Chapter 18 Polynomials

18.1 Roots

MATLAB performs numerous operations involving polynomials. The most common situation dealing with polynomials is finding its roots. A polynomial is uniquely defined by its coefficients. MATLAB represents a polynomial by a vector containing the coefficients of the powers in descending order. For example, the polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + ...+ a_1 x + a_0$$

is saved in MATLAB as a vector p where $p = [a_n, a_{n-1}, ..., a_1, a_0]$. If a coefficient is zero, it must appear in the vector. After the coefficient vector is defined, the roots of the polynomial are returned in a column vector from a call to '`roots(p)`'.

The roots of a polynomial are found in the example below. The polynomial is evaluated at the roots (where it should be zero) and then plotted to verify it passes through the real roots.

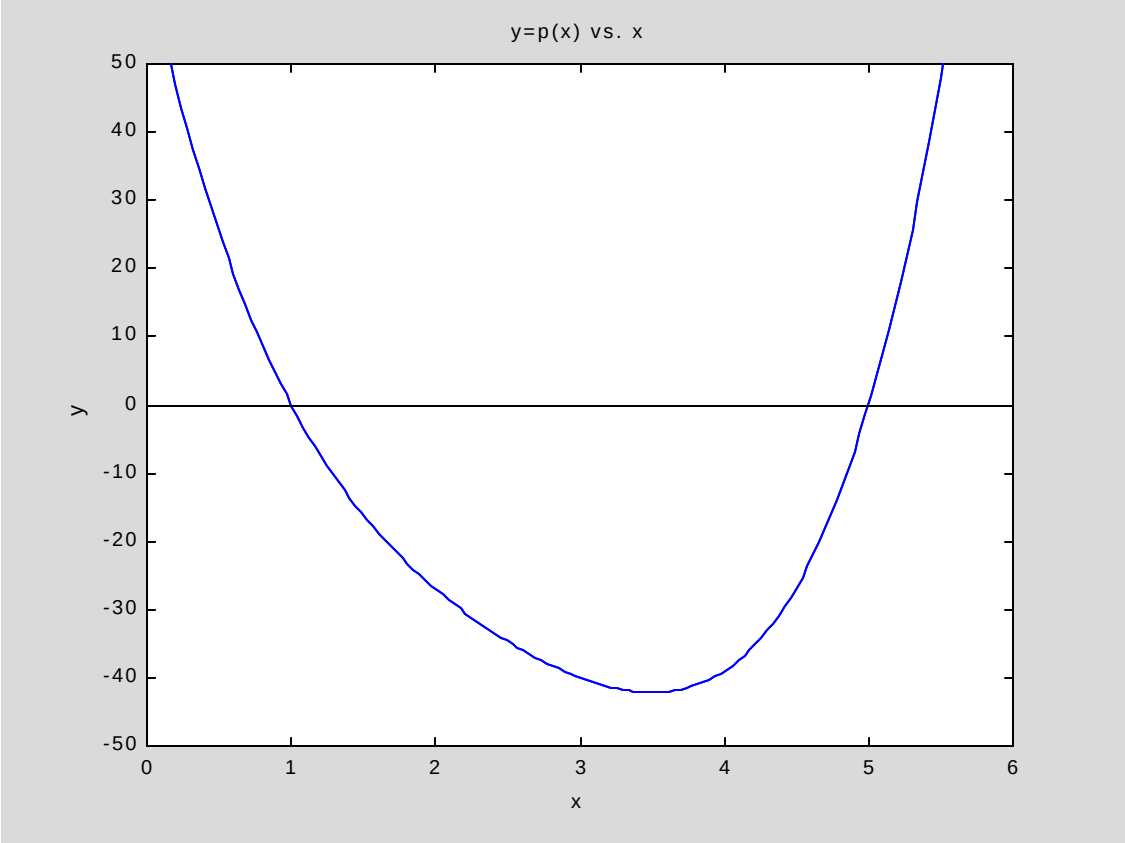
Example 18.1.1

```
p=[1,-10,42,-98,65]  % Define coefficient vector for polynomial
                     % p(x) = x^4 - 10x^3 + 42x^2 - 98x + 65
proots=roots(p);  % Find roots of p(x)
x=proots % Set x equal to row vector of roots of p(x)=0
y=p(1)*x.^4 + p(2)*x.^3 + p(3)*x.^2 + p(4)*x + p(5) % Evaluate p(x)
                                                    % at each of its 4 roots
xi=linspace(0,10,250); % Create xi vector for evaluating yi=p(xi)
yi=p(1)*xi.^4 + p(2)*xi.^3 + p(3)*xi.^2 + p(4)*xi + p(5); % yi=p(xi)
plot(xi,yi) % Plot y=p(x) vs x
hold on
axis([0 6 -50 50]) % Set scale for plot
plot([0 10],[0 0],'k') % Plot x-axis
xlabel('x'), ylabel('y')
title('y=p(x) vs. x')
```

```
p =     1   -10    42   -98    65

x =
   2.0000 + 3.0000i
   2.0000 - 3.0000i
   5.0000
   1.0000

y =
  1.0e-012 *
  -0.0568 + 0.0568i
  -0.0568 - 0.0568i
  -0.3411
        0
```



