Optimization Method Homework1

姓　　名：丁毅

专　　业：电信学部硕9065班

学　　号：3119105343

联系方式：1048585782@qq.com

1 问题描述

使用共轭梯度法求解高阶二次函数。共轭梯度法（Conjugate Gradient）是介于最速下降法与牛顿法之间的一个方法，它仅需利用一阶导数信息，但克服了最速下降法收敛慢的缺点，又避免了牛顿法需要存储和计算Hesse矩阵并求逆的缺点，共轭梯度法不仅是解决大型线性方程组最有用的方法之一，也是解大型非线性最优化最有效的算法之一。在各种优化算法中，共轭梯度法是非常重要的一种。其优点是所需存储量小，具有步收敛性，稳定性高，而且不需要任何外来参数。

使用共轭梯度法求解方程

$$f(x_1,x_2)=2x_1^2+x_2^2-2x_1x_2-4x_2 \tag{1}$$

的极小点和极小值。

2 求解

解：设定初始点为$[1,1]^T$, 迭代精度为$\epsilon=0.001$

1)第一次沿负梯度方向进行搜索

计算初始点的梯度：

$$\nabla f\left(\boldsymbol{x}^0\right)=\left[\begin{array}{c}4x_1-2x_2\\2x_2-2x_1-4\end{array}\right]_{x^0}=\left[\begin{array}{c}2\\-4\end{array}\right]$$

$$x^1=x^0+\alpha_0 d^0=\left[\begin{array}{c}1\\1\end{array}\right]+\alpha_0\left[\begin{array}{c}2\\-4\end{array}\right]=\left[\begin{array}{c}1+2\alpha_0\\1-4\alpha_0\end{array}\right]$$

一维最佳搜索步长应满足：

$$f\left(x^1\right)=\min_\alpha f\left(x^0+\alpha d^0\right)=\min_\alpha\left(40\alpha^2+20\alpha-3\right)$$

此时：

$$\alpha_0=0.25\quad \boldsymbol{x}^1=\left[\begin{array}{c}0.5\\2\end{array}\right]\quad \nabla f\left(\boldsymbol{x}^1\right)=\left[\begin{array}{c}-2\\-1\end{array}\right]$$

2)第二次迭代

$$\beta_0=\frac{\left\|\nabla f\left(\boldsymbol{x}^1\right)\right\|^2}{\left\|\nabla f\left(\boldsymbol{x}^0\right)\right\|^2}=\frac{5}{20}=0.25$$

$$\boldsymbol{d}^1=-\nabla f\left(\boldsymbol{x}^1\right)+\beta_0\boldsymbol{d}^0=\left[\begin{array}{c}1.5\\2\end{array}\right]$$

$$x^2=x^1+\alpha d^1=\left[\begin{array}{c}0.5\\2\end{array}\right]+\alpha\left[\begin{array}{c}1.5\\2\end{array}\right]=\left[\begin{array}{c}0.5+1.5\alpha\\2+2\alpha\end{array}\right]$$

带入目标函数：

$$\begin{aligned}f(x)=&2(0.5+1.5\alpha)^2+(2+2\alpha)^2-\\&2(0.5+1.5\alpha)(2+2\alpha)-4(2+2\alpha)=\phi(\alpha)\end{aligned}$$

求解得$\alpha=1$

因此得：

$$\boldsymbol{x}^2=\left[\begin{array}{c}2\\4\end{array}\right],\quad f\left(\boldsymbol{x}^2\right)=-8,\quad \nabla f\left(\boldsymbol{x}^2\right)=\left[\begin{array}{c}0\\0\end{array}\right]$$

