6.7.阻尼牛顿法

- 1.引入
  -
  注意到，牛顿法的迭代公式中没有步长因子，是定步长迭代。对于非二次型目标函数，有时候会出现$f(x_{k+1}) > f(x_k)$的情况，这表明，原始牛顿法不能保证函数值稳定的下降。在严重的情况下甚至会造成序列发散而导致计算失败。

  为消除这一弊病，人们又提出阻尼牛顿法。阻尼牛顿法每次迭代的方向仍然是x_k，但每次迭代会沿此方向做一维搜索，寻求最优的步长因子λ_k，即：

  $\lambda_k = minf(x_k + \lambda d_k)$
- 2.推导
  -
  - 1、给定初值x_0和精度阈值ε，并令$k = 0$;
  - 2、计算g_k（$f(x)$在x_k处的梯度值）和H_k;
  - 3、若$||g_k|| < \varepsilon$则停止迭代；否则确定搜索方向：$d_k = -H_k^{-1} \cdot g_k$;
  - 4、利用$d_k = -H_k^{-1} \cdot g_k$得到步长$\lambda_k$，并令$x_{k+1} = x_k + \lambda_k d_k$
  - 5、令$k = k + 1$，转至2。
- 3.python实现：

```
from linear_search.wolfe import *
from linear_search.Function import *
from numpy import *

# 阻尼牛顿法
def newton(f, start):
    fun = Function(f)
    x = array(start)
    g = fun.grad(x)
    while fun.norm(x) > 0.01:
        G = fun.hesse(x)
        d = (-dot(linalg.inv(G), g)).tolist()[0]
        alpha = wolfe(f, x, d)
        x = x + alpha * array(d)
        g = fun.grad(x)
    return x
```

6.8.拟牛顿法

- 1.概述：
  由于**牛顿法**每一步都要求解目标函数的**Hessen矩阵的逆矩阵**，**计算量比较大**（求矩阵的逆运算量比较大），因此提出一种**改进方法**，即**通过正定矩阵近似代替Hessen矩阵的逆矩阵，简化这一计算过程**，改进后的方法称为**拟牛顿法**。
- 2.推导

  先将目标函数在x_{k+1}处展开，得到：

  $$f(x) = f(x_{k+1}) + f'(x_{k+1})(x - x_{k+1}) + \frac{1}{2} f''(x_{k+1})(x - x_{k+1})^2$$

  两边同时取梯度，得：

  $$f'(x) = f'(x_{k+1}) + f''(x_{k+1})(x - x_{k+1})$$

  取上式中的$x = x_k$，得：

  $$f'(x_k) = f'(x_{k+1}) + f''(x_{k+1})(x - x_{k+1})$$

  即：

  $$g_{k+1} - g_k = H_{k+1} \cdot (x_{k+1} - x_k)$$

  可得：

  $$H_k^{-1} \cdot (g_{k+1} - g_k) = x_{k+1} - x_k$$

  上面这个式子称为**“拟牛顿条件”**，由它来对Hessen矩阵做约束。
- 3.python实现：

```
# coding=utf-8
from linear_search.wolfe import *
from linear_search.Function import *
from numpy import *


# 拟牛顿法
def simu_newton(f, start):
```

```
    n=size(start)
    fun = Function(f)
    x = array(start)
    g = fun.grad(x)
    B=eye(n)
    while fun.norm(x) > 0.01:
        d = (-dot(linalg.inv(B), g)).tolist()
        alpha = wolfe(f, x, d)
        x_d=array([alpha * array(d)])
        x = x + alpha * array(d)
        g_d=array([fun.grad(x)-g])
        g = fun.grad(x)
        B_d=dot(B,x_d.T)
        B=B+dot(g_d.T,g_d)/dot(g_d,x_d.T)-dot(B_d,B_d.T)/dot(x_d,B_d)
    return x
```