人工智能的数学基础

华东师范大学 数学科学学院 黎芳（教授）2019年9月4日

Chapter0 数学准备

Table of Contents

1 梯度下降法

梯度下降法（gradient descent）或最速下降法（steepest descent）是求解无约束优化问题的一种常用方法。

假设$f(x)$是$\mathbb{R}^n$上具有一阶连续偏导数的函数, 要求解的无约束最优化问题是

$$\min_{x\in\mathbb{R}^n} f(x)$$

梯度下降法是一种迭代算法. 选取适当的初值$x^{(0)}$，不断迭代，更新x的值，进行目标函数的极小化，直到收敛. 由于负梯度方向是使函数值下降最快的方向，在迭代的每一步，以负梯度方向更新x的值，从而达到减少函数值的目的。

由于$f(x)$具有一阶连续偏导数，若第k次迭代值为$x^{(k)}$，则可将$f(x)$在$x^{(k)}$附近进行一阶泰勒展开:

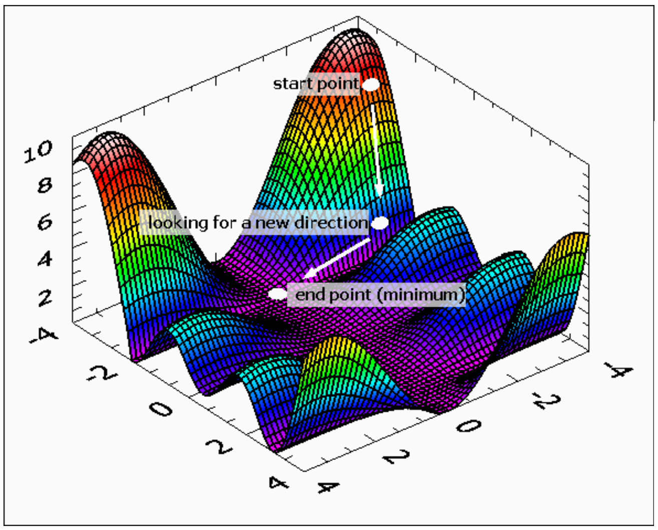
$$f(x) = f(x^{(k)}) + g_k^{\mathrm{T}}(x - x^{(k)}) \qquad (1)$$

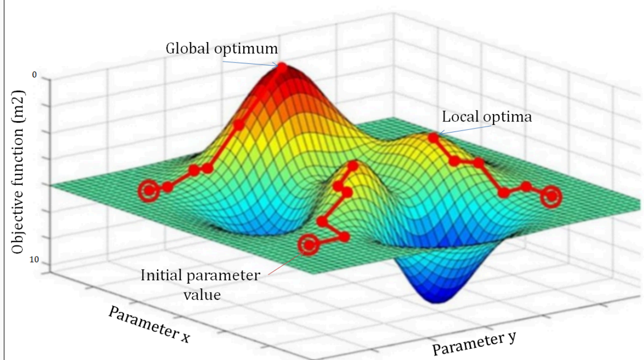
$g_k = g(x^{(k)}) = \nabla f(x^{(k)})$

$$x^{(k+1)} \leftarrow x^{(k)} + \lambda_k p_k$$

p_k--搜索方向，取负梯度方向$p_k = -\nabla f(x^{(k)})$；λ_k为搜索步长，最优步长可以由一维搜索确定

$$f\left(x^{(k)} + \lambda_k p_k\right) = \min_{\lambda \geq 0} f\left(x^{(k)} + \lambda p_k\right)$$





算法 **A.I** (梯度下降法)

输入: 目标函数$f(x)$, 梯度函数$g(x) = \nabla f(x)$, 计算精度ϵ;

输出: $f(x)$的极小点x^*.

- (1) 取初始值$x^{(0)} \in \mathbb{R}^n$, 置$k = 0$.
- (2) 计算$f(x^{(k)})$.
- (3) 计算梯度$g_k = g(x^{(k)})$, 当$\|g_k\| < \epsilon$时，停止迭代；否则，令$p_k = -g(x^{(k)})$, 求λ_k, 使$f(x^{(k)} + \lambda_k p_k) = \min_{\lambda \geq 0} f(x^{(k)} + \lambda p_k)$.
- (4) 置$x^{(k+1)} = x^{(k)} + \lambda_k p_k$, 计算$f(x^{(k+1)})$, 当$\|f(x^{(k+1)}) - f(x^{(k)})\| < \epsilon$或$\|x^{(k+1)} - x^{(k)}\| < \epsilon$时，停止迭代，令$x^* = x^{(k+1)}$.
- (5) 否则，置$k = k + 1$，转(3).

优点:

- (1) 方法简单，每迭代一次的工作量较小，存储量少.
- (2) 从一个不好的初始点出发，也能保证算法的收敛性.

缺点:

- 在极小点附近收敛的很慢。

梯度是函数的局部性质，从局部看在一点附近下降得快，但从总体上来看可能走许多弯路. 在相继两次迭代中，搜索方向正交. 因此，在最速下降法逼近极小点的路线是锯齿形的，并且越靠近极小点步长越小，即越走越慢.

最速下降法有着很好的整体收敛性，即使对很一般的目标函数，它也是整体收敛的。

收敛性定理：设 $f: \mathbf{R}^n \to \mathbf{R}^1$ 连续可微，若水平集 $L = \{x | f(x) \le f(x_0)\}$ 有界，令最速下降法产生的点列为 $\{x_k\}$，则

1. 对某个 k_0，$g(x_{k_0}) = 0$，算法在有限步迭代后停止，或者
2. 当 $k \to \infty$ 时，$g_k \to 0$，得到点列的任何极限点都是驻点.

若进一步假设 $f(x)$ 为凸函数，则应用最速下降法，或在有限步迭代后达到 $f(x)$ 的最小点，或者点列的任何极限点都是 $f(x)$ 的最小点.

目标函数凸时为全局最优解；非凸时容易陷入局部最优解.

