

# Report on Chapter 3—Interpolation and Polynomial Approximation

## Abstract

In this report, we briefly introduce three algorithms about interpolation and polynomial approximation, then use these algorithms to approximate a given function.

## Contents

1	Introduction on Algorithms	1
2	Codes and Numerical Experiments	2
2.1	Newton's Interpolatory Polynomial . . . . .	2
2.2	Piecewise Hermite Interpolatory Polynomial . . . . .	4
2.3	Natural Cubic Spline Interpolatory Polynomial . . . . .	7
3	Discussions and Conclusions	10
3.1	More Experiments . . . . .	10
3.2	Conclusions . . . . .	10
A	Code 1	12
B	Code 2	12
C	Code 3	13
D	Code 4	13
E	Code 5	14

## 1 Introduction on Algorithms

We will introduce three methods of interpolation here: Newton's interpolatory polynomial, piecewise Hermite interpolatory polynomial and natural cubic spline interpolatory polynomial.

Newton's method is a way to calculate the Lagrange polynomial for  $n$  nodes. It can be used to calculate the Lagrange polynomial for the first  $k$  nodes for all  $1 \leq k \leq n$  at the same time. It generates a polynomial.

Piecewise Hermite interpolatory method can be used when we know the derivative of the interpolated function. It uses the trick of piecewise interpolation to eliminate the Runge phenomenon. This works because it only uses the polynomial of degree three and no higher-degree polynomial is involved. It generates a  $C^1$  function.

Natural cubic spline method generates a  $C^2$  function with knowing the derivative of the interpolated function. It eliminates Runge phenomenon as well. The reason is the same as the one for piecewise Hermite interpolatory method.

## 2 Codes and Numerical Experiments

In this section, we use these three interpolation algorithms to give approximation to a given function

$$f(x) = \frac{10}{1+x^4}$$

, by interpolating on 11 points  $-5, -4, \dots, 4, 5$ .

We package the function  $f$  and its derivative  $df$  as functions in Matlab. The codes are as follows:

```
1 function f=f(x)
2 f=10./(1+x.^4);
3 end
```

```
1 function df=df(x)
2 df=-40*x.^3./(1+x.^4).^2;
3 end
```

### 2.1 Newton's Interpolatory Polynomial

Firstly, we use Newton's method. For convenience, the program is packaged as a function of interpolation nodes, the function approximated and a variable  $t$ , so do the program for the other two algorithms. The codes are as follows:

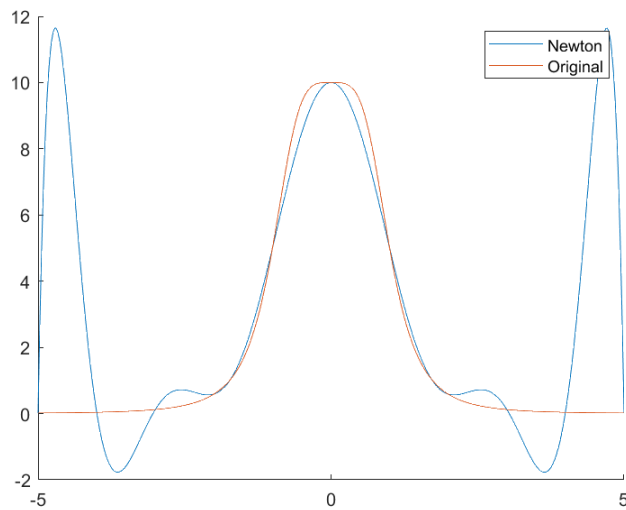
```
1 function newton=newton(x,t,f)
2 %x is interpolating nodes, t is variable,
3 %f is interpolated function
4 n=length(x);
5 F=zeros(n,n);
6 for i=1:n
7     F(i,1)=f(x(i));
8     for j=2:i
```

```

9          F(i,j)=(F(i,j-1)-F(i-1,j-1))./(x(i)-x(i-j+1));
10      end
11  end
12  newton=zeros(1,length(t));
13  for i=1:n
14      d=ones(1,length(t));
15      for j=1:i-1
16          d=d.*(t-x(j));
17      end
18      newton=newton+d.*F(i,i);
19  end
20  end

```

To see the behavior of the function given by interpolation and the function  $f$ , we plot their graph together. The codes used is put in appendix A and the picture is as below:



From the picture, we see the interpolating polynomial approximates  $f$  well in a neighborhood of 0. But when it comes to the edges of  $[-5, 5]$ , the interpolating polynomial oscillates wildly from

$f$ . This phenomenon is caused by the oscillatory nature of polynomials of high degree, which is called Runge Phenomenon.

