ME 594 - Numerical Methods

Fall 2022

Curve fitting IMU data.

Report by,

Viral Panchal

Introduction

Inertial measurement unit – IMU is a highly used sensor in robotics. By using accelerometer, gyroscope and magnetometer, it can measure angular velocity, orientation, acceleration, and a body’s specific force as well. This sensor is useful to maneuver modern vehicles, either ground or aerial. It is used in missiles, aircrafts, satellites, to measure altitude and heading reference systems. This useful device has been miniaturized in size in past few decades. Today it is available in extremely small packages which makes its application further wide. Because of the small packages, it can be used in manipulator robotics, indoor drones, bio-robotic systems, etc.

Figure 1 shows a sample IMU sensor available in the market. Sensors from different manufacturers have some difference but typically an IMU sensor, as it is switched on starts measuring several physics parameters like angular velocity (in X,Y,Z), acceleration (in X,Y,Z), orientation, etc. and saves the quaternion data either on an onboard storage memory or connected control station. The frequency of data being measured very high and hence for a 60 second testing, user may end up with a hundreds of data readings.

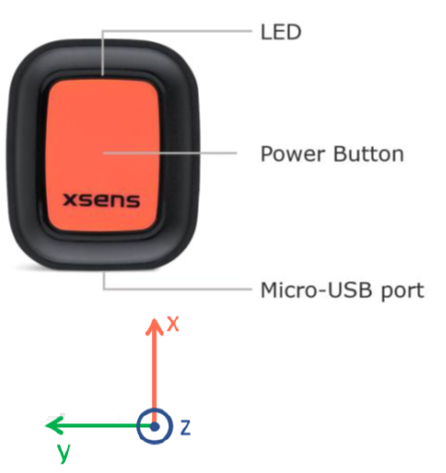
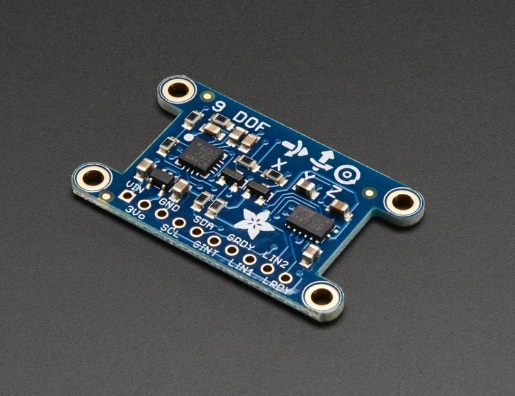


Figure 1: [a][b]

In this project, a comparison between curve fitting techniques, mainly Least Squares Methods and Cubic Splines method will be carried out. The idea is to implement the least squares method and cubic splines method to fit a curve for the sample data collected using a “Xsens dot” IMU sensor, See figure 1[b]. The purpose of doing this comparison is to understand the performance of these algorithms when the sample size is extremely huge and complex. Moreover, sample data and examples done in class had a particular trend of either increase or decreasing. In this project, the goal is to determine which algorithm is best suited when the data is similar to a sinusoidal/ repeating periodically kind of trend.

The figure below is a plot of raw data which gives an idea the kind of data that will be considered for this project. Increasing and decreasing magnitudes, area crowded with large number of readings, and complicated magnitudes.

