```
% Will Kramlinger
% CE 3101
% Gonella
% HW 4

% PROBLEM 1.1

load('set1_HW4.mat')
edit
```

```
function [a1,a0] = LinearRegression(x,y)
% Will Kramlinger; 2/28/14
% The following code is appropriated from the textbook.

% LinearRegression calculates the coefficients a1 and a0 of the linear
% equation y = a1*x + a0 that best fits n data points.
% Input variables:
% x = A row array with the coordinates x of the data points.
% y = A row array with the coordinates y of the data points.
% Output variables:
% a1 = The coefficient a1.
% a0 = The coefficient a0.

nx = length(x);
ny = length(y);
if nx ~= ny
    error('The number of elements in x must be the as in y.')
    a1 = 'Error';
    a0 = 'Error';
else
    Sx = sum(x);
    Sy = sum(y);
    Sxy = sum(x.*y);
    Sxx = sum(x.^2);
    a1 = (nx*Sxy - Sx*Sy)/(nx*Sxx - Sx^2);
    a0 = (Sxx*Sy - Sxy*Sx)/(nx*Sxx - Sx^2);
end
end
```

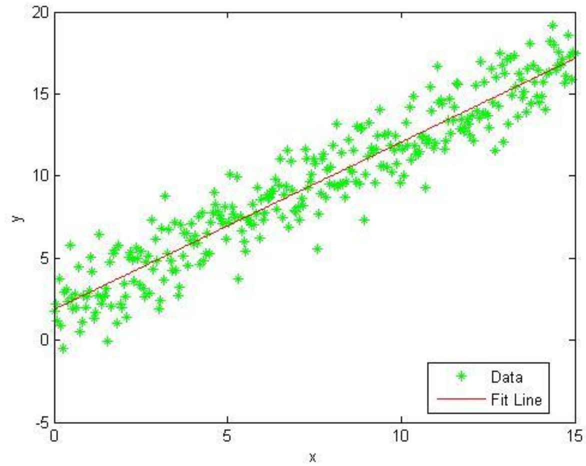
```
[a1,a0] = LinearRegression(xL,yL)

a1 =

    1.0197

a0 =

    1.8642

% The fit line found by LinearRegression is y = 1.0197x + 1.8642.
```

% PROBLEM 1.2

edit

```
function [a1,a0] = PseudoInverse(x,y)
% Will Kramlinger; 2/28/14
% The function calculates the coefficients a1 and a0 of the linear equation
% y = a1*x + a0 that best fits n data points.
% Input variables:
% x = A row array with the coordinates x of the data points.
% y = A row array with the coordinates y of the data points.
% Output variables:
% a1 = The coefficient a1.
% a0 = The coefficient a0.

nx = length(x);
ny = length(y);
if nx ~= ny
    error('The number of elements in x must be the as in y.')
    a1 = 'Error';
    a0 = 'Error';
else
    xmod = ones(nx,2);
    xmod(1:nx,1) = x';
    coeff = (xmod'*xmod)\xmod'*y';
    a1 = coeff(1,1);
    a0 = coeff(2,1);
    fprintf('The fit equation is y = %4.4fx + %4.4f.',a1,a0)
        plot(x,y,'green*'); hold on
        plot(x,a1*x+a0,'r');
        xlabel('x'); ylabel('y');
        legend('Data','Fit Line', 'Location', 'best'); hold off
end
```

```
[b1,b0] = PseudoInverse(xL,yL)
The fit equation is y = 1.0197x + 1.8642.
b1 =

    1.0197


b0 =

    1.8642


[a1,a0] == [b1,b0]
ans =

     0     0
```

