Table of Contents


```
clear
tic
p = [ 1 , 1.0142 , -19.3629 , 15.8398 ];
pf = p2d(p);
f = @(x) pf(1,1)*x.^3 + pf(1,2)*x.^2 + pf(1,3)*x + pf(1,4) ;
f1 = @(x) pf(2,1)*x^3 + pf(2,2)*x^2 + pf(1,3)*x + pf(2,4) ;
f2 = @(x) pf(3,1)*x^3 + pf(3,2)*x^2 + pf(3,3)*x + pf(3,4) ;

TOL = .000001 ; %Tolerance
```

Bracketing

```
%Inputs for function
n = 3;
h = .01 ;
g = [ -6 , 0 , 3 ] ;

%Runs function to create brackets around roots
bracket = bracketing( f , n , g , h );

%Creates starting guesses for the following functions by using average
 of
%each bracket
for ii = 1:n
    g(ii) = ( bracket( ii , 1 ) + bracket( ii , 2 ) ) / 2 ;
end
```

Bisection

```
tic
%Inputs
a = bracket( : , 1 ) ; %lower bound
b = bracket( : , 2 ) ; %upper bound

%Finds roots by bisection 10000 times
for bb = 1:10000
    rootB = bisecting( a , b , TOL , f ) ;
```

```
end
t_B = toc ;
disp( [ 'The time to solve for all roots with bisection method is ' ,
 num2str(t_B/10000) ] )
```

The time to solve for all roots with bisection method is 0.00010364

Newton's

```
tic
gn = g ; %Guess

%Finds roots by newtons method 10000 times
for cc = 1:10000
    rootN = newton( gn , TOL , f , f1 );
end
t_N = toc ;
disp( [ 'The time to solve for all roots with Newtons method is ' ,
 num2str(t_N/10000) ] )
```

The time to solve for all roots with Newtons method is 0.00022959

Halley's

```
tic

%Finds roots by halleys method 10000 times
for dd = 1:10000
    [rootH] = halley( gn , TOL , f , f1 , f2);
end
t_H = toc ;
disp( [ 'The time to solve for all roots with Halleys method is ' ,
 num2str(t_H/10000) ] )
```

The time to solve for all roots with Halleys method is 0.00039048

All the roots

```
%inputs
start = -100 ;
n = 3 ;
tic

%Finds all roots by my method 10000 times
for ee = 1:10000
    [rootL] = allroot( p , TOL , start , n );
end
t_mine = toc ;
disp( [ 'The time to solve for all the roots with my method is ' ,
 num2str(t_mine/10000) ] )

%Discussion of results
```

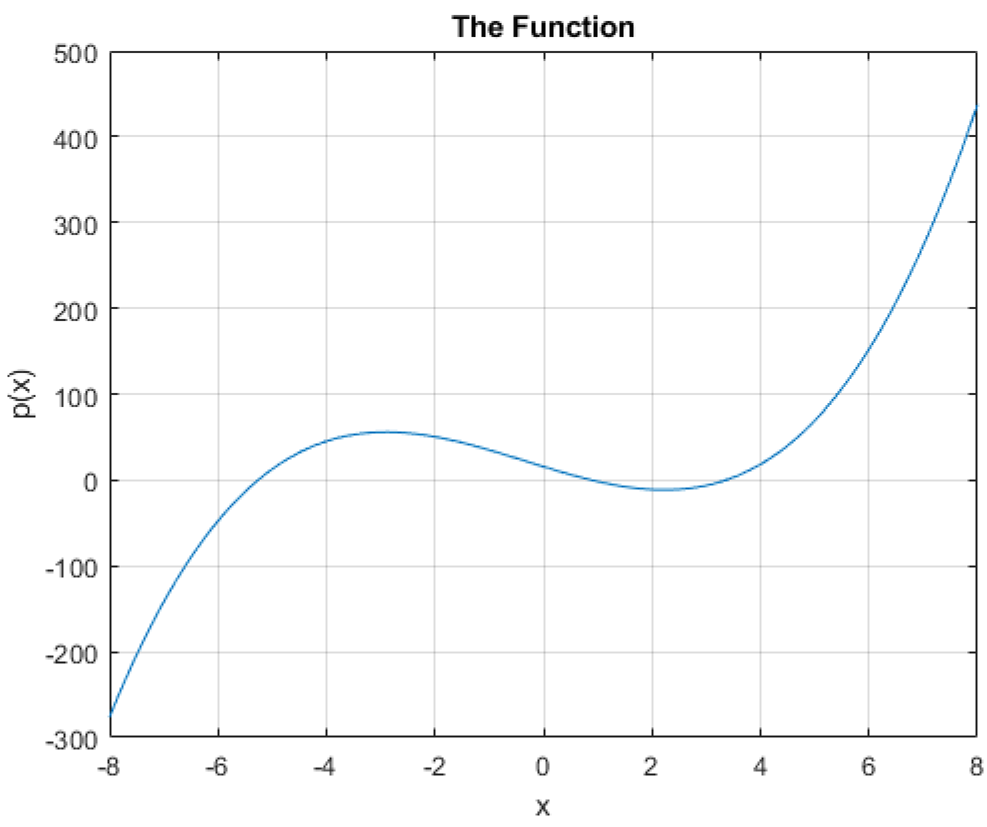
```
disp( 'Bisection was fastest so my function to find roots is based of
 bisection. It is slower ' )
disp( 'because it includes a bracketing algorithm as well as creating
 a function from inputted ' )
disp( 'coefficients. Newtons method was much slower with Halleys a
 ways behind that.' )
```

