Noe Lepez Da Silva Duarte

AE 4803 – Homework 2

All MATLAB code is provided in a separate zipped folder with the submission

1) a) Code converges after 10 iterations to Et <= 0.001, with the root as 0.8652

b) The approximate relative error for the root found at 0.8652 given Et <= 0.001 is 0.0023.

The function created for parts a & b is shown below.

```
function [x_r, iters, Ea] = bisection_true(func, bracket, error, x0)

iters = 0;
x_np = 0;
converged = false;

while not(converged)

    x_n = sum(bracket)/2;

    inter_n = func(x_n);
    inter_1 = func(bracket(1));

    Et = abs(x0-x_n);
    Ea = abs((x_n-x_np)/x_n);

    if Et <= error
        x_r = x_n;
        converged = true;
    elseif inter_n * inter_1<0
        bracket(2) = x_n;
    elseif inter_n * inter_1>0
        bracket(1) = x_n;
    end
    iters = iters +1;
    x_np = x_n;
end
end
```

The workspace code used for parts a & b is shown below

```
fnc = @(x) x^3 -exp(-x/2);
bracket = [0,2];
error = 0.001;
x0 = 0.866;
[ans, iterations, Ea] = bisection_true(fnc, bracket, error, x0)
```

c) Code converges after 13 iterations to ε_a <= 0.0023, with the root as 0.8625

Function and workspace code are shown below

```
function [x_r, iters] = false_position(func, bracket, error)

iters = 0;
x_np = 0;
converged = false;

while not(converged)

    f_u = func(bracket(2));
    f_l = func(bracket(1));
    x_n = (bracket(1) * f_u - bracket(2) * f_l)/(f_u-f_l);

    condition = f_l * func(x_n);

    Ea = abs((x_n-x_np)/x_n);

    if Ea <= error
        x_r = x_n;
        converged = true;
    elseif condition<0
        bracket(2) = x_n;
    elseif condition>0
        bracket(1) = x_n;
    end
    iters = iters +1;
    x_np = x_n;
end
end

% Q1_c
[ans, iterations] = false_position(fnc, bracket, Ea)
```

(Where Ea was taken from the previous workspace code)

d) Code converges after 6 iterations to $\varepsilon_a <= 0.0023$, with the root as 0.8656

Function and workspace code are shown below

```
function [x_n, iter] = Newt_Raph(x0, error, func, func_prime)

iter = 0;
x_p = x0;
Ea = 100;

while(Ea >= error)
       % Compute new solution.
       x_n = x_p - func(x_p)/func_prime(x_p);

       % Compute current solution tolerance.
       Ea = abs((x_n-x_p)/x_n);

       % Update Solutions
       x_p = x_n;
       iter = iter +1;
end
end
```

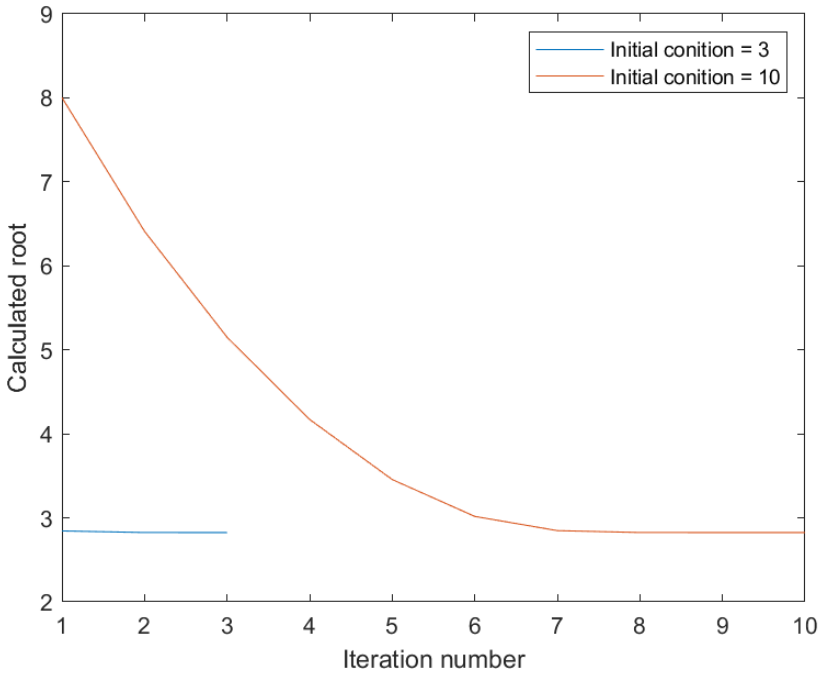
```
% Q1_d
fnc_p = @(x) 3*x^2 +0.5*exp(-x/2);
[ans, iterations] = Newt_Raph(0, Ea, fnc, fnc_p)
```