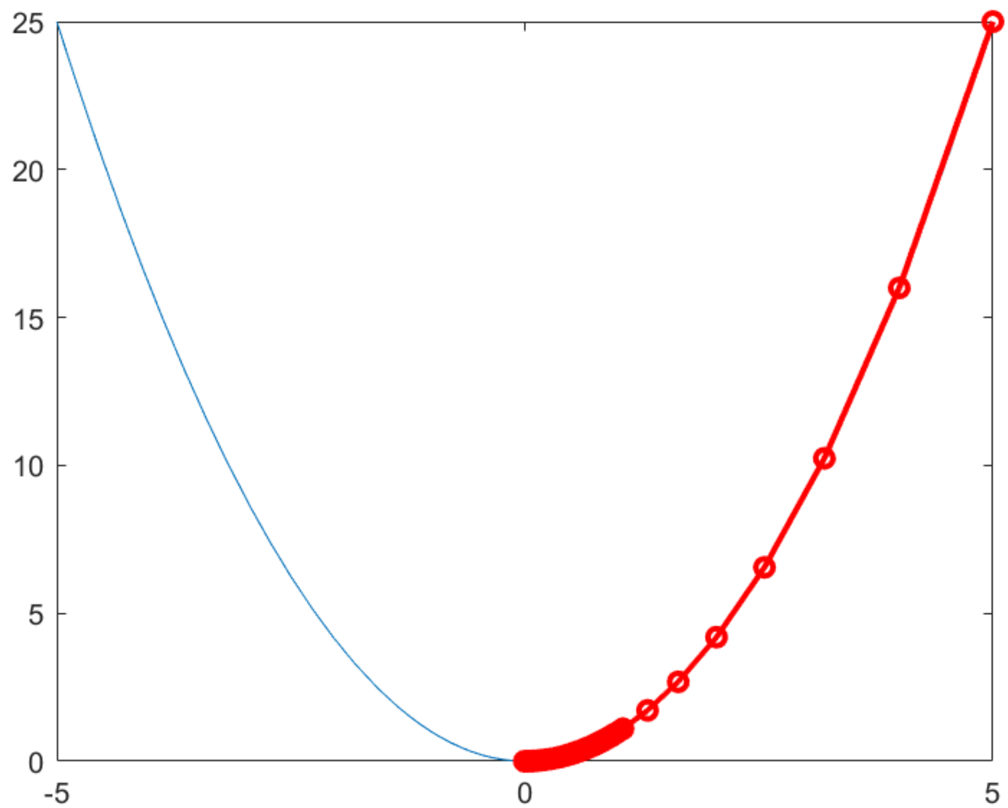
例1：求 $\min f(x) = x^2$

```
% gradient descent method
x0 = 5; % initial point
x = x0;
gradf = 2*x;
tol = 1e-5;% convergence tolerance
iter = 0;
dt = 0.1;
xs(1) = x;
figure,fplot(@(x)x.^2)
while norm(gradf)>tol
    iter = iter+1;
    x = x-dt*gradf;
    gradf = 2*x;
    fun_value = x^2;
    fprintf('iter_num = %3d  norm_grad = %2.6f  fun_value = %2.6f\n'...
        ,iter,norm(gradf),fun_value);
    xs(iter+1) = x;
end
```

```
iter_num =   1  norm_grad = 8.000000  fun_value = 16.000000
iter_num =   2  norm_grad = 6.400000  fun_value = 10.240000
iter_num =   3  norm_grad = 5.120000  fun_value = 6.553600
iter_num =   4  norm_grad = 4.096000  fun_value = 4.194304
iter_num =   5  norm_grad = 3.276800  fun_value = 2.684355
iter_num =   6  norm_grad = 2.621440  fun_value = 1.717987
iter_num =   7  norm_grad = 2.097152  fun_value = 1.099512
iter_num =   8  norm_grad = 1.677722  fun_value = 0.703687
```

```
iter_num =   9  norm_grad = 1.342177  fun_value = 0.450360
iter_num =  10  norm_grad = 1.073742  fun_value = 0.288230
iter_num =  11  norm_grad = 0.858993  fun_value = 0.184467
iter_num =  12  norm_grad = 0.687195  fun_value = 0.118059
iter_num =  13  norm_grad = 0.549756  fun_value = 0.075558
iter_num =  14  norm_grad = 0.439805  fun_value = 0.048357
iter_num =  15  norm_grad = 0.351844  fun_value = 0.030949
iter_num =  16  norm_grad = 0.281475  fun_value = 0.019807
iter_num =  17  norm_grad = 0.225180  fun_value = 0.012677
iter_num =  18  norm_grad = 0.180144  fun_value = 0.008113
iter_num =  19  norm_grad = 0.144115  fun_value = 0.005192
iter_num =  20  norm_grad = 0.115292  fun_value = 0.003323
iter_num =  21  norm_grad = 0.092234  fun_value = 0.002127
iter_num =  22  norm_grad = 0.073787  fun_value = 0.001361
iter_num =  23  norm_grad = 0.059030  fun_value = 0.000871
iter_num =  24  norm_grad = 0.047224  fun_value = 0.000558
iter_num =  25  norm_grad = 0.037779  fun_value = 0.000357
iter_num =  26  norm_grad = 0.030223  fun_value = 0.000228
iter_num =  27  norm_grad = 0.024179  fun_value = 0.000146
iter_num =  28  norm_grad = 0.019343  fun_value = 0.000094
iter_num =  29  norm_grad = 0.015474  fun_value = 0.000060
iter_num =  30  norm_grad = 0.012379  fun_value = 0.000038
iter_num =  31  norm_grad = 0.009904  fun_value = 0.000025
iter_num =  32  norm_grad = 0.007923  fun_value = 0.000016
iter_num =  33  norm_grad = 0.006338  fun_value = 0.000010
iter_num =  34  norm_grad = 0.005071  fun_value = 0.000006
iter_num =  35  norm_grad = 0.004056  fun_value = 0.000004
iter_num =  36  norm_grad = 0.003245  fun_value = 0.000003
iter_num =  37  norm_grad = 0.002596  fun_value = 0.000002
iter_num =  38  norm_grad = 0.002077  fun_value = 0.000001
iter_num =  39  norm_grad = 0.001662  fun_value = 0.000001
iter_num =  40  norm_grad = 0.001329  fun_value = 0.000000
iter_num =  41  norm_grad = 0.001063  fun_value = 0.000000
iter_num =  42  norm_grad = 0.000851  fun_value = 0.000000
iter_num =  43  norm_grad = 0.000681  fun_value = 0.000000
iter_num =  44  norm_grad = 0.000544  fun_value = 0.000000
iter_num =  45  norm_grad = 0.000436  fun_value = 0.000000
iter_num =  46  norm_grad = 0.000348  fun_value = 0.000000
iter_num =  47  norm_grad = 0.000279  fun_value = 0.000000
iter_num =  48  norm_grad = 0.000223  fun_value = 0.000000
iter_num =  49  norm_grad = 0.000178  fun_value = 0.000000
iter_num =  50  norm_grad = 0.000143  fun_value = 0.000000
iter_num =  51  norm_grad = 0.000114  fun_value = 0.000000
iter_num =  52  norm_grad = 0.000091  fun_value = 0.000000
iter_num =  53  norm_grad = 0.000073  fun_value = 0.000000
iter_num =  54  norm_grad = 0.000058  fun_value = 0.000000
iter_num =  55  norm_grad = 0.000047  fun_value = 0.000000
iter_num =  56  norm_grad = 0.000037  fun_value = 0.000000
iter_num =  57  norm_grad = 0.000030  fun_value = 0.000000
iter_num =  58  norm_grad = 0.000024  fun_value = 0.000000
iter_num =  59  norm_grad = 0.000019  fun_value = 0.000000
iter_num =  60  norm_grad = 0.000015  fun_value = 0.000000
iter_num =  61  norm_grad = 0.000012  fun_value = 0.000000
iter_num =  62  norm_grad = 0.000010  fun_value = 0.000000
```

```
hold on;plot(xs,xs.^2,'-ro','LineWidth',2);drawnow
```

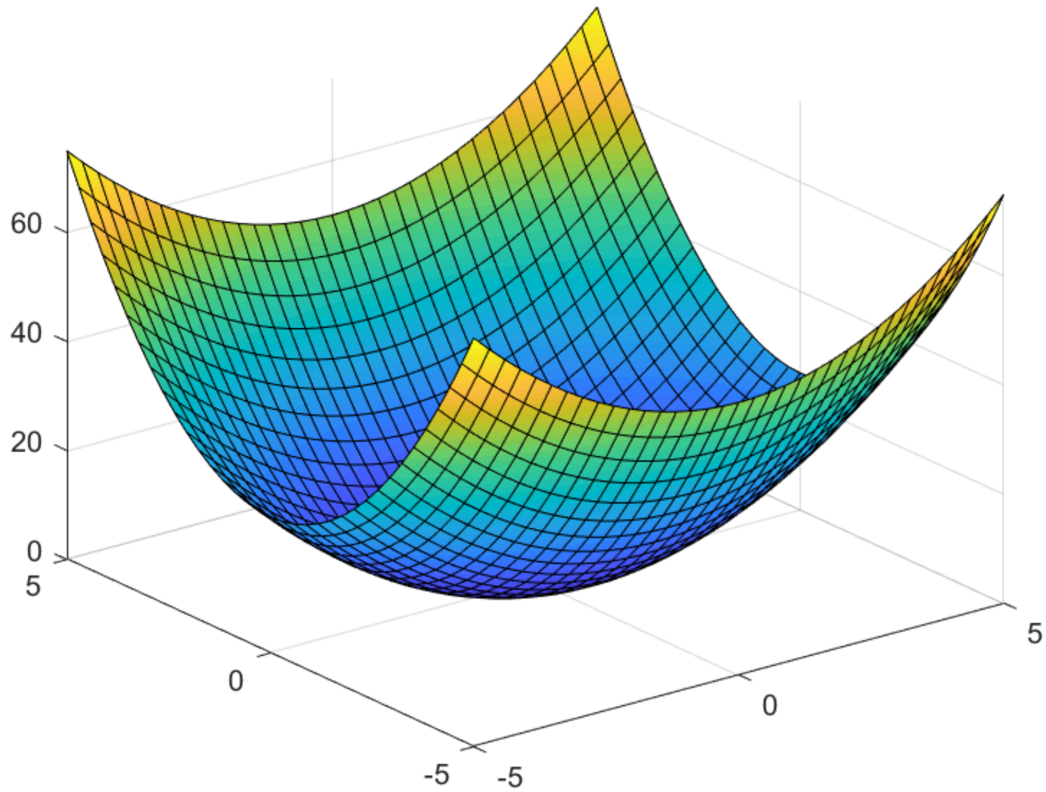


例2：求 $\min f(\mathbf{x}) = x_1^2 + 2x_2^2$

写成二次型

$$
\begin{aligned}
& f(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x} + 2\mathbf{b}^T\mathbf{x} + c \\
& A = \begin{bmatrix} 1, 0 \\ 0, 2 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, c = 0. \\
& \Rightarrow \nabla f(\mathbf{x}) = 2(Ax + b) \\
& \alpha = -\frac{\mathbf{d}^T \nabla f(\mathbf{x})}{2\mathbf{d}^T\mathbf{A}\mathbf{d}} (\mathbf{d} = \nabla f(\mathbf{x}))
\end{aligned}
$$

```
p = 2;
A = [1 0;0 p];
b = [0;0];
x0 = [1;1]; % initial point
x = x0;
gradf = 2*(A*x+b);
iter = 0;
figure,fsurf(@(x1,x2) x1.^2+p*x2.^2)
```

