

# ENGRD 3200 HW3

Shiyao Sun, Samuel Tome  
Group 34

February 28, 2016

## Problem 1

A

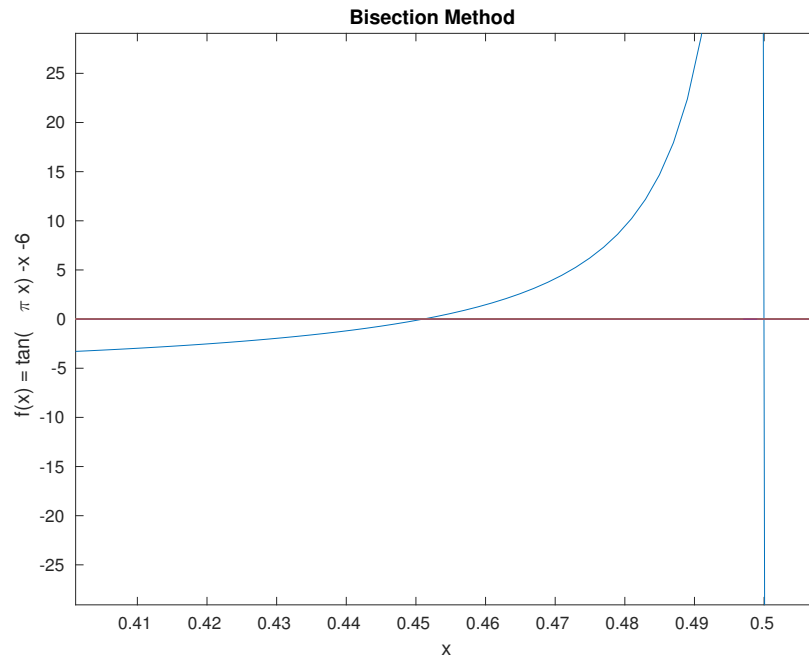


Figure 1: Detail of the first positive root of the function  $f(x) = \tan(\pi x) - x - 6$ .

The interval from  $[0.40, 0.48]$  is a good choice for the first bracketing interval since it encloses a single sign change in the function.

The first iteration of the bisection method evaluated  $f(x)$  at the lower and upper bounds of the interval and the point halfway between the two. It looked at the sign at these three points and determined between which two it changed. It then used these two points, which were the midpoint and the upper bound, as the lower and upper bounds for the second iteration.

Iteration	$x_l$	$x_u$	$x_r$
1	0.40	0.48	0.44
2	0.44	0.48	0.46

The approximate relative error for the second iteration is given by the change in  $x_r$ ,

$$\frac{|0.46 - 0.44|}{0.46} * 100\% = 4.3\%$$

The following Matlab program, modified from textbook Figure 5.7, was used to find the first root of the above function and find number of iterations necessary to reach an acceptable tolerance.

```

1 function [root,fx,ea,iter,cell]=bisect(func,xl,xu,es,maxit,varargin)
2 % bisect: root location zeroes
3 % [root,fx,ea,iter]=bisect(func,xl,xu,es,maxit,p1,p2,...):
4 % uses bisection method to find the root of func
5 % input:
6 % func = name of function
7 % xl, xu = lower and upper guesses
8 % es = desired relative error (default = 0.0001%)
9 % maxit = maximum allowable iterations (default = 50)
10 % p1,p2,... = additional parameters used by func
11 % output:
12 % root = real root
13 % fx = function value at root
14 % ea = approximate relative error (%)
15 % iter = number of iterations
16 % cell = table of bounds and errors
17
18 if nargin<3,error('at least 3 input arguments required'),end
19 test = func(xl,varargin{:})*func(xu,varargin{:});
20 if test>0,error('no sign change'),end
21 if nargin<4 || isempty(es), es=0.0001;end
22 if nargin<5 || isempty(maxit), maxit=50;end
23 iter = 0; xr = xl; ea = 100;
24
25 cell = {'Iteration','Lower bound','Upper bound','Midpoint',...
26 'Approx error','f(x-r)'};
27
28 while (1)
29     xrold = xr;
30     xr = (xl + xu)/2;
31     iter = iter + 1;
32     cell{iter+1,1} = iter;
33     cell{iter+1,2} = xl;
34     cell{iter+1,3} = xu;
35     cell{iter+1,4} = xr;
36     cell{iter+1,5} = ea;
37     cell{iter+1,6} = test;
38
39     if xr ~= 0,ea = abs((xr - xrold)/xr) * 100;end
40     test = func(xl,varargin{:})*func(xr,varargin{:});
41     if test < 0
42         xu = xr;
43     elseif test > 0
44         xl = xr;
45     else
46         ea = 0;
47     end
48
49     if ea <= es || iter >= maxit,break,end
50 end
51
52 root = xr; fx = func(xr, varargin{:});

```

which outputs:

