CHAPTER 6

6.1 The function can be set up for fixed-point iteration by solving it for x

$$x_{i+1} = \sin\left(\sqrt{x_i}\right)$$

Using an initial guess of $x_0 = 0.5$, the first iteration yields

$$x_1 = \sin\left(\sqrt{0.5}\right) = 0.649637$$

$$|\varepsilon_a| = \left|\frac{0.649637 - 0.5}{0.649637}\right| \times 100\% = 23\%$$

Second iteration:

$$x_2 = \sin\left(\sqrt{0.649637}\right) = 0.721524$$

$$|\varepsilon_a| = \left|\frac{0.721524 - 0.649637}{0.721524}\right| \times 100\% = 9.96\%$$

The process can be continued as tabulated below:

i	x_i	$\lvert\varepsilon_a\rvert$	E_t	$E_{t,i} / E_{t,i-1}$
0	0.500000		0.268648	
1	0.649637	23.0339%	0.119011	0.44300
2	0.721524	9.9632%	0.047124	0.39596
3	0.750901	3.9123%	0.017747	0.37660
4	0.762097	1.4691%	0.006551	0.36914
5	0.766248	0.5418%	0.002400	0.36632
6	0.767772	0.1984%	0.000876	0.36514
7	0.768329	0.0725%	0.000319	0.36432
8	0.768532	0.0265%	0.000116	0.36297
9	0.768606	0.0097%	0.000042	0.35956

Thus, after nine iterations, the root is estimated to be 0.768606 with an approximate error of 0.0097%.

To confirm that the scheme is linearly convergent, according to the book, the ratio of the errors between iterations should be

$$\frac{E_{i+1}}{E_i} = g'(\xi) = \frac{1}{2\sqrt{\xi}}\cos\left(\sqrt{\xi}\right)$$

Substituting the root for ξ gives a value of 0.365 which is close to the values in the last column of the table.

6.2 **(a)** The function can be set up for fixed-point iteration by solving it for x in two different ways. First, it can be solved for the linear x,

$$x_{i+1} = \frac{0.9x_i^2 - 2.5}{1.7}$$

Using an initial guess of 5, the first iteration yields

$$x_1 = \frac{0.9(5)^2 - 2.5}{1.7} = 11.76$$

$$|\varepsilon_a| = \left|\frac{11.76 - 5}{11.76}\right| \times 100\% = 57.5\%$$

Second iteration:

$$x_1 = \frac{0.9(11.76)^2 - 2.5}{1.7} = 71.8$$

$$|\varepsilon_a| = \left|\frac{71.8 - 11.76}{71.8}\right| \times 100\% = 83.6\%$$

Clearly, this solution is diverging. An alternative is to solve for the second-order *x*,

$$x_{i+1} = \sqrt{\frac{1.7x_i + 2.5}{0.9}}$$

Using an initial guess of 5, the first iteration yields

$$x_{i+1} = \sqrt{\frac{1.7(5) + 2.5}{0.9}} = 3.496$$

$$|\varepsilon_a| = \left|\frac{3.496 - 5}{3.496}\right| \times 100\% = 43.0\%$$

Second iteration:

$$x_{i+1} = \sqrt{\frac{1.7(3.496) + 2.5}{0.9}} = 3.0629$$

$$|\varepsilon_a| = \left|\frac{3.0629 - 3.496}{3.0629}\right| \times 100\% = 14.14\%$$

This version is converging. All the iterations can be tabulated as

iteration	x_i	$\lvert\varepsilon_a\rvert$
0	5.000000	
1	3.496029	43.0194%
2	3.062905	14.1410%
3	2.926306	4.6680%
4	2.881882	1.5415%
5	2.867287	0.5090%
6	2.862475	0.1681%
7	2.860887	0.0555%
8	2.860363	0.0183%
9	2.860190	0.0061%

Thus, after 9 iterations, the root estimate is 2.860190 with an approximate error of 0.0061%. The result can be checked by substituting it back into the original function,

$$f(2.860190) = -0.9(2.860190)^2 + 1.7(2.860190) + 2.5 = -0.000294$$

(b) The formula for Newton-Raphson is

$$x_{i+1} = x_i - \frac{-0.9x_i^2 + 1.7x_i + 2.5}{-1.8x_i + 1.7}$$

Using an initial guess of 5, the first iteration yields

$$x_{i+1} = 5 - \frac{-0.9(5)^2 + 1.7(5) + 2.5}{-1.8(5) + 1.7} = 3.424658$$

$$|\varepsilon_a| = \left|\frac{3.424658 - 5}{3.424658}\right| \times 100\% = 46.0\%$$

Second iteration:

$$x_{i+1} = 3.424658 - \frac{-0.9(3.424658)^2 + 1.7(3.424658) + 2.5}{-1.8(3.424658) + 1.7} = 2.924357$$

$$|\varepsilon_a| = \left|\frac{2.924357 - 3.424658}{2.924357}\right| \times 100\% = 17.1\%$$

The process can be continued as tabulated below:

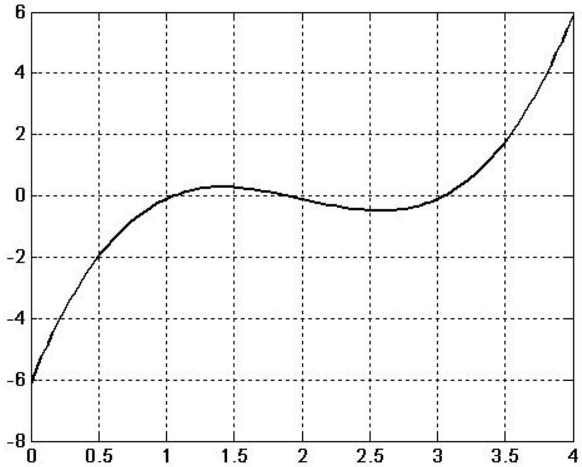
iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	5	−11.5	−7.3	
1	3.424658	−2.23353	−4.46438	46.0000%
2	2.924357	−0.22527	−3.56384	17.1081%
3	2.861147	−0.00360	−3.45006	2.2093%
4	2.860105	−9.8E−07	−3.44819	0.0364%
5	2.860104	−7.2E−14	−3.44819	0.0000%

After 5 iterations, the root estimate is **2.860104** with an approximate error of 0.0000%. The result can be checked by substituting it back into the original function,

$$f(2.860104) = -0.9(2.860104)^2 + 1.7(2.860104) + 2.5 = -7.2 \times 10^{-14}$$

6.3 (a)

```
>> x = linspace(0,4);
>> y = x.^3-6*x.^2+11*x-6.1;
>> plot(x,y)
>> grid
```



Estimates are approximately 1.05, 1.9 and 3.05.

