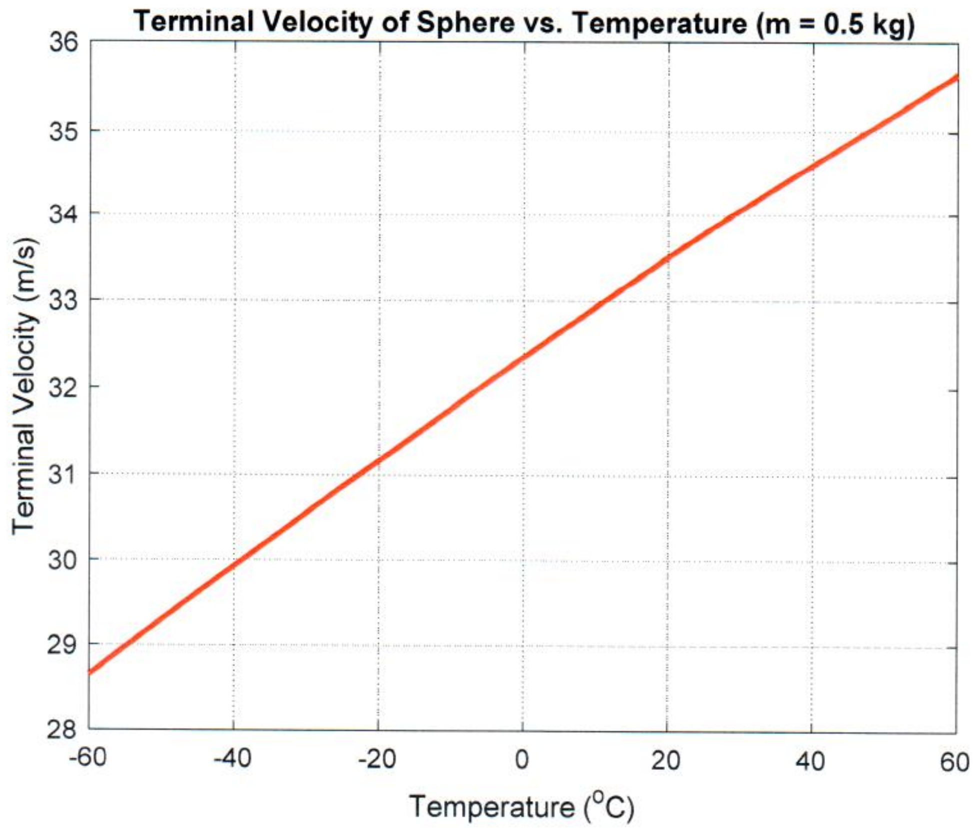
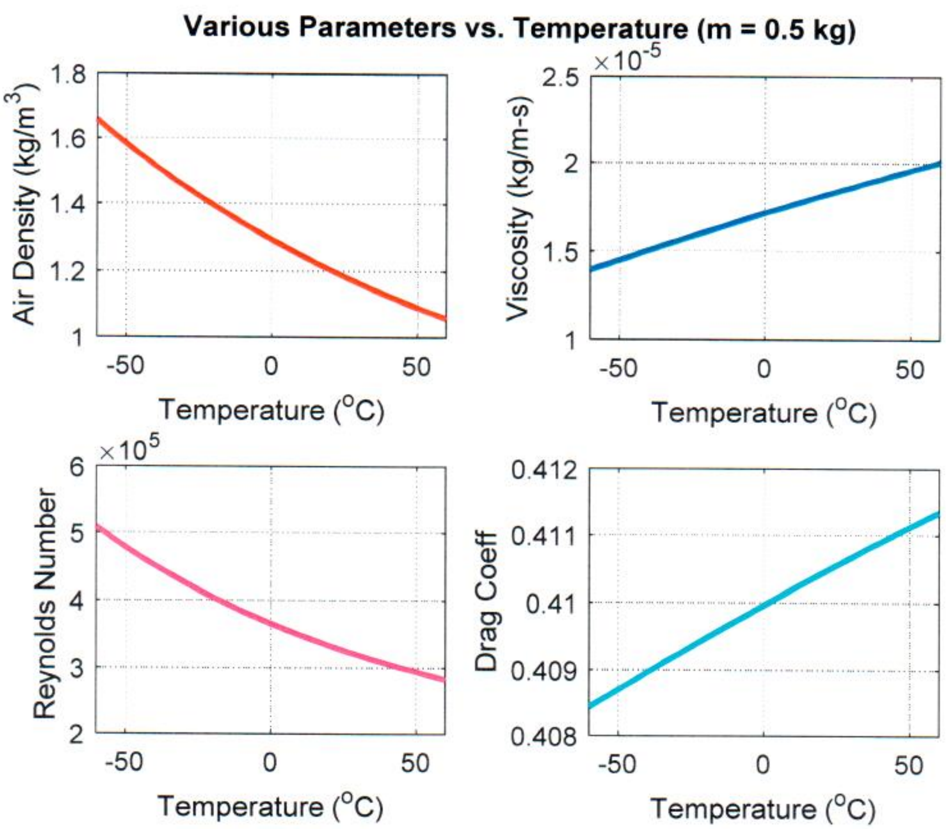
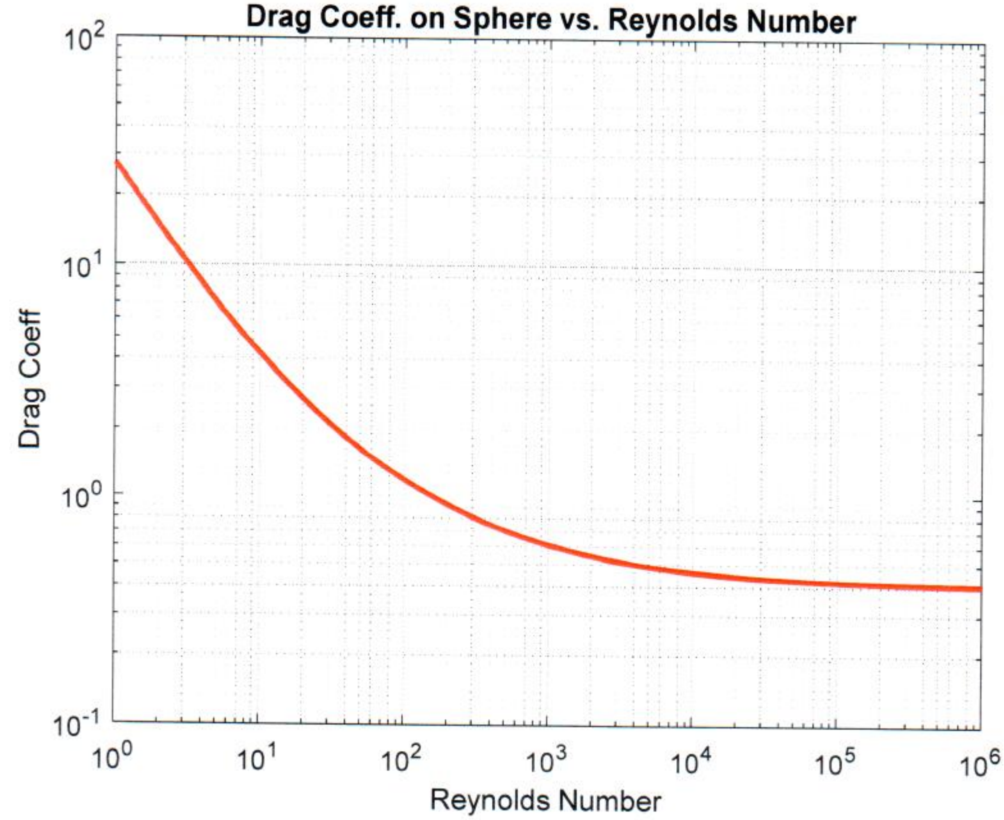
```
>> compare_methods_5

    Part a:
     Given radius =          3.000 ft
     Computed volume =  2326.412 gal


    Part b:
     Given tank's volume =  2500.000 gal

  Results for tank radius  ==>   Bisection       Secant   Matlab's FZERO

  A zero occurs at r =           3.0926026    3.0926026    3.0926026 ft
  The value of f(r) is =         0.0000006    0.0000000    0.0000000 ft^3
  # of function evaluations =        29            7             7
```

```
%
%   COMPARE_METHODS_5.M   Compare three root finding methods for relative efficiency
%
%   This project involves comparing the efficiency of the Bisection and Secant
%   Methods relative to Matlab's built-in FZERO function for finding the real roots
%   of nonlinear equations.
