交大資工
110550062 朱致伶

Numerical Method: Assignment 1

1. (30%) The function f(x) = $x^2 + \sin(x) - \frac{e^x}{4} - 1$ has zeros for two values near x = 0. Please compute both roots, starting with [−2, 0] and [0, 2], to attain an accuracy of 10^{-5} using the following methods: (a) the bisection method (b) the secant method, and (c) Newton's method.

Ans: Bisection method: 0.911896 and -1.431824 / Secant method: 0.911917 and -1.431807 / Newton's method: 0.911919 and -1.431809

(a) Bisection Method - m = (upper_bound + lower_bound) / 2

Root in [0, 2]

```
(N=1) a = 0.000000, b = 2.000000, root = 1.000000
(N=2) a = 0.000000, b = 1.000000, root = 1.000000
(N=3) a = 0.500000, b = 1.000000, root = 0.500000
(N=4) a = 0.750000, b = 1.000000, root = 0.750000
(N=5) a = 0.875000, b = 1.000000, root = 0.875000
(N=6) a = 0.875000, b = 0.937500, root = 0.937500
(N=7) a = 0.906250, b = 0.937500, root = 0.906250
(N=8) a = 0.906250, b = 0.921875, root = 0.921875
(N=9) a = 0.906250, b = 0.914062, root = 0.914062
(N=10) a = 0.910156, b = 0.914062, root = 0.910156
(N=11) a = 0.910156, b = 0.912109, root = 0.912109
(N=12) a = 0.911133, b = 0.912109, root = 0.911133
(N=13) a = 0.911621, b = 0.912109, root = 0.911621
(N=14) a = 0.911865, b = 0.912109, root = 0.911865
(N=15) a = 0.911865, b = 0.911987, root = 0.911987
(N=16) a = 0.911865, b = 0.911926, root = 0.911926
(N=17) a = 0.911896, b = 0.911926, root = 0.911896
```

Root in [-2, 0]

```
(N=1) a = -2.000000, b = 0.000000, root = -1.000000
(N=2) a = -2.000000, b = -1.000000, root = -1.000000
(N=3) a = -1.500000, b = -1.000000, root = -1.500000
(N=4) a = -1.500000, b = -1.250000, root = -1.250000
(N=5) a = -1.500000, b = -1.375000, root = -1.375000
(N=6) a = -1.437500, b = -1.375000, root = -1.437500
(N=7) a = -1.437500, b = -1.406250, root = -1.406250
(N=8) a = -1.437500, b = -1.421875, root = -1.421875
(N=9) a = -1.437500, b = -1.429688, root = -1.429688
(N=10) a = -1.433594, b = -1.429688, root = -1.433594
(N=11) a = -1.433594, b = -1.431641, root = -1.431641
(N=12) a = -1.432617, b = -1.431641, root = -1.432617
(N=13) a = -1.432129, b = -1.431641, root = -1.432129
(N=14) a = -1.431885, b = -1.431641, root = -1.431885
(N=15) a = -1.431885, b = -1.431763, root = -1.431763
(N=16) a = -1.431824, b = -1.431763, root = -1.431824
```

(b) Secant Method – x2 = x1– f(x1)*(x0-x1)/[f(x0)-f(x1)]

Root in [0, 2]

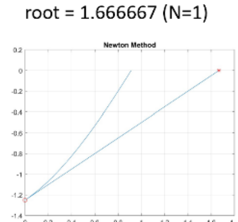
```
(N=1)  root = 0.754823
(N=2)  root = 0.902232
(N=3)  root = 0.911491
(N=4)  root = 0.911899
(N=5)  root = 0.911917
```

Root in [-2, 0]

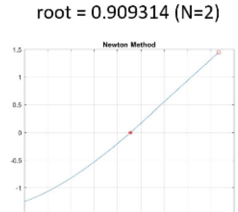
```
(N=1)  root = -0.756002    (N=6)  root = -1.431142
(N=2)  root = -1.221968    (N=7)  root = -1.431655
(N=3)  root = -1.379027    (N=8)  root = -1.431773
(N=4)  root = -1.419369    (N=9)  root = -1.431800
(N=5)  root = -1.428924    (N=10)  root = -1.431807
```

(c) Newton's Method – x1 = x0 – f(x0)/df(x0)