(b) The formula for Newton-Raphson is

$$x_{i+1} = x_i - \frac{x_i^3 - 6x_i^2 + 11x_i - 6.1}{3x_i^2 - 12x_i + 11}$$

Using an initial guess of 3.5, the first iteration yields

$$x_1 = 3.5 - \frac{(3.5)^3 - 6(3.5)^2 + 11(3.5) - 6.1}{3(3.5)^2 - 12(3.5) + 11} = 3.191304$$

$$|\varepsilon_a| = \left|\frac{3.191304 - 3.5}{3.191304}\right| \times 100\% = 9.673\%$$

Second iteration:

$$x_2 = 3.191304 - \frac{(3.191304)^3 - 6(3.191304)^2 + 11(3.191304) - 6.1}{3(3.191304)^2 - 12(3.191304) + 11} = 3.068699$$

$$|\varepsilon_a| = \left|\frac{3.068699 - 3.191304}{3.068699}\right| \times 100\% = 3.995\%$$

Third iteration:

$$x_3 = 3.068699 - \frac{(3.068699)^3 - 6(3.068699)^2 + 11(3.068699) - 6.1}{3(3.068699)^2 - 12(3.068699) + 11} = 3.047317$$

$$|\varepsilon_a| = \left|\frac{3.047317 - 3.068699}{3.047317}\right| \times 100\% = 0.702\%$$

(c) For the secant method, the first iteration:

$x_{-1} = 2.5$ $\qquad f(x_{-1}) = -0.475$
$x_0 = 3.5$ $\qquad f(x_0) = 1.775$

$$x_1 = 3.5 - \frac{1.775(2.5-3.5)}{-0.475-1.775} = 2.711111$$

$$|\varepsilon_a| = \left|\frac{2.711111-3.5}{2.711111}\right| \times 100\% = 29.098\%$$

Second iteration:

$x_0 = 3.5$ $f(x_0) = 1.775$
$x_1 = 2.711111$ $f(x_1) = -0.45152$

$$x_2 = 2.711111 - \frac{-0.45152(3.5-2.711111)}{1.775-(-0.45152)} = 2.871091$$

$$|\varepsilon_a| = \left|\frac{2.871091-2.711111}{2.871091}\right| \times 100\% = 5.572\%$$

Third iteration:

$x_1 = 2.711111$ $f(x_1) = -0.45152$
$x_2 = 2.871091$ $f(x_2) = -0.31011$

$$x_3 = 2.871091 - \frac{-0.31011(2.711111-2.871091)}{-0.45152-(-0.31011)} = 3.221923$$

$$|\varepsilon_a| = \left|\frac{3.221923-2.871091}{3.221923}\right| \times 100\% = 10.889\%$$

(d) For the modified secant method, the first iteration:

$x_0 = 3.5$ $f(x_0) = 1.775$
$x_0 + \delta x_0 = 3.535$ $f(x_0 + \delta x_0) = 1.981805$

$$x_1 = 3.5 - \frac{0.01(3.5)1.775}{1.981805-1.775} = 3.199597$$

$$|\varepsilon_a| = \left|\frac{3.199597-3.5}{3.199597}\right| \times 100\% = 9.389\%$$

Second iteration:

$x_1 = 3.199597$ $f(x_1) = 0.426661904$
$x_1 + \delta x_1 = 3.271725$ $f(x_1 + \delta x_1) = 0.536512631$

$$x_2 = 3.199597 - \frac{0.01(3.199597)0.426661904}{0.536513-0.426661904} = 3.075324$$

$$|\varepsilon_a| = \left|\frac{3.075324-3.199597}{3.075324}\right| \times 100\% = 4.041\%$$

Third iteration:

$x_2 = 3.075324$ $f(x_2) = 0.068096$
$x_2 + \delta x_2 = 3.143675$ $f(x_2 + \delta x_2) = 0.147105$

$$x_3 = 3.075324 - \frac{0.01(3.075324)0.068096}{0.147105 - 0.068096} = 3.048818$$

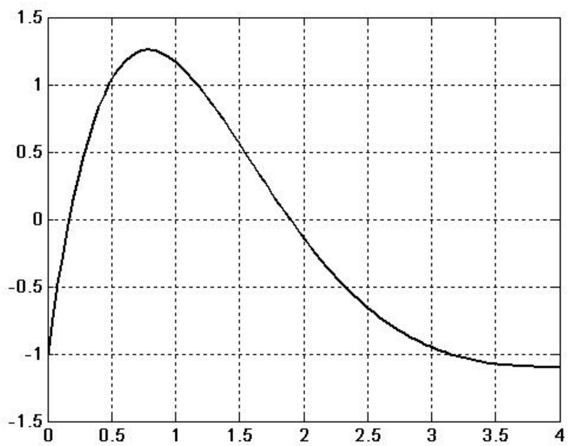
$$|\varepsilon_a| = \left|\frac{3.048818 - 3.075324}{3.048818}\right| \times 100\% = 0.869\%$$

(e)

```
>> a = [1 -6 11 -6.1]
a =
    1.0000   -6.0000   11.0000   -6.1000

>> roots(a)
ans =
    3.0467
    1.8990
    1.0544
```

6.4 (a)

```
>> x = linspace(0,4);
>> y = 7*sin(x).*exp(-x)-1;
>> plot(x,y)
>> grid
```



The lowest positive root seems to be at approximately 0.2.

(b) The formula for Newton-Raphson is

$$x_{i+1} = x_i - \frac{7\sin(x_i)e^{-x_i} - 1}{7e^{-x_i}(\cos(x_i) - \sin(x_i))}$$

Using an initial guess of 0.3, the first iteration yields

$$x_1 = 0.3 - \frac{7\sin(0.3)e^{-0.3} - 1}{7e^{-0.3}(\cos(0.3) - \sin(0.3))} = 0.3 - \frac{0.532487}{3.421627} = 0.144376$$

$$|\varepsilon_a| = \left|\frac{0.144376 - 0.3}{0.144376}\right| \times 100\% = 107.8\%$$

Second iteration:

$$x_2 = 0.144376 - \frac{7\sin(0.144376)e^{-0.144376} - 1}{7e^{-0.144376}(\cos(0.144376) - \sin(0.144376))} = 0.144376 - \frac{-0.12827}{5.124168} = 0.169409$$

$$|\varepsilon_a| = \left|\frac{0.169409 - 0.144376}{0.169409}\right| \times 100\% = 14.776\%$$

Third iteration:

$$x_1 = 0.169409 - \frac{7\sin(0.169409)e^{-0.169409} - 1}{7e^{-0.169409}(\cos(0.169409) - \sin(0.169409))} = 0.169409 - \frac{-0.00372}{4.828278} = 0.170179$$

$$|\varepsilon_a| = \left|\frac{0.170179 - 0.169409}{0.170179}\right| \times 100\% = 0.453\%$$

(c) For the secant method, the first iteration:

$x_{-1} = 0.5$ $f(x_{-1}) = 1.03550$
$x_0 = 0.4$ $f(x_0) = 0.827244$

$$x_1 = 0.4 - \frac{0.827244(0.5 - 0.4)}{1.03550 - 0.827244} = 0.002782$$

$$|\varepsilon_a| = \left|\frac{0.002782 - 0.4}{0.002782}\right| \times 100\% = 14{,}278\%$$

Second iteration:

$x_0 = 0.4$ $f(x_0) = 0.827244$
$x_1 = 0.002782$ $f(x_1) = -0.980580$

$$x_2 = 0.002782 - \frac{-0.98058(0.4 - 0.002782)}{0.827244 - (-0.98058)} = 0.218237$$

$$|\varepsilon_a| = \left|\frac{0.218237 - 0.002782}{0.218237}\right| \times 100\% = 98.725\%$$