The time to solve for all the roots with my method is 0.00013317
Bisection was fastest so my function to find roots is based of bisection. It is slower
because it includes a bracketing algorithm as well as creating a function from inputted
coefficients. Newtons method was much slower with Halleys a ways behind that.

Plots

```
jj = linspace( -8 , 8 , 1000 ); %The plot will be from -8 to 8 with
 1000 individual points

plot( jj , f(jj) ) %Plots p(x)
title( 'The Function' )
xlabel( 'x' )
ylabel( 'p(x)' )
grid on
hold off
```



Functions

```
function [bracket] = bracketing( f , n , g , h )
%Creates brackets around the roots of a function f, with n roots, from
 an
%intitial guess g. The brackets will be of size h
% i := 0
% a0 := initial guess
% while sign(f(ai)) =/= sign(f(a_i+1))
%       a_i+1 := a_i + h
%       i := i + 1
% end
% a:a_i ;
% b = a_i+1 ;


bracket = zeros(n,2) ; %preallocates

for jj = 1:n %repeats for assumed number of roots
    ii = 1 ;
    a(1) = g(jj) ; %first boundary is the initial guess
    while sign( f( a(ii) ) ) == sign( f( a(ii) - h ) ) %continues as
 long as each bracket is on
        a(ii+1) = a(ii) + h ;
        ii = ii + 1 ;
    end

    bracket( jj , 1 ) = a( ii - 1 ) ;
    bracket( jj , 2 ) = a( ii ) ;
end

end


function [ pf ] = p2d( p )
%Uses a given vector to create a polynomial of any size then finds all
%non-zero vectors
    lp = length(p); %saves the length of the input vector
    pf = [ p ; zeros( lp - 1 , lp )  ] ; %allocates a space for the
 derivatives below the original vector


    %Finds the coeffecient of each derivative. Each row down is
 another
    %derivative
    for ii = 2:lp %only adding information to the second row and below
        p = polyder( p ) ; %Takes derivative of last row
        for jj = 0:( length(p) - 1 ) %fills the values of p into the
 main matrix from right to left
        pf( ii , lp - jj ) = p( length(p) - jj ) ;
        end
    end
end
```

```
function rootB = bisecting( a , b , TOL , f )
% given [ a b ] such that f(a)*f(b) < 0 will find the roots of
% while ( b - a ) / 2 > TOL
    % c = ( b + a ) / 2
    % if f(c) is 0
        %stop
    % end
    % if sign of f(c)*f(b) is positive ;
        % b = c
    % else
        % a = c
    % end
% end
for ii = 1:length(a)
    while ( b(ii) - a(ii) ) / 2 > TOL %Continue running as long as
 half the difference between b and a is greater than the tolerance
        c(ii) = ( b(ii) + a(ii) ) / 2 ; % c is halway between a and b
        if f(c(ii)) == 0 %if c is the root the function ends
            stop
        end
        if f(c(ii))*f(b(ii)) > 0 %if the f(c) and f(b) are on the same
 side of the x axis
            b(ii) = c(ii) ; % new b is now c
        else %otherwise
            a(ii) = c(ii) ; % the lower bound is now c
        end
    end
    rootB(ii) = c(ii) ;
end
end

function [rootN] = newton( gn , TOL , f , f1)
%UNTITLED7 Summary of this function goes here
