TO: Professor Andreas Linninger / Grant Hartung

FROM: Kehinde Abioye

DATE: October 3, 2017

SUBJECT: Data fitting

The purpose of this assignment is to find the line that represents the least residual error amongst all the data points.

Introduction

The line of best fit is found using linear regression for the following data points:

Table 1. Experimental data

Xi	1	2	3	4	5	6
Yi	2	8	15	20	30	40

The minimum residual error for each point can also be found using this method. The first, second, and third order model is plotted using Table 1. Part 2 involves determining the rotation, scaling, and translation that aligns the two sets of coordinates provided by the TA.

Methods

Part 1a:

The linear regression coefficients were found for the first order model. This was done by rearranging the data points in the following style:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \\ 6 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix} \quad (1)$$

Then, the slope and offset were found using the following equation

$$(A^T * A)^{-1}(A^T * b) \quad (2)$$

After the vector was made, the slope and the y-intercept were assigned to the values in the vector. Then the values were used to find the residual error. The residual error was calculated using the linear equation for a straight line:

$$y = mx + b \quad (3)$$

"y" equals the residual

"m" is the slope

"x" is the x values for the data points in table 1

"b" equals Y_0-Y_i. Y_0 equals the y-intercept and Y_i is the y values for the data points in table 1

Part 1b:

Part b is like a, but the x coordinates are squared and added to the matrix as shown in equation 4:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \\ 16 & 4 & 1 \\ 25 & 5 & 1 \\ 36 & 6 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix} \quad (4)$$

Because of this another value is produced. The equation for the residual then turns into a quadratic equation:

$$y = ax^2 + bx + c \quad (5)$$

Where y is the residual, a is the slope, b is Y_0, x is the x values for the data points in table 1, and c = c-Y_i. C being the additional value.

Part 1c:

Part c is similar like a, but it is done for the third order. The x coordinates are cubed and added to the matrix as shown in equation 6:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 8 & 4 & 2 & 1 \\ 27 & 9 & 3 & 1 \\ 64 & 16 & 4 & 1 \\ 125 & 25 & 5 & 1 \\ 216 & 36 & 6 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix} \quad (6)$$

This produced four values and the residual equation was configured in equation 7:

$$y = ax^3 + bx^2 + cx + d \quad (7)$$

d is the fourth value which equals d-Y_i.

Part 1d:

The fitted functions and the data points were plotted with MATLAB. The points were plotted with the scatter function.

Part 2a:

Four points were chosen from the two data set given and plotted in MATLAB. These points were used to obtain the transformed matric and translational vector. A scatter plot was made for the four points. The coefficients were determined by creating a matrix as seen in part 1. The "\" command was used. This matrix was then transposed with the translation vector. Using "A\b" command for a square matrix takes the inverse of matrix A and multiplies it by vector b. A least-squares solution is needed for non-square matrices with more rows than columns. A matrix with the opposite is undetermined.

Part 2b:

The coefficients r and t were found in MATLAB and the two images were compared. The residual error was calculated and added to get a sum of error. The following equation as well as the "\" command was used.

$$Pseudo - inverse: x = (A^T A)^{-1}(A^T b) \quad (8)$$

Part 2c:

The same steps of part 2 was used for part c but using 5, 8, and 10 points

Results

Part 1:

The linear regressions for part 1 are shown in figures 1-3.

