

实验一 插值方法

姓名：徐昊博 学号：21013134

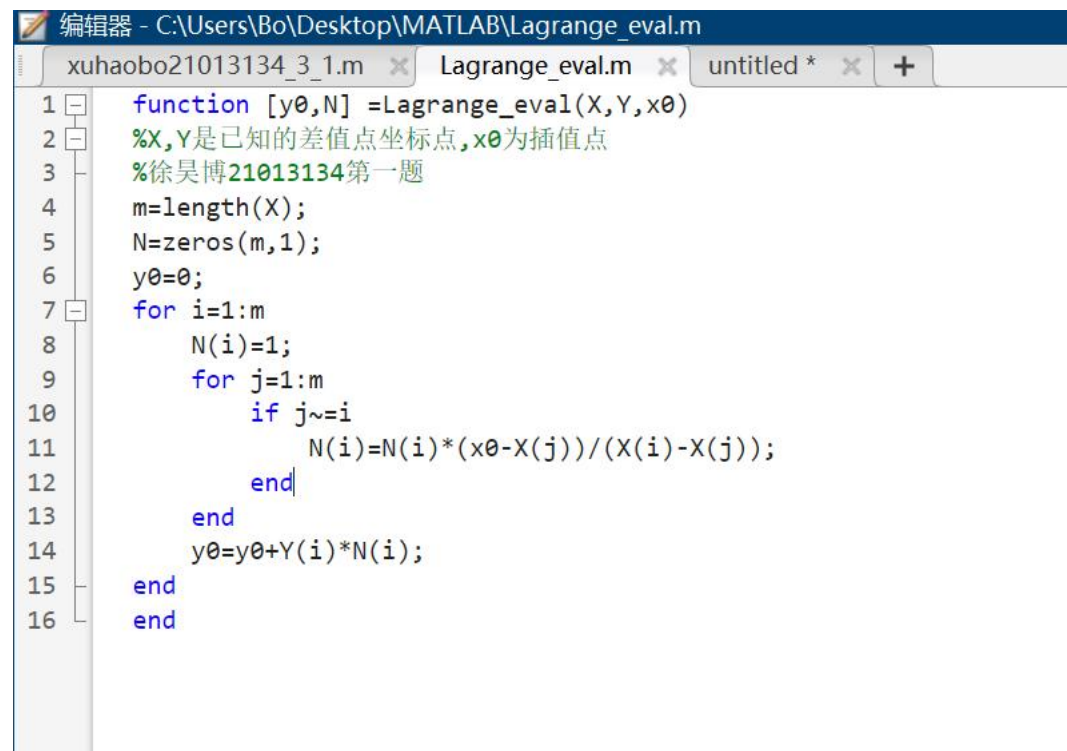
3.1

设计思想：由题意我们知道所需要的插值公式为拉格朗日插值公式，拉格朗日插值公式格式如下：

$$P_n(x) = \sum_{i=1}^n y_i \left(\prod_{\substack{j=1 \\ j \neq i}}^n \frac{(x - x_j)}{(x_i - x_j)} \right)$$

事实上，在逻辑上表现为二重循环，内循环（j 循环）累乘求得系数，然后再通过外循环（i 循环）累加得出插值结果 y。

根据上述公式得到了 Lagrange 算法的代码如图：



```
编辑器 - C:\Users\Bo\Desktop\MATLAB\Lagrange_eval.m
xuhaobo21013134_3_1.m  Lagrange_eval.m  untitled *  +
1  function [y0,N] =Lagrange_eval(X,Y,x0)
2  %X,Y是已知的差值点坐标点,x0为插值点
3  %徐昊博21013134第一题
4  m=length(X);
5  N=zeros(m,1);
6  y0=0;
7  for i=1:m
8      N(i)=1;
9      for j=1:m
10         if j~=i
11             N(i)=N(i)*(x0-X(j))/(X(i)-X(j));
12         end
13     end
14     y0=y0+Y(i)*N(i);
15 end
16 end
```