Third iteration:

$x_1 = 0.002782$ $f(x_1) = -0.980580$
$x_2 = 0.218237$ $f(x_2) = 0.218411$

$$x_3 = 0.218237 - \frac{0.218411(0.002782 - 0.218237)}{-0.98058 - 0.218411} = 0.178989$$

$$|\varepsilon_a| = \left|\frac{0.178989 - 0.218237}{0.178989}\right| \times 100\% = 21.927\%$$

(d) For the modified secant method:

First iteration:

$x_0 = 0.3$ $f(x_0) = 0.532487$
$x_0 + \delta x_0 = 0.303$ $f(x_0 + \delta x_0) = 0.542708$

$$x_1 = 0.3 - \frac{0.01(0.3)0.532487}{0.542708 - 0.532487} = 0.143698$$

$$|\varepsilon_a| = \left|\frac{0.143698 - 0.3}{0.143698}\right| \times 100\% = 108.8\%$$

Second iteration:

$x_1 = 0.14369799$ $\qquad f(x_1) = -0.13175$

$x_1 + \delta x_1 = 0.14513497$ $\qquad f(x_1 + \delta x_1) = -0.12439$

$$x_2 = 0.143698 - \frac{0.01(0.143698)(-0.13175)}{-0.12439 - (-0.13175)} = 0.169412$$

$$|\varepsilon_a| = \left|\frac{0.169412 - 0.143698}{0.169412}\right| \times 100\% = 15.18\%$$

Third iteration:

$x_2 = 0.169411504$ $\qquad f(x_2) = -0.00371$

$x_2 + \delta x_2 = 0.17110562$ $\qquad f(x_2 + \delta x_2) = 0.004456$

$$x_3 = 0.169411504 - \frac{0.01(0.169411504)(-0.00371)}{0.004456 - (-0.00371)} = 0.170180853$$

$$|\varepsilon_a| = \left|\frac{0.170181 - 0.169412}{0.170181}\right| \times 100\% = 0.452\%$$

Errata: In the first printing, the problem specified five iterations.

Fourth iteration:

$x_3 = 0.170180853$ $\qquad f(x_3) = 4.14\times10^{-6}$

$x_3 + \delta x_3 = 0.17188266$ $\qquad f(x_3 + \delta x_3) = 0.008189$

$$x_4 = 0.170180853 - \frac{0.01(0.170180853)(4.14\times10^{-6})}{0.008189 - 4.14\times10^{-6}} = 0.170179992$$

$$|\varepsilon_a| = \left|\frac{0.170179992 - 0.170180853}{0.170179992}\right| \times 100\% = 0.001\%$$

Fifth iteration:

$x_3 = 0.170179992$ $\qquad f(x_3) = -8.5\times10^{-9}$

$x_3 + \delta x_3 = 0.17188179$ $\qquad f(x_3 + \delta x_3) = 0.008185$

$$x_4 = 0.170179992 - \frac{0.01(0.170179992)(-8.5\times10^{-9})}{0.008185 - (-8.5\times10^{-9})} = 0.170179994$$

$$|\varepsilon_a| = \left|\frac{0.170179994 - 0.170179992}{0.170179994}\right| \times 100\% = 0.000\%$$

6.5 (a) The formula for Newton-Raphson is

$$x_{i+1} = x_i - \frac{x_i^5 - 16.05x_i^4 + 88.75x_i^3 - 192.0375x_i^2 + 116.35x_i + 31.6875}{5x_i^4 - 64.2x_i^3 + 266.25x_i^2 - 384.075x_i + 116.35}$$

Using an initial guess of 0.5825, the first iteration yields

$$x_1 = 0.5825 - \frac{50.06217}{-29.1466} = 2.300098$$

$$|\varepsilon_a| = \left|\frac{2.300098 - 0.5825}{2.300098}\right| \times 100\% = 74.675\%$$

Second iteration

$$x_1 = 2.300098 - \frac{-21.546}{0.245468} = 90.07506$$

$$|\varepsilon_a| = \left|\frac{90.07506 - 2.300098}{90.07506}\right| \times 100\% = 97.446\%$$

Thus, the result seems to be diverging. However, the computation eventually settles down and converges (at a very slow rate) on a root at $x = 6.5$. The iterations can be summarized as

iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	0.582500	50.06217	−29.1466	
1	2.300098	−21.546	0.245468	74.675%
2	90.07506	4.94E+09	2.84E+08	97.446%
3	72.71520	1.62E+09	1.16E+08	23.874%
4	58.83059	5.3E+08	47720880	23.601%
5	47.72701	1.74E+08	19552115	23.265%
6	38.84927	56852563	8012160	22.852%
7	31.75349	18616305	3284098	22.346%
8	26.08487	6093455	1346654	21.731%
9	21.55998	1993247	552546.3	20.987%
10	17.95260	651370.2	226941	20.094%
11	15.08238	212524.6	93356.59	19.030%
12	12.80590	69164.94	38502.41	17.777%
13	11.00952	22415.54	15946.36	16.317%
14	9.603832	7213.396	6652.03	14.637%
15	8.519442	2292.246	2810.851	12.728%
16	7.703943	710.9841	1217.675	10.585%
17	7.120057	209.2913	556.1668	8.201%
18	6.743746	54.06896	286.406	5.580%
19	6.554962	9.644695	187.9363	2.880%
20	6.503643	0.597806	164.8912	0.789%
21	6.500017	0.00285	163.32	0.056%
22	6.5	6.58E−08	163.3125	0.000%

(b) For the modified secant method:

<u>First iteration:</u>

$x_0 = 0.5825$ $f(x_0) = 50.06217$

$x_0 + \delta x_0 = 0.611625$ $f(x_0 + \delta x_0) = 49.15724$

$$x_1 = 0.5825 - \frac{0.05(0.5825)50.06217}{49.15724 - 50.06217} = 2.193735$$

$$|\varepsilon_a| = \left|\frac{2.193735 - 0.5825}{2.193735}\right| \times 100\% = 73.447\%$$

<u>Second iteration:</u>

$x_1 = 2.193735$ $f(x_1) = -21.1969$

$x_1 + \delta x_1 = 2.303422$ $f(x_1 + \delta x_1) = -21.5448$

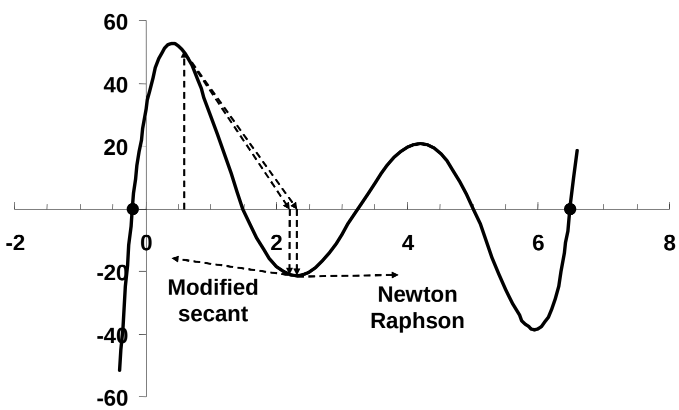
$$x_2 = 2.193735 - \frac{0.05(2.193735)(-21.1969)}{-21.5448 - (-21.1969)} = -4.48891$$

$$\left|\varepsilon_a\right| = \left|\frac{-4.48891 - 2.193735}{-4.48891}\right| \times 100\% = 148.87\%$$

