```
%created by Jason Sayre on 02/22/2018
clear %housekeeping
clc
pt = 3 %variables
k = 0.05
func = @(x) (((x)./(1-x)).*(sqrt((2.*pt)./(2+x))))-k %function we want
 to approximate the root location of
fplot(func,'g-') %graphs function
ylim([-10 10]) %adds windows/limits for x and y axis
xlim([-10 10])
refline(0) %graphical elements
xroot = fzero(func,[0 0.4]) %use of a function to approximate the root
hold on %keeps graph elements
plot(xroot,0,'bd') %adds diamond on the root location

[root, fx, ea, iter] = falsePosition (func, 0, 0.4)
%being that this function has a point where it diverges to infinity,
 it can
%be assumed that a closed bracket method would be more accurate

function [root, fx, ea, iter] = falsePosition (func, xl ,xu, es,
 maxiter) %bracket false position method
%This function estimates the root of a given function
%Inputs:
%   func - the function being evaluated. Please enter function in the
%   format of "@(x)" followed by the desired function. Example: '@(x)
%   x.^3+5*x-4 '
%   xl - the lower guess
%   xu - the upper guess
%   es - the desired relative error as a (default = 0.0001%)
%   maxiter - the number of iterations desired (default = 200)
%Outputs:
%   root - the estimated root location
%   fx - the function evaluated at the root location
%   ea - the approximate relative error as a percent
%   iter - how many iterations were performed
%Program created by Jason Sayre on 02/17/18
%Last edit on 01/17/18

%For testing purposes, un-comment while in use, comment while not in
 use
%func = @(x) x.^3+4*x-1    root is ~0.24626
%func = @(x) x.^2+2*x-1    root is ~0.4145
%func = @(x) x.^2          root is 0.00
%xl = -3
%xu = 3
%es = 0.1
%maxiter= 200

%input arguments for testing%
%func = input('enter the function to be evaluated: ')
%xl = input('the lower guess: ')
```

```
%xu = input('the upper guess: ')
%es = input('the desired relative error: ')
%maxiter = input('the max iterations: ')


%x= sym('x'); %for testing purpose, i was unable to figure out how to
 use
%this notation and instead starting using the notation:
%x = @(x) ...
%source for where i learned how to do this notation: https://
www.mathworks.com/matlabcentral/answers/63664-evaluate-f-x-for-
multiple-evenly-spaced-values

clc %housekeeping, clears command window. Cannot have clear otherwise
 the input variables will be cleared immediately after being entered.


if nargin < 3 %sets minimum input arguments as 3
    warning('the requested function does not have enough info to
 properly calculate the root of the function')
end

if nargin == 3 %sets 4th and 5th arg as a default value
    es = 0.0001;
    maxiter = 200;
end

if nargin == 4 %sets 5th arg as default
    maxiter = 200;
end

if nargin > 5 %limits inputs to 5 arguments
    warning('You have not entered the proper amount of input
 arguments')
end



fxu = func(xu); %intent is to check sign in next step
fxl = func(xl);
if fxu*fxl > 0  %checks sign to determine if there is a root in
 between the bounds entered
    warning('if printed above is root = 0, the root is zero. Otherwise
 there is either multiple roots or no root between the upper and lower
 bound estimates entered. Try reentering the bounds. If this error
 continues to occur, try graphing your function and re-choosing bounds
 ')
end

iter=1;%starting interation
ea = 1; %100 percent error to start
rootstorage = ones(1,200); %starting vector values
```

```
fxzero = func(0); %tests if the root is at the value of zero
if fxzero == 0 %will display root is zero and will then stop program
 if it is true.