上述程序中 X,Y 是长度相等的向量，用来表示插值点的序列，该长度也决定了插值的精度。x=0.5 时，通过改变插值点向量长度分别得到了一次、二次和三次 Lagrange 插值结果分别如图所示：（下一页）

```
编辑器 - C:\Users\Bo\Desktop\MATLAB\xuhaobo21013134_3_1.m
xuhaobo21013134_3_1.m Lagrange_eval.m +
1 x=[0.4,0.55];
2 y=[0.41075,0.57815];
3 x0=0.5;
4 disp('21013134 徐昊博 第一题');
5 disp('一次Lagrange插值公式: ');
6 disp('y0=');
7 disp(Lagrange_eval(x,y,x0));
8
9 x=[0.4,0.55,0.8];
10 y=[0.41075,0.57815,0.88811];
11 disp('二次Lagrange插值公式: ');
12 disp('y0=');
13 disp(Lagrange_eval(x,y,x0));
14
15 x=[0.4,0.55,0.8,0.9];
16 y=[0.41075,0.57815,0.88811,1.02652];
17 disp('三次Lagrange插值公式: ');
18 disp('y0=');
19 disp(Lagrange_eval(x,y,x0));

命令行窗口
>> xuhaobo21013134_3_1
21013134 徐昊博 第一题
一次Lagrange插值公式:
y0=
    0.5223500000000000

二次Lagrange插值公式:
y0=
    0.5208020000000000

三次Lagrange插值公式:
y0=
    0.521109714285714

fx >>
```

$x=0.7$ 时，一次，二次，三次插值结果分别如图所示：

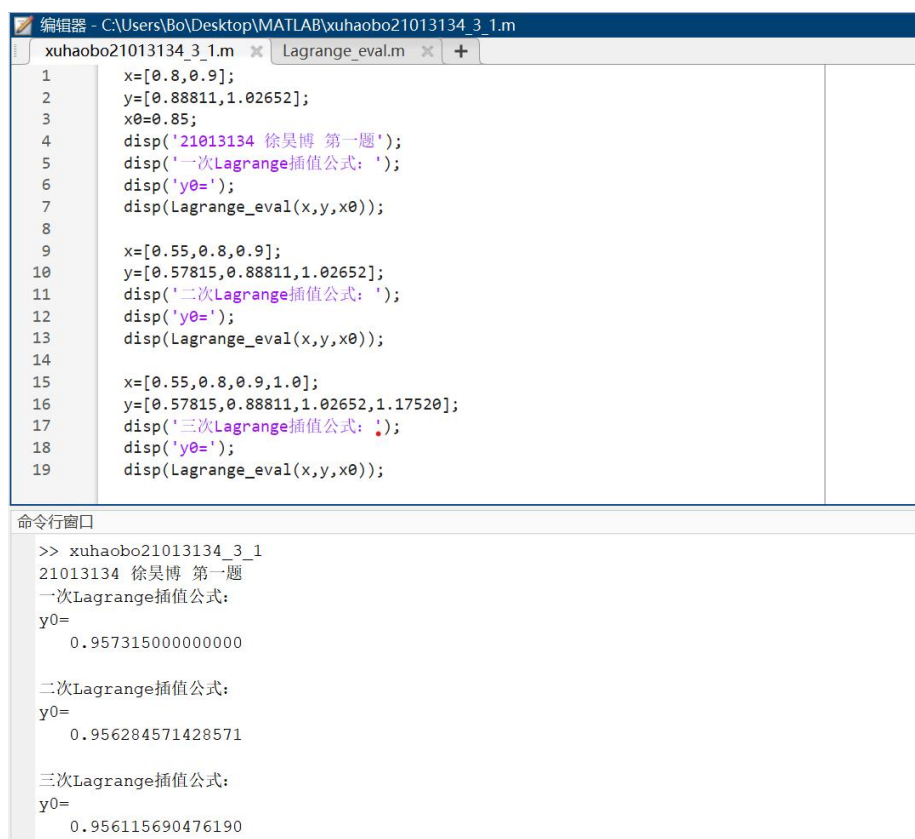
```
编辑器 - C:\Users\Bo\Desktop\MATLAB\xuhaobo21013134_3_1.m
xuhaobo21013134_3_1.m Lagrange_eval.m +
1 x=[0.55,0.8];
2 y=[0.57815,0.88811];
3 x0=0.7;
4 disp('21013134 徐昊博 第一题');
5 disp('一次Lagrange插值公式: ');
6 disp('y0=');
7 disp(Lagrange_eval(x,y,x0));
8
9 x=[0.4,0.55,0.8];
10 y=[0.41075,0.57815,0.88811];
11 disp('二次Lagrange插值公式: ');
12 disp('y0=');
13 disp(Lagrange_eval(x,y,x0));
14
15 x=[0.4,0.55,0.8,0.9];
16 y=[0.41075,0.57815,0.88811,1.02652];
17 disp('三次Lagrange插值公式: ');
18 disp('y0=');
19 disp(Lagrange_eval(x,y,x0));

命令行窗口
>> xuhaobo21013134_3_1
21013134 徐昊博 第一题
一次Lagrange插值公式:
y0=
    0.7641260000000000

二次Lagrange插值公式:
y0=
    0.7594820000000000

三次Lagrange插值公式:
y0=
    0.758558857142857
```

