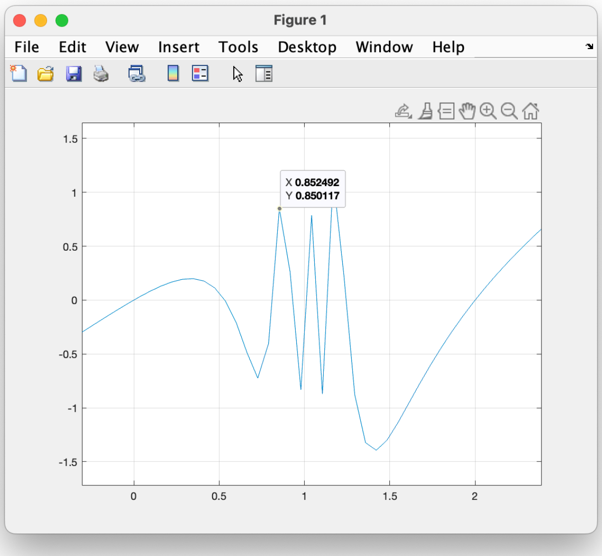
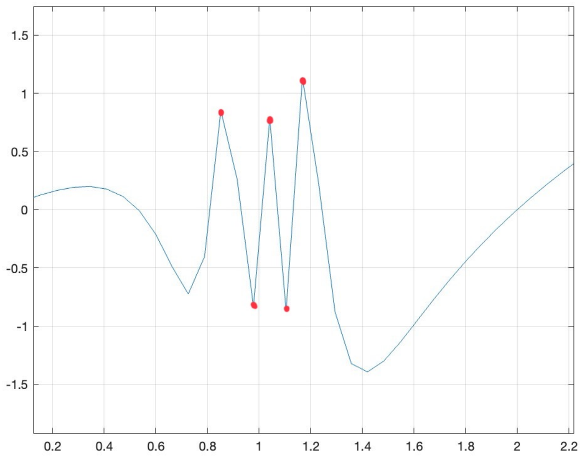
Assignment 1

111550057 資工15 莊婷馨

1. To find good starting intervals, I displayed the function in Matlab. Using the points displayed in the figure, I am able to find intervals that brackets the roots.





Then start the process of bisection method. Iterate by halving the bounds by half. Always replace one of the original bound that has the same sign with the new bound.

```
function [root,iter] = bisection(f,x0,x1,tol)
    if f(x0) * f(x1) > 0
        error('Function has same sign at interval endpoints.');
    end
    iter = 0;
    while abs(x0-x1) > tol
        iter = iter+1;
        m = (x0+x1)/2;
        if f(m) == 0
            root = m;
            return;
        elseif f(m)*f(x0) < 0
            x1 = m;
        else
            x0 = m;
        end
    end
        root = (x0+x1)/2;
```

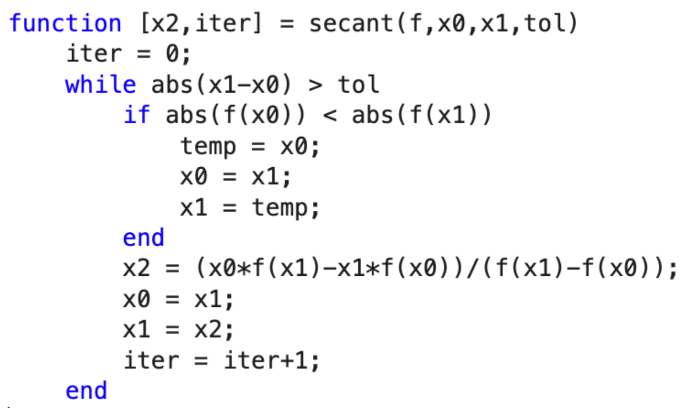
in “bisection.m”

Once $|x_0 - x_1| \leq tol$, the function returns the value of root and iterations used.