(Where Ea was taken from the previous workspace code)

e) From the number of iterations of each method, we can clearly see that the Newton-Raphson method was quickest to converge to an answer that fulfilled the termination criterion of $\varepsilon_a <= 0.0023$, with also having the closest result to the actual root. In addition, we can also see that in its blind search for the root, the bisection method ended up converging faster than the false position method.

Noe Lepez Da Silva Duarte

2) a) The fifth root of 180 is equal to 2.8252 with ε_a<=0.0001. Shown below is a graph of the convergence of the Newton-Raphson method at each iteration step. We can clearly see that the Newton-Raphson method converges much faster, after 3 iterations, when the initial condition is close to the actual root. By coincidence, it seems that the code converges in 3 iterations when the initial condition is 3 and in 10 iterations when the initial condition is 10.



Shown below is the workspace code created to solve the problem and graph the iterations.

```
%Q2_a
fnc = @(x) x^5 - 180;
fnc_p = @(x) 5*x^4;
init = 3 ;
Ea = 0.0001;
[sol_1, iterations_1] = Newt_Raph(init, Ea, fnc, fnc_p);
init = 10;
[sol_2, iterations_2] = Newt_Raph(init, Ea, fnc, fnc_p);

figure(1)
plot(linspace(1,length(iterations_1), length(iterations_1)), iterations_1)
hold on
plot(linspace(1,length(iterations_2), length(iterations_2)), iterations_2)
xlabel("Iteration number")
ylabel("Calculated root")
legend("Initial conition = 3", "Initial conition = 10")
saveas(figure(1),"Q2_a","png")
```

b) If the initial condition is set equal to 0, the Newton-Raphsen method fails as the derivative of x^5-180 is $5x^4$. As such, since the method uses the following equation to approximate the root, $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, where x_n is equal to the initial condition for the first iteration, the equation becomes $x_{n+1} = 0 - \frac{-180}{0}$, which is undefined. Therefore the code and method fails in this case.

3) I created the following code to solve the system of non-linear equations using Newton-Raphson. With the termination criterion of $\varepsilon_a <= 0.001$, the code was able to find the answer in 13 iterations with the initial condition of [100;6]. Should the correct answer be the initial condition, the code will run one iteration and output the correct answer of [3;4]. However, should one of the initial inputs be 0, the code will error and return Not A Number (NaN) as the answer. If the initial condition is [1;1], the code will find the solution in 9 iterations.