因为：$\left\|\nabla f\left(\boldsymbol{x}^2\right)\right\|=0<\varepsilon$

迭代收敛。

因此,目标函数得极小点为$[2,4]^T$,极小值为-8。

3 程序

首先是共轭梯度算法函数：

```
function [f_v, best_x, iteration_num] = conjungate_gradient(f, x, x0, epsilon)
%input
% f - object functiuon
% x - var
% x_0 - Initial point
% epsilon - tolerance

%output
% f_v - minimum f value
% best_x - mimimum point
% iteration_num - iteration count
%% init
syms betas

n = length(x)

n_f = cell(1, n)

for i = 1 : n
    n_f{i} = diff(f, x{i});
end

%initial gradient
n_fv = subs(n_f, x, x0);

n_fv_pre = n_fv;


count = 0;
k = 0;
x_v = x0;

%initial search direction
d = - n_fv;

fprintf('Initial:\n');
fprintf('f = %s, x0 = %s, epsilon = %f\n\n', char(f), num2str(x0), epsilon);

while (norm(n_fv) > epsilon)
    x_v = x_v + betas * d;
    phi = subs(f, x, x_v);
    n_phi = diff(phi);
    beta = solve(n_phi);
    beta = double(beta);


    if beta < 1e-5
        break;
    end

    x_v = subs(x_v, betas, beta);
    x_v = double(x_v);
    n_fv = subs(n_f, x, x_v);
    count = count + 1;
    k = k + 1;
```

```
        alpha = sumsqr(n_fv) / sumsqr(n_fv_pre);

        fprintf('Iteration: %d\n', count);
        fprintf('x(%d) = %s, lambda = %f\n', count, num2str(x_v), beta);
        fprintf('nf(x) = %s, norm(nf) = %f\n', num2str(double(n_fv)), norm(double(n_fv)));
        fprintf('d = %s, alpha = %f\n', num2str(double(d)), double(alpha));
        fprintf('\n');

        d = - n_fv + alpha * d;
        n_fv_pre = n_fv;
        if k >= n
            k = 0;
            d = - n_fv;
        end
    end

    f_v = double(subs(f, x, x_v));
    best_x = double(x_v);
    iteration_num = count;

    end
```

使用上述函数进行测试：

```
syms x1 x2;
f = 2 * x1^2 + x2^2 - 4*x2 - 2*x1*x2;
x = {x1, x2};

x0 = [1 1];

epsilon = 1e-8;

[bestf, bestx, count] = conjungate_gradient(f, x, x0, epsilon);
% print result
fprintf('bestx = %s, bestf = %f, count = %d\n', num2str(bestx), bestf, count);
```

4 结果

```
Initial:
f = 2*x1^2 - 2*x1*x2 - 4*x2 + x2^2, x0 = 1  1, epsilon = 0.000000

Iteration: 1
x(1) = 0.5            2, lambda = 0.250000
nf(x) = -2 -1, norm(nf) = 2.236068
d = -2  4, alpha = 0.250000

Iteration: 2
x(2) = 2  4, lambda = 1.000000
nf(x) = 0  0, norm(nf) = 0.000000
d = 1.5            2, alpha = 0.000000

bestx = 2  4, bestf = -8.000000, count = 2
```

Figure 1: 程序运行结果1

可以看出，程序在进行2次迭代后收敛，并且求得得结果和精确解一致。
而面对更复杂的函数如：

$$f(x_1, x_2) = (x_1 - 1)^4 + (x_1 - x_2)^2$$

则需要更多的迭代次数才能够收敛：

```
命令行窗口
nf(x) = -0.0014077  -0.0016788, norm(nf) = 0.002191
d = 0.022884   -0.019188, alpha = 0.005382

Iteration: 8
x(8) = 0.97476      0.97586, lambda = 43.429293
nf(x) = -0.0022668    0.0022025, norm(nf) = 0.003161
d = 0.0015309    0.0015756, alpha = 2.080989

Iteration: 9
x(9) = 0.97533      0.97531, lambda = 0.249812
nf(x) = -2.9597e-05 -3.0461e-05, norm(nf) = 0.000042
d = 0.0022668  -0.0022025, alpha = 0.000181

bestx = 0.97533      0.97531, bestf = 0.000000, count = 9
```

Figure 2: 程序运行结果2

5 总结

- 由结果看出，程序经过两次迭代后就收来奶得到了结果，证明共轭梯度法收敛速度较快，计算量较小，稳定性高，一般来说，n维的优化问题迭代n次就可以收敛。
- 选择不同地初始点坐标，经过迭代后得到的结果是一样的，说明共轭梯度法的结果不受初始点选择的影响。

- 迭代精度越高时，程序的运行时间越长，迭代次数越多，因此在解决实际问题中我们需要选择一个合适的迭代精度，充分利用计算的效率。