The reverse process, i.e. obtaining the polynomial coefficients from the roots is easily accomplished using the function '`poly`'.

Example 18.1.2

```
pcoeffs=poly(proots) % Find coeff's of polynomial with roots 'proots'

pcoeffs =
    1.0000  -10.0000   42.0000  -98.0000   65.0000
```

18.2 Multiplication

Multiplying two polynomials in MATLAB is as simple as defining the coefficient vectors and then performing a convolution using the 'conv' function.

Example 18.2.1

```
p=[1:2 -5:-4]  % First polynomial is p(x) = x^4 + 2x^3 - 5x - 4
q=[1 0 -2 0] % Second polynomial is q(x) = x^3 -2x
u=conv(p,q)

p =  1     2    -5    -4
q =  1     0    -2     0
u =  1     2    -7    -8    10     8     0
```

Executing Example 18.2.1 shows that

$$p(x)\cdot q(x) = (x^4+2x^3-5x-4)\cdot(x^3-2x)$$

$$= x^7+2x^6-7x^5-8x^4+10x^3+8x$$

The roots of the product polynomial and the individual polynomials must agree. This is easily verified.

Example 18.2.2

```
rp=roots(p) % Find roots of p(x)
rq=roots(q) % Find roots of q(x)
ru=roots(u) % Find roots of u(x)=p(x)*q(x)

rp =
   -3.1774
    1.8558
   -0.6784

rq =
    0
    1.4142
   -1.4142

ru =
    0
   -3.1774
   -1.4142
    1.8558
    1.4142
   -0.6784
```

18.3 Addition

Addition of polynomials is a bit more complicated than multiplication when the polynomials involved are not the same order. That is, MATLAB cannot add polynomials of different order directly because the coefficient vectors representing them are not the same size. The solution is to pad the lower order polynomial coefficient vector with leading zeros until the two coefficient vectors are the same size. The function '`polyadd`' in Example 18.3.1 below adds two polynomials. Several test cases are included after the function.

Example 18.3.1

```
function h=polyadd(f,g)
% This function adds two polynomials. If the polynomials are not the
% same order, it pads the coefficient vector of the lower order
% polynomial with zeros so the two vectors can be added.

if length(f)==length(g) % Check if polynomials are the same order
  h=f+g
elseif length(f)>length(g) % Check if f is higher order than g
  diff=length(f)-length(g);
  gg=[zeros(1:diff) g] % Pad g with 'diff' zeros
  h=f+gg % Compute sum
else % g is higher order than f
  diff=length(g)-lenght(f)
  ff=[zeros(1:diff) f] % Pad f with 'diff' zeros
  h=ff+g % Compute sum
end % if length(f)==length(g)

disp(' '), disp('Case 1')
g=[1 2 -3 4] % Define g(x) = x^3 + 2x^2 - 3x + 4
h=[-1 -2 3  -4] % Define h(x) = -x^3 - 2x^2 + 3x - 4
z=polyadd(g,h) % Add polynomials g and h
disp(' '), disp('Case 2')
f=[1 3 -2 0 5]  % Define f(x) = x^4 +3x^3 -2x^2 + 5x
g=[4 -1 6] % Define g(x) = 4x^2 - x +6
y=polyadd(f,g) % Add polynomials f and g
disp(' '), disp('Case 3')
a=[1]  % a(x)=1
b=[1 0 0 0 -1]  % b(x) = x^4 - 1
y=polyadd(a,b)

Case 1

g =  1     2    -3     4
h = -1    -2     3    -4
z =  0     0     0     0

Case 2

f =  1     3    -2     0     5
g =  4    -1     6
y =  1     3     2    -1    11
```

```
Case 3

a =  1
b =  1     0     0     0    -1
y =  1     0     0     0     0
```

