Spline Interpolation

Name: Hardik Kapoor
Entry Number: 2019MCB1220

Motivation behind such a method

The Polynomial Interpolation Problem

Is the problem of finding a smooth function P(x) (polynomial) which interpolates (passes) through the given set of points $\{(x_i,y_i)\}^N_{i=0}$, which are N+1 ordered pairs of x and y's.
This is possible due to the Weierstrass Approximation theorem, which states that for all continuous functions, there exists a polynomial such that absolute error between the given function and the polynomial is very less (less than $\varepsilon \ \forall \ \varepsilon>0$).

Why do we need such a problem?

In engineering and science, one often has a number of data points, obtained by sampling or experimentation, which represent the values of a function for a limited number of values of the independent variable (x), as we can’t store the whole set of points, due to memory limitations.

Hence, it is often required to interpolate, i.e., estimate the value of that function for an intermediate value of the independent variable.

Continued...

Suppose the formula for some given function is known, but too complicated to evaluate efficiently. A few data points from the original function can be interpolated to produce a simpler function which is still fairly close to the original. The resulting gain in simplicity may outweigh the loss from interpolation error.

Introduction to spline interpolation

There are many methods of interpolation, like Lagrange's interpolation, Newton's forward interpolation method, etc which use all the points to give a sufficient approximation to the function using a single polynomial of order N (number of points given).

But sometimes, if we keep the order of the polynomial fixed, and use different polynomials over different intervals, with the length of the intervals getting smaller and smaller, the resulting interpolating polynomial may be a better approximation to the original function. This method of using different polynomials over different intervals is known as **Piecewise Interpolation**.

Spline interpolation definition

Spline Interpolation is a method of piecewise interpolation, which finds an interpolating function that is sufficiently smooth and does a better approximation to f(x) (the original function).

Algorithm

Introduction

A spline interpolating function of degree d with nodes x_i ,i=0,1,...n ($x_0<x_1<..<x_n$) is a function S(x) with the following properties:

1. On each subinterval $[x_{i-1},x_i]$, i=1,2,..n s(x) is a polynomial of degree less than equal to d.
2. S(x) and its first (d-1) derivatives are continuous on $[x_0,x_n]$.
3. The interpolation conditions $s(x_i)=f(x_i)$, i=0,1,...n are satisfied.

We will look at linear, quadratic and cubic spline (splines or degree upto 3)

Let us assume we are constructing the Spline in such a way that,

$$S(x) = \begin{cases} P_1(x) & x_0 \leq x \leq x_1 \\ \vdots & \vdots \\ P_i(x) & x_{i-1} \leq x \leq x_i \\ \vdots & \vdots \\ P_n(x) & x_{n-1} \leq x \leq x_n \end{cases}$$

Construction of Linear Spline

For each interval $[x_{i-1},x_i]$, we already know two points, which are sufficient to find a linear polynomial between those two points.

Let $P_i(x)=a_i x+b_i$

Now, $P_i(x)$ has to pass through $(x_i\ y_i)$ and $(x_{i-1}\ y_{i-1})$

$$P_i(x_i) = a_i x_i + b_i = f(x_i) = y_i, \qquad P_i(x_{i-1}) = a_i x_{i-1} + b_i = f(x_{i-1}) = y_{i-1}$$

Solving the above two equations, we get

$$P_i(x) = a_i x + b_i = \frac{y_i - y_{i-1}}{h_i} x + \frac{x_i y_{i-1} - x_{i-1} y_i}{h_i}$$
$$= \frac{x - x_{i-1}}{h_i} y_i + \frac{x_i - x}{h_i} y_{i-1}, \qquad (h_i = x_i - x_{i-1})$$

As we can see, $P_i(x_i)=P_{i+1}(x_i)=y_i$, the linear spline is continuous at x_i, but in general

$$P'_i(x_i) = \frac{y_i - y_{i-1}}{h_i} \neq P'_{i+1}(x_i) = \frac{y_{i+1} - y_i}{h_i}$$

Hence, the Spline is not smooth.

Construction of Quadratic Spline

$$Q_i(x) = a_i x^2 + b_i x + c_i$$

It has three parameters a_i , b_i and c_i.

We can get two equations by putting the end points x_{i-1} and x_i in the equation,

Hence, we get:

$$Q_i(x_i) = y_i, \quad Q_i(x_{i-1}) = y_{i-1}$$

Also, as per the property of splines, the first derivative must also be continuous, hence,

$$Q'_i(x_i) = Q'_{i+1}(x_i)$$

To obtain the three parameters a_i , b_i and c_i, we consider

$$Q_i'(x) = 2a_i x + b_i$$

Which as a linear function can be fit using two end points x_{i-1} and x_i , let

$$f'(x_{i-1}) = D_{i-1} \text{ and } f'(x_i) = D_i$$

Hence,

$$Q_i'(x) = \frac{x - x_{i-1}}{h_i} D_i + \frac{x_i - x}{h_i} D_{i-1}$$

Integrating, we get

$$Q_i(x) = \int Q_i'(x)\,dx = \frac{D_i}{2h_i}(x - x_{i-1})^2 - \frac{D_{i-1}}{2h_i}(x_i - x)^2 + c_i$$

As $Q_i(x_i)=y_i$, we have

$$Q_i(x_i) = \frac{D_i}{2h_i}(x_i - x_{i-1})^2 + c_i = \frac{D_i h_i}{2} + c_i = y_i$$

Solving this for c_i, we get

$$c_i = y_i - \frac{D_i h_i}{2}$$

Hence,

$$Q_i(x) = \frac{D_i}{2h_i}(x - x_{i-1})^2 - \frac{D_{i-1}}{2h_i}(x_i - x)^2 + y_i - \frac{D_i h_i}{2}$$

Also as $Q_i(x_{i-1})=y_{i-1}$, we have

$$Q_i(x_{i-1}) = -\frac{D_{i-1} h_i}{2} + y_i - \frac{D_i h_i}{2} = y_{i-1}$$

Finally, we get

$$D_i = 2\frac{y_i - y_{i-1}}{h_i} - D_{i-1}, \qquad (i = 1, \cdots, n)$$

Given $f'(x_0)=D_0$, we can find all D_i ($i=1,2,\ldots n$) iteratively, and thereby $Q_i(x)$.

Alternatively, given $f'(x_n)=D_n$, we can find all D_i ($i=n-1,n-2,\ldots$) iteratively, and thereby $Q_i(x)$. The spline interpolation will be continuous and differentiable at all the points.

