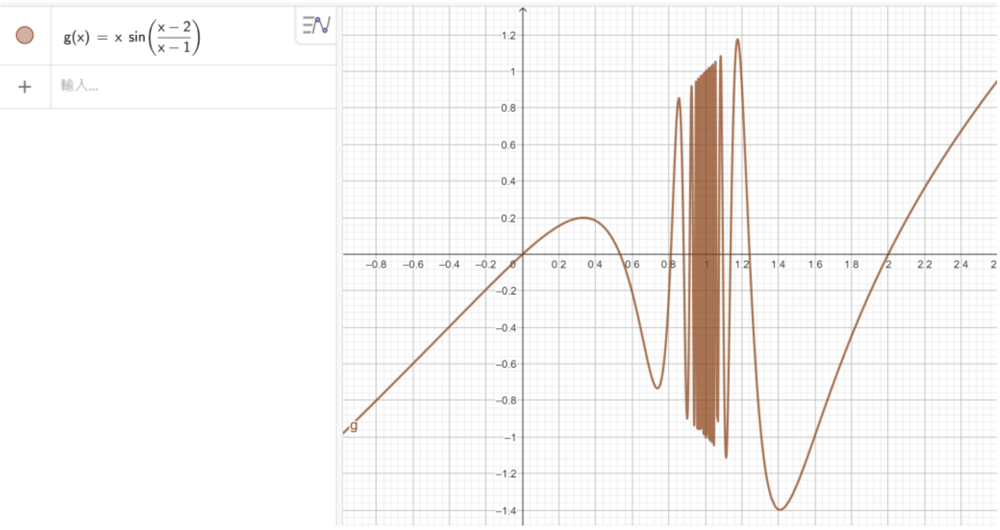
111550113 謝詠晴

Numerical Methods_ Assignment 1

1. Using bisection method, <u>the four zeros are 0.94397, 0.95235, 0.95856, 0.96333.</u>

   First, I use GeoGebra to draw the function to get a clearer insight.



Then define the ‘bisection’ function, which is written in ‘bisection.m’

- N is the iteration count.
- M[ ] is an array to store the midpoints of each iteration.
- Repeatedly divide the interval, and check if the root is in the left or right side of the interval, and update the boundary.

bisection.m | secant.m | q1_main.m

```
% Bisection method

function [N,M] = bisection(f, L, R, tol)
N = 0;
M = [];
while abs(L-R) > tol
    N = N+1;
    mid = (L+R)/2;
    if(f(mid)==0)
       M(N) = mid;
       break;
    elseif (f(L)*f(mid)<0)
        R = mid;
    elseif (f(R)*f(mid)<0)
        L = mid;
    end
    M(N) = mid;
end
end
```

Since the question said that the zero should be near 0.95, so I try the interval [0.94, 0.95] and [0.95, 0.96] first. And the roots are close to each other, so I increase the precision of the decimal places for the interval by trying [0.955, 0.965] and [0.959, 0.965], thus find the other two roots.

The implementation is written in ‘q1_main.m’.

```
>> q1_main
interval:  [L, R] = [0.94000, 0.95000]
(N=1): mid = 0.94500
(N=2): mid = 0.94250
(N=3): mid = 0.94375
(N=4): mid = 0.94437
(N=5): mid = 0.94406
(N=6): mid = 0.94391
(N=7): mid = 0.94398
(N=8): mid = 0.94395
(N=9): mid = 0.94396
(N=10): mid = 0.94397
find root at around: 0.94397
```

```
>> q1_main
interval:  [L, R] = [0.95000, 0.96000]
(N=1): mid = 0.95500
(N=2): mid = 0.95250
(N=3): mid = 0.95125
(N=4): mid = 0.95187
(N=5): mid = 0.95219
(N=6): mid = 0.95234
(N=7): mid = 0.95242
(N=8): mid = 0.95238
(N=9): mid = 0.95236
(N=10): mid = 0.95235
find root at around: 0.95235
```

```
>> q1_main
interval:  [L, R] = [0.95500, 0.96500]
(N=1): mid = 0.96000
(N=2): mid = 0.95750
(N=3): mid = 0.95875
(N=4): mid = 0.95813
(N=5): mid = 0.95844
(N=6): mid = 0.95859
(N=7): mid = 0.95852
(N=8): mid = 0.95855
(N=9): mid = 0.95857
(N=10): mid = 0.95856
find root at around: 0.95856
```

```
>> q1_main
interval:  [L, R] = [0.95900, 0.96500]
(N=1): mid = 0.96200
(N=2): mid = 0.96350
(N=3): mid = 0.96275
(N=4): mid = 0.96313
(N=5): mid = 0.96331
(N=6): mid = 0.96341
(N=7): mid = 0.96336
(N=8): mid = 0.96334
(N=9): mid = 0.96332
(N=10): mid = 0.96333
find root at around: 0.96333
```

