CHAPTER 7

7.1 In a fashion similar to Example 7.1, $n = 4$, $a_0 = –20$, $a_1 = 3$, $a_2 = 14.5$, $a_3 = –7.5$, $a_4 = 1$ and $t = 2$. These can be used to compute

$r = a_4 = 1$
$a_4 = 0$

For $i = 3$,
$s = a_3 = –7.5$
$a_3 = r = 1$
$r = s + rt = –7.5 + 1(2) = –5.5$

For $i = 2$,
$s = a_2 = 14.5$
$a_2 = r = –5.5$
$r = s + rt = 14.5–5.5(2) = 3.5$

For $i = 1$,
$s = a_1 = 3$
$a_1 = r = 3.5$
$r = s + rt = 3+3.5(2) = 10$

For $i = 0$,
$s = a_0 = –20$
$a_0 = r = 10$
$r = s + rt = –20+10(2) = 0$

Therefore, the quotient is $x^3 – 5.5x^2 + 3.5x +10$ with a remainder of zero. Thus, 2 is a root. This result can be easily verified with MATLAB,

```
>> a = [1 -7.5 14.5 3 -20];
>> b = [1 -2];
>> [d,e] = deconv(a,b)

d =
    1.0000   -5.5000    3.5000   10.0000
e =
     0     0     0     0     0
```

7.2 In a fashion similar to Example 7.1, $n = 5$, $a_0 = 10$, $a_1 = –7$, $a_2 = –6$, $a_3 = 1$, $a_4 = –5$, $a_5 = 1$, and $t = 2$. These can be used to compute

$r = a_4 = 1$
$a_4 = 0$

For $i = 4$,
$s = a_4 = –5$
$a_3 = r = 1$

$r = s + rt = –5 + 1(2) = –3$

For $i = 3$,
$s = a_3 = 1$
$a_3 = r = –3$
$r = s + rt = 1 – 3(2) = –5$

For $i = 2$,
$s = a_2 = –6$
$a_2 = r = –5$
$r = s + rt = –6 – 5(2) = –16$

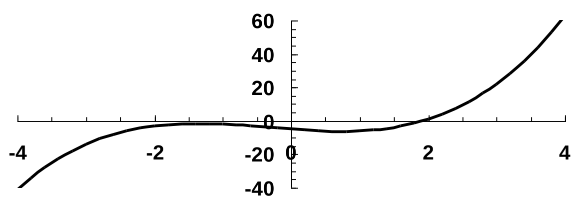
For $i = 1$,
$s = a_1 = –7$
$a_1 = r = –16$
$r = s + rt = –7 – 16(2) = –39$

For $i = 0$,
$s = a_0 = 10$
$a_0 = r = –39$
$r = s + rt = 10 – 39(2) = –68$

Therefore, the quotient is $x^4 – 3x^3 – 5x^2 – 16x – 39$ with a remainder of –68. Thus, 2 is not a root. This result can be easily verified with MATLAB,

```
>> a=[1 -5 1 -6 -7 10];
>> b = [1 -2];
>> [d,e] = deconv(a,b)

d =
     1    -3    -5   -16   -39
e =
     0     0     0     0     0   -68
```

7.3 (a) A plot indicates a root at about $x = 2$.



Try initial guesses of $x_0 = 1$, $x_1 = 1.5$, and $x_2 = 2.5$. Using the same approach as in Example 7.2,

<u>First iteration:</u>

$f(1) = –6$	$f(1.5) = –3.875$	$f(2.5) = 9.375$
$h_0 = 0.5$	$h_1 = 1$	

$\delta_0 = 4.25$ $\qquad$ $\delta_1 = 13.25$

$$a = \frac{13.25 - 4.25}{1 + 0.5} = 6 \qquad b = 6(1) + 13.25 = 19.25 \qquad c = 9.375$$

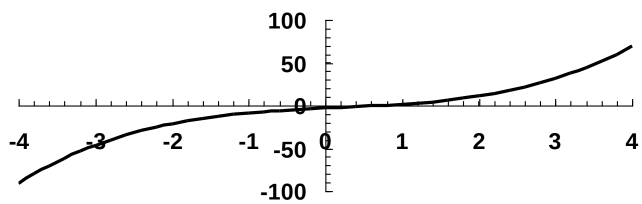
$$x_3 = 2.5 + \frac{-2(9.375)}{19.25 + \sqrt{19.25^2 - 4(6)(9.375)}} = 1.901244$$

$$\varepsilon_a = \left| \frac{1.901244 - 2.5}{1.901244} \right| \times 100\% = 31.49\%$$

The iterations can be continued as tabulated below:

i	x_3	ε_a
0	1.901244	31.4929%
1	1.919270	0.9392%
2	1.919639	0.0192%
3	1.919640	0.0000%

(b) A plot indicates a root at about $x = 0.7$.