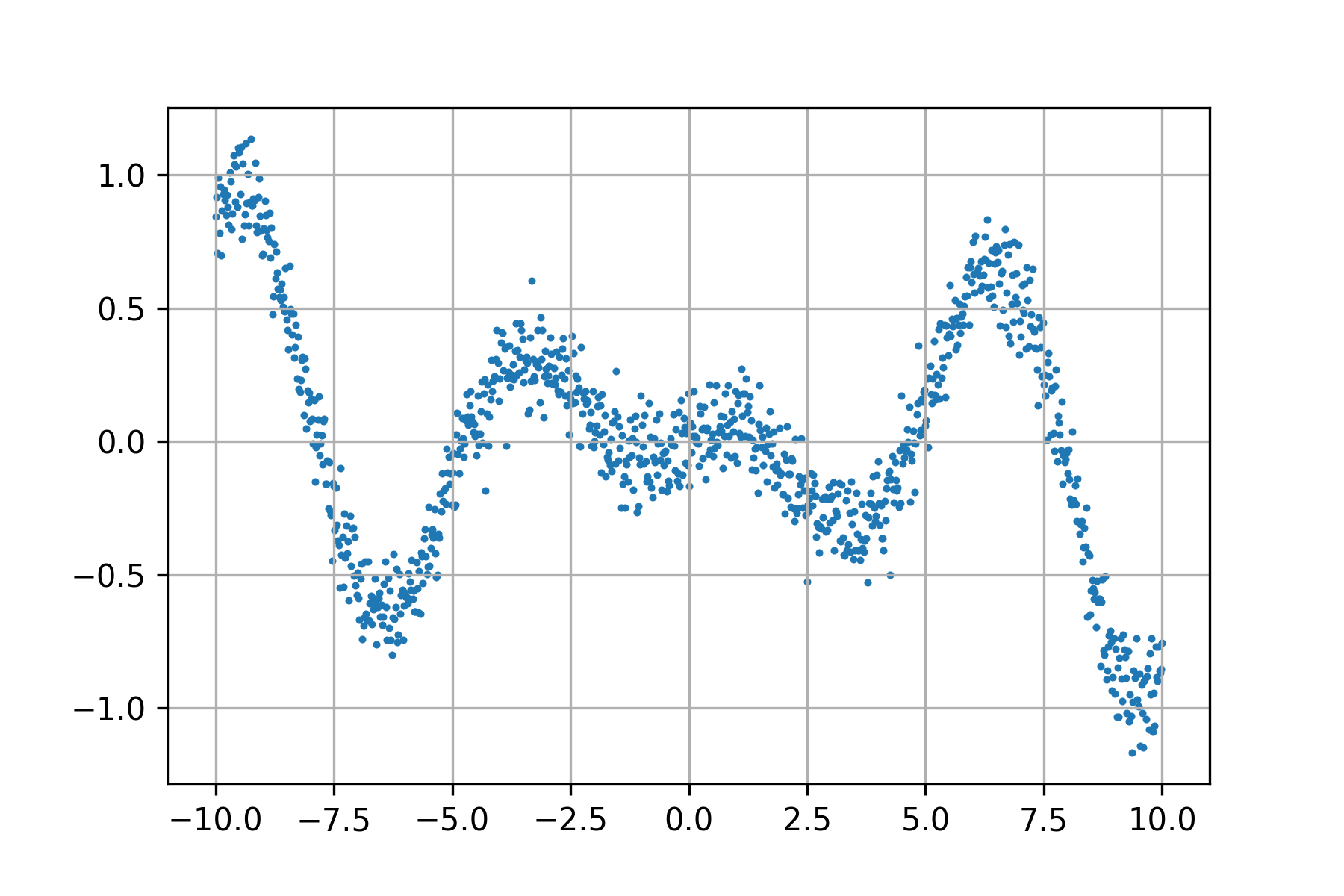


Figure 2

Looking at the data and the algorithms, an initial hypothesis was made, least squares method would be an easier approach, but the result achieved may be highly inaccurate and useless. For which an experiment could be done to increase the order of the polynomial and come up with an order which gives a good output, i.e., results to a plot which is not very far from the original raw plot. On the other hand, it was clear that a program for cubic splines would be difficult, but the output would be immaculate and very similar to the original raw plot.

In the following sections, literature on the applied numerical methods and discussion on concepts, algorithm, and implementation will be covered. Finally, results achieved will be discussed and a conclusion made from the whole project will be presented.

Numerical Methods

# Least Squares Method

This is a highly used method to find the best fit for a set of data points. In this approach, the sum of the offsets or residuals are minimized and plotted. This is a useful technique to predict the behavior of a dependent variable if some of the training data is with input and output is known to the user.

As mentioned above, in this project the sample size is huge, and the trend is repetitive as seen from the raw plot. Considering a line equation as the go-to function and plotting it would make it easier to reach to a solution, but it is highly unlikely that the solution will be of any help to predict any behavior.

And hence, a sample test with a line equation (Y = mx + c) as the function was done but immediately the focus was shifted towards making a generic least squares algorithm which could increase the order of the polynomial by one in every attempt and determine an order which would have the error less than the tolerance and provide a plot finally.

This idea of making a generic least squares algorithm was influenced by looking at patterns of similarity in the equations and matrices being generated as the order was increase manually and as the program was modified every time when the order was increased.

**Concept – Simple least squares for line equation**

Let’s consider a simple line equation/ linear least square. The process reduces to two equations,

= A \* + B \* N …. Eq.1

= A \* + B \* .…Eq.2

The above two equations can then be converted to matrices to compute the A and B variables. The matrices are as follows,

= ……Eq.3

**“X” matrix**

**“RHS” / “Y function” matrix**

**“C” / “Coefficient” matrix**

The above matrix could be easily solved in MATLAB, where

N 🡪 Number of data points

🡪 Sum of all “X” data points

🡪 Sum of all “” data points

🡪 Sum of all “Y” data points

🡪 Sum of product of all “X” and “Y” data points

To find the coefficient variables, an inverse of “X” matrix is computed and is pre-multiplied with the “RHS”/ “Y function” matrix.

**C = \* RHS … Eq.4**

The “X” matrix is always square since the rows increase as the order of the polynomial is increased.

