Understanding the Cubic Spline Interpolation

Zexuan Sun and Huqing Yang

October 7, 2019

Contents

1 Introduction

Cubic spline interpolation is a special case for Spline interpolation that is used very often to avoid the problem of Runge's phenomenon. This method gives an interpolating polynomial that is smoother and has smaller error than some other interpolating polynomials such as Lagrange polynomial and Newton polynomial.[1]

In this paper, firstly, we give solution to completely specify the spline using three different end condition. Secondly, with the help of MATLAB, we implement cubic spline interpolation for Runge's example.[1]And we explore the accuracy and stability of cubic spline interpolation for Runge's example on [-5,5].

2 Solution to specify the spline

First, we have n+1 interpolation points $a = x_0 < x_1 < \ldots < x_n = b$,and know S($x_i$)=$y_i$, i=0,1,. . . ,n. The function S(x) is a spline of degree 3 on [a,b] and

$$S(x) = \begin{cases} S_0(x) & x_0 \le x \le x_1 \\ S_1(x) & x_1 \le x \le x_2 \\ \ldots & \ldots \\ S_{n-1}(x) & x_{n-1} \le x \le x_n \end{cases}$$

We introduce $h_i = x_{i+1} - x_i$, i=0,1,. . . ,n.
Let:

$$m_i = S_i'(x_i)$$

[1]In this paper, we consider the function $\frac{25}{1+x^2}$ as Runge's example.

where m_i is a parameter and i=0,1,. . . ,n.

Then let the first derivative of the ith piece of the spline be represented by:

$$S_i'(x) = m_i + a * \frac{x - x_i}{h_i} + b * (\frac{x - x_i}{h_i})^2$$

We now integrate once to get the representation of the ith piece of the spline:

$$S_i(x) = y_i + m_i * (x - x_i) + \frac{1}{2}h_i * a * (\frac{x - x_i}{h_i})^2 + \frac{1}{3}h_i * b * (\frac{x - x_i}{h_i})^3$$

where a and b are parameters.

Taking the value of $S_i(x_{i+1})$ and the derivative of $S_i(x_{i+1})$.

$$S_i(x_{i+1}) = y_i + m_i * h_i + \frac{1}{2} * h_i * a + \frac{1}{3} * h_i * b = y_{i+1} \tag{1}$$

$$S_i'(x_{i+1}) = m_i + a + b = m_{i+1} \tag{2}$$

Solving (1)-(2) for a and b, then gives

$$a = 6 * \frac{y_{i+1} - y_i}{h_i} - 4m_i - 2m_{i+1}$$

$$b = -6 * \frac{y_{i+1} - y_i}{h_i} + 3m_i + 3m_{i+1}$$

We know $S_i''(x) = \frac{a}{h_i} + \frac{2b}{h_i}\frac{x - x_i}{h_i}$.Then taking a and b we can get the representation of the second derivatives of the ith piece of the spline:

$$S_i''(x) = \frac{6(y_{i+1} - y_i)}{{h_i}^3} * (x_{i+1} + x_i - 2x) + \frac{m_i}{{h_i}^2} * (6x - 2x_i - 4x_{i+1}) + \frac{m_{i+1}}{{h_i}^2} * (6x - 4x_i - 2x_{i+1})$$

Then we can get the value of $S_i''(x_i + 0)$ and $S_{i-1}''(x_i - 0)$, they are:

$$S_i''(x_i + 0) = \frac{6(y_{i+1} - y_i)}{{h_i}^2} - \frac{4m_i}{h_i} - \frac{2m_{i+1}}{h_i} \tag{3}$$

$$S_i''(x_i + 0) = \frac{6(y_{i+1} - y_i)}{{h_i}^2} - \frac{4m_i}{h_i} - \frac{2m_{i+1}}{h_i} \tag{4}$$

Let (3) = (4) we have:

$$\frac{h_i}{h_{i-1} + h_i} * m_{i-1} + 2m_i + \frac{h_{i-1}}{h_{i-1} + h_i} * m_{i+1} = 3(\frac{h_i}{h_{i-1} + h_i} * \frac{y_i - y_{i-1}}{h_{i-1}} + \frac{h_{i-1}}{h_{i-1} + h_i} * \frac{y_{i+1} + y_i}{h_i})$$

Introduce

$$\lambda_i = \frac{h_i}{h_{i-1} + h_i}$$

$$\mu_i = \frac{h_{i-1}}{h_{i-1} + h_i} = 1 - \lambda_i$$

$$g_i = 3(\lambda_i * \frac{y_i - y_{i-1}}{h_{i-1}} + \mu_i * \frac{y_{i+1} + y_i}{h_i})$$

And we get:

$$\lambda_i * m_{i-1} + 2m_i + \mu_i * m_{i+1} = g_i,\ i = 1, 2, 3, ..., n - 1 \tag{5}$$

Totally we have n+1 unknowns: m_i, i = 0,1,. . . ,n and there are n-1 equations in (5). So we need the other two equations to solve m_i uniquely.