Note the use of the local variable `diff` in the function workspace. Despite the fact a built-in MATLAB function named `diff` exists, there is no problem assigning it a value and refering to it in the next statement. Recall that MATLAB first looks in the current workspace for named variables before checking if its a built-in function.

18.4 Division

Division of polynomials is achieved by the deconvolution function '`deconv`'. In the example below, the polynomial $N(x) = 4x^5 + 2x^3 - x^2 + x + 1$ is divided by the lower order polynomial $D(x) = x^2 - 2$. The output is returned in two vectors. The first is the coefficient vector of the quotient, and the second produces the remainder polynomial coefficients. In general,

$$\frac{N(x)}{D(x)} = Q(x) + \frac{R(x)}{D(x)}$$

where $Q(x)$ is returned as zero if the order of $N(x)$ is less than the order of $D(x)$.

Example 18.4.1

```
N=[4 0 2 -1 1 1] % Define N(x) = 4x^5 + 2x^3 -x^2 + x + 1
D=[1 0 -2] % Define D(x) = x^2 - 2
[Aq,Ar]=deconv(N,D) % Find N(x)/D(x)

N =   4     0     2    -1     1     1

D =   1     0    -2

Aq =  4     0    10    -1

Ar =  0     0     0     0    21    -1
```

After running Example 18.4.1, the following result is apparent.

$$\frac{4x^5 + 2x^3 - x^2 + x + 1}{x^2 - 2} = 4x^3 + 10x - 1 + \frac{21x - 1}{x^2 - 2}$$

The above result is easily verified in MATLAB by showing that

$$N(x) = Q(x) \cdot D(x) + R(x)$$

Example 18.4.2

```
N % Display N(x) coefficients
N_check=polyadd(conv(Aq,D),Ar) % Check result from Example 18.4.1

N =  4     0     2    -1     1     1

N_check =  4     0     2    -1     1     1
```

The next example demonstrates the case when the numerator polynomial is lower order than the denominater polynomial.

Example 18.4.3

```
A=[1 2 3] % Define A(x) = x^2 + 2x + 3
B=[1 0 1 0 5] % Define B(x) = x^4 + x^2 + 5
[Cq,Cr]=deconv(A,B) % Find A(x)/B(x)

A =   1     2     3
B =   1     0     1     0     5

Cq =  0

Cr =  1     2     3
```

18.5 Differentiation

Differentiation of polynomial functions is a straightforward procedure, one that MATLAB implements with the function '`polyder`'.

Example 18.5.1

```
f(1)=3; % Coefficient of highest order term
f(2)=5; % Coefficient of next highest order term
f(3)=0;
f(4)=-1;
f(5)=2; % Coefficient of zero order term, i.e. the constant
f % f(x) = 3x^4 + 5x^3 - x + 2
dfdx=polyder(f) % Find df/dx coefficient vector
```

```
f =    3     5     0    -1     2

dfdx =      12    15     0    -1
```

Example 18.5.2

```
f=[1 0 3 2] % f(x) = x^3 + 3x +2
g=[1 -4 5] % g(x) = x^2 - 4x +5
fg=conv(f,g) % Find f(x)*g(x) coefficient vector
d_fg_dx=polyder(fg) % Differentiate the product f(x)*g(x)
u1=conv(f,polyder(g)); % Find f*dg/dx
u2=conv(g,polyder(f)); % Find g*df/dx
d_fg_dx=polyadd(u1,u2) % Differentiate f(x)*g(x) using product rule
```

```
f =                  1     0     3     2

g =                        1    -4     5

fg =     1    -4     8   -10     7    10

d_fg_dx =      5   -16    24   -20     7

d_fg_dx =      5   -16    24   -20     7
```

18.6 Evaluation

Polynomials are usually evaluated at one or more discrete points over some interval of interest. In MATLAB, '`polyval`' is the built-in function for doing this. To illustrate polynimial evaluation, consider an annual deposit A which earns interest at i % per annum. The future worth of the account after n years is

$$F = A + A(1+i) + A(1+i)^2 + \ldots.. + A(1+i)^{n-2} + A(1+i)^{n-1} + A(1+i)^n$$

$$= A(u^n + u^{n-1} + u^{n-2} + \ldots.. + u^2 + u + 1) \qquad \text{where } u = 1+i$$

The following example computes the account balance after n years by evaluating the polynomial function above.