$x=0.85$ 时，一次，二次，三次插值结果分别如图所示：



The image shows a MATLAB editor window with a script named 'Lagrange_eval.m'. The script defines three sets of data points and calculates the Lagrange interpolation results for $x=0.85$ using first, second, and third degree polynomials. The command window below shows the execution results.

```
1 x=[0.8,0.9];
2 y=[0.88811,1.02652];
3 x0=0.85;
4 disp('21013134 徐昊博 第一题');
5 disp('一次Lagrange插值公式: ');
6 disp('y0=');
7 disp(Lagrange_eval(x,y,x0));
8
9 x=[0.55,0.8,0.9];
10 y=[0.57815,0.88811,1.02652];
11 disp('二次Lagrange插值公式: ');
12 disp('y0=');
13 disp(Lagrange_eval(x,y,x0));
14
15 x=[0.55,0.8,0.9,1.0];
16 y=[0.57815,0.88811,1.02652,1.17520];
17 disp('三次Lagrange插值公式: ');
18 disp('y0=');
19 disp(Lagrange_eval(x,y,x0));
```

命令行窗口

```
>> xuhaobo21013134_3_1
21013134 徐昊博 第一题
一次Lagrange插值公式:
y0=
    0.9573150000000000

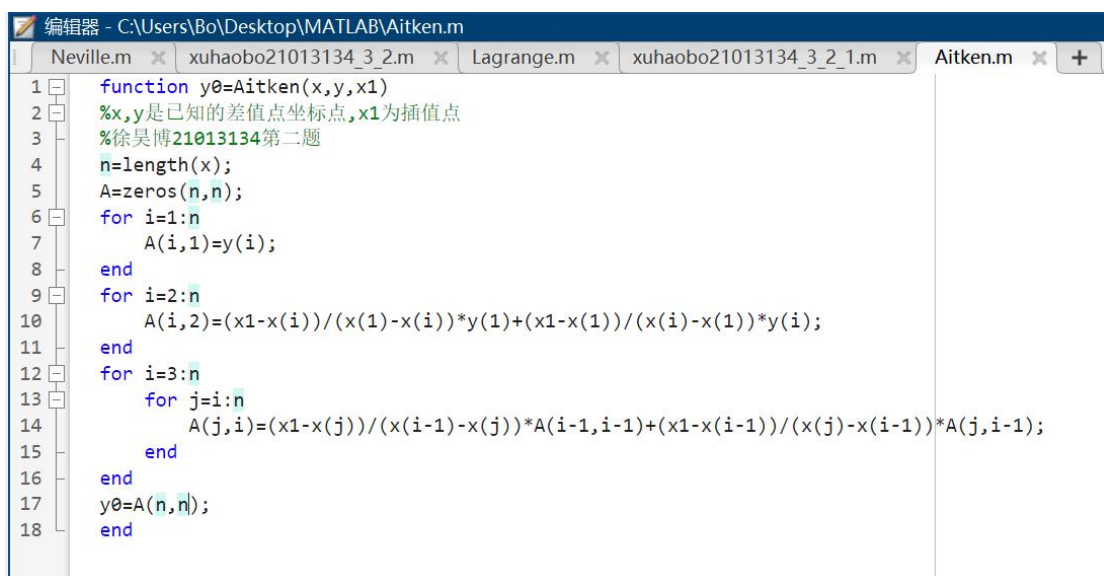
二次Lagrange插值公式:
y0=
    0.956284571428571

三次Lagrange插值公式:
y0=
    0.956115690476190
```

说明：从上述三个插值点的实验我们不难看出，插值次数越高，得到的插值数值就越精确。

3.2

Aitken 插值算法主要思想是化多点插值为两点插值，先将 n 个插值点中所有点与排列好的第一个点进行插值运算，得到了 $n-1$ 个新的插值点，再重复上述步骤，将 $n-1$ 个插值点中所有点与排列好的第一个点进行插值运算，得到 $n-2$ 个新的插值点，如此往复直到得到了一个数值，该数值即为最终结果，因而设计时采用双重循环，外循环控制插值点个数，内循环控制每次与第一个点进行插值运算。代码如下：



