Contents


```
% Do not edit this part
% It reads the matrix from the excel file and creates your x and y arrays

% a = readmatrix('problem2.csv');
% x = a(:,1);
% y = a(:,2);
```

Write your code here to calculate the MLS for a 2nd-order polynomial.

The system must be solved by any method YOU programmed (please note if reusing code from another assignment, such as the take-home midterm). Do not use RREF or any other functions built into matlab to fit the polynomial, or to solve the system.

```
%-------------------------------------------------------------------------

% kept getting an error from above code so I had to comment out but i did
% not edit any of the code above besides commenting them out
% I SIMPLY USED MATLAB'S OPEN FILE OPTION OF IMPORTING FILES THROUGH THE
% INTERFACE TO OBTAIN THE DATA NEEDED



% Calculate the tabled values (and their sum) that we will use in the
% matrix
xSquared = x.^2;
xCubed = x.^3;
x4thOrder = x.^4;
ySum = sum(y);
yxSum = sum(x.*y);
yx2Sum = sum(xSquared.*y);
N = length(x); % determine number of points used

% Matrix setup for MLS:
A = [N,sum(x),sum(xSquared); sum(x) sum(xSquared), sum(xCubed);
     sum(xSquared), sum(xCubed), sum(x4thOrder)]; % Matrix of X-values (based on formula in class)
B = [ySum; yxSum; yx2Sum]; % Matrix of Y-values
```

```
augmentedA = [A,B]; % Augmented matrix needed for performing Gauss-Jordan

[m, n] = size(augmentedA); % obtain dimensions needed in the loop

% Gauss-Jordan Elimination loop
for i = 1:min(m, n-1)  % Only go up to the second-to-last column

    % CODE PARTIALLY REUSED FROM MY MIDTERM SUBMISSION
    % EDITS WERE MADE TO ADJUST TO THE DIFERENCES IN MATLAB AND PYTHON

    % Find the pivot in the current column
    [~, pivot_row] = max( abs(augmentedA(i:m, i)) );
    pivot_row = pivot_row + i - 1;

    % Row/Pivot swapping
    if pivot_row ~= i
        temp = augmentedA(i, :);
        augmentedA(i, :) = augmentedA(pivot_row, :);
        augmentedA(pivot_row, :) = temp;
    end

    % Dives pivot rows
    augmentedA(i, :) = augmentedA(i, :) / augmentedA(i, i);

    % Eliminate the other rows
    for j = 1:m

        if j ~= i

            augmentedA(j, :) = augmentedA(j, :) - augmentedA(j, i) * augmentedA(i, :);

        end

    end

end

% places solved coefficients of the polynomial into an array
c = [augmentedA(1,4), augmentedA(2,4), augmentedA(3,4)] % [c0, c1, c2]

%-----------------------------------------------------------------------------------------------------
% WAS NOT USED IN CODE, JUST TO VERIFY VALUES
% Solution verifiction:
% B = [ySum; yxSum; yx2Sum];
```

```
% X = A\B
%-------------------------------------------------------------------------------------------------------
```

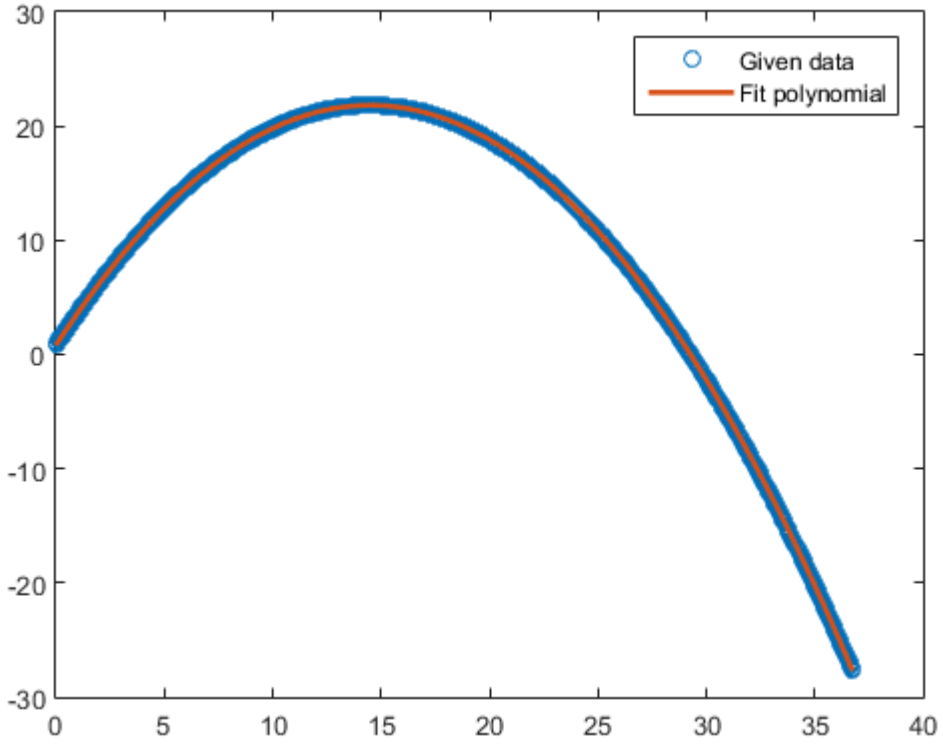
```
c =

    0.8000    2.9000   -0.1000
```

This section plots the original data and your result on a plot that must be submitted.

I use the variables c(1), c(2), and c(3) to store the constants in the polynomial. You can edit this if needed

```
plot(x,y,'o') % plots the original data, do not edit
hold
plot(x,c(1)+c(2).*x+c(3)*x.^2,'LineWidth',2) % edit if needed
legend('Given data','Fit polynomial') % leave this line
text(.1,70,char(java.lang.System.getProperty('user.name'))) % leave this line, proves your work
text(.1,65,char(java.net.InetAddress.getLocalHost.getHostName)) % leave this line, proves your work
```

```
Current plot held
```



Published with MATLAB® R2015a