小？沿线性化方向。迭代过程可以更快的收敛。

定义 5.1 设矩阵 $P \in R^{n\times n}$。若 $P^T = P, P^2 = P$，则称 P 为投影矩阵。

设矩阵 $A \in R^{l\times n}$ 行满秩，记 A 的零空间为 $L_A = \{\boldsymbol{x} \in R^n \mid A\boldsymbol{x} = \boldsymbol{0}\}$，$L_A$ 的正交空间为 $L_A{}^{\perp} = \{A^T\boldsymbol{y} \mid \boldsymbol{y} \in R^l\}$。对 $\boldsymbol{x} \in R^n$ 进行正交分解 $\boldsymbol{x} = \boldsymbol{x}^1 + \boldsymbol{x}^2$，使 $\boldsymbol{x}^1 \in L_A$，$\boldsymbol{x}^2 \in L_A{}^{\perp}$，则 $\boldsymbol{x}^1 = P_A\boldsymbol{x}$，其中 $P_A = I - A^T(AA^T)^{-1}A$。则 P_A 是投影矩阵，称为 A 的投影矩阵。当 $\boldsymbol{x} \in L_A$ 时，$P_A\boldsymbol{x} = \boldsymbol{x}$，$\boldsymbol{x} \in L_A{}^{\perp}$ 时，$P_A\boldsymbol{x} = \boldsymbol{0}$。

线性化可行方向：

设$x^* \in$可行域$X, d \in R^n$

$d^T \nabla c_i(x^*) = 0, i \in E; d^T \nabla c_i(x^*) \geq 0, i \in I(x^*)$

则d为X的x^*处的线性化可行方向。

为使$d_k^{(1)}$沿线性化可行方向，应选取S_k使得$(S_k)_N^T \nabla c(x_k)^T = 0.$

设$A_k = \nabla c(x_k)^T$,则由$(S_k)_N$的列向量所张成的子空间就是A_k^T的零空间。

而另一方面，$P = I - A_k (A_k^T A_k)^{-1} A_k^T$为到该零空间的投影算子。

当在广义消去法中用最速下降法时，可得$d_k^{(1)} = -\alpha_k (S_k)_N (S_k)_N^T \nabla f(x_k)$,

因为$d_k^{(1)}$为线性可行方向，故$d_k^{(1)}$在A_k^T的零空间中，进而可知$(S_k)_N (S_k)_N^T$为投影算子P。

在计算中，可利用A_k的QR分解，

$$A_k = QR = \begin{bmatrix} Y_k & Z_k \end{bmatrix} \begin{bmatrix} R_k \\ 0 \end{bmatrix}, A_k \in R^{n \times m}, Q$$为正交阵，$Y_k \in R^{n \times m}$，$Z_k \in R^{n \times (n-m)}$,

$R_k \in R^{m \times m}$可逆上三角矩阵。可取$(S_k)_N = Z_k, (S_k)_B = I$。

若将搜索方向换成$d_k = -Z_k z_k, z_k \in R^{n-m}$,且满足$z_k^T \tilde{g}_k < 0$，则算法是一个一般形式的线性化可行方向法，简称可行方向法。

罚函数法

- 利用目标函数和约束函数构造具有“罚性质”的函数

$$P(\mathbf{x})= P(f(\mathbf{x}), c(\mathbf{x}))$$

所谓的罚性质，即要求对于可行点$P(\mathbf{x})=f(\mathbf{x})$, 当约束条件破坏很大时，$P(\mathbf{x})>>f(\mathbf{x})$