    root = [fxzero]
    fx = [func]
    ea = [0]
    iter = ('the root was found without running the program through
 iterations. ')

else while ea >= es %runs while calculated error is greater than
 desired error
xu = xu; %upper x value
xl = xl; %lower x value
fxu = func(xu); %calculated upper y value
fxl = func(xl); %calculated lower y value

xr = (xu-((fxu*(xl-xu))/ (fxl - fxu))); %plugs values in to find the
 functions calculated root
rootstorage(iter) = xr;
if (fxl * fxu < 0) %tests which side to redefine the bracket as
    xu = xr;
else
    xl = xr;
end

if xr == rootstorage(1)%considers that the first time through wont
 have 2 values to calculate error, sets default error as 100 percent
    ea = 1;
end
if xr~= rootstorage(1)
  %xr ~= rootstorage(1)
ea = ((((rootstorage(iter) - rootstorage((old)))/
rootstorage(iter)))*100); %calculated percent error from the created
 vector
ea = abs(ea);
end




old = iter; %stores previous iter as old, updates each loop
iter = iter +1; %counts iterations and stores most recent iter as iter


if iter == maxiter
    break
end

end

[root] = xr %returns output variables in command window
[fx] = func(xr) %evaluated function at root
[ea] = ea %error
```

```
[iter] = iter %# of iterations
end
end
```

```
pt =

     3
```

```
k =

    0.0500
```

```
func =

  function_handle with value:

    @(x)(((x)./(1-x)).*(sqrt((2.*pt)./(2+x))))-k
```

```
xroot =

    0.0282
```

```
root =

    0.0282
```

```
fx =

  -7.0506e-14
```

```
ea =

   8.5394e-07
```

```
iter =

     6
```

```
root =

    0.0282
```

```
fx =
```

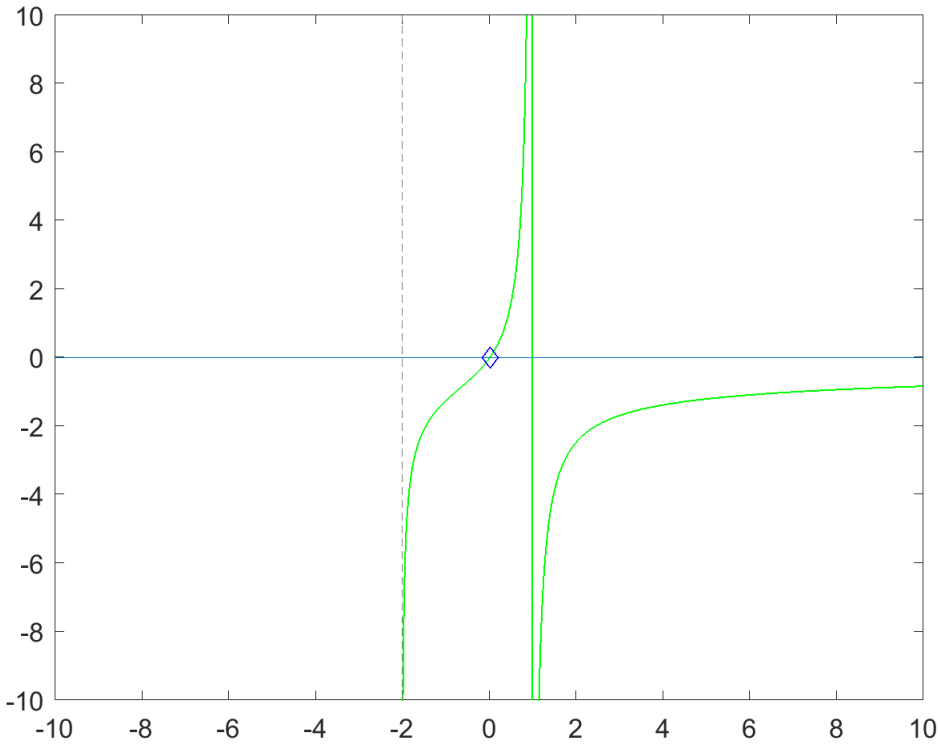
```
   -7.0506e-14

ea =

   8.5394e-07

iter =

     6
```



Published with MATLAB® R2017a