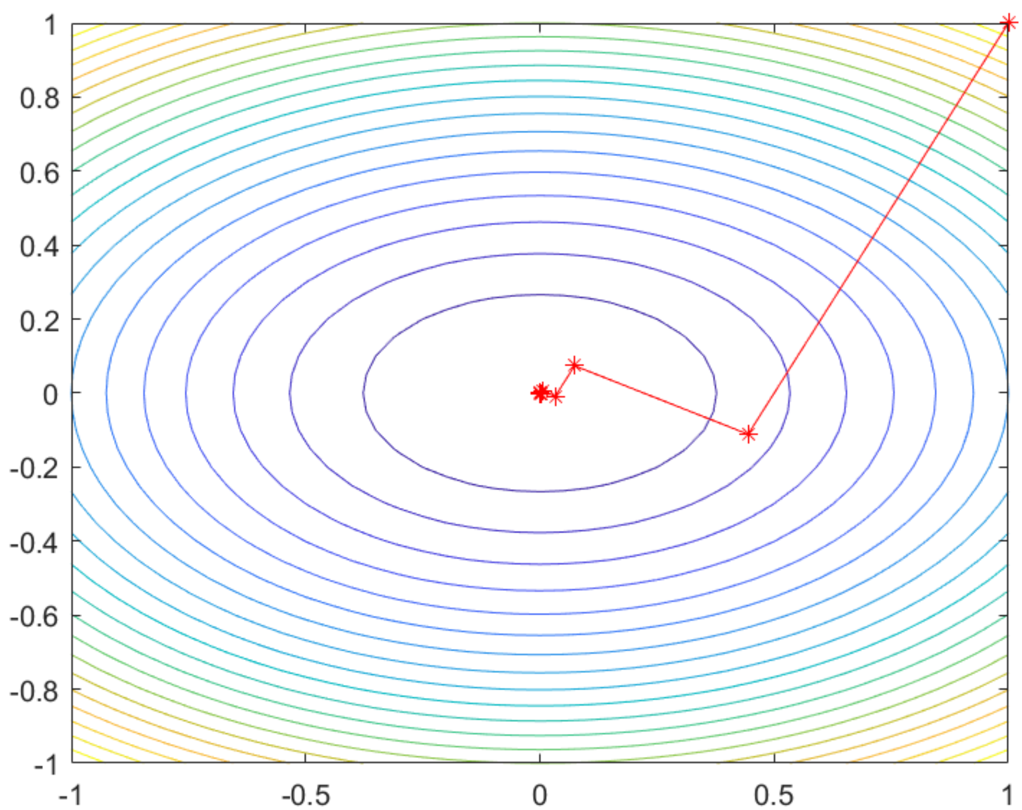



```
[x1,x2] = meshgrid(-1:0.05:1,-1:0.05:1);
f = x1.^2+p*x2.^2;
figure,contour(x1,x2,f,20)
xs(1,:) = x;
while norm(gradf)>eps
    iter = iter+1;
    alpha = norm(gradf).^2./(2*gradf'*A*gradf);
%     alpha = 0.01;
    x = x-alpha*gradf;
    gradf = 2*(A*x+b);
    fun_value = x'*A*x+b'*x;
    fprintf('iter_num = %3d  norm_grad = %2.6f  fun_value = %2.6f\n'...
        ,iter,norm(gradf),fun_value);
    xs(iter+1,:)=x;
end
```

```
iter_num =   1  norm_grad = 0.993808  fun_value = 0.222222
iter_num =   2  norm_grad = 0.331269  fun_value = 0.016461
iter_num =   3  norm_grad = 0.073615  fun_value = 0.001219
iter_num =   4  norm_grad = 0.024538  fun_value = 0.000090
iter_num =   5  norm_grad = 0.005453  fun_value = 0.000007
iter_num =   6  norm_grad = 0.001818  fun_value = 0.000000
iter_num =   7  norm_grad = 0.000404  fun_value = 0.000000
iter_num =   8  norm_grad = 0.000135  fun_value = 0.000000
iter_num =   9  norm_grad = 0.000030  fun_value = 0.000000
iter_num =  10  norm_grad = 0.000010  fun_value = 0.000000
iter_num =  11  norm_grad = 0.000002  fun_value = 0.000000
```

```
iter_num =  12  norm_grad = 0.000001  fun_value = 0.000000
iter_num =  13  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  14  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  15  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  16  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  17  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  18  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  19  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  20  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  21  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  22  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  23  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  24  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  25  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  26  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  27  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  28  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  29  norm_grad = 0.000000  fun_value = 0.000000
```

```
hold on;plot(xs(:,1),xs(:,2),'-r*');drawnow
```




```
x
```

x = 2×1

10^{-16} ×

```
    0.6655
   -0.1664
```

更多例子参见 Rosenbrock.mlx

2 牛顿法和拟牛顿法

2.1 牛顿法

考虑无约束最优化问题 $\min_{x\in\mathbb{R}^n} f(x)$

假设$f(x)$具有二阶连续偏导数，将$f(x)$在$x^{(k)}$附近进行二阶泰勒展开：

$$f(x) = f(x^{(k)}) + g_k^{\mathrm{T}}(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T H(x^{(k)})(x - x^{(k)}) \tag{2}$$

其中$H(x) = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{n\times n}$为$f(x)$的Hesse矩阵.

从$x^{(k)}$出发，求目标函数的极小点，作为第k+1次的迭代值$x^{(k+1)}$，有

$$\nabla f(x^{(k+1)}) = 0$$
$$(2) \Rightarrow \nabla f(x) = g_k + H(x^{(k)})(x - x^{(k)})$$
$$\Rightarrow \nabla f(x^{(k+1)}) = g_k + H(x^{(k)})(x^{(k+1)} - x^{(k)})$$
$$\Rightarrow g_k + H(x^{(k)})(x^{(k+1)} - x^{(k)}) = 0$$
$$\Rightarrow x^{(k+1)} = x^{(k)} - H_k^{-1} g_k$$
$$\text{OR} \quad x^{(k+1)} = x^{(k)} + p_k,\ p_k = -H_k^{-1} g_k$$

算法B.1(牛顿法)

输入: 目标函数$f(x)$,梯度$g(x) = \nabla f(x)$, 海赛矩阵$H(x)$,精度要求ϵ;

输出: $f(x)$的极小点x^*.

- (1) 取初始点x^*, 置$k = 0$.
- (2) 计算$g_k = g(x^{(k)})$.
- (3) 若$\|g_k\| < \epsilon$时，则停止迭代，得近似解$x^* = x^{(k)}$.
- (4) 计算$H_k = H(x^{(k)})$, 并求$p_k = -H_k^{-1} g_k$.
- (5) 置$x^{(k+1)} = x^{(k)} + p_k$.
- (6) 置k=k+l,转(2).

```
% 例子（牛顿法）
p = 2;
A = [1 0;0 p];
b = [0;0];
```