## 2.2 Piecewise Hermite Interpolatory Polynomial

Secondly, we use piecewise Hermite interpolation. The codes is as follow:

```

1  function hermite=hermite(x,t,f,df)
2  %x is interpolating nodes,t is variable,
3  %f is interpolated function,df is derivative of f
4  n=length(x);
5  hermite=zeros(1,length(t));
6  for i=1:n
7  hermite=hermite+H(i,t,0).*f(x(i))+H(i,t,1).*df(x(i));
8  end
9  function H=H(m,t,a)
10     if a==0
11         if m==1
12             c=(x(1)<=t) & (t<=x(2));
13             H(c)=(1+2.*(t(c)-x(1))./(x(2)-x(1))). ...
14                 .*((t(c)-x(2))./(x(1)-x(2))).^2;
15             H(~c)=0;
16         else
17             if 2<=m && m<=n-1
18                 c=(x(m-1)<=t) & (t<=x(m));
19                 d=(x(m)<t) & (t<=x(m+1));
20                 H=zeros(1,length(t));

```

```

21         H(c)=(1+2.*(t(c)-x(m))./(x(m-1)-x(m)))...
22             .*((t(c)-x(m-1))./(x(m)-x(m-1))).^2;
23         H(d)=(1+2.*(t(d)-x(m))./(x(m+1)-x(m)))...
24             .*((t(d)-x(m+1))./(x(m)-x(m+1))).^2;
25     else
26         c=(x(n-1)<=t) & (t<=x(n));
27         H=zeros(1,length(t));
28         H(c)=(1+2.*(t(c)-x(n))./(x(n-1)-x(n)))...
29             .*((t(c)-x(n-1))./(x(n)-x(n-1))).^2;
30     end
31 end
32 else
33     if m==1
34         c=(x(1)<=t) & (t<=x(2));
35         H(c)=(t(c)-x(1)).*((t(c)-x(2))./(x(1)-x(2))).^2;
36         H(~c)=0;
37     else
38         if 2<=m && m<=n-1
39             c=(x(m-1)<=t) & (t<=x(m));
40             d=(x(m)<t) & (t<=x(m+1));
41             H=zeros(1,length(t));
42             H(c)=(t(c)-x(m)).*((t(c)-x(m-1))...
43                 ./ (x(m)-x(m-1))).^2;
44             H(d)=(t(d)-x(m)).*((t(d)-x(m+1))...
45                 ./ (x(m)-x(m+1))).^2;

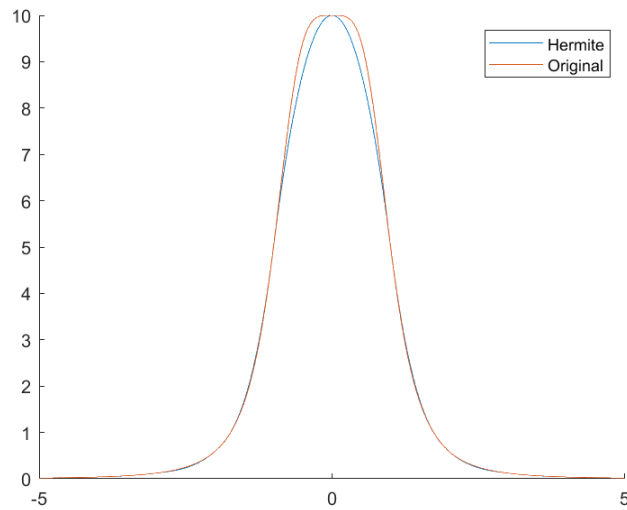
```

```

46         else
47             c=(x(n-1)<=t) & (t<=x(n));
48             H=zeros(1,length(t));
49             H(c)=(t(c)-x(n)).*((t(c)-x(n-1))...
50                 ./(x(n)-x(n-1))).^2;
51         end
52     end
53 end
54 end
55
56 end

```

Again, to see the behavior of the function given by interpolation and the function  $f$ , we plot their graph together. The codes used is put in appendix B and the picture is as below:



From the picture, we see that the piecewise Hermite interpolatory polynomial approximates  $f$  very well in the whole interval and it eliminates the Runge Phenomenon successfully.