Try initial guesses of $x_0 = 0.5$, $x_1 = 1$, and $x_2 = 1.5$. Using the same approach as in Example 7.2,

First iteration:

$f(0.5) = -1$ $\qquad$ $f(1) = 1.5$ $\qquad$ $f(1.5) = 5.25$

$h_0 = 0.5$ $\qquad$ $h_1 = 0.5$

$\delta_0 = 5$ $\qquad$ $\delta_1 = 7.5$

$$a = \frac{7.5 - 5}{0.5 + 0.5} = 2.5 \qquad b = 2.5(0.5) + 7.5 = 8.75 \qquad c = 5.25$$

$$x_3 = 1.5 + \frac{-2(5.25)}{8.75 + \sqrt{8.75^2 - 4(2.5)(5.25)}} = 0.731071$$

$$\varepsilon_a = \left| \frac{0.731071 - 1.5}{0.731071} \right| \times 100\% = 105.18\%$$

The iterations can be continued as tabulated below:

i	x_3	ε_a
0	0.731071	105.1785%
1	0.720767	1.4296%

2	0.721231	0.0643%
3	0.721230	0.0001%

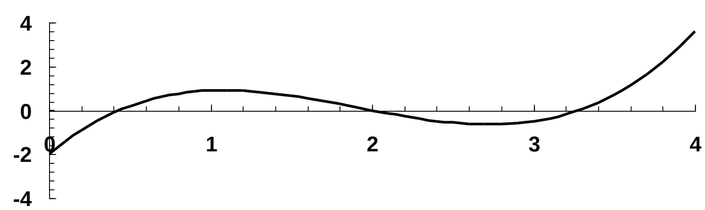
7.4 Here are MATLAB sessions to determine the roots:

(a)
```
>> a=[1 -1 3 -2];
>> roots(a)

ans =
   0.1424 + 1.6661i
   0.1424 - 1.6661i
   0.7152
```

(b)
```
>> a=[2 0 6 0 10];
>> roots(a)

ans =
  -0.6067 + 1.3668i
  -0.6067 - 1.3668i
   0.6067 + 1.3668i
   0.6067 - 1.3668i
```

(c)
```
>> a=[1 -2 6 -8 8];
>> roots(a)

ans =
  -0.0000 + 2.0000i
  -0.0000 - 2.0000i
   1.0000 + 1.0000i
   1.0000 - 1.0000i
```

7.5 (a) A plot suggests 3 real roots: 0.44, 2 and 3.3.



Try $r = 1$ and $s = -1$, and follow Example 7.3

1st iteration:
$\Delta r = 1.085$ $\Delta s = 0.887$
$r = 2.085$ $s = -0.1129$

2nd iteration:
$\Delta r = 0.4019$ $\Delta s = -0.5565$

$r = 2.487$ $\qquad s = –0.6694$

3rd iteration:
$\Delta r = –0.0605$ $\qquad \Delta s = –0.2064$
$r = 2.426$ $\qquad s = –0.8758$

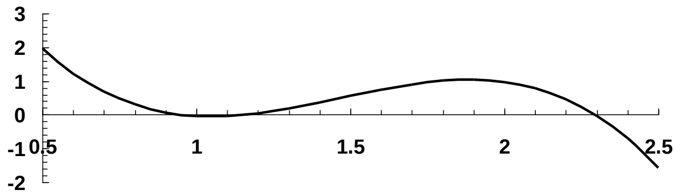
4th iteration:
$\Delta r = 0.00927$ $\qquad \Delta s = 0.00432$
$r = 2.436$ $\qquad s = –0.8714$

$$\text{root}_1 = \frac{2.436 + \sqrt{2.436^2 + 4(-0.8714)}}{2} = 2$$

$$\text{root}_2 = \frac{2.436 - \sqrt{2.436^2 + 4(-0.8714)}}{2} = 0.4357$$

The remaining $\text{root}_3 = 3.279$.

(b) Plot suggests 3 real roots at approximately 0.9, 1.2 and 2.3.



Try $r = 2$ and $s = –0.5$, and follow Example 7.3

1st iteration:
$\Delta r = 0.2302$ $\qquad \Delta s = –0.5379$
$r = 2.2302$ $\qquad s = –1.0379$

2nd iteration:
$\Delta r = –0.1799$ $\qquad \Delta s = –0.0422$
$r = 2.0503$ $\qquad s = –1.0801$

3rd iteration:
$\Delta r = 0.0532$ $\qquad \Delta s = –0.01641$
$r = 2.1035$ $\qquad s = –1.0966$

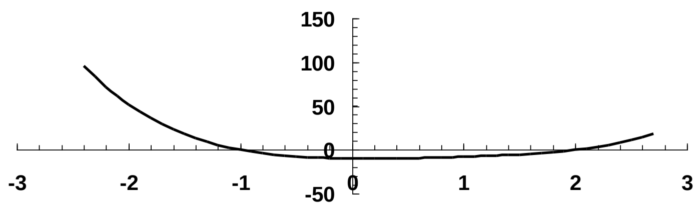
4th iteration:
$\Delta r = 0.00253$ $\qquad \Delta s = –0.00234$
$r = 2.106$ $\qquad s = –1.099$

$$\text{root}_1 = \frac{2.106 + \sqrt{2.106^2 + 4(-1.099)}}{2} = 1.1525$$

$$\text{root}_2 = \frac{2.106 - \sqrt{2.106^2 + 4(-1.099)}}{2} = 0.9535$$

The remaining root_3 = 2.2947

(c) Plot suggests 2 real roots at approximately –1 and 2.2. This means that there should also be 2 complex roots



