Selected Solutions for Exercises in

Numerical Methods with MATLAB: Implementations and Applications

Gerald W. Recktenwald

Chapter 6

Finding the Roots of $f(x) = 0$

The following pages contain solutions to selected end-of-chapter Exercises from the book *Numerical Methods with* MATLAB*: Implementations and Applications*, by Gerald W. Recktenwald, © 2000, Prentice-Hall, Upper Saddle River, NJ. The solutions are © 2000 Gerald W. Recktenwald. The PDF version of the solutions may be downloaded or stored or printed only for noncommercial, educational use. Repackaging and sale of these solutions in any form, without the written consent of the author, is prohibited.

The latest version of this PDF file, along with other supplemental material for the book, can be found at `www.prenhall.com/recktenwald`.

6–2 The function $f(x) = \sin(x^2) + x^2 - 2x - 0.09$ has four roots in the interval $-1 \leq x \leq 3$. Given the m-file `fx.m`, which contains

```
function f = fx(x)
f = sin(x.^2) + x.^2 - 2*x - 0.09;
```

the statement

```
>> brackPlot('fx',-1,3)
```

produces only two brackets. Is this result due to a bug in `brackPlot` or `fx`? What needs to be changed so that all four roots are found? Demonstrate that your solution works.

Partial Solution: The statement

```
>> Xb = brackPlot('fx',-1,3)
Xb =
   -0.1579    0.0526
    2.1579    2.3684
```

returns two brackets. A close inspection of the plot of $f(x)$ reveals that $f(x)$ crosses the x-axis twice near $x = 1.3$. These two roots are missed by `brackPlot` because there default search interval is too coarse. There is no bug in `brackPlot`. Implementing a solution using a finer search interval is left as an exercise.

6–11 Use the `bisect` function to evaluate the root of the Colebrook equation (see Exercise 8) for $\epsilon/D = 0.02$ and $\mathrm{Re} = 10^5$. *Do not modify* `bisect.m`. This requires that you write an appropriate function m-file to evaluate the Colebrook equation.

Partial Solution: Using `bisect` requires writing an auxiliary function to evaluate the Colebrook equation in the form $F(f) = 0$, where f is the friction factor. The following form of $F(f)$ is used in the `colebrkz` function listed below.

$$F(f) = \frac{1}{\sqrt{f}} + 2\log_{10}\left(\frac{\epsilon/D}{3.7} + \frac{2.51}{Re_D\sqrt{f}}\right)$$

Many other forms of $F(f)$ will work.

```
function ff = colebrkz(f)
% COLEBRKZ  Evaluates the Colebrook equation in the form F(f) = 0
%           for use with root-finding routines.
%
% Input:   f = the current guess at the friction factor
%
%  Global Variables:
%    EPSDIA = ratio of relative roughness to pipe diameter
%    REYNOLDS = Reynolds number based on pipe diameter
%
% Output:  ff = the "value" of the Colebrook function written y = F(f)

% Global variables allow EPSDIA and REYNOLDS to be passed into
% colebrkz while bypassing the bisect.m or fzero function
global EPSDIA REYNOLDS
ff = 1.0/sqrt(f) + 2.0*log10( EPSDIA/3.7 + 2.51/( REYNOLDS*sqrt(f) ) );
```

Because the `bisect` function (unlike `fzero`) does not allow additional parameters to be passed through to the $F(f)$ function, the values of ϵ/D and Re are passed to `colebrkz` via global variables. Running `bisect` with `colebrkz` is left to the reader. For $\mathrm{Re} = 1 \times 10^5$ and $\epsilon/D = 0.02$ the solution is $f = 0.0490$.

Copyright © 2000, Gerald W. Recktenwald. Photocopying is permitted only for non-commercial educational purposes.

6–13 Derive the $g_3(x)$ functions in Example 6.4 and Example 6.5. (*Hint*: What is the fixed-point formula for Newton's method?)

Partial Solution: The fixed point iteration formulas designated as $g_3(x)$ in Example 6.4 and Example 6.5 are obtained by applying Newton's method. The general form of Newton's method for a scalar variable is

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Example 6.4: The $f(x)$ function and its derivative are

$$f(x) = x - x^{1/3} - 2 \qquad\qquad f'(x) = 1 - \frac{1}{3}x^{-2/3}$$

Substituting these expressions into the formula for Newton's method and simplifying gives

$$\begin{aligned}
x_{k+1} = x_k - \frac{x_k - x_k^{1/3} - 2}{1 - (1/3)x_k^{-2/3}} &= \frac{x_k(1 - (1/3)x_k^{-2/3}) - (x_k - x_k^{1/3} - 2)}{1 - (1/3)x_k^{-2/3}} \\
&= \frac{x_k - (1/3)x_k^{1/3} - x_k + x_k^{1/3} + 2}{1 - (1/3)x_k^{-2/3}} \\
&= \frac{(2/3)x_k^{1/3} + 2}{1 - (1/3)x_k^{-2/3}} \\
&= \frac{2x_k^{1/3} + 6}{3 - x_k^{-2/3}}
\end{aligned}$$

Repeating this analysis for Example 6.5 is left as an exercise.

