Project 2 Report

User Guide:

Solve Tridiagonal Method: `function [ m ] = solve_tridiagonal(n, A, r)`

Description:

This function solves the linear system $A*m=r$ where A is a tridiagonal matrix. It does so by implementing the Crout Factorization for Tridiagonal Linear Systems (Algorithm 6.7). This function is used to solve equation 3 as specified in the project. The function returns the solution m as a row vector.

This function accepts as input:

- A: A tridiagonal matrix
- n: the dimensions of matrix A
- r: The right hand side of the equation (n dimensional column vector)

Calling Syntax:

To call the function we must supply a value for its parameters. So for example:

```
A=magic(4);
r=[1;2;3;4];
solve_tridiagonal(4,A,r)
```

will return `m=[0.0729  -0.0834  0.2553  0.1710]` since this is the solution to the system $A*m=r$

Solve Velocity: `function [] = solve_velocity( n,h,s_x, s_y,x,y, time)`

Description:

This function attempts to answer Section 2.3 in the project. It finds the velocity of the car at the data points given. It calculates the velocity using two methods: first by

differentiating the x-splines and y-splines and second by using the numerical solution of the three-point midpoint method.

This function accepts as input:

- n: The number of measurements
- h: As defined in Section 2.2.2 of the project
- s_x: An array that represents the x-cubic splines, i.e. $S^x(t)$
- s_y: An array that represents the x-cubic splines, i.e. $S^y(t)$
- time: An array representing the data points $t_0 \ldots t_n$

Solve Tracking: `function [] = solve_tracking( n,s_x, s_y, time)`

Description:

This function does not return anything but plots the solution to the tracking problem as specified in Section 2.3 of the project. This routine produces three plots $S^x(t)$ vs t, $S^y(t)$ vs t and $S^y(t)$ vs $S^x(t)$.

Find r: `function [ r ] = find_r( h, n, coord)`

Description:

This is a very simple function that calculates r in the equation $A * m = r$. If the coordinate is x then it returns r_x while if coordinate is y it returns r_y.

This function accepts as input:

- h: Array of time differences as defined in Section 2.2.2 of the project
- n: The number of data

- coord: This is the coordinate; either x-data points or y-data points

Calling Syntax:

To call the function we must supply a value for its parameters. So for example:

```
r_x=find_r(h, n, x);
```
finds r_x.