2. Using secant method, the four zeros are 0.94398, 0.95236, 0.95856, 0.96333.

   It requires less iterations (4 to 8) than bisection method (10 to 12).

   Define the 'secant' function, which is written in 'secant.m'.

   - N is the iteration count and X[] is an array which stores x2 of each iteration.
   - It uses x0, x1 and f(x0), f(x1) to compute a secant line and find the intersection point x2 of the line and x-axis. Then update the boundary value until abs(x1-x0) is smaller than tolerance value.

bisection.m | secant.m * | q1_main.m

```
% Secant method

function [N,X] = secant(f, x0, x1, tol)
N = 0;
X = [];

while abs(x1-x0) > tol
    N = N+1;
    if abs(f(x0)) < abs(f(x1))
    tmp = x0;
    x0 = x1;
    x1 = tmp;
    end
    x2 = x1-f(x1)*(x0-x1)/(f(x0)-f(x1));
    x0 = x1;
    x1 = x2;
    X(N) = x2;
end
end
```

I tried the same interval as in bisection method, but the output results are not ideal. Therefore, I made a slightly modification to the interval. The implementation is written in 'q2_main.m'.

```
>> q1_main
interval:  [L, R] = [0.94000, 0.95000]
(N=1): x2 = 0.94523
(N=2): x3 = 0.94365
(N=3): x4 = 0.94398
find root at around: 0.94398
```

```
>> q1_main
interval:  [L, R] = [0.94800, 0.95800]
(N=1): x2 = 0.95554
(N=2): x3 = 0.95172
(N=3): x4 = 0.95254
(N=4): x5 = 0.95236
find root at around: 0.95236
```

```
>> q1_main
interval:  [L, R] = [0.95900, 0.96500]
(N=1): x2 = 0.96025
(N=2): x3 = 0.95847
(N=3): x4 = 0.95858
(N=4): x5 = 0.95856
find root at around: 0.95856
```

```
>> q1_main
interval:  [L, R] = [0.94000, 0.96000]
(N=1): x2 = 0.95086
(N=2): x3 = 0.94657
(N=3): x4 = 0.96673
(N=4): x5 = 0.96026
(N=5): x6 = 0.96483
(N=6): x7 = 0.96246
(N=7): x8 = 0.96339
(N=8): x9 = 0.96333
find root at around: 0.96333
```

3. The implementation is written in ‘q3_main.m.’

(a) By trial and error, bisection method <u>can only find the triple root x = 2, it can’t find the double root x = 4</u>. The reason is that when checking $f(a) \cdot f(b) < 0$, this circumstance won’t meet since $f(a)$ and $f(b)$ have the same sign.

```
>> q3_main
interval:  [L, R] = [1.50000, 3.50000]
(N=1): mid = 2.50000
(N=2): mid = 2.00000
find root at around: 2.00000
```

It will turn out to be an infinite loop since having same sign.

```
>> q3_main
interval:  [L, R] = [3.50000, 5.50000]
```