	'Iteration'	'Lower bound'	'Upper bound'	'Midpoint'	'Approx error'	'f(x_r)'						
1	[	1]	[	0.4000]	[	0.4800]	[	0.4400]	[	100]	[	-31.2781]
3	[	2]	[	0.4400]	[	0.4800]	[	0.4600]	[	9.0909]	[	3.9795]

4	[	3]	[	0.4400]	[	0.4600]	[	0.4500]	[	4.3478]	[	-1.7438]
5	[	4]	[	0.4500]	[	0.4600]	[	0.4550]	[	2.2222]	[	0.1632]
6	[	5]	[	0.4500]	[	0.4550]	[	0.4525]	[	1.0989]	[	-0.0778]
7	[	6]	[	0.4500]	[	0.4525]	[	0.4512]	[	0.5525]	[	-0.0271]
8	[	7]	[	0.4500]	[	0.4512]	[	0.4506]	[	0.2770]	[	-0.0037]
9	[	8]	[	0.4506]	[	0.4512]	[	0.4509]	[	0.1387]	[	0.0076]
10	[	9]	[	0.4509]	[	0.4512]	[	0.4511]	[	0.0693]	[	8.0956e-04]
11	[	10]	[	0.4509]	[	0.4511]	[	0.4510]	[	0.0346]	[	-8.9988e-05]
12	[	11]	[	0.4510]	[	0.4511]	[	0.4511]	[	0.0173]	[	6.1132e-05]
13	[	12]	[	0.4510]	[	0.4511]	[	0.4510]	[	0.0087]	[	-4.1475e-06]
14	[	13]	[	0.4510]	[	0.4511]	[	0.4510]	[	0.0043]	[	6.7542e-06]
15	[	14]	[	0.4510]	[	0.4511]	[	0.4510]	[	0.0022]	[	4.9927e-07]
16	[	15]	[	0.4510]	[	0.4510]	[	0.4510]	[	0.0011]	[	-1.0506e-07]

Because the bisection method must evaluate a function three times per iteration (once per endpoint plus the midpoint) and it took 15 iterations to reach the desired tolerance, this method had to evaluate the function a total of 45 times.

## Problem 2

### A

Textbook Figure 6.7 was modified to evaluate the Newton-Raphson method while also outputting information for each iteration:

```

1 function [root,ea,iter,cell]=newtraph(func,dfunc,xr,es,maxit,varargin)
2 % newtraph: Newton-Raphson root location zeroes
3 % [root,ea,iter]=newtraph(func,dfunc,xr,es,maxit,p1,p2,...):
4 % uses Newton-Raphson method to find the root of func
5 % input:
6 % func = name of function
7 % dfunc = name of derivative of function
8 % xr = initial guess
9 % es = desired relative error (default = 0.0001%)
10 % maxit = maximum allowable iterations (default = 50)
11 % p1,p2,... = additional parameters used by function
12 % output:
13 % root = real root
14 % ea = approximate relative error (%)
15 % iter = number of iterations
16
17 if nargin < 3,error('at least 3 input arguments required'),end
18 if nargin < 4 || isempty(es),es=0.0001;end
19 if nargin < 5 || isempty(maxit),maxit=50;end
20
21 cell = {'Iteration','Previous estimate','New estimate','Percent approx error'};
22
23 iter = 0;
24
25 while (1)
26     xrold = xr;
27     cell{iter+2,2} = xr;
28     xr = xr - func(xr)/dfunc(xr);
29     iter = iter + 1;
30
31     cell{iter+1,1} = iter;
32     cell{iter+1,3} = xr;
33
34     if xr ~= 0
35         ea = abs((xr - xrold)/xr) * 100;
36         cell{iter+1,4} = ea;
37     end
38     if ea <= es || iter >= maxit, break, end

```

```

39 end
40 root = xr;

```

which outputs the following table:

	'Iteration'	'Previous estimate'	'New estimate'	'Percent approx error'
2	[ 1]	[ 0.4800]	[ 0.4682]	[ 2.5268]
3	[ 2]	[ 0.4682]	[ 0.4570]	[ 2.4363]
4	[ 3]	[ 0.4570]	[ 0.4518]	[ 1.1634]
5	[ 4]	[ 0.4518]	[ 0.4511]	[ 0.1599]
6	[ 5]	[ 0.4511]	[ 0.4510]	[ 0.0024]
7	[ 6]	[ 0.4510]	[ 0.4510]	[ 5.4277e-07]