定义约束违反度函数：

$$C^{(-)}(\mathbf{x})=(c_1^{(-)}(\mathbf{x}),\ldots,c_m^{(-)}(\mathbf{x})),$$

其中：

$$c_i^{(-)}(\mathbf{x})=c_i(\mathbf{x}), \mathrm{i}=1,\ldots,m_e;$$

$$c_i^{(-)}(\mathbf{x})=\min\{c_i(\mathbf{x}), 0\}, \mathrm{i}=m_e+1,\ldots,m$$

罚函数法

罚函数一般可表示为目标函数与一项与$c(x)$有关的“罚项”之和，即

$$P(x) = f(x) + h(c^{(-)}(x))$$

罚项是定义在R^m上的函数，满足$h(0) = 0, \lim_{\|c\| \to \infty} h(c) = +\infty$

最早的罚函数是*Courant*罚函数，定义如下

$P(x) = f(x) + \sigma \left\| c^{(-)}(x) \right\|_2^2, \sigma > 0$是一正常数，罚因子。

事实上，更一般的可定义为

$$P(x) = f(x) + \sigma \left\| c^{(-)}(x) \right\|^{\alpha}, \quad \alpha > 0.$$

罚函数法

一般定义为

$$P(x) = f(x) + \sigma \left\| c^{(-)}(x) \right\|^{\alpha}, \quad \alpha > 0.$$

若罚函数在可行域边界上取值为无穷，则称为内点罚函数。

内点罚函数仅适合不等式约束问题。

常见有倒数罚函数和对数罚函数：

$$P(x) = f(x) + \sigma^{-1} \sum_{i=1}^{m} \frac{1}{c_i(x)}, \quad P(x) = f(x) + \sigma^{-1} \sum_{i=1}^{m} \log c_i(x)$$

内点罚函数在可行域的边界上形成一堵无穷高的“障碍墙”，

所以也称为障碍罚函数。

罚函数法的基本点是：

每次迭代（求解一个无约束优化问题）增加罚函数因子，直到使$\left\| c^{(-)}(x) \right\|$缩小到给定误差。

罚函数法

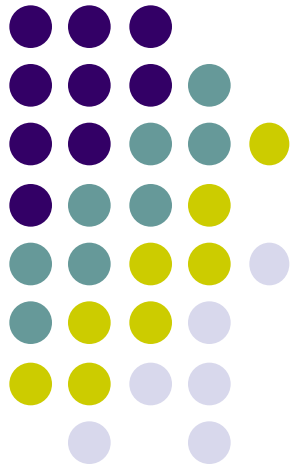
设x^*为原问题的$K-T$点，而x^*一般不是*Courant*函数的稳定点。为了克服这一缺点，引入参数$\theta=(\theta_1,\ldots,\theta_m)$,其中$\theta_i \geq 0 (i=m_e+1,\ldots,m)$。则

$$P(x)=f(x)+\sum_{i=1}^{m_e}\left[-\lambda_i c_i(x)+\frac{1}{2}\sigma_i(c_i(x))^2\right]+$$

$$\sum_{i=m_e+1}^{m}\begin{cases}-\lambda_i c_i(x)+\frac{1}{2}\sigma_i(c_i(x))^2, & \text{如} c_i(x)<\lambda_i/\sigma_i \\ -\frac{1}{2}\lambda_i^2/\sigma_i, & \text{否则}\end{cases}$$

其中$\lambda_i=\sigma_i\theta_i$

2. 二次规划



非线性最优化

- 最优化的问题的一般形式为

$$\text{Min } f(\mathbf{x}) \quad \text{s.t. } \mathbf{x} \in X$$

$f(\mathbf{x})$为目标函数，$X \in E^n$为可行域。
如$X = E^n$，则以上最优化问题为**无约束最优化**问题。

约束最优化问题通常写为

$$\begin{aligned} &\text{Min } f(\mathbf{x}) \\ &\text{s.t. } c_i(\mathbf{x}) = 0, \ i \in E, \\ &\qquad c_i(\mathbf{x}) \geq 0, \ i \in I, \end{aligned}$$

其中E, I分别为等式约束的指标集和不等式约束的指标集，$c_i(\mathbf{x})$是约束函数。

非线性优化中的概念

- 可行点，可行域
- 极小，全局极小（总体极小点），全局严格极小，局部极小，局部严格极小
- 积极与非积极，积极约束，非积极约束
- 可行方向集，线性可行方向集，序列可行方向集

- Farkas引理与K-T条件

- 以上参见《最优化理论与方法》第八章

无约束二次最优化

- $\min f(\mathbf{x}) = \frac{1}{2}\,\mathbf{x}^{\mathrm{T}} H\mathbf{x} + \mathbf{c}^{\mathrm{T}}\mathbf{x},\ \mathbf{x} \in R^n$