Again, the result seems to be diverging. However, the computation eventually settles down and converges on a root at $x = -0.2$. The iterations can be summarized as

iteration	x_i	$x_i+\delta x_i$	$f(x_i)$	$f(x_i+\delta x_i)$	$\|\varepsilon_a\|$
0	0.5825	0.611625	50.06217	49.15724	
1	2.193735	2.303422	–21.1969	–21.5448	73.447%
2	–4.48891	–4.71336	–20727.5	–24323.6	148.870%
3	–3.19524	–3.355	–7201.94	–8330.4	40.487%
4	–2.17563	–2.28441	–2452.72	–2793.57	46.865%
5	–1.39285	–1.46249	–808.398	–906.957	56.200%
6	–0.82163	–0.86271	–250.462	–277.968	69.524%
7	–0.44756	–0.46994	–67.4718	–75.4163	83.579%
8	–0.25751	–0.27038	–12.5942	–15.6518	73.806%
9	–0.20447	–0.2147	–0.91903	–3.05726	25.936%
10	–0.20008	–0.21008	–0.01613	–2.08575	2.196%
11	–0.2	–0.21	–0.0002	–2.0686	0.039%
12	–0.2	–0.21	–2.4E–06	–2.06839	0.000%

Explanation of results: The results are explained by looking at a plot of the function. The guess of 0.5825 is located at a point where the function is relatively flat. Therefore, the first iteration results in a prediction of 2.3 for Newton-Raphson and 2.193 for the secant method. At these points the function is very flat and hence, the Newton-Raphson results in a very high value (90.075), whereas the modified false position goes in the opposite direction to a negative value (-4.49). Thereafter, the methods slowly converge on the nearest roots.



6.6

```
function root = secant(func,xrold,xr,es,maxit)
% secant(func,xrold,xr,es,maxit):
%   uses secant method to find the root of a function
% input:
%   func = name of function
%   xrold, xr = initial guesses
%   es = (optional) stopping criterion (%)
%   maxit = (optional) maximum allowable iterations
% output:
%   root = real root

% if necessary, assign default values
```

```
if nargin<5, maxit=50; end     %if maxit blank set to 50
if nargin<4, es=0.001; end     %if es blank set to 0.001
% Secant method
iter = 0;
while (1)
  xrn = xr - func(xr)*(xrold - xr)/(func(xrold) - func(xr));
  iter = iter + 1;
  if xrn ~= 0, ea = abs((xrn - xr)/xrn) * 100; end
  if ea <= es | iter >= maxit, break, end
  xrold = xr;
  xr = xrn;
end
root = xrn;
```

Test by solving Prob. 6.3:

```
format long
f=@(x) x^3-6*x^2+11*x-6.1;
secant(f,2.5,3.5)
ans =
   3.046680527126298
```

6.7

```
function root = modsec(func,xr,delta,es,maxit)
% modsec(func,xr,delta,es,maxit):
%   uses modified secant method to find the root of a function
% input:
%   func = name of function
%   xr = initial guess
%   delta = perturbation fraction
%   es = (optional) stopping criterion (%)
%   maxit = (optional) maximum allowable iterations
% output:
%   root = real root

% if necessary, assign default values
if nargin<5, maxit=50; end     %if maxit blank set to 50
if nargin<4, es=0.001; end     %if es blank set to 0.001
if nargin<3, delta=1E-5; end   %if delta blank set to 0.00001

% Secant method
iter = 0;
while (1)
  xrold = xr;
  xr = xr - delta*xr*func(xr)/(func(xr+delta*xr)-func(xr));
  iter = iter + 1;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  if ea <= es | iter >= maxit, break, end
end
root = xr;
```

Test by solving Prob. 6.3:

```
format long
f=@(x) x^3-6*x^2+11*x-6.1;
modsec(f,3.5,0.02)
ans =
   3.046682670215557
```

6.8 The equation to be differentiated is

$$f(m) = \sqrt{\frac{gm}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}} t\right) - v$$

Note that

$$\frac{d \tanh u}{dx} = \text{sech}^2 u \frac{du}{dx}$$

Therefore, the derivative can be evaluated as

$$\frac{df(m)}{dm} = \sqrt{\frac{gm}{c_d}} \text{sech}^2\left(\sqrt{\frac{gc_d}{m}} t\right)\left(-\frac{1}{2}\sqrt{\frac{m}{c_d g}}\right) t \frac{c_d g}{m^2} + \tanh\left(\sqrt{\frac{gc_d}{m}} t\right)\frac{1}{2}\sqrt{\frac{c_d}{gm}} \frac{g}{c_d}$$

The two terms can be reordered

$$\frac{df(m)}{dm} = \frac{1}{2}\sqrt{\frac{c_d}{gm}} \frac{g}{c_d} \tanh\left(\sqrt{\frac{gc_d}{m}} t\right) - \frac{1}{2}\sqrt{\frac{gm}{c_d}}\sqrt{\frac{m}{c_d g}} \frac{c_d g}{m^2}\; t\; \text{sech}^2\left(\sqrt{\frac{gc_d}{m}} t\right)$$

The terms premultiplying the tanh and sech can be simplified to yield the final result

$$\frac{df(m)}{dm} = \frac{1}{2}\sqrt{\frac{g}{mc_d}} \tanh\left(\sqrt{\frac{gc_d}{m}} t\right) - \frac{g}{2m} t\; \text{sech}^2\left(\sqrt{\frac{gc_d}{m}}\; t\right)$$

6.9 **(a)** The formula for Newton-Raphson is

$$x_{i+1} = x_i - \frac{-2 + 6x_i - 4x_i^2 + 0.5x_i^3}{6 - 8x_i + 1.5x_i^2}$$

Using an initial guess of 4.5, the iterations proceed as

iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	4.5	–10.4375	0.375	
1	32.333330	12911.57	1315.5	86.082%
2	22.518380	3814.08	586.469	43.586%
3	16.014910	1121.912	262.5968	40.609%
4	11.742540	326.4795	118.8906	36.384%
5	8.996489	92.30526	55.43331	30.524%
6	7.331330	24.01802	27.97196	22.713%
7	6.472684	4.842169	17.06199	13.266%
8	6.188886	0.448386	13.94237	4.586%
9	6.156726	0.005448	13.6041	0.522%
10	6.156325	8.39E–07	13.59991	0.007%

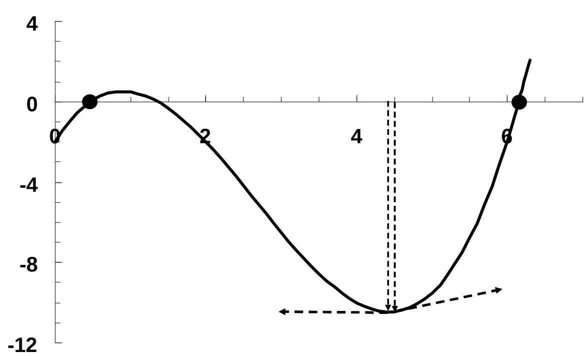
Thus, after an initial jump, the computation eventually settles down and converges on a root at x = 6.156325.

