HOMEWORK 4

Dingcheng

Problem 1 (By hand): (10 points)

Consider the nonlinear equation

$$f(x) = x^2 - 2 = 0$$

(a) With $x_0 = 1$ as a starting point, what is the value of x_1 if you use Newton's method for solving this problem?

$$f'(x) = 2x$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1 + \frac{1}{2} = 1.5$$

(b) With $x_0 = 1$ and $x_1 = 2$ as starting points, what is the value of x_2 if you use the secant method for the same problem?

$$x_2 = x_1 - f(x_1)(x_1 - x_0)/(f(x_1) - f(x_0)) = 0$$

Problem 2: (20 points)

For the equation

$$x^2 - 3x + 2 = 0$$

each of the following functions yields an equivalent fixed-point problem:

$$\begin{aligned} g_1(x) &= (x^2+2)/3 \\ g_2(x) &= \sqrt{3x-2} \\ g_3(x) &= 3-2/x \\ g_4(x) &= (x^2-2)/(2x-3) \end{aligned}$$

(a) Analyze the convergence properties of each of the corresponding fixed-point iteration schemes for the root $x=2$ by considering $|g_i'(2)|$.

$|g_1'(2)| = 4/3 > 1$ it will not converge to 2
$|g_2'(2)| = 1/2 < 1$ it will converge, with rate 2
$|g_i'(2)| = 7/2 > 1$ it will not converge
$|g_i'(2)| = 0$ it will converge quadratically.

(b) Confirm your analysis by implementing each of the schemes and verifying its convergence (or lack thereof) and approximate convergence rate.

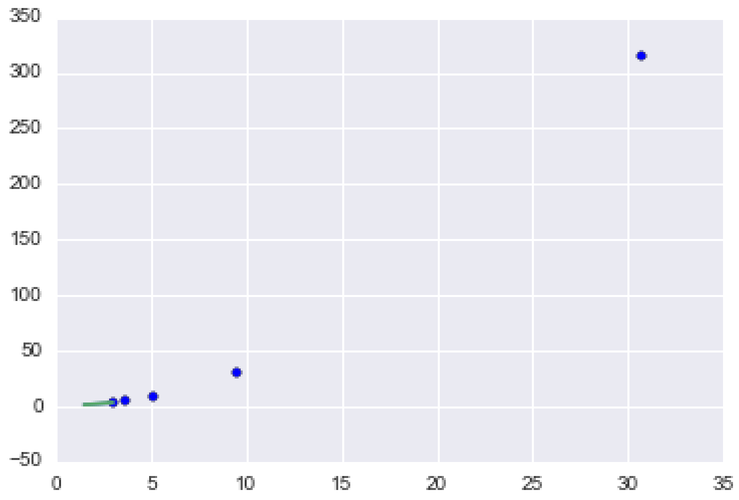
The code is given below. It produce the graph of how the each function is approaching and how the error is changing.

```
from math import sqrt
import matplotlib.pyplot as plt
import seaborn
import numpy as np

def fixpoint(f, iter_times):
    x = [3.0]
    for i in range(iter_times):
        x.append(f(x[-1]))
    return np.array(x)

g1 = lambda x: (x*x+2)/3
g2 = lambda x: np.sqrt(3*x-2)
g3 = lambda x: 3-2/x
g4 = lambda x: (x*x-2)/(2*x-3)

def scatterplot(g, points):
    x = np.linspace(1.5, 3.0)
    plt.plot(x,x)
    plt.plot(x,g(x))
    plt.scatter(points[:-1], points[1:])
    plt.show()

def errorplot(points, value):
    plt.semilogy(np.arange(len(points)),np.abs(points-value))
    print np.abs(points-value)
```

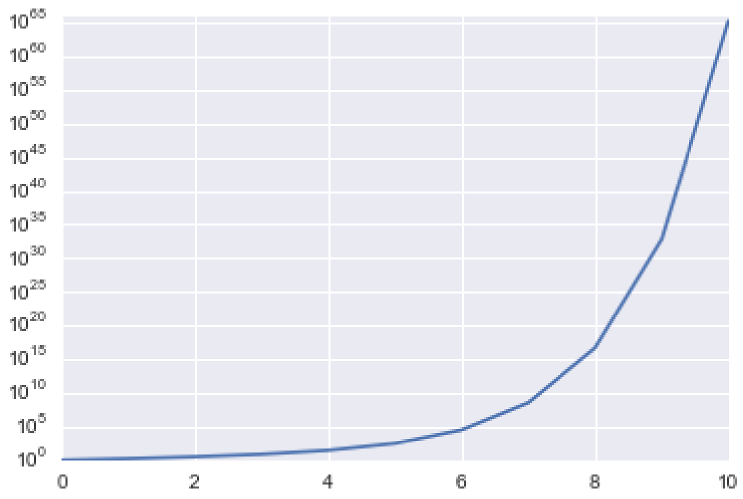
```
        plt.show()

    %matplotlib inline
    for g in [g1, g2, g3, g4]:
        scatterplot(g1, fixpoint(g, 5))
        errorplot(fixpoint(g,10), 2)
```

The result are shown as below.

The first 5 iteration results for first function.