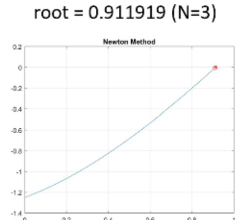
root = 1.666667 (N=1)



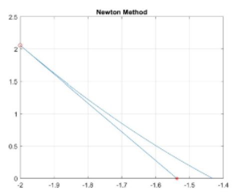
root = 0.909314 (N=2)



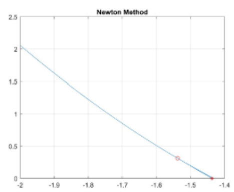
root = 0.911919 (N=3)



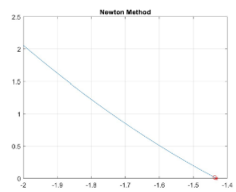
root = -1.537780 (N=1)



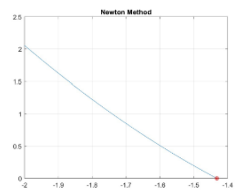
root = -1.437143 (N=2)



root = -1.431823 (N=3)



root = -1.431809 (N=4)



交大資工
110550062 朱致伶

Previous result shows that

- **Bisection method has two roots, 0.911896 and -1.431824**
  Divide the bracket length into half each time, check whether the root is in the left or right side of the bracket, and renew the upper or lower bound with the middle value.
- **Secant method has two roots, 0.911917 and -1.431807**
  Update the intersection as the bisection does, instead of taking the middle point as the new value, extract x2 from the point intersected by x-axis and secant line**.**
- **Newton's method has two roots, 0.911919 and -1.431809**
  Approximate the root by using the first derivative, extract the point that tangent line intersect with x-axis.

Bisection Method

main.m | bisection.m | newton.m | secant.m

```matlab
function [x1, xa, xb, N] = bisection(f, a, b, tol, save)
N = 1;
X(1) = (a+b)/2;
A(1) = a;
B(1) = b;
while abs(f((a+b)/2)) > tol
    N = N+1;
    m = (a+b)/2;
    if (f(a)*f(m)<0)
        b = m;
    else
        a = m;
    end
    if save==1
        X(N) = m;
        A(N) = a;
        B(N) = b;
    end
end

if save == 1
    x1 = X;
    xa = A;
    xb = B;
end
```

Secant Method

main.m | bisection.m | newton.m | secant.m

```matlab
function [xn, N] = secant(f, x0, x1, tol, save)
N = 2;
X(1) = x0;
X(2) = x1;
x2 = x1;
while abs(f(x2)) > tol
    N = N+1;
    x2 = x1 - f(x1)*(x0-x1)/(f(x0)-f(x1));
    if (f(x0)*f(x2)<0)
        x1 = x2;
    else
        x0 = x2;
    end
    if save==1
        X(N) = x2;
    end
end

if save == 1
    xn = X;
end
```

Newton Method

main.m | bisection.m | newton.m | secant.m

```matlab
function [x1, N] = newton(f, df, x0, tol, save)
N = 1;
X(1) = x0;
while abs(f(x0)) > tol
    N = N+1;
    x1 = x0 - f(x0)/df(x0);
    if save == 1
        X(N) = x1;
    end
    x0 = x1;
end

if save == 1
    x1 = X;
end
```

交大資工
110550062 朱致伶

2. (25%) Use Newton's method on the polynomial $P(x) = (x-2)^3\,(x-4)^2$ with $x_0 = 3$. Does it converge? To which root? Is convergence quadratic?

<u>Ans: Converges, root x=2, the convergence is quadratic</u>

The polynomial $P(x) = (x-2)^3\,(x-4)^2$ with $x_0 = 3$ as starting value, converges at root x=2.

```
dg:0.000000 -> Convergence
root = 1.998900
(N=1)   error = g(2) - g(xn) = 0.500000 - 1.500000 = -1.000000
(N=2)   error = g(2) - g(xn) = 0.500000 - 0.632495 = -0.132495
(N=3)   error = g(2) - g(xn) = 0.500000 - 0.486453 = 0.013547
(N=4)   error = g(2) - g(xn) = 0.500000 - 0.499725 = 0.000275
```

By Newton's method, we get the value of root=1.998900 which is the approximate value of x=2.