The image shows a MATLAB editor window with a script named 'Aitken.m'. The script defines a function 'Aitken(x,y,x1)' that implements the Aitken interpolation algorithm. The code uses nested loops to iteratively reduce the number of points until a single value is obtained.

```
1 function y0=Aitken(x,y,x1)
2 %x,y是已知的差值点坐标点,x1为插值点
3 %徐昊博21013134第二题
4 n=length(x);
5 A=zeros(n,n);
6 for i=1:n
7     A(i,1)=y(i);
8 end
9 for i=2:n
10     A(i,2)=(x1-x(i))/(x(1)-x(i))*y(1)+(x1-x(1))/(x(i)-x(1))*y(i);
11 end
12 for i=3:n
13     for j=i:n
14         A(j,i)=(x1-x(j))/(x(i-1)-x(j))*A(i-1,i-1)+(x1-x(i-1))/(x(j)-x(i-1))*A(j,i-1);
15     end
16 end
17 y0=A(n,n);
18 end
```

X=0.5,0.7,0.85 时插值结果如下所示:

```
编辑器 - C:\Users\Bo\Desktop\MATLAB\xuhaobo21013134_3_2_1.m
Neville.m x xuhaobo21013134_3_2.m x Lagrange.m x xuhaobo21013134_3_2_1.
1 x=[0.4,0.55,0.8,0.9,1.0];
2 y=[0.41075,0.57815,0.88811,1.02652,1.17520];
3 x0=0.5;
4 disp('21013134 徐昊博 第二题');
5 disp('当x=0.5时, Aitken插值结果: ');
6 disp(Aitken(x,y,x0));
7 x0=0.7;
8 disp('当x=0.7时, Aitken插值结果: ');
9 disp(Aitken(x,y,x0));
10 x0=0.85;
11 disp('当x=0.85时, Aitken插值结果: ');
12 disp(Aitken(x,y,x0));

命令行窗口
>> xuhaobo21013134_3_2_1
21013134 徐昊博 第二题
当x=0.5时, Aitken插值结果:
    0.521109714285714

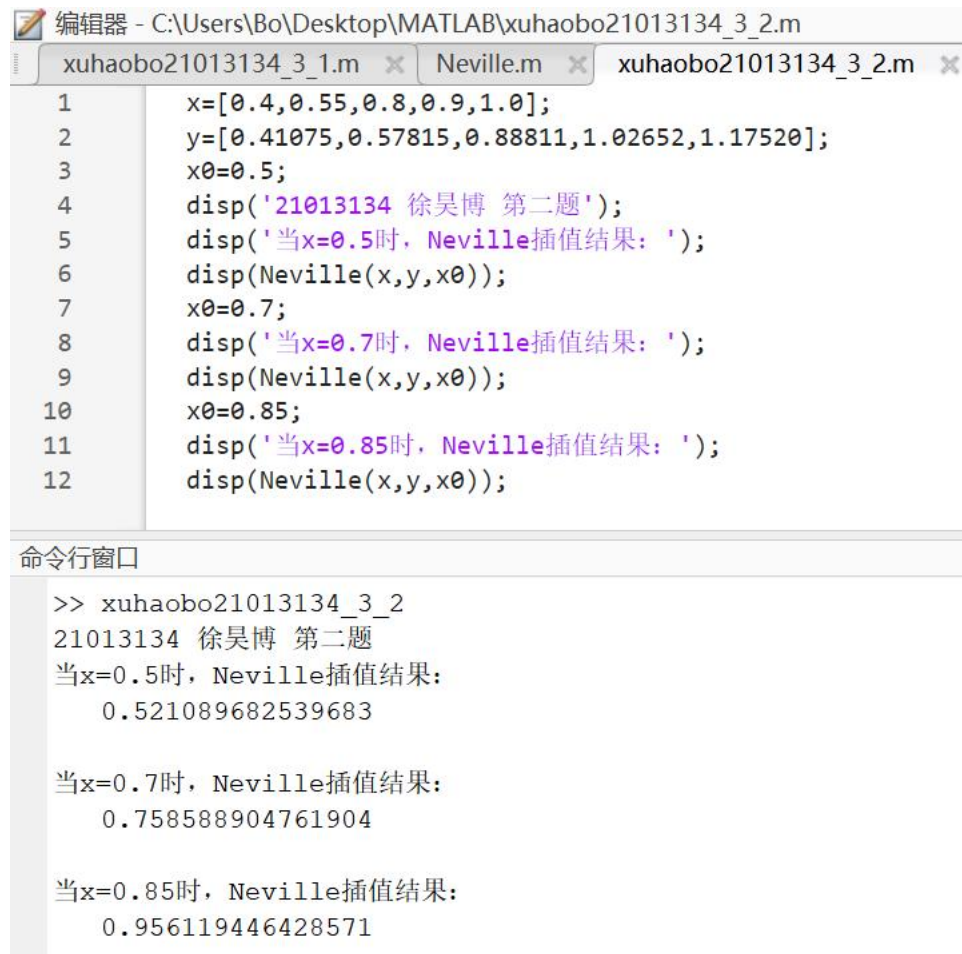
当x=0.7时, Aitken插值结果:
    0.758558857142857

当x=0.85时, Aitken插值结果:
    0.956130714285714
```

