Table of Contents


```
%Roshan Jaiswal-Ferri
%Section - 03
%Aero 300 Lab 3 - Iterative Methods to Solve Scalar Equations: 4/19/24
```

Workspace Prep

```
format long      %Allows for more accurate decimals
close all;       %Clears all
clear all;       %Clears Workspace
clc;             %Clears Command Window
```

PART 1: Defining the Function

```
fx = @(x) x^3+1.0142*x^2-19.3629*x+15.8398;
gx = @(x) 3*x^2+2.0284*x-19.3629;
hx = @(x) 6*x+2.0824;
p = [1 1.0142 -19.3629 15.8398];
r = roots(p);

d = 3; %Degree of polynomial
s = 1; %Step size
g = -6; %inital guess
tol = 1e-6;
```

PART 2: Using Bracketing Method

```
M = bracket(g,s,d,fx);
disp('Bracketing Method');
disp(M);
disp('')
```

PART 3: Using Bisection Method

```
timebi = 0;
for i = 1:10000
    tic
    [M1,eB,iB] = bisection(g,d,fx,tol);
    timebi = timebi + toc;
end
avgtimebi = timebi/10000;
disp('Bisection Method Result:');
disp(['Avg time to calculate: ', num2str(avgtimebi),'s'])
disp(M1);
disp('')
aeB = abs(((eB(:,2)-r(1,1))/r(1,1))*100); %absolute error Bisection
```

PART 4: Using Newton's Method

```
timeN = 0;
for i = 1:10000
    tic
    [M2,eN,iN] = newton(d,M,fx,gx,tol);
    timeN = timeN + toc;
end
avgtimeN = timeN/10000;
disp('Newtons Method Results:');
disp(['Avg time to calculate: ', num2str(avgtimeN),'s'])
disp(M2);
disp('');
aeN = abs(((eN(:,2)-r(1,1))/r(1,1))*100); %absolute error Newton
```

PART 5: Using Halley's Method

```
timeH = 0;
for i = 1:10000
    tic
    [M3,eH,iH] = halley(d,M,fx,gx,hx,tol);
    timeH = timeH + toc;
end
avgtimeH = timeH/10000;
disp('Halleys Method Results:');
disp(['Avg time to calculate: ', num2str(avgtimeH),'s'])
disp(M3);
disp('');
aeH = abs(((eH(:,2)-r(1,1))/r(1,1))*100); %absolute error Halley
```

PART 6: Plotting P(x)

```
x = linspace(-8,6,10000);
y = x.^3+1.0142*x.^2-19.3629*x+15.8398;
figure('name','P(x)')
plot(x,y);
```

```
xlabel('X')
ylabel('Y')
title('P(x)')
```

PART 7: Plotting absolute error

```
figure('name','Absolute error first root')
xlabel('Iteration');
ylabel('Absolute Error of first root');
title('Convergence of Each Method');
semilogy(iH,aeH, '*r')
grid on;
hold on;
semilogy(iN,aeN, '*g')
semilogy(iB,aeB, '*bl')
xlabel('Iteration #')
ylabel('Percent Error')
legend('Halley','Newton','Bisection')
```

Comments on Time and Slope

```
%TIME:
%All three methods are pretty quick, however the Newton and Halley methods
%are not only faster but much much more accurate. Theoretically halleys
%method should be faster than newtons method but probably because of the
%way I programmed each function or the brackets that I gave each function.
%But the halley and newton method are incredibly similar.

%SLOPE:
%Firstly the function semilogy() turns the y axis into a logarithmic scale
%while leaving the x axis alone. From this scale you can visually see how
%the slopes of Newtons and Halleys method are much steeper than that of the
%bisection. This is because the bisection method will close in linearly
%through its method of finding a midpoint every iteration, which is slow
%compared to Newtons and Halleys method where they use derivitives to point
%to the root and converge much quicker. The steeper a negative slope is the
%faster that method converges.
```

