《数学建模与最优化》课程作业

刘铭宸

软件工程 2003 班

U202010783

2022 年 12 月 14 日

目录

1 最速下降法

1.1 方法介绍

在基本迭代公式 $x_{k+1} = x_k + t_k p_k$ 中，每次迭代搜索方向 p_k 取为目标函数 $f(x)$ 的负梯度方向，即 $p_k = -\nabla f(x_k)$，而每次迭代的步长 t_k 取为最优步长，由此所确定的算法称为最速下降法。最速下降法是法国数学家 Cauchy 于 1874 年提出的，是求解无约束最优化问题最早使用的方法之一，也是现代优化方法的基础。

1.2 迭代步骤

已知目标函数 $f(x)$ 及其梯度 $g(x)$，终止限 ε，最速下降法的迭代步骤如下：

1. 选定初始点 x_0，置 $k = 0$.
2. 计算 $g_k = g(x_k)$.
3. 若 $||g_k|| < \varepsilon$，则 $x^* = x_k$，输出 x^*，结束；否则令 $p_k = -g_k$，由一维搜索求步长 t_k，使得

$$f(x_k + t_k p_k) = min_t f(x_k + tp_k), t > 0 \tag{1}$$

4. 令 $x_{k+1} = x_k + t_k p_k$，置 $k = k + 1$，转 2.

1.3 编程实现

例：用最速下降法求函数 $f(x) = 2x_1^2 + x_2^2$ 的极小点。设初始点为 $x_0 = (1, 1)^T, \varepsilon = 1/10$.

根据老师关于最速下降法的介绍，编写代码如下：

```
import numpy as np
from sympy import *
import math

```

```
    x1, x2, t = symbols('x1, x2, t')

    def func():
        return 2*pow(x1, 2) + pow(x2, 2)

    def grad(data):
        f = func()
        grad_vec = [diff(f, x1), diff(f, x2)]
        grad = []
        for item in grad_vec:
            grad.append(item.subs(x1, data[0]).subs(x2, data[1]))
        return grad

    def grad_len(grad):
        vec_len = math.sqrt(pow(grad[0], 2) + pow(grad[1], 2))
        return vec_len

    def zhudian(f):
        t_diff = diff(f)
        t_min = solve(t_diff)
        return t_min

    def main(X0, theta):
        f = func()
        grad_vec = grad(X0)
        grad_length = grad_len(grad_vec)
        print("梯度模长",grad_length)
        k = 0
        print("x"+str(k)+"=(",X0[0],",",X0[1],")")
```

```
        data_x = [0]
        data_y = [0]
        while grad_length > theta:
            k += 1
            p = -np.array(grad_vec)
            X = np.array(X0) + t*p
            t_func = f.subs(x1, X[0]).subs(x2, X[1])
            t_min = zhudian(t_func)
            X0 = np.array(X0) + t_min*p
            grad_vec = grad(X0)
            grad_length = grad_len(grad_vec)
            print('梯度模长', grad_length)
            print("x"+str(k)+"=(",X0[0],",",X0[1],")")
            data_x.append(X0[0])
            data_y.append(X0[1])
        print("迭代次数: ",k)

    if __name__ == '__main__':
        main([1, 1], 0.1)
```

计算结果如下图:

```
梯度模长 4.47213595499958
x0=( 1 , 1 )
梯度模长 0.9938079899999065
x1=( -1/9 , 4/9 )
梯度模长 0.33126932999996883
x2=( 2/27 , 2/27 )
梯度模长 0.07361540666665974
x3=( -2/243 , 8/243 )
迭代次数: 3
函数极小值为: 8/6561
```

图 1: 最速下降法结果

1.4 结果分析

结果显示, 最速下降法总共迭代了 3 次, 最终求得函数极小值为 8/6561, 此时自变量 x 为 $(-2/243, 8/243)^T$。最终迭代误差为 0.07361540666665974, 显然达到了实验预设误差要求。

2 牛顿法

2.1 方法介绍

Newton 法的基本思想是利用目标函数 $f(x)$ 在迭代点 x_k 处的二次 Taylor 多项式作为二次函数，并用这个二次函数的极小点序列去逼近目标函数的极小点。

2.2 迭代步骤

已知目标函数 $f(x)$，终止限 ε，Newton 法的迭代步骤如下:

1. 选定初始点 x_0，置 $k = 0$.