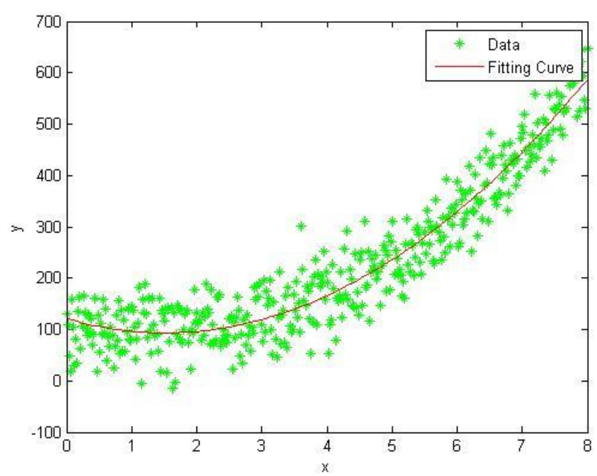



```
% The coefficients found by PseudoInverse are not exactly equal to those
% found by LinearRegression but good up to 4 decimal places.
% Both fit lines are of the equation y = 1.0197x + 1.8642.
% The plot shows a decent amount of variance about the fit line.

% PROBLEM 2

load('set2_HW4.mat')
plot(xC,yC,'green*');

% From visual inspection, the data seems somewhat quadratic.

edit
```

```
function coeff = PolyFit(x,y,m)
% Will Kramlinger; 2/28/14
% PolyFit expands on Linear Regression and provides a fitting curve of the
% general form y = (a_m)(x^m) + (a_m-1)(x^(m-1)) + ... + (a_1)(x) + a_0
% where n = number of data points & m < n-1.
% Input variables:
% x = A row array with the coordinates x of the data points.
% y = A row array with the coordinates y of the data points.
% m = The desired order of the fitting curve.
% Output variables:
% coeff = An array with the coefficients of the polynomial.
%         coeff(1) = a_0, coeff(2) = a_1, ...

if length(x) ~= length(y)
    disp('The number of elements in x must be the as in y.')
    coeff = 'DOES NOT COMPUTE, DOGGY!';
end
if m >= length(x)
    disp('M must be less than length(x) - 1')
    coeff = 'DOES NOT COMPUTE, DOGGY!';
end

V = zeros(m+1,m+1); % Use (m + 1) instead of n, in case m << n
for i = 1 :(m+1)
    for j = 1: (m+1)
```

```
        V(i,j) = sum(x.^(i + j - 2));
    end
end

S = zeros(m+1,1);
for k = 1: (m+1)
    S(k,1) = sum(x.^(k-1).*y);
end

coeff = (V\S);
disp('NOTE TO SELF: coeff(1) = a_0, coeff(2) = a_1, and so on.')

% Plotting Section
xlin = linspace(min(x),max(x),100);
yfit = zeros(1,100);
total = zeros(1,m+1);
for a = 1:100
    for b = 1: (m+1)
        total(b) = (coeff( (m+1) - b + 1)*xlin(a).^(m+1 - b));
    end
    yfit(a) = sum(total);
end
plot(x,y,'green*'); hold on
plot(xlin,yfit,'r');
xlabel('x'); ylabel('y');
legend('Data','Fitting Curve')
hold off
% /Plotting Section
end
```

```
coeff = PolyFit(xC,yC,2)
NOTE TO SELF: coeff(1) = a_0, coeff(2) = a_1, and so on.

coeff =

  120.6944
  -35.7861
   11.7601
```

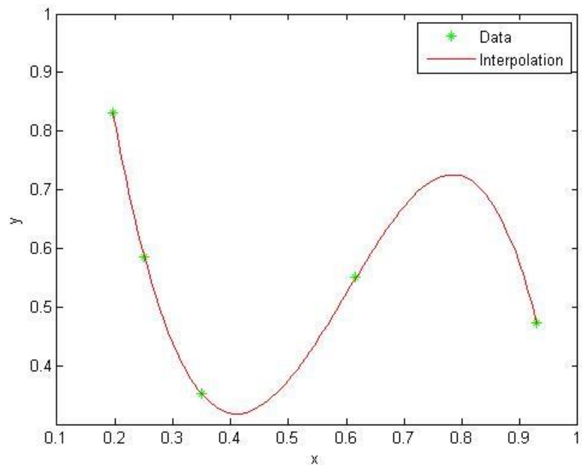


```
% The curve for m = 2 appears to be a good fit for the data.

% PROBLEM 3

x = rand(1,5); y = rand(1,5);
edit
```