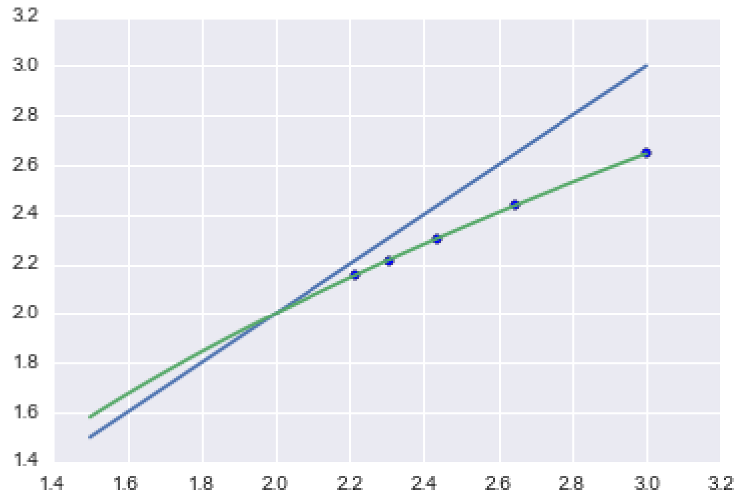


The rate of the error change (semi-log)

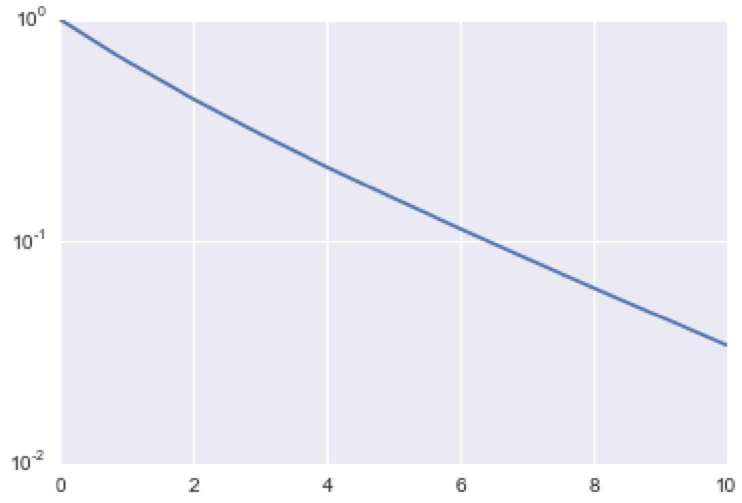


As we can clearly see it is obviously divergent

The first 5 iteration results for the second function

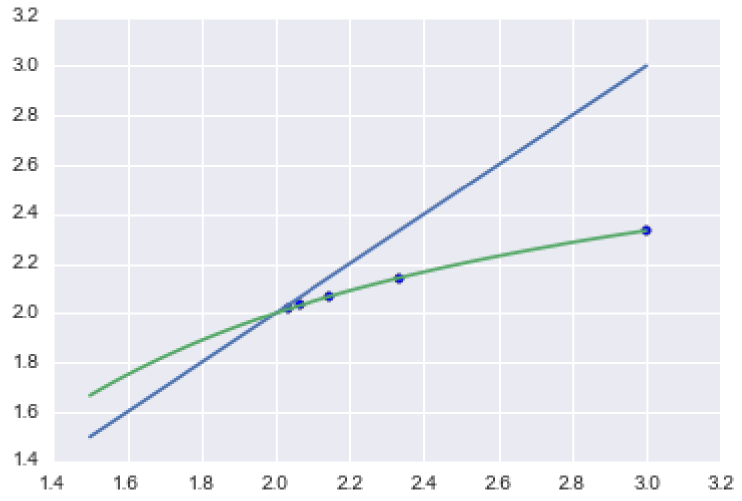


The rate of error change (semi-log)

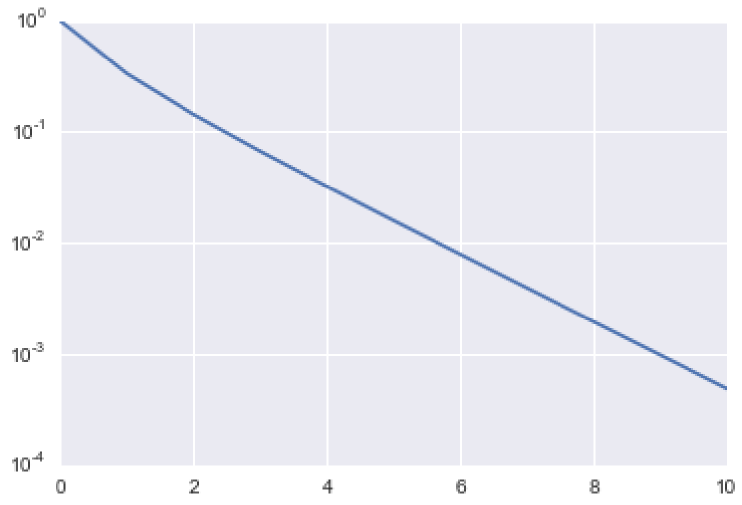


As we can clearly see, it is convergent

The first 5 iteration results for the third function

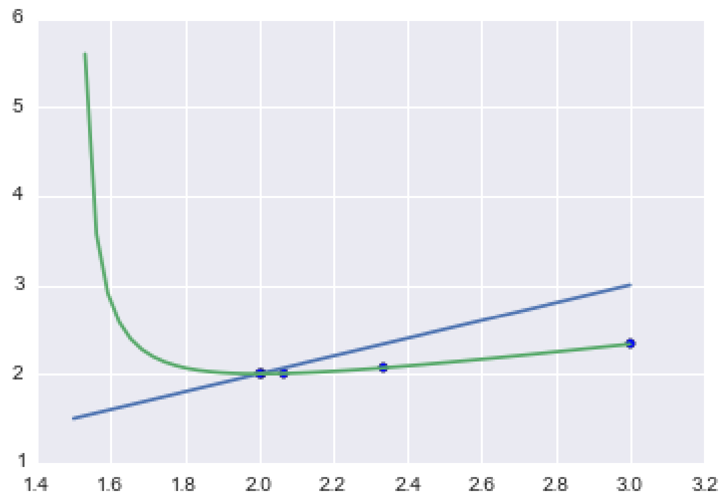


The rate of error change (semi-log)