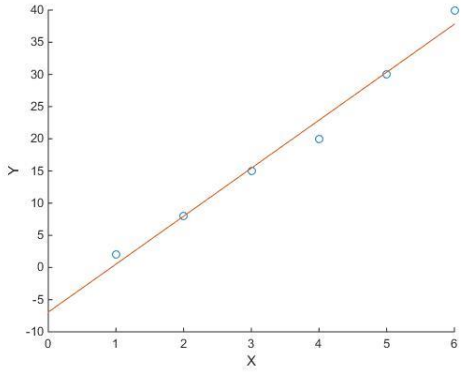

Figure 1 First order

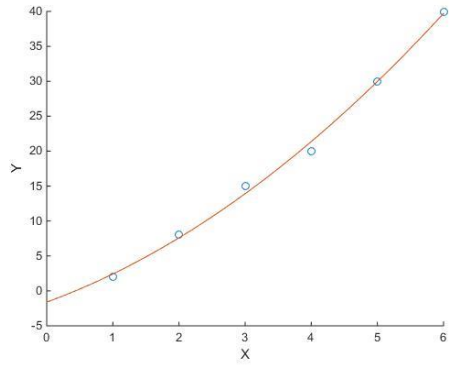

Figure 2 Second order

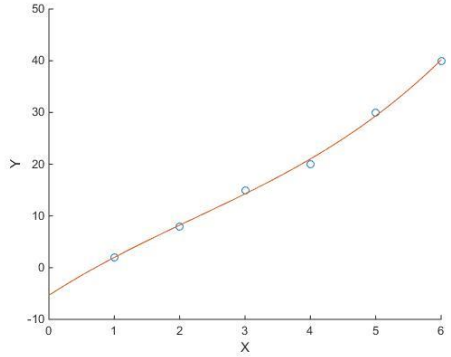

Figure 3 Third order

Part 2:

Idealized registration for 4 data points of the data set given shown in figure 4:

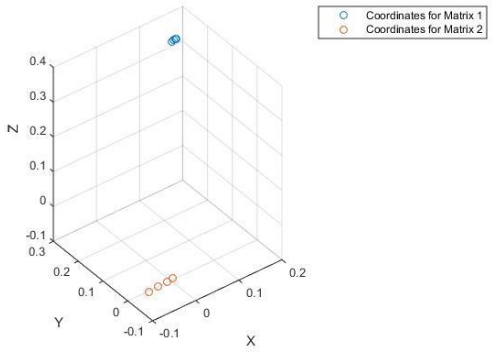

Figure 4 Idealized registration for four data points

Transformation matrices from all combinations of 5, 8, and 10 points:

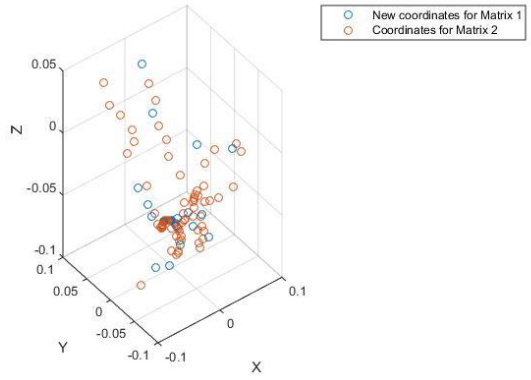

Figure 5 Transformed matrix

Conclusion

In part 1, linear regression was used to find the line of best fit for first, second, and third order models. The second part was used to find a transformation matrix and translation vector for a set of points (4, 5, 8, and 10) to compare two images.

Discussion

Data fitting is useful for fitting models to data and analyzing the precision of it. It is used to make educated guesses for where data would be if obtained. This is useful when data is unavailable at certain points.