## 2.3 Natural Cubic Spline Interpolatory Polynomial

Secondly, we use natural cubic spline interpolation. The codes is as follow:

```
1 function NCS=NCS(x,t,f)
2 %x is interpolating nodes,t is variable,
3 %f is interpolated function
4 n=length(x);
5 a=f(x);
6 b=zeros(1,n);
7 c=zeros(1,n);
8 d=zeros(1,n);
9 h=zeros(1,n);
10 alpha=zeros(1,n);
11 l=zeros(1,n);
12 mu=zeros(1,n);
13 z=zeros(1,n);
14 for i=1:n-1
15     h(i)=x(i+1)-x(i);
16 end
17 for i=2:n-1
18     alpha(i)=(a(i+1)-a(i)).*3./h(i)-(a(i)-a(i-1)).*3./h(i-1);
19 end
20 l(1)=1;
21 mu(1)=0;
22 z(1)=0;
23 for i=2:n-1
```

```

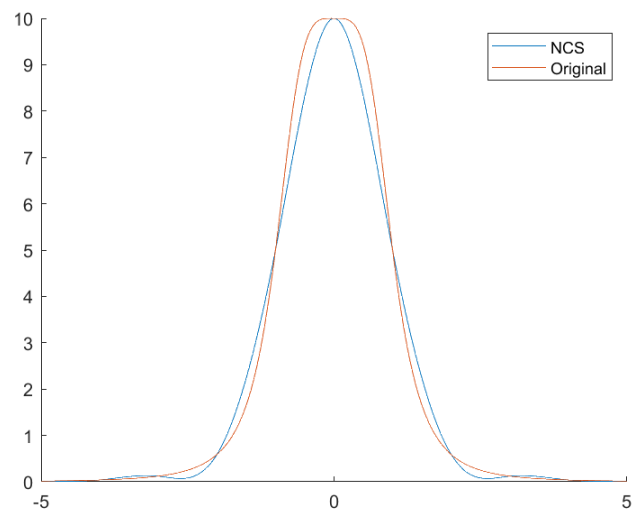
24     l(i)=2.*(x(i+1)-x(i-1))-h(i-1).*mu(i-1);
25     mu(i)=h(i)./l(i);
26     z(i)=(alpha(i)-h(i-1).*z(i-1))./l(i);
27 end
28 l(n)=1;
29 z(n)=0;
30 c(n)=0;
31 for j=1:n-1
32     c(n-j)=z(n-j)-mu(n-j).*c(n-j+1);
33     b(n-j)=(a(n-j+1)-a(n-j))./h(n-j)-h(n-j)...
34         .*(c(n-j+1)+2.*c(n-j))./3;
35     d(n-j)=(c(n-j+1)-c(n-j))./(3.*h(n-j));
36 end
37 S=zeros(1,length(t));
38 for j=1:n-1
39     p=(x(j)<=t) & (t<x(j+1));
40     S(p)=a(j)+b(j).*(t(p)-x(j))+c(j).*(t(p)-x(j))...
41         .^2+d(j).*(t(p)-x(j)).^3;
42 end
43 S(length(t))=a(n);
44 NCS=S;
45 end

```

As before, to see the behavior of the function given by interpolation and the function  $f$ , we plot their graph together. The codes used is put in appendix C and the picture is as below:

From the picture, we see that the natural cubic interpolatory polynomial approximates  $f$  very well (although not so well as the piecewise hermite interpolation). It also eliminates the Runge



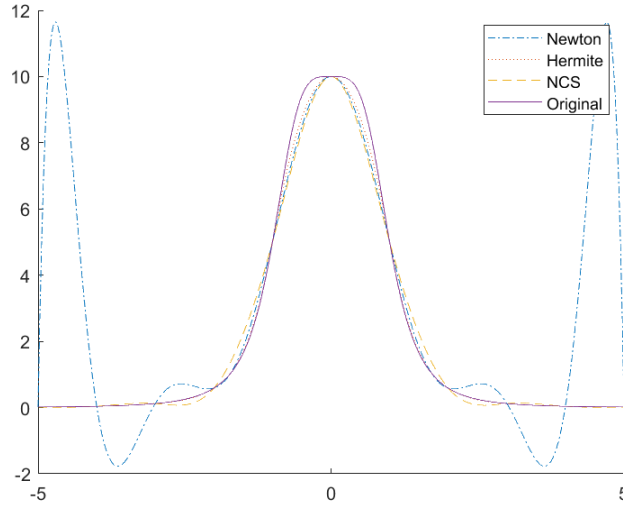


Phenomenon successfully.

