Numerical Analysis Assignment #2

Xinglu Wang   Student Number: 3140102282
College of Information Science & Electronic Engineering

Problem 1

According to Newton's method, $g(x) = x - \frac{f(x)}{f'(x)}$. We get

$$g(x) = x + \frac{cosx + x^3}{sinx - 3x^2} \tag{1}$$

Using $g(x) = x$ to iterate, We get iteration point shown below:

Iteration Point
-1.000000000000000
-0.880332899571582
-0.865684163176082

Table 1.1 Data for problem 1

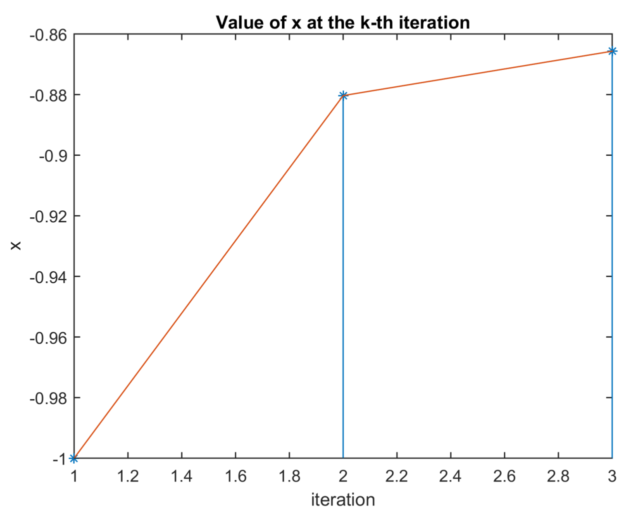

Figure 1.1 Plot point for problem 1

Therefore, $p_2 = -0.8656684163176082$
And p_0 should not be 0, otherwise $f'(p_0)$ would be 0, and we cannot get an appropriate p_1.
The usage of Implement of Newton's method is shown below:

```
%% problem 1
format long;
clear
f1=@(x) -x^3-cos(x);
Newton(f1,-1,10^-5,2);
```

The Implement of Newton's method is shown below:

```
function res=Newton(func,IniGuess,TOL,MaxIter)
if nargin==3
    MaxIter=1000;
elseif nargin==2
    TOL=10^-2;
    MaxIter=1000;
elseif nargin<2 || nargin>4
    disp('Check You Input!');
end
%%
res=IniGuess;sol=res;
syms f gsym symx
f=func(symx);
gsym=symx-f/diff(f);
g=matlabFunction(gsym);
%%
k=1;
while(abs(g(res)-res)>TOL && k<MaxIter)
    res=g(res);
    sol(end+1)=res;
    k=k+1;
end
close all;
sol(end+1)=g(res);
disp('The Iteration Point of X is ' );
disp(sol');
figure('color',[1,1,1]); box on ; hold on;
stem(sol,'Marker','*','BaseValue',min(sol));
plot(sol);
xlabel('iteration');
ylabel('x');
title('Value of x at the k-th iteration');
%export_fig q1_1.png -m3
```

Refer to code Problem1.m. When p0 = -1, we get p2 = -0.865684163176082.

Problem 2

i). According to Newton-Raphson method, $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$,where $k \in \mathbb{N}$. We infer that

$$x_{k+1} = 2x_k - b{x_k}^2 \tag{2}$$

Since $\varepsilon_k = \frac{\frac{1}{b} - x_k}{\frac{1}{b}}$, we infer that

$$|\varepsilon_{k+1}| = \frac{\frac{1}{b} - x_{k+1}}{\frac{1}{b}} = (1 - bx_k)^2 = {\varepsilon_k}^2 \tag{3}$$

ii). When $0 < x_0 < \frac{2}{b}$, $\varepsilon_0 = \frac{\frac{1}{b} - x_0}{\frac{1}{b}} < 1$, according to **i).**, $\{\varepsilon_k\}$ is a geometric progression series whose common ratio is below 1. So the relative error $\varepsilon_k \to 0$, and $x \to \frac{1}{b}$

Problem 3

a).