%
%   This code also illustrates the difference between explicit and implicit equations.
%   An expression that relates the radius, r, and volume, V, of a particular tank
%   was given in the problem description.  Given r, V can be easily determined via
%   an explicit evaluation of the given equation, V = f(r).
%   However, if V is given, we need to use a root finding routine to evaluate r,
%   since the relationship, r(V), is implicit  --  that is, we can't easily write
%   the equation in the form of r = f(V).  This implicit equation is used to
%   illustrate the relative efficiency of the various root finding methods:
%     Bisection Method              -- bisection.m
%     Secant Method                 -- secant.m
%     Matlab's built-in function    -- fzero.m
%
%   The function evaluations use anonymous functions since the equations are simple.
%
%   File written by J. R. White, UMass-Lowell  (last update:  Nov. 2017)
%

      clear all,  close all,   nfig = 0;
%
      L = 7;              % length of cylindrical portion of tank (ft)
      cf = 7.48;          % conversion factor (7.48 gal/ft^3)

%   Part a:  Find V given r  (explicit equation)
      Vr = @(r) pi*r^2*L + (4/3)*pi*r^3; % explicit function for tank's volume (ft^3)
      ra = 3;   Va = cf*Vr(ra);          % volume in gallons
      fprintf('\n    Part a: \n')
      fprintf('     Given radius =     %8.3f ft  \n',ra)
      fprintf('     Computed volume =  %8.3f gal \n',Va)
%
%   Part b:  Find radius given the tank volume  (implicit equation)
      Vbg = 2500;                  % desired tank vol (gal)
      Vb = Vbg/cf;                 % tank volume in ft^3
      fr = @(r) Vb - Vr(r);        % implicit relation for f(r) = Vdesired - V(r) = 0
      fprintf('\n\n    Part b: \n')
      fprintf('     Given tank''s volume =  %8.3f gal  \n ',Vbg)
%
%   call the various root finding methods and tabulate the results
      M = 50;   tol = 1e-6;   display = 0;
      [rb1,k1] = bisection(fr,ra,ra+1,tol,M,display);   f1 = fr(rb1);
      [rb2,k2] = secant(fr,ra,ra+1,tol,M,display);      f2 = fr(rb2);
      [rb3,f3,flag,output] = fzero(fr,[ra ra+1]);
      fprintf('\n  Results for tank radius  ==>   Bisection     Secant     Matlab''s↙
FZERO \n\n')
      fprintf('  A zero occurs at r =        %12.7f %12.7f %12.7f ft\n', ...
              rb1,rb2,rb3)
      fprintf('  The value of f(r) is =      %12.7f %12.7f %12.7f ft^3\n',f1,f2,f3)
      fprintf('  # of function evaluations =    %6i %12i  %12i\n', ...
              k1+2,k2+2,output.funcCount)
%
```


Terminal Velocity of Sphere vs. Temperature (m = 0.5 kg)
Terminal Velocity (m/s)
36
35
34
33
32
31
30
29
28
-60
-40
-20
0
20
40
60
Temperature (°C)


Various Parameters vs. Temperature (m = 0.5 kg)
Air Density (kg/m³)
1.8
1.6
1.4
1.2
1
-50
0
50
Temperature (°C)
Viscosity (kg/m-s)
×10⁻⁵
2.5
2
1.5
1
-50
0
50
Temperature (°C)
Reynolds Number
×10⁵
6
5
4
3
2
-50
0
50
Temperature (°C)
Drag Coeff
0.412
0.411
0.41
0.409
0.408
-50
0
50
Temperature (°C)



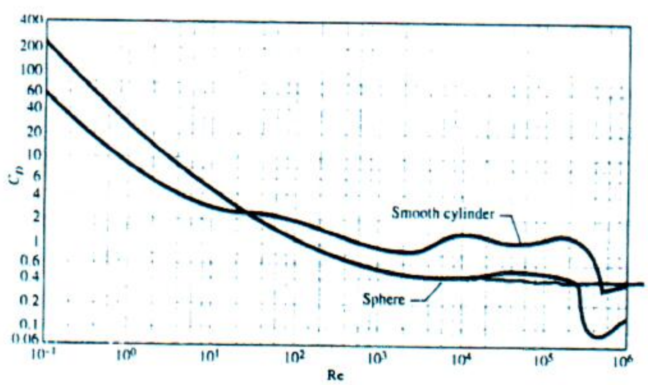


FIGURE 11–34
Average drag coefficient for cross-flow over a smooth circular cylinder and a smooth sphere.