Construction of Cubic spline

Let us denote M_1, M_2 ..... M_n,

$$M_i = s''(x_i), \quad i = 0, 1, \cdots, n$$

Since $s_i(x)$ is cubic on each $[x_{i-1}, x_i]$, the function s''(x) is linear on the interval such that

$$s''(x_{i-1}) = M_{i-1}, \quad s''(x_i) = M_i.$$

Therefore,

$$s''(x) = \frac{(x_i - x)M_{i-1} + (x - x_{i-1})M_i}{x_i - x_{i-1}}$$

Integrating the above equation twice with respect to x, we get

$$s(x) = \frac{(x_i - x)^3 M_{i-1}}{6(x_i - x_{i-1})} + \frac{(x - x_{i-1})^3 M_i}{6(x_i - x_{i-1})} + K_1 x + K_2$$

Where K_1 and K_2 are integrating constants determined by using interpolating conditions $s(x_{i-1})=f(x_{i-1})$ and $s(x_i)=f(x_i)$

Hence,

$$K_1 = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} - \frac{(M_i - M_{i-1})(x_i - x_{i-1})}{6}$$

$$K_2 = \frac{x_i f(x_{i-1}) - x_{i-1} f(x_i)}{x_i - x_{i-1}} - \frac{(M_{i-1} x_i - M_i x_{i-1})(x_i - x_{i-1})}{6}$$

Substituting these values back into s(x),

$$s(x) = \frac{(x_i - x)^3 M_{i-1} + (x - x_{i-1})^3 M_i}{6(x_i - x_{i-1})} + \frac{(x_i - x) f(x_{i-1}) + (x - x_{i-1}) f(x_i)}{x_i - x_{i-1}}$$

$$- \frac{1}{6}(x_i - x_{i-1})[(x_i - x)M_{i-1} + (x - x_{i-1})M_i],$$

This formula applies to each of the intervals $[x_1,x_2],[x_2,x_3],\ldots\ldots,[x_{n-1},x_n]$

For adjacent intervals $[x_{i-1},x_i]$ and $[x_i,x_{i+1}]$ we will agree at their common point x=xi because of interpolating condition $s(x_i)=f(x_i)$. This implies that s(x) is continuous over the entire [a,b]. Also, s''(x) is also continuous over [a,b].

To find the values of M_i, we will ensure continuity of s'(x) over [a,b].
Hence we get the equation:

$$\frac{x_i - x_{i-1}}{6} M_{i-1} + \frac{x_{i+1} - x_{i-1}}{3} M_i + \frac{x_{i+1} - x_i}{6} M_{i+1} = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} - \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}.$$

These n-1 equations along with the assumption that $M_0=M_N=0$, leads to values of all $M_1, M_2, \ldots M_{N-1}$. (we can solve using gauss elimination method)
Such a spline is called natural cubic spline (due to the choice of M_0 and M_N).

Convergence Criteria

As we can find a spline for all possible given pair points on the real axis, hence no “convergence criteria” is defined for spline interpolation method.

Error order

The error bounds (for cubic splines) are of the form $\| f^{(r)} - s^{(r)} \|_{\infty} \leqslant C_r \| f^{(4)} \|_{\infty} h^{4-r}$, where *s* is a cubic spline interpolant of $f \in C^4[a, b]$, h is the max step distance between the points, and C_r is a constant.

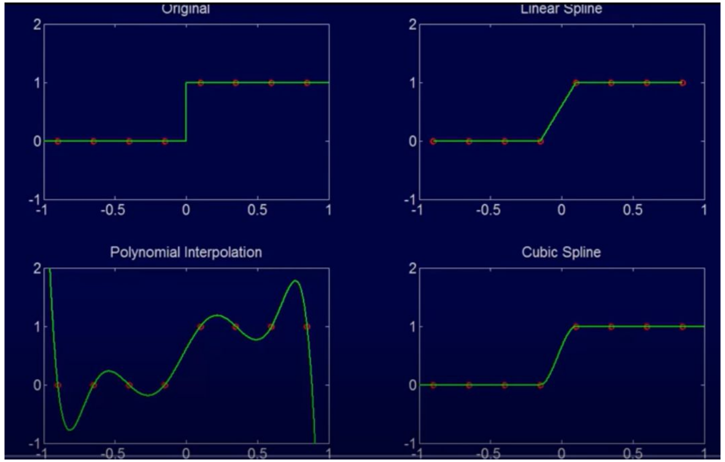
Advantages and Disadvantages

Comparison with normal piecewise interpolation

Comparing with **normal piecewise interpolation**, we have to take 3 consecutive points and do Lagrange interpolation/ Newton's interpolation and create such functions for all sets of three points for the beginning.

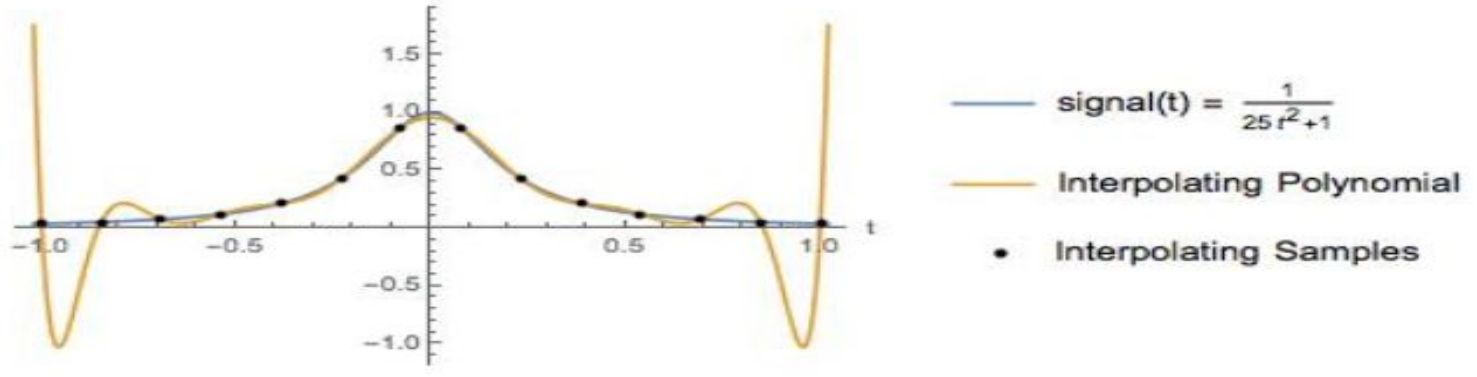
The main **advantage** of spline interpolation over normal piecewise interpolation is that the piecewise interpolant won't always be differentiable at the end points, but the spline interpolant will be,also the method to calculate the spline interpolant can be easily implemented on a computer.