(b) Using an initial guess of 4.43, the iterations proceed as

iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	4.43	-10.4504	-0.00265	
1	-3939.13	-3.1E+10	23306693	100.112%
2	-2625.2	-9.1E+09	10358532	50.051%
3	-1749.25	-2.7E+09	4603793	50.076%
4	-1165.28	-8E+08	2046132	50.114%
.				
.				
.				
21	0.325261	-0.45441	3.556607	105.549%
22	0.453025	-0.05629	2.683645	28.203%
23	0.474	-0.00146	2.545015	4.425%
24	0.474572	-1.1E-06	2.541252	0.121%
25	0.474572	-5.9E-13	2.541249	0.000%

This time the solution jumps to an extremely large negative value The computation eventually converges at a very slow rate on a root at $x = 0.474572$.

Explanation of results: The results are explained by looking at a plot of the function. Both guesses are in a region where the function is relatively flat. Because the two guesses are on opposite sides of a minimum, both are sent to different regions that are far from the initial guesses. Thereafter, the methods slowly converge on the nearest roots.



6.10 The function to be evaluated is

$$x = \sqrt{a}$$

This equation can be squared and expressed as a roots problem,

$$f(x) = x^2 - a$$

The derivative of this function is

$$f'(x) = 2x$$

These functions can be substituted into the Newton-Raphson equation (Eq. 6.6),

$$x_{i+1} = x_i - \frac{x_i^2 - a}{2x_i}$$

which can be expressed as

$$x_{i+1} = \frac{x_i + a / x_i}{2}$$

6.11 (a) The formula for Newton-Raphson is

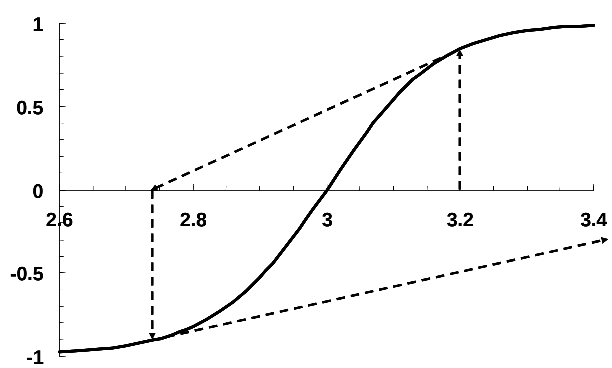
$$x_{i+1} = x_i - \frac{\tanh\left(x_i^2 - 9\right)}{2x_i \operatorname{sech}^2\left(x_i^2 - 9\right)}$$

Using an initial guess of 3.2, the iterations proceed as

iteration	x_i	$f(x_i)$	$f'(x_i)$	ε_a
0	3.2	0.845456	1.825311	
1	2.736816	–0.906910	0.971640	16.924%
2	3.670197	0.999738	0.003844	25.431%
3	–256.413			101.431%

Note that on the fourth iteration, the computation should go unstable.

(b) The solution diverges from its real root of $x = 3$. Due to the concavity of the slope, the next iteration will always diverge. The following graph illustrates how the divergence evolves.



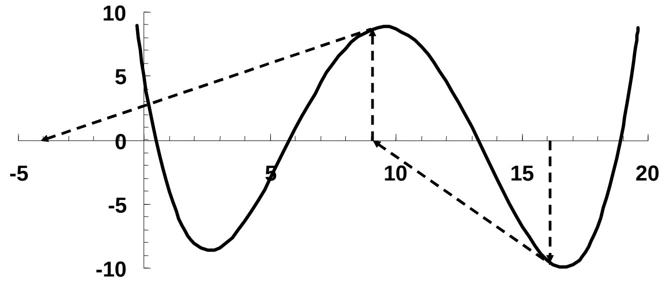
6.12 The formula for Newton-Raphson is

$$x_{i+1} = x_i - \frac{0.0074x_i^4 - 0.284x_i^3 + 3.355x_i^2 - 12.183x_i + 5}{0.0296x_i^3 - 0.852x_i^2 + 6.71x_i - 12.1832}$$

Using an initial guess of 16.15, the iterations proceed as

iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	16.15	–9.57445	–1.35368	
1	9.077102	8.678763	0.662596	77.920%
2	–4.02101	128.6318	–54.864	325.742%
3	–1.67645	36.24995	–25.966	139.852%
4	–0.2804	8.686147	–14.1321	497.887%
5	0.334244	1.292213	–10.0343	183.890%
6	0.463023	0.050416	–9.25584	27.813%
7	0.46847	8.81E–05	–9.22351	1.163%
8	0.46848	2.7E–10	–9.22345	0.002%

As depicted below, the iterations involve regions of the curve that have flat slopes. Hence, the solution is cast far from the roots in the vicinity of the original guess.

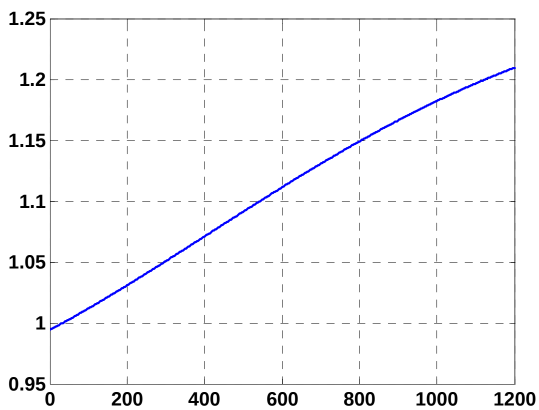


6.13 The solution can be formulated as

$$f(T) = 0 = -0.10597 + 1.671\times10^{-4}T + 9.7215\times10^{-8}T^2 - 9.5838\times10^{-11}T^3 + 1.9520\times10^{-14}T^4$$

A MATLAB script can be used to generate the plot and determine all the roots of this polynomial,

```
clear,clc,clf,format long g
cp=[1.952e-14 -9.5838e-11 9.7215e-8 1.671e-4 0.99403];
T=[0:1200];
cp_plot=polyval(cp,T);
plot(T,cp_plot),grid
x=[1.952e-14 -9.5838e-11 9.7215e-8 1.671e-4 -0.10597];
roots(x)

ans =
       2748.3 +     1126.3i
       2748.3 -     1126.3i
        -1131
       544.09
```



The only realistic value is 544.09. This value can be checked using the `polyval` function,

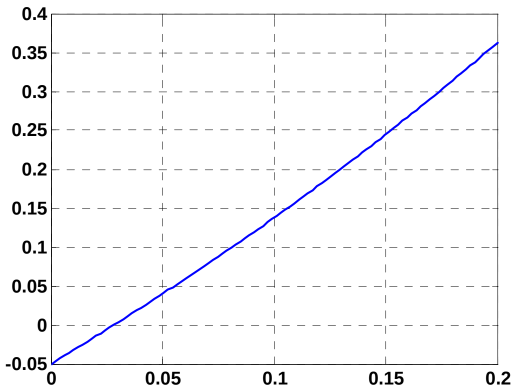
```
>> polyval(x,544.09)
   ans =
     4.9333e-007
```