**Concept – Generic least squares method.**

As the order of the polynomial is increased, a pattern within the formation of matrices to determine the solution is seen.

**\* =** … Eq.5

From the above equations, N is the order of the polynomial set. i ϵ [0,N]

Once the matrices are built in the MATLAB, the magnitudes of all the coefficient can be determined following the final step as shown in eq.4.

# Cubic Splines Method

This method is used in many interpolation applications. In cubic splines, a cubic polynomial is fit between two neighboring data points. In this method, the first and second derivative match at any points.

Considering that the sample data set considered is huge and does not have a single trend throughout. That is, either increasing or decreasing from left to right. Rather the trend varies and is kind of repetitive which makes fitting a line for such kind of scattered data not so useful. Cubic splines on the other hand has the potential to produce a curve making sure that all the data points pass through the curve and at the same time make it look slightly pleasant compared to the raw plot or linear spline/ piecewise spline plot usually made by the system.

**Concept:**

In this technique, a cubic polynomial as shown below is parameterized between the current data point and the next data point.

**(u) = + \* u + \* + \* …Eq.6**

**& u =**

In the above equation, u always lies between 0 and 1.

The goal is to determine the coefficients a, b, c, d in every iteration to get a polynomial equation that can be plotted using the given data.

Substituting **“u = 0”** in eq.6, result is

**(0) = = … (i)**

Substituting “u = 1” in eq.6, result is

**(1) = + + + … (ii)**

To determine the b, c, and d coefficients, the derivative of above polynomial equation is taken.

Taking the 1st derivative at knots **“1 ≤ i ≤ n+1”**

**(0) = = … (iii)**

**(1) = + 2 \* + 3\* = … (iv)**

From the (i), (ii), (iii), (iv)

**=**

**=**

**= 3( - ) - 2 \* -**

**= 2( - ) + +**

In the above equations, and are known, but andare unknown which makes it difficult to substitute and converge to the solution. To determine the and  **,**  the first derivative of the polynomial is differentiated again and on resubstituting **“u = 0”** and **“u = 1”** as done after the first derivative results to complex equations which on simplifying, reduces to equations as shown below,

**+ 4 \* + = 3( - )** 2 ≤ i ≤ n {INTERIOR points}

**2 \* + = 3\* ( - )** {LEFT end}

**+ 2 \* = 3 \* ( - )** {RIGHT end}

In a huge dataset, these equations form a tridiagonal matrix as shown below,

**=**

Solving the above tridiagonal matrix, the coefficient of the polynomial equation is obtained. Thomas algorithm is used in this experiment to solve the above tridiagonal matrix in MATLAB.

Results

# Least Squares Method

***Linear Least squares method***



Linear Least squares method – Output Plot

The above figure displays the line plot achieved after implementing the least squares method. The equation achieved is as shown below,

**Y = 9.253e-11 \* x + 0.0591 ….** Line Equation achieved

Text

Description automatically generated

Linear Least squares – Computation Time

It takes 1.1 seconds for the program to run and give the output plot.

***Generic Least Squares method – Increasing order until “mse < tolerance”***



Generic LS – MSE<Tolerance

Text

Description automatically generated with medium confidence

Generic LS – MSE < Tolerance – Computation Time

It takes 1.2 seconds for the program to process and give the output plot. The algorithm stops at second order.

***Generic LS – Plotting till order 18***

# Cubic Splines Method



Output Plot

Graphical user interface

Description automatically generated with medium confidence

Output plot zoomed in.

The first figure above, displays the spline curve fit for the whole data set. Since the size of the sample data set is huge, the first image does not help much in concurring to something. But on zooming into the same plot, the cubic polynomial is plotted between each intermediate points. The highlighted regions in the bottom figure show the difference between the piecewise spline and cubic spline.

A picture containing text

Description automatically generated

Generic CS - Computational time

It takes ~2 seconds, see above figure, for the program to process and provide the outputs displayed earlier.

Discussion

Appendix

# Least Squares Method

***Program for linear least square method***

% Program for least squares solving for a line

% Y = Ax + B

close all

clear all

clc

% Importing the data

data = readtable('test.csv');

% Raw plot

plot(data.SampleTimeFine, data.dv\_1\_,'\*')

% 2 equations

% sum(yx) = sum(x^2)\*A + sum(x)\*B

% sum(y) = sum(x)\*A + (N)\*B

% Defining x and y

x = data.SampleTimeFine;

y = data.dv\_1\_;

sigma\_x = 0;

sigma\_y = 0;

sigma\_xy = 0;

sigma\_x2 = 0;

for i = 1:length(x)

sigma\_x = x(i) + sigma\_x;

sigma\_y = y(i) + sigma\_y;

sigma\_xy = x(i)\*y(i) + sigma\_xy;

sigma\_x2 = x(i)^2 + sigma\_x2;

end

% matrix

RHS = [sigma\_xy ; sigma\_y];