(b) While using secant method, <u>it can find both roots x = 2 and x = 4</u> but requiring more iterations. First, use interval [1.5, 3] to find root x = 2.

```
interval:  [L, R] = [1.50000, 3.00000]
secant:
(N=1): x2 = 2.15789
(N=2): x3 = 2.14684
(N=3): x4 = 2.09846
(N=4): x5 = 2.07596
(N=5): x6 = 2.05600
(N=6): x7 = 2.04218
(N=7): x8 = 2.03161
(N=8): x9 = 2.02378
(N=9): x10 = 2.01790
(N=10): x11 = 2.01348
(N=11): x12 = 2.01016
(N=12): x13 = 2.00766
(N=13): x14 = 2.00578
(N=14): x15 = 2.00436
(N=15): x16 = 2.00329
(N=16): x17 = 2.00248
(N=17): x18 = 2.00187
(N=18): x19 = 2.00141
(N=19): x20 = 2.00107
(N=20): x21 = 2.00080
```

```
(N=21): x22 = 2.00061
(N=22): x23 = 2.00046
(N=23): x24 = 2.00035
(N=24): x25 = 2.00026
(N=25): x26 = 2.00020
(N=26): x27 = 2.00015
(N=27): x28 = 2.00011
(N=28): x29 = 2.00008
(N=29): x30 = 2.00006
(N=30): x31 = 2.00005
(N=31): x32 = 2.00004
(N=32): x33 = 2.00003
find root at around: 2.00003
```

Then use interval [3.4, 4.6] to find root x = 4.

```
>> q3_main
interval:  [L, R] = [3.40000, 4.60000]
secant:
(N=1): x2 = 3.17799
(N=2): x3 = 5.27933
(N=3): x4 = 3.36728
(N=4): x5 = 4.31197
(N=5): x6 = -0.79178
(N=6): x7 = 3.39173
(N=7): x8 = 4.25694
(N=8): x9 = 7.09813
(N=9): x10 = 4.25524
(N=10): x11 = 4.14667
(N=11): x12 = 4.10344
(N=12): x13 = 4.06542
(N=13): x14 = 4.04225
(N=14): x15 = 4.02659
(N=15): x16 = 4.01670
(N=16): x17 = 4.01041
(N=17): x18 = 4.00648
(N=18): x19 = 4.00402
(N=19): x20 = 4.00249
(N=20): x21 = 4.00154
(N=21): x22 = 4.00095
(N=22): x23 = 4.00059
(N=23): x24 = 4.00036
(N=24): x25 = 4.00023
(N=25): x26 = 4.00014
(N=26): x27 = 4.00009
(N=27): x28 = 4.00005
(N=28): x29 = 4.00003
(N=29): x30 = 4.00002
(N=30): x31 = 4.00001
find root at around: 4.00001
```

(c) Begin with the interval [1, 5]. <u>All three methods can only find the root x = 2</u>.

```
>> q3_main
interval:  [L, R] = [1.00000, 5.00000]
bisection:
(N=1): mid = 3.00000
(N=2): mid = 2.00000
find root at around: 2.00000
secant:
(N=1): x2 = 2.00000
(N=2): x3 = 2.00000
find root at around: 2.00000
false_position:
(N=1): x2 = 2.00000
find root at around: 2.00000
```

The bisection and secant method use the same function in problem 1 and 2.

And the false position method is performed by the function ‘false_position’ written in ‘false_position.m’.

It updates the new intersection point x2 like secant method, but the difference is that it is a bracketing method, and it checks f(x0)*f(x2)<0 to determine the next boundary.

```
% false position method

function [N,X] = false_position(f, x0, x1, tol)
N = 1;
X = [];
x2 = x1-f(x1)*(x0-x1)/(f(x0)-f(x1));
X(N) = x2;
if f(x0)*f(x2)<0
    x1 = x2;
else
    x0 = x2;
end

while abs(f(x2)) > tol
    N = N+1;
    if f(x0)*f(x2)<0
        x1 = x2;
    else
        x0 = x2;
    end
    X(N) = x2;
    x2 = x1-f(x1)*(x0-x1)/(f(x0)-f(x1));
end
end
```

3. Using Muller's method to find the root.

   Define the function 'muller' in 'muller.m'.