Example 18.6.1

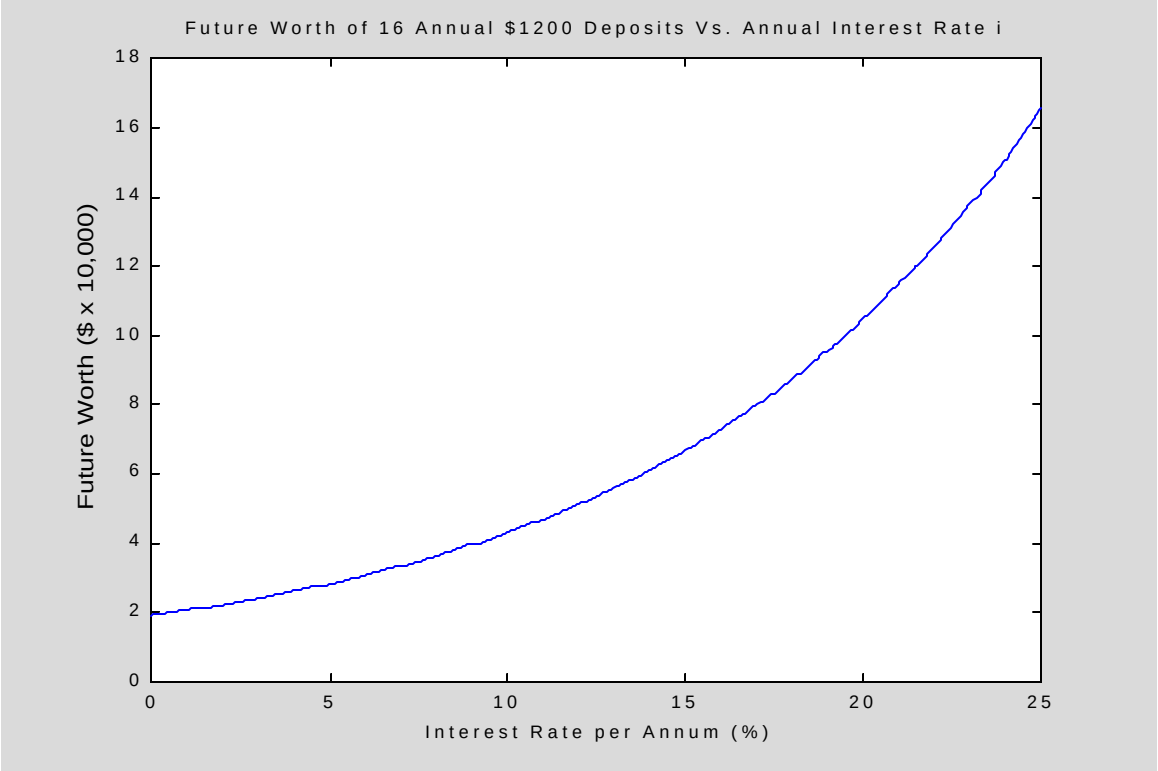
```
A=1200 % Annual deposit
i=10 % Annual interest rate (per cent)
i=i/100; % Convert i to decimal value
u=1+i;
n=15; % Number of years
p=ones(1,n+1); % Coefficient vector of polynomial
F=A*polyval(p,u) % Compute future amount
```

```
A =  1200
i =    10
F =  4.3140e+004
```

The '`polyval`' function provides a quick way to establish the values of a polynomial so it can be plotted.