$$
\begin{aligned}
3x_1 - cos(x_1x_2) - \frac{1}{2} &= 0 \\
4{x_1}^2 - 625{x_2}^2 + 2x_2 - 1 &= 0 \\
\mathrm{e}^{-x_1x_2} + 20x_3 + \frac{10\pi - 3}{3} &= 0
\end{aligned}
$$

After running the code,we get x of each iteration.

initial	first iter	second iter
0	0.500000000000000	0.500166686911463
0	0.500000000000000	0.250803638439239
0	-0.523598775598299	-0.517387427392491

Thus the answer x_2 is

$$x_2 = \begin{Bmatrix} 0.500166686911463 \\ 0.250803638439239 \\ -0.517387427392491 \end{Bmatrix}$$

The usage of Implement of Newton's method for nonlinear equation system is shown below:

```
%% problem 3
clear
format long;
syms f1 f2;
x=sym('x',[3,1]);
f1=[3*x(1)-cos(x(2)*x(3))-1/2;...
    4*x(1)^2-625*x(2)^2+2*x(2)-1;...
    exp(-x(1)*x(2))+20*x(3)+(10*pi-3)/3];
%MetrixPlot(f1(1));
f2=[x(1)^2+x(2)-37;...
    x(1)-x(2)^2-5;...
    x(1)+x(2)+x(3)-3];
NewtonNonlin(f1,x,[0,0,0].',2)
NewtonNonlin(f2,x,[0,0,0].',2)
```

Implement of Newton's method for nonlinear equation system is shown below:

```
function res=NewtonNonlin(symfunc,symx,inix,maxIter)

if ¬isa(symfunc,'sym') || ¬isa(symx,'sym')
    disp(['check Input!',class(symfunc),class(symx)]);
    error('ClassError.');
end
if size(symx,1)≠size(inix,1)
   error('Dimension miscount.')
end
symJac=jacobian(symfunc,symx);
symJac=symJac+symx(3)*10^-100; %for problem_3_a, make the matrix nonsingular. Not the best ...
    way, but the easiest.
func=matlabFunction(symfunc);
Jac=matlabFunction(symJac);
res=zeros(3,1);
res(:,1)=inix;
for iter=1:maxIter
```

```
    XLnum=res(:,end);
    XL=num2cell(XLnum); %for matrix input, we can also use cell2mat(arrayfun(f, b(:, 1), b(:, ...
        2), 'UniformOutput', 0)')'
    res(:,end+1)=XLnum-inv(Jac(XL{:}))*func(XL{:});
end
```

I still have many thing to optimize, this implement is not robust and can not be applied into common use. There are many skill to deal with matrix in math, but I am not experienced enough.

b). Similarly, we get x for each iteration shown below

initial	first iter	second iter
0	5.000000000000000	4.350877192982456
0	37.000000000000000	18.491228070175438
0	-39.000000000000000	-19.842105263157897

Thus the answer is

$$x_2 = \left\{ \begin{array}{c} 4.350877192982456 \\ 18.491228070175438 \\ -19.842105263157897 \end{array} \right\}$$

Problem 4

a). Although Steepest Descent method is not as strict with initial value as Newton method for nonlinear system, it still need a estimation for initial value. (It is apparent by thinking the gradient decides where this points will go in next iteration.) So I use the matlab function **solve** to find symbolic answer for this problem, then add initial estimation for my programme. The way for prediction is shown below

```
%%for estimating and predicting
[x1,x2,x3]=solve([f1==0,f2==0,f3==0]);
for ii=1:size(x1,1)
    res=[];
    for jj=1:3
        tmp=eval(['x',num2str(jj)]);
        tmp2=tmp(ii);
        res=[res;tmp2];
    end
    disp(double(res));
end
```

The result is show below,

solution 1	-8.441429707360641
	-7.940157258463179
	-19.143837080321028
solution 2	1.036400470329211
	1.085706550741678
	0.931191442315390
solution 3	$8.708234096655289 + 6.936510819683741i$
	$-0.793852371079888 - 9.117849140990071i$
	$8.779635019761287 + 29.631028653675870i$
solution 4	$0.619280521860425 + 8.523979282877251i$
	$10.471077724940638 + 4.493153163724587i$
	$21.436062799241533 + 55.489000314501233i$
solution 5	$8.708234096655289 - 6.936510819683741i$
	$-0.793852371079888 + 9.117849140990071i$
	$8.779635019761287 - 29.631028653675870i$
solution 6	$0.619280521860425 - 8.523979282877251i$
	$10.471077724940638 - 4.493153163724587i$
	$21.436062799241533 - 55.489000314501233i$