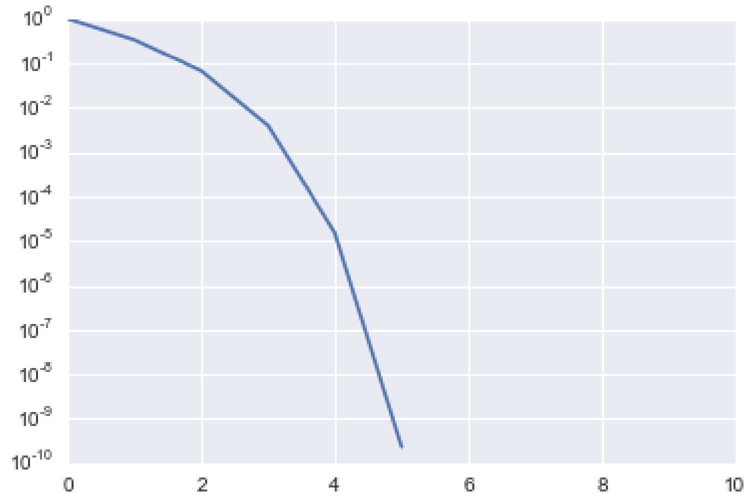


As we can see, it is convergent

The first iteration for the forth function



The rate of change



As we can see, it is convergent and it converge pretty fast.

Problem 3: (15 points)

Bioremediation involves the use of bacteria to consume toxic wastes. At steady state, the bacterial density x and nutrient concentration y satisfy the system of nonlinear equations

$$\begin{aligned} \gamma xy - x(1+y) &= 0 \\ -xy + (\delta - y)(1+y) &= 0 \end{aligned}$$

where γ and δ are parameters that depend on various physical features of the system; typical values are $\gamma = 5$ and $\delta = 1$. Implement Newton's method for solving a system of nonlinear equations, and use your implementation to solve this system numerically. You should find at least one solution with a nonzero bacterial density ($x \neq 0$), and one in which the bacterial population has died out ($x = 0$). Clearly indicate your initial guesses and the solutions to which they converged in your writeup.

Plug in γ and δ, we get

$$\begin{aligned} 5xy - x(1+y) &= 0 \\ -xy + (1 - y)(1+y) &= 0 \end{aligned}$$

The Jacobean then becomes:

$$J = \begin{bmatrix} 4y-1 & 4x \\ -y & -x-2y \end{bmatrix}$$

Thus, the steepest-descent newton's method is coded as follow

```
import numpy.linalg as la

def F(xs):
    x,y = xs
    return np.array([5*x*y-x*(1+y), -x*y+(1-y)*(1+y)])

def J(xs):
    x,y = xs
    return np.array([[4*y-1, 4*x], [-y, -x-2*y]])

def newtons(F, J, xs):
    x = []
    y = []
    s = np.array([1,1])
    while la.norm(s)!=0:
        s = la.solve(J(xs), -F(xs))
        x.append(xs[0])
        y.append(xs[1])
```

```
            xs = xs+s
        x.append(xs[0])
        y.append(xs[1])
        plt.plot(x,y)
        return xs

i = 0
print newtons(F, J, np.array([3, 2]))
print newtons(F, J, np.array([2, -3]))
print newtons(F, J, np.array([3, 0]))
```

We get the following points:

> [0. 1.]
> [0. -1.]
> [3.75 0.25]

As the solution

Problem 4: Optimization Theory (20 points)

(a) Determine the critical points of the function, $f(x) = 2x^3 - 25x^2 - 12x + 15$, and characterize each as a minimum, maximum, or inflection point. Also, determine whether the function has a global minimum or maximum on $\mathbb{R}$.

$$\begin{aligned}\frac{df}{dx} &= 6x^2 - 50x - 12 \\ \frac{d^2f}{dx^2} &= 12x - 50\end{aligned}$$

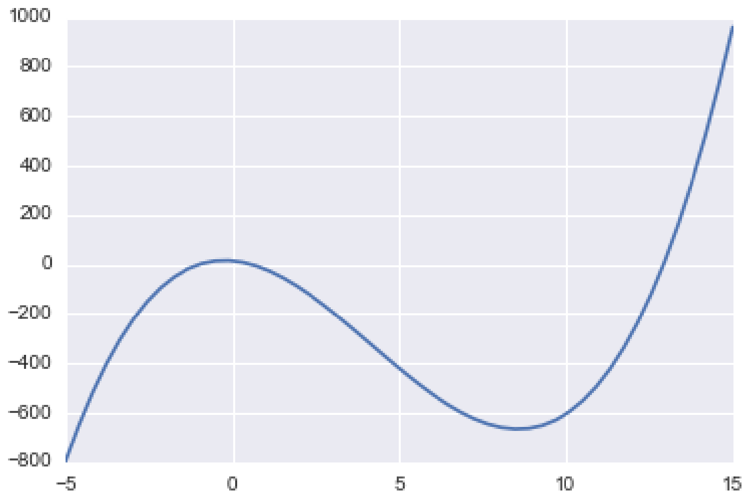
For $\dot{f} = 0, x = \frac{1}{6}(25 \pm \sqrt{697})$,
For $x_1 = \frac{1}{6}(25 + \sqrt{697})$, $\ddot{f} > 0$, and $x_2 = \frac{1}{6}(25 - \sqrt{697})$ $\ddot{f} < 0$

Thus x_1 is a local maximum and x_2 is a local minimum. And it does not have a global minimum or a maximum.