Try r = –1 and s = 1, and follow Example 7.3

<u>1st iteration:</u>
Δr = 2.171 Δs = 3.947
r = 1.171 s = 4.947

<u>2nd iteration:</u>
Δr = –0.0483 Δs = –2.260
r = 1.123 s = 2.688

<u>3rd iteration:</u>
Δr = –0.0931 Δs = –0.6248
r = 1.030 s = 2.063

<u>4th iteration:</u>
Δr = –0.0288 Δs = –0.0616
r = 1 s = 2

$$\text{root}_1 = \frac{1 + \sqrt{1^2 + 4(2)}}{2} = 2$$

$$\text{root}_2 = \frac{1 - \sqrt{1^2 + 4(2)}}{2} = -1$$

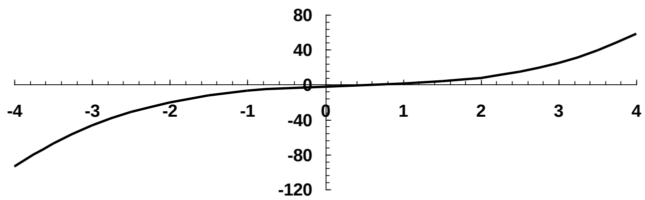
The remaining roots are 1 + 2i and 1 – 2i.

7.6 Here is a VBA program to implement the Müller algorithm and solve Example 7.2.

```
Option Explicit

Sub TestMull()
Dim maxit As Integer, iter As Integer
Dim h As Double, xr As Double, eps As Double
h = 0.1
xr = 5
eps = 0.001
maxit = 20
Call Muller(xr, h, eps, maxit, iter)
MsgBox "root = " & xr
MsgBox "Iterations: " & iter
End Sub

Sub Muller(xr, h, eps, maxit, iter)
Dim x0 As Double, x1 As Double, x2 As Double
Dim h0 As Double, h1 As Double, d0 As Double, d1 As Double
Dim a As Double, b As Double, c As Double
Dim den As Double, rad As Double, dxr As Double
x2 = xr
x1 = xr + h * xr
x0 = xr - h * xr
Do
  iter = iter + 1
  h0 = x1 - x0
  h1 = x2 - x1
  d0 = (f(x1) - f(x0)) / h0
  d1 = (f(x2) - f(x1)) / h1
  a = (d1 - d0) / (h1 + h0)
  b = a * h1 + d1
  c = f(x2)
  rad = Sqr(b * b - 4 * a * c)
  If Abs(b + rad) > Abs(b - rad) Then
    den = b + rad
  Else
    den = b - rad
  End If
  dxr = -2 * c / den
  xr = x2 + dxr
  If Abs(dxr) < eps * xr Or iter >= maxit Then Exit Do
  x0 = x1
  x1 = x2
  x2 = xr
Loop
End Sub

Function f(x)
f = x ^ 3 - 13 * x - 12
End Function
```

When this program is run, it yields the correct result of 4 in 3 iterations.

7.7 The plot suggests a real root at 0.7.



Using initial guesses of x_0 = 0.63, x_1 = 0.77 and x_2 = 0.7, the software developed in Prob. 7.6 yields a root of 0.715225 in 2 iterations.

7.8 Here is a VBA program to implement the Bairstow algorithm to solve Example 7.3.

```
Option Explicit

Sub PolyRoot()
Dim n As Integer, maxit As Integer, ier As Integer, i As Integer
Dim a(10) As Double, re(10) As Double, im(10) As Double
Dim r As Double, s As Double, es As Double
n = 5
a(0) = 1.25: a(1) = -3.875: a(2) = 2.125: a(3) = 2.75: a(4) = -3.5: a(5) = 1
maxit = 20
es = 0.0001
r = -1
s = -1
Call Bairstow(a(), n, es, r, s, maxit, re(), im(), ier)
For i = 1 To n
  If im(i) >= 0 Then
    MsgBox re(i) & " + " & im(i) & "i"
  Else
    MsgBox re(i) & " - " & Abs(im(i)) & "i"
  End If
Next i
End Sub

Sub Bairstow(a, nn, es, rr, ss, maxit, re, im, ier)
Dim iter As Integer, n As Integer, i As Integer
Dim r As Double, s As Double, ea1 As Double, ea2 As Double
Dim det As Double, dr As Double, ds As Double
Dim r1 As Double, i1 As Double, r2 As Double, i2 As Double
Dim b(10) As Double, c(10) As Double
r = rr
s = ss
n = nn
ier = 0
ea1 = 1
ea2 = 1
Do
  If n < 3 Or iter >= maxit Then Exit Do
  iter = 0
  Do
    iter = iter + 1
    b(n) = a(n)
    b(n - 1) = a(n - 1) + r * b(n)
    c(n) = b(n)
    c(n - 1) = b(n - 1) + r * c(n)
    For i = n - 2 To 0 Step -1
      b(i) = a(i) + r * b(i + 1) + s * b(i + 2)
      c(i) = b(i) + r * c(i + 1) + s * c(i + 2)
```