Here are three different conditions.
Prescribing Slopes:

$$S_0'(x_0) = m_0,\ S_{n-1}'(x_n) = m_n \tag{6.1}$$

Prescribing Second Derivatives:

$$S_0''(x_0) = M_0,\ S_{n-1}''(x_n) = M_n \tag{6.2}$$

Periodic Conditions:

$$S_0'(x_0) = S_{n-1}'(x_n),\ S_0''(x_0) = S_{n-1}''(x_n) \tag{6.3}$$

Now we can get other two equations in each of these three conditions, then we can solve m_i, i = 0,1,. . . ,n. Finally, we get the representation of the spline. We will introduce detailed algorithms about these three conditions in the following chapters.

2.1 Prescribing Slopes

From conditions:

$$S_0'(x_0) = m_0, S_{n-1}'(x_n) = m_n$$

We have equations:

$$2m_1 + \mu_1 * m_2 = g_1 - \lambda_1 * m_0$$

$$\lambda_k * m_{k-1} + 2m_k + \mu_k * m_{k+1} = g_k \quad k = 1, 2, 3, ..., n-1$$

$$\lambda_{n-1} * m_{n-2} + 2m_{n-1} = g_{n-1} - \mu_{n-1} * m_n$$

Then solve the following tridiagonal system for m_i, i = 1,2 . . . ,n-1.

$$\begin{bmatrix} 2 & \mu_1 & 0 & \cdots & 0 \\ \lambda_2 & 2 & \mu_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \lambda_{n-2} & 2 & \mu_{n-2} \\ 0 & 0 & \cdots & \lambda_{n-1} & 2 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ ... \\ m_{n-1} \end{bmatrix} = \begin{bmatrix} g_1 - \lambda_1 m_0 \\ g_2 \\ ... \\ g_{n-1} \\ g_{n-1} - \mu_{n-1} m_n \end{bmatrix}$$

2.2 Prescribing Second Derivatives

From conditions:

$$S_0''(x_0) = M_0, S_{n-1}''(x_n) = M_n$$

$$S_0''(x_0) = \frac{6(y_1 - y_0)}{{h_0}^2} - \frac{4m_0}{h_0} - \frac{2m_1}{h_0} = M_0$$

$$S_{n-1}''(x_n) = \frac{-6(y_n - y_{n-1})}{{h_{n-1}}^2} + \frac{2m_{n-1}}{2h_{n-1}} + \frac{4m_n}{h_{n-1}} = M_n$$

We have equations:

$$2m_0 + m_1 = 3\frac{y_1 - y_0}{h_0} - \frac{h_0}{2}M_0 = g_0$$

$$m_{n-1} + 2m_n = 3\frac{y_n - y_{n-1}}{h_{n-1}} - \frac{h_{n-1}}{2}M_n = g_n$$

Then solve the following tridiagonal system for m_i, i = 0,1 . . . ,n.

$$\begin{bmatrix} 2 & 1 & 0 & \cdots & 0 \\ \lambda_1 & 2 & \mu_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \lambda_{n-1} & 2 & \mu_{n-1} \\ 0 & 0 & \cdots & 1 & 2 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ ... \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} g_0 \\ g_1 \\ ... \\ g_{n-1} \\ g_n \end{bmatrix}$$

2.3 Periodic Conditions

From conditions:

$$S_0'(x_0) = S_{n-1}'(x_n),\ S_0''(x_0) = S_{n-1}''(x_n)$$

$$m_0 = m_n,\ M_0 = M_n$$

$$S_0''(x_0) = \frac{6(y_1 - y_0)}{{h_0}^2} - \frac{4m_0}{h_0} - \frac{2m_1}{h_0} \tag{7}$$

$$S_n''(x_n) = \frac{-6(y_n - y_{n-1})}{{h_{n-1}}^2} + \frac{2m_{n-1}}{2h_{n-1}} + \frac{4m_n}{h_{n-1}} \tag{8}$$