Appendix A

```
clc
clear all
close all
%% Problem 1
xData = [1, 2, 3, 4, 5, 6];
yData = [2, 8, 15, 20, 30, 40];
% Problem 1A-1st order
Bvec = yData';
xVec = xData';
onesVec = ones(length(xData), 1);
A1 = [xVec, onesVec];
Slope_Offset1 = ((A1') * A1) \ ((A1') * Bvec);
for i = 1:6
rVec1(i) = A1(i, :) * Slope_Offset1(:) - Bvec(i);
end
tot_r1 = norm(rVec1, 2);
figure;
scatter(xData, yData);
hold on;
x = 0:0.1:6;
plot(x, Slope_Offset1(1)*x + Slope_Offset1(2));
xlabel('X');
ylabel('Y');
% Problem 1 B-2nd order
BVec2 = yData';
x_2 = xVec .^2;
onesVec = ones(length(xData), 1);
A2 = [xVec, x_2, onesVec];
Slope_Offset2 = ((A2') * A2) \ ((A2') * BVec2);
for i = 1:6
rVec2(i) = A2(i, :) * Slope_Offset2(:) - BVec2(i);
end
tot_r2 = norm(rVec2, 2);
figure;
scatter(xData, yData);
hold on;
x = 0:0.1:6;
plot(x, Slope_Offset2(1) * x + Slope_Offset2(2) * (x .^ 2) +
Slope_Offset2(3));
xlabel('X');
ylabel('Y');
% Problem 1C-3rd order
BVec3 = yData';
x_3 = xVec .^3;
onesVec = ones(length(xData), 1);
A3 = [xVec, x_2, x_3, onesVec];
Slope_Offset3 = ((A3') * A3) \ ((A3') * BVec3);
for i = 1:6
rVec3(i) = A3(i, :) * Slope_Offset3(:) - BVec3(i);
end
tot_r3 = norm(rVec3, 2);
figure;
scatter(xData, yData);
hold on;
x = 0:0.1:6;
```

```
plot(x, Slope_Offset3(1) * x + Slope_Offset3(2) * (x .^ 2) + Slope_Offset3(3)
* (x .^ 3) + Slope_Offset3(4));
xlabel('X');
ylabel('Y');
%% Problem 2
ptMatrix1File = load('ptCoordinates1.mat');
ptMatrix2File = load('ptCoordinates2.mat');
ptMatrix1 = ptMatrix1File.ptCoord1;
ptMatrix2 = ptMatrix2File.ptCoord1;
figure;
scatter3(ptMatrix1(1:4, 1), ptMatrix1(1:4, 2), ptMatrix1(1:4, 3));
hold on;
scatter3(ptMatrix2(1:4, 1), ptMatrix2(1:4, 2), ptMatrix2(1:4, 3));
xlabel('X');
ylabel('Y');
zlabel('Z');
legend('Coordinates for Matrix 1', 'Coordinates for Matrix 2');
% Problem 2A
pt1 = ptMatrix1(1:4, 1:3);
pt2 = ptMatrix2(1:4, 1:3);
ones_2 = ones(size(pt1, 1), 1);
pt1Prime = [pt1 ones_2];
pt2Prime = [pt2 ones_2];
transMatA = (pt1Prime \ pt2Prime);
transMatA = transMatA';
transMatA2 = transMatA(1:3, 1:3);
offsetVecA = transMatA(1:3, 4);
[Row, Col] = size(pt1);
for i = 1:Row
ptColA = (pt1(i, :))';
transColA = transMatA2 * ptColA + offsetVecA;
transPtA(i, :) = transColA';
end
% Problem 2B
ones_3 = ones(size(ptMatrix1, 1), 1);
pt1Prime2 = [ptMatrix1 ones_3];
pt2Prime2 = [ptMatrix2 ones_3];
transMat = (pt1Prime2 \ pt2Prime2);
transMat = transMat';
transMat2 = transMat(1:3, 1:3);
offsetVec = transMat(1:3, 4);
[Row, Col] = size(ptMatrix1);
for i = 1:Row
ptCol = ptMatrix1(i, :)';
transPtCol = transMat2 * ptCol + offsetVec;
transPtMat(i, :) = transPtCol';
end