We can also see then from the graph

```
%matplotlib inline
x = np.linspace(-5, 15)
f = lambda x: 2*x**3-25*x**2-12*x+15
```

```
plt.plot(x, f(x))
plt.show()
```



(b) Determine the critical points of the function, $f(x) = 2x^3 - 3x^2 - 6xy(x-y-1)$, and characterize each as a minimum, maximum, or inflection point. Also, determine whether the function has a global minimum or maximum on $\mathbb{R}$.

$$\begin{aligned}
\frac{\partial f}{\partial x} &= 6x^2 + 6y^2 - 12xy - 6x + 6y \\
\frac{\partial f}{\partial y} &= -6x^2 + 12xy + 6x \\
\frac{\partial^2 f}{\partial x^2} &= 12x - 12y - 6 \\
\frac{\partial^2 f}{\partial x \partial y} &= -12x - 12y - 6 \\
\frac{\partial^2 f}{\partial y^2} &= 12x
\end{aligned}$$

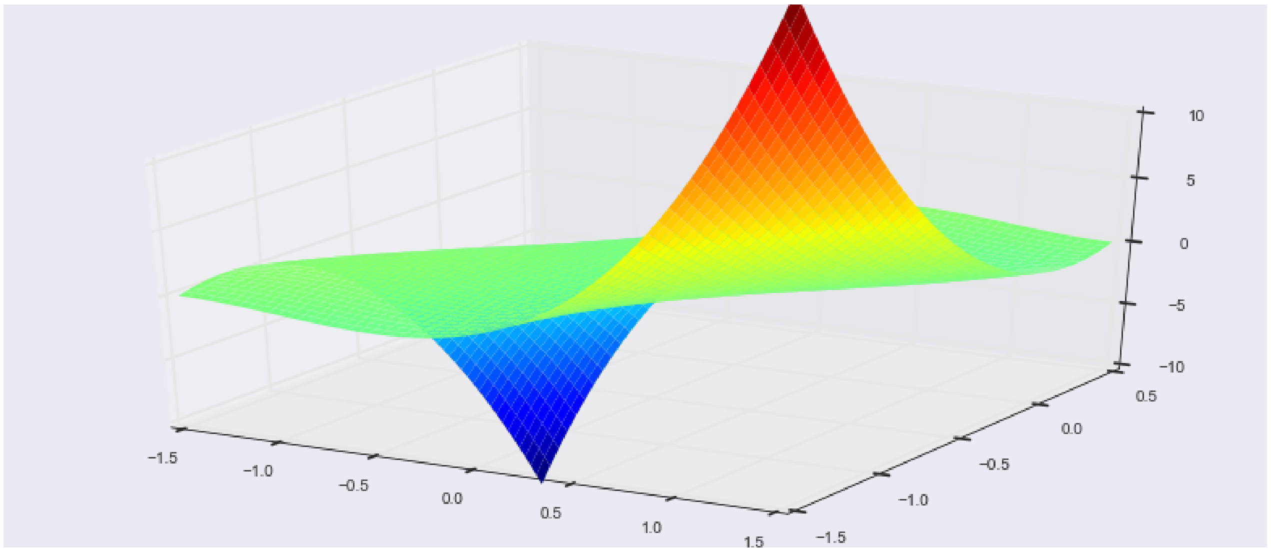
Thus, solve $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = 0$ we get

$$\begin{aligned}
x = -1 \ y = -1 \ &\textbf{local maximum} \\
x = 0 \ y = -1 \ &\textbf{saddle point} \\
x = 0 \quad y = 0 \ &\textbf{saddle point} \\
x = 1 \quad y = 0 \ &\textbf{local minimum}
\end{aligned}$$

From the picture it is pretty clear that the function dose not have global minimum or maximum. However, it is not very easy to see from the graph for saddle points or local extremum.

```
from mpl_toolkits.mplot3d.axes3d import Axes3D
import mpld3
```

```
mpld3.enable_notebook()

fig = plt.figure(figsize=(14,6))

# `ax` is a 3D-aware axis instance because of the projection='3d'
keyword argument to add_subplot
ax = plt.axes(projection='3d')

f = lambda x,y: 2*x**3-3*x**2-6*x*y*(x-y-1)
x = np.outer(np.linspace(1.5, -1.5, 60), np.ones(60))
y = np.outer(np.ones(60), np.linspace(0.5, -1.5, 60))
ax.plot_surface(x,y, f(x,y), cmap=plt.cm.jet, rstride=1, cstride=1,
linewidth=0)
ax.set_zlim3d(-10, 10);
```



And there is no global maximum and global minimum.

(c) Show that any local minimum of a convex function f on a convex set $S \subseteq \mathbb{R}^n$ is a global minimum of f on S. (Hint: If a local minimum x is not a global minimum, then let y be a point in S such that $f(y) < f(x)$ and consider the line segment between x and y to obtain a contradiction.)

Suppose there is a y s.t. $f(y) < f(x)$, thus, all the points on the line segment $g(t)$shall be

$$g(t) = yt + x(1-t)\ where\ t \in [0,1]$$

So, since x is a local point, there always exists a t_0 small enough, $g(t_0)$ close enough to x where $f(g(t_0)) > f(x)$ since x is a local minimum.
However, since function is a convex,

$$f(g(t_0)) = f(yt_0 + x(1-t_0)) \leq t_0 f(y) + f(x) - t_0 f(x) = t_0(f(y) - f(x)) + f(x) < f(x)$$

However, we know that $f(g(t_0)) < f(x)$, it leads to a contradiction. Thus, there is no y exists.