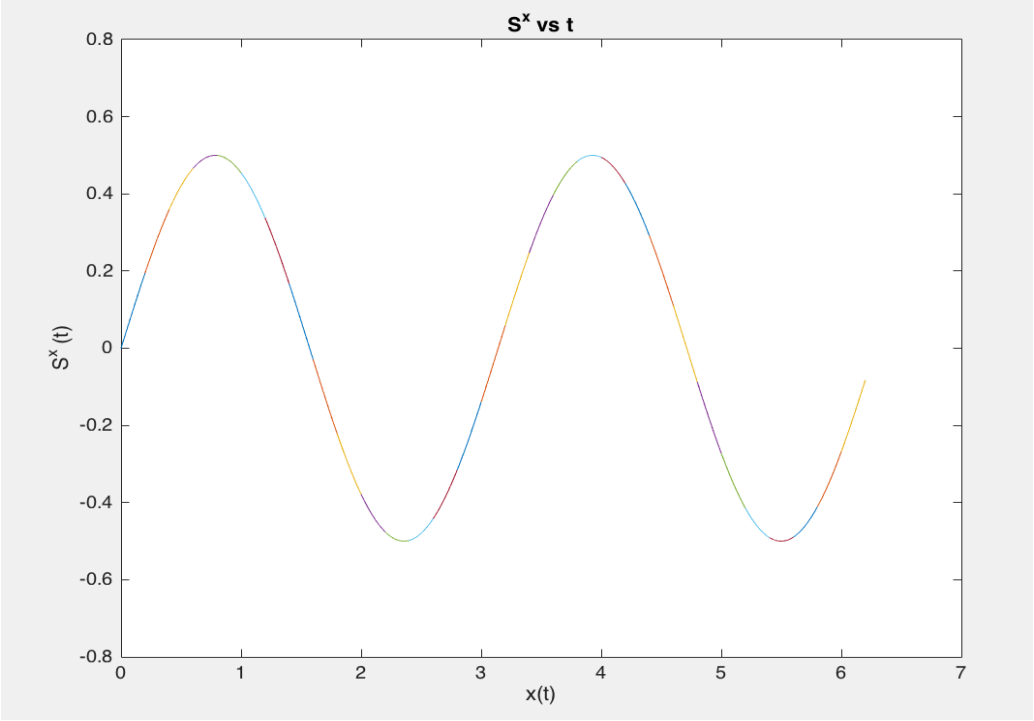
INIT: `(script) INIT`

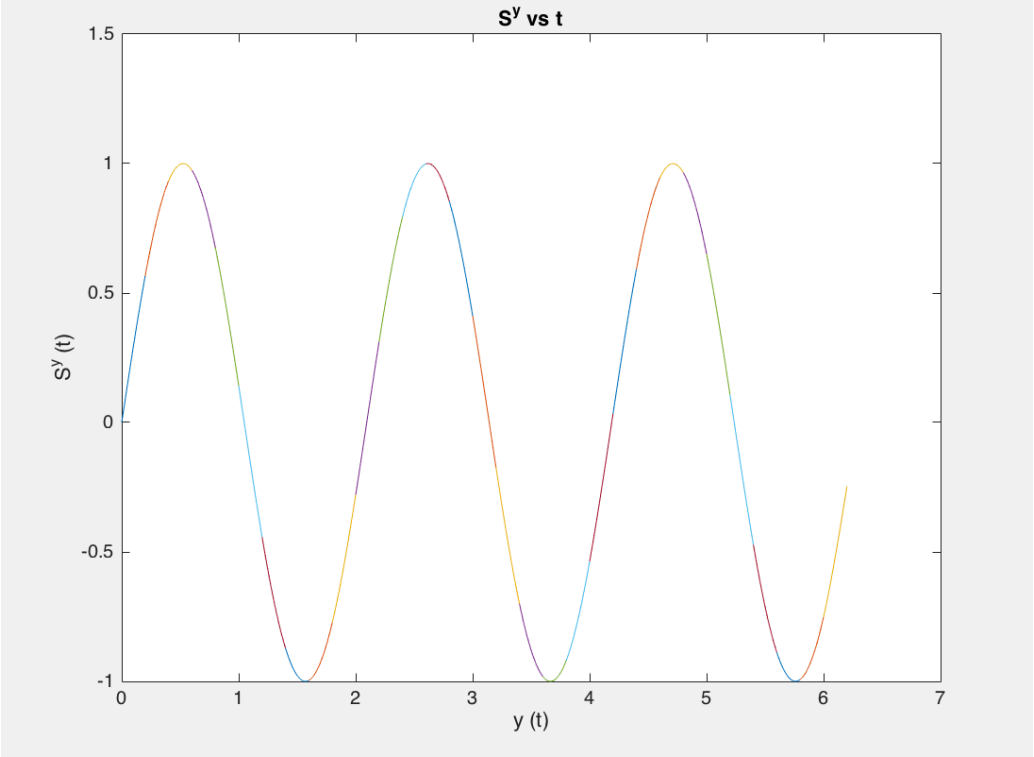
Description:

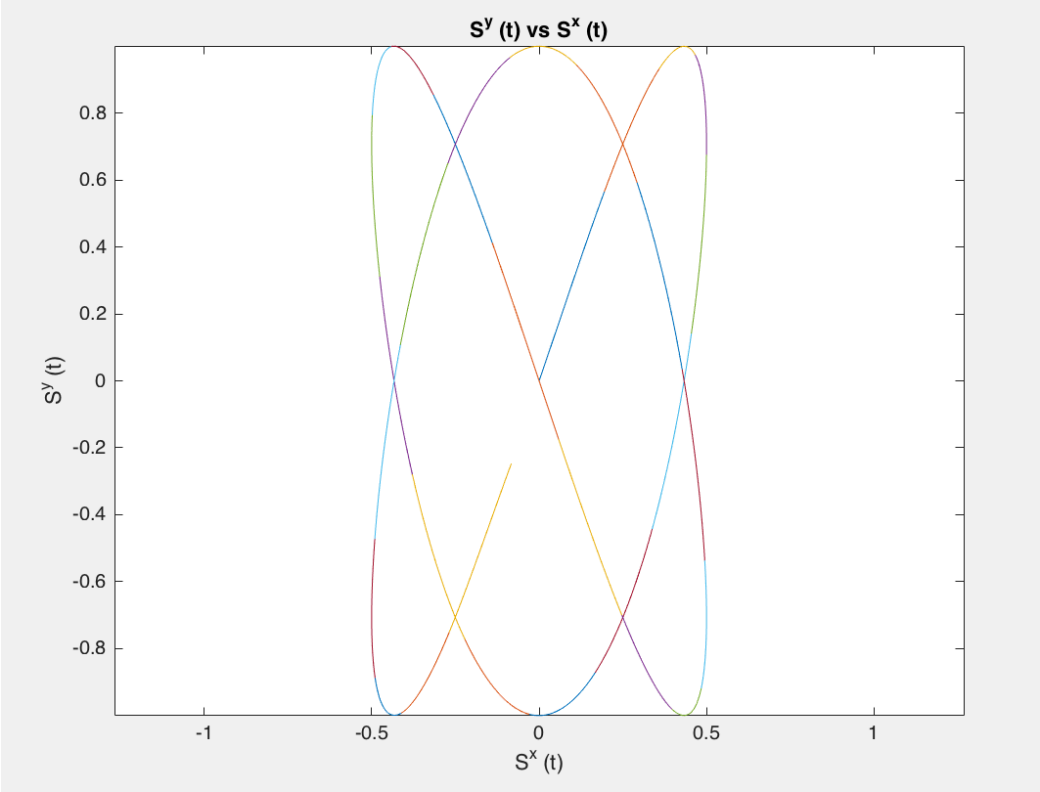
This is a script that initializes the solvers described above and also calculates the x-splines as well as the y-splines. Before the script runs it loads the "data.mat" file. The script first prompts the user to enter whether he wants to use Natural Boundary Condition or the Clamped Boundary Condition. It then sets up the matrix *A*, solves the system $A*m=r$ to find the value of the vector *m* and then calculates the coefficients. The splines are then represented as symbolic expressions and are stored in the arrays s_x and s_y.

Solutions:

Below are the graphs of $S^x(t)$ vs t, $S^y(t)$ vs t and $S^y(t)$ vs $S^x(t)$ as they were produced by the solve_tracking function. The natural boundary condition was chosen.