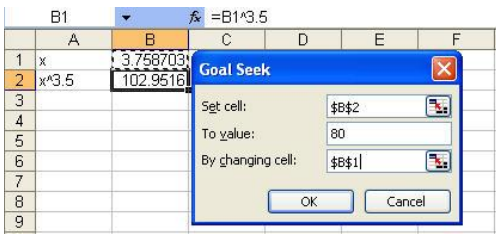
```
    Next i
    det = c(2) * c(2) - c(3) * c(1)
    If det <> 0 Then
      dr = (-b(1) * c(2) + b(0) * c(3)) / det
      ds = (-b(0) * c(2) + b(1) * c(1)) / det
      r = r + dr
      s = s + ds
      If r <> 0 Then ea1 = Abs(dr / r) * 100
      If s <> 0 Then ea2 = Abs(ds / s) * 100
    Else
      r = r + 1
      s = s + 1
      iter = 0
    End If
    If ea1 <= es And ea2 <= es Or iter >= maxit Then Exit Do
  Loop
  Call Quadroot(r, s, r1, i1, r2, i2)
  re(n) = r1
  im(n) = i1
  re(n - 1) = r2
  im(n - 1) = i2
  n = n - 2
  For i = 0 To n
    a(i) = b(i + 2)
  Next i
Loop
If iter < maxit Then
  If n = 2 Then
    r = -a(1) / a(2)
    s = -a(0) / a(2)
    Call Quadroot(r, s, r1, i1, r2, i2)
    re(n) = r1
    im(n) = i1
    re(n - 1) = r2
    im(n - 1) = i2
  Else
    re(n) = -a(0) / a(1)
    im(n) = 0
  End If
Else
  ier = 1
End If
End Sub

Sub Quadroot(r, s, r1, i1, r2, i2)
Dim disc
disc = r ^ 2 + 4 * s
If disc > 0 Then
  r1 = (r + Sqr(disc)) / 2
  r2 = (r - Sqr(disc)) / 2
  i1 = 0
  i2 = 0
Else
  r1 = r / 2
  r2 = r1
  i1 = Sqr(Abs(disc)) / 2
  i2 = -i1
End If
End Sub
```

When this program is run, it yields the correct result of –1, 0.5, 2, 1 + 0.5*i*, and 1 – 0.5*i*.

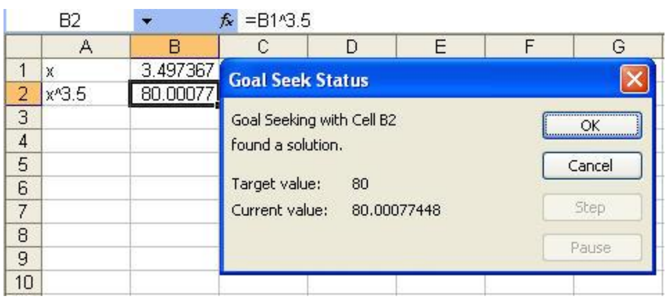
7.9 Using the software developed in Prob. 7.8 the following results should be generated for the three parts of Prob. 7.5:

(a) 3.2786, 2.0000, 0.4357
(b) 2.2947, 1.1525, 0.9535
(c) 2.0000, 1.0000 + 2.0000*i*, 1.0000 – 2.0000*i*, –1