Let (7)=(8), we have equations:

$$\frac{h_{n-1}}{h_0 + h_{n_1}} m_1 + \frac{h_0}{h_0 + h_{n_1}} m_{n-1} + 2m_n = 3\frac{y_1 - y_0}{h_0}\frac{h_{n-1}}{h_0 + h_{n-1}} + 3\frac{y_n - y_{n-1}}{h_{n-1}}\frac{h_0}{h_0 + h_{n-1}}$$

$$\lambda_n = \frac{h_0}{h_0 + h_{n-1}}$$

$$\mu_n = \frac{h_{n-1}}{h_0 + h_{n-1}} = 1 - \lambda_n$$

$$g_n = 3(\mu_n \frac{y_1 - y_0}{h_0} + \lambda_n \frac{y_n - y_{n-1}}{h_{n-1}})$$

Then solve the following generalized tridiagonal system for m_i, i = 1,2 . . . ,n.

$$\begin{bmatrix} 2 & \mu_1 & 0 & \cdots & \lambda_1 \\ \lambda_1 & 2 & \mu_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \lambda_{n-1} & 2 & \mu_{n-1} \\ \mu_n & 0 & \cdots & \lambda_n & 2 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ ... \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ ... \\ g_{n-1} \\ g_n \end{bmatrix}$$

3 Numerical Error Analysis

As we have learned in class, when the supplied data points are equispaced, using Newton polynomial to interpolate Runge's example would cause Runge's phenomenon. Thus, we divide the interval—[-5,5] into equal subintervals to carry out some numerical experiments. Our work is divided into two parts. In the fist part, we compare the interpolation with Runge's function to illustrate the difference between the two under various circumstances. In the second part, we explore if the problem has regular perturbation. In other words, if a *small* change in the problem induces a *small* change in the solution.

3.1 Numerical Accuracy

To explore the accuracy of cubic spline interpolation to Runge's example, we do the following steps:

1) *Let i = 20*
2) *Generate i interpolation points from [-5,5]*
3) *Find the cubic spline interpolation for Runge's example using the MATLAB function* ***csape***
4) *Plot the Runge's example, interpolation, and the interpolation points*
5) *Let i = i + 20*

For the three end conditions discussed in section 2, we repeat the steps four times. The following code can be used in MATLAB to carry out the above steps when the end condition is prescribing slopes:

```
%clamped cubic spline interpolant

%generate data points to plot Runge's example
xx  = linspace(-5,5,150);

%slopes of Runge's example at the left and right boundary on [-5,5]
sl =  0.3698;
sr = -sl;

for i = 1:4
    %generate intepolation points
    x = linspace(-5,5,20+20*(i-1));

    %calculate the value of Runge's example at intepolation points
    y  = 25 ./ (1+x.^2);


    subplot(2,2,i);
    %plot Runge's example
    plot(xx,25 ./ (1+xx.^2),'y','LineWidth',4);

    hold on

    %find the cubic spline interpolation for Runge's example
    pp = csape(x,[sl,y,sr],'clamped');

    %plot intepolation and intepolation points
    plot(xx , fnval(pp,xx),x,y,'ro');

    %set title for each figure
    tmp_title =['Interpolant to ',num2str(20+20*(i-1)),' Points'];
    title(tmp_title);

    %set legend
    legend('Exact','Interpolated','Points');

end
```

And we get the result below:

As for the other two end conditions, we could simply modify the parameter of the function **csape**. The main change is shown below:

```
%prescribing second derivatives

%caculate the second derivatives of Runge's example
%at the left and right boundary on [-5,5]
endcond = (200*xx([1 end]).^2)./(xx([1 end]).^2 + 1).^3 ...
- 50./(xx([1 end]).^2 + 1).^2;

%find the cubic spline interpolation for Runge's example
pp = csape(x,[endcond(1) y endcond(2)],'second');



%periodic conditions

%find the cubic spline interpolation for Runge's example
pp = csape(x,y,'periodic');
```

The results of the other two end conditions are shown below:

From the figures, we could find that how amazing cubic spline interpolation is when dealing with the problem which could not be handled by Newton polynomial. It seems that as the data points increase there is not an enhancement of the accuracy of the cubic spline approximation. With just 20 points, the interpolation is capable of fitting Runge's example well if we just observe from the figures above.