2. 计算 $g_k = \nabla f(x_k)$.

3. 若 $||g_k|| < \varepsilon$，则 $x^* = x_k$，结束；否则计算 $G_k = G(x_k) = \nabla^2 f(x_k)$.

4. 由方程 $G_k p_k = -g_k$ 解出 p_k.

5. 令 $x_{k+1} = x_k + p_k$，置 $k = k + 1$，转 2.

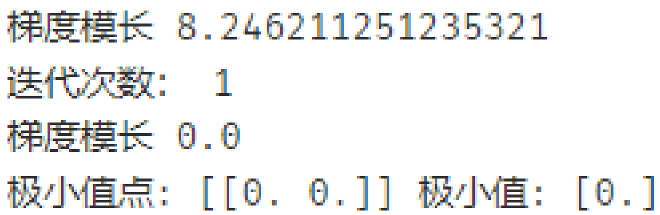
2.3 编程实现

例：试用 Newton 法求函数 $f(x_1, x_2) = x_1^2 + 4x_2^2$ 的极小点, 初始点为 $x_0 = (1, 1)^T$，精度要求 10^{-6}. 根据老师关于 Newton 法的介绍，编写代码如下：

```python
import numpy as np

def fun(x):
    return x[0] ** 2 + 4 * x[1] ** 2

def gfun(x):
    return np.array([2 * x[0], 8 * x[1]], dtype=
        float)

def Gfun(x):
    return np.array([[2, 0], [0, 8]], dtype=float)

def Newton(x0, eps=10 ** (-6)):
    xk, count = x0, 0
    print("梯度模长", np.linalg.norm(gfun(xk)))
    while np.linalg.norm(gfun(xk)) > eps:
        count += 1
        gk = gfun(xk)
        Gk = Gfun(xk)
        dk = -np.linalg.inv(Gk) @ gk
```

```
            xk = xk + dk
            print("迭代次数: ",count)
            print("梯度模长", np.linalg.norm(gfun(xk)))
        return xk, count

if __name__ == '__main__':
    x0 = np.array([[1], [1]])
    x = Newton(x0)
    print('极小值点:', x[0].T, '极小值:', fun(x[0]))
```

计算结果如下图:



梯度模长 8.246211251235321
迭代次数: 1
梯度模长 0.0
极小值点: [[0. 0.]] 极小值: [0.]

图 2: Newton 法结果

2.4 结果分析

结果显示,Newton 法总共迭代了 1 次, 最终求得函数极小值为 0, 极小值点 x 为 $(0,0)^T$, 最终迭代误差为 0, 得到了精确解, 显然达到了实验预设误差要求。

3 DFP 方法

3.1 方法介绍

DFP 方法是最早的拟牛顿法, 该算法的核心是: 通过迭代的方法, 对 H_{k+1}^{-1} 做近似, 迭代格式为 $D_{k+1} = D_k + \nabla D_k, k = 1, 2, \ldots$

3.2 迭代步骤

已知目标函数$f(x)$及其梯度$g(x)$，问题的维数n，终止限ε

（1）选定初始点x_0，置$H_0=I$.

（2）计算g_0，若$\|g_0\|<\varepsilon$，则输出$x^*=x_0$，结束；否则转（3）.

（3）取$p_0=-H_0g_0=-g_0$，置$k=0$，转（4）.

（4）一维搜索求t_k，使得$f(x_k+t_kp_k)=\min_{t\geq 0} f(x_k+tp_k)$，令$x_{k+1}=x_k+t_kp_k$，转（5）.

（5）计算g_{k+1}，若$\|g_{k+1}\|<\varepsilon$，则输出$x^*=x_{k+1}$，结束；否则转（6）.