Problem 5: (20 points)

Write a program to find a minimum of Rosenbrock's function

$$f(x,y) = 100(y-x^2)^2 + (1-x)^2$$

using each of the following methods:
(a) Steepest descent
(b) Newton

You should try each of the methods from each of the three starting points $[-1,1]^T$, $[0,1]^T$, $[2,1]^T$. Provide the returned solution (x and y), its residual, and the number of iterations for each method and starting guess. Comment on whether the returned solution is a minimum.

$$\nabla f = (400x^3 + 2x - 400xy - 2)\hat{x} + (-200x^2 + 200y)\hat{y}$$

$$H = \begin{bmatrix} 1200x^2 - 400y + 2 & -400x \\ -400x & 200 \end{bmatrix}$$

And by hand, the optimised solution shall be $[1,1]^T$

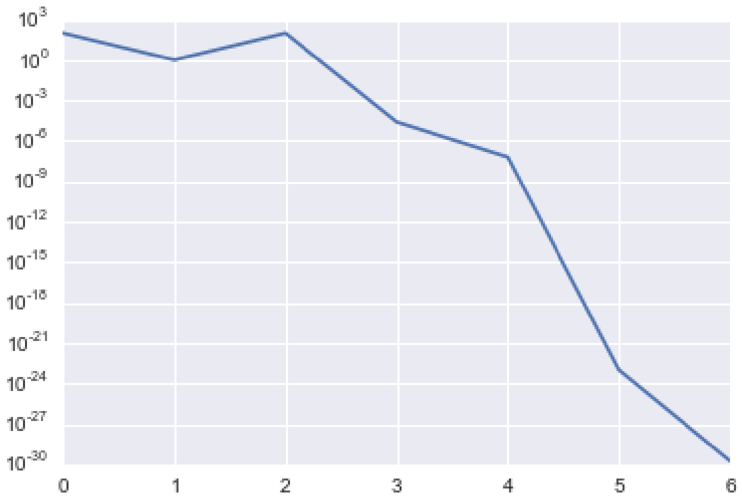
Thus, the methods are like below

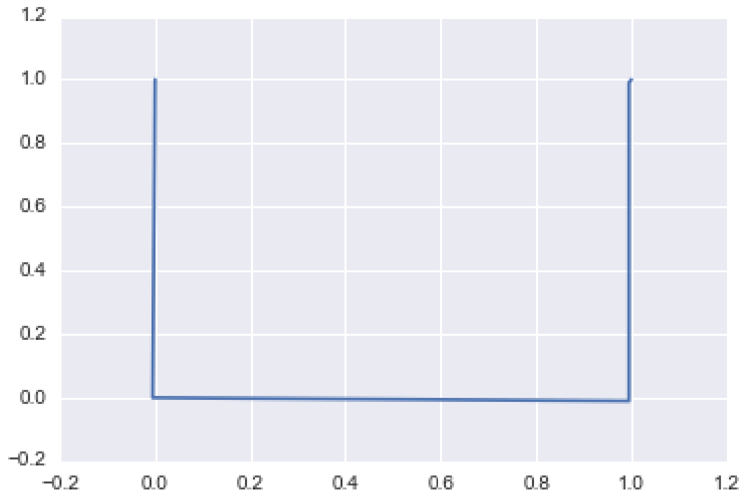
```
def f(xs):
    x, y = xs
    return 100*(y-x**2)**2+(1-x)**2

def gradf(xs):
    x,y = xs
    return np.array([400*x**3+2*x-400*x*y-2, -200*x**2+200*y])

def H(xs):
    x,y = xs
    return np.array([[1200*x**2+2-400*y, -400*x],[-400*x, 200]])

def R(xs):
    return f(xs)-f(np.array([1,1]))
```

```
def Newton(f, gradf, H, xs):
    s = la.solve(H(xs), gradf(xs))
    x = [xs[0]]
    y = [xs[1]]
    r = [R(xs)]
#     while la.norm(s)!=0:
    while R(xs)>1e-26:
        xs -= s
        s = la.solve(H(xs), gradf(xs))
        x.append(xs[0])
        y.append(xs[1])
        r.append(R(xs))
    plt.semilogy(range(0, len(r)), r)
    plt.show()
    plt.plot(x,y)
    plt.show()
    return xs, len(r), r[-1]

print(Newton(f, gradf, H, np.array([0.0,1.0])))
print(Newton(f, gradf, H, np.array([-1.0,1.0])))
print(Newton(f, gradf, H, np.array([2.0,1.0])))

def steepestdescent(f, gradf, xs):
    x = [xs[0]]
    y = [xs[1]]
    r = [R(xs)]
#     for i in range(200):
    while R(xs)>1e-26:
        a = sp.optimize.minimize_scalar(lambda a: f(xs-
a*gradf(xs))).x
        xs = xs-a*0.9*gradf(xs)
        x.append(xs[0])
        y.append(xs[1])
        r.append(R(xs))
    plt.semilogy(range(1, len(r)+1), r)
    plt.show()
    plt.plot(x,y)
    plt.show()
    return xs, len(r), r[-1]

print(steepestdescent(f, gradf, np.array([0.0,1.0])))
print(steepestdescent(f, gradf, np.array([-1.0,1.0])))
print(steepestdescent(f, gradf, np.array([2.0,1.0])))
```

And the results are as follows,
For newton’s method from init point [0, 1], it goes through 6 iteration and reaches the optimised point.
The first graph is it’s error and the path.