From H. Schlichting, Boundary Layer Theory 7e. *Copyright © 1979 The McGraw-Hill Companies, Inc. Used by permission.*

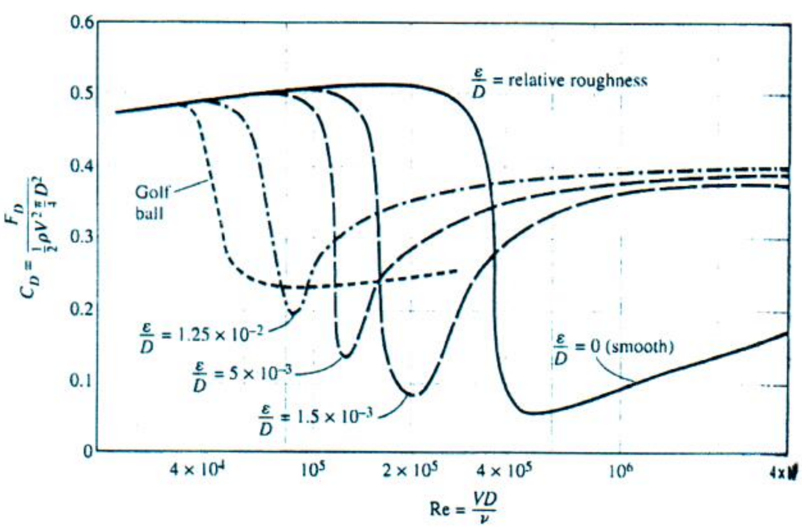


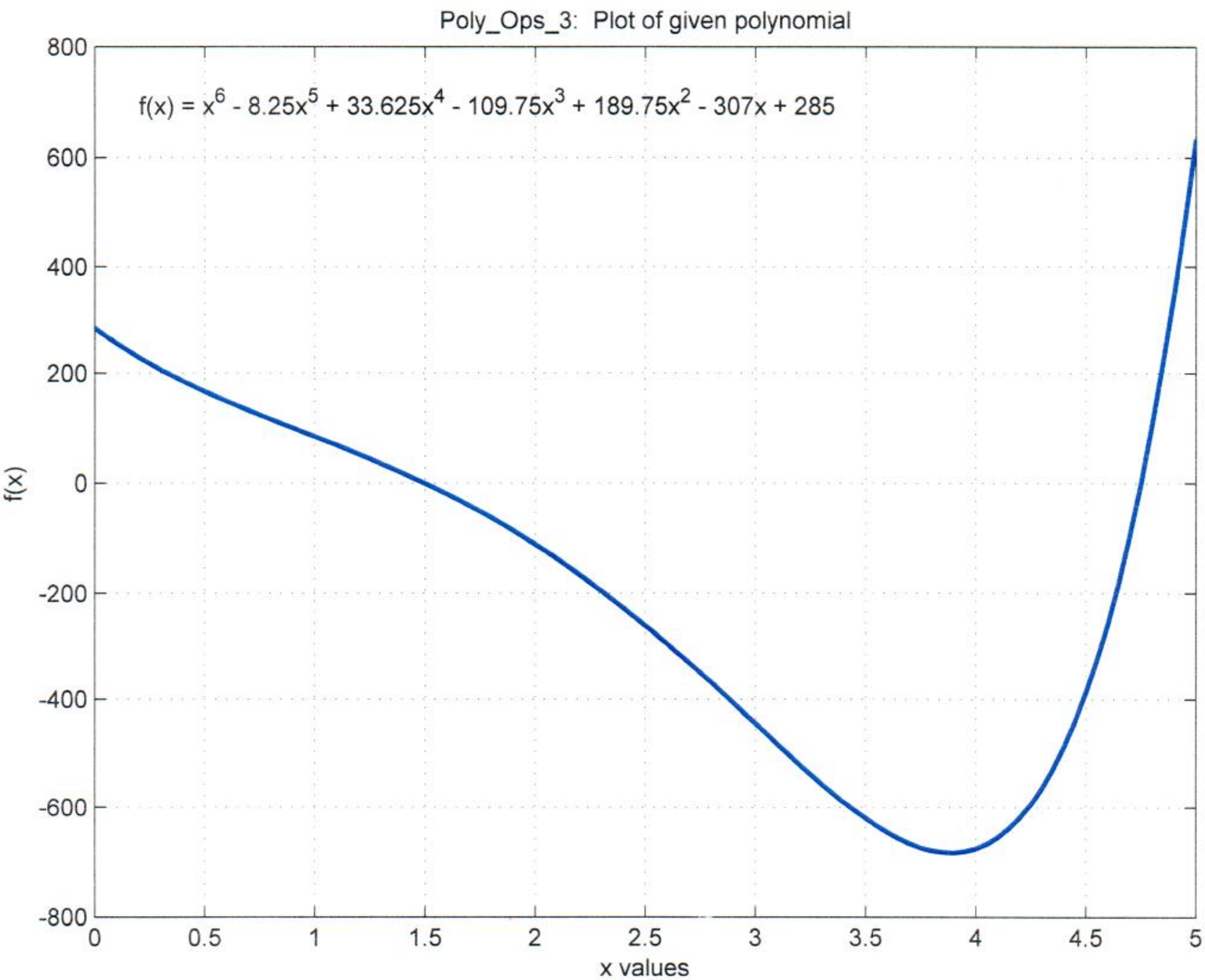
FIGURE 11–36
The effect of surface roughness on the drag coefficient of a sphere.

From Blevins (1984).

```
%
%   FALLING_SPHERE_1.M      Terminal Velocity of a Falling Sphere
%
%   This file solves for the terminal velocity of a sphere falling through the air.
%   This equilibrium condition is reached when the friction force due to the air just
%   balances the weight of the sphere.  The full relationship is a little complicated,
%   however, because the drag coefficent in the friction term is also a nonlinear
%   function of the velocity.  Thus, the system is written in implicit form, where we
%   ask the question, "What is v such that f(v) = 0?", with v being the desired
%   terminal velocity..
%
%   Also, since the properties of air are functions of the air temperature, we loop
%   over temperature to investigate how the terminal velocity varies with T (due
%   to the variation of density and visocity with T).
%
%   The goal of this problem is to illustrate how to solve nonlinear equations using
%   the built-in fzero routine in Matlab.  It also demonstates how, with a simple
%   for ... end loop, to do a parametric study for a single variable -- in this case,
%   we use a range of temperature values to show how the terminal velocity varies
%   with T.  For each value of T, a nonlinear force balance equation must be solved
%   (here we use fzero to do this) to determine the correct terminal velocity.
%
%   Some intermediate results that help validate the overall solution are also given.
