Math 151A: Applied Numerical Methods

Winter 2016, Lecture 2

Project 2: Cubic Spline Interpolation

Applied to Vehicle Pathing

Theodore Nguyen

704-156-701

User guide:

Note that a more detailed, step-by-step explanation is found in the comments of each code file. Inside this submission are the following files:

Files I created and used primarily to facilitate the project:

CroutSolver.m – This file is essentially the linear system of equations solver for tridiagonal systems. Implemented almost entirely identical to the psudocode in section 3.5 of the book, it takes a square, tridiagonal matrix **A,** and a vector **r** with the same dimension as a side of **A**. The function proceeds to solve the linear equation **Am = r.** The function, naturally, outputs a vector **m** of same dimension as r which are the solutions to the linear equation.

function [m] = CroutSolver(A, r)

proj2.m – Forgive me for the badly named code, but this function essentially does this project for us. It takes two arguments, **option** and **file**. **Option** is of type integer; if it equals 1, then it will eventually interpolate using Natural Boundary Condition, while if it equals 2 then it will interpolate using Clamped Boundary Condition. If **option** equals anything else the function will return with an error code. **File** is of type string, and essentially is the name of the data file we will take as input. In our case we will probably pass 'data.mat' as **file. File** must be formatted identically to data.mat – that is, it must be of some structure type and have an element labeled "ip" and "ip" must be a **n x 3** array, whose columns are, left to right, the t-values, x-values, and y-values of data we will use for interpolation. The function returns four items, **SplineX, SplineY, Sx, Sy. SplineX** and **SplineY** are of type function handle and are the entire interpolated piecewise concatenated cubic spline polynomials with respect x and t, and y and t respectively. They are however, defined anonymously and recursively, and as a result have many limitations. They are continuous and vectorized, so you can evaluate the polynomial at any t that is in the range of t-values in the original loaded file (outside this range, the functions will evaluate to zero) and they can be evaluated at a set of t-values for plotting, but their derivative cannot be taken and they have many more limitations. **Sx** and **Sy** are the piecewise components of **SplineX** and **SplineY.** That is, literally, if we added up all the elements of **Sx,** we will obtain **SplineX** as a result (and the same applies to **SplineY)** – this was exactly how the **SplineX** and **SplineY** were, so to say, recursively defined. **Sx** and **Sy** are of type cell, where each entry of the cell is an anonymously defined polynomial.

Proj2.m implements the entire Cubic Spline Interpolation. Essentially, it takes the n x 3 data matrix, and then interpolates with respect to both x(t) and y(t) and gives the final piecewise polynomials. More details on this can be found by reading the comments in the code, but generally the code is pretty self-explanatory based off the project specification.

function [SplineX, SplineY, Sx, Sy] = proj2(option, file)

quickplot.m – This script (not a function) does exactly what it is named – it plots the necessary plots for this project. It calls proj2 to get the necessary interpolation polynomials, then it creates many t values for where we want to evaluate the polynomials at. Then, it evaluates all the polynomials for all the cases – x, y, Natural boundary, Clamped boundary – and stores the results into another array. Eventually, it plots **X vs t**, **Y vs t, X vs Y** (with direction!) in that order. It then computes the derivatives numerically and kind of through the cubic spline functions, and then plots both of the latter on the same plot as a vector field.

Files that seem like they satisfy the submission requirements of the project: These files literally take everything out the ones above; they're just formatted to be functions to appear to satisfy the conditions of the two functions we needed in the specification. Essentially, these are extra functions.

(Construction of cubic splines as described in Section 2.2. The implementation should provide options to use natural boundary condition and clamped boundary condition.)

CUBICSPLINE.m – Takes an integer **option** as argument, if 1 uses Natural Boundary Condition, if 2 uses Clamped Boundary Condition, returns error if anything other than 1 or 2. Takes two arrays **T** and **X,** where **T** is the data points corresponding to the horizontal axis and **X** is data points corresponding to vertical axis. Interpolates via Cubic Spline Interpolation. Returns vectors **B, C, D** which contain coefficients of the Cubic Spline polynomial.

function[B, C, D] = CUBICSPSLINE(option, T, X)

(Functions for solving the tracking problem as described in Section 2.1, using both natural boundary condition and clamped boundary condition.)