H是对称矩阵

基本解法：求导然后找局部极值。

二次规划的一般形式

$$\min\ f(\mathbf{x}) = \tfrac{1}{2}\,\mathbf{x}^{\mathrm{T}} H\mathbf{x} + \mathbf{c}^{\mathrm{T}}\mathbf{x},\ \mathbf{x} \in R^n$$

$$\text{s.t. } A\mathbf{x} \le \mathbf{b} \qquad (1)$$

- 当H为对称矩阵时，被称为**二次规划**(Quadratic Programming，记作QP)。

- 特别，当H**正定**时，目标函数为凸函数，线性约束下可行域又是凸集。上式被称为**凸二次规划**。

二次规划的一般形式

$$\min\ f(\mathbf{x}) = \tfrac{1}{2}\,\mathbf{x}^{\mathrm{T}} H\mathbf{x} + \mathbf{c}^{\mathrm{T}}\mathbf{x},\ \mathbf{x}\in R^n$$

$$\text{s.t. } \boldsymbol{a}_i{}^{T}\boldsymbol{x} \geq b_i, i \in I, \qquad (1)$$

$$\boldsymbol{a}_i{}^{T}\boldsymbol{x} = b_i, i \in E.$$

- 当H为对称矩阵时，被称为二次规划(Quadratic Programming，记作QP)。

- 特别，当H正定时，目标函数为凸函数，线性约束下可行域又是凸集。上式被称为凸二次规划。

二次规划的性质

- QP是一种最简单的非线性规划。QP有如下良好的性质，当H是半正定时：
  - K-T条件不仅是最优解的必要条件，而且是充分条件；
  - 局部最优解就是全局最优解。

等式约束下的二次规划

$$\min\ f(\mathbf{x}) = \tfrac{1}{2}\,\mathbf{x}^{\mathrm{T}}H\mathbf{x} + \mathbf{c}^{\mathrm{T}}\mathbf{x},\ \mathbf{x} \in R^n$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b} \qquad (2)$$

求解方法：Lagrange乘子法，求解以下无约束二次最优化问题。

$$L(\mathbf{x}, \boldsymbol{\lambda}) = \tfrac{1}{2}\mathbf{x}^{\mathrm{T}}H\mathbf{x} + \mathbf{c}^{\mathrm{T}}\mathbf{x} + \boldsymbol{\lambda}^{\mathrm{T}}(A\mathbf{x} - \mathbf{b})$$

令$L(\mathbf{x}, \boldsymbol{\lambda})$对$\mathbf{x}$和$\boldsymbol{\lambda}$的导数为零，得线性方程组

$$H\mathbf{x} + \mathbf{c}^{\mathrm{T}} + A^{\mathrm{T}}\boldsymbol{\lambda} = 0$$

$$A\mathbf{x} - \mathbf{b} = \mathbf{0}$$

可解得$\mathbf{x}$，即为上式的解。

凸二次规划的有效集方法（1）

- 直观解释：将不起作用约束去掉，将起作用约束作为等式约束，通过解一系列等式约束的二次规划来实现不等式约束的优化。

- 基本原理：若$\mathbf{x}$为问题（1）的最优解，则它也是问题

$$\min \frac{1}{2}\mathbf{x}^{\mathrm{T}} H\mathbf{x} + \mathbf{c}^{\mathrm{T}}\mathbf{x}$$
$$\text{s.t. } \mathbf{a}_i^{\mathrm{T}}\mathbf{x} = b_i, i \in I \qquad (3)$$

- 的最优解，其中$\mathbf{a}_i$是A的第i行，I为起作用约束指标集（有效集）。
- 反之，若$\mathbf{x}$为（1）的可行解，又是（3）的K-T点，且相应的乘子$\lambda_i \geq 0$，则$\mathbf{x}$为（1）的最优解。

凸二次规划的有效集方法（2）

- 算法步骤（迭代法）：
  - 设当前迭代点为$\mathbf{x}_k$，它也是（1）的可行解。该点的有效集记作I_k，为寻求$\mathbf{x}_k$点的迭代方向$\mathbf{d}$，用乘子法求解

$$\min \frac{1}{2}(\mathbf{x}_k+\mathbf{d})^{\mathrm{T}} H(\mathbf{x}_k+\mathbf{d})+\mathbf{c}^{\mathrm{T}}(\mathbf{x}_k+\mathbf{d})$$

$$\text{s.t. } \mathbf{a}_i^{\mathrm{T}}\mathbf{d}=0, i\in I_k$$

  - 若所得最优值$\mathbf{d}_k=0$，则$\mathbf{x}_k$是（3）的最优解。
    - 为判断它是否（1）的最优解，考察对应于有效约束的乘子$\lambda_i \geq 0$是否成立。若成立，则$\mathbf{x}_k$是K-T点，由二次规划性质$\mathbf{x}_k$是（1）的最优解。