Copyright © 2000, Gerald W. Recktenwald. Photocopying is permitted only for non-commercial educational purposes.

6–17 K. Wark and D. E. Richards (*Thermodynamics*, 6th ed., 1999, McGraw-Hill, Boston, Example 14-2, pp. 768–769) compute the equilibrium composition of a mixture of carbon monoxide and oxygen gas at one atmosphere. Determining the final composition requires solving

$$3.06 = \frac{(1-x)(3+x)^{1/2}}{x(1+x)^{1/2}}$$

for x. Obtain a fixed-point iteration formula for finding the roots of this equation. Implement your formula in a MATLAB function and use your function to find x. If your formula does not converge, develop one that does.

Partial Solution: One fixed point iteration formula is obtained by isolating the factor of $(3 + x)$ in the numerator.

$$\frac{3.06x(1+x)^{1/2}}{1-x} = (3+x)^{1/2} \quad \Longrightarrow \quad x = \left[\frac{3.06x(1+x)^{1/2}}{1-x}\right]^2 - 3$$

$$\Longrightarrow \quad g_1(x) = \left[\frac{3.06x(1+x)^{1/2}}{1-x}\right]^2 - 3$$

Another fixed point iteration formula is obtained by solving for the isolated x in the denominator to get

$$x = \frac{(1-x)(3+x)^{1/2}}{3.06(1+x)^{1/2}} \quad \Longrightarrow \quad g_2(x) = \frac{(1-x)(3+x)^{1/2}}{3.06(1+x)^{1/2}}$$

Performing 10 fixed point iterations with $g_1(x)$ gives

```
it     xnew
 1  -7.6420163e-01
 2  -2.5857113e+00
 3  -1.0721050e+01
 4  -7.9154865e+01
 5  -7.1666488e+02
 6  -6.6855377e+03
 7  -6.2575617e+04
 8  -5.8590795e+05
 9  -5.4861826e+06
10  -5.1370394e+07
```

Thus, $g_1(x)$ does not converge. The $g_2(x)$ function does converge to the true root of $x = 0.340327\ldots$. MATLAB implementations of the fixed point iterations are left as an Exercise.

Copyright © 2000, Gerald W. Recktenwald. Photocopying is permitted only for non-commercial educational purposes.

6–24 Create a modified `newton` function (say, `newtonb`) that takes a bracket interval as input instead of a single initial guess. From the bracket limits take one bisection step to determine x_0, the initial guess for Newton iterations. Use the bracket limits to develop relative tolerances on x and $f(x)$ as in the `bisect` function in Listing 6.4.

Solution: The `newtonb` function is listed below. The `demoNewtonb` function, also listed below, repeats the calculations in Example 6.8 with the original `newton` function and with the new `newtonb` function.

Running `demoNewtonb` gives

```
>> demoNewtonb

Original newton function:

Newton iterations for fx3n.m
  k      f(x)         dfdx         x(k+1)
  1   -4.422e-01    8.398e-01   3.52664429313903
  2    4.507e-03    8.561e-01   3.52138014739733
  3    3.771e-07    8.560e-01   3.52137970680457
  4    2.665e-15    8.560e-01   3.52137970680457
  5    0.000e+00    8.560e-01   3.52137970680457

newtonb function:

Newton iterations for fx3n.m
  k      f(x)         dfdx         x(k+1)
  1   -4.422e-01    8.398e-01   3.52664429313903
  2    4.507e-03    8.561e-01   3.52138014739733
  3    3.771e-07    8.560e-01   3.52137970680457
  4    2.665e-15    8.560e-01   3.52137970680457
  5    0.000e+00    8.560e-01   3.52137970680457
```