Example 18.6.2

```
A=1200;
i=linspace(0,25,500); % Interval of i values for plot
i=i/100; % Convert array of i values to decimal numbers
u=1+i;
n=15; % Number of years
p=ones(1,n+1); % Coefficient vector of polynomial
F=A*polyval(p,u)/10000; % Future worth/10000
i=100*i; % Convert i back to per cent
plot(i,F) % Plot F vs i
title('Future Worth of 16 Annual $1200 Deposits Vs. Annual Interest Rate
i')
xlabel('Interest Rate per Annum (%)')
ylabel('Future Worth ($ x 10,000)')
```

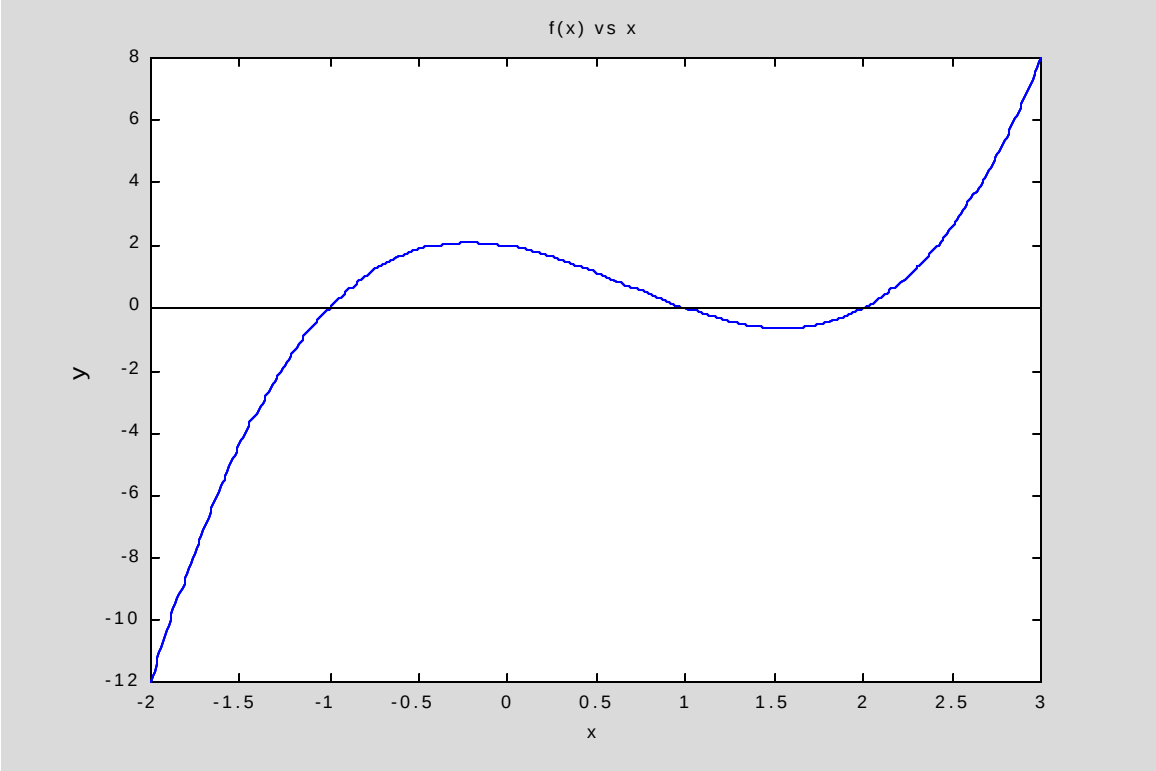


Example 18.6.3

```
a=conv([1 0 -1],[1 -2]) % Coefficient vector of f(x)=(x^2-1)*(x-2)
f_roots=roots(a)
xi=-2:0.01:3; %  Values of x to evaluate f(x) at
yi=polyval(a,xi); % yi=f(xi)
plot(xi,yi) % Plot y=f(x) vs x
hold on
plot([-2,3],[0,0],'k') % Plot x-axis
xlabel('x')
ylabel('y')
title('f(x) vs x')
```

```
a =  1    -2    -1     2

f_roots =
    2.0000
    1.0000
   -1.0000
```


f(x) vs x
y
x
8
6
4
2
0
-2
-4
-6
-8
-10
-12
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3

18.7 Rational Polynomials

There are times when the ratio of two rational polynomials needs to be expressed in partial fraction form. For example, the expression $\frac{x+3}{x^3+5x^2+7x+2}$ can be expanded into the sum of 3 terms using the MATLAB '`residue`' function.