Also, in some cases, due to the piecewise nature of spline interpolation, it fits the set of points much better than all the other methods, as visible in the example given below:



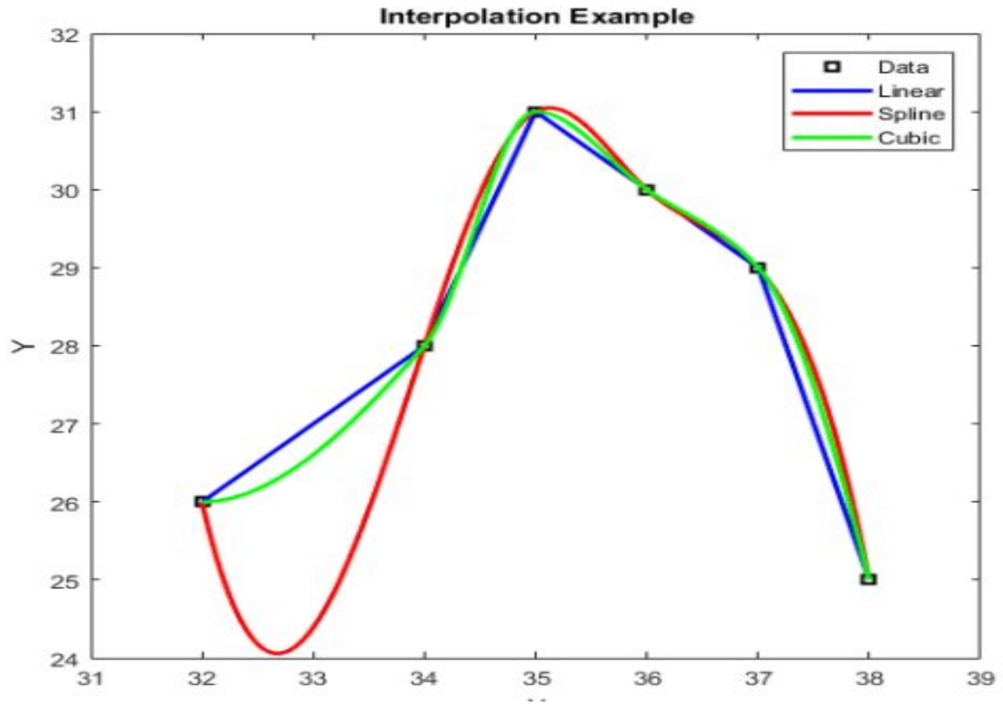
As we can see, the cubic spline is a much better fit, while the polynomial interpolation is not a good fit at all.

Spline interpolation is also time and memory efficient, compared to many interpolation methods.
In the following example,



We can see that there is much distortion in the end points so that the interpolating polynomial fits the given set of points, but if we use cubic spline interpolation, it will give a much better fit,even at the end points with much less distortion.

One more such example,



Computational Costs

The main computational cost in computing the spline interpolation is because of solving to find the values of M_i. Hence, we have to solve N-1 equations, with N-1 unknown variables (M_0=M_n=0 for natural spline). If we use Gauss elimination method as a subroutine, the complexity will be $O((n-1)^3/3)$, approximately $O(n^3)$.

Hence, to find the $s_i(x)$, we need to put values of M_i, taking O(4*n) time (4 places values to be put).

Hence the total time complexity is **$O(n^3+n)$ or nearly $O(n^3)$**.

Real life model Examples

Interpolation, and in turn spline interpolation is used in many real world applications.

In many cases, we can't store the complete information about the function, and hence some set of points are provided, and while reverting, we can use spline interpolation to get an approximate function.

For example, in **statistics,** we can perform a hypothesis on a small random sample, and hence get input output pairs based on it, and hence we can guess the output for a previously unknown value by creating an interpolating function using the given set of points, and outputting the value of function at that point.

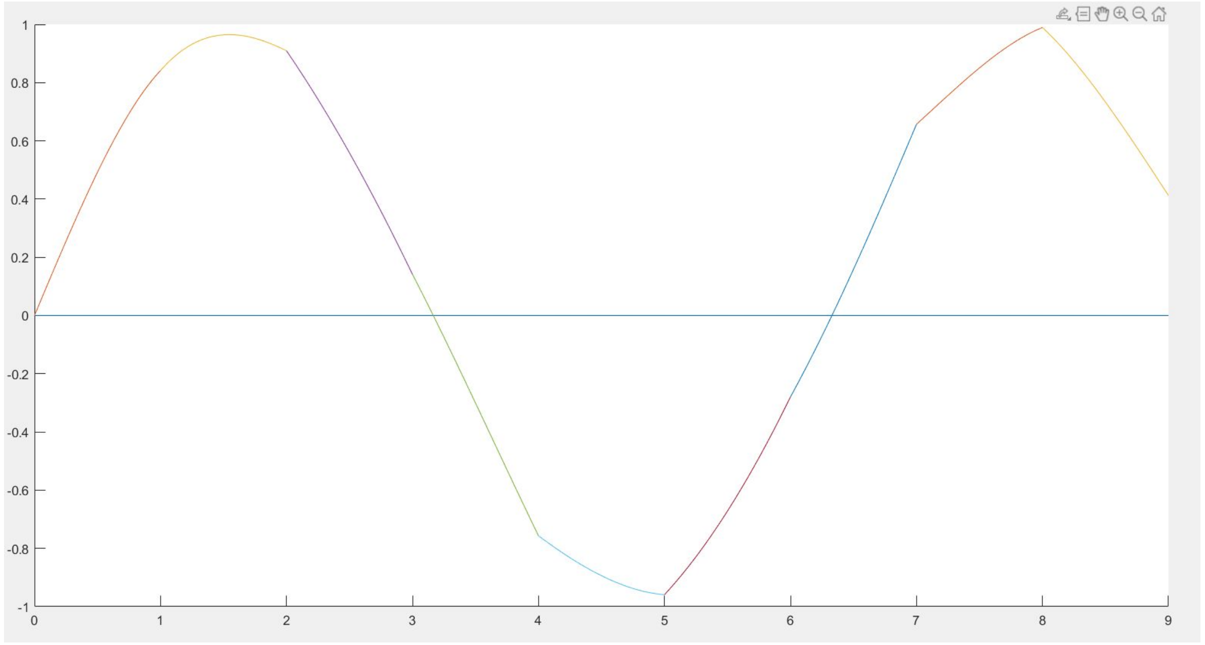
In **electrical**, given discrete pairs of input and output, we can use the interpolating polynomial to guess the values and do analysis on the function, as the spline interpolation will give a very good estimate of the original function.

It is also widely used in many different applications like **computer graphics**, **animations**, **robotics** and **finance**. Many of these applications are run in real-time with constraints on computational complexity, thus fueling the need for computational inexpensive, real-time, continuous and loop-free data interpolation techniques.

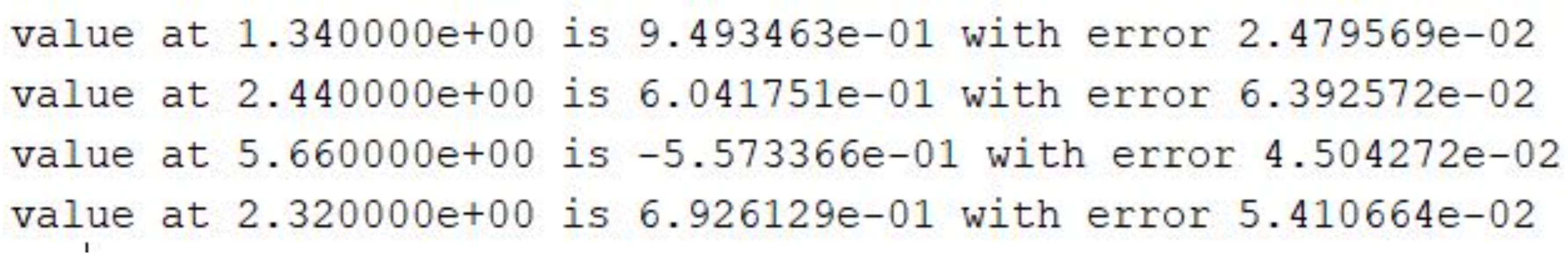
Some examples using code

Sin(x)