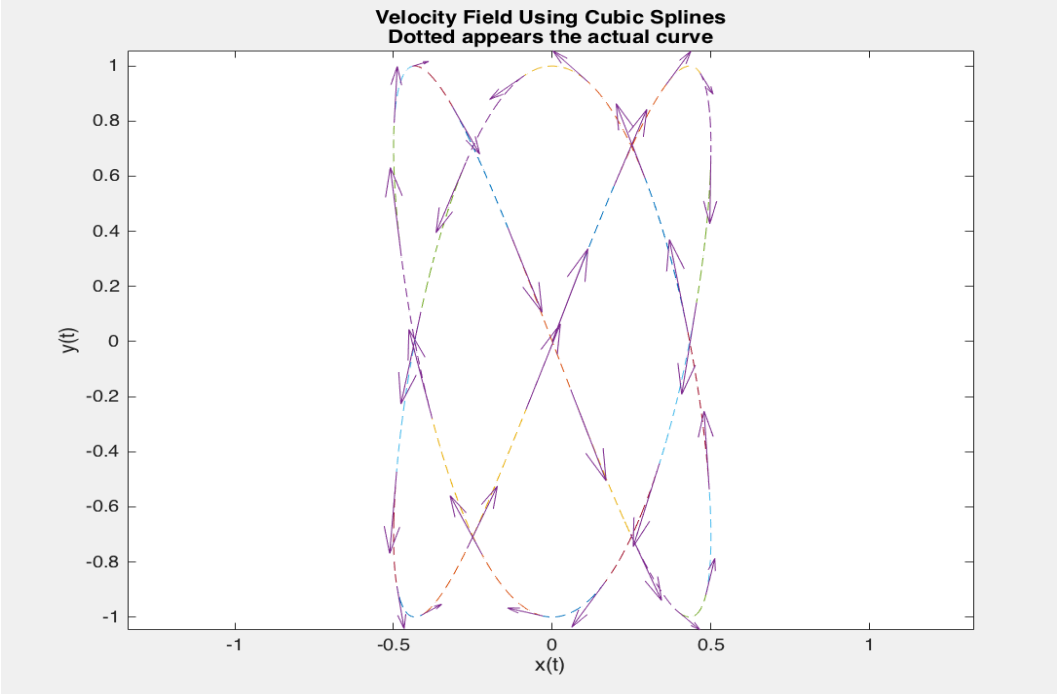


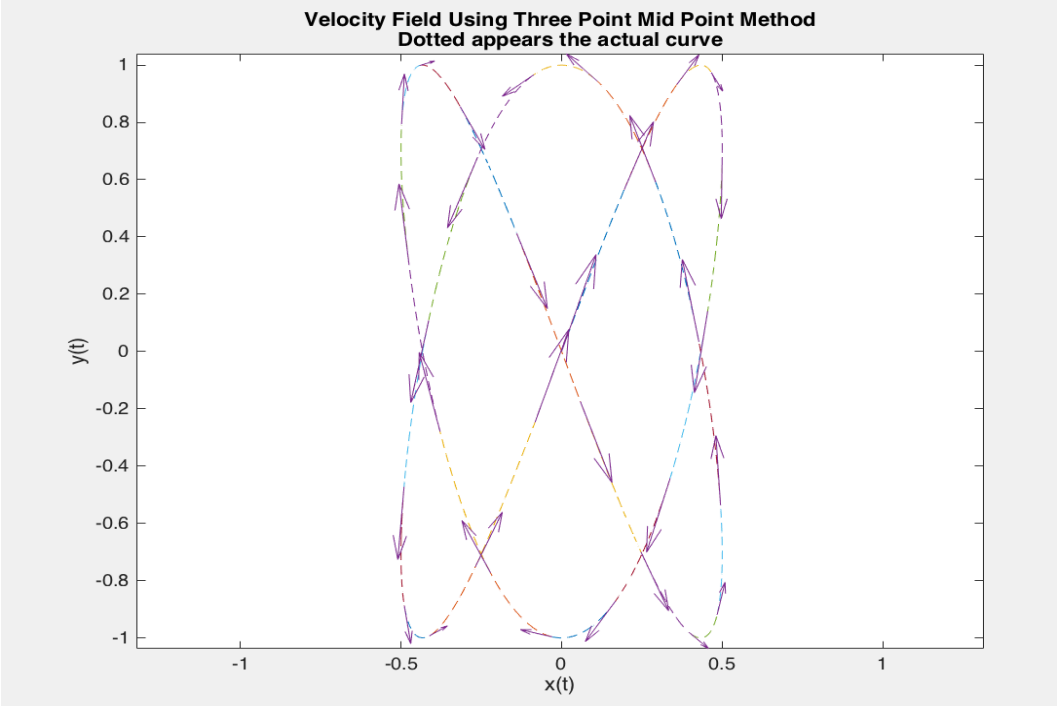


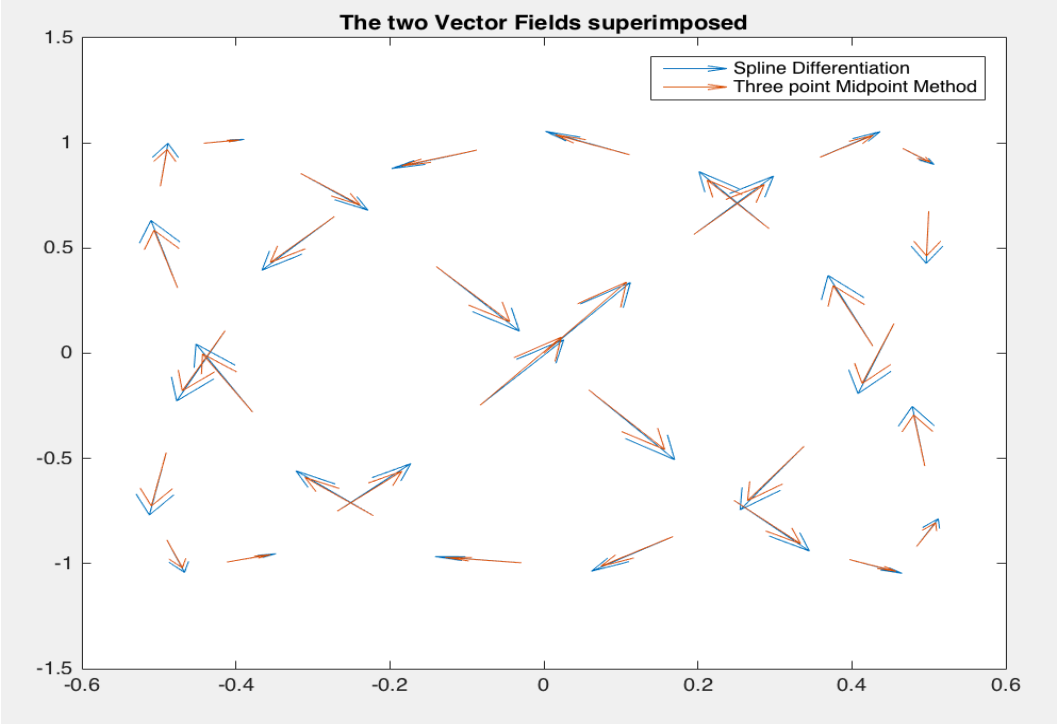


Experiments:

Below are the graphs of the velocities produced by the quiver function. In all of the graphs below the natural boundary condition was chosen arbitrarily.




Velocity Field Using Three Point Mid Point Method
Dotted appears the actual curve
y(t)
x(t)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.5
0
0.5
1


The two Vector Fields superimposed
Spline Differentiation
Three point Midpoint Method
1.5
1
0.5
0
-0.5
-1
-1.5
-0.6
-0.4
-0.2
0
0.2
0.4
0.6

Which velocity field is likely to be more accurate?

Simply by looking at the superimposed velocity field we see that the two velocity fields are very close to one another. This small script gives us that:

```
u_xT=0;
v_xT=0;
for i=[1:1:n-1 ]
    u_xT=u_xT+u_xNum(i);
    v_xT=v_xT+v_x(i);
end

Average_ux=u_xT/(n-1);
Average_vx=v_xT/(n-1);
difference=abs(Average_ux-Average_vx)
```

On average, $|u(i)-v(i)|=0.0047$ for $0\le i\le n$. This is clearly a very small difference therefore we have to use a mathematical formula to decide which one is better. As discussed in class, the error in the derivate of the cubic spline interpolation method when clamped boundary conditions are used is:

If $f\in C^4$ and $|f^{(4)}(x)|\le M$

$$|f'(x)-s'(x)|\le\frac{9+\sqrt{3}}{216}*M*\max_{i=0..n-1}(h_i^{\,3}) \text{ with } h_i=x_{i+1}-x_i$$

Therefore we have that differentiating the cubic splines produces an error of:

$$e_{splines}=O(h^3)$$

In contrast for the tree point method:

If $f\in C^3$ then:

$$f'(x_0)=\frac{1}{2h}*\left[f(x_0+h)-f(x_0-h)\right]+\frac{h^2}{6}*f^{(3)}(\xi) \text{ with } \xi \text{ between } x_0-h \text{ and } x_0+h$$

Therefore the error produced by this numerical method is:

$$e_{midpoint}=O(h^2)$$

So we can conclude that $e_{splines} < e_{midpoint}$.

As a result the cubic spline differentiation produces a more accurate velocity field.