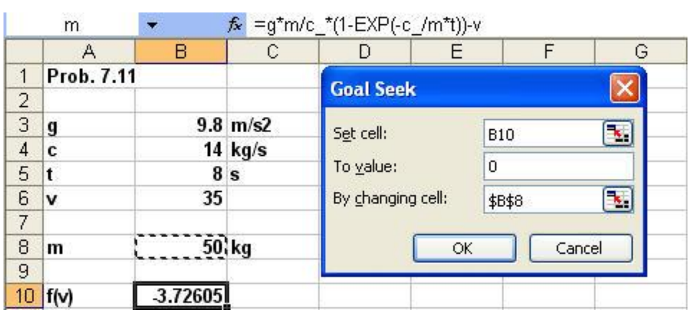
7.10 The goal seek set up is



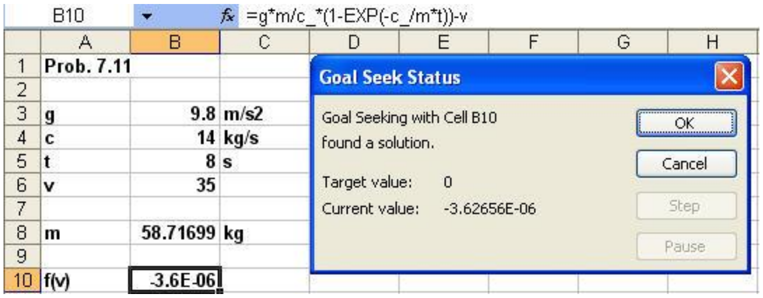
The result is



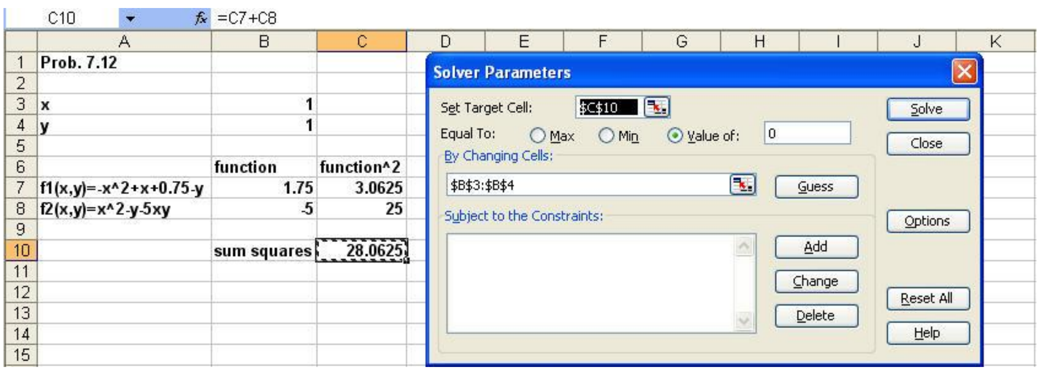
7.11 The goal seek set up is shown below. Notice that we have named the cells containing the parameter values with the labels in column A.



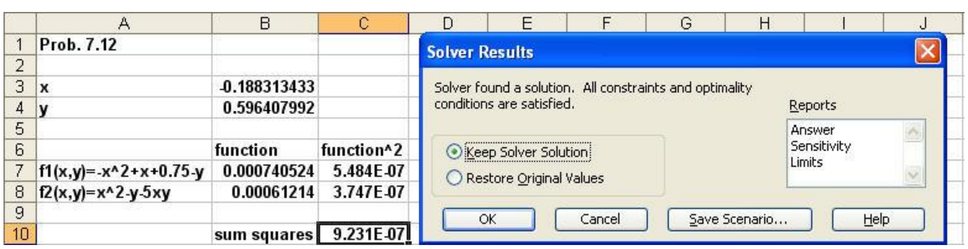
The result is 58.717 kg as shown here:



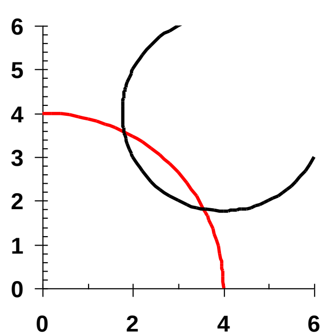
7.12 The Solver set up is shown below using initial guesses of $x = y = 1$. Notice that we have rearranged the two functions so that the correct values will drive both to zero. We then drive the sum of their squared values to zero by varying x and y. This is done because a straight sum would be zero if $f_1(x,y) = -f_2(x,y)$.



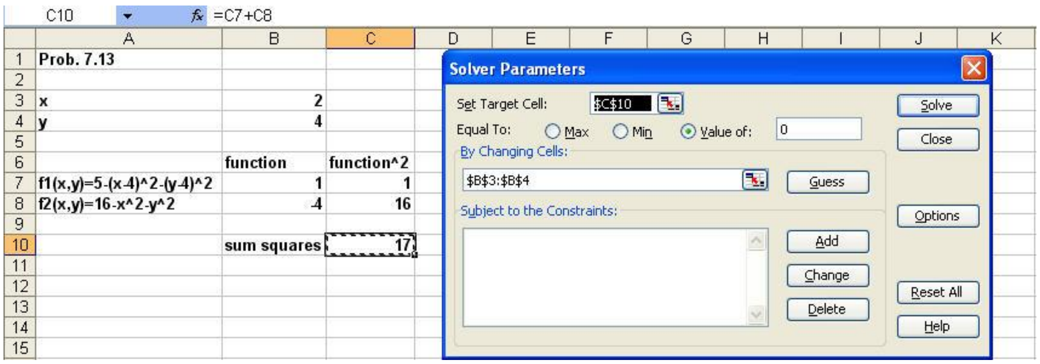
The result is



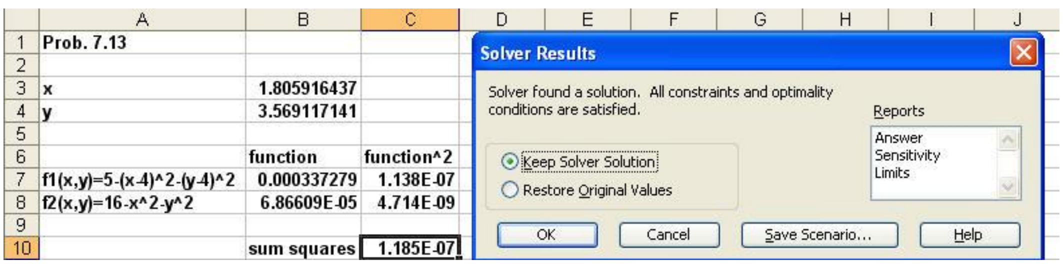
7.13 A plot of the functions indicates two real roots at about (1.8, 3.6) and (3.6, 1.8).