The spline interpolation looks like this:



Error in the interpolation

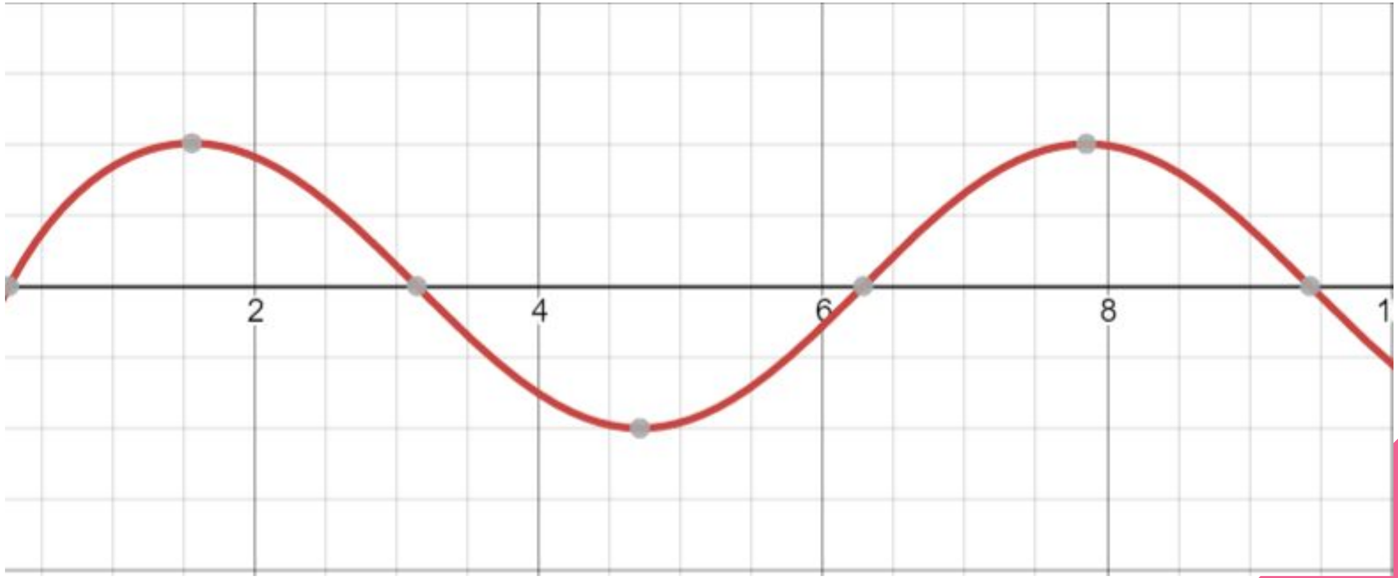


```
value at 1.340000e+00 is 9.493463e-01 with error 2.479569e-02
value at 2.440000e+00 is 6.041751e-01 with error 6.392572e-02
value at 5.660000e+00 is -5.573366e-01 with error 4.504272e-02
value at 2.320000e+00 is 6.926129e-01 with error 5.410664e-02
```

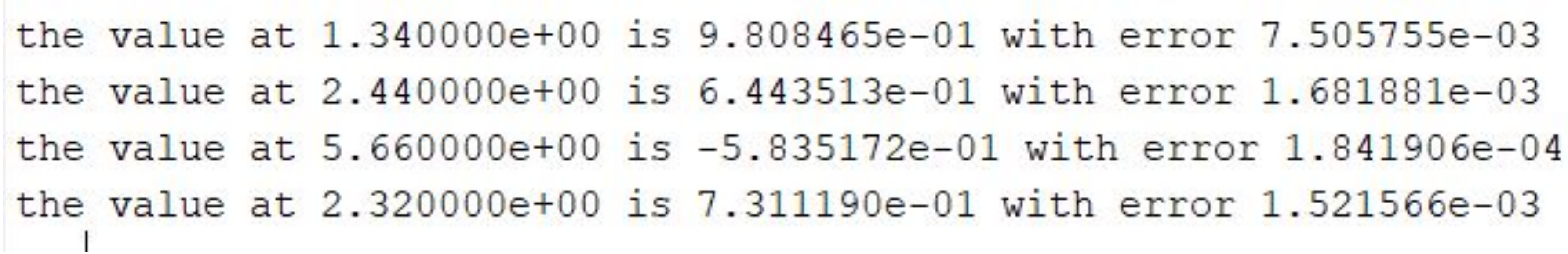
As we can see, the relative error is around 2%, which is very less (as i took 10 points from 0 to 9).

Comparison with Gregory-Newton polynomial interpolation

We get the following graph using Gregory newton interpolation:-



Error in the approximation

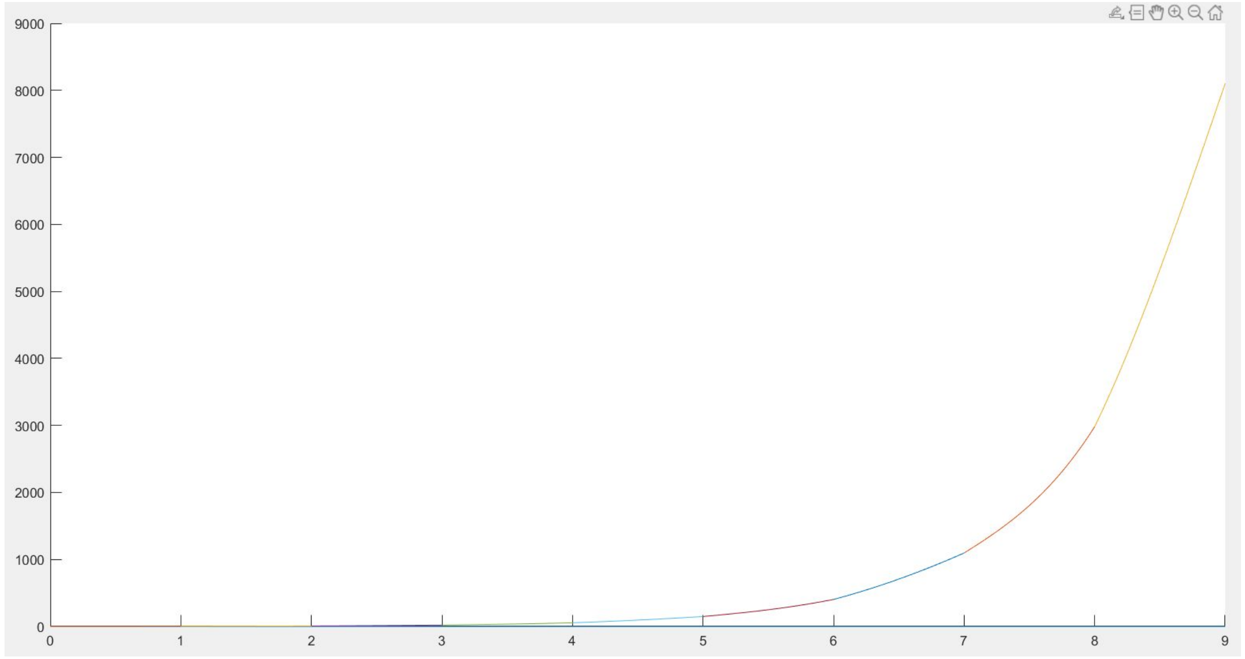


```
the value at 1.340000e+00 is 9.808465e-01 with error 7.505755e-03
the value at 2.440000e+00 is 6.443513e-01 with error 1.681881e-03
the value at 5.660000e+00 is -5.835172e-01 with error 1.841906e-04
the value at 2.320000e+00 is 7.311190e-01 with error 1.521566e-03
```