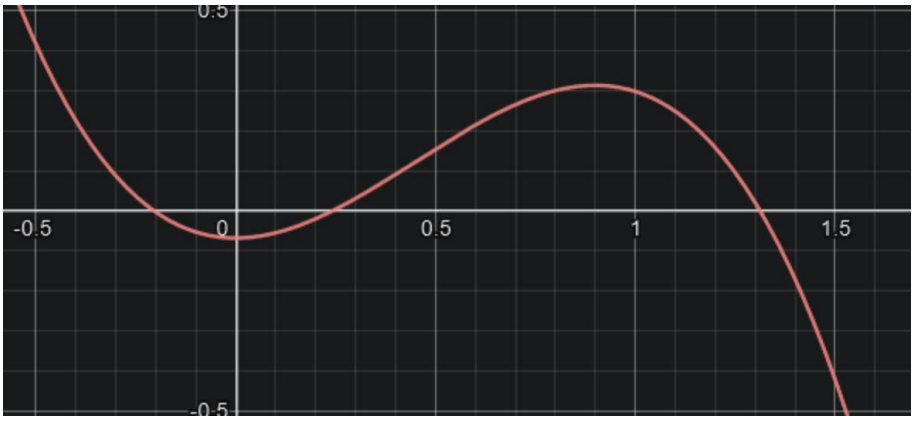
```
function [x_n, iter] = Newt_Raph_simu(x0, f1, f2, J1_dx, J1_dy, J2_dx, ...
    J2_dy, error)
syms x y
iter = 0;
x_p = x0;
Ea = 100;

while(Ea >= error)
       J10_dx = J1_dx(x_p(1));
       J10_dy = J1_dy(x_p(2));
       J20_dx = J2_dx(x_p(1));
       J20_dy = J2_dy(x_p(2));

       f1c = f1(x_p(1), x_p(2));
       f2c = f2(x_p(1), x_p(2));
       J0 = [J10_dx, J10_dy;J20_dx, J20_dy];
       x_n = -inv(J0)*[f1c;f2c];
       x_n = x_p+x_n;
       Ea = abs(norm(x_n)-norm(x_p))/norm(x_n);
       x_p = x_n;

       iter = iter + 1;
end
end
```

```
%Q3
initial_guess = [100;6];
error = 0.001;

syms x y
func_1 = @(x,y) 4*x^2 - y^3 + 28;
func_2 = @(x,y) 3*x^3 + 4*y^2 - 145;

func_1_dx = @(x) 8*x;
func_1_dy = @(y) -3*y^2;
func_2_dx = @(x) 9*x^2;
func_2_dy = @(y) 8*y;

Newt_Raph_simu(initial_guess,func_1, func_2, func_1_dx, func_1_dy, ...
    func_2_dx, func_2_dy, error)
```

4) We know that the general equation for the volume of water displaced is

$V_{slice} = \frac{1}{3}\pi d^2(3r - d)$. As such, in order to find d, we must know the volume of water to be displaced. This can be done using Archimedes' principle, which states that $F_b = W_{fluid} = \rho V_{displaced} g$. This can be rearranged as $V_{displaced} = \frac{F_b}{\rho g} = \frac{m \times g}{\rho g} = \frac{70}{1030}$.

We now know that $V_{slice} = \frac{7}{103} = \frac{1}{3}\pi d^2(1.35 - d)$. We can now find the root of the equation using the Newton-Raphson method by quickly rearranging the equation as $0 = \frac{1}{3}\pi d^2(1.35 - d) - \frac{7}{103}$. However, one must also notice that the equation has 3 roots, 2 of which will be incorrect for our case. Indeed, if the initial condition is set as 1, the code will yield a root at 1.3123m, which would imply that the ball if fully submerged and is incorrect. Similarly, if the root is set at -1, the code yields an answer of -0.2043m, implying the ball is out of the water which is impossible. Yet, if the initial condition is set at 0.1, We obtain a root at 0.2420m, which appears to be correct. The graph of $y = \frac{1}{3}\pi d^2(1.35 - d) - \frac{7}{103}$ is shown below, and shows why the code yields different answers depending on the initial condition, as 3 roots are present.



Noe Lepez Da Silva Duarte

Shown below is the code used to run my Newton-Raphson code.

```
% Q4
init = -1;
Ea = 0.001;
fnc = @(d) (1/3) * pi * d^2 *(1.35 - d) - (7/103);
fnc_p = @(d) -(pi*d*(10*d-9))/10;
[sol, iterations] = Newt_Raph(init, Ea, fnc, fnc_p);
figure(2)
plot(linspace(1,length(iterations), length(iterations)), iterations)
hold on
xlabel("Iteration number")
ylabel("Calculated root")
saveas(figure(2),"Q2_b","png")
```