6.14 The solution involves determining the root of

$$f(x) = \frac{x}{1-x}\sqrt{\frac{6}{2+x}} - 0.05$$

MATLAB can be used to develop a plot that indicates that a root occurs in the vicinity of $x = 0.03$.

```
f=@(x) x./(1-x).*sqrt(6./(2+x))-0.05;
x = linspace(0,.2);
y = f(x);
plot(x,y),grid
```



The `fzero` function can then be used to find the root

```
format long
fzero(f,0.03)

ans =
   0.028249441148471
```

6.15 The coefficient, *a* and *b*, can be evaluated as

```
>> format long
>> R = 0.518;pc = 4600;Tc = 191;
>> a = 0.427*R^2*Tc^2.5/pc
a =
  12.55778319740302
>> b = 0.0866*R*Tc/pc
b =
   0.00186261539130
```

The solution, therefore, involves determining the root of

$$f(v) = 65{,}000 - \frac{0.518(233.15)}{v - 0.0018626} + \frac{12.557783}{v(v + 0.0018626)\sqrt{233.15}}$$

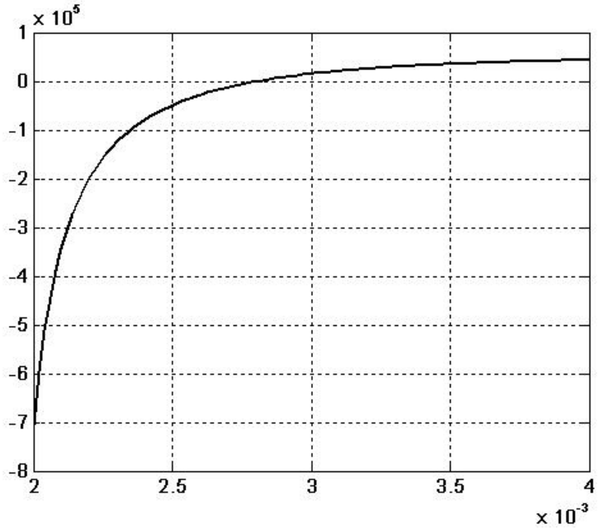
MATLAB can be used to generate a plot of the function and to solve for the root. One way to do this is to develop an M-file for the function,

```
function y = fvol(v)
R = 0.518;pc = 4600;Tc = 191;
a = 0.427*R^2*Tc^2.5/pc;
b = 0.0866*R*Tc/pc;
T = 273.15-40;p = 65000;
```

```
y = p - R*T./(v-b)+a./(v.*(v+b)*sqrt(T));
```

This function is saved as `fvol.m`. It can then be used to generate a plot

```
>> v = linspace(0.002,0.004);
>> fv = fvol(v);
>> plot(v,fv)
>> grid
```



Thus, a root is located at about 0.0028. The `fzero` function can be used to refine this estimate,

```
>> vroot = fzero('fvol',0.0028)
vroot =
   0.00280840865703
```

The mass of methane contained in the tank can be computed as

$$\text{mass} = \frac{V}{v} = \frac{3}{0.0028084} = 1068.317\ \text{m}^3$$

6.16 The function to be evaluated is

$$f(h) = V - \left[r^2 \cos^{-1}\left(\frac{r-h}{r}\right) - (r-h)\sqrt{2rh-h^2} \right] L$$

To use MATLAB to obtain a solution, the function can be written as an M-file

```
function y = fh(h,r,L,V)
y = V - (r^2*acos((r-h)/r)-(r-h)*sqrt(2*r*h-h^2))*L;
```

The `fzero` function can be used to determine the root as

```
>> format long
>> r = 2;L = 5;V = 8;
>> h = fzero('fh',0.5,[],r,L,V)
h =
   0.74001521805594
```

6.17 (a) The function to be evaluated is

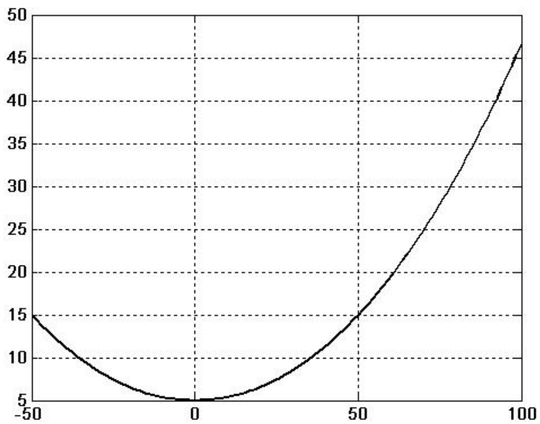
$$f(T_A) = 10 - \frac{T_A}{10}\cosh\left(\frac{500}{T_A}\right) + \frac{T_A}{10}$$

The solution can be obtained with the `fzero` function as

```
>> format long
>> TA = fzero(inline('10-x/10*cosh(500/x)+x/10'),1000)

TA =
    1.266324360399887e+003
```

(b) A plot of the cable can be generated as

```
>> x = linspace(-50,100);
>> w = 10;y0 = 5;
>> y = TA/w*cosh(w*x/TA) + y0 - TA/w;
>> plot(x,y),grid
```

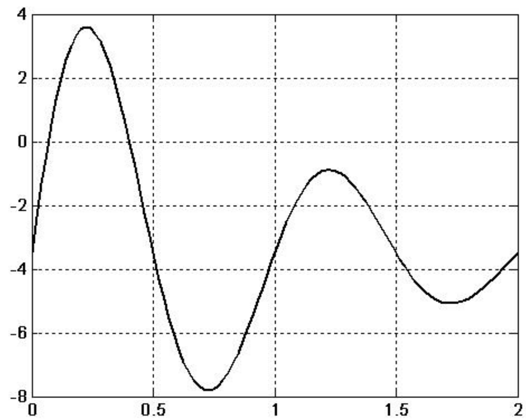


6.18 The function to be evaluated is

$$f(t) = 9e^{-t}\sin(2\pi t) - 3.5$$

A plot can be generated with MATLAB,

```
>> t = linspace(0,2);
>> ft = @(t) 9*exp(-t) .* sin(2*pi*t) – 3.5;
>> y=ft(t);
>> plot(t,y),grid
```



Thus, there appear to be two roots at approximately 0.1 and 0.4. The `fzero` function can be used to obtain refined estimates,

```
>> t = fzero(ft,[0 0.2])
t =
   0.06835432096851

>> t = fzero(ft,[0.2 0.8])
t =
   0.40134369265980
```

6.19 The function to be evaluated is

$$f(\omega) = \frac{1}{Z} - \sqrt{\frac{1}{R^2} + \left(\omega C - \frac{1}{\omega L}\right)^2}$$

Substituting the parameter values yields

$$f(\omega) = 0.01 - \sqrt{\frac{1}{50625} + \left(0.6 \times 10^{-6}\,\omega - \frac{2}{\omega}\right)^2}$$

The `fzero` function can be used to determine the root as

```
>> fzero('0.01-sqrt(1/50625+(0.6e-6*x-2./x).^2)',[1 1000])
ans =
  220.0202
```