Example 18.7.1

```
n=[1 3]  % Numerator polynomial n(x) = x + 3
d=[1 5 7 2] % Denominator polynomial d(x) = x^3 + 5x^2 + 7x + 2
[r,p,k]=residue(n,d) % Find residues and poles of n(x)/d(x)

n =  1     3
d =  1     5     7     2

r =
    0.2764
   -1.0000
    0.7236

p =
   -2.6180
   -2.0000
   -0.3820

k =
     []
```

From the results obtained in the above example, it follows that

$$\frac{x+3}{x^3+5x^2+7x+2}=\frac{0.2764}{x+2.6180}-\frac{1}{x+2}+\frac{0.7236}{x+0.382}$$

When the order of the numerator polynomial is greater than or equal to the order of the denominater polynomial, the last output `k` will not be an empty vector. Instead it will contain the coefficients defining the non-fractional term in the expansion.

Example 18.7.2

```
n1=d, d1=n, % Reverse n(x) and d(x)
[r1,p1,k1]=residue(n1,d1) % Find partial fraction expansion

n1 =     1     5     7     2
d1 =     1     3
r1 =    -1
p1 =    -3
k1 =     1     2     1
```

In this case, $$\frac{x^3+5x^2+7x+2}{x+3}=\frac{-1}{x+3}+x^2+2x+1$$

The '`residue`' function can also be used to sum the individual terms of the partial fraction expansion. There must be 3 input vector arguments `r`, `p`, and `k` (an empty vector in the absence of a non-fractional term).

Example 18.7.3

```
 r,p
[n,d]=residue(r, p, [ ]) % Combine partial fraction terms into n/d

r =
    0.2764
   -1.0000
    0.7236

p =
   -2.6180
   -2.0000
   -0.3820

n =   -0.0000    1.0000    3.0000
d =    1.0000    5.0000    7.0000    2.0000

r1,p1,k1
[n1,d1]= residue(r1, p1, k1) % Combine r1/p1 + k1 polynomial

r1 =    -1
p1 =    -3
k1 =     1     2     1

n1 =     1     5     7     2
d1 =     1     3
```

18.8 Curve Fitting

There are a variety of ways to represent a set of empirical data. One method, Least Squares Regression, is based on the use of low order polynomials to fit the data in the least squares sense. A least squares fit is a unique curve with the property that the sum of the squares of the differences between the fitted curve and the given data at the data points is a minimum. MATLAB generates least squares polynomials for a set of data by using the function `polyfit`'. It accepts vectors of `x` and `y` data and a scalar `n` as the order of the polynomial approximation.

The target heart rate during exercise for different ages is tabulated below

Age	20	25	30	35	40	45	50	55	60	65	70
Target Heart Rate	150	146	142	139	135	131	127	124	120	116	112