The result :

```
Bisection:
    0.8107
    0.9633
    1.0105
    1.0599
```

2. Use the same f(x) and bounds as Problem 1. Start the process of secant method. Iterate by calculating the secant line formed by two given points. The new value x2 comes from the point (x2,0) where the secant line intersects with y=0. Then replace x0 with x1, x1 with x2 until $|x_1 - x_0| \leq tol$

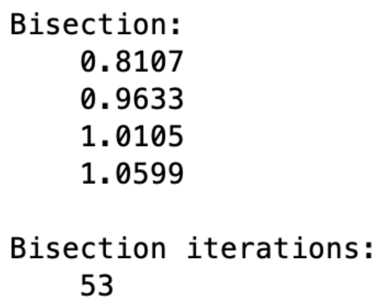


```
function [x2,iter] = secant(f,x0,x1,tol)
    iter = 0;
    while abs(x1-x0) > tol
        if abs(f(x0)) < abs(f(x1))
            temp = x0;
            x0 = x1;
            x1 = temp;
        end
        x2 = (x0*f(x1)-x1*f(x0))/(f(x1)-f(x0));
        x0 = x1;
        x1 = x2;
        iter = iter+1;
    end
```

in "secant.m"

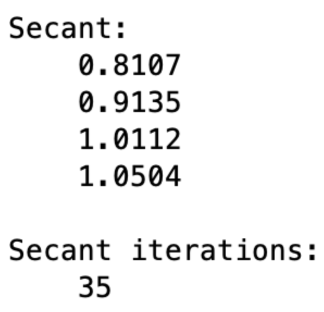
Calculate the number of iterations during the process. Compare the results of both methods.

The results :



```
Bisection:
     0.8107
     0.9633
     1.0105
     1.0599

Bisection iterations:
     53
```



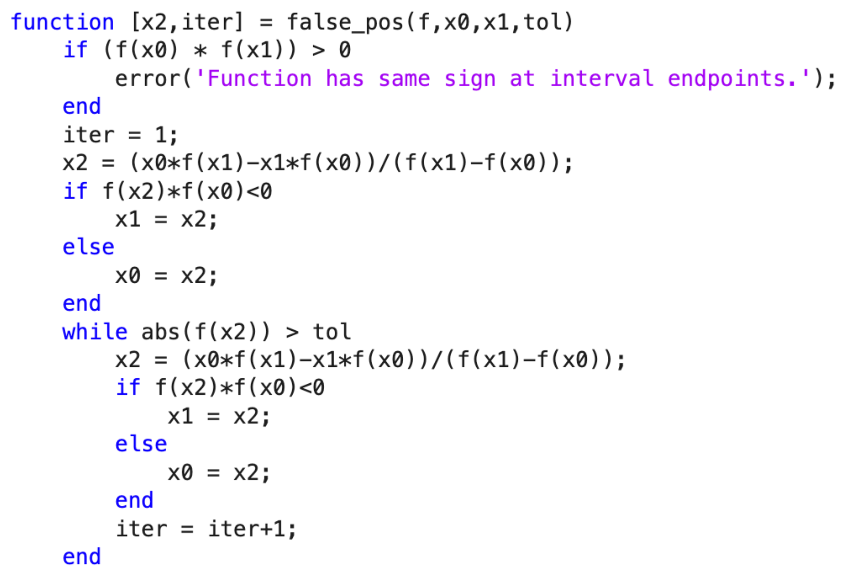
```
Secant:
     0.8107
     0.9135
     1.0112
     1.0504

Secant iterations:
     35
```

It turns out that secant method requires 18 less iterations than bisection method.

3. For $p(x) = (x-2)^3(x-4)^2$
   (a) Bisection method use sign changes to detect roots. The root at $x = 2$ has multiplicity of 3, so it can be detected by bisection method. The root at $x = 4$ has multiplicity of 2, so it cannot be detected using bisection method.
   (b) Both roots can be detected by secant method if proper bounds are chosen. However, it might take longer to detect the root at $x = 2$ since it has higher multiplicity.
   (c) Use the same bisection and secant method function in the previous problems. Implement false position method by following code. Iterate by calculating the secant line formed by previous two points. The new x2 is the x value of the

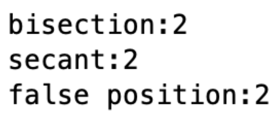
intersection of secant line and y=0. Replace one of the previous points that has the same sign with the new point (x2,f(x2)).



```
function [x2,iter] = false_pos(f,x0,x1,tol)
    if (f(x0) * f(x1)) > 0
        error('Function has same sign at interval endpoints.');
    end
    iter = 1;
    x2 = (x0*f(x1)-x1*f(x0))/(f(x1)-f(x0));
    if f(x2)*f(x0)<0
        x1 = x2;
    else
        x0 = x2;
    end
    while abs(f(x2)) > tol
        x2 = (x0*f(x1)-x1*f(x0))/(f(x1)-f(x0));
        if f(x2)*f(x0)<0
            x1 = x2;
        else
            x0 = x2;
        end
        iter = iter+1;
    end
```

in “false_pos.m”