凸二次规划的有效集方法（3）

- 算法步骤（迭代法）：

$$\min \frac{1}{2}(\mathbf{x}_k + \mathbf{d})^{\mathrm{T}} H(\mathbf{x}_k + \mathbf{d}) + \mathbf{c}^{\mathrm{T}}(\mathbf{x}_k + \mathbf{d})$$

$$\text{s.t. } \mathbf{a}_i^{\mathrm{T}}\mathbf{d} = 0, i \in I_k$$

  - 若最优值$\mathbf{d}_k \neq 0$，则取$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{d}_k$，在$\mathbf{x}_{k+1}$为可行点的条件下确定$\mathbf{d}_k$方向的步长$\alpha_k$
    - 如果存在p不在I_k中，使得$\mathbf{a}_p \mathbf{x}_{k+1} = \mathrm{b}_p$，则将$p$加入有效集
  - 如果存在I_k中的指标q，使得$\lambda_i < 0$, 则$\mathbf{x}^{(k)}$不是最优解，从有效集中去掉q

可行步长的选取：阻塞约束

$$\alpha_k = \min\{1, \min_{i \notin I_k, \mathbf{a}_i^T \mathbf{d}_k < 0} \frac{b_i - \mathbf{a}_i^T \mathbf{x}_k}{\mathbf{a}_i^T \mathbf{d}_k}\}$$

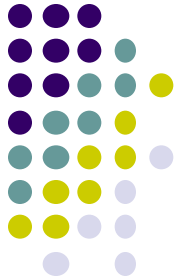
- $a_k = 1$时，对应约束集不影响，保持不变
- $a_k < 1$时，对应约束称为阻塞约束，此时沿着$\mathbf{d}_k$运动，会被不在指标集中的约束给阻塞了，约束集因此改变。

凸二次规划实例

- SVM …

- libSVM代码
  - http://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - 有效集方法

二次规划在python中的实现方法



http://cvxopt.org

- Version 1.2.0 includes:
- efficient Python classes for dense and sparse matrices (real and complex), with Python indexing and slicing and overloaded operations for matrix arithmetic
- an interface to most of the double-precision real and complex BLAS
- an interface to LAPACK routines for solving linear equations and least-squares problems, matrix factorizations (LU, Cholesky, LDL^T and QR), symmetric eigenvalue and singular value decomposition, and Schur factorization
- an interface to the fast Fourier transform routines from FFTW
- interfaces to the sparse LU and Cholesky solvers from UMFPACK and CHOLMOD
- routines for linear, second-order cone, and semi-definite programming problems
- routines for nonlinear convex optimization
- interfaces to the linear programming solver in GLPK, the semi-definite programming solver in DSDP5, and the linear, quadratic and second-order cone programming solvers in MOSEK
- a modeling tool for specifying convex piecewise-linear optimization problems.

更多凸二次规划

- 线性互补问题(LCP)，Lemke算法

- 可行方向法：
  - Zoutendijk可行方向法
  - Rosen梯度投影法
  - Wolfe既约梯度法
- 罚函数法
- 逐次二次规划法
- 信赖域法

Homework 05

- 实现SVM：
  - Kernel：线性核，指数核
  - 使用python
- 二次规划方法：
  - 有效集方法
  - 奖励 => 其他更高效的优化方法
- 5次编程作业，请统一提供一个实验报告
  - 格式与课程论文的要求相似

The End

- 数学学习是一种修行
- 大音希声，大象无形
  - 节选自《道德经》

无所得，即是得
以是得，无所得

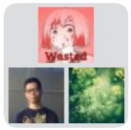
--《金刚经》

新浪微博：@浙大张宏鑫

微信公众号：

中国移动 4G 下午8:15 65%

群二维码名片



csmath2019课程群

该二维码7天内(4月23日前)有效，重新进入将更新