# Cubic Spline Fitting and Interpolation

Yazhi Fan (yf92) and Yijia Chen (yc2366)

## Implementation

**1. CRSmatrix.cpp/.h**
The first thing we did is fix the sparse matrix from program 2. The original sparse matrix had many problems such as no class structure and bad file organization.
This time, we implemented a sparse matrix structure which greatly reduced the number of parameters for all matrix methods. We added class member methods and constructor which made matrix operations much easier. We also re-implemented Jacobi method in top down programming style. Now our sparse matrix solver is much more concise than the previous implementation. The new version sparse matrix solver is also pushed to our github repository as sparseMatrix_ver2.0.

**2. cubicSpline.cpp/.h**
Our cubic spline method allows users to input x and y anchor points and return qx and qy which contains interpolated points. The method allows non-monotonic curves by introducing new variable t. Slope functions are calculated separately for x(t) and y(t). It also makes sure the curve is C-2 continuous at the point of enclosure if there is any by checking if the first point is the same as the last point.

**3. test.cpp/.h**
In order to validate the result with less user interaction, we added a method that check for second derivative continuity. This can make sure the result x and y are has continuous second derivative. More about this is explained in the test result section.

## Code Reuse and Code Improvement

1. Sparse Matrix from Project 2
2. Spline Interpolation from note 5 hacker practice
3. Helper functions such as file read, vector norm, computation time check

# Testing Results and Observations

**1. Continuous Second Derivative Test**

Our cubic spline algorithm is written based on continuous second derivative. So we reverse calculated the second derivatives of the result data set and tested whether it is C-2 continuous. Second derivative is found by finite difference:

$$U_{xxi} \approx \delta_x^2 U_i \equiv \frac{1}{\Delta x^2}(U_{i+1} - 2U_i + U_{i-1}).$$

One of the requirement of a continuous function is that its limit approaching from both sides are the same. So we compared second derivative at each point with its neighbor's. If the difference is within a threshold (currently set at 1E-7), then the output data satisfies C-2 continuous.
This test works the best when there is a large number of points (>10000) returned. The more points there are, the closer we can calculate the "limit". Large number of result points does not affect the computation time too much (O(n)).

Example command output for hacker practice problem with 6 anchor points and 10000 interpolated points:

```
file read completed in 0.35ms
file read completed in 0.044ms
slope function calculation completed in 1.124ms
spline interpolation completed in 29.916ms
Resulting curve has continuous second derivative, : 7.49762e-08
```

**2. Graphical Validations**

Although graphical validation is not the best way to validate because it requires user validation, it is very important in this program because it is the most intuitive way.

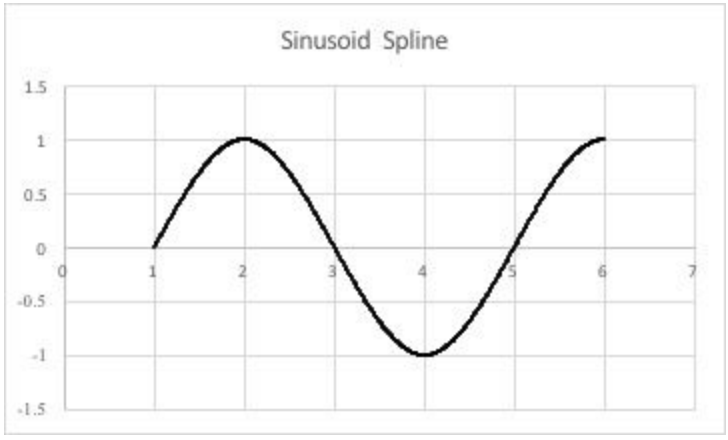
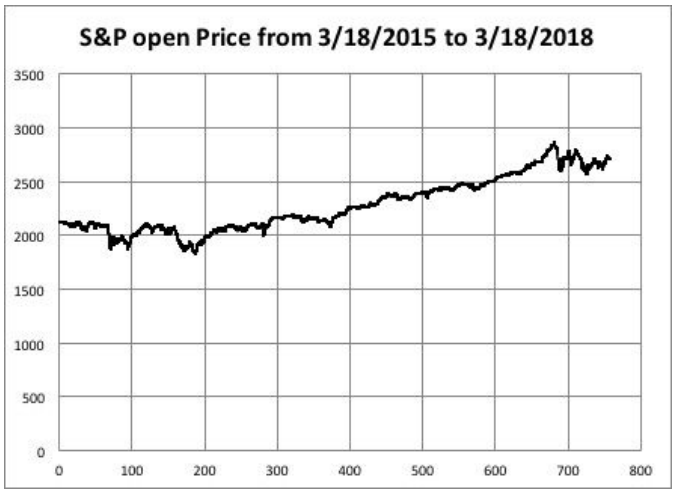



Figure 1. Hacker practice spline example, 6 anchor points

We also tested a very large set of S&P open price in the last 3 years that we found on Yahoo Finance. This set of data has over 700 anchor points, and we interpolated 10000 points from that. The process finished in very reasonable time.

```
file read completed in 0.743ms
file read completed in 0.769ms
slope function calculation completed in 2759.96ms
spline interpolation completed in 330.153ms
```




We also tested a few enclosed non-monotonic curves, two examples are shown below.