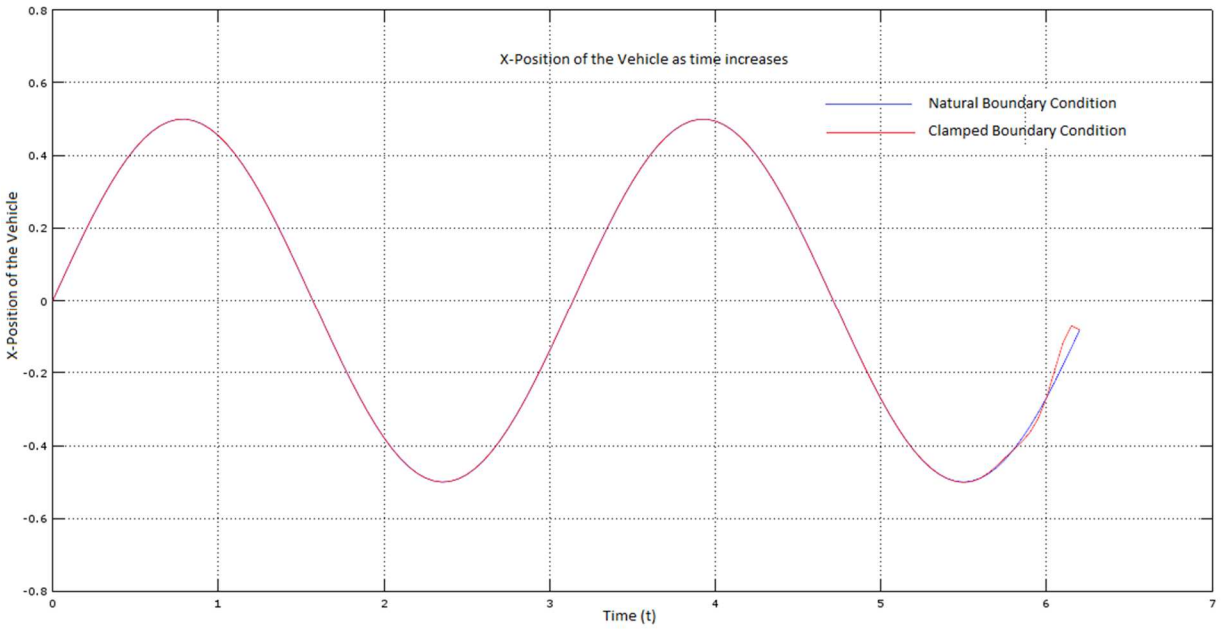
VEHICLETRACK.m – takes a string **file** as argument that is the name of the data file formatted identically to the form of the provided data.mat for this project. Returns 12 arguments

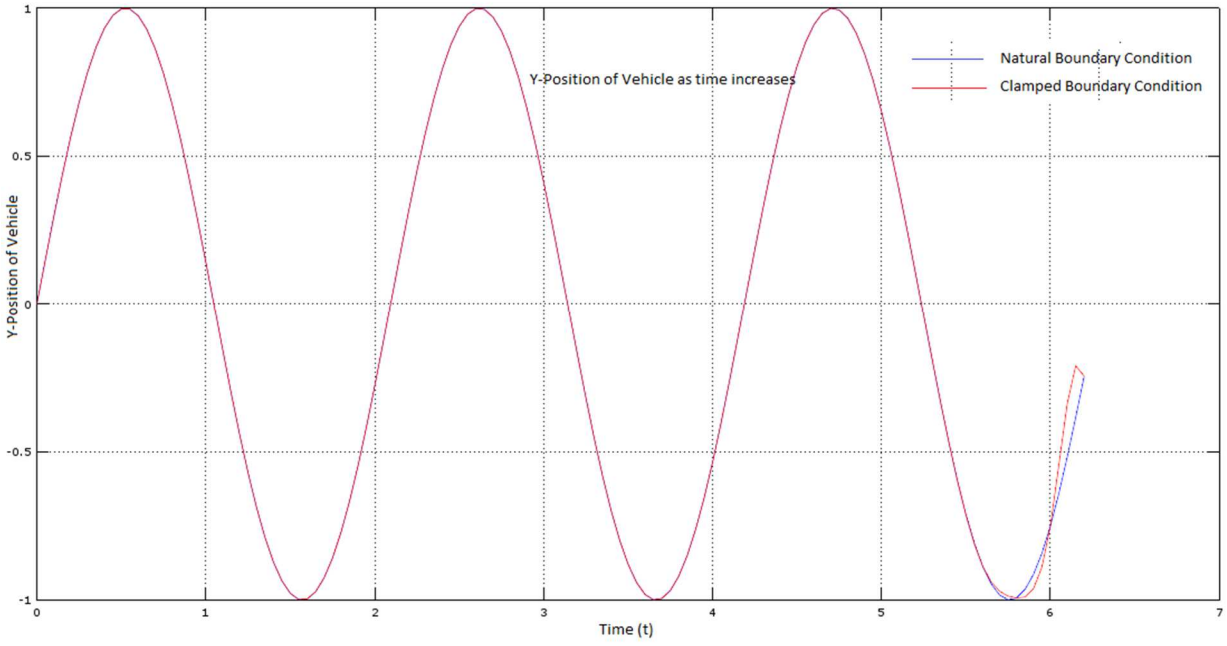
Bxi, Cxi, Dxi, Byi, Cyi, Dyi, Bxii, Cxii, Dxii, Byii, Cyii, Dyii