   - 'iter' is the iteration count until it finds the root.
   - Find three points on the function: (x0, f(x0)), (x1, f(x1)), (x2, f(x2))
   - Set $h_0 = x_1 - x_0,\ h_1 = x_2 - x_1$
   - Set $d_0 = \frac{f(x_1)-f(x_0)}{h0},\ d_1 = \frac{f(x_2)-f(x_1)}{h1}$
   - $a = \frac{d_1 - d_0}{h0+h1}$, $b = ah_1 + d_1$, $c = f(x_2)$
   - $x_r = x_2 - \frac{2*c}{b \pm \sqrt{b^2-4ac}}$, and update the x0, x1,x2 values with x1, x2, xr

```
% Muller's method
function [iter,root] = muller(f, x0, x1, x2, max_iter, tol)
iter = 0;
h0 = x1-x0;
h1 = x2-x1;
d0 = (f(x1)-f(x0))/ h0;
d1 = (f(x2)-f(x1))/ h1;
while iter < max_iter
    a = (d1 - d0)/(h0 + h1);
    b = a*h1 + d1;
    c = f(x2);
    disc = sqrt(b^2 - 4*a*c);
    if abs(b-disc) < abs(b+disc)
        den = b + disc;
    else
        den = b - disc;
    end
    xr = x2 - 2*c/den;
    if abs(- 2*c/den) < tol
        root = xr;
        break;
    end

    iter = iter + 1;
    x0 = x1;
    x1 = x2;
    x2 = xr;
    h0 = x1-x0;
    h1 = x2-x1;
    d0 = (f(x1)-f(x0))/ h0;
    d1 = (f(x2)-f(x1))/ h1;
end
end
```

The implementation is written in ‘q4_main.m’.

(a) $4x^3 - 3x^2 + 2x - 1 = 0$, root near x = 0.6. The answer is: 0.60583.

```
x0 = 0.3; x1 = 0.5; x2 = 0.8;
```

```
>> q4_main
iter: 3, root: 0.60583
```

(b) $x^2 + e^x = 5$, root near x = 1, x = -2. The answers are: 1.24114 and -2.21144.

```
x0 = 0.5; x1 = 1.1; x2 = 1.5;
```

```
>> q4_main
iter: 2, root: 1.24114
```

```
x0 = -2.5; x1 = -1.5; x2 = -1;
```

```
>> q4_main
iter: 2, root: -2.21144
```

5. (a) $x = \sqrt{\frac{e^x}{2}}$ converges to 1.48796 and $x = -\sqrt{\frac{e^x}{2}}$ converges to -0.53984.

The implementation of fixed-point method is written in ‘q5_main.m’

Start $x = \sqrt{\frac{e^x}{2}}$ with x = 1:

q5_main.m | q5_3.m

```
% fixed-point method
max_iter = 40;
iter = 0;
x = 1;
fprintf('x0 = %.2f\n', x);

while iter < max_iter
    iter = iter + 1;
    x = sqrt(exp(x)/2);
    fprintf('(iter=%.2d), x:%.5f    ', iter, x);
    if mod(iter, 4) ==0
        fprintf('\n')
    end
end
```

```
>> q5_main
x0 = 1.00
(iter=01), x:1.16582    (iter=02), x:1.26660    (iter=03), x:1.33206    (iter=04), x:1.37638
(iter=05), x:1.40722    (iter=06), x:1.42909    (iter=07), x:1.44480    (iter=08), x:1.45619
(iter=09), x:1.46451    (iter=10), x:1.47062    (iter=11), x:1.47511    (iter=12), x:1.47843
(iter=13), x:1.48089    (iter=14), x:1.48271    (iter=15), x:1.48406    (iter=16), x:1.48506
(iter=17), x:1.48581    (iter=18), x:1.48636    (iter=19), x:1.48677    (iter=20), x:1.48708
(iter=21), x:1.48730    (iter=22), x:1.48747    (iter=23), x:1.48760    (iter=24), x:1.48769
(iter=25), x:1.48776    (iter=26), x:1.48781    (iter=27), x:1.48785    (iter=28), x:1.48788
(iter=29), x:1.48790    (iter=30), x:1.48792    (iter=31), x:1.48793    (iter=32), x:1.48794
(iter=33), x:1.48794    (iter=34), x:1.48795    (iter=35), x:1.48795    (iter=36), x:1.48795
(iter=37), x:1.48796    (iter=38), x:1.48796    (iter=39), x:1.48796    (iter=40), x:1.48796
```

Start $x = -\sqrt{\frac{e^x}{2}}$ with x = -1:

```
x0 = -1.00
(iter=01), x:-0.42888    (iter=02), x:-0.57063    (iter=03), x:-0.53159    (iter=04), x:-0.54207
(iter=05), x:-0.53923    (iter=06), x:-0.54000    (iter=07), x:-0.53979    (iter=08), x:-0.53985
(iter=09), x:-0.53983    (iter=10), x:-0.53984    (iter=11), x:-0.53984    (iter=12), x:-0.53984
(iter=13), x:-0.53984    (iter=14), x:-0.53984    (iter=15), x:-0.53984    (iter=16), x:-0.53984
```