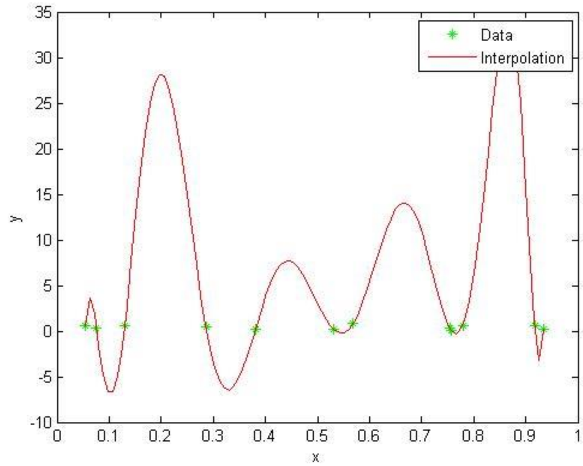
```
function coeff = Interpolate(x,y)
% Will Kramlinger; 2/28/14
% Interpolate performs standard interpolation and provides a fitting curve
% of the general form y = (a_m)(x^m) + (a_m-1)(x^(m-1)) + ...
%                          + (a_1)(x) + a_0
% where n = number of data points & m < n-1.
% Input variables:
% x = A row array with the coordinates x of the data points.
% y = A row array with the coordinates y of the data points.
% % Output variables:
% coeff = An array with the coefficients of the polynomial.
%         coeff(1) = a_0, coeff(2) = a_1, ...

if length(x) ~= length(y)
    disp('The number of elements in x must be the as in y.')
    coeff = 'DOES NOT COMPUTE, DOGGY!';
else
    n = length(x);
end

vdm = zeros(n,n); % The Van Der Monde (sp?) Matrix
for k = 1:n
    vdm(1:n,k) = x'.^(k-1);
end

coeff = vdm\y';
disp('NOTE TO SELF: coeff(1) = a_0, coeff(2) = a_1, and so on.')

% Plotting Section
xlin = linspace(min(x),max(x),100);
yfit = zeros(1,100);
total = zeros(1,n);
for a = 1:100
    for b = 1:n
        total(b) = (coeff(n - b + 1)*xlin(a).^(n - b)); % Creates array of
                                                          monomials
    end
    yfit(a) = sum(total); % Sums monomials for a particular xlin
end
plot(x,y,'green*'); hold on
plot(xlin,yfit,'r');
xlabel('x'); ylabel('y');
legend('Data','Interpolation')
hold off
% /Plotting Section
end
```

```
coeff = Interpolate(x,y)
NOTE TO SELF: coeff(1) = a_0, coeff(2) = a_1, and so on.
```

```
coeff =

    2.6029
  -13.0546
   22.4567
   -9.1333
   -2.7645
```




```
% The interpolation for n = 5 appears to represent the data effectively.

x = rand(1,12); y = rand(1,12);
coeff = Interpolate(x,y)
NOTE TO SELF: coeff(1) = a_0, coeff(2) = a_1, and so on.

coeff =

   1.0e+09 *

   -0.0000
    0.0000
   -0.0007
    0.0077
   -0.0505
    0.2086
   -0.5635
    1.0089
   -1.1868
    0.8813
   -0.3744
    0.0694
```




```
% The interpolation of the n = 12 sample is laughably bad.

% Per lecture, this rapid deterioration in interpolation quality as
% n increases is expected.  At a certain point, the versatility which
% having a few monomials of different powers provides is lost, and eventually
% becomes a hindrance in representing the behavior of the data.
```