The Solver set up is shown below using initial guesses of (2, 4). Notice that we have rearranged the two functions so that the correct values will drive both to zero. We then drive the sum of their squared values to zero by varying x and y. This is done because a straight sum would be zero if $f_1(x,y) = -f_2(x,y)$.



The result is



For guesses of (4, 2) the result is (3.5691, 1.8059).

7.14 MATLAB session:

```
>> a = poly([4 -2 1 -5 7])
a =
     1    -5   -35   125   194  -280

>> polyval(a,1)
ans =
     0

>> polyder(a)
ans =
     5   -20  -105   250   194

>> b = poly([4 -2])
b =
     1    -2    -8

>> [d,e] = deconv(a,b)
d =
     1    -3   -33    35
```

```
e =
     0     0     0     0     0     0

>> roots(d)
ans =
    7.0000
   -5.0000
    1.0000

>> conv(d,b)
ans =
     1    -5   -35   125   194  -280

>> r = roots(a)
r =
    7.0000
   -5.0000
    4.0000
   -2.0000
    1.0000
```

7.15 MATLAB sessions:

Prob. 7.5a:

```
>> a=[.7 -4 6.2 -2];
>> roots(a)
ans =
    3.2786
    2.0000
    0.4357
```

Prob. 7.5b:

```
>> a=[-3.704 16.3 -21.97 9.34];
>> roots(a)
ans =
    2.2947
    1.1525
    0.9535
```

Prob. 7.5c:

```
>> a=[1 -3 5 -1 -10];
>> roots(a)
ans =
   2.0000
   1.0000 + 2.0000i
   1.0000 - 2.0000i
  -1.0000
```

7.16 Here is a program written in Fortran 90:

```
PROGRAM Root
Use IMSL !This establishes the link to the IMSL libraries
Implicit None  !forces declaration of all variables
Integer::nroot
```

```
Parameter(nroot=1)
Integer::itmax=50
Real::errabs=0.,errrel=1.E-5,eps=0.,eta=0.
Real::f,x0(nroot) ,x(nroot)
External f
Integer::info(nroot)
Print *, "Enter initial guess"
Read *, x0
Call ZReal(f,errabs,errrel,eps,eta,nroot,itmax,x0,x,info)
Print *, "root = ", x
Print *, "iterations = ", info
End

Function f(x)
Implicit None
Real::f,x
f = x**3-x**2+3.*x-2.
End
```

The output for Prob. 7.4*a* would look like

```
Enter initial guess
.5
 root =   0.7152252
 iterations =            6
Press any key to continue
```

The other parts of Probs 7.4 have complex roots and therefore cannot be evaluated with ZReal. The roots for Prob. 7.5 can be evaluated by changing the function and obtaining the results:

7.5 (a) 2, 0.4357, 3.279
7.5 (b) 1.1525, 0.9535, 2.295
7.5 (c) 2, –1

7.17 $x_2 = 0.62$, $x_1 = 0.64$, $x_0 = 0.60$

$h_0 = 0.64 - 0.60 = 0.04$
$h_1 = 0.62 - 0.64 = -0.02$

$$\delta_0 = \frac{60-20}{0.64-0.60} = 1000$$

$$\delta_1 = \frac{50-60}{0.62-0.64} = 500$$

$$a = \frac{\delta_1 - \delta_0}{h_1 + h_0} = \frac{500-1000}{-0.02+0.04} = 25000$$

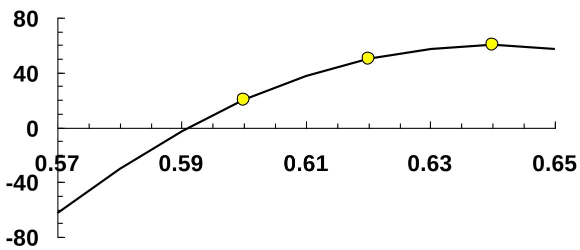
$$b = ah_1 + \delta_1 = -25000(-0.02) + 500 = 1000$$

$c = 50$

$$\sqrt{b^2 - 4ac} = \sqrt{1000^2 - 4(-25000)50} = 2449.49$$

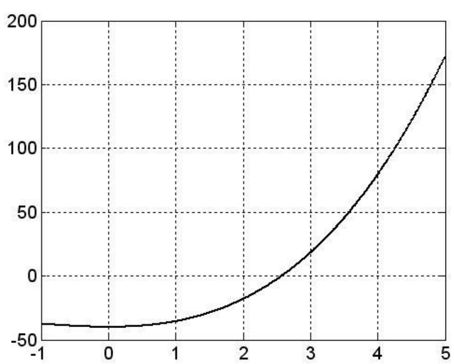
$$t_0 = 0.62 + \frac{-2(50)}{1000 + 2449.49} = 0.591$$