%
%   The basic idea for this problem comes from Prob. 6.37 in the text, Numerical
%   Methods with Matlab, Implementation and Application, by G. Recktenwald, Prentice
%   Hall (2000).
%
%   File written by J. R. White, UMass-Lowell (last update: Nov. 2017)
%

      clear all;   close all;   nfig = 0;
%
%   set parameters for the problem
      m = 0.5;                   % mass of sphere (kg)
      d = 0.15;                  % diameter of sphere (m)
      g = 9.8;                   % gravitational acceleration (m/s^2)
      W = m*g;                   % weight of sphere (N)
      A = pi*d*d/4;              % frontal area of sphere (m^2)
      P = 101300;                % air pressure (N/m^2)
      R = 287.0;                 % gas constant for air (J/kg-K)
      Tc = -60:5:60;             % range of temperature values (C)
      Tk = Tc + 273.15;          % range of temperature values (K)
      b = [2.156954157e-14 -5.332634033e-11 7.477905983e-8 2.527878788e-7]; % mu(T) ↙
coeffs
%
%   find terminal velocity at each temperature
      NT = length(Tc);                    % number of temps
      v = zeros(size(Tc));                % allocate space for storage of velocities
      vg = 100;                           % initial guess of v for first value of T
      fv = @(v) falling_sphere_1a(v,Tk(1),P,R,b,d,A,W);
      v(1) = fzero(fv,vg);                % v for 1st T
      for n = 2:NT
        fv = @(v) falling_sphere_1a(v,Tk(n),P,R,b,d,A,W);
        v(n) = fzero(fv,v(n-1));          % v for all T
      end
```

```
%
%   plot v vs Tc
      nfig = nfig+1;   figure(nfig)
      plot(Tc,v,'r-','LineWidth',2), grid on
      title(['Terminal Velocity of Sphere vs. Temperature (m = ', ...