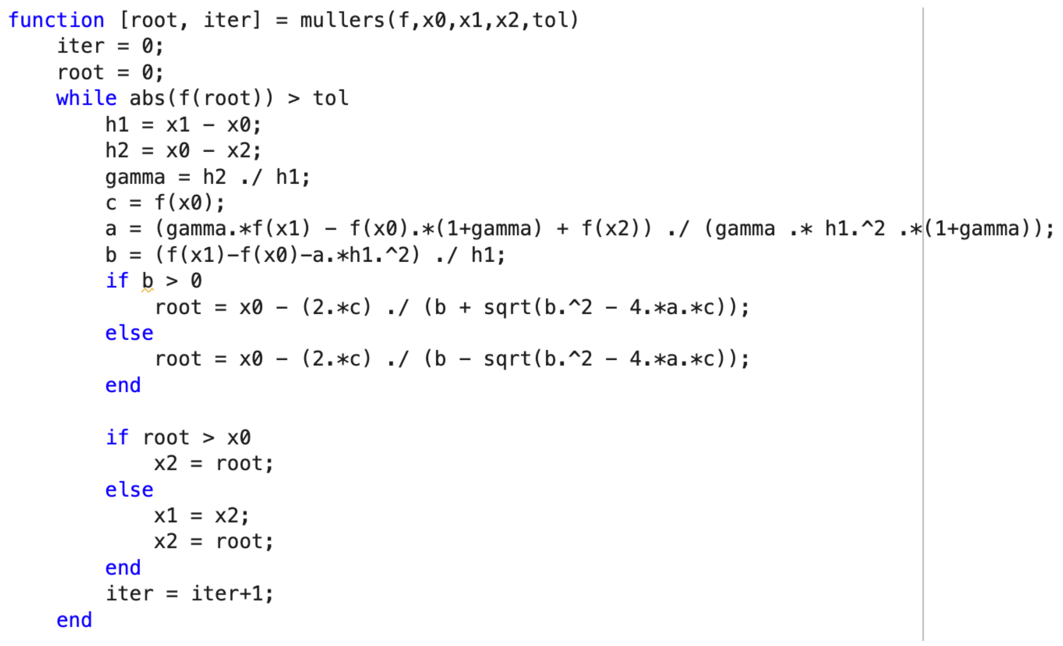
Given tolerance = 1e-5. The results:



```
bisection:2
secant:2
false position:2
```

With all three methods, the root obtained in [1,5] is x=2.

4. Implement Muller’s method by following code. Calculate all parameters to form a quadric equation $av^2 + bv + c$ that fits the three points (x0, f(x0)), (x1, f(x1)) and (x2, f(x2)). Choose root of the equation closest to x0. Update the points according to the value of the root, then do the next iteration. Iterate until $|f(root)| \leq tol$.



```
function [root, iter] = mullers(f,x0,x1,x2,tol)
    iter = 0;
    root = 0;
    while abs(f(root)) > tol
        h1 = x1 - x0;
        h2 = x0 - x2;
        gamma = h2 ./ h1;
        c = f(x0);
        a = (gamma.*f(x1) - f(x0).*(1+gamma) + f(x2)) ./ (gamma .* h1.^2 .*(1+gamma));
        b = (f(x1)-f(x0)-a.*h1.^2) ./ h1;
        if b > 0
            root = x0 - (2.*c) ./ (b + sqrt(b.^2 - 4.*a.*c));
        else
            root = x0 - (2.*c) ./ (b - sqrt(b.^2 - 4.*a.*c));
        end

        if root > x0
            x2 = root;
        else
            x1 = x2;
            x2 = root;
        end
        iter = iter+1;
    end
```

in “mullers.m”

(a) To get the root near x=0.6, set x0=0.6. I choose x1 and x2 by a rather reasonable range that brackets x0. Then I set x1=0 and x2=1 since this will bracket 0.6. Set tolerance = 1e-5.

