Octave Project 2 MAT300

Jon Galaz, Pablo Riesco

March 2022

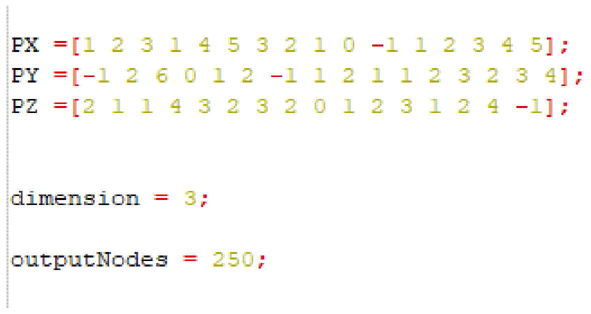
1 Cubic Splines

1.1 Problem Description

The problem this program solves is getting a piece wise defined parametric polynomial curve that goes through all the defined points. The input is expected to be typed in the file data.m (as vectors containing the X, Y and optionally Z coordinates of the points [lines 11, 12 and optionally 13 of said file]). Then, the user would have to set the dimension in which the user wants the plot to happen (2 for 2D or 3 for 3D [line 16]) and finally the number of nodes that would define the quality and precision of the output polynomial's plot (line 18).

1.2 Expected Input and Output

For example by having this input in data.m

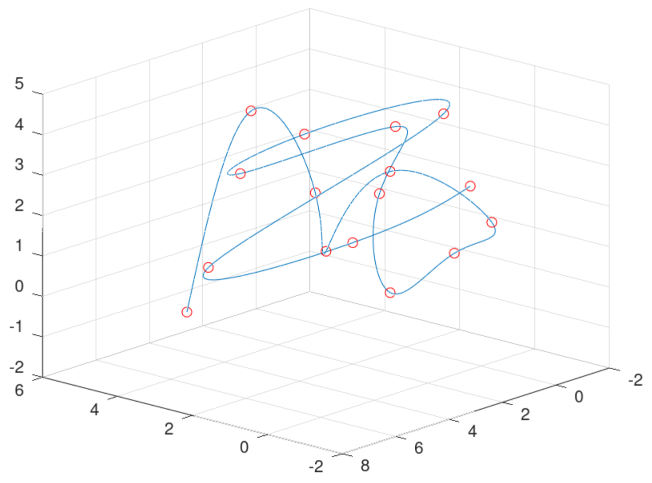


```
PX =[1 2 3 1 4 5 3 2 1 0 -1 1 2 3 4 5];
PY =[-1 2 6 0 1 2 -1 1 2 1 1 2 3 2 3 4];
PZ =[2 1 1 4 3 2 3 2 0 1 2 3 1 2 4 -1];


dimension = 3;

outputNodes = 250;
```

The correct output would be the following image



2 Cubic Spline Method

Given a set of points $\{(t_0, P_0), (t_1, P_1), ..., (t_n, P_n)\}$ with $t_i < t_{i+1}$ and $t \in$ a regular mesh $[0, n]$ satisfying $0 = t_0 < t_1 < t_2 < ... < t_n = n$ with equal distance between nodes, we can define a cubic spline as a piece wise defined set of polynomials each one of them bound to an interval. The polynomial $p \in P^n_{3,2[t_0,t_1,...,t_n]}$ can be defined as

$$\begin{cases} \mathrm{p}_1(t) & t \in [t_0, t_1) \\ \mathrm{p}_2(t) & t \in [t_1, t_2) \\ ... & \\ \mathrm{p}_n(t) & t \in [t_{n-1}, t_n] \end{cases} \tag{1}$$

where $p_i(t) \in P_3$, $i = 1, 2, ..., n$. Moreover we assume that p is continuous with first and second derivatives continuous in $[t_0, t_n]$. Then, we can define a basis for the space $P^n_{3,2[t_0,t_1,...,t_n]}$ as $B = \{1, t, t^2, t^3, (t - t_1)^3_+, (t - t_2)^3_+, ..., (t - t_{n-1})^3_+\}$. Our polynomial $p(t)$ can be expressed as

$$p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4(t - t_1)^3_+ + a_5(t - t_2)^3_+ + ... + a_(n + 2)(t - t_{n-1})^3_+$$

