刘程华 2018011687 计91

1、分别利用 l_1, l_2, l_∞ 范数的最速下降方向求解下述无约束优化问题

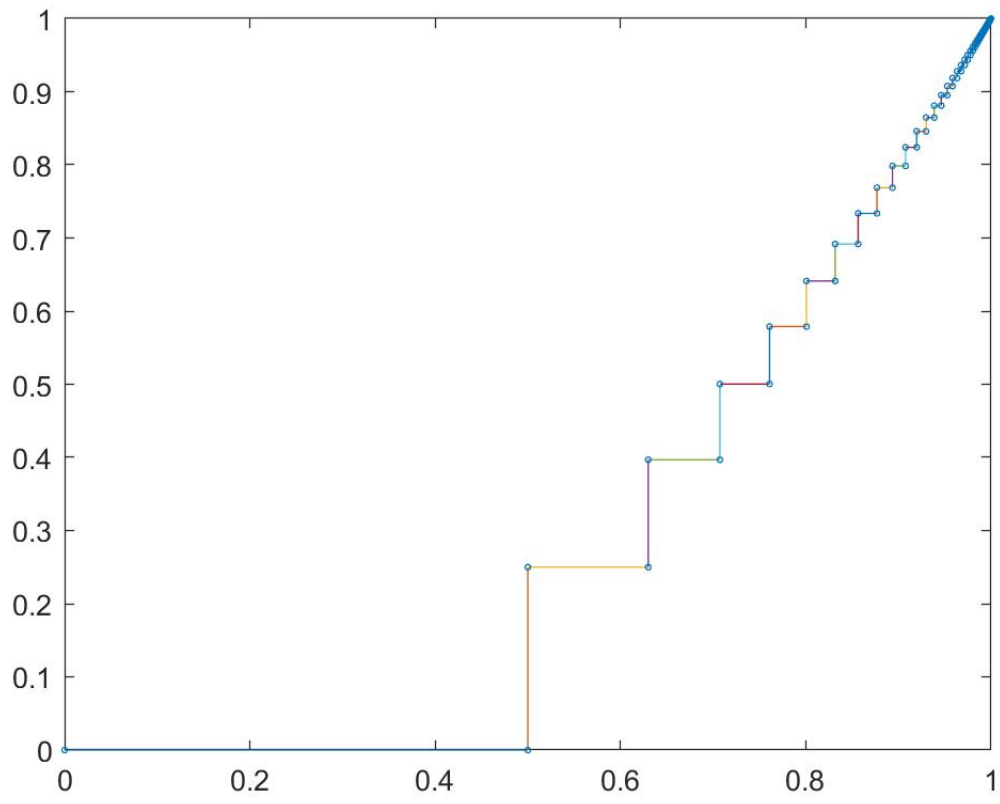
$$\min_{x \in R^2} f(x) = (1 - x_1)^2 + 2(x_2 - x_1^2)^2$$

直线搜索均采用精确直线搜索（ 0.618 法 ） 。初始点取为 $x^0 = (0,0)^T$, 停止准则为 $\|\nabla f(x)\|_2 \le 10^{-8}$ 。要求画出迭代点 x^k 在 2 维平面上的轨迹（将每个点连线）以及目标函数值 $f\left(x^k\right)$ 关于迭代次数 k 的图像。

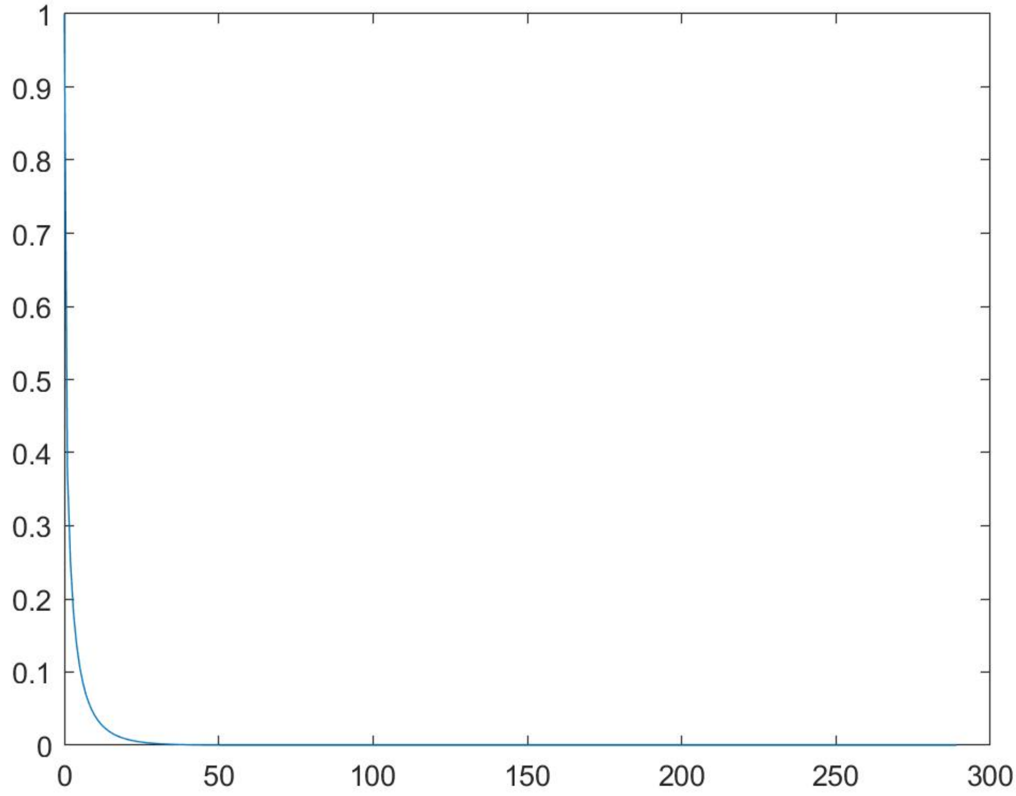
最优值为0，最优解为 [1,1]