From the result we can know that every method has its privilege and inferiority. Steepest method still depends on initial value, We can conclude that

Advantage	No matter what initial value we give it, as long as gradient is not 0, it will converge.
Disadvantage	It determines a local minimum for a multivariable function.
	It converges only linearly to the solution
	Unlike symbolic method, it cannot give complex answer such as $a + bi$
	If we choose a bad alpha, it will become quite slow.

And my answer is shown below. Compared with the answer calulated by matlab, my answer is quite accurate in the terms of $TOL = 0.05$. When I choose $TOL = 10^{-5}$ and find that if TOL become smaller, then size of step should also be smaller.To sum up, size of step ,max iteration times, tolerance are closely related. I have not tried the self-adaptation method for calculate α. If it is applied,I think, it can calculate more fast but no more accurate, since I have chosen a small step size.

solution $x_1 \in \mathbb{R}$	1.036498009046320
	1.085380893216069
	0.931139784999289
solution $x_2 \in \mathbb{R}$	-8.440255795243777
	-7.939370693306589
	-19.137864730324921

The usage of Steepest Descent method is shown below,

```
% problem 4
clear
format long
syms x1 x2 x3 f1 f2 f3 g1 g2;
f1=15*x1+x2^2-4*x3-13;
f2=x1^2+10*x2-x3-11;
f3=x2^3-25*x3+22;
g1=f1^2+f2^2+f3^2;
x=[x1;x2;x3];
tol=10^-5;
MaxIter=10000;

```

```
ini=[1;1;1];
res=SDescent(g1,x,ini,tol,MaxIter);
disp(res(:,end));

ini=[-8,-8,-19];
res=SDescent(g1,x,ini,tol,MaxIter);
disp(res(:,end));

f1=10*x1-2*x2^2+x2-2*x3-5;
f2=8*x2^2+4*x3^2-9;
f3=8*x2*x3+4;
g2=f1^2+f2^2+f3^2;

ini=[1;0;1];
res=SDescent(g2,x,ini,tol,MaxIter);
disp(res(:,end));

ini=[0,1,-1];
res=SDescent(g2,x,ini,tol,MaxIter);
disp(res(:,end));

ini=[1,-1,0];
res=SDescent(g2,x,ini,tol,MaxIter);
disp(res(:,end));

ini=[0,0,-1];
res=SDescent(g2,x,ini,tol,MaxIter);
disp(res(:,end));
```

The implement of Steepest Descent method is shown below, and we will see there are munch space to improve our implement to make the programme more robust.

```
function res=SDescent(symfunc,symx,ini,TOL,MaxIter,alpha)
if nargin==5
    alpha=10^-5;
elseif argin >6 || size(symx,2)≠1
        error('Check Input');
end

grad=jacobian(symfunc,symx);
grad=grad.';%!!! associate !!!

func=matlabFunction(symfunc);
grad=matlabFunction(grad);
res(:,1)=ini;

for iter=1:MaxIter
    XLnum=res(:,end);
    XL=num2cell(XLnum);
    res(:,end+1)=XLnum-alpha*grad(XL{:});
    if abs(func(XL{:}))<TOL
        break;
    end
% % A different measure way;
%     if norm(res(:,end)-res(:,end-1))<tol
%         break;
%     end
end
%disp(iter);
```

b). Similarly, via estimating first and applying Steepest Descent method, We can get

$x_1 \in \mathbb{R}$	0.843203979094425
	-0.353845516901669
	1.413882789399937
$x_2 \in \mathbb{R}$	0.497133950969613
	0.997545468636543
	-0.505700092951828
$x_3 \in \mathbb{R}$	0.900005537385528
	-1.000190835623339
	0.499540078284082
$x_4 \in \mathbb{R}$	0.206590365596643
	0.353736566702580
	-1.414033729757102