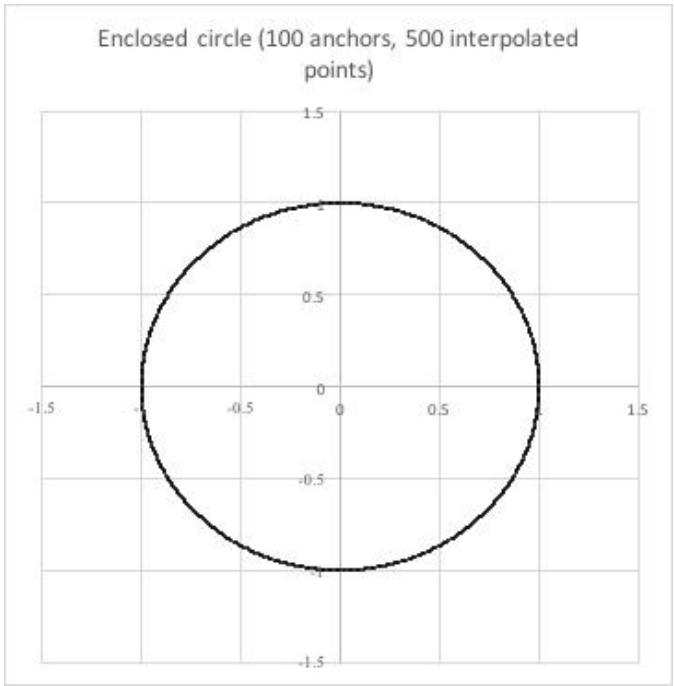



Figure 3. enclosed circle example

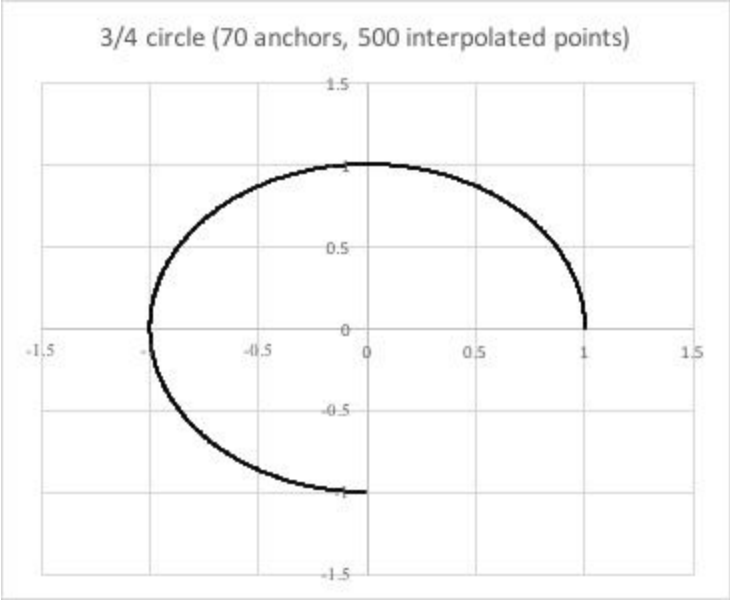


Figure 4. ¾ circle example

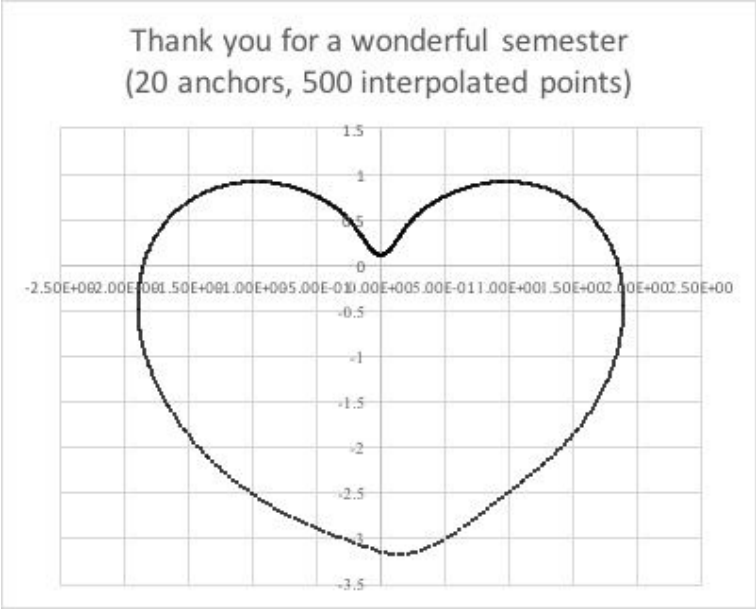


Figure 5. enclosed heart example