### 3 Discussions and Conclusions

#### 3.1 More Experiments

In the previous sections, we interpolate the function  $f = \frac{10}{1+x^4}$ . Plotting the function given by this three algorithms together, we get the picture below (codes of plotting this picture is in appendix D).



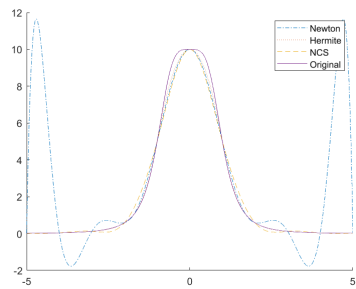
To see the difference between these three algorithms more generally, we change the interpolated function  $f$  and plot a picture as the one of  $f$  for each of them. These picture is in 3.1 (the codes is in appendix E).

From the pictures, we can see that both of the Hermite method and the natural cubic spline method eliminate the Runge phenomenon successfully for every function. However, at the point with big second derivative, the functions generated by this two method will oscilate mildly. This can be seen from the construction of the interpolatory polynomial. So these method will behave better on those functions with small second derivative.

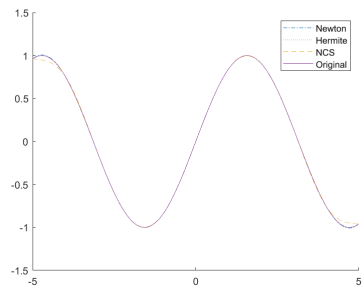
On the other hand, Newton's method is tremendously effected by Runge phenomenon. However, we observe that the Runge phenomenon almost never occurs when the function is monetone.

#### 3.2 Conclusions

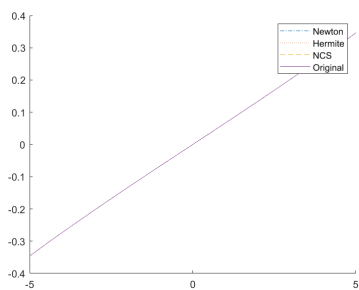
Newton's method is the most convenient in this three method, but Runge phenomenon cause disaster of oscillation. The other two method eliminates Runge phenomenon by using polynomial of degree three piecewisely. But the piecewise Hermite method requires the derivative of the interpolated function while the natural cubic spline method not.