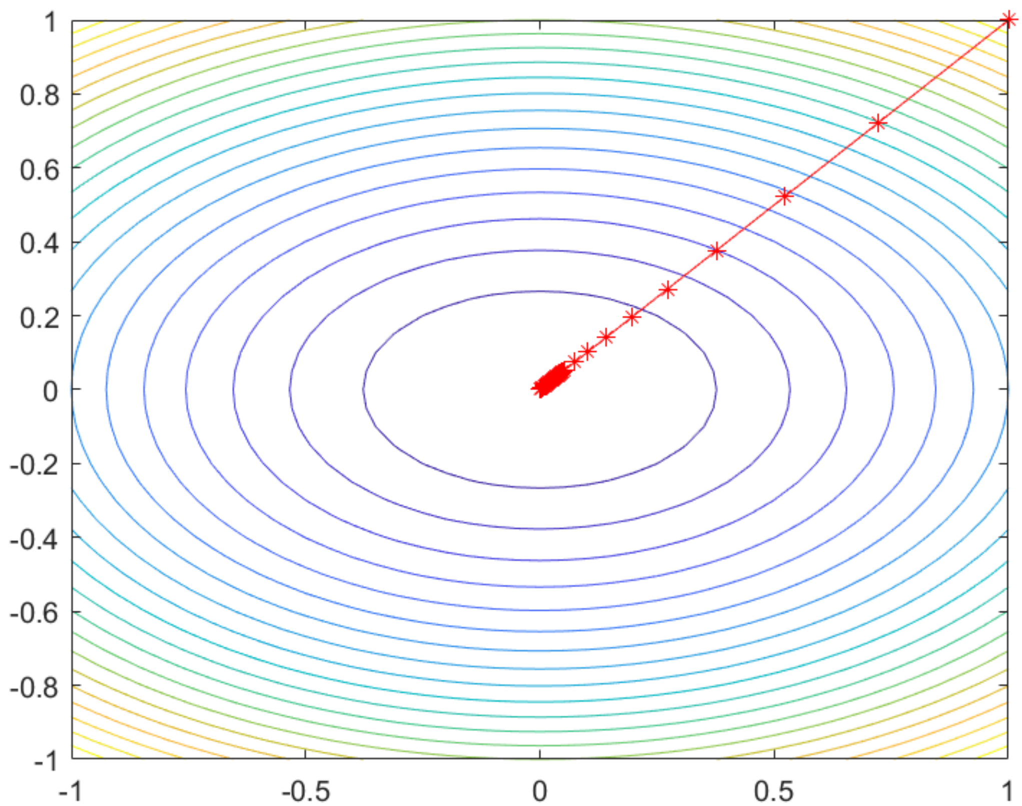
```
x0 = [1;1]; % initial point
x = x0;
gradf = 2*(A*x+b);
H = 2*A;
iter = 0;
[x1,x2] = meshgrid(-1:0.05:1,-1:0.05:1);
f = x1.^2+p*x2.^2;
figure,contour(x1,x2,f,20)
xs(1,:) = x;
while norm(gradf)>eps
    iter = iter+1;
    alpha = norm(gradf).^2./(2*gradf'*A*gradf);
%     alpha = 0.01;
    x = x-alpha*inv(H)*gradf;
    gradf = 2*(A*x+b);

    fun_value = x'*A*x+b'*x;
    fprintf('iter_num = %3d  norm_grad = %2.6f  fun_value = %2.6f\n'...
        ,iter,norm(gradf),fun_value);
    xs(iter+1,:)=x;
end
```

```
iter_num =   1  norm_grad = 3.229876  fun_value = 1.564815
iter_num =   2  norm_grad = 2.332688  fun_value = 0.816215
iter_num =   3  norm_grad = 1.684719  fun_value = 0.425742
iter_num =   4  norm_grad = 1.216742  fun_value = 0.222069
iter_num =   5  norm_grad = 0.878758  fun_value = 0.115832
iter_num =   6  norm_grad = 0.634658  fun_value = 0.060419
iter_num =   7  norm_grad = 0.458364  fun_value = 0.031515
iter_num =   8  norm_grad = 0.331041  fun_value = 0.016438
iter_num =   9  norm_grad = 0.239085  fun_value = 0.008574
iter_num =  10  norm_grad = 0.172673  fun_value = 0.004472
iter_num =  11  norm_grad = 0.124708  fun_value = 0.002333
iter_num =  12  norm_grad = 0.090067  fun_value = 0.001217
iter_num =  13  norm_grad = 0.065048  fun_value = 0.000635
iter_num =  14  norm_grad = 0.046979  fun_value = 0.000331
iter_num =  15  norm_grad = 0.033930  fun_value = 0.000173
iter_num =  16  norm_grad = 0.024505  fun_value = 0.000090
iter_num =  17  norm_grad = 0.017698  fun_value = 0.000047
iter_num =  18  norm_grad = 0.012782  fun_value = 0.000025
iter_num =  19  norm_grad = 0.009231  fun_value = 0.000013
iter_num =  20  norm_grad = 0.006667  fun_value = 0.000007
iter_num =  21  norm_grad = 0.004815  fun_value = 0.000003
iter_num =  22  norm_grad = 0.003478  fun_value = 0.000002
iter_num =  23  norm_grad = 0.002512  fun_value = 0.000001
iter_num =  24  norm_grad = 0.001814  fun_value = 0.000000
iter_num =  25  norm_grad = 0.001310  fun_value = 0.000000
iter_num =  26  norm_grad = 0.000946  fun_value = 0.000000
iter_num =  27  norm_grad = 0.000683  fun_value = 0.000000
iter_num =  28  norm_grad = 0.000494  fun_value = 0.000000
iter_num =  29  norm_grad = 0.000356  fun_value = 0.000000
iter_num =  30  norm_grad = 0.000257  fun_value = 0.000000
iter_num =  31  norm_grad = 0.000186  fun_value = 0.000000
iter_num =  32  norm_grad = 0.000134  fun_value = 0.000000
iter_num =  33  norm_grad = 0.000097  fun_value = 0.000000
iter_num =  34  norm_grad = 0.000070  fun_value = 0.000000
iter_num =  35  norm_grad = 0.000051  fun_value = 0.000000
iter_num =  36  norm_grad = 0.000037  fun_value = 0.000000
iter_num =  37  norm_grad = 0.000026  fun_value = 0.000000
iter_num =  38  norm_grad = 0.000019  fun_value = 0.000000
```

```
iter_num =  39  norm_grad = 0.000014  fun_value = 0.000000
iter_num =  40  norm_grad = 0.000010  fun_value = 0.000000
iter_num =  41  norm_grad = 0.000007  fun_value = 0.000000
iter_num =  42  norm_grad = 0.000005  fun_value = 0.000000
iter_num =  43  norm_grad = 0.000004  fun_value = 0.000000
iter_num =  44  norm_grad = 0.000003  fun_value = 0.000000
iter_num =  45  norm_grad = 0.000002  fun_value = 0.000000
iter_num =  46  norm_grad = 0.000001  fun_value = 0.000000
iter_num =  47  norm_grad = 0.000001  fun_value = 0.000000
iter_num =  48  norm_grad = 0.000001  fun_value = 0.000000
iter_num =  49  norm_grad = 0.000001  fun_value = 0.000000
iter_num =  50  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  51  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  52  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  53  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  54  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  55  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  56  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  57  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  58  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  59  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  60  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  61  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  62  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  63  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  64  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  65  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  66  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  67  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  68  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  69  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  70  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  71  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  72  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  73  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  74  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  75  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  76  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  77  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  78  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  79  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  80  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  81  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  82  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  83  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  84  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  85  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  86  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  87  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  88  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  89  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  90  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  91  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  92  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  93  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  94  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  95  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  96  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  97  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  98  norm_grad = 0.000000  fun_value = 0.000000
iter_num =  99  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 100  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 101  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 102  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 103  norm_grad = 0.000000  fun_value = 0.000000
```

```
iter_num = 104  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 105  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 106  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 107  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 108  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 109  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 110  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 111  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 112  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 113  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 114  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 115  norm_grad = 0.000000  fun_value = 0.000000
iter_num = 116  norm_grad = 0.000000  fun_value = 0.000000
```

```
hold on;plot(xs(:,1),xs(:,2),'-r*');drawnow
```



2.2 拟牛顿法

$(2) \Rightarrow \nabla f(x^{(k+1)}) = g_k + H(x^{(k)})(x^{(k+1)} - x^{(k)})$

$\quad \Rightarrow g_{k+1} - g_k = H_k(x^{(k+1)} - x^{(k)})$

$y_k := g_{k+1} - g_k, \delta_k := x^{(k+1)} - x^{(k)}$

$\quad \Rightarrow y_k = H_k \delta_k$ OR $\delta_k = H_k^{-1} y_k$(拟牛顿条件)

结论：如果H_k正定，可以保证牛顿法的搜索方向$p_k = -H_k^{-1} g_k$是下降的(证明)。

拟牛顿法：用G_k近似H_k^{-1}，要求G_k正定且满足拟牛顿条件 $\delta_k = G_{k+1}y_k$,其中$G_{k+1} = G_k + \Delta G_k$.