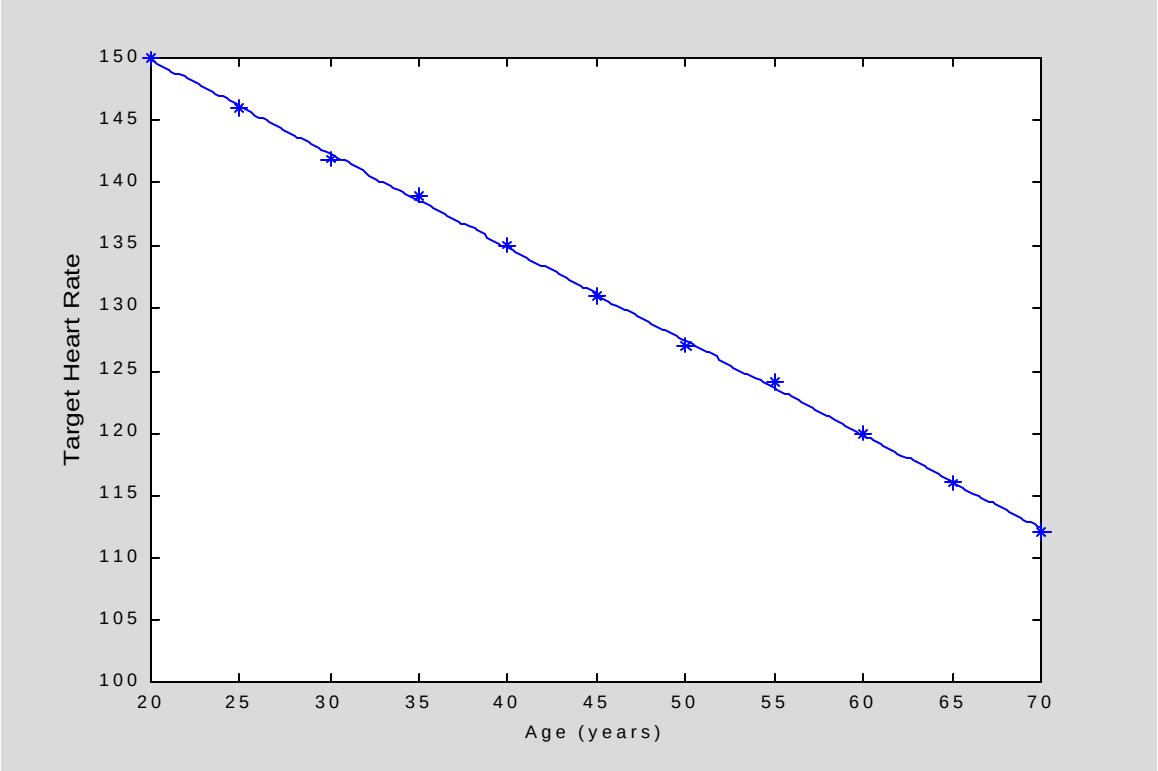
The data is plotted and a first order least square polynomial is found and presented on the same graph in the next example.

Example 18.8.1

```
x=20:5:70 % Create vector of age values
y=[150 146 142 139 135 131 127 124 120 116 112] % Create vector of
                                        % target heart rate values
plot(x,y,'*') % Plot raw data
a1=polyfit(x,y,1) % Find coefficients of least square line
xi=linspace(20,70,250); % Create vector of x values for plotting
yi=polyval(a1,xi); % Evaluate points on least square line for plotting
hold on
v=[20 70 100 150];
plot(xi,yi)
axis(v)
xlabel('Age (years)')
ylabel('Target Heart Rate')
```

```
x =  20    25    30    35    40    45    50    55    60    65    70
y = 150   146   142   139   135   131   127   124   120   116   112

a1 =  -0.7527  164.9636
```



A higher order polynomial curve is warranted in the next example where air pressure and the speed of sound variations with altitude in the troposphere are tabulated.

Altitude, *h* (ft)	Pressure, *p* (psi)	Speed of Sound, *v* (ft/sec)
0	14.7	1116.4
5000	12.2	1097.1
10000	10.1	1077.4
20000	6.76	1036.9
30000	4.37	994.85
36000	3.31	968.75

Example 18.8.2

```
h=[0 5 10 20 30 36]; % Altitude data
p=[14.7, 12.2, 10.1, 6.76, 4.37, 3.31]; % Pressure data
v=[1116.4, 1097.1, 1077.4, 1036.9, 994.85, 968.75]; % Speed data
p2=polyfit(h,p,2); % Find quadratic fit coefficients for p
hi=linspace(0,40,500); % Create h data for evaluation of 2nd order fit
pi=polyval(p2,hi); % Compute quadratic fit p values
v2=polyfit(h,v,2); % Find quadratic fit coefficients for v
vi=polyval(v2,hi); % Compute quadratic fit v values
subplot(1,2,1) % Divide figure window for 2 side by side graphs and
               % make the left side active
```

```
hold on
plot(h,p,'*') % Plot p vs h table data in left side
plot(hi,pi) % Plot quadratic fit hi vs pi in left side
xlabel('Altitude, h (ft x 1000)')
ylabel ('Air Pressure, p (psi)')
title('Air Pressure Vs. Altitude')
subplot(1,2,2) % Make right side active
hold on
plot(h,v,'*') % Plot v  vs h table data in right side
plot(hi,vi) % Plot quadratic fit vi vs hi in right side
xlabel('Altitude, h (ft x 1000)')
ylabel ('Speed of Sound, v (ft/sec)')
title('Speed of Sound Vs. Altitude')
```