Since P(x) has root x=2 with multiplicity k=3, we can factor out $(x-r)^k$ from $P(x)$ to get $P(x) = (x-r)^k * Q(x)$.

With slightly modified Newton's method, $x_{n+1} = x_n - k * \frac{p(x_n)}{p'(x_n)} = g_k(x_n)$. Although the Newton's method in fixed-point iteration form is modified, it still remains the properties:

- $g'_k(r) = 0$
- Newton's method still converges quadratically at a root=r of multiplicity k

Therefore, the Newton.m code is slightly modified as the formula above, $x_{n+1} = x_n - k * \frac{p(x_n)}{p'(x_n)}$ when calculating the new x.

To check whether the polynomial P(x) converges at root=1.998900, we must make sure that $|g'(x)| = |\frac{p(x)\,p''(x)}{[p'(x)]^2}| < 1$. With the code main.m below, we can find $p(x),\ p'(x),\ p''(x)$ at line 8, 9, 10, knowing the 3 formulas allows us to obtain the result | $g'(1.998900)$ | = 0.00000 < 1.

To check whether the convergence is quadratic, we combine the results of

- | $g'(r)$ | = 0 < 1
- $g(x_n) = g(r) + g'(r) * (r - x_n) + g''(\alpha) * (r - x_n)^2 / 2$

Therefore, $g'(r) * (r - x_n)$ can be eliminated. we get $g(x_n) = g(r) + g''(\alpha) * (r - x_n)^2 / 2$, which can be represent in the error formula.

Finally, $e_{n+1} = g(r) - g(x_n) = -g''(\alpha) * (r - x_n)^2 / 2$, in other words the convergence is quadratic.

交大資工
110550062 朱致伶

Main.m

main.m | Newton.m

```matlab
clear all;
clf;
save = 1;
x0 = 3;
k=3;
tol = 10^(-7);

p = inline('x.^5 - 14*x.^4 + 76*x.^3 - 200*x.^2 + 256*x - 128','x');
dp = inline('5*x.^4 - 56*x.^3 + 228*x.^2 - 400*x + 256', 'x');
ddp = inline('20*x.^3 - 168*x.^2 + 456*x - 400', 'x');
g = inline('x - 3*(x.^5 - 12*x.^4 + 76*x.^3 - 200*x.^2 + 256*x - 128)/(5*x.^4 - 48*x.^3 + 228*x.^2 - 400*x + 256)', 'x');

[Xn, N] = Newton(p, dp, x0, k, tol, save);
dg = p(Xn(N))*ddp(Xn(N))/dp(Xn(N))*dp(Xn(N));

if abs(dg)<1
    fprintf("dg:%f\t-> Convergence\n", dg);
else
    fprintf("dg:%f\t-> Divergence\n", dg);
end
fprintf("root = %f\n", Xn(N));

X = linspace(Xn(1),Xn(N),100);
Y = p(X);
for i=1:N-1
    clf;
    plot(X,Y);
    hold on;
    line([Xn(i), Xn(i+1)],[p(Xn(i)), 0]);
    plot(Xn(i), p(Xn(i)), 'ro');
    plot(Xn(i+1), 0, 'r*');
    grid;
    fprintf('(N=%i)\terror = g(2) - g(xn) = %f - %f = %f\n', i, g(2), g(Xn(i+1)), g(2)-g(Xn(i+1)));
    pause();
end
```

Newton.m

main.m | Newton.m

```matlab
function [x1,N] = Newton(f, df, x0, k, tol, save)
N = 1;
X(1) = x0;
while abs(f(x0)) > tol
    N = N+1;
    x1 = x0 - k*f(x0)/df(x0);
    if save == 1
        X(N) = x1;
    end
    x0 = x1;
    %fprintf('val = %f\n', abs(f(x0)))
end

if save == 1
    x1 = X;
end
```

交大資工
110550062 朱致伶

3. (30%) Below are three different g(x) functions. All are rearrangements of the same f(x). What is f(x)?

(a) $\frac{(4+2x^3)}{x^2} - 2x$ **(b)** $\sqrt{4/x}$ **(c)** ($16 + x^3$) / ($5x^2$)

Which of them converge? What x-value is obtained? Are there starting values for which one or more diverge? Which diverge?