Therefore, the pressure was zero at 0.591 seconds. The result is graphically displayed below:

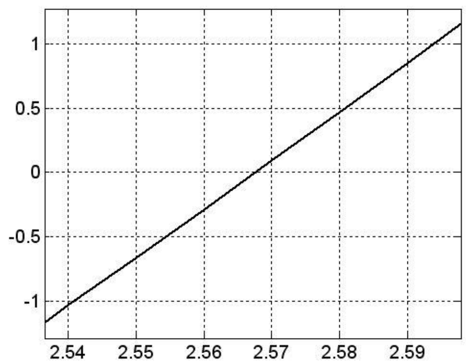


7.18 (a) First we will determine the root graphically

```
>> x=-1:0.01:5;
>> f=x.^3+3.5.*x.^2-40;
>> plot(x,f);grid
```



The zoom in tool can be used several times to home in on the root. For example, as shown in the following plot, a real root appears to occur at $x = 2.567$:

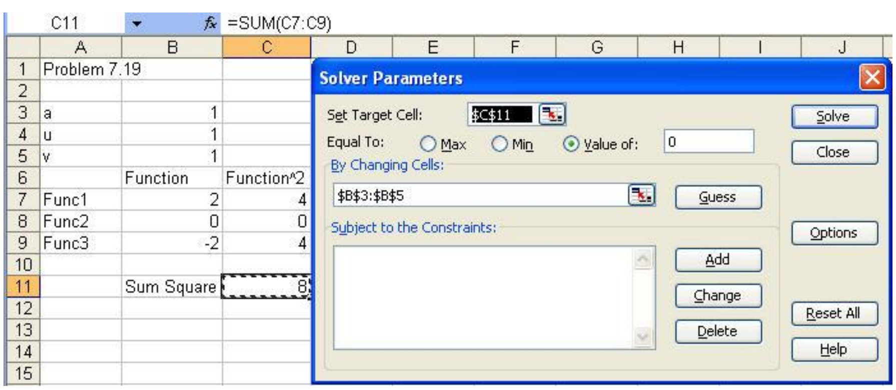


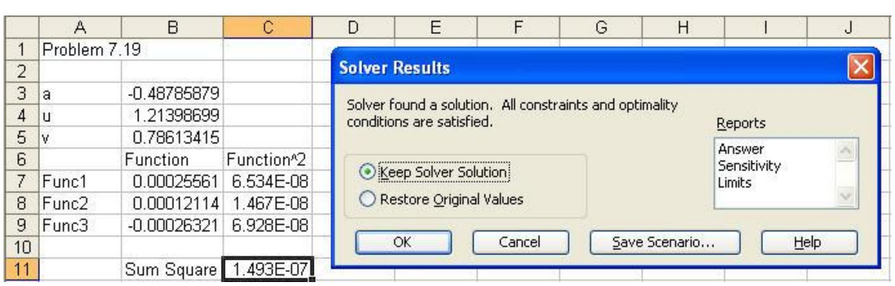
(b) The roots function yields both real and complex roots:

```
>> a=[1 3.5 0 -40];
>> roots(a)
ans =
  -3.0338 + 2.5249i
  -3.0338 - 2.5249i
   2.5676
```

7.19 (a) Excel Solver Solution: The 3 functions can be set up as a roots problems:

$$f_1(a,u,v) = a^2 - u^2 + 2v^2 = 0$$
$$f_2(a,u,v) = u + v - 2 = 0$$
$$f_3(a,u,v) = a^2 - 2a - u = 0$$





If you use initial guesses of $a = -1$, $u = 1$, and $v = -1$, the Solver finds another solution at $a = -1.6951$, $u = 6.2634$, and $v = -4.2636$

(b) Symbolic Manipulator Solution:

<u>MATLAB:</u>

```
>> syms a u v
>> S=solve(u^2-2*v^2-a^2,u+v-2,a^2-2*a-u);

>> double(S.a)
ans =
   3.0916 + 0.3373i
   3.0916 - 0.3373i
  -0.4879
  -1.6952

>> double(S.u)
ans =
   3.2609 + 1.4108i
   3.2609 - 1.4108i
   1.2140
   6.2641

>> double(S.v)
ans =
  -1.2609 - 1.4108i
  -1.2609 + 1.4108i
   0.7860
  -4.2641
```

<u>Mathcad:</u>

Problem 7.19 (Mathcad)

$f(a,u,v) := u^2 - 2 \cdot v^2 - a^2$ $g(a,u,v) := u + v - 2$ $h(a,u,v) := a^2 - 2 \cdot a - u$

$a := -1$ $u := 1$ $v := 1$ Initial Guesses

Given

$f(a,u,v) = 0$ $g(a,u,v) = 0$ $h(a,u,v) = 0$

$$\mathrm{Find}(a,u,v) = \begin{pmatrix} -0.4879 \\ 1.214 \\ 0.786 \end{pmatrix}$$

$a := -1$ $u := 6$ $v := -4$

Given

$f(a,u,v) = 0$ $g(a,u,v) = 0$ $h(a,u,v) = 0$

$$\text{Find}(a,u,v) = \begin{pmatrix} -1.6952 \\ 6.2641 \\ -4.2641 \end{pmatrix}$$