             num2str(m),' kg)'])
      xlabel('Temperature (^oC)'),ylabel('Terminal Velocity (m/s)')
%
%   evaluate and plot intermediate results (note 'dot arithmetic' where appropriate)
      den = P./(R*Tk);                       % air density at given temperature
      mu = polyval(b,Tk);                    % dynamic viscosity
      Re = den.*v*d./mu;                     % Reynolds number
      cd = 24./Re + 6./(1+sqrt(Re)) + 0.4;   % drag coefficient
%
      nfig = nfig+1;   figure(nfig)
      subplot(2,2,1),plot(Tc,den,'r-','LineWidth',2), grid on
      rr = axis;  rr(1:2) = [-60 60];  axis(rr);
      title(['Various Parameters vs. Temperature (m = ', ...
             num2str(m),' kg)'])
      xlabel('Temperature (^oC)'),ylabel('Air Density (kg/m^3)')
      subplot(2,2,2),plot(Tc,mu,'b-','LineWidth',2), grid on
      rr = axis;  rr(1:2) = [-60 60];  axis(rr);
      xlabel('Temperature (^oC)'),ylabel('Viscosity (kg/m-s)')
      subplot(2,2,3),plot(Tc,Re,'m-','LineWidth',2), grid on
      rr = axis;  rr(1:2) = [-60 60];  axis(rr);
      xlabel('Temperature (^oC)'),ylabel('Reynolds Number ')
      subplot(2,2,4),plot(Tc,cd,'c-','LineWidth',2), grid on
      rr = axis;  rr(1:2) = [-60 60];  axis(rr);
      xlabel('Temperature (^oC)'),ylabel('Drag Coeff')
%
%   as a final task here, let's plot the drag coefficient correlation given over a
%   wide range of Re values just to compare it to the results given in many Fluid
%   Mechanics texts...
      Re = logspace(0,6,1000);
      cd = 24./Re + 6./(1+sqrt(Re)) + 0.4;
      nfig = nfig+1;   figure(nfig)
      loglog(Re,cd,'r-','LineWidth',2),grid on
      title('Drag Coeff. on Sphere vs. Reynolds Number')
      xlabel('Reynolds Number'),ylabel('Drag Coeff')
%
%   end of problem
```

```
%
%   FALLING_SPHERE_1A.M   fzero function file for use with FALLING_SPHERE_1.M
%
%   This file simply evaluates a function of the form f(v) = W - Fd.  It is used
%   by the fzero routine to find the value v such that f(v) = 0.  The function
%   actually represents a force balance on the sphere.  The desired terminal
%   velocity, v, occurs when the downward force of gravity, W, is balanced by the
%   upward drag force, Fd.