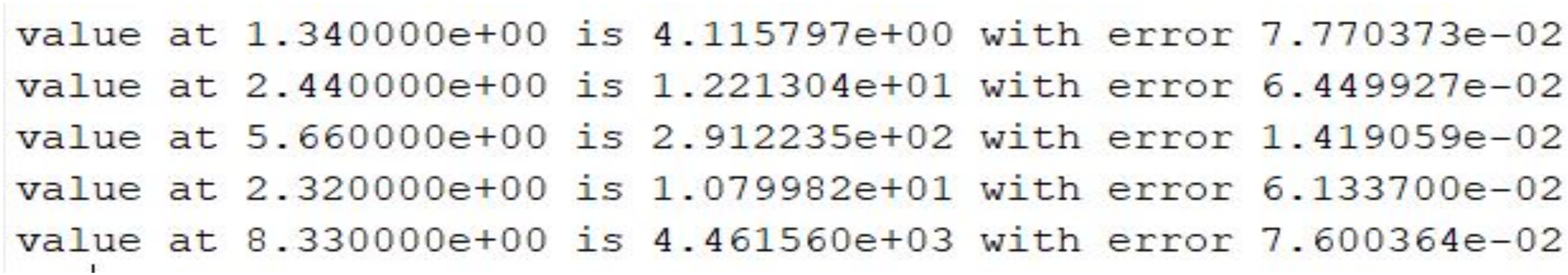
As we can see, the error is even less than spline interpolation, and here the error is 0.7%, which is because it is a smooth continuous function, with less distortion and hence the interpolating polynomial will give a much better fit.

e^x

The spline interpolation looks like this:



Error in the interpolation

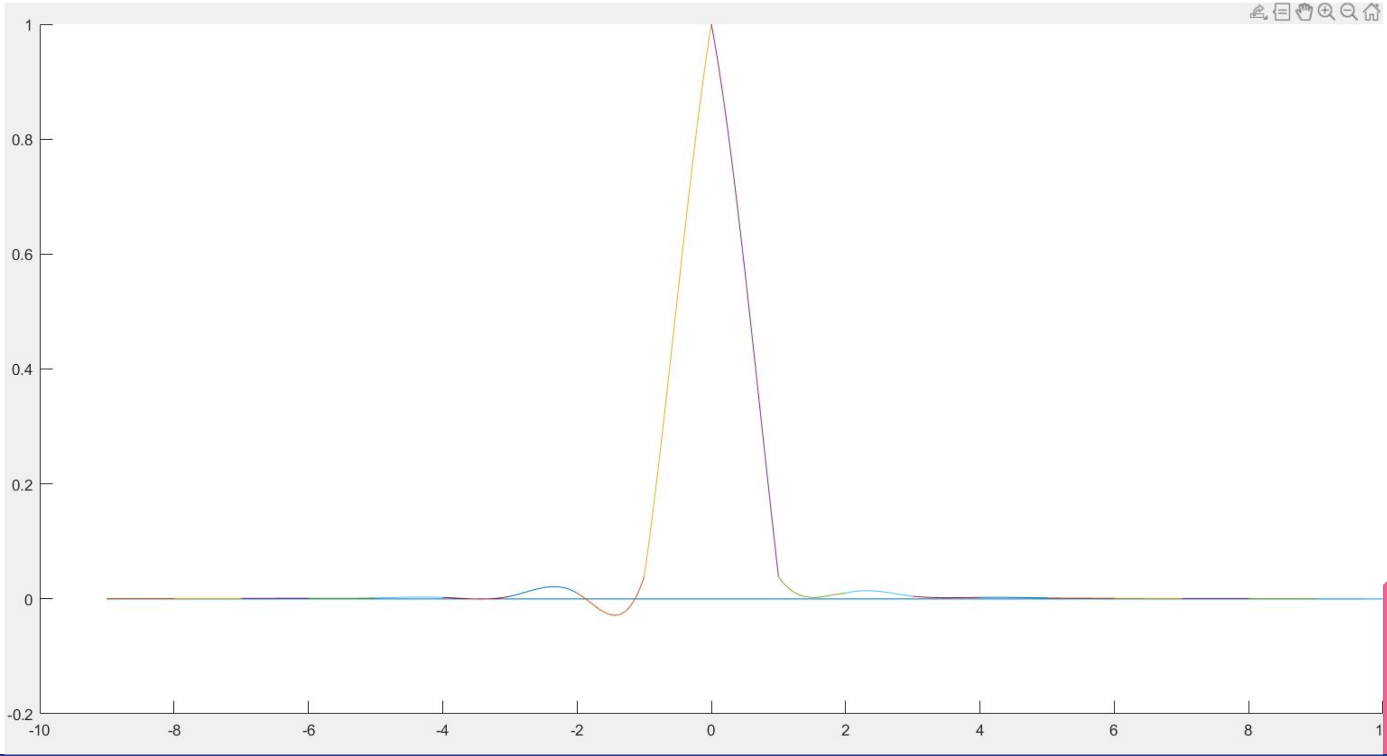


```
value at 1.340000e+00 is 4.115797e+00 with error 7.770373e-02
value at 2.440000e+00 is 1.221304e+01 with error 6.449927e-02
value at 5.660000e+00 is 2.912235e+02 with error 1.419059e-02
value at 2.320000e+00 is 1.079982e+01 with error 6.133700e-02
value at 8.330000e+00 is 4.461560e+03 with error 7.600364e-02
```