## B

Example code from MathWorks at <http://www.mathworks.com/matlabcentral/fileexchange/36737-secant-method/content/secant.m> was modified to produce the secant method while also outputting information for each iteration:

```

1 % It will take function and initial value as the input of function.
2 % a, b are two initial guesses
3 % maxerr is the acceptable approximate relative error
4 % The function returns y and cell, where y is the root to the function
5 % cell is a cell array which stores the number of iteration, previous
6 % estimate, new estimate and relative error
7
8 function [y,cell] = secant(f,a,b,maxerr)
9 c = (a*f(b) - b*f(a))/(f(b) - f(a));
10 iter = 1;
11 cell = {'iteration','Previous estimate','New estimate','relative error'};
12 while abs(f(c)) > maxerr
13     a = b;
14     b = c;
15     c = (a*f(b) - b*f(a))/(f(b) - f(a));
16     cell{iter+1,1} = iter;
17     cell{iter+1,2} = a;
18     cell{iter+1,3} = b;
19     cell{iter+1,4} = abs(f(c));
20     iter = iter + 1;
21     if(iter == 25)
22         break;
23     end
24 end
25 y = c;

```

which gives

	'Iteration'	'Previous estimate'	'New estimate'	'Relative error'
2	[ 1]	[ 0.4800]	[ 0.5037]	[ 11.3492]
3	[ 2]	[ 0.5037]	[ 0.4822]	[ 14.0317]
4	[ 3]	[ 0.4822]	[ 0.4845]	[ 4.9946]
5	[ 4]	[ 0.4845]	[ 0.4723]	[ 2.7451]
6	[ 5]	[ 0.4723]	[ 0.4656]	[ 0.9625]
7	[ 6]	[ 0.4656]	[ 0.4574]	[ 0.2587]
8	[ 7]	[ 0.4574]	[ 0.4529]	[ 0.0322]
9	[ 8]	[ 0.4529]	[ 0.4513]	[ 0.0012]
10	[ 9]	[ 0.4513]	[ 0.4511]	[ 6.0377e-06]

C

D

The following script was used to find the first root of the equation  $f(x) = \tan(\pi x) - x - 6$  using all three methods to within an approximate relative error of  $10^{-6}$  for the Newton-Raphson and secant methods and  $10^{-5}$  for the bisection method. The script then plots the approximate relative error as a function of iteration number.

```
1 % Gathers together tables of iterations and relative error
2 % Plots these
3
4 % Create function and derivative
5
6 fx = @(x) tan(pi*x) -x -6;
7 df = @(x) pi*(sec(pi*x))^2 -1;
8
9 % Get bisection data
10 [mass blah ea iter bsect]=bisection(fx,.4,.48,.001,100);
11 bsect1 = cell2mat(bsect(2:16,:)); %Convert to numeric values
12
13 % Get Newton-Raphson data
14 [root,ea,iter,nr] = newtraph(fx,df,0.48,.0001,25);
15 nr1 = cell2mat(nr(2:7,:)); %Convert to numeric values
16
17 % Get secant data
18 [y,sc] = secant(fx,0.54,.48,.0001);
19 sc1 = cell2mat(sc(2:10,:)); %Convert to numeric values
20
21 close all
22
23 figure(1)
24 semilogy(bsect1(1:15,1),bsect1(1:15,5),'-o')
25 hold on
26 plot(nr1(1:6,1),nr1(1:6,4),'-o')
27 plot(sc1(1:9,1),sc1(1:9,4),'-o')
28 legend('Bisect','Newton-Raphson','Secant')
29 xlabel('Iteration')
30 ylabel('log_{10}Error')
31 hold off
```

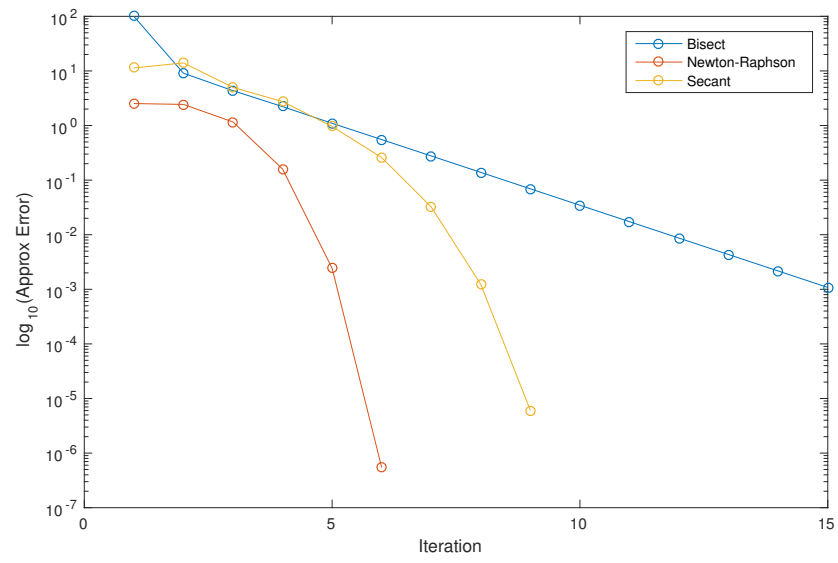


Figure 2: Approximate relative error as a function of iteration number