$a := 3 + i$ $\quad$ $u := 3 + i$ $\quad$ $v := -1 - i$

Given

$f(a,u,v) = 0$ $\quad$ $g(a,u,v) = 0$ $\quad$ $h(a,u,v) = 0$

$$\text{Find}(a,u,v) = \begin{pmatrix} 3.0916 + 0.3373i \\ 3.2609 + 1.4108i \\ -1.2609 - 1.4108i \end{pmatrix}$$

Therefore, we see that the two real-valued solutions for *a*, *u*, and *v* are (–0.4879, 1.2140, 0.7860) and (–1.6952, 6.2641, –4.2641). In addition, MATLAB and Mathcad also provide the complex solutions as well.

7.20 MATLAB can be used to determine the roots of the numerator and denominator:

```
>> n=[1 12.5 50.5 66];
>> roots(n)
ans =
   -5.5000
   -4.0000
   -3.0000

>> d=[1 19 122 296 192];
>> roots(d)
ans =
   -8.0000
   -6.0000
   -4.0000
   -1.0000
```

The transfer function can be written as

$$G(s) = \frac{(s+5.5)(s+4)(s+3)}{(s+8)(s+6)(s+4)(s+1)}$$

7.21

```
function root = bisection(func,xl,xu,es,maxit)
% root = bisection(func,xl,xu,es,maxit):
%   uses bisection method to find the root of a function
% input:
%   func = name of function
%   xl, xu = lower and upper guesses
%   es = (optional) stopping criterion (%)
%   maxit = (optional) maximum allowable iterations
% output:
%   root = real root
```

```
if func(xl)*func(xu)>0  %if guesses do not bracket a sign change
  disp('no bracket')    %display an error message
  return                %and terminate
end
% 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
% bisection
iter = 0;
xr = xl;
while (1)
  xrold = xr;
  xr = (xl + xu)/2;
  iter = iter + 1;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  test = func(xl)*func(xr);
  if test < 0
    xu = xr;
  elseif test > 0
    xl = xr;
  else
    ea = 0;
  end
  if ea <= es | iter >= maxit, break, end
end
root = xr;
```

The following is a MATLAB session that uses the function to solve Example 5.3 with ε_s = 0.0001.

```
>> fcd=inline('(9.81*68.1/cd)*(1-exp(-0.146843*cd))-40','cd');
>> format long
>> bisection(fcd,5,15,0.0001)
ans =
  14.80114936828613
```

7.22

```
function root = falsepos(func,xl,xu,es,maxit)
% falsepos(func,xl,xu,es,maxit):
%   uses the false position method to find the root of the function func
% input:
%   func = name of function
%   xl, xu = lower and upper guesses
%   es = (optional) stopping criterion (%) (default = 0.001)
%   maxit = (optional) maximum allowable iterations (default = 50)
% output:
%   root = real root

if func(xl)*func(xu)>0  %if guesses do not bracket a sign change
  error('no bracket')    %display an error message and terminate
end
% default values
if nargin<5, maxit=50; end
if nargin<4, es=0.001; end
% false position
iter = 0;
xr = xl;
while (1)
  xrold = xr;
```

```
  xr = xu - func(xu)*(xl - xu)/(func(xl) - func(xu));
  iter = iter + 1;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  test = func(xl)*func(xr);
  if test < 0
    xu = xr;
  elseif test > 0
    xl = xr;
  else
    ea = 0;
  end
  if ea <= es | iter >= maxit, break, end
end
root = xr;
```

The following is a MATLAB session that uses the function to solve Example 5.5:

```
>> fcd=inline('(9.81*68.1/cd)*(1-exp(-0.146843*cd))-40','cd');
>> format long
>> falsepos(fcd,5,15,0.0001)
ans =
  14.80114660933235
```

7.23

```
function root = newtraph(func,dfunc,xr,es,maxit)
% root = newtraph(func,dfunc,xguess,es,maxit):
%   uses Newton-Raphson method to find the root of a function
% input:
%   func = name of function
%   dfunc = name of derivative of function
%   xguess = initial guess
%   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
% Newton-Raphson
iter = 0;
while (1)
  xrold = xr;
  xr = xr - func(xr)/dfunc(xr);
  iter = iter + 1;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  if ea <= es | iter >= maxit, break, end
end
root = xr;
```

The following is a MATLAB session that uses the function to solve Example 6.3 with ε_s = 0.0001.

```
>> format long
>> f=inline('exp(-x)-x','x');
>> df=inline('-exp(-x)-1','x');
>> newtraph(f,df,0)
ans =
   0.56714329040978
```

7.24

```
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 Example 6.6:

```
>> format long
>> f=inline('exp(-x)-x','x');
>> secant(f,0,1)
ans =
   0.56714329040970
```

7.25

```
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 Example 6.8:

```
>> format long
>> f=inline('exp(-x)-x','x');
>> modsec(f,1,0.01)
ans =
   0.56714329027265
```