%   Detailed explanation goes here
x = gn ;
ii = 1 ;
for jj = 1:length(x)
    while abs( 0 - f( x(ii,jj) ) ) > TOL
        x( ii+1 , jj ) = x( ii , jj ) - ( f ( x( ii , jj ) ) /
 f1( x( ii , jj )) ) ;
        ii = ii + 1 ;
    end
    rootN(jj) = x(ii,jj) ;
end
end


function [rootH] = halley( gn , TOL , f , f1 , f2)
%UNTITLED7 Summary of this function goes here
%   Detailed explanation goes here
x = gn ;
ii = 1 ;
for jj = 1:length(x)
```

```
    while abs( 0 - f( x(ii,jj) ) ) > TOL
        x( ii+1 , jj ) = x( ii , jj ) - (2*f(x(ii,jj))*f1(x(ii,jj)))/
(2*(f1(x(ii,jj)))^2-f(x(ii,jj))*f2(x(ii,jj))) ;
        ii = ii + 1 ;
    end
    rootH(jj) = x(ii,jj) ;
end
end




function rootL = allroot( p , TOL , start , n )
%Finds all roots greater than starting value
%Step with increasing size until finding bracket around root then
 bisects to find root
    %each iteration without finding the root the step size increases
    %
rootL = zeros( 1 , n ) ;
ii = 1 ;
jj = 1 ;
atr = start ;
ss = size( atr );
while ss(2) < n %repeats if less roots than assumed
    h = .01 ; %initial step size

    %adds a column to fill in every time it repeats
    if ii > 1
        if ii <= n
            atr = [ atr , zeros(ss(1),1) ] ;
        end
    end

    %starts the next iteration of root where the last root was found
    if ii > 1
        atr(1,ii) = atr( jj , ii - 1 ) ;
    end

    jj = 1 ;
        %Bracketing but with increasing step size
        while sign( fun( p , atr(jj,ii) ) ) == sign( fun( p ,
 atr(jj,ii) - h ) ) %Stops when values are found to be on either side
 of a root
            ss = size(atr) ;

            if  jj + 1 == ss(1)
                atr = [ atr ; zeros( 1 , ss(2) ) ] ; %adds a row if
 there is not one
            end

            atr(jj + 1,ii) = atr(jj,ii) + h ; %Increase boundary by
 step size
            jj = jj + 1 ;
            h = h + .1 ;
```

```
        end

        if fun( p , atr( jj , ii ) ) == 0 %stop if root is already
 found

        else
            b = atr( jj , ii ) ;
            a = atr( jj - 1 , ii ) ;

            while ( b - a ) / 2 > TOL %Continue running as long as
 half the difference between b and a is greater than the tolerance
                c = ( b + a ) / 2 ; % c is halway between a and b

                if fun(p,c) == 0 %if c is the root the function ends
                    stop
                end

                if fun(p,c)*fun(p,b) > 0 %if the f(c) and f(b) are on
 the same side of the x axis
                    b = c ; % new b is now c
                else %otherwise
                    a = c ; % the lower bound is now c
                end

            end

            rootL(ii) = c ;


        end


ii = ii + 1 ;
ss = size( atr );
end
end



function [f] = fun(p,x)
%turns p vector of coefecients into function
f = 0;
    lop = length(p) ;
    for kk = 1:lop
    f = f + x^(lop-kk) * p( kk ) ;
    end

end
```

Published with MATLAB® R2017b