CISC271
Fall 2006
Homework for week 2
in preparation for quiz 1
Solutions

This homework will give you some practice with using Matlab, programming Matlab functions, and root finding algorithms.

1. Consider the following Matlab code for a function that implements the bisection algorithm for finding the zeroes of a function.

```
function x = bisectDR(fun,a,b);
% bisect  Use bisection to find a root of the scalar equation f(x) = 0
%
% Synopsis:  x = bisectDR(fun,a,b)
%
%
% Input: fun     = (string) name of function for which roots are sought
%        a,b     =  xleft = a, xright = b
%

if a > b
    temp = a
    a = b
    b = temp
end

fa = feval(fun,a); fb = feval(fun,b);
if sign(fa) == sign(fb)
    error(sprintf( 'error signs of f(a) and f(b) are the same'))
end
```

```
it = 0; % Count the number of iterations
while abs(b-a) > eps*abs(b)
        it = it + 1;
        x = a + 0.5*(b-a);
        fx = feval(fun,x);
        if fx == 0
            break
        end
        if sign(fa) == sign(fx)
            a = x; fa = fx;
        else
            b = x; fb = fx;
        end
end
%Print the number of iterations and final values for a and b
it
a
b
```

The function f1 below has zeros that are the square root of 2.

```
function f = f1(x)
f =  x^2 - 2;
```

To try this out yourselves copy the bisection algorithm to a file called "bisectDR.m", and copy f1 to a separate file called "f1.m". Now run the bisection algorithm from the command line by typing: " r = bisectDR('f1', 1,2)".

To make sure that Matlab knows where to find your file use the path command, "help path" will explain how this works. For example if the directory you are using is called /Users/myname/Programming/MyMatlab/ then type

"path(path, ' /Users/myname/Programming/MyMatlab/')". If you want Matlab to remember this directry, type "savepath".

This should give you an estimate of the sqrt(2) accurate to within eps in 52 iterations. You can see this explicitly if you type "format hex" and then run " r = bisectDR('f1', 1,2)". If you examine the hex values the bits used by a and b you see that there is only a one bit difference between the two values. Now try " r = bisectDR('f1', 1,10^{10})".

Do you get the same answer? How many iterations were used? Can you explain this phenomenon?

Solution: Note: I have edited some Matlab output to make it prettier.

```
> bisectDR('f1',1,2);
it = 52
a = 1.4142
b = 1.4142
```

Displaying a and b in hex clearly shows how the two values differ in a single bit.

```
a = 3ff6a09e667f3bcc
b = 3ff6a09e667f3bcd
```

Going back to format short and:

```
> bisectDR('f1',1,10^10)
it =  85
a = 1.4142
b = 1.4142
```

It took 33 extra iterations because we had to wittle down the value of the exponent. It's no coincidence that $10^{10} \approx 2^{33}$.

2. We saw three algorithms for root finding bisection, secant and Newton's algorithm. Consider the function $x^2 - 2x - 3$. Each of the algorithms can be viewed as having a principle step that can be characterized as:

   newvalue = function of one or two previous values.

   Assume that bisect and secant are given starting values 1,5 and Newton's method is given the starting value 1. Using hand calculations what is the first new value for each algorithm.

   Solution:

   (a) Bisection
   newx = (a + b) / 2 or newx = 3

   (b) Secant
   dx = (f(b) - f(a)) / (b - a) = (12 + 4)/ 4 = 4
   newx = b - f(b)/dx = 5 - 12/4 = 2

   (c) Newton
   $f'(x)$ = 2x -2
   newx = x - f(x) / f'(x) = 1 + 4/0 OVERFLOW

3. The secant method has more or less the same advantages and disadvantages as Newton's algorithm for finding a root of a function f with one exception. We don't need to know the first derivative of f.

   Implement the secant method in Matlab.

   Now compare the performance of your secant method with that of the bisection method in terms of the number of iterations used. Do this by determining solutions to the following functions. One way to organize your results is to fill the following table.

   Solution: I started with the solution in Recktenwald. Note, how I changed the tolerance checks and test against eps*(abs(b)).

```
function x = secantDR(fun,a,b, verbose)
% Taken from code provided by Recktenwald NMM, modified by DR.
% secant  Secant method for finding roots of scalar f(x) = 0
%
% Synopsis:  x = secant(fun,a,b)
%
%
%  Input:  fun  = (string) name of function for which roots are sought
%          a,b  = endpoints of initial bracket interval
%          verbose = 1 add additional print statements
%
%
%  Output:  x = the root of the function
if nargin <  4 verbose = 0; end

if verbose
  fprintf('\nSecant iterations for %s.m\n',fun);
  fprintf('   it             x                   fx\n');
end
it = 0;  maxit = 100;      %  Current and max number of iterations
fa = feval(fun,a);fb = feval(fun,b);
while it < maxit
  it = it + 1;
  dx = abs(b-a);
  x = b - fb*((b-a)/(fb-fa+eps));
  fx = feval(fun,x);
  if verbose fprintf('%4d  %20f  %20f \n',it,x,fx); end
  if (abs(b-a) < eps*b) | (fx == 0) % True when root is found
    r = x;  break;
    else a = b; fa = fb; b = x; fb = fx;
  end
end
if it == 100 warning(sprintf('root not within tolerance after %d iterations\n',it))
x
it
```

function	method	iterations	solution
$e^x + 2^{-x} + 2\cos x - 6 = 0, [1,2]$	bisection	52	1.82938360193385
	secant	7	1.82938360193385
$ln(x-1) + \cos(x-1) = 0, [1.3, 2]$	bisection	51	1.39774847595875
	secant	11	1.39774847595875
$e^x - 3x^2 = 0, [0,1]$	bisection	53	0.91000757248871
	secant	7	0.91000757248871
$e^x - 3x^2 = 0, [3,5]$	bisection	52	3.73307902863281
	secant	13	3.73307902863281

4. What are the advantages of combining the secant and biesction methods to obtain a hybrid method.

   Solution: Secant is quick to converge, but may not always do so. Bisection is guaranteed to converge, usually at a slower rate than secant. So the hybrid combines the two so that the algorithm is guaranteed to converge, but is sometimes quicker than the Bisection method.