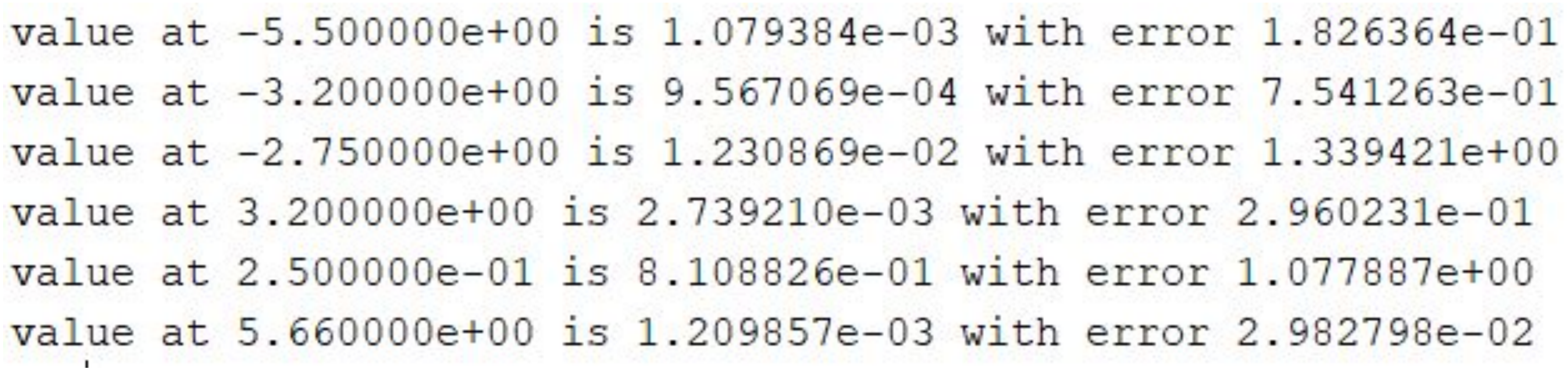
As we can see, the relative error is around 6%, which is very less (as I took only 10 points from 0 to 9).

y=1/(25*x^2+1)

The interpolant looks like this:



Error

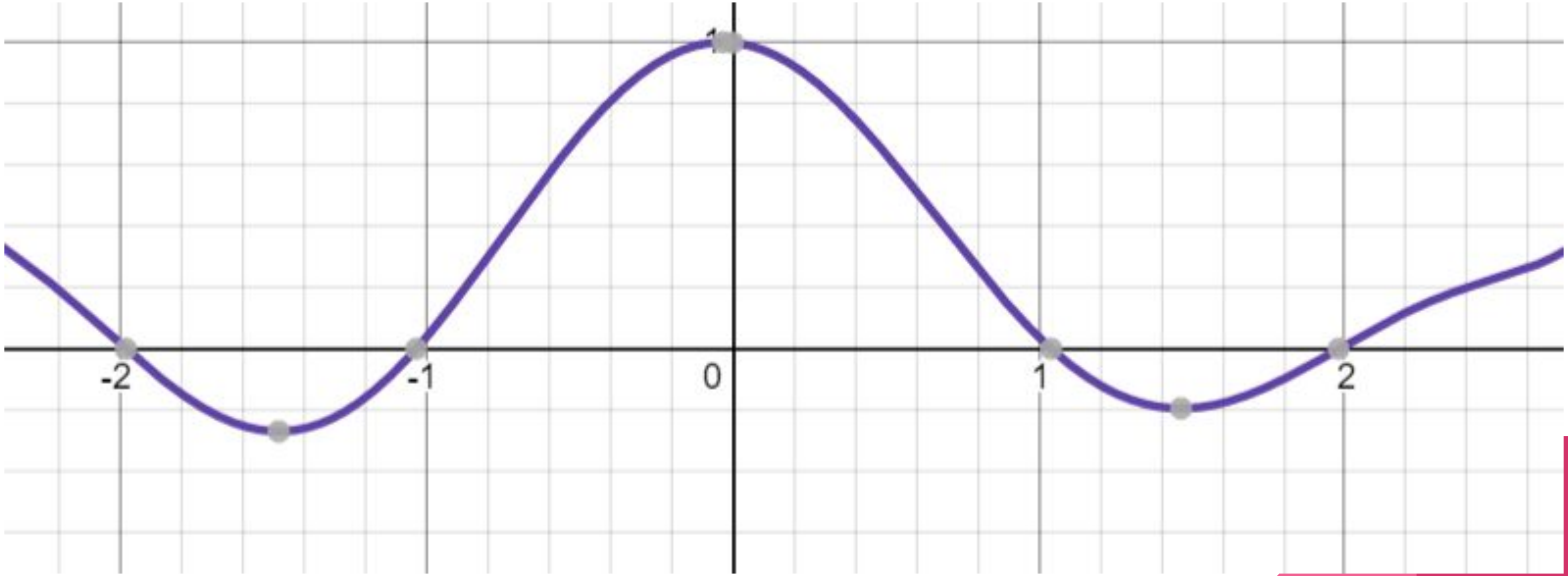


```
value at -5.500000e+00 is 1.079384e-03 with error 1.826364e-01
value at -3.200000e+00 is 9.567069e-04 with error 7.541263e-01
value at -2.750000e+00 is 1.230869e-02 with error 1.339421e+00
value at 3.200000e+00 is 2.739210e-03 with error 2.960231e-01
value at 2.500000e-01 is 8.108826e-01 with error 1.077887e+00
value at 5.660000e+00 is 1.209857e-03 with error 2.982798e-02
```

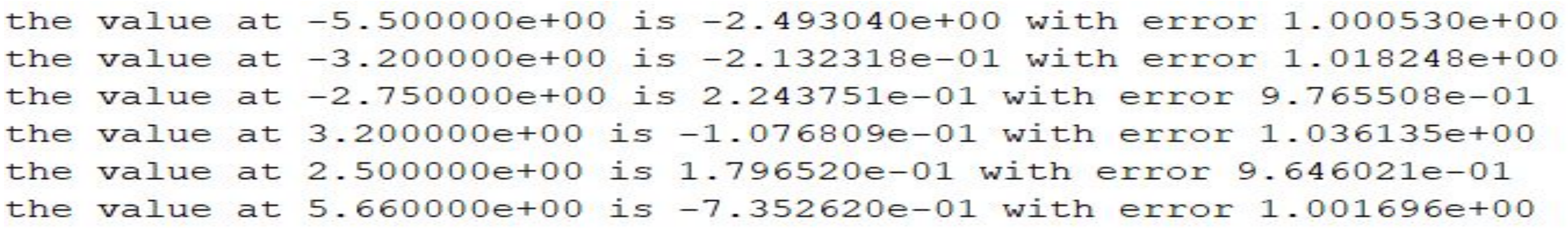
As we can see, the relative error is around 26-70%, which is very less (as i took 20 points from -9 to 10).

Using Gregory newton interpolation

We get the following graph:



As we can see, there are much distortions, even the value goes negative for many x’s, which is not possible.



```
the value at -5.500000e+00 is -2.493040e+00 with error 1.000530e+00
the value at -3.200000e+00 is -2.132318e-01 with error 1.018248e+00
the value at -2.750000e+00 is 2.243751e-01 with error 9.765508e-01
the value at 3.200000e+00 is -1.076809e-01 with error 1.036135e+00
the value at 2.500000e+00 is 1.796520e-01 with error 9.646021e-01
the value at 5.660000e+00 is -7.352620e-01 with error 1.001696e+00
```

As we can see, the error is consistently around 100%, which is a lot as compared to the spline interpolation, where the error was around 26-70%.

Hence in such a case, where there is a lot of distortions in the graph, the spline interpolation works much better.