1.1 l_1 范数

迭代点 x^k 在 2 维平面上的轨迹：

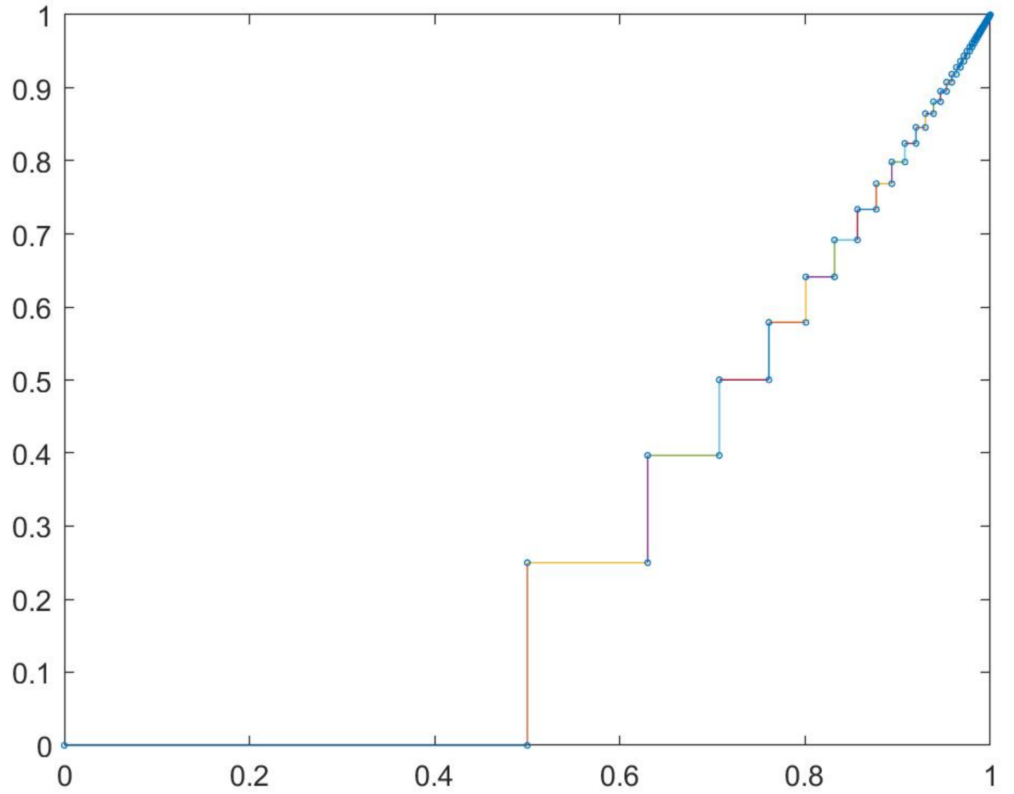


函数值 $f\left(x^k\right)$ 关于迭代次数 k 的图像：

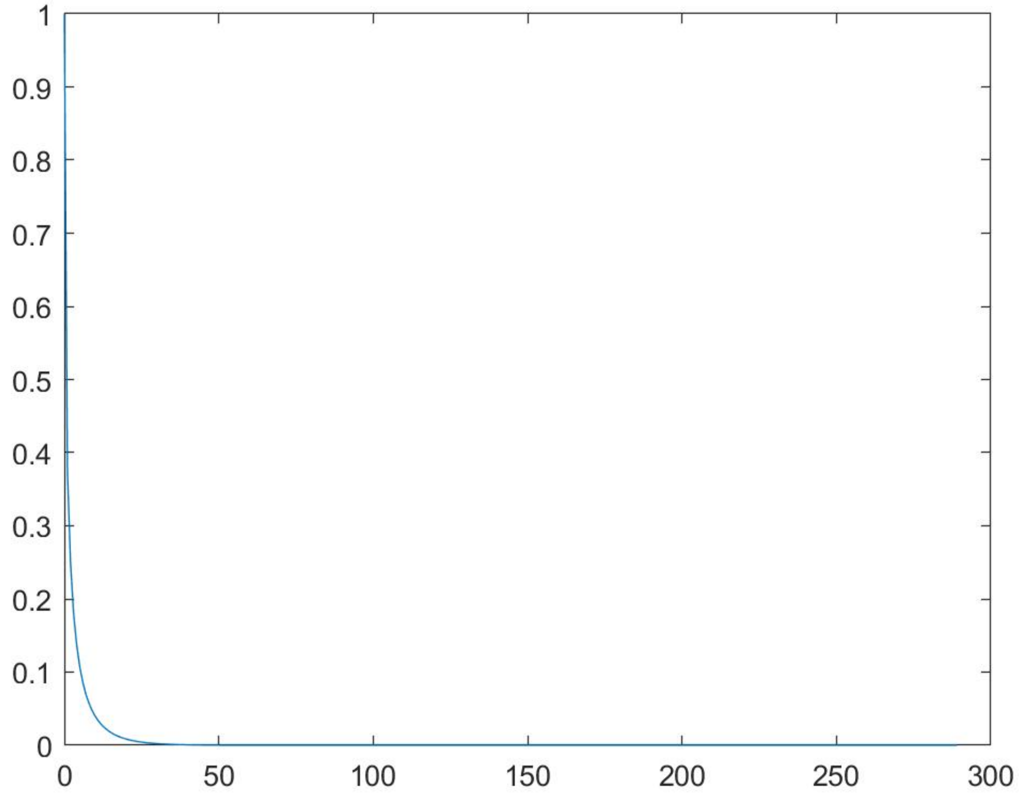


1.2 l_2范数

迭代点 x^k 在 2 维平面上的轨迹：

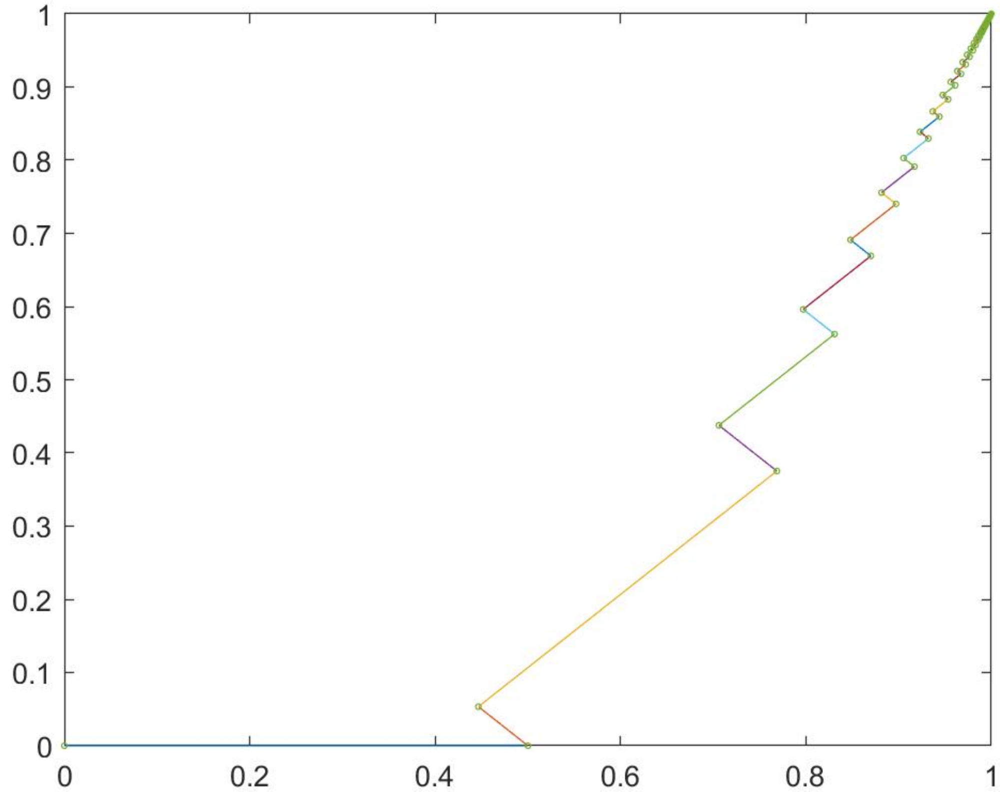


函数值 $f\left(x^k\right)$ 关于迭代次数 k 的图像：

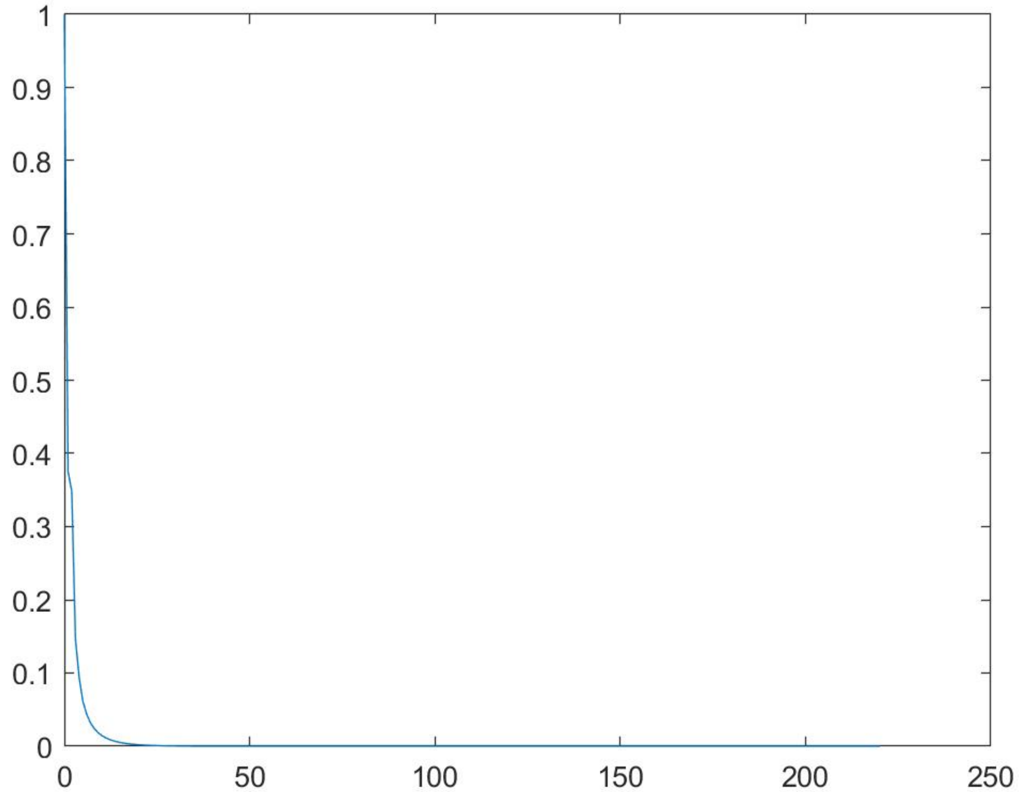


1.3 l_∞ 范数

迭代点 x^k 在 2 维平面上的轨迹：



函数值 $f\left(x^k\right)$ 关于迭代次数 k 的图像：



2、考虑无约束优化问题:

$$\min \quad f(x)=-\sum_{i=1}^{m} \log\left(1-a_i^T x\right)-\sum_{i=1}^{n} \log\left(1-x_i^2\right)$$

其中 $x \in R^n, \quad$ **dom** $f=\left\{x \left| a_i^T x<1, i=1, \cdots, m ; \right| x_i \mid<1, i=1, \cdots, n\right\}$ **用 Newton 法并结合回溯直线搜索求解上述** $f(x)$ **在** $m=50, n=50$ **和** $m=100, n=100$ **两种规模下的最优解** x^* **和最优值** p^* **。请合理选择回溯参数,要求停止误差为** $\|\nabla f(x)\|_2 \leq 10^{-8}$**,分别画出对数误差** $\log\left(f\left(x^k\right)-p^*\right)$ **和迭代步长** t^k **关于迭次次数** k **的图像。**

对目标函数求导我们有:

$$\frac{\partial f}{\partial x_j}=\sum_{i=1}^m \frac{a_{ij}}{1-a_i^T x}+\frac{2 x_j}{1-x_j^2}$$

$$\frac{\partial^2 f}{\partial x_j^2}=\sum_{i=1}^m \frac{a_{ij}^2}{\left(1-a_i^T x\right)^2}+\frac{2\left(1+x_j^2\right)}{\left(1-x_j^2\right)^2}$$

$$\frac{\partial^2 f}{\partial x_j \partial x_k}=\sum_{i=1}^m \frac{a_{ij} a_{ik}}{\left(1-a_i^T x\right)^2}$$