（6）若$k+1=n$，令$x_0=x_n$，转（3）；否则，转（7）.

（7）计算

$$\delta_{k+1}=x_{k+1}-x_k,\quad \gamma_{k+1}=g_{k+1}-g_k$$

$$H_{k+1}=H_k+\frac{\delta_{k+1}\delta_{k+1}^{\mathrm{T}}}{\delta_{k+1}^{\mathrm{T}}\gamma_{k+1}}-\frac{H_k\gamma_{k+1}\gamma_{k+1}^{\mathrm{T}}H_k}{\gamma_{k+1}^{\mathrm{T}}H_k\gamma_{k+1}}$$

$$p_{k+1}=-H_{k+1}g_{k+1}$$

置$k=k+1$，转（4）.

图 3: DFP 方法迭代步骤

3.3 编程实现

例：用 DFP 方法求函数 $f(x_1,x_2)=x_1^2+4x_2^2$ 的极小点。初始点为 $x_0=(1,1)^T$，精度要求 10^{-6}.

根据老师关于 DFP 方法的介绍，编写代码如下：

```
import numpy as np
import sympy as sp
def jacobian(f,x):
    a,b=np.shape(x)
    x1,x2=sp.symbols('x1 x2')
    x3=[x1,x2]
    df=np.array([[0.00000],[0.00000]])
    for i in range(a):
        df[i,0]=sp.diff(f,x3[i]).subs({x1:x[0][0],
          x2:x[1][0]})
    return df

```

```
    def hesse(f,x):
        a,b=np.shape(x)
        x1,x2=sp.symbols('x1 x2')
        x3=[x1,x2]
        G=np.zeros((a,a))
        for i in range(a):
            for j in range(a):
                G[i,j]=sp.diff(f,x3[i],x3[j]).subs({x1
                    :x[0][0],x2:x[1][0]})
        return G

    def dfp_newton(f, x, iters):
        a = 1
        H = np.eye(2)
        G=hesse(f,x)
        epsilon = 1e-6
        for i in range(1, iters):
            g = jacobian(f, x)
            if i == 1:
                print("||g0||=",np.linalg.norm(g))
            else:
                print("||g(k+1)||=",np.linalg.norm(g))
            if np.linalg.norm(g) < epsilon:
                xbest=[]
                for a in x:
                    xbest.append(a[0])
                break
            d= -np.dot(H,g)
            a=-(np.dot(g.T,d)/np.dot(d.T,np.dot(G,d)))
            x_new = x +a*d
            print("第 %d 次迭代"%i)
```

```
            print("x=",x_new)
            g_new = jacobian(f, x_new)
            y = g_new - g
            s = x_new - x
            H=H+np.dot(s,s.T)/np.dot(s.T,y)-np.dot(H,
                np.dot(y,np.dot(y.T,H)))/np.dot(y.T,np.
                dot(H,y))
            G=hesse(f,x_new)
            x = x_new
        return xbest

x1,x2=sp.symbols('x1 x2')
x=np.array([[1],[1]])
f=x1**2+4*x2**2
X = dfp_newton(f,x,20)
print("极小值为：",X[0]**2+4*X[1]**2)
```

计算结果如下图:

```
||g0||= 8.246211251235321
第 1 次迭代
x= [[ 0.73846154]
 [-0.04615385]]
||g(k+1)||= 1.5223774617665211
第 2 次迭代
x= [[ 1.11022302e-16]
 [-5.55111512e-17]]
||g(k+1)||= 4.965068306494546e-16
极小值为:  2.465190328815662e-32
```

图 4: DFP 方法结果

3.4 结果分析

结果显示,DFP 方法总共迭代了 2 次, 最终求得函数的极小值为 2.465×10^{-32}，此时自变量 x 为 $(1.110\times10^{-16},-5.551\times10^{-17})^T$，最终迭代误差为 4.965×10^{-16}, 显然达到了实验预设误差要求。

感谢卢力老师一学期的教学!