Ans: f(x) = x^3-4, (b) converges to x=1.587400 when x≠0 , x=0 as starting value makes (a), (b) and (c) diverged, (a) and (c) diverges anyway

(a) (4+2x^3)/x^2 – 2x = x
(4+2x^3)/x^2 = 3x
4+2x^3 = 3x^3
4 = x^3

(b) (4/x)^1/2 = x
4/x = x^2
4 = x^3

(c) (16+x^3) / (5x^2) = x
16+x^3 = 5x^3
16 = 4x^3
4 = x^3

From above mathematic derivation, we know f(x) = $x^3 - 4$.

When taking x0 = 0 as the starting value, the three equations all diverge. As a consequence, I set x0 = 1 as the starting value.

Only equation B converges, which converges to root x= 1.587400 when x≠0. To proof f(x) converges at root=1.587402, I calculated that $|g'(x)| = |\frac{f(x)\,f''(x)}{[f'(x)]^2}|$ = 0.000001 < 1, which implies that root=1.587402 converges in Newton's method.

The reason why equation A diverges is that the root value(x) accidently equals 0 during the Newton's approaching process. However, we know that 0 can't be divided by any number. Therefore, when x was substituted with 0 in equation A, the number become infinity, which diverges.

As for equation C, the reason why it diverges is that the root values are looped, the g value of roots are switching between g(x) = 43.094011 and g(x) = 3.094011.

Equation A

```
root = 2.777778
root = 0.518400
root = 14.884355
root = 0.018055
root = 12270.437031
root = 0.000000
root = 5665237650236087.000000
root = 0.000000
root = Inf
root = NaN
```

Equation B

```
root = 2.000000
root = 1.414214
root = 1.681793
root = 1.542211
root = 1.610490
root = 1.575981
root = 1.593142
root = 1.584538
root = 1.588834
root = 1.586685
root = 1.587759
root = 1.587222
root = 1.587491
root = 1.587356
root = 1.587423
root = 1.587390
root = 1.587407
root = 1.587398
root = 1.587402
root = 1.587400
```

Equation C

```
val = 43.094011
val = 3.094011
val = 43.094011
val = 3.094011
val = 43.094011
val = 3.094011
val = 43.094011
val = 3.094011
Operation terminated by user during inlineeva
```

交大資工
110550062 朱致伶

main.m

```
clear all;
clf;
save = 1;
f = inline('x.^3 - 4','x');
df = inline('3*x.^2', 'x');
ddf = inline('6*x', 'x');
%g = inline('(4+2*x.^3)/x.^2 - 2*x', 'x');
g = inline('2/(x.^(1/2))', 'x');
%g = inline('(16+x.^3)/(5*x.^2)', 'x');
x0 = 1;
tol = 0.00001;
[Xn, N] = Newton(f,g,x0,tol,save);
dg = f(Xn(N))*ddf(Xn(N))/(df(Xn(N))*df(Xn(N)));
fprintf("dg = %f\n", dg)

X = linspace(Xn(1),Xn(N),100);
Y = f(X);
for i=1:N-1
    clf;
    plot(X,Y);
    hold on;
    line([Xn(i), Xn(i+1)],[f(Xn(i)), 0]);
    plot(Xn(i), f(Xn(i)), 'ro');
    plot(Xn(i+1), 0, 'r*');
    grid;
    fprintf('root = %f\n', Xn(i+1));
    pause();
end
```

Newton.m

```
function [x1,N] = Newton(f,g,x0,tol,save)
N = 1;
X(1) = x0;
while abs(f(x0)) > tol
    N = N+1;
    x1 = g(x0);
    if save == 1
        X(N) = x1;
    end
    x0 = x1;
    fprintf('val = %f\n', abs(f(x0)))
end

if save == 1
    x1 = X;
end
```

交大資工
110550062 朱致伶

4. (30%) Solve the following system of nonlinear equations using Newton's method or fixed-point method.

$$\boldsymbol{x - 3y - z^2 = -3}$$
$$\boldsymbol{2x^3 + y - 5z^2 = -2}$$
$$\boldsymbol{4x^2 + y + z = 7}$$

Hints: There are six solutions to this system. Two of the real solutions are near (1, 1, 1) and (1.3, 0.9, -1.2). Your score of this question will depend on how many solutions you find.