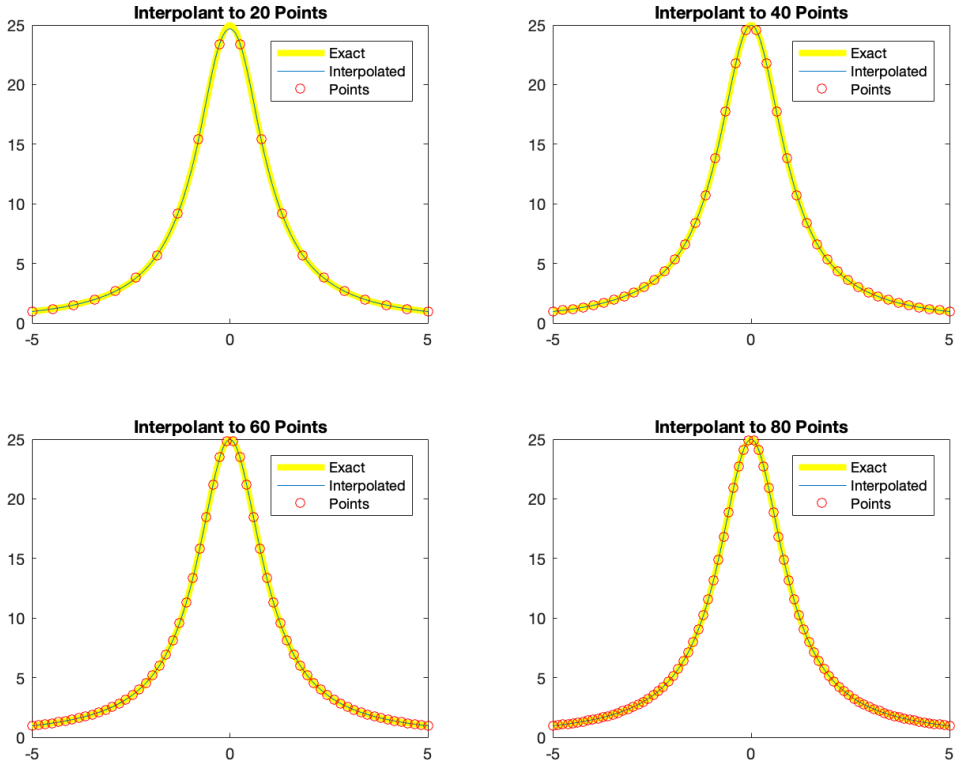

Figure 1: Clamped Cubic Spline Interpolant to $\frac{25}{1+x^2}$

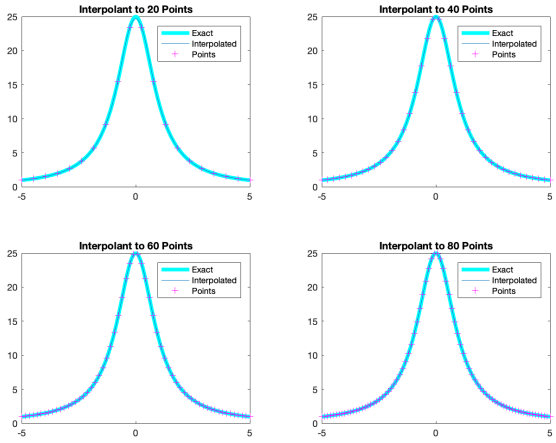

Figure 2: Spline Interpolant to $\frac{25}{1+x^2}$ When Matching the 2nd Derivative at Ends