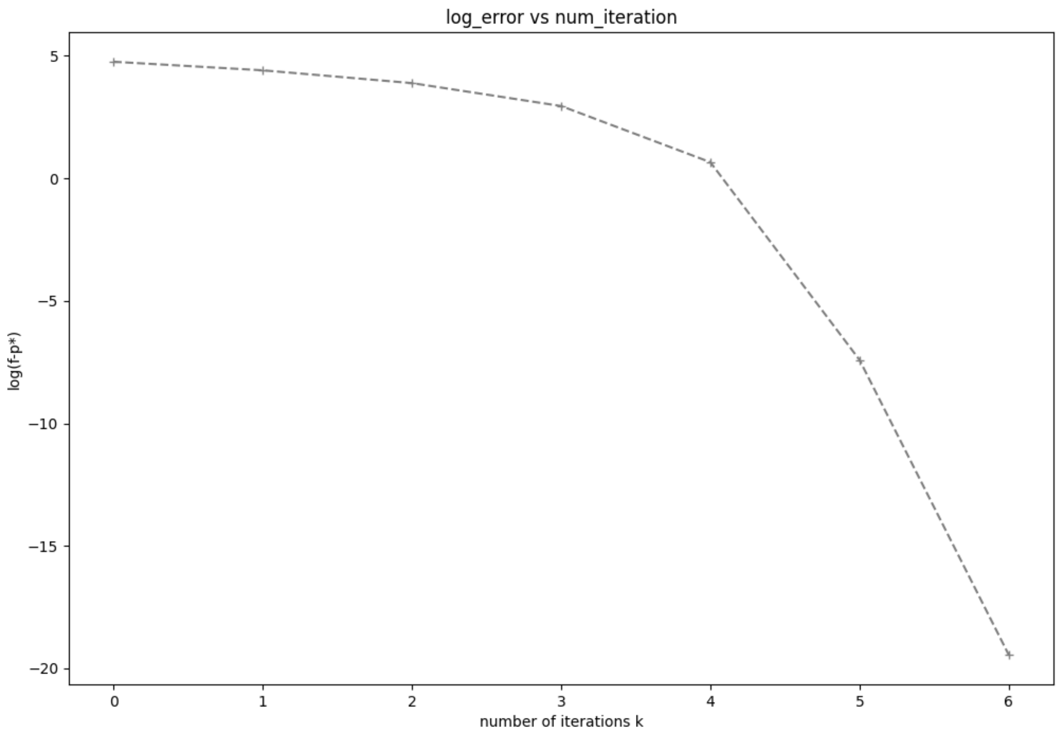
$m=50, n=50$

回溯法中 $\alpha=0.25, \beta=0.8$, 我们得到 $p^*=-116.62827595746981$

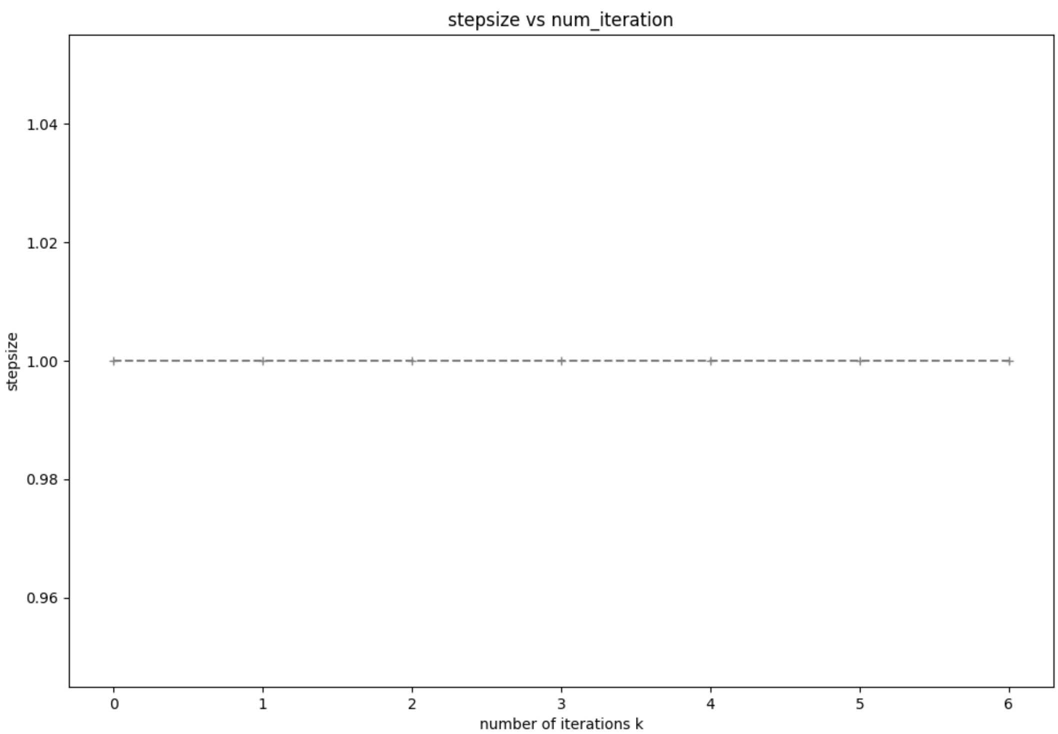
x^* :

```
[-0.59989229 -0.55790326 -0.52910835 -0.55619739 -0.5392275  -0.58680392
 -0.55272886 -0.54720767 -0.55931505 -0.56023489 -0.5617475  -0.56577888
 -0.59808214 -0.54125764 -0.52454361 -0.54236799 -0.54507625 -0.49091829
 -0.55484647 -0.58451957 -0.5536803  -0.58982603 -0.56181038 -0.59386582
 -0.55305731 -0.6049877  -0.61402046 -0.55671553 -0.56123833 -0.57006356
 -0.50505816 -0.53908902 -0.53748288 -0.58624017 -0.5741891  -0.58429682
 -0.53495476 -0.58359626 -0.56892761 -0.59013854 -0.57374283 -0.60052892
 -0.58193671 -0.5560532  -0.57672612 -0.53851854 -0.57880576 -0.53185222
 -0.56575694 -0.56221889]
```

对数误差 $\log\left(f\left(x^k\right)-p^*\right)$ 关于迭次次数 k 的图像:



迭代步长 t^k 关于迭次次数 k 的图像:



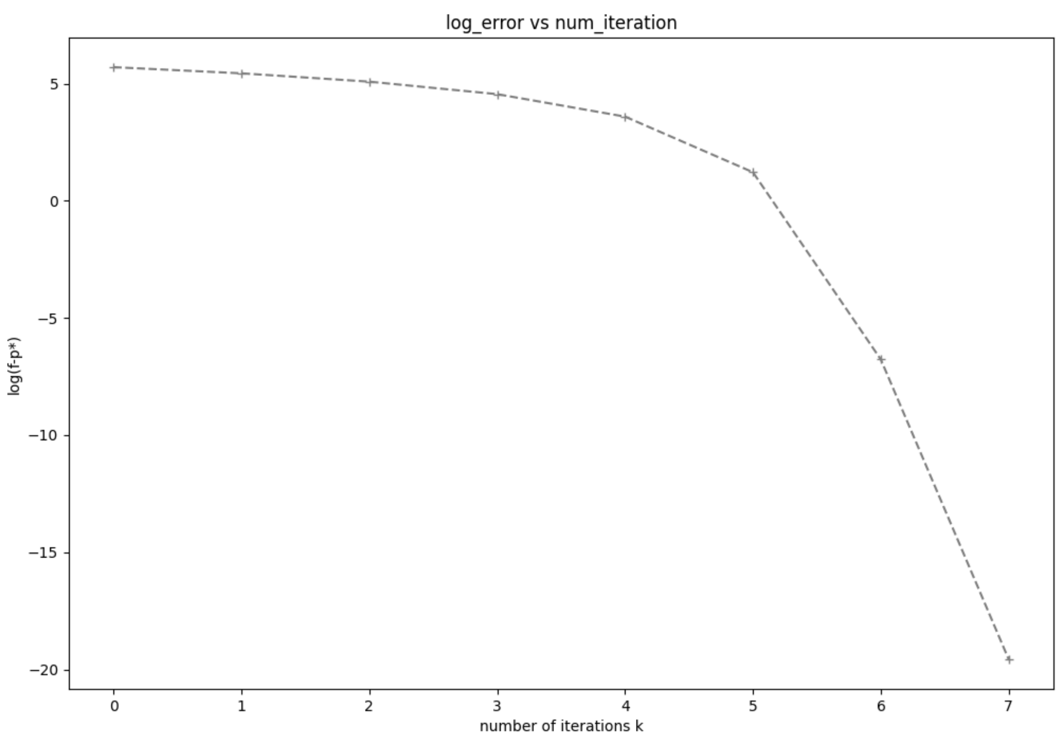
$m = 100, n = 100$

回溯法中 $\alpha = 0.25, \beta = 0.8$, 我们得到 $p^* = -298.83984990703334$

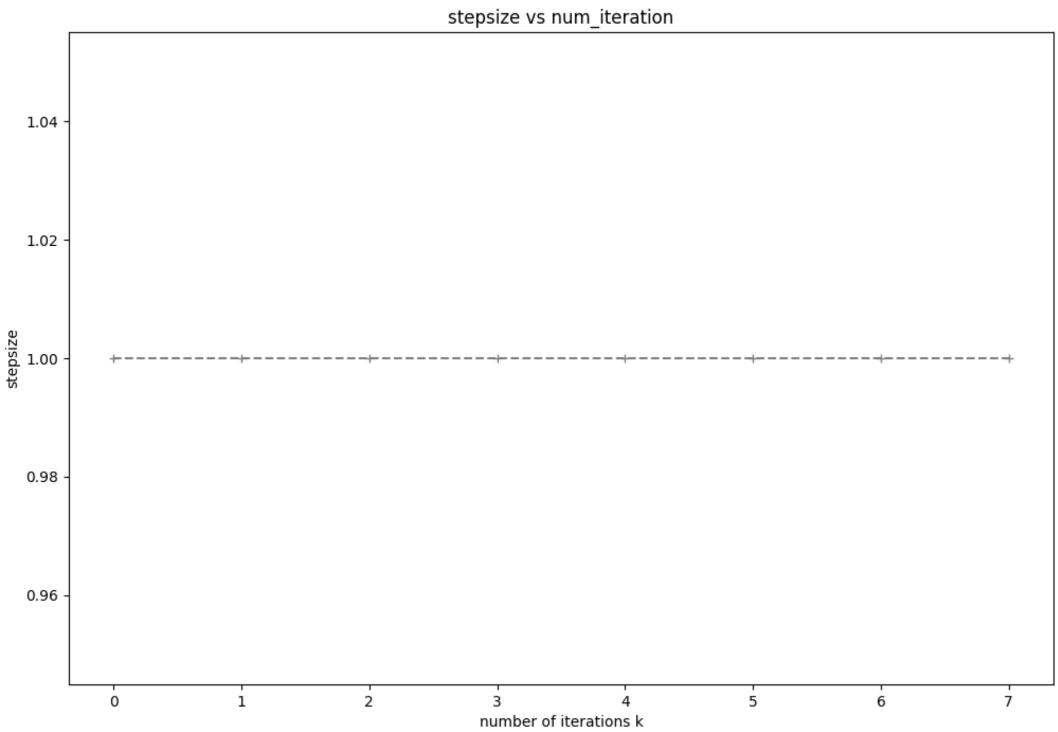
x^* :

```
[-0.57595503 -0.58924811 -0.56501433 -0.5856453  -0.58914572 -0.55424797
 -0.57378666 -0.58428098 -0.59553611 -0.57106049 -0.58293674 -0.57231097
 -0.58196992 -0.57383663 -0.58458882 -0.57210716 -0.57092246 -0.57829367
 -0.54449074 -0.57294463 -0.57526659 -0.57219743 -0.5940691  -0.56707064
 -0.56319415 -0.57049631 -0.58257559 -0.56673603 -0.55355142 -0.56457683
 -0.55628293 -0.54669836 -0.56466356 -0.56754889 -0.56074083 -0.55114558
 -0.57210261 -0.58599356 -0.5669856  -0.54502515 -0.58933532 -0.58815644
 -0.57527897 -0.58058042 -0.56595005 -0.56239337 -0.57378925 -0.56323303
 -0.5413641  -0.57920111 -0.57848995 -0.57324296 -0.57291018 -0.58019014
 -0.55231889 -0.58263743 -0.56829639 -0.55370906 -0.5441478  -0.56330905
 -0.55708437 -0.57343976 -0.52387468 -0.55998829 -0.57334106 -0.55879304
 -0.56426066 -0.57403509 -0.60271864 -0.56920873 -0.54396938 -0.61321573
 -0.59682928 -0.56315768 -0.55149225 -0.56460546 -0.56829515 -0.58568047
 -0.55833897 -0.58119791 -0.61032633 -0.59190966 -0.59953304 -0.56166136
 -0.56583814 -0.55563325 -0.55902644 -0.58689323 -0.57737758 -0.58564442
 -0.56297376 -0.5635474  -0.56044998 -0.51945539 -0.5506375  -0.56552479
 -0.57546758 -0.56041577 -0.56774534 -0.56650011]
```