(b) Start $x = \sqrt{\frac{e^x}{2}}$ with x = 2.59, it does not converge to the third root x = 2.6.

```
>> q5_main
(iter=01), x:2.58164    (iter=02), x:2.57088    (iter=03), x:2.55708    (iter=04), x:2.53950
(iter=05), x:2.51727    (iter=06), x:2.48945    (iter=07), x:2.45506    (iter=08), x:2.41321
(iter=09), x:2.36324    (iter=10), x:2.30492    (iter=11), x:2.23868    (iter=12), x:2.16575
(iter=13), x:2.08820    (iter=14), x:2.00877    (iter=15), x:1.93057    (iter=16), x:1.85653
(iter=17), x:1.78906    (iter=18), x:1.72972    (iter=19), x:1.67914    (iter=20), x:1.63722
(iter=21), x:1.60325    (iter=22), x:1.57626    (iter=23), x:1.55512    (iter=24), x:1.53878
(iter=25), x:1.52625    (iter=26), x:1.51672    (iter=27), x:1.50951    (iter=28), x:1.50408
(iter=29), x:1.50000    (iter=30), x:1.49695    (iter=31), x:1.49466    (iter=32), x:1.49296
(iter=33), x:1.49168    (iter=34), x:1.49073    (iter=35), x:1.49002    (iter=36), x:1.48950
(iter=37), x:1.48910    (iter=38), x:1.48881    (iter=39), x:1.48859    (iter=40), x:1.48843
```

If x0 = 2.5, it will converge to 1.48796.

```
>> q5_main
x0 = 2.50
(iter=01), x:2.46805    (iter=02), x:2.42893    (iter=03), x:2.38188    (iter=04), x:2.32650
(iter=05), x:2.26297    (iter=06), x:2.19221    (iter=07), x:2.11601    (iter=08), x:2.03690
(iter=09), x:1.95791    (iter=10), x:1.88209    (iter=11), x:1.81207    (iter=12), x:1.74973
(iter=13), x:1.69603    (iter=14), x:1.65110    (iter=15), x:1.61442    (iter=16), x:1.58508
(iter=17), x:1.56200    (iter=18), x:1.54408    (iter=19), x:1.53030    (iter=20), x:1.51980
(iter=21), x:1.51184    (iter=22), x:1.50583    (iter=23), x:1.50132    (iter=24), x:1.49793
(iter=25), x:1.49540    (iter=26), x:1.49350    (iter=27), x:1.49209    (iter=28), x:1.49104
(iter=29), x:1.49025    (iter=30), x:1.48967    (iter=31), x:1.48923    (iter=32), x:1.48891
(iter=33), x:1.48866    (iter=34), x:1.48848    (iter=35), x:1.48835    (iter=36), x:1.48825
(iter=37), x:1.48818    (iter=38), x:1.48812    (iter=39), x:1.48808    (iter=40), x:1.48805
(iter=41), x:1.48803    (iter=42), x:1.48801    (iter=43), x:1.48800    (iter=44), x:1.48799
(iter=45), x:1.48798    (iter=46), x:1.48798    (iter=47), x:1.48797    (iter=48), x:1.48797
(iter=49), x:1.48797    (iter=50), x:1.48797    (iter=51), x:1.48797    (iter=52), x:1.48796
(iter=53), x:1.48796    (iter=54), x:1.48796    (iter=55), x:1.48796    (iter=56), x:1.48796
(iter=57), x:1.48796    (iter=58), x:1.48796    (iter=59), x:1.48796    (iter=60), x:1.48796
```

If x0 = 2.7, it will diverge.

```
>> q5_main
x0 = 2.70
(iter=01), x:2.72761    (iter=02), x:2.76553    (iter=03), x:2.81846    (iter=04), x:2.89405
(iter=05), x:3.00552    (iter=06), x:3.17780    (iter=07), x:3.46366    (iter=08), x:3.99585
(iter=09), x:5.21402    (iter=10), x:9.58730    (iter=11), x:85.37680    (iter=12), x:2448060658594177024.000
(iter=13), x:Inf    (iter=14), x:Inf    (iter=15), x:Inf    (iter=16), x:Inf
(iter=17), x:Inf    (iter=18), x:Inf    (iter=19), x:Inf    (iter=20), x:Inf
```

(c) $e^x = 2x^2$, perform ln to each side and we have: $x = \ln(2x^2)$. Therefore, we use $\ln(2x^2)$ as $g(x)$ and start with x = 2.5 and it converges to 2.61786.