Verdict

Hence, as we can see, in the functions where the normal interpolating polynomial must distort a lot to fit all the points, has a lot of error, like in the function 1/(25*x^2+1).

But in some smooth functions with less distortion, the normal interpolating polynomial fits better sometimes.

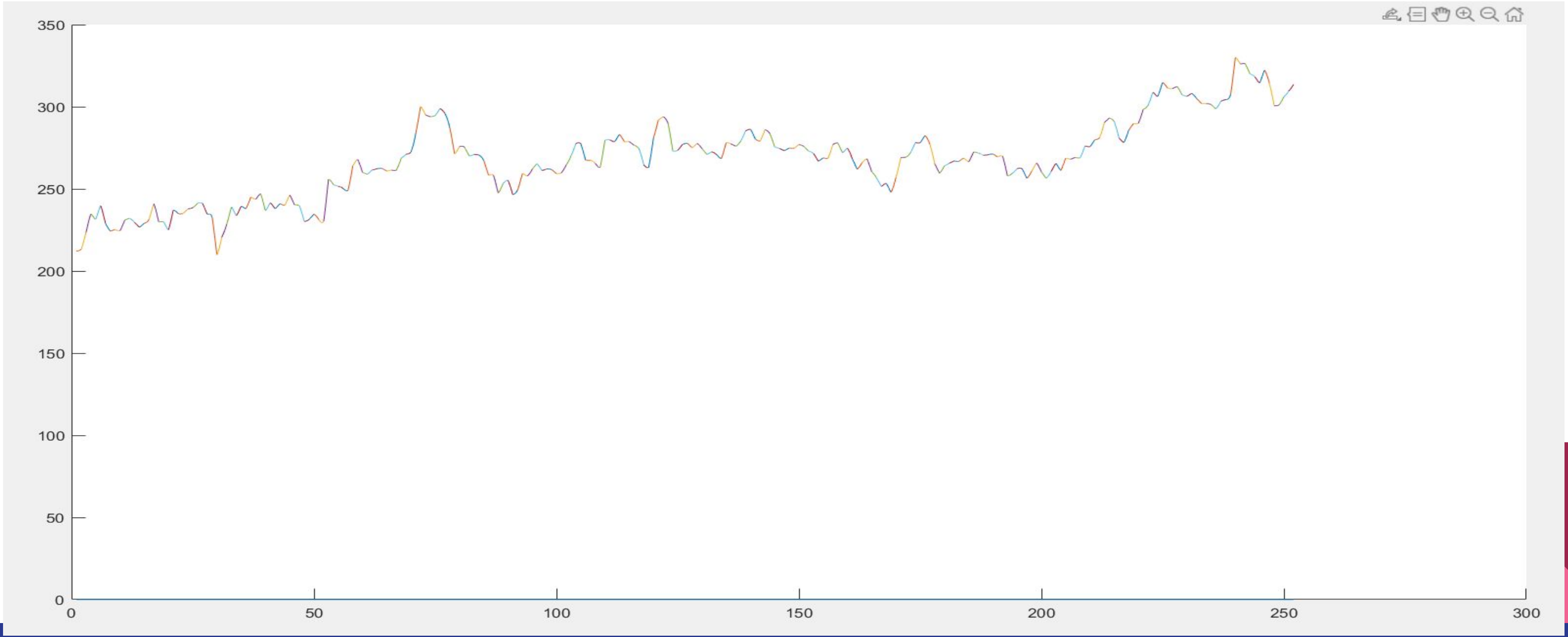
Examples using real life data

Introduction

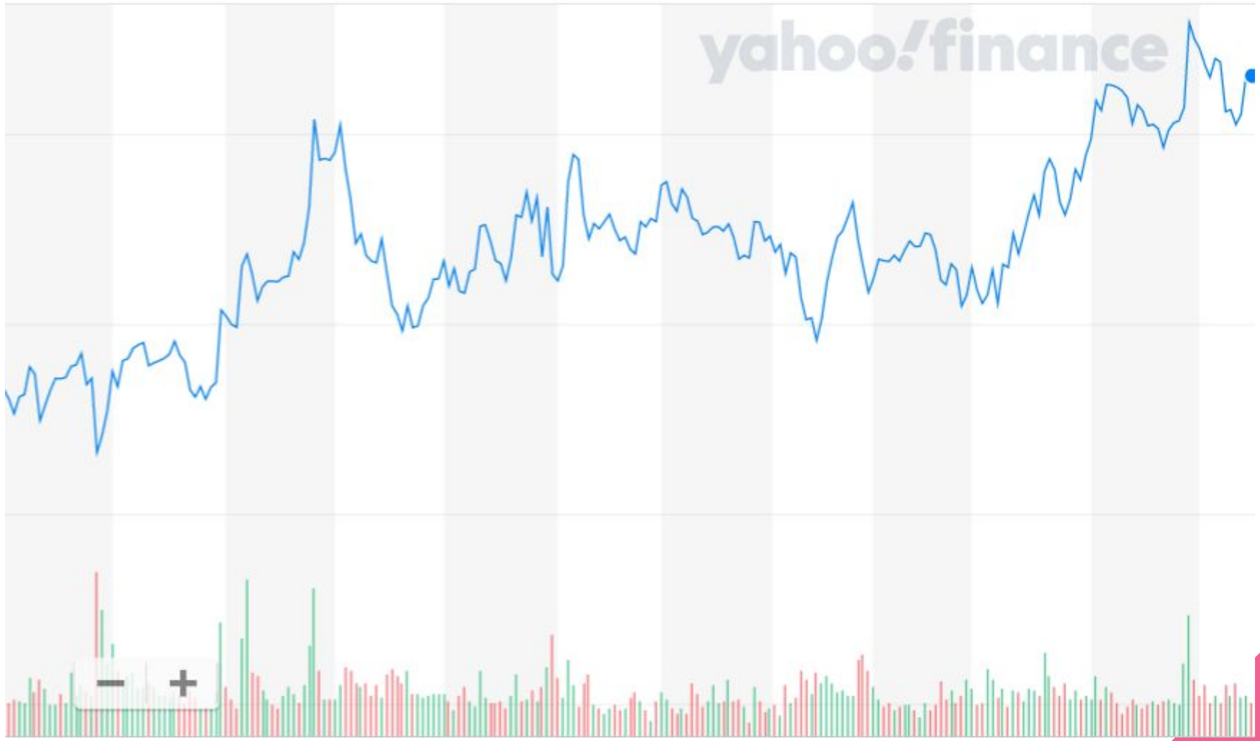
I have done the interpolation of two different stocks opening costs, taking data of previous one year (365 days).

I have taken FB (facebook) and BTC-INR (Bitcoin), and compared the graph we got from the code with yahoo finance graph, to see if our interpolating polynomial gives a good fit or not.