2.3 DFP（Davidon-**Fletcher-**Powell）算法

假设$G_{k+1} = G_k + P_k + Q_k$，其中$P_k, Q_k$待定，由拟牛顿条件

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

令$P_k y_k = \delta_k, Q_k y_k = -G_k y_k$,不难找出满足条件的$P_k, Q_k$，比如

$$P_k = \frac{\delta_k \delta_k^{\mathrm{T}}}{\delta_k^{\mathrm{T}} y_k},\ Q_k = -\frac{G_k y_k y_k^{\mathrm{T}} G_k}{y_k^{\mathrm{T}} G_k y_k}$$

DFP更新公式

$$G_{k+1} = G_k + \frac{\delta_k \delta_k^{\mathrm{T}}}{\delta_k^{\mathrm{T}} y_k} - \frac{G_k y_k y_k^{\mathrm{T}} G_k}{y_k^{\mathrm{T}} G_k y_k} \qquad (3)$$

推导过程：

- $G_{k+1} = G_k + \alpha u u^T + \beta v v^T$
- 两边乘以y_k,有$\delta_k = G_k y_k + (\alpha u^T y_k)u + (\beta v^T y_k)v = G_k y_k + u - v$，其中$(\alpha u^T y_k) = 1$，$(\beta v^T y_k) = -1$
- 解出$\alpha = \frac{1}{u^T y_k}, \beta = \frac{-1}{v^T y_k}$，且有$u - v = \delta_k - G_k y_k$，可得$u$和$v$，从而最终解得DFP更新公式：$G_{k+1} = G_k + \frac{\delta_k \delta_k^{\mathrm{T}}}{\delta_k^{\mathrm{T}} y_k} - \frac{G_k y_k y_k^{\mathrm{T}} G_k}{y_k^{\mathrm{T}} G_k y_k}$

证明如果G_0正定，则$G_k, k \geq 1$正定(自己查阅资料，补充证明).

算法**B.2 (DFP**算法）

输入: 目标函数$f(x)$,梯度$g(x) = \nabla f(x)$，海赛矩阵$H(x)$,精度要求ϵ;

输出: $f(x)$的极小点x^*.

- (l)选定初始点$x^{(0)}$，取G_0为正定对称矩阵， 置$k = 0$
- (2)计算$g_k = g(x^{(k)})$，若$\|g_k\| < \epsilon$时，则停止迭代，得近似解$x^* = x^{(k)}$；否则转(3)
- (3)置$p_k = -G_k g_k$，
- (4) 一维搜索； 求λ_k使得 $f\left(x^{(k)} + \lambda_k p_k\right) = \min_{\lambda \geq 0} f\left(x^{(k)} + \lambda p_k\right)$
- (5)置$x^{(k+1)} = x^{(k)} + \lambda_k p_k$
- (6)计算$g_{k+1} = g(x^{(k+1)})$，若$\|g_{k+1}\| < \epsilon$时，则停止迭代，得近似解$x^* = x^{(k+1)}$；否则， 按式（3）算出G_{k+1}
- (7)置$k = k + 1$，转(3).

2.4 BFGS（Broyden-Fletcher-Goldfard-Shanno） 算法

BFGS算法是最流行的拟牛顿算法.

可以考虑用G_k逼近海赛矩阵的逆矩阵H_k^{-1},也可以考虑用B_k逼近海赛矩阵H_k. 这时，相应的拟牛顿条件是:
$y_k = B_{k+1}\delta_k$.

假设$B_{k+1} = B_k + P_k + Q_k$, 其中$P_k, Q_k$待定，由拟牛顿条件

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

令$P_k\delta_k = y_k, Q_k\delta_k = -B_k\delta_k$, 找到满足条件的$P_k, Q_k$，得到BFGS迭代公式

$$B_{k+1} = B_k + \frac{y_k y_k^{\mathrm{T}}}{y_k^{\mathrm{T}}\delta_k} - \frac{B_k\delta_k\delta_k^{\mathrm{T}}B_k}{\delta_k^{\mathrm{T}}B_k\delta_k} \qquad (4)$$

可以证明, 如果初始矩阵B_0是正定的，则迭代过程中的每个矩阵B_k都是正定的.

算法**B.3 (BFGS**算法）

输入：目标函数$f(x)$, $g(x) = \nabla f(x)$, 精度要求ϵ;

输出：$f(x)$的极小点x^*

- (I) 选定初始点$x^{(0)}$, 取B_0为正定对称矩阵， 置$k = 0$
- (2) 计算$g_k = g(x^{(k)})$, 若$\|g_k\| < \epsilon$, 则停止计算， 得近似解$x = x^{(k)}$; 否则, 转(3)
- (3) 由$B_k p_k = -g_k$, 求出p_k.
- (4) 一维搜索，求λ_k使得: $f\left(x^{(k)} + \lambda_k p_k\right) = \min_{\lambda \geq 0} f\left(x^{(k)} + \lambda p_k\right)$
- (5) 置$x^{(k+1)} = x^{(k)} + \lambda_k p_k$
- (6) 计算$g_{k+1} = g(x^{(k+1)})$, 若$\|g_{k+1}\| < \epsilon$, 则停止计算，得近似解$x^* = x^{(k+1)}$; 否则, 按BFGS迭代公式（4）求出B_{k+1}
- (7) 置$k = k + 1$, 转(3).

2.5 Broyden类算法（Broyden's algorithm）

Sherman-Morrison公式：假设A是n阶可逆矩阵，u, v是n维向量， 且$A + uv^T$也是可逆矩阵，
则$(A + uv^{\mathrm{T}})^{-1} = A^{-1} - \dfrac{A^{-1}uv^{\mathrm{T}}A^{-1}}{1 + v^{\mathrm{T}}A^{-1}u}$ or

$$\left(\mathrm{A} + \frac{uu^T}{t}\right)^{-1} = A^{-1} - \frac{A^{-1}uu^TA^{-1}}{t + u^TA^{-1}u}$$

对式（4）两次应用Sherman--Morrison公式，即得BFGS算法关于G_{k+1}的迭代公式

$$B_{k+1}^{-1} = G_{k+1} = \left(I - \frac{\delta_k y_k^{\mathrm{T}}}{\delta_k^{\mathrm{T}} y_k}\right) G_k \left(I - \frac{\delta_k y_k^{\mathrm{T}}}{\delta_k^{\mathrm{T}} y_k}\right)^{\mathrm{T}} + \frac{\delta_k \delta_k^{\mathrm{T}}}{\delta_k^{\mathrm{T}} y_k} \tag{5}$$

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

（5）式推导过程：

Sherman Morrison 公式：

$$\left(\mathrm{A} + \frac{uu^T}{t}\right)^{-1} = A^{-1} - \frac{A^{-1}uu^TA^{-1}}{t + u^TA^{-1}u}$$

$$\left(\mathrm{H} + \frac{yy^T}{y^T s} - \frac{Hss^T\mathrm{H}}{s^THs}\right)^{-1}$$

$$= \left(\mathrm{H} + \frac{yy^T}{y^T\mathrm{s}}\right)^{-1} + \left(\mathrm{H} + \frac{yy^T}{y^T\mathrm{s}}\right)^{-1} \frac{Hss^TH}{s^TH^Ts - s^TH\left(\mathrm{H} + \frac{yy^T}{y^T\mathrm{s}}\right)^{-1} Hs} \left(\mathrm{H} + \frac{yy^T}{y^T\mathrm{s}}\right)^{-1}$$

$$= (H^{-1} - \frac{H^{-1}yy^TH^{-1}}{y^T\mathrm{s} + y^TH^{-1}y}) + (H^{-1} - \frac{H^{-1}yy^TH^{-1}}{y^T\mathrm{s} + y^TH^{-1}y}) \frac{Hss^TH}{s^THs - s^TH(H^{-1} - \frac{H^{-1}yy^TH^{-1}}{y^T\mathrm{s} + y^TH^{-1}y})Hs} (H^{-1} - \frac{H^{-1}yy^TH^{-1}}{y^T\mathrm{s} + y^TH^{-1}y})$$

$$= (H^{-1} - \frac{H^{-1}yy^TH^{-1}}{y^T\mathrm{s} + y^TH^{-1}y}) + (H^{-1} - \frac{H^{-1}yy^TH^{-1}}{y^T\mathrm{s} + y^TH^{-1}y}) \frac{Hss^TH}{\frac{s^Tyy^Ts}{y^T\mathrm{s} + y^TH^{-1}y}} (H^{-1} - \frac{H^{-1}yy^TH^{-1}}{y^T\mathrm{s} + y^TH^{-1}y})$$