Where each B, C, and D trio are each vectors containing the coefficients to the cubic spline interpolation.
If the trio has an "x" in the name, then it is interpolating between x and t.
If the trio has a "y" in the name, then it is interpolating between y and t.
If the trio has one "i" in the name, then it used natural boundary condition in interpolating.
If the trio has two "ii" in the name, then it used clamped boundary condition in interpolating.

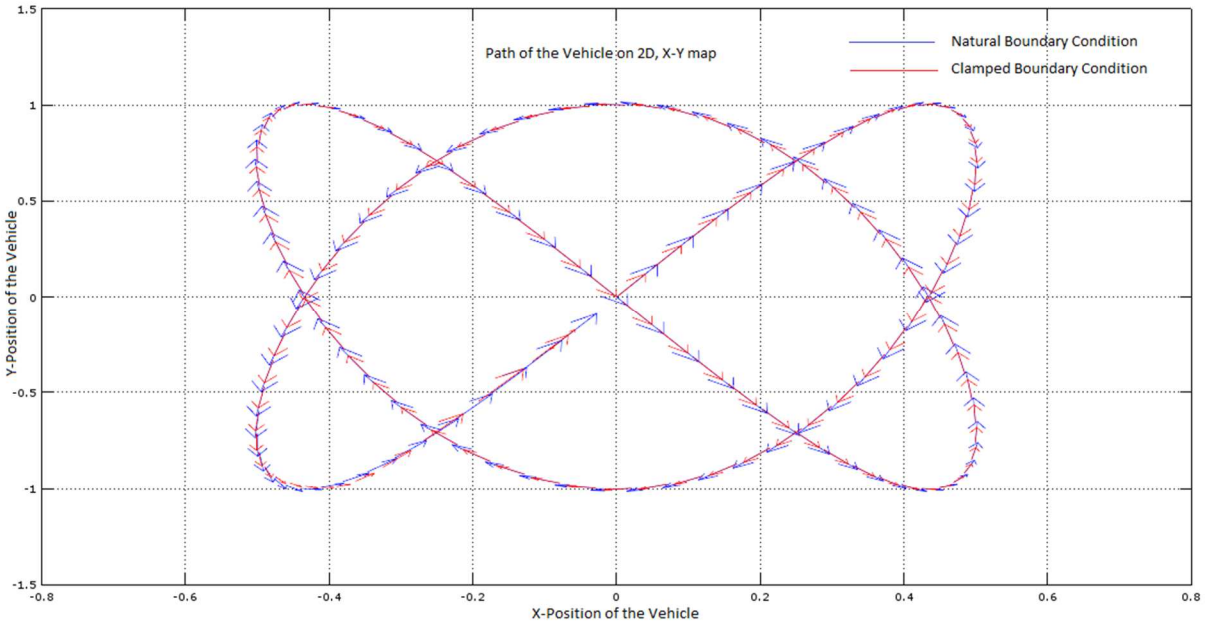
function[Bxi, Cxi, Dxi, Byi, Cyi, Dyi, Bxii, Cxii, Dxii, Byii, Cyii, Dyii] = VEHICLETRACK(file)

Plots of X(t) and Y(t):





Solution – the path of the vehicle:



The above graph was created by plotting X1(t) and Y1(t) at points T, where T is the set of points from 0 to 6.2 with spacing of 0.05, as well as X2(t) and Y2(t) at points T. All four of these functions are the Cubic Interpolating Polynomials generated from the previously mentioned functions. X1(t) and Y1(t) are shown in blue and were created adhering to the Natural Boundary Condition, while X2(t) and Y2(t) are shown in red and were created adhering to the Clamped Boundary condition. For convenience, the velocity vector is shown at several points along the plot – this gives us a more real-life plot of the path of a vehicle, as we can see where the vehicle started at the origin and follow it throughout its path via the vector arrows and the pathing.