对数误差 $\log\left(f\left(x^k\right)-p^*\right)$ 关于迭次次数 k 的图像:



迭代步长 t^k 关于迭次次数 k 的图像:



附录：

Code1：

Gradient_descend.m:最速下降法的实现，在调用GD_finddir()时，第二个参数选择1，2，3分别表示 一范数，二范数和无穷范数

```
point = [double(0.0),double(0.0)];
epison = 10^-8;
MAX_ITER = 100000;
prime1 = double(0.0);
prime2 = double(0.0);
[prime1,prime2] = fprime(point);
disp(prime1);
disp(prime2);
prime = [prime1,prime2];
Ys =  f(point);
Iters = 0;
dir1 = double(0.0);
dir2 = double(0.0);
ans1=zeros(300,1);
ans2=zeros(300,1);
i=1;
while(true)
    ans1(i,1)=point(1,1);
    ans2(i,1)=point(1,2);
```

```
    if(double(sqrt(prime1^2 +prime2^2)) <= epison)
        break%&& Iters<= MAX_ITER
    end
    disp(log10(norm(prime,2)) );
    [dir1,dir2] = GD_finddir(point,1);
    dir = [dir1,dir2];
    t = linearSearch(point,dir,1,0.0001);
    point = point + t.*dir;
    [prime1,prime2] = fprime(point);
    prime = [prime1,prime2];
    Ys = [Ys f(point)];
    Iters = Iters + 1;
    i=i+1;
end
disp(point);
T=[ans1,ans2]
Iters = 0:Iters;
A=zeros(2,2);
for i=1:200
    A(1,1)=T(i,1);
    A(1,2)=T(i,2);
    A(2,1)=T(i+1,1);
    A(2,2)=T(i+1,2);
    plot(A(:,1),A(:,2));
    hold on
end
 scatter(T(:,1),T(:,2),4)
```

GD_finddir.m:3种范数最速下降法寻找方向的函数

```
function [outputArg1,outputArg2] = GD_finddir(X,fanshu)
%pfpx = 2*(x1-1) - 8*x1*(x1*x1 - x2);
%pfpy = 4*(x2-x1*x1);
pfpx = double(0);
pfpy = double(0);
[pfpx ,pfpy] = fprime(X);
switch fanshu
    case 1
        if abs(pfpx) > abs(pfpy)
            outputArg1 = sign(-pfpx);
            outputArg2 = 0;
        else
            outputArg1 = 0;
            outputArg2 = sign(-pfpy);
        end
    case 2
        outputArg1 = -pfpx/sqrt(pfpx^2 +pfpy^2) ;
        outputArg2 = -pfpy/sqrt(pfpx^2 +pfpy^2) ;
```

```
    otherwise
        outputArg1 =- sign(pfpx);
        outputArg2 =- sign(pfpy);
end
```

f.m:原函数

```
function outputArg = pfpt(X, D ,t)
    x1 = X(1); x2 = X(2);dir1 = D(1); dir2 = D(2);
    outputArg = 2*dir1*(x1+t*dir1-1) + 4*(x2+t*dir2-(x1+t*dir1)^2) * (dir2-2*dir1*(x1+t*dir1));
end
```

fprime.m:原函数的导数

```
function [outputArg1,outputArg2] = fprime(X)
x1 = double(X(1)); x2 = double(X(2));
outputArg1 = double(2*(x1-1) + 8*x1*(x1^2 - x2));
outputArg2 = double(4*(x2-x1^2));
end
```