Plotting FB (facebook) opening cost against date from 18/5/20 to 17/05/21

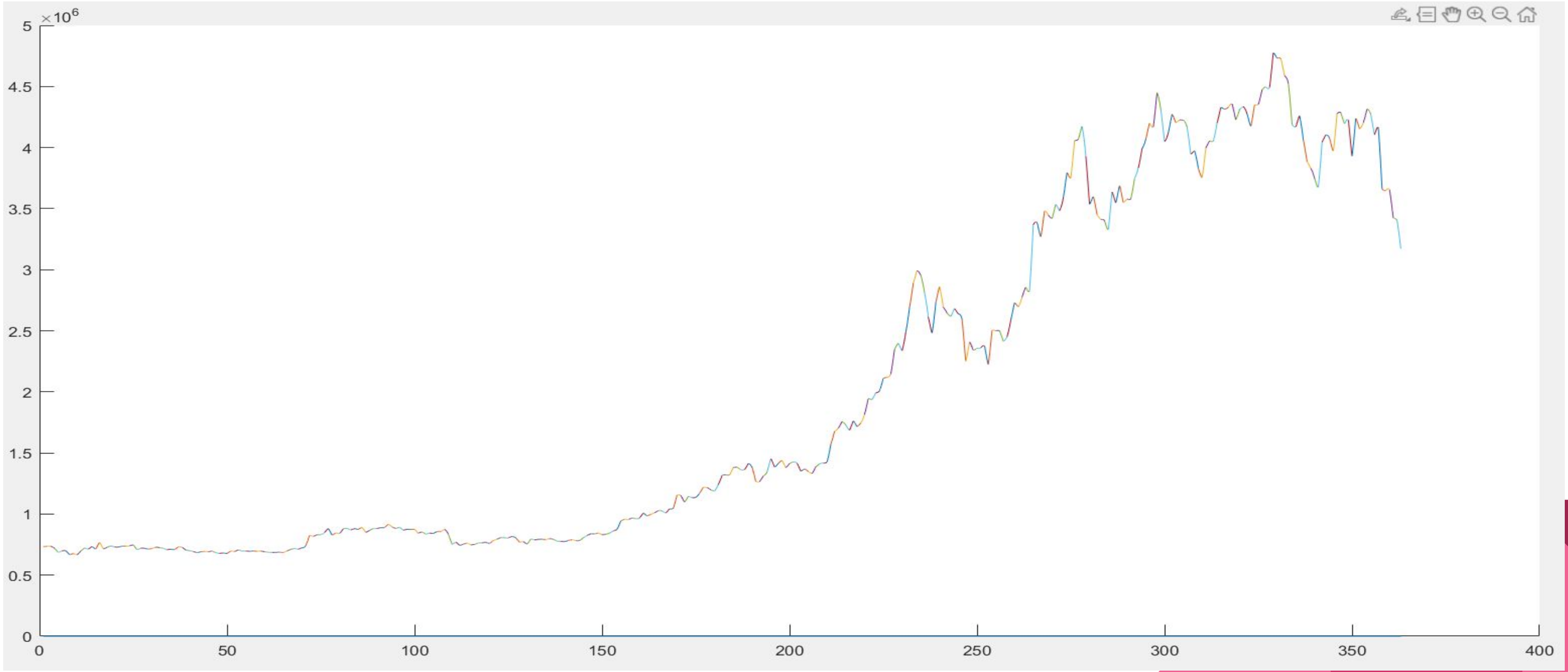


The yahoo finance graph, which also uses some kind of interpolation, gives the following graph:

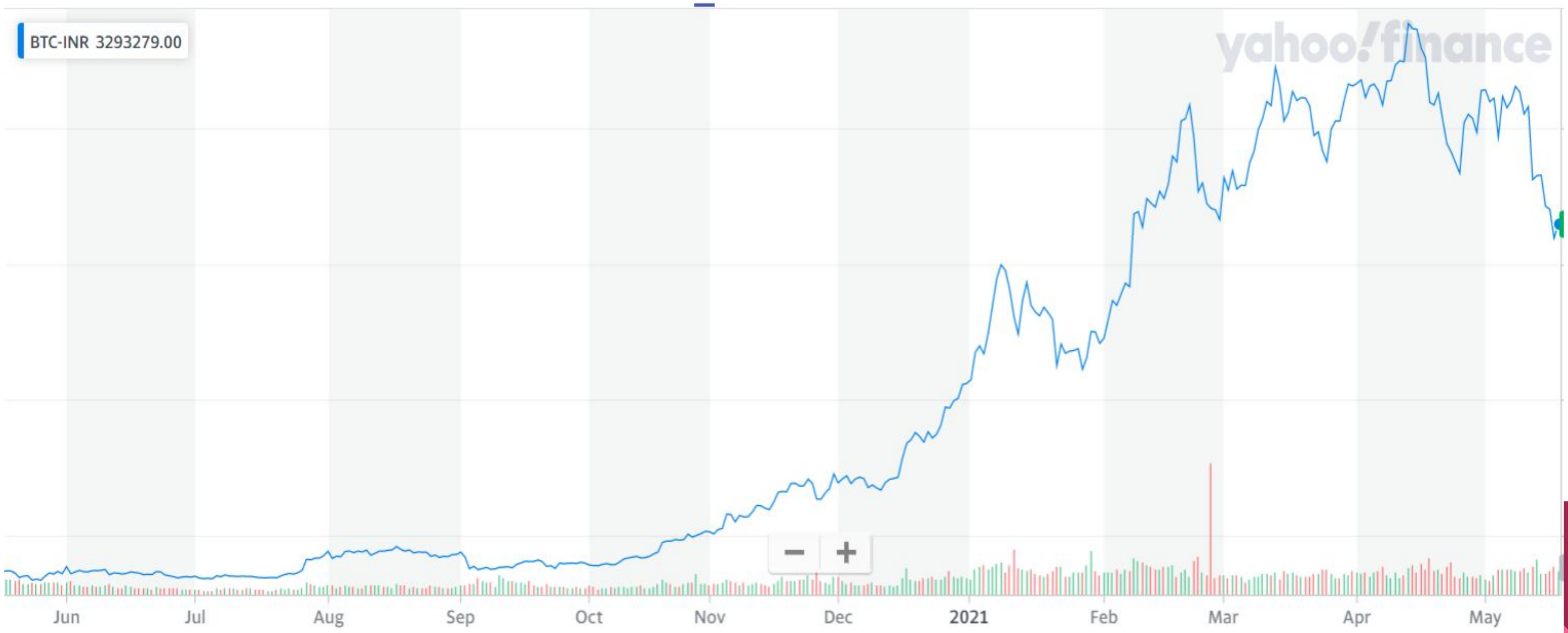


We can see, the our graph using spline interpolation is much closer the the official yahoo finance graph, which means that spline interpolation gives a very good approximation to such functions.

Plotting BTC-INR (Bitcoin) opening cost against date from 18/5/20 to 18/05/21



The yahoo finance graph, which also uses some kind of interpolation, gives the following graph:



As we can see, the interpolation gives a good fit to the function.

Hence, this is why spline interpolation is used in many different areas such as finance, computer graphics and all, as it gives a good and easy function to work with, and we can extrapolate values at many points with ease.

The spline interpolation is also very fast and memory efficient as compared to other interpolation methods, as it only takes O(n^3) time, with very little constant factors.

Thank You!