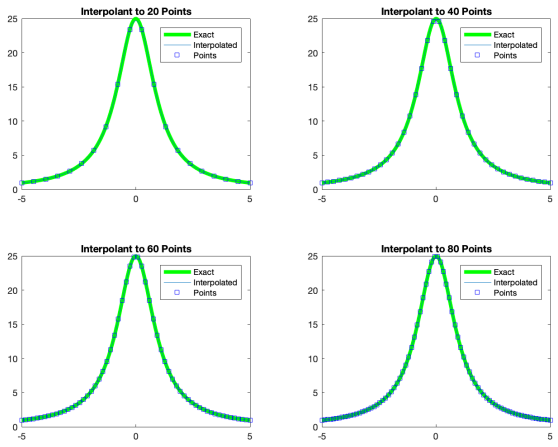

Figure 3: Periodic Cubic Spline Interpolant to $\frac{25}{1+x^2}$

As a further test, we plot the error in cubic spline interpolation to Runge's example as the data points increase by 20 each time. We do similar work to the above. Just make a little change to the previous codes we could plot the figures we want. The main change is shown below:

```
%plot the error when prescribing slopes
plot(xx , 25 ./ (1+xx.^2)- fnval(pp,xx));

%plot the error when prescribing second derivatives
plot(xx , 25 ./ (1+xx.^2)-fnval(pp,xx),'rx');

%plot the error when prescribing periodic end conditions
plot(xx , 25 ./ (1+xx.^2)-fnval(pp,xx),'m*--');
```

The result are shown as below:

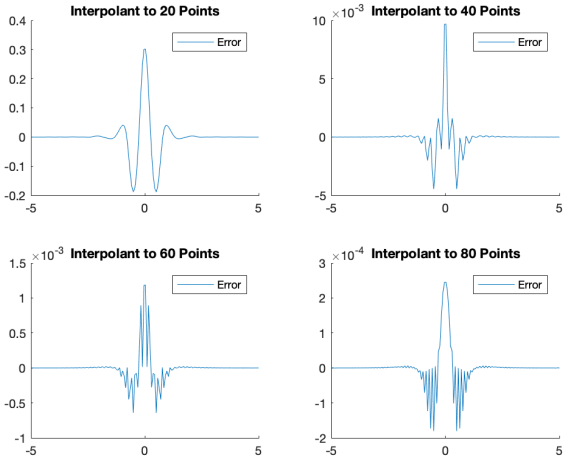

Figure 4: Error in Clamped Cubic Spline Interpolant to $\frac{25}{1+x^2}$

From the figures, we can notice that the error descends rapidly, probably by 10^{-1} order each time. Also, for all the end conditions, there is a large error at the middle of the interval. Note that for the first and

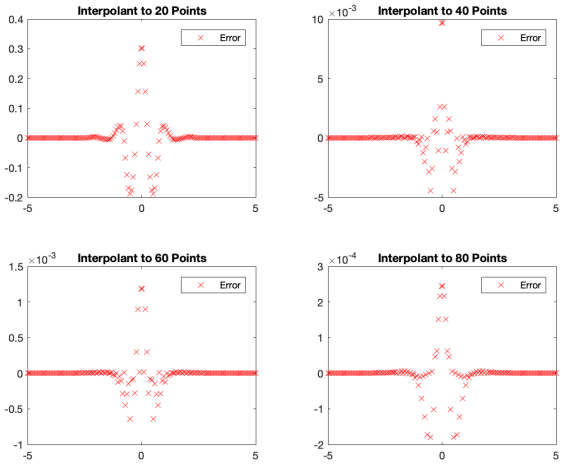

Figure 5: Error in Spline Interpolant to $\frac{25}{1+x^2}$ When Matching the 2nd Derivative at Ends

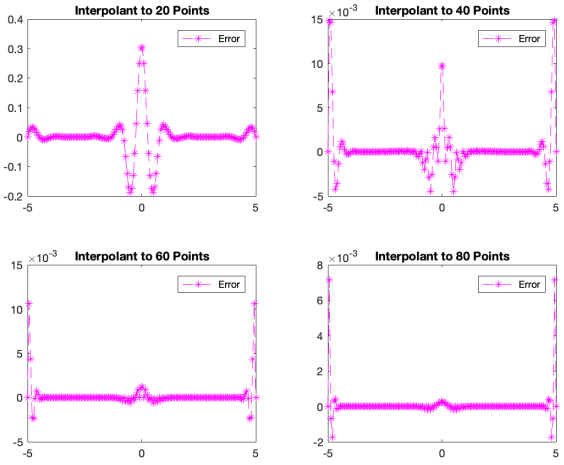

Figure 6: Error in Periodic Cubic Spline Interpolant to $\frac{25}{1+x^2}$

second end conditions, we have the maximum error at the middle of the interval. That is because the number of the interpolation points is always even. In other words, the missing of the value of Runge's example at zero because the error to be greater than the other parts of the interval. However, the error decays rapidly as we move away from the middle of the interval. This illustrates that cubic spline interpolation is essentially local.