6.20 The following script uses the `fzero` function can be used to determine the root as

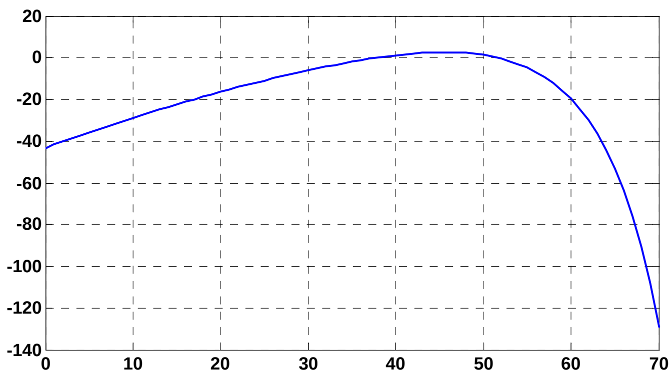
```
format long
k1=40000;k2=40;m=95;g=9.81;h=0.43;
fd=@(d) 2*k2*d^(5/2)/5+0.5*k1*d^2-m*g*d-m*g*h;
fzero(fd,1)

ans =
   0.166723562437785
```

6.21 If the height at which the throw leaves the right fielders arm is defined as $y = 0$, the y at 90 m will be –0.8. Therefore, the function to be evaluated is

$$f(\theta) = 0.8 + 90\tan\left(\frac{\pi}{180}\theta_0\right) - \frac{44.145}{\cos^2(\pi\theta_0/180)}$$

Note that the angle is expressed in degrees. First, MATLAB can be used to plot this function versus various angles:

```
format long
g=9.81;v0=30;y0=1.8;
fth=@(th) 0.8+90*tan(pi*th/180)-44.1./cos(pi*th/180).^2;
thplot=[0:70];fplot=fth(thplot);
plot(thplot,fplot),grid
```



Roots seem to occur at about 40° and 50°. These estimates can be refined with the `fzero` function,

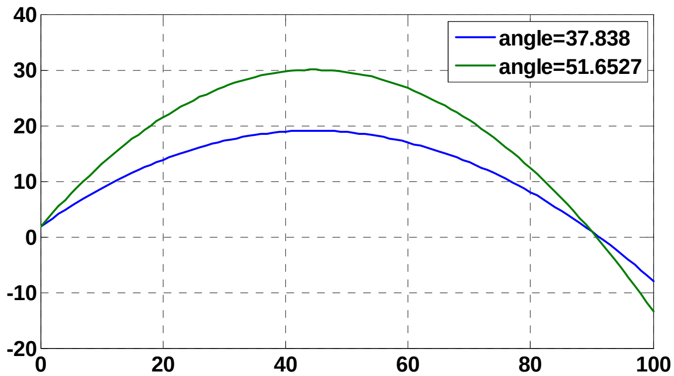
```
theta1 = fzero(fth,[30 45])
theta2 = fzero(fth,[45 60])
```

with the results

```
theta1 =
  37.837972746140331
theta2 =
  51.652744848882158
```

Therefore, two angles result in the desired outcome. We can develop plots of the two trajectories:

```
yth=@(x,th) tan(th*pi/180)*x-g/2/v0^2/cos(th*pi/180)^2*x.^2+y0;
xplot=[0:xdist];yplot=yth(xplot,theta1);yplot2=yth(xplot,theta2);
plot(xplot,yplot,xplot,yplot2,'--')
grid;legend(['angle=' num2str(theta1)],['angle=' num2str(theta2)])
```



Note that the lower angle would probably be preferred as the ball would arrive at the catcher sooner.

6.22 The equation to be solved is

$$f(h) = \pi R h^2 - \left(\frac{\pi}{3}\right) h^3 - V$$

Because this equation is easy to differentiate, the Newton-Raphson is the best choice to achieve results efficiently. It can be formulated as

$$x_{i+1} = x_i - \frac{\pi R x_i^2 - \left(\frac{\pi}{3}\right) x_i^3 - V}{2\pi R x_i - \pi x_i^2}$$

or substituting the parameter values,

$$x_{i+1} = x_i - \frac{\pi(3) x_i^2 - \left(\frac{\pi}{3}\right) x_i^3 - 30}{2\pi(3) x_i - \pi x_i^2}$$

The iterations can be summarized as

iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	3	26.54867	28.27433	
1	2.061033	0.866921	25.50452	45.558%
2	2.027042	0.003449	25.30035	1.677%
3	2.026906	5.68E-08	25.29952	0.007%

Thus, after only three iterations, the root is determined to be 2.026906 with an approximate relative error of 0.007%.

6.23

```
>> format short g
>> r = [-2 -5 6 4 8];
>> a = poly(r)
a =
     1      -11      -12      356      -304     -1920
>> polyval(a,1)
ans =
        -1890
```

```
>> b = poly([-2 -5])
b =
     1     7    10
>> [q,r] = deconv(a,b)
q =
     1   -18   104  -192
r =
     0     0     0     0     0     0
>> x = roots(q)
x =
    8
    6
    4
>> a = conv(q,b)
a =
     1    -11  -12   356  -304  -1920
>> x = roots(a)
x =
    8
    6
    4
   -5
   -2
>> a = poly(x)
a =
        1      -11      -12      356     -304    -1920
```

6.24

```
>> a = [1 9 26 24];
>> r = roots(a)
r =
   -4.0000
   -3.0000
   -2.0000
>> a = [1 15 77 153 90];
>> r = roots(a)
r =
   -6.0000
   -5.0000
   -3.0000
   -1.0000
```

Therefore, the transfer function is

$$G(s) = \frac{(s+4)(s+3)(s+2)}{(s+6)(s+5)(s+3)(s+1)}$$

6.25 The equation can be rearranged so it can be solved with fixed-point iteration as

$$H_{i+1} = \frac{(Qn)^{3/5}(B+2H_i)^{2/5}}{BS^{3/10}}$$

Substituting the parameters gives,

$$H_{i+1} = 0.2062129(20+2H_i)^{2/5}$$

This formula can be applied iteratively to solve for H. For example, using and initial guess of $H_0 = 0$, the first iteration gives

$$H_1 = 0.2062129(20 + 2(0))^{2/5} = 0.683483$$

Subsequent iterations yield

i	*H*	ε_a
0	0.000000	
1	0.683483	100.000%
2	0.701799	2.610%
3	0.702280	0.068%
4	0.702293	0.002%

Thus, the process converges on a depth of 0.7023. We can prove that the scheme converges for all initial guesses greater than or equal to zero by differentiating the equation to give

$$g' = \frac{0.16497}{(20 + 2H)^{3/5}}$$

This function will always be less than one for $H \geq 0$. For example, if $H = 0$, $g' = 0.027339$. Because H is in the denominator, all values greater than zero yield even smaller values. Thus, the convergence criterion that $|g'| < 1$ always holds.