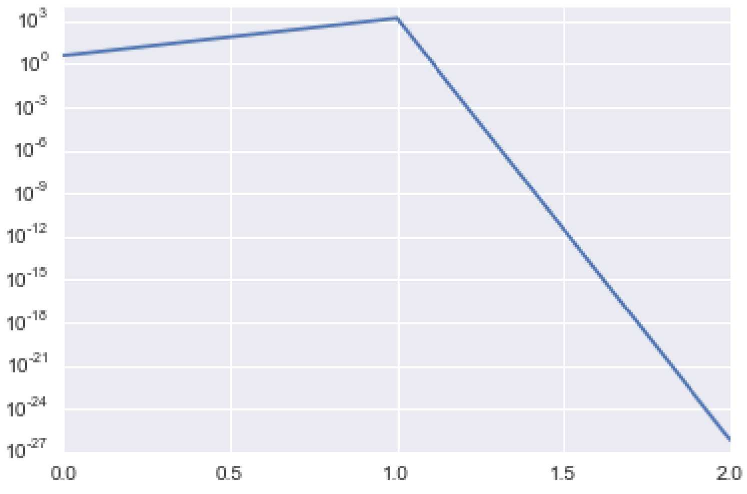
```
(array([ 1., 1.]), 7, 1.7749370367472766e-30)
```

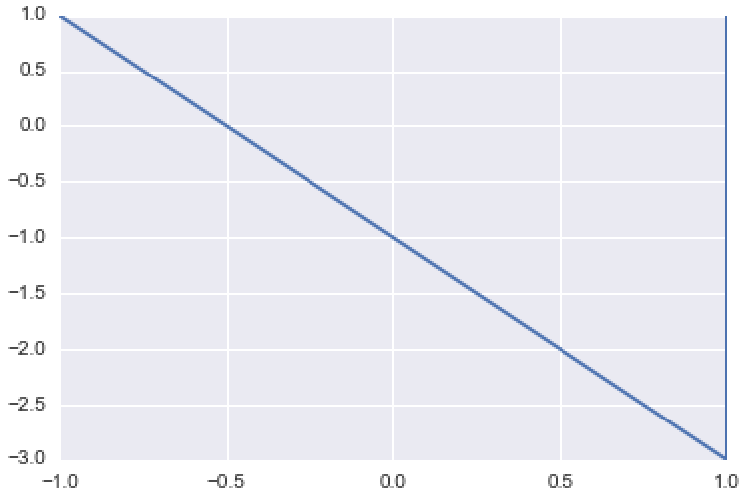




For newton’s method from init point [-1, 1], it goes through 2 iteration and reaches the optimised point.

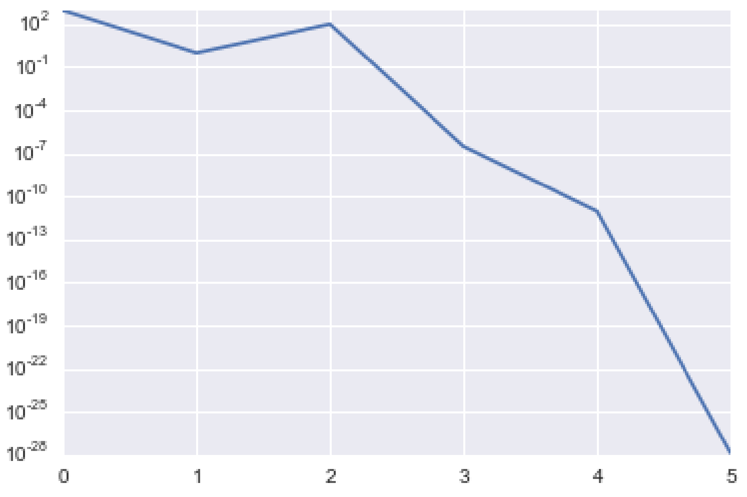
```
(array([ 1., 1.]), 3, 6.2332337464104011e-27)
```

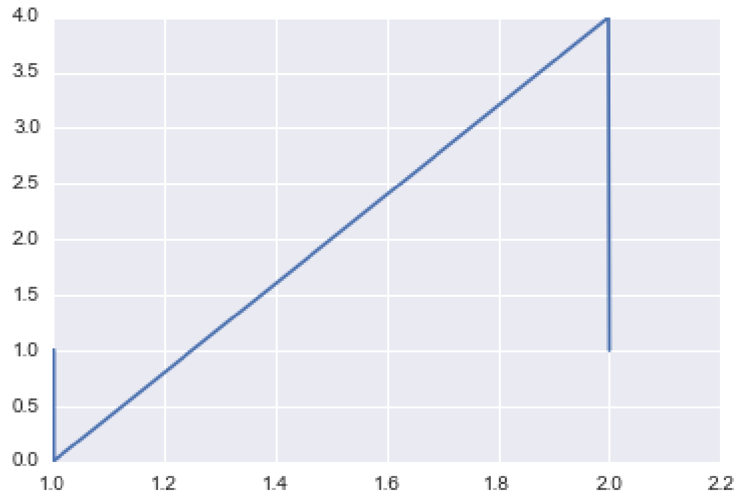




For newton's method from init point [2, 1], it goes through 5 iteration and reaches the optimised point.

> (array([1., 1.]), 6, 1.3316958156262206e-28)