%
%   File written by J. R. White, UMass-Lowell (last update: Nov. 2017)
%
      function f = falling_sphere_1a(v,T,P,R,b,d,A,W)
      den = P/(R*T);                          % air density at given temperature
      mu = polyval(b,T);                      % dynamic viscosity
      Re = den*v*d/mu;                        % Reynolds number
      cd = 24/Re + 6/(1+sqrt(Re)) + 0.4;      % drag coefficient
      Fd = 0.5*cd*den*v*v*A;                  % drag force
      f = W - Fd;                             % measure of balance
%
%   end of function
```


Poly_Ops_3: Plot of given polynomial
f(x) = x6 - 8.25x5 + 33.625x4 - 109.75x3 + 189.75x2 - 307x + 285
800
600
400
200
0
-200
-400
-600
-800
f(x)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
x values

```
>> poly_ops_3

  The real roots of f(x) are:   1.50000   4.75000
  The two linear factors are given by p1  = [   1.000    -1.500]
                                      p2  = [   1.000    -4.750]
  Multiplication of these gives pp  = [   1.000   -6.2500    7.1250]
  Factoring this from the original polynomial gives
    quotient  --> q = [   1.000   -2.000   14.000   -8.000   40.000]
    remainder --> r = [ 0.00e+000  0.00e+000  0.00e+000  0.00e+000  0.00e+000  1.71e-013 -1.14e-013]
  The roots of q are =
rootsq =
   1.0000 + 3.0000i
   1.0000 - 3.0000i
  -0.0000 + 2.0000i
  -0.0000 - 2.0000i
  The roots of p are =
rootsp =
   4.7500
   1.0000 + 3.0000i
   1.0000 - 3.0000i
  -0.0000 + 2.0000i
  -0.0000 - 2.0000i
   1.5000
>>
```

```
%
%   POLY_OPS_3.M    Working with polynomials in Matlab
%
%   This file simply does a number of polynomial manipulations in Matlab to
%   demonstrate several of the available functions for working with polynomials.
%   The function used in this exercise is:
%     f(x) = x^6 - 8.25x^5 + 33.625x^4 - 109.75x^3 + 189.75x^2 - 307x + 285
%
%   File prepared by J. R. White, UMass-Lowell  (last update:  Nov. 2017)
%

     clear all,  close all,  format short,  format compact
%
%   Part A  plot f(x)
     p = [1.0 -8.25 33.625 -109.75 189.75 -307.0  285.0];  % coeffs
     x = linspace(0,5,101);                                % define x domain
     f = polyval(p,x);                                     % evaluate polynomial
     plot(x,f, 'LineWidth',2), grid                        % plot polynomial
     title('Poly\_Ops\_3:  Plot of given polynomial')
     xlabel('x values'),ylabel('f(x)')
     text(0.20,700,'f(x) = x^6 - 8.25x^5 + 33.625x^4 - 109.75x^3 + 189.75x^2 - 307x +↙
285')
%
%   Note:  From the above plot, there are clearly two (2) real roots to this
%   polynomial.  One is in the range of 1 < x1 < 2 and the second is between
%   4.5 < x2 < 5.  This implies that the remaining four (4) roots are complex.
%   Thus, there should be two pairs of complex conjugate roots.
%
%   Part B  find the real roots using Matlab's FZERO function
     fx = @(x) polyval(p,x);
     a = 1;    b = 2;    x1 = fzero(fx,[a b]);
     a = 4.5;  b = 5;    x2 = fzero(fx,[a b]);
     fprintf('\n  The real roots of f(x) are:  %8.5f  %8.5f \n',x1,x2)
%
%   Part C  factor out the real roots to give the remaining 4th order polynomial
     p1 = [1 -x1];  p2 = [1 -x2]; % linear factor associated with roots x1 & x2
     pp = conv(p1,p2);            % multiplies two polynomials  --> quadratic
     [q,r] = deconv(p,pp);        % divides original polynomial by quadratic poly
     fprintf('  The two linear factors are given by p1  = [%8.3f  %8.3f] \n',p1)
     fprintf('                                      p2  = [%8.3f  %8.3f] \n',p2)
     fprintf('  Multiplication of these gives pp  = [%8.3f  %8.4f  %8.4f] \n',pp)
     fprintf('  Factoring this from the original polynomial gives\n')
     fprintf('    quotient  --> q = [%8.3f %8.3f %8.3f %8.3f %8.3f] \n',q)
     fprintf('    remainder --> r = [%10.2e %10.2e %10.2e %10.2e %10.2e %10.2e %10.2e]↙
\n',r)
%
%   Part D  find the roots of the quotient polynomial
     rootsq = roots(q);       % roots of q
     fprintf('  The roots of q are =   \n');  rootsq
%
%   Part E   find roots of original polynomial
     rootsp = roots(p);       % roots of p
     fprintf('  The roots of p are =   \n');  rootsp
%
%  end of problem
```