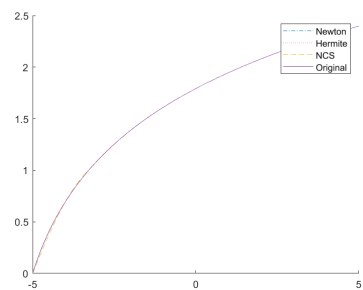
(a)  $\frac{10}{1+x^4}$



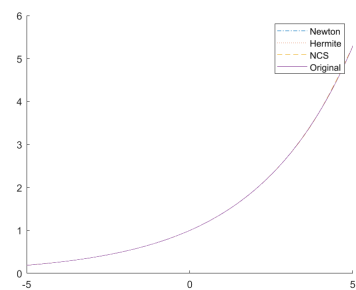
(b)  $\sin(x)$



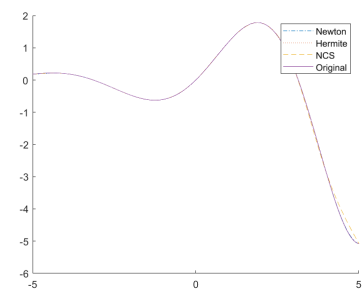
(c)  $\tan(\frac{x}{15})$



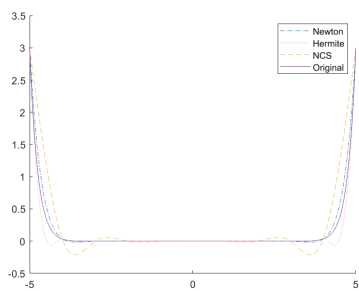
(d)  $\log(6+x)$



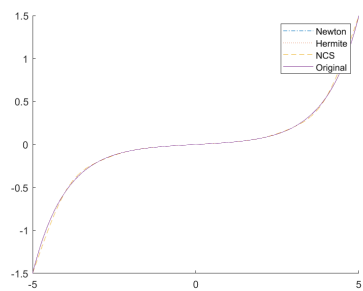
(e)  $e^{\frac{x}{3}}$



(f)  $e^{\frac{x}{3}}\sin(x)$



(g)  $3 \frac{x^2+1}{5^{10}}$



(h)  $3 \frac{\sinh(x)}{e^5}$

## A Code 1

```
1 clear
2 clc
3 hold on
4 a=[-5:0.01:5];%plotting nodes
5 x=[-5:5];%interpolating nodes
6 plot(a,newton(x,a,@f))
7 plot(a,f(a))
8 legend('Newton','Original')%adding legend
9 hold off
```

## B Code 2

```
1 clear
2 clc
3 hold on
4 a=[-5:0.01:5];%plotting nodes
5 x=[-5:5];%interpolating nodes
6 plot(a,hermite(x,a,@f,@df))
7 plot(a,f(a))
8 legend('Hermite','Original')%adding legend
9 hold off
```

## C Code 3

```
1 clear
2 clc
3 a=[-5:0.01:5];%plotting nodes
4 x=[-5:5];%interpolating nodes
5 hold on
6 plot(a,NCS(x,a,@f))
7 plot(a,f(a))
8 legend('NCS','Original')%adding legend
9 hold off
```

## D Code 4

```
1 clear
2 clc
3 hold on
4 a=[-5:0.01:5];%plotting nodes
5 x=[-5:5];%interpolating nodes
6 plot(a,newton(x,a,@f),'-.',a,hermite(x,a,@f,@df),...
7      ': ',a,NCS(x,a,@f),'—',a,f(a))
8 legend('Newton','Hermite','NCS','Original')%adding legend
9 hold off
```

## E Code 5

```
1 function mf=mf(x,n)
2 switch n
3     case 1
4         mf=10./(1+x.^4);
5     case 2
6         mf=sin(x);
7     case 3
8         mf=tan(x./15);
9     case 4
10        mf=log(6+x);
11    case 5
12        mf=exp(x./3);
13    case 6
14        mf=exp(x./3).*sin(x);
15    case 7
16        mf=(x.^20+1)./5^20.*3;
17    case 8
18        mf=sinh(x)./exp(5).*3;
19 end
20 end
```

```
1 function dmf=dmf(x,n)
2 switch n
```

```

3      case 1
4          dmf=-40.*x.^3./(1+x.^4).^2;
5      case 2
6          dmf=cos(x);
7      case 3
8          dmf=1/15.*1./((cos(x./15)).^2);
9      case 4
10         dmf=1./(6+x);
11     case 5
12         dmf=1/3.*exp(x./3);
13     case 6
14         dmf=1/3.*exp(x./3).*sin(x)+cos(x).*exp(x./3);
15     case 7
16         dmf=20.*x.^19./5^20.*3;
17     case 8
18         dmf=cosh(x)./exp(5).*3;
19 end
20 end

```

```

1  clear
2  clc
3  a=[-5:0.01:5];%plotting nodes
4  x=[-5:5];%interpolating nodes
5  for i=1:8
6      %plotting for every function

```

```

7  figure(i)

8  hold on

9  plot(a,newton(x,a,@(a)mf(a,i)), '-.', ...
10      a, hermite(x,a,@(a)mf(a,i),@(a)dmf(a,i)), ...
11      ': ', a, NCS(x,a,@(a)mf(a,i)), '—', a, mf(a,i))

12  legend('Newton', 'Hermite', 'NCS', 'Original')%adding legend

13  hold off

14  end

```

## References

- [1] Richard L. Burden, J. Douglas Faires, Annette M. Burden, Numerical Analysis (Tenth Edition), Cengage Learning, 2014.