The two implementations of Newton's method give identical results because the input to `newtonb` is the bracket $[2, 4]$. This causes the initial bisection step to produce the same initial guess for the Newton iterations that is used in the call to `newton`.

```
function demoNewtonb
% demoNewtonb  Use newton and newtonb to find the root of f(x) = x - x^(1/3) - 2
%
% Synopsis:  demoNewton
%
%  Input:    none
%
%  Output    print out of convergence history, and comparison of methods

fprintf('\nOriginal newton function:\n');
r = newton('fx3n',3,5e-16,5e-16,1);

fprintf('\nnewtonb function:\n');
rb = newtonb('fx3n',[2 4],5e-16,5e-16,1);
```

Copyright © 2000, Gerald W. Recktenwald. Photocopying is permitted only for non-commercial educational purposes.

```
function r = newtonb(fun,x0,xtol,ftol,verbose)
% newtonb    Newton's method to find a root of the scalar equation f(x) = 0
%            Initial guess is a bracket interval
%
% Synopsis:  r = newtonb(fun,x0)
%            r = newtonb(fun,x0,xtol)
%            r = newtonb(fun,x0,xtol,ftol)
%            r = newtonb(fun,x0,xtol,ftol,verbose)
%
% Input:  fun     = (string) name of mfile that returns f(x) and f'(x).
%         x0      = 2-element vector providing an initial bracket for the root
%         xtol    = (optional) absolute tolerance on x.    Default: xtol=5*eps
%         ftol    = (optional) absolute tolerance on f(x). Default: ftol=5*eps
%         verbose = (optional) flag.  Default: verbose=0, no printing.
%
%  Output:  r = the root of the function

if nargin < 3,  xtol = 5*eps;  end
if nargin < 4,  ftol = 5*eps;  end
if nargin < 5,  verbose = 0;   end
xeps = max(xtol,5*eps);  feps = max(ftol,5*eps);  %  Smallest tols are 5*eps

if verbose
  fprintf('\nNewton iterations for %s.m\n',fun);
  fprintf('  k      f(x)         dfdx         x(k+1)\n');
end

xref = abs(x0(2)-x0(1));         %  Use initial bracket in convergence test
fa = feval(fun,x0(1));
fb = feval(fun,x0(2));
fref = max([abs(fa) abs(fb)]);   %  Use max f in convergence test
x = x0(1) + 0.5*(x0(2)-x0(1));   %  One bisection step for initial guess
k = 0;  maxit = 15;              %  Current and max iterations
while k <= maxit
  k = k + 1;
  [f,dfdx] = feval(fun,x);      %   Returns f( x(k-1) ) and f'( x(k-1) )
  dx = f/dfdx;
  x = x - dx;
  if verbose, fprintf('%3d %12.3e %12.3e %18.14f\n',k,f,dfdx,x);  end

  if ( abs(f/fref) < feps ) | ( abs(dx/xref) < xeps ),  r = x;  return;  end
end
warning(sprintf('root not found within tolerance after %d iterations\n',k));
```

Copyright © 2000, Gerald W. Recktenwald. Photocopying is permitted only for non-commercial educational purposes.

6–27 Implement the secant method using Algorithm 6.5 and Equation (6.13). Test your program by re-creating the results in Example 6.10. What happens if 10 iterations are performed? Replace the formula in Equation (6.13) with

$$x_{k+1} = x_k - f(x_k)\left[\frac{(x_k - x_{k-1})}{f(x_k) - f(x_{k-1}) + \varepsilon}\right],$$

where ε is a small number on the order of ε_m. How *and why* does this change the results?

Partial Solution: The `demoSecant` function listed below implements Algorithm (6.5) using Equation (6.13). The $f(x)$ function, Equation 6.3, is hard-coded into `demoSecant`. Note also that `demoSecant` performs ten iterations without checking for convergence.

```
function demoSecant(a,b);
% demoSecant  Secant method for finding the root of f(x) = x - x^(1/3) - 2 = 0
%             Implement Algorithm 6.5, using Equation (6.13)
%
% Synopsis:   demoSecant(a,b)
%
% Input:      a,b = initial guesses for the iterations
%
% Output:     print out of iterations; no return values.

% copy initial guesses to local variables
xk   = b;            %  x(k)
xkm1 = a;            %  x(k-1)
fk   = fx3(b);       %  f( x(k) )
fkm1 = fx3(a);       %  f( x(k-1) )

fprintf('\nSecant method: Algorithm 6.5, Equation (6.13) \n');
fprintf('  n      x(k-1)         x(k)       f( x(k) )\n');
fprintf('%3d  %12.8f  %12.8f  %12.5e\n',0,xkm1,xk,fk);

for n=1:10
  x = xk - fk*( xk-xkm1 )/( fk - fkm1);    % secant formula for updating the root
  f = fx3(x);
  fprintf('%3d  %12.8f  %12.8f  %12.5e\n',n,xk,x,f);
  xkm1 = xk;  xk = x;   %  set-up for next iteration
  fkm1 = fk;  fk = f;
end
```

Copyright © 2000, Gerald W. Recktenwald. Photocopying is permitted only for non-commercial educational purposes.