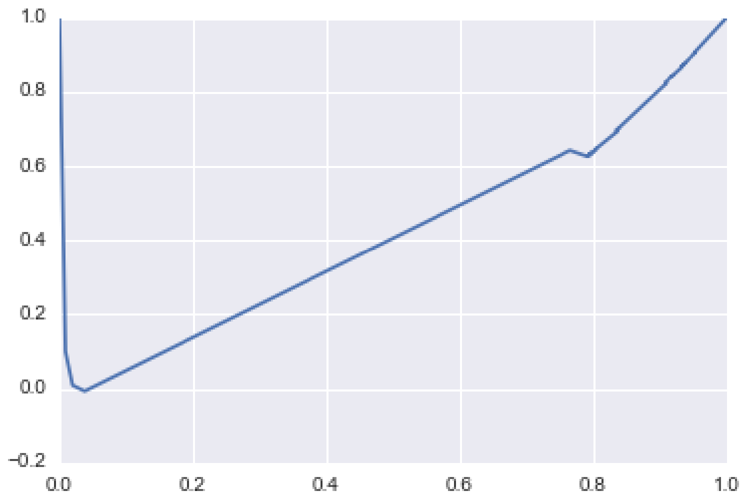


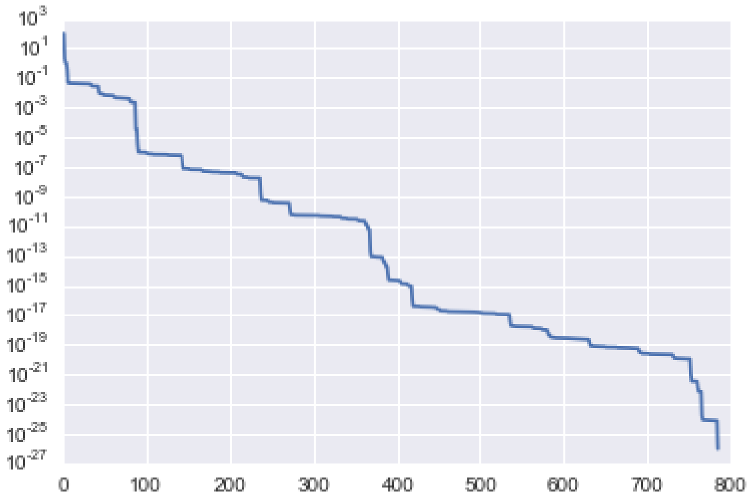


As we can see that the steepest-descent take much more steps, and as professor suggested, damping how far each steps go might sometimes mean a faster route. (0.9 for this case)
Thus, we get solution as follow

For steepest-descent method from init point [0, 1], it goes through 785 iteration and reaches the optimised point.
The first graph is it’s path and the error.

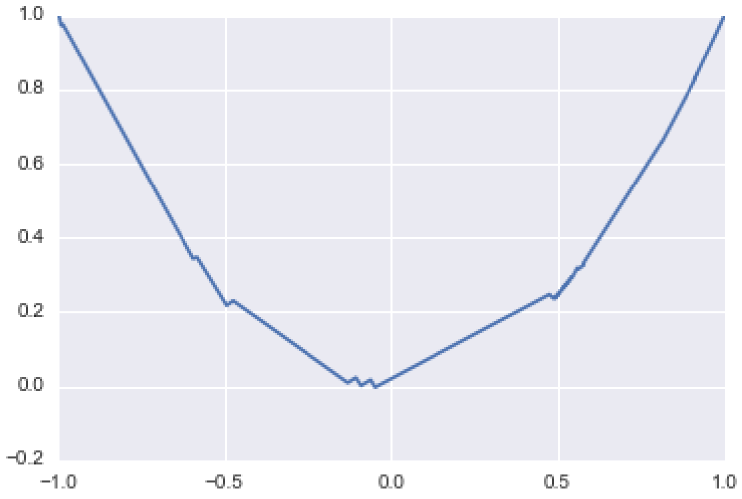
> (array([1., 1.]), 785, 9.2715808266757044e-27)

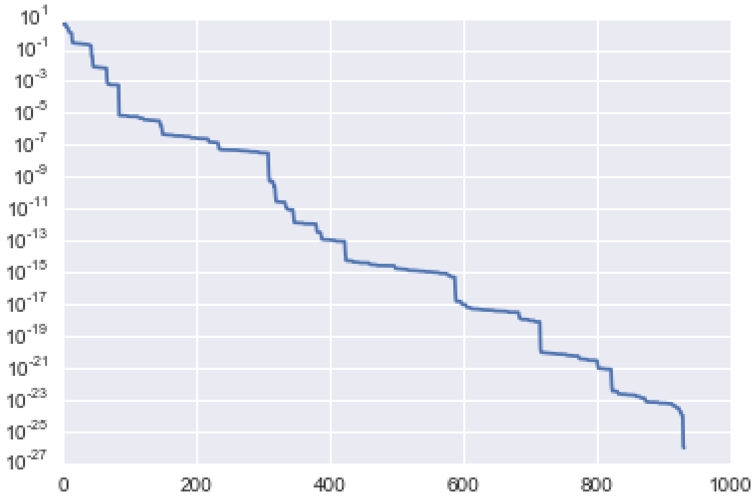




For steepest-descent method from init point [-1, 1], it goes through 930 iteration and reaches the optimised point.
The first graph is it’s path and the error.

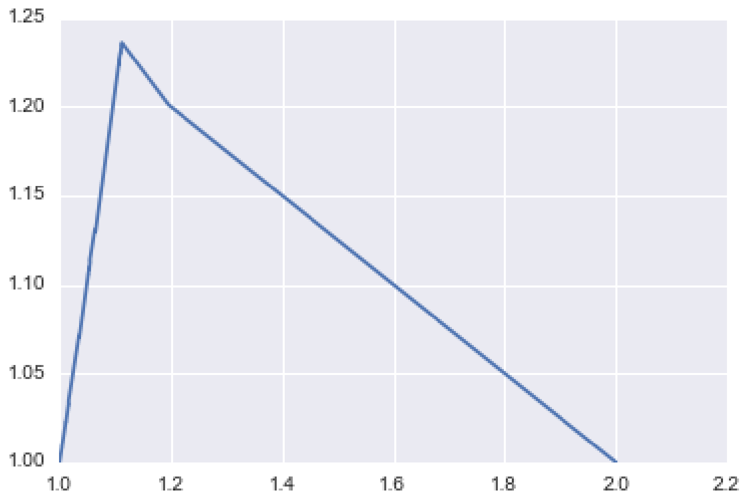
> (array([1., 1.]), 930, 9.2715808266757044e-27)