X = [sigma\_x2 sigma\_x; sigma\_x length(x)];

C = X\RHS;

y\_f = C(1)\*x + C(2);

hold on

plot(x,y\_f,'LineWidth',2)

grid on

axis padded

legend('Raw data','LS method')

***Generic Least squares – Increasing order until “mse > tolerance”***

% Generic least squares

close all

clear all

clc

tic

% Importing the data

data = readtable('test.csv');

X = data.SampleTimeFine;

Y = data.dv\_1\_;

% Raw Plot

plot(X,Y)

% Starting MSE

mse = 1;

MSE\_arr(1,1) = 1;

% defining the starting order of the polynomial

O = 1;

while mse>1e-6

% Getting a matrix for all x values

x = zeros();

for j = 1:O+1

o\_n = O+j-1;

for i = 1:O+1

x(i,j) = sum(X.^(o\_n-i+1));

end

end

x\_req = x';

% Getting RHS - Y matrix

y = zeros();

for i = 1:O+1

y(i,1) = sum(Y.\*X.^(i-1));

end

% Getting the coefficients

c = zeros();

c = inv(x\_req)\*y;

% Solving to get Y\_LS considering all the X values

Y\_LS = zeros();

for i = 1:length(X)

y\_temp = 0;

o\_n = O;

for j = 1:length(c)

y\_temp = y\_temp + (c(j,1)\* X(i,1).^(o\_n));

o\_n = o\_n - 1;

end

Y\_LS(i,1) = y\_temp;

end

% Computing the MSE and incrementing the order

mse = error(Y\_LS,Y,length(Y));

MSE\_arr(O+1,1) = mse;

O = O+1;

end

% fprintf("Final MSE = %f ", MSE\_arr(length(MSE\_array),1));

% Plotting the curve

hold on

plot(X,Y\_LS,LineWidth=3)

legend('Raw Plot','Polynomial Curve')

axis padded

grid on

toc

***Generic Least squares – Plotting curves upto order 18***

# Cubic Splines Method

***Generic Cubic Splines Methods***

% Cubic splines for IMU data

close all

clear all

clc

% Computing time to run the algorithm

tic

% Importing the csv data

data = readtable('test.csv');

% Defining the abscissa and ordinate

X = data.SampleTimeFine;

Y = data.dv\_1\_;

% Defining the number of samples

N = length(X)-1;

% Defining the initial variables

A = ones(N,1);

B = ones(N+1,1);

B = B\*4;

B(1) = 2;

B(N+1) = 2;

C = ones(N,1);

RHS(1) = 3\*(Y(2) - Y(1));

RHS(N+1) = 3\* (Y(N+1)- Y(N));

for i = 2:N

RHS(i) = 3\*(Y(i+1) - Y(i-1));

end

% Calling Thomas algorithm

D = thomas\_alg(A,B,C,RHS);

% Defining 4 interested variables

a = zeros(N,1);

b = zeros(N,1);

c = zeros(N,1);

d = zeros(N,1);

for i = 1:N

a(i) = Y(i);

b(i) = D(i);

c(i) = 3\*(Y(i+1)-Y(i)) - 2\*D(i) - D(i+1);

d(i) = 2\*(Y(i)-Y(i+1))+D(i)+D(i+1);

end

% Defining u parameter which is always b/w 0 & 1

u = linspace(0,1,101);

% plotting linear spline / raw plot

plot(X,Y,'o-');

hold on

for i=1:N

X\_CS = u\*(X(i+1) - (X(i))) + X(i);

Y\_CS = a(i) + b(i) \* u(:) + c(i) \* u(:).^2 + d(i) \* u(:).^3;

% plotting the cublic spline

plot(X\_CS,Y\_CS,'-m',LineWidth=1)

end

legend('Linear spline','Cubic spline')

xlabel('Sample time');

ylabel('Velocity');

title('Cubic Splines');

grid on

axis padded

toc

***Thomas Algorithm – To solve the Tridiagonal Matrix***

% Function to implement thomas algorithm

function D = thomas\_alg(A,B,C,RHS)

N = length(RHS);

C(1) = C(1)/B(1);

RHS(1) = RHS(1)/B(1);

B(1) = 1;

for k=2:N

B(k) = B(k) - A(k-1)\*C(k-1);

RHS(k) = RHS(k) - A(k-1)\*RHS(k-1);

if (k < N)

C(k) = C(k)/B(k);

end

RHS(k) = RHS(k)/B(k);

B(k) = 1;

end

D(N) = RHS(N);

for k = N-1:-1:1

D(k) = (RHS(k) - (C(k) \* D(k+1)));

end

D = D';

end