This is implemented in ‘q5_3.m’.

```
>> q5_3
x0 = 2.50
(iter=01), x:2.52573   (iter=02), x:2.54621   (iter=03), x:2.56236   (iter=04), x:2.57500
(iter=05), x:2.58485   (iter=06), x:2.59248   (iter=07), x:2.59838   (iter=08), x:2.60292
(iter=09), x:2.60642   (iter=10), x:2.60910   (iter=11), x:2.61116   (iter=12), x:2.61273
(iter=13), x:2.61394   (iter=14), x:2.61487   (iter=15), x:2.61557   (iter=16), x:2.61611
(iter=17), x:2.61653   (iter=18), x:2.61684   (iter=19), x:2.61708   (iter=20), x:2.61727
(iter=21), x:2.61741   (iter=22), x:2.61752   (iter=23), x:2.61760   (iter=24), x:2.61766
(iter=25), x:2.61771   (iter=26), x:2.61775   (iter=27), x:2.61778   (iter=28), x:2.61780
(iter=29), x:2.61781   (iter=30), x:2.61783   (iter=31), x:2.61784   (iter=32), x:2.61784
(iter=33), x:2.61785   (iter=34), x:2.61785   (iter=35), x:2.61786   (iter=36), x:2.61786
(iter=37), x:2.61786   (iter=38), x:2.61786   (iter=39), x:2.61786   (iter=40), x:2.61786
```

6. $\begin{cases} y = cos^2(x) \\ x^2 + y^2 - x = 2 \end{cases}$, we can get $x^2 + cos^4 x - x - 2 = 0$

Using Newton’s method, the answer is (1.99076, 0.16624) and (-0.96442, 0.32478).

Define the ‘newton’ function in ‘newton.m’ and implement in ‘q6_main.m’.

- ‘N’ is the iteration count
- ‘X[]’ is an array that stores the x1 value of each iteration.
- We update x_1 as $x_0 - \frac{f(x_0)}{df(x_0)}$, where $df(x)$ is the first derivative of $f(x)$.
- Keep iterating until abs(f(x0)) > tolerance value.

q6_main.m   newton.m *

```
% newton method
function [N,X] = newton(f, df, x0 , tol)
N = 0;
X = [];
while( abs(f(x0)) > tol)
    N = N+1;
    x1 = x0 - f(x0)/df(x0);
    x0 = x1;
    X(N) = x0;
end
```

q6_main.m   newton.m

```
% Define function
f = @(x) x^2 + cos(x)^4 - x -2;
df = @(x) 2*x - 4*cos(x)^3*sin(x)-1;
tol = 1e-5;
x0 = -1;
fprintf('x0 = %.5f\n', x0)
[N,X] = newton(f, df, x0 , tol);
for i=1:N
    fprintf('(N=%d): x = %.5f\n', i, X(i))
    if i==N
        fprintf('find root at around: %.5f\n', X(i))
        fprintf('ANS: x=%.5f, y=%.5f\n', X(i),cos(X(i))^2)
    end
end
```

<table>
<tr>
<td>Setting x0 = 1

>> q6_main
x0 = 1.00000
(N=1): x = 5.08178
(N=2): x = 3.07306
(N=3): x = 2.08355
(N=4): x = 1.99537
(N=5): x = 1.99077
(N=6): x = 1.99076
find root at around: 1.99076
ANS: x=1.99076, y=0.16624</td>
<td>Setting x0 = -1

>> q6_main
x0 = -1.00000
(N=1): x = -0.96548
(N=2): x = -0.96442
find root at around: -0.96442
ANS: x=-0.96442, y=0.32478</td>
</tr>
</table>