Then, the cubic spline is determined by the solution $(a_0, a_1, a_2, ..., a_{n+2})$ of the system of equations

$$\begin{cases} \mathrm{P}_0 = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4(t_0 - t_1)^3_+ + a_5(t_0 - t_2)^3_+ + ... + a_(n + 2)(t_0 - t_{n-1})^3_+ \\ \mathrm{P}_1 = a_0 + a_1 t_1 + a_2 t_1^2 + a_3 t_1^3 + a_4(t_1 - t_1)^3_+ + a_5(t_1 - t_2)^3_+ + ... + a_(n + 2)(t_1 - t_{n-1})^3_+ \\ ... \\ \mathrm{P}_n = a_0 + a_1 t_n + a_2 t_n^2 + a_3 t_n^3 + a_4(t_n - t_1)^3_+ + a_5(t_n - t_2)^3_+ + ... + a_(n + 2)(t_n - t_{n-1})^3_+ \end{cases} \tag{2}$$

for which we also impose the conditions $p''(t_0) = p''(t_n) = 0$ as equations. As a result we would get the following matrix

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & c_0 \\ 0 & 1 & 0 & 0 & c_1 \\ ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 1 & c_n \end{array}\right)$$

where $(c_0, c_1, c_2, ..., c_n)$ would be the the coefficients of the X, Y and/or Z coordinates that would multiply our basis B in order to obtain the piece wise defined polynomial p as follows

$$\begin{cases} c_0 + c_1 t + c_2 t^2 + c_3 t^3 \quad t \in [t_0, t_1) \\ c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 (t - t_1)^3 \quad t \in [t_1, t_2) \\ ... \\ c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 (t - t_1)^3 + ... + c_n (t - t_n)^3 \quad t \in [t_{n-1}, t_n] \end{cases} \tag{3}$$

3 Relation Between Method and Code

In line 17, we make n equal the amount of input points given. We make n equal to this to reduce the amount of computations that are needed if we make the number of points equal to n-1, so for this section, we will assume that n is equal to the amount of points to be traversed by the curve. In line 19, we construct the mesh of nodes from 0 to n-1 using n nodes. The output mesh is constructed in line 21, going from 0 to n-1 using the precision given in the input.

Firstly, the matrix used to calculate the coefficients of the curve is set up. This is done from code block 1 to 4. In code block 1, the first four columns are placed using the mesh of nodes corresponding to the first four values in the cubic spline basis. In code block 2, we continue adding values following the cubic spline basis, but now manually, as negative values are not to be placed. In code block 3, the point's x, y and z values are placed in the last 3 columns. Finally, in code block 4, the last two additional constraints are placed in the last two rows, stating that the double derivative at the first and last values of the mesh of nodes should be 0. At line 72, RREF is used to get the coefficients of the polynomial.

Having the coefficients of the polynomial, we evaluate it in code block 5, 6 and 7. It is done separately to include some optimizations for efficiency. In code block 5, the first value is added to the results. In code block 6, the second, third and fourth values following the cubic spline basis are added. Finally, in code block 7, the remaining values are added with the positive value constraint. After this, the values are plotted.

4 Examples

For the sake of simplicity, instead of comparing the code using the example in section 1.2, we will test it using a curve with 4 points: (0, 1), (1, 3), (2, -1), (4, 0). Since we have 4 points, the regular mesh is of 4 Ts: Ts[0] = 0, Ts[1] = 1, Ts[2] = 2, Ts[3] = 3. The code uses this mesh as well.
Comparing the results of the coefficients of the curve computed by the code by the analytical solution, we get the following results (first table are the results calculated analytically and the second are the results when done running the code):

0	1
16/15	59/15
0	0
-1/15	-29/15
2/5	28/5
-3/5	-27/5

0.00000	1.00000
1.06667	3.93333
0.00000	0.00000
-0.06667	-1.93333
0.40000	5.60000
-0.60000	-5.40000

Since the results are the exact same, we can conclude that the code calculates the coefficients correctly. Next, we will check that the results of the curve are correct. Checking the output given in the first image, the red dots correspond to the points that are to be crossed. We can clearly see that the output curve trespasses through these points. However, in order to check the results even further, we will also test that the curve goes through the same points as the analytical result when using the same T values.

0	1	1
1	1	3
2	2	-1
3	4	0

0.00000	1.00000	1.00000
1.00000	1.00000	3.00000
2.00000	2.00000	-1.00000
3.00000	4.00000	0.00000

Here we can see that the points are in fact crossed exactly, so we can conclude that the code computes the curve correctly.

5 Observations

Since this method of computing the curve is done using values with exponents of maximum of three, it gives results which are very stable. Unlike what happens when we use other methods such as Lagrange and Gaus Jordan, the curve at the extremes are stable regardless of the amount of points that we give as input. In fact, testing the code with 24 points works and is the curve is stable at the extremes.

This method is also good when we decide to add more points. Adding additional points does not affect previous parts of the function that much, making this method worthy in these case.

The most problematic part of this method is the computation time. In order to create the matrix, it is necessary to iterate checking that the values of the result are within certain ranges in order to add more parts of an exponent or not. In this aspect, other methods may be more efficient.