Halley's Method Function

```
%Halley Pseudocode:
%almost the same as newtons method except now also use the 2nd derivitive,
%it iterates until it reaches desired tolerance. It uses the derivitives to
%find roots by following the slope

function [M1,error,it] = halley(d, M, fx, gx, hx, tol)
    M1 = zeros(d,2); % stores the roots found
    for n = 1:d
        a = M(n,1); % defines where to place the roots
        b = M(n,2);
        xo = (a+b)/2;
        xoo = 0;
```

```
        count = 0;
        xO = 0;
        while abs(fx(xoo))>tol %check tol
            xoo = xO; %reset for iteration
            xO = xo - (2*fx(xo)*gx(xo))/((2*gx(xo)^2) - fx(xo)*hx(xo));
%formula
            xo = xO;
            count = count + 1;
            if n == 1
                error(count,1) = count;
                error(count,2) = xO;
                it(count,1) = count;
            end
        end
        M1(n) = xO; %save roots
        M1(n,2) = fx(xo);
    end
end
```

Newton's Method Function

```
%Newtons psuedocode:
%Use previous brackets to find general locations of all roots, then use
%dervitive to follow slope to x-axis and close in on root until the
%tolerance is met.

function [M1,error,it] = newton(d, M, fx, gx, tol)
    M1 = zeros(d,2); % stores the roots found
    for n = 1:d
        a = M(n,1); %finding roots
        b = M(n,2);
        xo = (a+b)/2;
        xoo = 0;
        count = 0;
        xO = 0;
        while abs(fx(xoo))>tol %checking tol
            xoo = xO; %reset for iteration
            xO = xo - fx(xo)/gx(xo); %Formula
            xo = xO;
            count = count + 1;
            if n == 1
                error(count,1) = count;
                error(count,2) = xO;
                it(count,1) = count;
            end
        end
        M1(n) = xO; %save roots
        M1(n,2) = fx(xo);
    end
end
```

Bisection Function

```
%Bisection Pseudocode:
%Step in big steps like the bracketing except when it finds a big step with
%a root it then starts closing in by finding out which side the root is
%closer to and then setting the respective endpoint to the midpoint and
%then restarts until the root is found at desired tolerance.

function [M,error,it] = bisection(g, d, fx, tol) %biteration, berror
    format long
    M = zeros(d,2);
    a = g;
    d1 = 1;
    while d1 ~= d + 1
        b = a + 1;
        ax = fx(a);
        bx = fx(b);
        b2 = b;
        a2 = a;
        if ax*bx<0
            c = (b2+a2)/2; %midpoint
            count = 0;
            while abs(fx(c)) > tol %checking tol
                ax2 = fx(a2); %new bounds
                bx2 = fx(b2);
                c = (b2+a2)/2; %new midpoint
                cx = fx(c);
                count = count + 1;
                if d1 == 1
                    error(count,1) = count; %Logging error and iteration for
first root only
                    error(count,2) = c;
                    it(count,1) = count;
                end
                if ax2*cx < 0 %Checking for root
                    b2 = c;
                elseif bx2*cx < 0
                    a2 = c;
                else
                    disp('error')
                end
            end
            M(d1,1) = c;
            M(d1,2) = fx(c);
            a = b;
            d1 = d1 + 1;
        else
            a = b;
        end
    end
end
```

Bracketing Function

```
function [M] = bracket(g, s, d, fx) %g is initial guess, s is step, d is #
roots, fx is function
    format long
    M = zeros(d,2);
    lBracket = g;
    d1 = 1;

    while d1 ~= d+1
        hBracket = lBracket+s;
        h = fx(hBracket);
        l = fx(lBracket);
        done = 0;
        if h*l<0 %if it is <0 then ans is negative and there is root in
bracket
            while done == 0
                hBracket2 = lBracket + 1e-1; %Smaller stepping to proper
tol !!MAKE THE NUMBER ONE WHATEVER YOU WANT TOLERANCE TO BE!!
                h = fx(hBracket2);
                l = fx(lBracket);
                if h*l<0 %Checking for root
                    M(d1,1) = lBracket;
                    M(d1,2) = hBracket2;
                    lBracket = hBracket2;
                    done = 1;
                    d1 = d1 + 1;
                else
                    lBracket = hBracket2;
                end
            end
        elseif h*l == 0
            if h == 0
                M(d,1) = h;
                M(d,2) = h;
            elseif l == 0
                M(d,1) = l;
                M(d,2) = l;
            else
                disp('error 0');
            end
        else
            lBracket = hBracket;
        end

    end

end
```

Published with MATLAB® R2023b