The result:
```
root near x=0.6: 0.60583
```

(b) To get the root near x=1, I set x0=1, x1=0 and x2=2.
To get the root near x=-2, I set x0=-2, x1=-3 and x2 =-1.
Set tolerance = 1e-5.

The result:
```
root near x=1: 1.2576
root near x=-2: -3.9759
```

5. Implement fixed point method by the following code. If the absolute value of the previous point and current is less or equal to the tolerance, that means there is convergence. Iterate by getting the new x from g(x).

```
function root = fixed_point(g, x, tol)
    i = 0;
    prev = x-1;
    while abs(prev-x) > tol
        prev = x;
        x = g(x);
        i = i+1;
    end
    root = x;
```

in "fixed_point.m"

(a) Use both positive and negative values of g(x) to begin the fixed point method. Both use x=0.5 as the starting point. Set tolerance = 1e-5.

The result:
```
root(positive):1.4879
root(negative):-0.53984
```

The result proves that using the positive value converges to the root near x=1.5, and using the negative value converges to the root near x=-0.5.

(b) The results:

```
x=1.488(positive g(x), starts at 2.5)
x=Inf(positive g(x), starts at 2.7)
x=-0.53983(negative g(x), starts at 2.5)
x=-0.53983(negative g(x), starts at 2.7)
```

As you can see, it either converges to the root near x=1.5 or x=-0.5. The one using positive value and starts at 2.7 even diverges.

(c) To find another rearrangement, we need to find another g(x). By calculation:

$$e^x = 2x^2$$

$$\log(e^x) = \log(2x^2)$$

$$x = \log(2x^2) = g(x)$$

Using $\log(2x^2)$ as g(x) and both x=2.5 and 2.7 as starting point. Apply fixed point method.

The results:

```
g(x)=log(2*x^2)
root:2.6178(starts at 2.5)
root:2.6179(starts at 2.7)
```

It shows that both starting at 2.5 and 2.7 will converge to the root near x=2.6.

6. By substituting *y* with $\cos^2(x)$, we can get

$$f(x) = x^2 + \cos^4(x) - x - 2$$

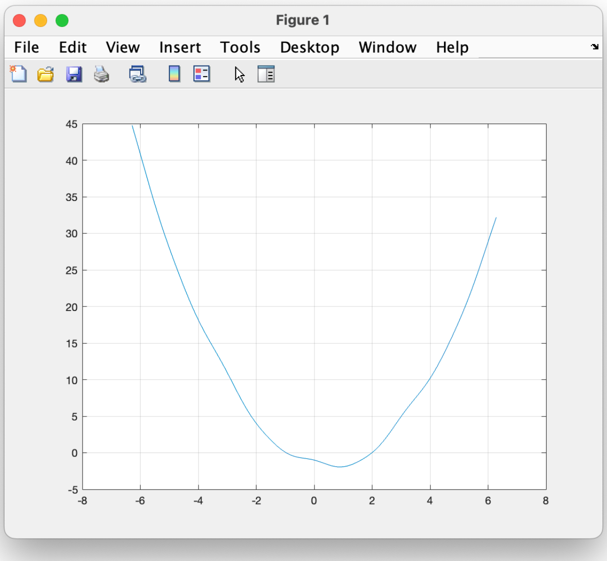
Calculate the derivative of *f*(*x*), *f'*(*x*)

$$f'(x) = -4\cos^3 x \sin x + 2x - 1$$

Implement Newton's method by the following code. Find the next x0 by finding the intersection of the tangent line and *y=0*. The slope is obtain through *f'*(*x*). Iterate until $|f(x_0)| \le tol$.

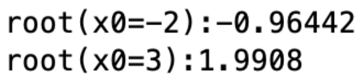
```
function x1 = newton(f,df,x0,tol)
    i = 0;
    while abs(f(x0)) > tol
        x1 = x0-f(x0)/df(x0);
        x0 = x1;
        i = i+1;
    end
end
```

In order to find the root of the system, I first checked the plot of *f*(*x*).



It is clear that there is one root near x=1.5 and another root near x=2. Choose x=-2 and x=3 as the starting points to enable the approximation, since the tangent line has to tilt toward the root for the root to be discovered. Set tolerance = 1e-5.

The results:



```
root(x0=-2):-0.96442
root(x0=3):1.9908
```