As for periodic cubic spline interpolant, the error is largest at both left and right endpoint when we have more than 20 points to interpolate. This is due to the fact that the periodic end conditions implicitly insist on having the same derivatives and second derivatives at the left and right endpoints. Of course, we know that Runge's example itself does not match the periodic end conditions which contributes to what we see in Figure 6. The error also decays rapidly as we move away from the endpoints which also illustrate that cubic spline interpolation is essentially local.

In order to compare the accuracy of cubic spline interpolation with the first two end conditions, we use 100 data points to interpolate and then plot the errors altogether. The result is shown below:

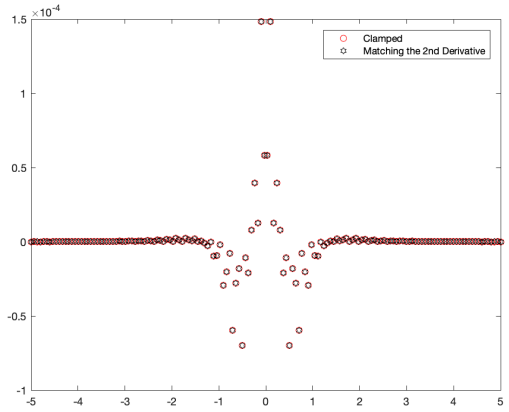

Figure 7: Error in Clamped vs. Matching the 2nd Derivative

Almost all icons overlap with each other at each point. Thus cubic spline interpolation with both the first two end conditions perform well, without any obvious difference.

3.2 Numerical Stability

To explore if the solution is stable, we introduce a disturbance to the input value of Runge's function. To be specific, we generate 10 random perturbation for each end conditions ranging from -0.01 to 0.01. We use 100 interpolation points and plot the error before and after the perturbation. We carry this out with the following MATLAB codes:

```
1
2  x = linspace(-5,5,100);
3  y  = 25 ./ (1+x.^2);
4  xx  = linspace(-5,5,150);
5
6  %use an uniform distribution function to get the random perturbation
7  pd = makedist('Uniform','lower',-0.01,'upper',0.01);
8  dis = random(pd,1,10);
9
10 %plot the error in clamped cubic spline
11 %before and after perturbation
12 subplot(3,1,1);
13
14 %find the cubic spline interpolation for Runge's example
15 %before perturbation
```

```
sl =  0.3698;
sr = -sl;
pp1 = csape(x,[sl,y,sr],'clamped');

%plot the error in clamped cubic spline before perturbation
plot(xx , 25 ./ (1+xx.^2)- fnval(pp1,xx),'ro');
hold on

%add the perturbation to the first part of the interval
%and find the cubic spline interpolation after perturbation
y(1:10) = y(1:10) + dis;
pp1_d = csape(x,[sl,y,sr],'clamped');

%plot the error in clamped cubic spline after perturbation
plot(xx , 25 ./ (1+xx.^2)- fnval(pp1_d,xx),'c*');
legend('Before','After');
title('Error in Clamped Before vs. After Perturbation');


%plot the error in cubic spline mathing the 2nd derivative
%before and after perturbation
subplot(3,1,2);

%find the cubic spline interpolation matching the 2nd
%derivative for Runge's example before perturbation
endcond = (200*xx([1 end]).^2)./(xx([1 end]).^2 + 1).^3 ...
- 50./(xx([1 end]).^2 + 1).^2;
y  = 25 ./ (1+x.^2);
pp2 = csape(x,[endcond(1) y endcond(2)],'second');

%plot the error in cubic spline matching the 2nd derivative
%before perturbation
plot(xx , 25 ./ (1+xx.^2)-fnval(pp2,xx),'b');
hold on

%add the perturbation to the middle part of the interval
%and find the cubic spline interpolation after perturbation
dis_2 =[zeros(1,40),dis,zeros(1,50)];
y = y + dis_2;
pp2_d = csape(x,[endcond(1) y endcond(2)],'second');

%plot the error in cubic spline matching the 2nd derivative
%after perturbation
plot(xx , 25 ./ (1+xx.^2)-fnval(pp2_d,xx),'m');
legend('Before','After');
title('Error in Matching the 2nd Derivative Before vs. After Perturbation');

%plot the error in  periodic cubic spline
%before and after Perturbation
subplot(3,1,3);

%find the periodic interpolation for Runge's example
%before perturbation
y  = 25 ./ (1+x.^2);
pp2 = csape(x, y ,'periodic');

%plot the error in periodic cubic spline
%before perturbation
plot(xx , 25 ./ (1+xx.^2)-fnval(pp2,xx),'kh--');
hold on

%add the perturbation to the latter part of the interval
%and find the cubic spline interpolation after perturbation
dis_3 =[zeros(1,60),dis,zeros(1,30)];
y = y + dis_3;
pp2_d = csape(x,y,'periodic');

%plot the error in periodic cubic spline
%after perturbation
plot(xx , 25 ./ (1+xx.^2)-fnval(pp2_d,xx),'bp-');
```

```
legend('Before','After');
title('Error in Periodic Before vs. After Perturbation');
```