Error = sum(sum(ptMatrix2)) - sum(sum(transPtMat));
figure;
scatter3(ptMatrix1(:, 1), ptMatrix1(:, 2), ptMatrix1(:, 3));
hold on;
scatter3(ptMatrix2(:, 1), ptMatrix2(:, 2), ptMatrix2(:, 3));
xlabel('X');
ylabel('Y');
zlabel('Z');
legend('Coordinates for Matrix 1', 'Coordinates for Matrix 2');
figure;
```

```
scatter3(transPtMat(:, 1), transPtMat(:, 2), transPtMat(:, 3));
hold on;
scatter3(ptMatrix2(:, 1), ptMatrix2(:, 2), ptMatrix2(:, 3));
xlabel('X');
ylabel('Y');
zlabel('Z');
legend('New coordinates for Matrix 1', 'Coordinates for Matrix 2');
% Problem 2C-5 points
pt1C = ptMatrix1(1:5, 1:3);
pt2C = ptMatrix2(1:5, 1:3);
ones_3 = ones(size(pt1C, 1), 1);
pt1CPrime = [pt1C ones_3];
pt2CPrime = [pt2C ones_3];
transMatC = (pt1CPrime \ pt2CPrime);
transMatC = transMatC';
transMatC2 = transMatC(1:3, 1:3);
offsetVecC = transMatC(1:3, 4);
[Row, Col] = size(pt1C);
for i = 1:Row
ptVecC = pt1C(i, :)';
transColC = transMatC2 * ptVecC + offsetVecC;
transPtC(i, :) = transColC';
end
[Row, Col] = size(ptMatrix1);
for i = 1:Row
ptColC = ptMatrix1(i, :)';
transVecC = transMatC2 * ptColC + offsetVecC;
transformedMatC(i, :) = transVecC';
end
ErrorC = sum(sum(ptMatrix2)) - sum(sum(transformedMatC));
% Problem 2C-8 points
pt1C2 = ptMatrix1(1:8, 1:3);
pt2C2 = ptMatrix2(1:8, 1:3);
ones_3 = ones(size(pt1C2, 1), 1);
pt1CPrime2 = [pt1C2 ones_3];
pt2CPrime2 = [pt2C2 ones_3];
transMatC2 = (pt1CPrime2 \ pt2CPrime2);
transMatC2 = transMatC2';
transformationMatC2 = transMatC2(1:3, 1:3);
offsetVecC2 = transMatC2(1:3, 4);
[Row, Col] = size(pt1C2);
for i = 1:Row
ptVecC = pt1C2(i, :)';
transColC2 = transformationMatC2 * ptVecC + offsetVecC2;
transPt1C2(i, :) = transColC2';
end
[Row, Col] = size(ptMatrix1);
for i = 1:Row
ptVecC2 = ptMatrix1(i, :)';
transVecC2 = transformationMatC2 * ptVecC2 + offsetVecC2;
transformedMatC2(i, :) = transVecC2';
end
ErrorC2 = sum(sum(ptMatrix2)) - sum(sum(transformedMatC2));
% Problem 2C-10 points
pt1C3 = ptMatrix1(1:10, 1:3);
pt2C3 = ptMatrix2(1:10, 1:3);
ones_3 = ones(size(pt1C3, 1), 1);
```

```
pt1CPrime3 = [pt1C3 ones_3];
pt2CPrime3 = [pt2C3 ones_3];
transMatC3 = (pt1CPrime3 \ pt2CPrime3);
transMatC3 = transMatC3';
transformationMatC3 = transMatC3(1:3, 1:3);
offsetVecC3 = transMatC3(1:3, 4);
[Row, Col] = size(pt1C3);
for i = 1:Row
ptVecC = pt1C3(i, :)';
transColC3 = transformationMatC3 * ptVecC + offsetVecC3;
transPt1C3(i, :) = transColC3';
end
[Row, Col] = size(ptMatrix1);
for i = 1:Row
ptVecC3 = ptMatrix1(i, :)';
transVecC3 = transformationMatC3 * ptVecC3 + offsetVecC3;
transformedMatC3(i, :) = transVecC3';
end
ErrorC3 = sum(sum(ptMatrix2)) - sum(sum(transformedMatC3));
```