Experiments:

We have points T. All of T's elements are equally spaced from one another, a length h. This h is 0.2 for our dataset. We evaluate our cubic spline function (specifically, the one generated using the Natural Boundary Condition) at all t in T, and then compute the gradient on the resulting vectors to get the derivative of our spline function at T. These are done for both x and y components and are stored in a respective U vectors for both x and y.

We then numerically compute the derivative at all t in T using the 3-point midpoint formula (and 3-point endpoint formula for both endpoints). This uses only the raw data we were given earlier. These values are stored in a respective V vectors for both x and y.

Both U and V vectors were plotted on the same plot using the "quiver" function in MATLAB, and the resulting vector field was generated.

Three-Point Endpoint Formula

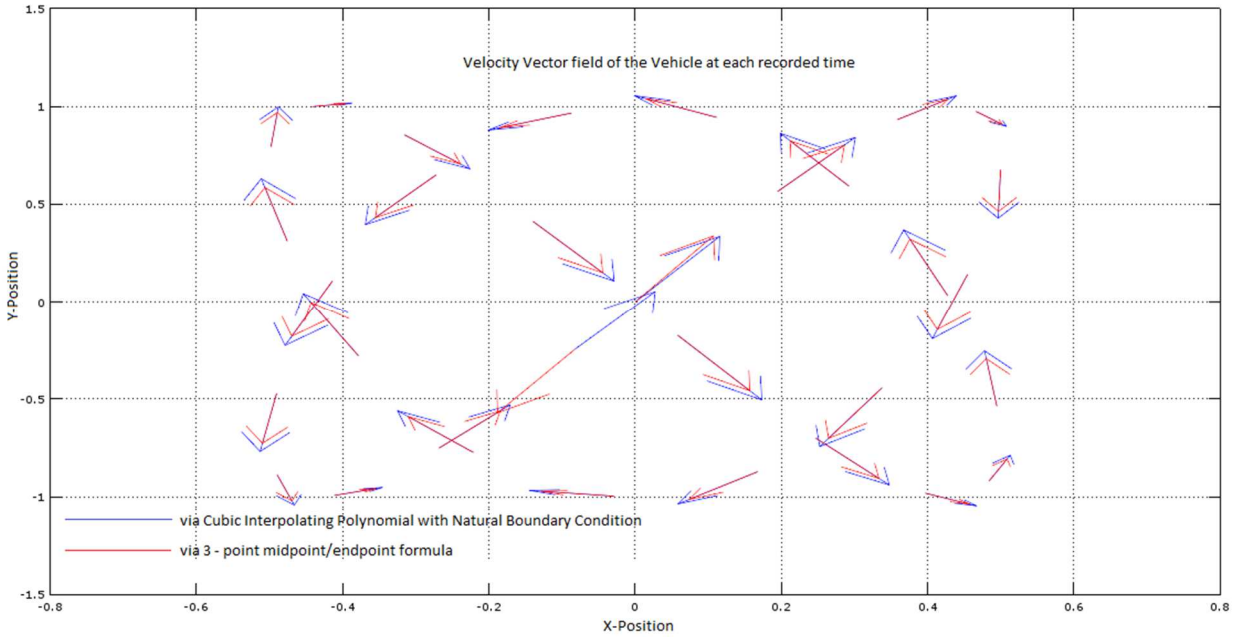
- $f'(x_0) = \frac{1}{2h}[-3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h)] + \frac{h^2}{3}f^{(3)}(\xi_0)$,

where ξ_0 lies between x_0 and $x_0 + 2h$.

Three-Point Midpoint Formula

- $f'(x_0) = \frac{1}{2h}[f(x_0 + h) - f(x_0 - h)] - \frac{h^2}{6}f^{(3)}(\xi_1)$,

where ξ_1 lies between $x_0 - h$ and $x_0 + h$. (Burden)



As we can see above in the 3-point midpoint and endpoint formulas, the error term is of degree 2 by looking at the degree of the h-term. We let S be the unique clamped cubic spline interpolant to a function f respect to nodes in the set X, where X has n+1 elements. The error bound of interpolation is seen:

If $f \in C^4[a, b]$, let $M = \max_{a \le x \le b} |f^4(x)|$

$$h = \max_{0 \le j \le n-1} \left(x_{j+1} - x_j\right)$$

$$\max_{a \le x \le b} |f(x) - S(x)| \le \frac{5Mh^4}{384}$$ (http://www3.nd.edu/~zxu2/acms40390F12/Lec-3.4-5.pdf)

Then, cubic spline interpolation has an error term of 4, greater than the 2 of 3-point formulas. We say that the cubic spline velocity field is more likely to be accurate, as the error at the highest power is the smallest.