Running `demoSecant` with an initial bracket of $[3, 4]$ (the same bracket used in Example 6.10) gives

```
>> demoSecant(3,4)

Secant method: Algorithm 6.5, Equation (6.13)
  n      x(k-1)         x(k)       f( x(k) )
  0    3.00000000    4.00000000   4.12599e-01
  1    4.00000000    3.51734262  -3.45547e-03
  2    3.51734262    3.52135125  -2.43598e-05
  3    3.52135125    3.52137971   1.56730e-09
  4    3.52137971    3.52137971  -8.88178e-16
  5    3.52137971    3.52137971  -2.22045e-16
  6    3.52137971    3.52137971   0.00000e+00
  7    3.52137971    3.52137971   0.00000e+00

Warning: Divide by zero.
> In /werk/MATLAB_Book/SolutionManual/roots/mfiles/demoSecant.m at line 22
  8    3.52137971           NaN           NaN
  9           NaN           NaN           NaN
 10           NaN           NaN           NaN
```

The secant method has fully converged in 6 iterations. Continuing the calculations beyond convergence gives a floating point exception because $f(x_k) - f(x_{k-1}) = 0$ in the denominator of Equation (6.13). In general, it is possible to have $f(x_k) - f(x_{k-1}) = 0$ before the secant iterations reach convergence. Thus, the floating point exception exposed by `demoSecant` should be guarded against in any implementation of the secant method.

Implementing the fix suggested in the problem statement is left as an exercise for the reader.

Copyright © 2000, Gerald W. Recktenwald. Photocopying is permitted only for non-commercial educational purposes.

6–33 Write an m-file function to compute h, the depth to which a sphere of radius r, and specific gravity s, floats. (See Example 6.12 on page 281.) The inputs are r and s, and the output is h. Only compute h when $s < 0.5$. The $s \geq 0.5$ case is dealt with in the following Exercise. If $s \geq 0.5$ is input, have your function print an error message and stop. (The built-in `error` function will be useful.) Your function needs to include logic to select the correct root from the list of values returned by the built-in `roots` function.

Partial Solution: The `floata` function listed below performs the desired computations. We briefly discuss three of the key statements in `floata` The coefficients of the polynomial are stored in the `p` vector. Then

```
c = getreal(roots(p));
```

finds the real roots of the polynomial. The `getreal` subfunction returns only the real elements of a vector. Using `getreal` is a defensive programming strategy. The sample calculation in Example 6.12 obtained only real roots of the polynomial, so `getreal` would not be necessary *in that case*. The

```
k = find(c>0 & c<r);
```

statement extracts the indices in the c vector satisfying the criteria $0 \leq c_k \leq r$. Then

```
h = c(k);
```

copies those roots satisfying the criteria to the h vector. No assumption is made that only one root meets the criteria. If more than one root is found a warning message is issued before leaving `floata`.

Testing of `floata` is left to the reader.

Copyright © 2000, Gerald W. Recktenwald. Photocopying is permitted only for non-commercial educational purposes.

```
function h = floata(r,s)
% float  Find water depth on a floating, solid sphere with specific gravity < 0.5
%
% Synopsis:  h = floata(r,s)
%
% Input:     r = radius of the sphere
%            s = specific gravity of the sphere (0 < s < 1)
%
% Output:    h = depth of the sphere

if s>=0.5
  error('s<0.5 required in this version')
else
  p = [1   -3*r   0   4*s*r^3];      %  h^3 - 3*r*h + 4*s*r^3 = 0
  c = getreal(roots(p));
  k = find(c>0 & c<r);      % indices of elements in c such that 0 < c(k) < r
  h = c(k);                 % value of elements in c satisfying above criterion
end

if length(h)>1, warning('More than one root found');  end

% ==============================
function cr = getreal(c)
% getreal  Copy all real elements of input vector to output vector
%
% Synopsis:  cr = getreal(c)
%
% Input:  c = vector of numerical values
%
% Output  cr = vector of only the real elements of c
%              cr = [] if c has only imaginary elements
n = 0;
for k=1:length(c)
  if isreal(c(k))
    n = n + 1;
    cr(n) = c(k);
  end
end
if n==0,  cr = [];  warning('No real elements in the input vector');  end
```

Copyright © 2000, Gerald W. Recktenwald. Photocopying is permitted only for non-commercial educational purposes.