The result is shown below:

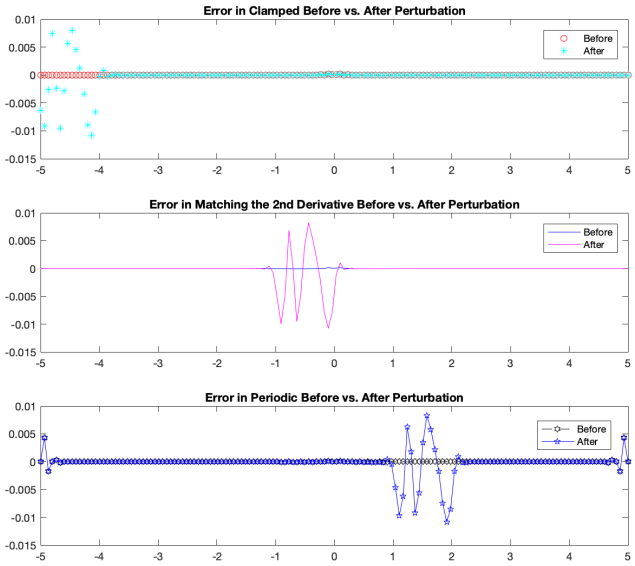

Figure 8: Errors in Interpolation with Three End conditions Before and After Perturbation

Note that we add the perturbation to different parts of the interval under three different end conditions. There is a distinct augment in the error at those certain parts of the interval. However, the error decreases dramatically as we leave that part which, again, demonstrate that cubic spline interpolation is essentially local. Still, there is a large error at the endpoints when we apply periodic end conditions to Runge's example. The reason for this is the same as we have discussed in the former subsection.

4 Conclusions

As we have learned in class, when the supplied data points are equispaced, using Newton polynomial to interpolate Runge's example would cause Runge's phenomenon. Runge's phenomenon is a problem of oscillation at the edges of an interval that occurs when using polynomial interpolation with polynomials of a high degree over a set of equispaced interpolation points. The discovery was important because it shows that going to higher degrees does not always improve accuracy.

Runge's phenomenon is the consequence of two characteristics of this problem:

- *The magnitude of the n-th order derivatives of this particular function grows quickly when n increases.*
- *The equidistance between points leads to a Lebesgue constant that increases quickly when n increases.*

The phenomenon is graphically obvious because both properties combine to increase the magnitude of the oscillations.

In fact, Ahlberg, Nilson and Walsh, p. 29 have proved that:

Theorem 1 *Let $f(x) \in C^4([a,b])$, if $S(x)$ is the cubic spline, then there exist a constant C_k such that*

$$\|S^{(k)} - f^{(k)}\|_\infty \leq C_k \|f^{(4)}\|_\infty h^{4-k}, \quad k = 0, 1, 2,$$

where $\|f\|_\infty = max\{f(x) : x \in [a,b]\}$.[2]

The theorem above shows that cubic spline interpolation converges to the given function as the mesh size goes to zero. Therefore, we can come to the conclusion that with the help of spline curves, which are piecewise polynomials, we are able to avoid Runge's phenomenon. In order to reduce the interpolation error, we can increase the number of polynomial pieces which are used to construct the spline instead of increasing the degree of the polynomials used.

The numerical experiments we have carried out show that not only there is no Runge's phenomenon when using cubic spline but also the interpolant to Runge's example is pretty accurate. Further, the solution is stable, which means a *small* change in the problem yields only a *small* perturbation of the true solution.

References

[1] `https://en.wikiversity.org/wiki/Cubic_Spline_Interpolation`.

[2] E.N. Nilson J.H. Ahlberg and J.L. Walsh. *The Theory of Splines and Their Applications*. 1967.

[2] If interested in detailed proof of this theorem, you could refer to[2].