linearSearch.m:一维精确搜索

```
function tout = linearSearch(X,D,l,precision)
 a=0;b=l;
 t=(sqrt(5)-1)/2;
 h=b-a;
 delta=10^-4;
 phia=double(pfpt(X,D,a));
 phib=double(pfpt(X,D,b));
 p=a+(1-t)*h;
 q=a+t*h;
 phip=double(pfpt(X,D,p));
 phiq=double(pfpt(X,D,q));
 k=1;
 while(abs(phib-phia)>precision)|(h>delta)
   if(phip<phiq)
       b=q;
       phib=phiq;
       q=a+t*(b-a);
       phiq=double(pfpt(X,D,q));
       h=b-a;
       p=a+(1-t)*h;
       phip=double(pfpt(X,D,p));
   else
       a=p;
       phia=phip;
       p=q;
```

```
        phip=phiq;
        h=b-a;
        q=a+t*h;
        phiq=double(pfpt(X,D,q));
    end
    k=k+1
  end
  tout=a;
end
```

pfpt.m:一维函数的对t一阶导数

```
Gilmour  23:35:17
function outputArg = pfpt(X, D ,t)
    x1 = X(1); x2 = X(2);dir1 = D(1); dir2 = D(2);
    outputArg = 2*dir1*(x1+t*dir1-1) + 4*(x2+t*dir2-(x1+t*dir1)^2) * (dir2-2*dir1*(x1+t*dir1));
end
```

Code2：

```
import numpy as np
import matplotlib.pyplot as plt
import mat4py
import math

class Obj_Func(object):

    def __init__(self, a):
        self.a = np.array(a).T

    def Check(self, x):
        temp = np.matmul(np.transpose(self.a), x)
        if np.any(temp >= 1):
            return False
        elif np.any(x ** 2 >= 1):
            return False
        else:
            return True

    def Cal(self, x):
        value = - np.sum(np.log(1. - np.matmul(np.transpose(self.a), x))) - np.sum(np.log(1 - x ** 2))
        return value

```

```
    def Gradient(self, x):
        tmp = self.a / (1 - np.matmul(np.transpose(self.a), x))
        gradient = np.sum(tmp, axis=1) + 2 * x / (1 - x ** 2)
        return gradient

    def Hessian(self, x):
        denom = (1 - np.matmul(np.transpose(self.a), x)) ** 2
        tmp = np.sum(self.a ** 2 / denom, axis=1) + 2 * (1 + x ** 2) / ((1 - x** 2) ** 2)
        hessian = np.diag(tmp)
        for j in range(len(x)):
            for k in range(j + 1, len(x)):
                hessian[j, k] += np.sum(self.a[j, :] * self.a[k, :] / denom)
        hessian = hessian + hessian.T - np.diag(tmp)
        return hessian


class Solve(object):
    def __init__(self, obj_func):
            self.obj_fun = obj_func
            self.process = []
            self.stepsize = []

    def Backtracking(self, x, direction, gradient, alpha=0.25, beta=0.8):
        t = 1
        while True:
            left = x - t * direction
            if self.obj_fun.Check(left):
                if self.obj_fun.Cal(left) <= self.obj_fun.Cal(x) - alpha * t * np.dot(gradient, direction):
                    break
                t *= beta
        print("Backtracking finished")
        return t

    def Search(self, initial):
        self.process = []
        if not self.obj_fun.Check(initial):
            raise ValueError("Initial point is infeasible")
        else:
            ctr = 0
            current_x = initial
            while True:
                ctr += 1
                print("Iteration {:d}".format(ctr))
                gradient = obj_fun.Gradient(current_x)
                self.process.append(self.obj_fun.Cal(current_x))
                if np.linalg.norm(gradient, 2) <= 1e-8:
```

```
                        break
                    hessian = obj_fun.Hessian(current_x)
                    direction = np.dot(np.linalg.inv(hessian), gradient)
                    t = self.Backtracking(current_x, direction, gradient)
                    self.stepsize.append(t)
                    current_x -= t * direction
                    print("stepsize:", t)
                    print("gradient norm", np.linalg.norm(gradient, 2))
        print("Total number of iterations {:d}".format(ctr))
        print("Answer", self.process[-1])
        print(x)
        return x

    def Plot(self, fname):
        optimum = self.process[-1]
        y = [math.log(f - optimum) for f in self.process[:-1]]
        plt.figure(figsize=(12, 8))
        plt.plot(y, color="gray", linestyle='--', marker='+')
        plt.xlabel("number of iterations k")
        plt.ylabel("log(f-p*)")
        plt.title("log_error vs num_iteration")
        plt.savefig(fname + "k.png")
        plt.figure(figsize=(12, 8))
        plt.plot(self.stepsize, color="gray", linestyle='--', marker='+')
        plt.xlabel("number of iterations k")
        plt.ylabel("stepsize")
        plt.title("stepsize vs num_iteration")
        plt.savefig(fname + "t.png")


if __name__ == "__main__":
    data = mat4py.loadmat('Homework_9_DATA.mat')
    obj_fun = Obj_Func(data['A_50'])
    solve1 = Solve(obj_fun)
    x = np.zeros(50)
    solve1.Search(x)
    solve1.Plot('50_')
    obj_fun = Obj_Func(data['A_100'])
    solve2 = Solve(obj_fun)
    x = np.zeros(100)
    solve2.Search(x)
    solve2.Plot('100_')
```