$$= (H^{-1} - \frac{H^{-1}yy^TH^{-1}}{y^T\mathrm{s} + y^TH^{-1}y}) + \frac{H^{-1}Hss^THH^{-1}}{\frac{s^Tyy^Ts}{y^T\mathrm{s} + y^TH^{-1}y}} - \frac{H^{-1}Hss^TH}{\frac{s^Tyy^Ts}{y^T\mathrm{s} + y^TH^{-1}y}} H^{-1} \frac{yy^T}{y^T\mathrm{s} + y^TH^{-1}y} H^{-1}$$

$$- \frac{H^{-1}yy^TH^{-1}}{y^T\mathrm{s} + y^TH^{-1}y} \frac{Hss^TH}{\frac{s^Tyy^Ts}{y^T\mathrm{s} + y^TH^{-1}y}} H^{-1}$$

$$+ H^{-1} \frac{yy^T}{y^T\mathrm{s} + y^TH^{-1}y} H^{-1} \frac{Hss^TH}{\frac{s^Tyy^Ts}{y^T\mathrm{s} + y^TH^{-1}y}} H^{-1} \frac{yy^T}{y^T\mathrm{s} + y^TH^{-1}y} H^{-1}$$

$$= (H^{-1} - \frac{H^{-1}yy^TH^{-1}}{y^Ts + y^TH^{-1}y}) + \frac{ss^T(y^Ts + y^TH^{-1}y)}{s^Tyy^Ts} - \frac{ss^Tyy^TH^{-1}}{s^Tyy^Ts} - \frac{H^{-1}yy^Tss^T}{s^Tyy^Ts} + \frac{H^{-1}yy^Tss^Tyy^TH^{-1}}{(y^Ts + y^TH^{-1}y)s^Tyy^Ts}$$

$$= (H^{-1} - \frac{H^{-1}yy^TH^{-1}}{y^Ts + y^TH^{-1}y}) + \frac{ss^T(y^Ts + y^TH^{-1}y)}{(s^Ty)^2} - \frac{s(s^Ty)y^TH^{-1}}{(s^Ty)^2} - \frac{H^{-1}y(y^Ts)s^T}{(s^Ty)^2} + \frac{H^{-1}y(y^Tss^Ty)y^TH^{-1}}{(y^Ts + y^TH^{-1}y)s^Tyy^Ts}$$

$$= (H^{-1} - \frac{H^{-1}yy^TH^{-1}}{y^Ts + y^TH^{-1}y}) + \frac{ss^T(y^Ts + y^TH^{-1}y)}{(s^Ty)^2} - \frac{sy^TH^{-1}}{s^Ty} - \frac{H^{-1}ys^T}{s^Ty} + \frac{H^{-1}yy^TH^{-1}}{(y^Ts + y^TH^{-1}y)}$$

$$= H^{-1} + \frac{ss^T(y^Ts + y^TH^{-1}y)}{(s^Ty)^2} - \frac{sy^TH^{-1}}{s^Ty} - \frac{H^{-1}ys^T}{s^Ty}$$

$$= H^{-1} + \frac{ss^Ty^Ts}{(s^Ty)^2} + \frac{ss^Ty^TH^{-1}y}{(s^Ty)^2} - \frac{sy^TH^{-1}}{s^Ty} - \frac{H^{-1}ys^T}{s^Ty}$$

$$= H^{-1}\left(I - \frac{ys^T}{s^Ty}\right) - \frac{sy^TH^{-1}}{s^Ty}\left(I - \frac{ys^T}{s^Ty}\right) + \frac{ss^T}{s^Ty}$$

$$= \left(I - \frac{sy^T}{s^Ty}\right)H^{-1}\left(I - \frac{ys^T}{s^Ty}\right) + \frac{ss^T}{s^Ty}$$

https://blog.csdn.net/langb2014/article/details/48915425

3 拉格朗日对偶性

在约束最优化问题中，常常利用拉格朗日对偶性（Lagrange duality）将原始问题转换为对偶问题，通过解对偶问题而得到原始问题的解．该方法应用在许多统计学习方法中，例如，最大熵模型与支持向量机．这里简要叙述拉格朗日对偶性的主要概念和结果.

3.1 原始问题

假设$f(x), c_i(x), h_j(x)$是定义在$\mathbb{R}^n$上的连续可微函数，考虑约束最优化问题

$$\begin{aligned} &\min_{z\in\mathbb{R}^n} f(x) \\ \text{s.t.}\quad & c_i(x) \le 0, \quad i = 1, 2, \cdots, k \\ & h_j(x) = 0, \quad j = 1, 2, \cdots, l \end{aligned}$$

（P）--原始问题

首先，引进广义拉格朗日函数（generalized Lagrange function）

$$L(x, \alpha, \beta) = f(x) + \sum_{i=1}^{k} \alpha_i c_i(x) + \sum_{j=1}^{l} \beta_j h_j(x)$$

这里$x = (x^{(1)}, x^{(2)}, \cdots, x^{(n)})^{\mathrm{T}} \in \mathbf{R}^n$，$\alpha_i, \beta_j$是拉格朗日乘子，$\alpha_i \ge 0$，考虑$x$的函数:

$$\theta_P(x) = \max_{\alpha,\beta;\alpha_i\ge0} L(x, \alpha, \beta)$$

$$\theta_p(x) = \begin{cases} f(x), & x\text{满足原始问题约束} \\ +\infty, & \text{其他} \end{cases}$$

极小化问题 $\min_x \theta_P(x) = \min_x \max_{\alpha,\beta;\alpha_i \ge 0} L(x,\alpha,\beta)$（广义拉格朗日函数的极小极大问题）与原问题（P）有相同的解。

原始问题的最优值：$p^* = \min_x \theta_p(x)$

3.2 对偶问题

定义 $\theta_D(\alpha,\beta) = \min_x L(x,\alpha,\beta)$

$\max_{\alpha,\beta;\alpha_i \ge 0} \theta_D(\alpha,\beta) = \max_{\alpha,\beta;\alpha_i \ge 0} \min_x L(x,\alpha,\beta)$（广义拉格朗日函数的极大极小问题）

$$\Longleftrightarrow \quad \begin{aligned} & \max_{\alpha,\beta} \theta_D(\alpha,\beta) = \max_{\alpha,\beta} \min_x L(x,\alpha,\beta) \\ & \text{s.t.} \quad \alpha_i \ge 0, \quad i = 1,2,\cdots,k \end{aligned} \qquad \text{（D）--对偶问题}$$

定义对偶问题的最优值 $d^* = \max_{\alpha,\beta;\alpha_i \ge 0} \theta_D(\alpha,\beta)$

定理**C.1** 若原始问题和对偶问题都有最优值，则

$d^* = \max_{\alpha,\beta;\alpha_i \ge 0} \min_x L(x,\alpha,\beta) \le \min_x \max_{\alpha,\beta;\alpha_i \ge 0} L(x,\alpha,\beta) = p^*$

证明：$\theta_D(\alpha,\beta) = \min_x L(x,\alpha,\beta) \le L(x,\alpha,\beta) \le \max_{\alpha,\beta;\alpha_i \ge 0} L(x,\alpha,\beta) = \theta_P(x)$

所以 $\max_{\alpha,\beta:\alpha_i \ge 0} \theta_D(\alpha,\beta) \le \min_x \theta_P(x)$.

推论**C.1** 设x^*和α^*,β^*分别是原始问题(P)和对偶问题(D)的可行解，并且$d^* = p^*$，x^*和α^*,β^*分别是原始问题和对偶问题的最优解.