Ans:

1. (1.111408, 0.988210, 1.070878)
2. (1.353748, 0.925431, -1.255969)
3. (31.151405, -3768.157126, -106.482969)
4. (32.884631, -4434.086620, 115.490885)
5. (- 1.25060 - 0.04899i , 0.66505 + 0.01341i , 0.08859 – 0.50359i)
6. (- 1.25060 + 0.04899i , 0.66505 - 0.01341i , 0.08859 + 0.50359i)

In mainNewtonRaphson.m, the three for loops aim to traverse through the range around the given solution with x decreases -0.001, y decreases -0.001 and z increases 0.1 each time. To find complex solution, use complex(a, b) function as a+bi, and modify the input and output format as 6 number with 3 pairs of (a, b).

In NewtonRaphson.m, make sure the founded solution is under $e = 10^{-5}$, and check whether the founded solution already exists.

mainNewtonRaphson.m | NewtonRaphson_nl_print.m | main.m | Newton.m

```
fn = @(v) [v(1)-3*v(2)-v(3)^2+3 ; 2*v(1)^3+v(2)-5*v(3)^2+2 ; 4*v(1)^2+v(2)+v(3)-7];
jacob_fn = @(v) [1 -3 -2*v(3);6*v(1)^2 1 -10*v(3);8*v(1) 1 1];
error = 10^-5 ;
x_init = 2.5;
y_init = 1;
z_init = -0.2;
no_itr = 20 ;
roots = [];
fprintf('      Iteration|     x      |      y      |     z       | Error       | \n')

start = 200;
distance = -0.2;
theand = 150;

for xi = -1.865:-0.001:-1.875
    for yi = 3.1:-0.001:3
        for zi = 15:0.1:25
            x_init = xi;
            y_init = yi;
            z_init = zi;
            v = [x_init;y_init;z_init] ;
            %fprintf("\n%f %f %f",v(1),v(2),v(3))
            [v1,no_itr,norm1,success,exists,roots]=NewtonRaphson_nl_print(v,fn,jacob_fn,no_itr,error,roots);
            if success == 1
                new = [v1(1) v1(2) v1(3)];
                if exists==0
                    roots = [roots;new];
                end
                %s= size(roots);
                %display(size(roots));
                %display(s(2))
                %for j = 1:s(1)
                %    fprintf("%f %f %f",roots(j,1),roots(j,2),roots(j,3));
                %end
            end

        end
    end
end
s = size(roots);
for j = 1:s(1)
    fprintf("\n%f %f %f",roots(j,1),roots(j,2),roots(j,3));
end
fprintf("\n");
```

交大資工

110550062 朱致伶

mainNewtonRaphson.m | NewtonRaphson_nl_print.m | main.m | Newton.m

```
function [v1 , no_itr, norm1, success,exists, roots] = NewtonRaphson_nl_print(v,fn,jacob_fn,no_itr,error,roo
    v1 = v;
    fnv1 = feval(fn,v1);
    i = 0;

    success=0;
    exists=0;
    while true
        norm1 = norm(fnv1);
        %fprintf('%10d     |%10.4f| %10.4f | %10.4f| %10.4d |\n',i,v1(1),v1(2),v1(3),norm1)
        jacob_fnv1 = feval(jacob_fn,v1);
        H = jacob_fnv1\fnv1;
        v1 = v1 - H;
        fnv1 = feval(fn,v1);
        i = i + 1 ;
        norm1 = norm(fnv1);
        if norm1 < error
            success=1;

            s=size(roots);
            for i = 1:s(1)
                dis1= abs(v1(1)-roots(i,1));
                dis2= abs(v1(2)-roots(i,2));
                dis3= abs(v1(3)-roots(i,3));
                if (dis1+dis2+dis3)<3*1e-4
                    exists=1;
                end
            end
            if exists == 0
                fprintf('%10d     |%10.4f| %10.4f | %10.4f| %10.4d |\n',i,v1(1),v1(2),v1(3),norm1)
            end
            break
        end
        if i > no_itr% && norm1 > error
            fprintf("      fail");
            %fprintf('%10d     |%10.4f| %10.4f | %10.4f| %10.4d | fail\n',i,v1(1),v1(2),v1(3),norm1)
            break;
        end

```