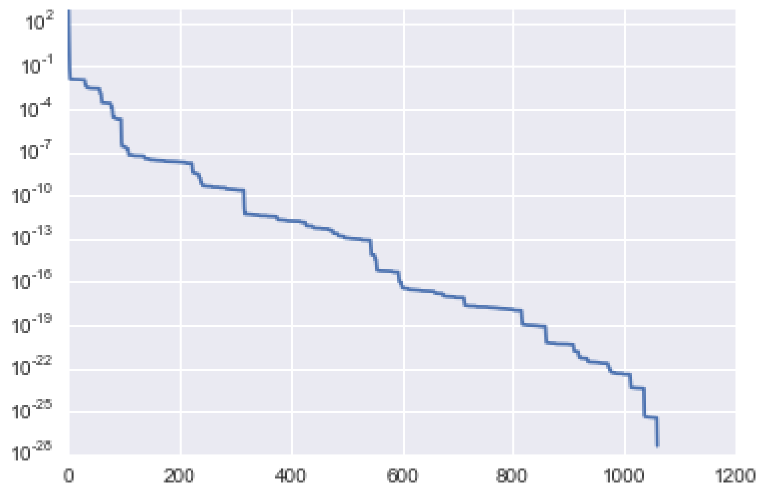




For steepest-descent method from init point [2, 1], it goes through 1059 iteration and reaches the optimised point.
The first graph is it’s path and the error.

> (array([1., 1.]), 1059, 3.7589222133781213e-28)





Problem 6: (15 points)

A bacterial population P grows according to the geometric progression

$$P_k = rP_{k-1}$$

where r is the growth rate. The following population counts (in the billions) are observed:

k	1	2	3	4	5	6	7	8
P_k	0.91	0.36	0.69	1.3	2.5	4.7	8.5	14

(a) Perform a nonlinear least squares fit of the growth function to these data to estimate the initial population P_0 and the growth rate r.

(b) By taking the logarithm of the model function, a fit to these data can also be done by linear least squares. Perform such a linear least squares fit to obtain estimates for P0 and r, and compare your results with those from your nonlinear fit

Solve the difference equation, we can get a model:

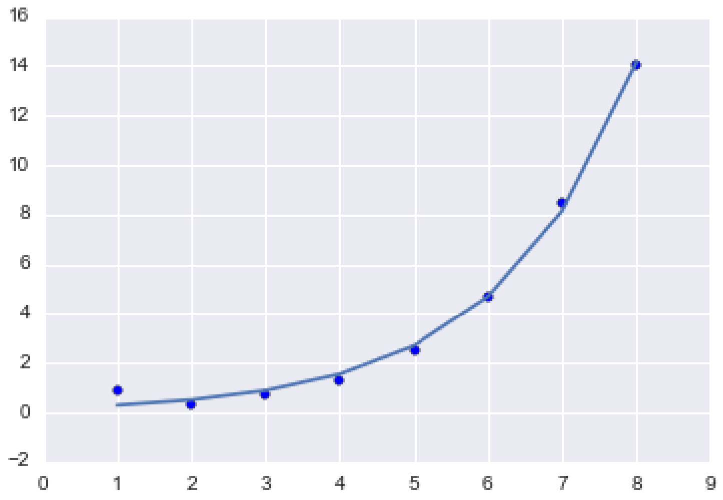
$$P(k) = P_0 e^{kr}$$

,

If we tried to fit the data, we get

```
k = np.array([1,2,3,4,5,6,7,8], dtype=np.float)
P = np.array([0.91,0.36,0.69,1.3,2.5,4.7,8.5,14])

def F(t, p0, r):
    return p0*np.exp(t*r)

def R(t, y, p0, r):
    return y-F(t, p0, r)

def J(t ,p0, r):
    return np.array([-np.exp(r*t), -p0*t*np.exp(r*t)]).T

def GuassNewton(J, F, R, x):
    p0, r = x
    for i in range(50):
        s, _, _, _ = la.lstsq(J(k, p0, r),-R(k, P, p0, r))
        small_norm = la.norm(R(k, P, p0, r))
        p0, r = x = x+s
    return p0, r
p0, r = GuassNewton(J, F, R, np.array([0, 0]))
plt.scatter(k, P)
plt.plot(k, F(k, p0, r))
plt.show()
```

result:

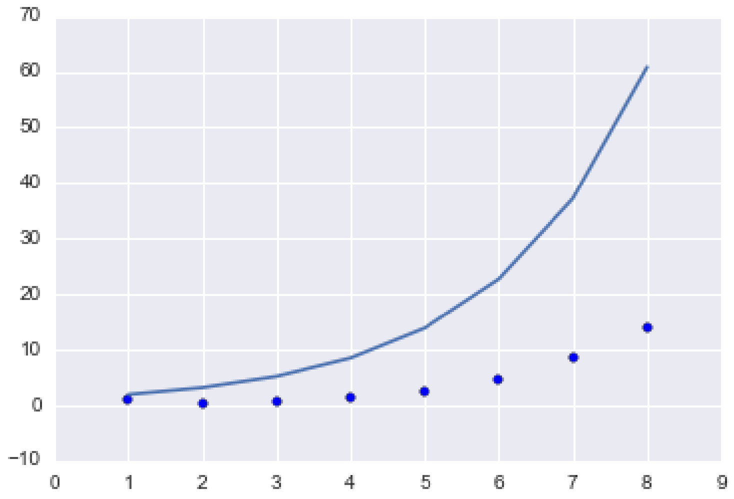


If we take lg on both sides,
we have new equation:

$$\log(P_0) + rk = \log(P(k))$$

And here is the code

```
(log_p0, r), _, _, _ = la.lstsq(np.array([np.ones(len(k)), k]).T,
np.log(P))
plt.scatter(k, P)
plt.plot(k, F(k, np.exp(p0), r))
plt.show()
```



Here is the result I got. As we can clearly see, the non-linear fitting look much better than linear fitting. Since the non-linear one does not distort the graphics.