定理**C.2** 考虑原始问题(P)和对偶问题(D)，假设函数$f(x)$和$c_i(x)$是凸函数，$h_j(x)$是仿射函数， 并且假设不等式约束$c_i(x)$是严格可行的， 即存在x, 对所有i，有$c_i(x) < 0$, 则存在x^*, α^*,β^*, 使x^*是原始问题的解，α^*,β^*是对偶问题的解， 并且

$$p^* = d^* = L(x^*,\alpha^*,\beta^*)$$

定理**C.3** 对原始问题(P)和对偶问题(D)，假设函数$f(x)$和$c_i(x)$是凸函数，$h_j(x)$是仿射函数，并且不等式约束$c_i(x)$是严格可行的，则x^*和α^*,β^*分别是原始问题和对偶问题的解的充分必要条件是x^*,α^*,β^*满足下面的Karush-Kuhn-Tucker (KKT)条件：

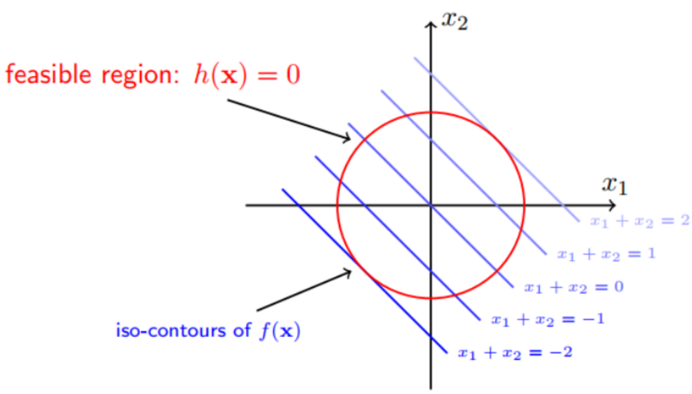
$$
\begin{aligned}
\nabla_x L(x^*, \alpha^*, \beta^*) &= 0 \\
\nabla_\alpha L(x^*, \alpha^*, \beta^*) &= 0 \\
\nabla_\beta L(x^*, \alpha^*, \beta^*) &= 0 \\
\alpha_i^* c_i(x^*) &= 0, \quad i = 1, 2, \cdots, k \\
c_i(x^*) &\le 0, \quad i = 1, 2, \cdots, k \\
\alpha_i^* &\ge 0, \quad i = 1, 2, \cdots, k \\
h_j(x^*) &= 0 \quad j = 1, 2, \cdots, l
\end{aligned}
$$

$\alpha_i^* c_i(x^*) = 0$称为对偶互补条件.

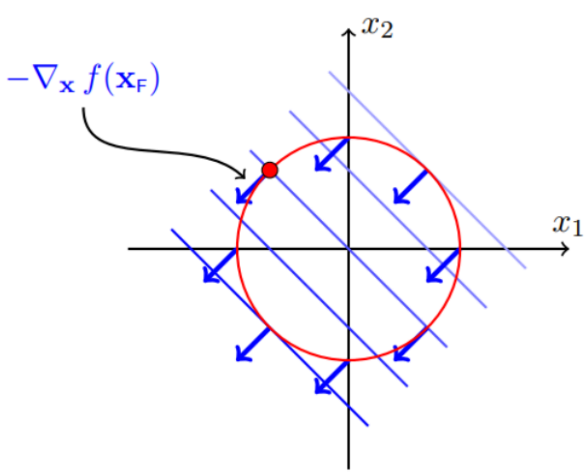
KKT条件的直观理解

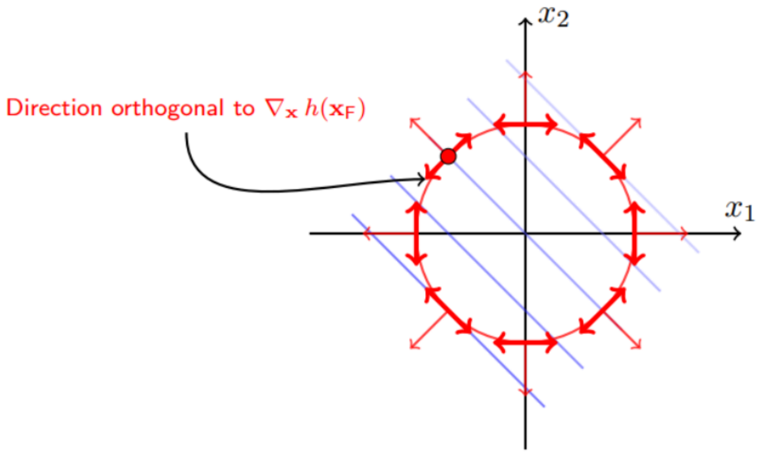
等式约束

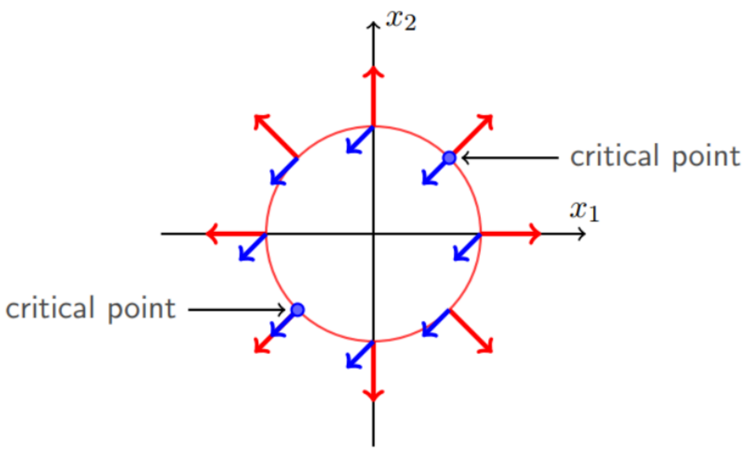
考虑一个简单的问题目标函数$f(x) = x_1 + x_2$，等式约束 $h(x) = x_1^2 + x_2^2 - 2$，求解极小值点。



$$\boxed{h(\mathbf{x}) = x_1^2 + x_2^2 - 2}$$

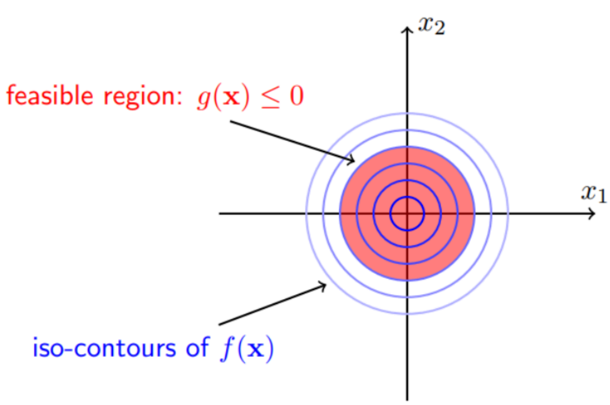






$$\boxed{\nabla_{\mathbf{x}} f(\mathbf{x}^*) = \mu \nabla_{\mathbf{x}} h(\mathbf{x}^*)}$$

不等式约束

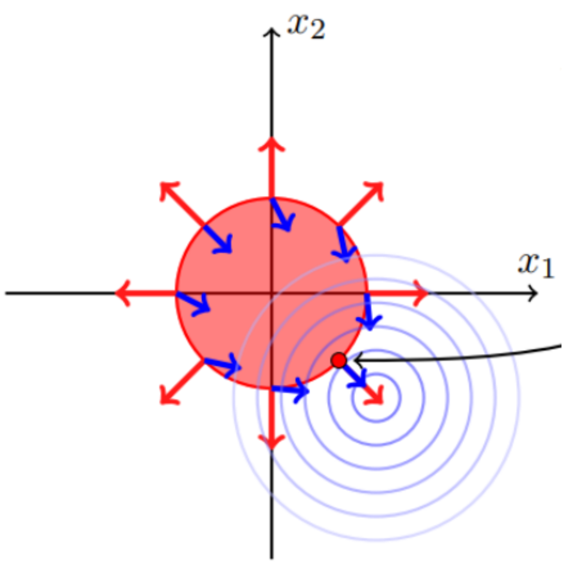


$$\boxed{g(\mathbf{x}) = x_1^2 + x_2^2 - 1}$$

极小值点落在可行域内（不包含边界）

考虑目标函数$f(x)=x_1^2+x_2^2$，不等值约束$g(x)=x_1^2+x_2^2-1$，显然f(x)的极小值为原点(0,0)，落在可行域内。可行域以原点为圆心，半径为1。

这种情况约束不起作用，考虑极小值点x*，这个时候，g(x*) < 0，f(x*)的梯度等于0。



极小值点落在可行域外（包含边界）

考虑目标函数$f(x)=(x_1-1.1)^2+(x_2+1.1)^2$，不等值约束$g(x)=x_1^2+x_2^2-1$，显然f(x)的极小值为原点(1.1, -1.1)，落在可行域外。可行域以原点为圆心，半径为1。

这种情况约束起作用，要考虑求解f(x)在可行域内的极小值点。

$$\boxed{-\nabla_{\mathbf{x}} f(\mathbf{x})=\lambda\nabla_{\mathbf{x}} g(\mathbf{x}) \quad \textbf{and} \quad \lambda>0}$$

4 矩阵的基本子空间

4.1 向量空间的基本子空间

若S是向量空间V的非空子集，且S满足以下条件：

(1) 对任意实数a，若$x\in S$，则$ax\in S$；

(2) 若$x\in S$，$y\in S$，则$x+y\in S$；

则称S为V的子空间.