6.26 This problem can be solved in a number of ways. One approach involves using the modified secant method. This approach is feasible because the Swamee-Jain equation provides a sufficiently good initial guess that the method is always convergent for the specified parameter bounds. The following functions implement the approach:

```
function ffact = prob0626(eD,ReN)
% prob0626: friction factor with Colebrook equation
%   ffact = prob0626(eD,ReN):
%      uses modified secant equation to determine the friction factor
%      with the Colebrook equation
% input:
%   eD = e/D
%   ReN = Reynolds number
% output:
%   ffact = friction factor
maxit=100;es=1e-8;delta=1e-5;
iter = 0;
% Swamee-Jain equation:
xr = 1.325 / (log(eD / 3.7 + 5.74 / ReN ^ 0.9)) ^ 2;
% modified secant method
while (1)
  xrold = xr;
  xr = xr - delta*xr*func(xr,eD,ReN)/(func(xr+delta*xr,eD,ReN)...
                                                -func(xr,eD,ReN));
  iter = iter + 1;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  if ea <= es | iter >= maxit, break, end
end
ffact = xr;

function ff=func(f,eD,ReN)
ff = 1/sqrt(f) + 2*log10(eD/3.7 + 2.51/ReN/sqrt(f));
```

Here are implementations for the extremes of the parameter range:

```
>> prob0626(0.00001,4000)
ans =
    0.0399
>> prob0626(0.05,4000)
ans =
    0.0770
>> prob0626(0.00001,1e7)
ans =
    0.0090
>> prob0626(0.05,1e7)
ans =
    0.0716
```

6.27 The Newton-Raphson method can be set up as

$$x_{i+1} = x_i - \frac{e^{-0.5x_i}(4-x_i)-2}{-e^{-0.5x_i}(3-0.5x_i)}$$

(a)

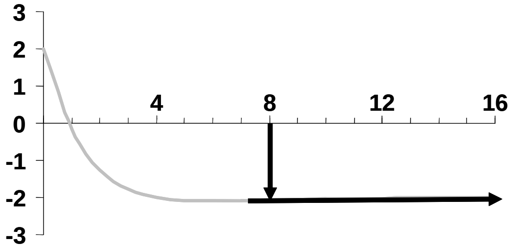
i	*x*	*f*(*x*)	*f'*(*x*)	$\lvert\varepsilon_a\rvert$
0	2	-1.26424	-0.73576	
1	0.281718	1.229743	-2.48348	609.93%
2	0.776887	0.18563	-1.77093	63.74%
3	0.881708	0.006579	-1.64678	11.89%
4	0.885703	9.13E-06	-1.64221	0.45%
5	0.885709	1.77E-11	-1.6422	0.00%
6	0.885709	0	-1.6422	0.00%

(b) The case does not work because the derivative is zero at $x_0 = 6$.

(c)

i	*x*	*f*(*x*)	*f'*(*x*)
0	8	-2.07326	0.018316
1	121.1963	-2	2.77E-25
2	7.21E+24	-2	0

This guess breaks down because, as depicted in the following plot, the near zero, positive slope sends the method away from the root.



6.28 The optimization problem involves determining the root of the derivative of the function. The derivative is the following function,

$$f'(x) = -12x^5 - 6x^3 + 10$$

The Newton-Raphson method is a good choice for this problem because

- The function is easy to differentiate
- It converges very rapidly

The Newton-Raphson method can be set up as

$$x_{i+1} = x_i - \frac{-12x_i^5 - 6x_i^3 + 10}{-60x_i^4 - 18x_i^2}$$

First iteration:

$$x_1 = x_0 - \frac{-12(1)^5 - 6(1)^3 + 10}{-60(1)^4 - 18(1)^2} = 0.897436$$

$$\varepsilon_a = \left|\frac{0.897436 - 1}{0.897436}\right| \times 100\% = 11.43\%$$

Second iteration:

$$x_1 = x_0 - \frac{-12(0.897436)^5 - 6(0.897436)^3 + 10}{-60(0.897436)^4 - 18(0.897436)^2} = 0.872682$$

$$\varepsilon_a = \left|\frac{0.872682 - 0.897436}{0.872682}\right| \times 100\% = 2.84\%$$

Since $\varepsilon_a < 5\%$, the solution can be terminated.

6.29 Newton-Raphson is the best choice because:

- You know that the solution will converge. Thus, divergence is not an issue.
- Newton-Raphson is generally considered the fastest method
- You only require one guess
- The function is easily differentiable

To set up the Newton-Raphson first formulate the function as a roots problem and then differentiate it

$$f(x) = e^{0.5x} - 5 + 5x$$
$$f'(x) = 0.5e^{0.5x} + 5$$

These can be substituted into the Newton-Raphson formula

$$x_{i+1} = x_i - \frac{e^{0.5x_i} - 5 + 5x_i}{0.5e^{0.5x_i} + 5}$$

First iteration:

$$x_1 = 0.7 - \frac{e^{0.5(0.7)} - 5 + 5(0.7)}{0.5e^{0.5(0.7)} + 5} = 0.7 - \frac{-0.08093}{5.7095} = 0.714175$$

$$\varepsilon_a = \left|\frac{0.714175 - 0.7}{0.714175}\right| \times 100\% = 1.98\%$$

Therefore, only one iteration is required.

6.30 (a)

```
function [b,fb] = fzeronew(f,xl,xu,varargin)
% fzeronew: Brent root location zeroes
% [b,fb] = fzeronew(f,xl,xu,p1,p2,...):
%   uses Brent's method to find the root of f
% input:
%   f = name of function
%   xl, xu = lower and upper guesses
%   p1,p2,... = additional parameters used by f
% output:
%   b = real root
%   fb = function value at root
if nargin<3,error('at least 3 input arguments required'),end
a = xl; b = xu; fa = f(a,varargin{:}); fb = f(b,varargin{:});
c = a; fc = fa; d = b - c; e = d;
while (1)
  if fb == 0, break, end
  if sign(fa) == sign(fb)              %If necessary, rearrange points
    a = c; fa = fc; d = b - c; e = d;
  end
  if abs(fa) < abs(fb)
    c = b; b = a; a = c;
    fc = fb; fb = fa; fa = fc;
  end
  m = 0.5 * (a - b);      %Termination test and possible exit
  tol = 2 * eps * max(abs(b), 1);
  if abs(m) <= tol | fb == 0.
    break
  end
  %Choose open methods or bisection
  if abs(e) >= tol & abs(fc) > abs(fb)
    s = fb / fc;
    if a == c                              %Secant method
      p = 2 * m * s; q = 1 - s;
    else                 %Inverse quadratic interpolation
      q = fc / fa; r = fb / fa;
      p = s * (2 * m * q * (q - r) - (b - c) * (r - 1));
      q = (q - 1) * (r - 1) * (s - 1);
    end
    if p > 0, q = -q; else p = -p; end;
    if 2 * p < 3 * m * q - abs(tol * q) & p < abs(0.5 * e * q)
      e = d; d = p / q;
    else
      d = m; e = m;
    end
  else                                          %Bisection
    d = m; e = m;
  end
  c = b; fc = fb;
  if abs(d) > tol, b = b + d; else b = b - sign(b - a) * tol; end
  fb = f(b,varargin{:});
end
```

(b)

```
>> [x,fx] = fzeronew(@(x,n) x^n-1,0,1.3,10)
x =
     1
fx =
     0
```