$\mathrm{span}(v_1,v_2,\cdots,v_n)$：由向量$v_1,v_2,\cdots,v_n$的线性组合所构成的子空间

4.2 向量空间的基和维数

向量空间V中的向量$v_1, v_2, \cdots, v_n$称为V的一个基，如果满足条件

(1) $v_1, v_2, \cdots, v_n$线性无关；

(2) $\text{span}(v_1, v_2, \cdots, v_n) = V$.

向量空间基的个数=向量空间的维数

4.3 矩阵的行空间和列空间

设A为一$m \times n$的矩阵. A的每一行称为行向量，每一列称为列向量. 由A的行向量张成的$\mathbf{R}^n$的子空间称为A的行空间，由A的列向量所张成的$\mathbf{R}^m$的子空间，称为A的列空间.

矩阵A的行空间的维数=列空间的维数=矩阵A的秩

4.4 矩阵的零空间

设A为$m \times n$的矩阵，令$N(A)$为齐次方程组$Ax = 0$的所有解的集合，则称$N(A)$为A的零空间.

$N(A) = \{x \in \mathbf{R}^n | Ax = 0\}$

一个矩阵零空间的维数称为零度.

<u>秩-零度定理</u>：设A为一$m \times n$的矩阵，则A的秩与A的零度之和为n. 事实上，若A的秩为r，则方程$Ax = 0$的独立变量个数为r，自由变量个数为n-r，$N(A)$的维数=自由变量维数.

4.5 子空间的正交补

设X, Y为$\mathbf{R}^n$的子空间，若对任一$x \in X, y \in Y$都满足$x^T y = 0$，则称X和Y正交，记作$X \perp Y$.

$Y^\perp = \{x \in \mathbf{R}^n | x^T y = 0, \forall y \in Y\}$称为$Y$的正交补.

4.6 矩阵的基本子空间

设A为一$m \times n$的矩阵，可以将A看成从$\mathbf{R}^n$映射到$\mathbf{R}^m$的线性变换.

A的值域$R(A) = \{z \in \mathbf{R}^m | \exists x \in \mathbf{R}^n, z = Ax\}$=A的列空间

$R(A^T) = \{y \in \mathbf{R}^n | \exists x \in \mathbf{R}^m, y = A^T x\}$=A的行空间

矩阵A有四个基本子空间；列空间，行空间，零空间，转置零空间（左零空间）

<u>定理**D.1**</u> 若A为一$m \times n$的矩阵，则$N(A) = R(A^T)^\perp$，且$N(A^T) = R(A)^\perp$

5 KL散度的定义和狄利克雷分布的性质

5.1 KL散度的定义

（KL divergence， Kullback-Leibler divergence）

KL散度是描述两个概率分布$Q(x)$和$P(x)$相似度的一种度量，记为$D(Q||P)$

离散：$D(Q||P) = \sum_i Q(i) \log \frac{Q(i)}{P(i)}$

连续：$D(Q||P) = \int Q(x) \log \frac{Q(x)}{P(x)} dx$

性质：$D(Q||P) \geq 0, D(Q||P) = 0 \Leftrightarrow Q = P$， 非对称，不满足三角不等式

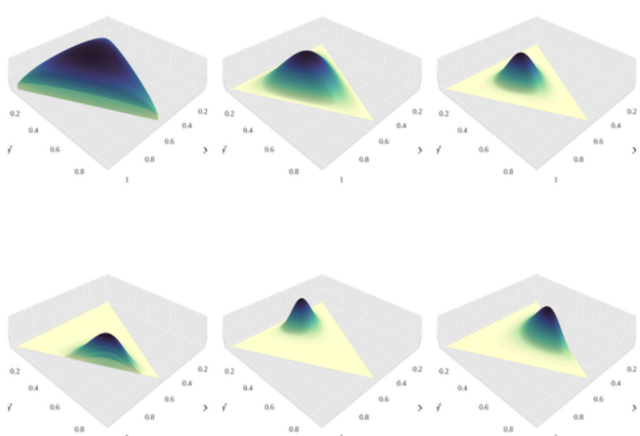
$$-D(Q||P) = \int Q(x) \log \frac{P(x)}{Q(x)} dx \leq \log \int Q(x) \frac{P(x)}{Q(x)} dx = \log \int P(x) dx = 0$$

Jensen不等式：https://zh.wikipedia.org/wiki/%E5%BB%B6%E6%A3%AE%E4%B8%8D%E7%AD%89%E5%BC%8F

5.2 狄利克雷分布的性质

设随机变量$\theta \sim \mathrm{Dir}(\theta|\alpha)$，求$E(\log\theta)$

https://zh.wikipedia.org/wiki/%E7%8B%84%E5%88%A9%E5%85%8B%E9%9B%B7%E5%88%86%E5%B8%83



狄利克雷分布概率密度函数

指数分布族是指概率分布密度可以写成如下形式的概率分布集合：

$$p(x|\eta) = h(x) \exp\{\eta^T T(x) - A(\eta)\}$$

其中η是自然参数，$T(x)$是充分统计量，$h(x)$是潜在测度，$A(\eta)$是对数规范化因子

$$A(\eta) = \log \int h(x) \exp\{\eta^T T(x)\} dx$$

指数分布族具有如下性质:

$$
\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}\eta}A(\eta)=\frac{\mathrm{d}}{\mathrm{d}\eta}\log\int h(x)\exp\left\{\eta^{\mathrm{T}}T(x)\right\}\mathrm{d}x\\
&=\frac{\int T(x)\exp\left\{\eta^{\mathrm{T}}T(x)\right\}h(x)\mathrm{d}x}{\int h(x)\exp\left\{\eta^{\mathrm{T}}T(x)\right\}\mathrm{d}x}\\
&=\int T(x)\exp\left\{\eta^{\mathrm{T}}T(x)-A(\eta)\right\}h(x)\mathrm{d}x\\
&=\int T(x)p(x|\eta)\mathrm{d}x=E[T(X)]
\end{aligned}
$$

狄利克雷分布属于指数族，因为其密度函数可以写成指数分布族的密度函数形式

$$
\begin{aligned}
&p(\theta|\alpha)=\frac{\Gamma\left(\sum_{l=1}^{K}\alpha_l\right)}{\prod_{k=1}^{K}\Gamma(\alpha_k)}\prod_{k=1}^{K}\theta_k^{\alpha_k-1}\\
&=\exp\left\{\left(\sum_{k=1}^{K}(\alpha_k-1)\log\theta_k\right)+\log\Gamma\left(\sum_{l=1}^{K}\alpha_l\right)-\sum_{k=1}^{K}\log\Gamma(\alpha_k)\right\}
\end{aligned}
$$

自然参数是$\eta_k=\alpha_k-1$，充分统计量$T(\theta_k)=\log\theta_k$，对数规范化因子是

$$
A(\alpha)=\sum_{k=1}^{K}\log\Gamma(\alpha_k)-\log\Gamma\left(\sum_{l=1}^{K}\alpha_l\right)
$$

利用指数分布族的性质，可得

$$
\begin{aligned}
E_{p(\theta|\alpha)}[\log\theta_k]&=\frac{\mathrm{d}}{\mathrm{d}\alpha_k}A(\alpha)=\frac{\mathrm{d}}{\mathrm{d}\alpha_k}\left[\sum_{k=1}^{K}\log\Gamma(\alpha_k)-\log\Gamma\left(\sum_{l=1}^{K}\alpha_l\right)\right]\\
&=\Psi(\alpha_k)-\Psi\left(\sum_{l=1}^{K}\alpha_l\right),\quad k=1,2,\cdots,K
\end{aligned}
$$

其中Ψ是digamma函数，即对数gamma函数的一阶导数.

作业

用梯度下降法求如下peaks函数的极值，可视化结果并加以分析.

$$
z=3(1-x)^2e^{-x^2-(y+1)^2}-10\left(\frac{x}{5}-x^3-y^5\right)e^{-x^2-y^2}-\frac{1}{3}e^{-(x+1)^2-y^2}
$$

```
% f=@(x,y)3*(1-x).^2.*exp(-(x.^2) - (y+1).^2)- 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2)- 1/3*exp
% ezmesh(f);
peaks
```

```
z =  3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) ...
   - 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...
   - 1/3*exp(-(x+1).^2 - y.^2)
```