而 Neville 插值算法总体上与 Aitken 算法大同小异, 区别就是内循环中每次不是与第一个插值点进行插值运算, 而是与相邻点进行插值运算, 对于连续性较好的函数来说, 与相邻的点进行插值或许会得到更好的插值结果。代码如下:

```
编辑器 - C:\Users\Bo\Desktop\MATLAB\Neville.m
+1 xuhaobo21013134_3_3_2.m x Lagrange.m x Hermite3.m x xuhaobo21013134_3_3_3
1 function y0=Neville(X,Y,x0)
2 %X,Y是已知的差值点坐标点,x0为插值点
3 %徐昊博21013134第二题
4 m=length(X);
5 P=zeros(m,1);
6 P1=zeros(m,1);
7 P=Y;
8 for i=1:m
9     P1=P;
10    k=1;
11    for j=i+1:m
12        k=k+1;
13        P(j)=P1(j-1)+(P1(j)-P1(j-1))*(x0-X(k-1))/(X(j)-X(k-1));
14    end
15 end
16 y0=P(m);
17 end
18
```

X=0.5,0.7,0.85 时插值结果如下所示:



The image shows a MATLAB Editor window with a script named 'xuhaobo21013134_3_2.m'. The script defines a set of points $x = [0.4, 0.55, 0.8, 0.9, 1.0]$ and $y = [0.41075, 0.57815, 0.88811, 1.02652, 1.17520]$. It then uses the `Neville` function to interpolate at $x_0 = 0.5, 0.7, 0.85$. The Command Window shows the results of these interpolations.

```
1 x=[0.4,0.55,0.8,0.9,1.0];
2 y=[0.41075,0.57815,0.88811,1.02652,1.17520];
3 x0=0.5;
4 disp('21013134 徐昊博 第二题');
5 disp('当x=0.5时, Neville插值结果: ');
6 disp(Neville(x,y,x0));
7 x0=0.7;
8 disp('当x=0.7时, Neville插值结果: ');
9 disp(Neville(x,y,x0));
10 x0=0.85;
11 disp('当x=0.85时, Neville插值结果: ');
12 disp(Neville(x,y,x0));
```

命令行窗口

```
>> xuhaobo21013134_3_2
21013134 徐昊博 第二题
当x=0.5时, Neville插值结果:
    0.521089682539683


当x=0.7时, Neville插值结果:
    0.758588904761904

当x=0.85时, Neville插值结果:
    0.956119446428571
```

说明: 通过比较发现在本题当中, 使用 Aitken 插值方法和 Neville 插值方法得到的结果非常相似, 两个方法都很好地进行了对未知函数值的估计。

3.3

(1) 由于由题意需要作图, 所以图上每一个函数值都应当由插值函数来得到点的坐标数值, 因此为了便于观察图像, 我将函数自变量分为 1000 份, 而针对不同的 n 则只要相应改变插值函数 (之前编写过的) 输入当中的插值向量规模即可。作图代码如下几张图所示:



The image shows a MATLAB Editor window with a script named 'xuhaobo21013134_3_3_1.m'. The script defines the original function $y = 1/(1+x^2)$ and plots it. It then generates a fine grid of points x_0 and uses the `Lagrange` function to interpolate at these points. The plot shows the original function as blue dots and the interpolated function as red dots.

```
1 x=-5:0.001:5;
2 y=1./(1+x.^2);
3 plot(x,y,'b')
4 text(-4,0,'原函数')
5 text(0,2.5,'徐昊博21013134第三题')
6 hold on
7
8 x=linspace(-5,5,3);
9 y=1./(1+x.^2);
10 x0=linspace(-5,5,1000);
11 y0=zeros(1000);
12 for index=1:1000
13     y0(index)=Lagrange(x,y,x0(index));
14 end
15 plot(x0,y0,'r')
16 ylim([-1,3])
17 text(2,1,'n=2')
18 hold on
19
```



```

x=linspace(-5,5,5);
y=1./(1+x.^2);
x0=linspace(-5,5,1000);
y0=zeros(1000);
for index=1:1000
    y0(index)=Lagrange(x,y,x0(index));
end
plot(x0,y0,'r')
ylim([-1,3])
text(2,0.6,'n=4')
hold on

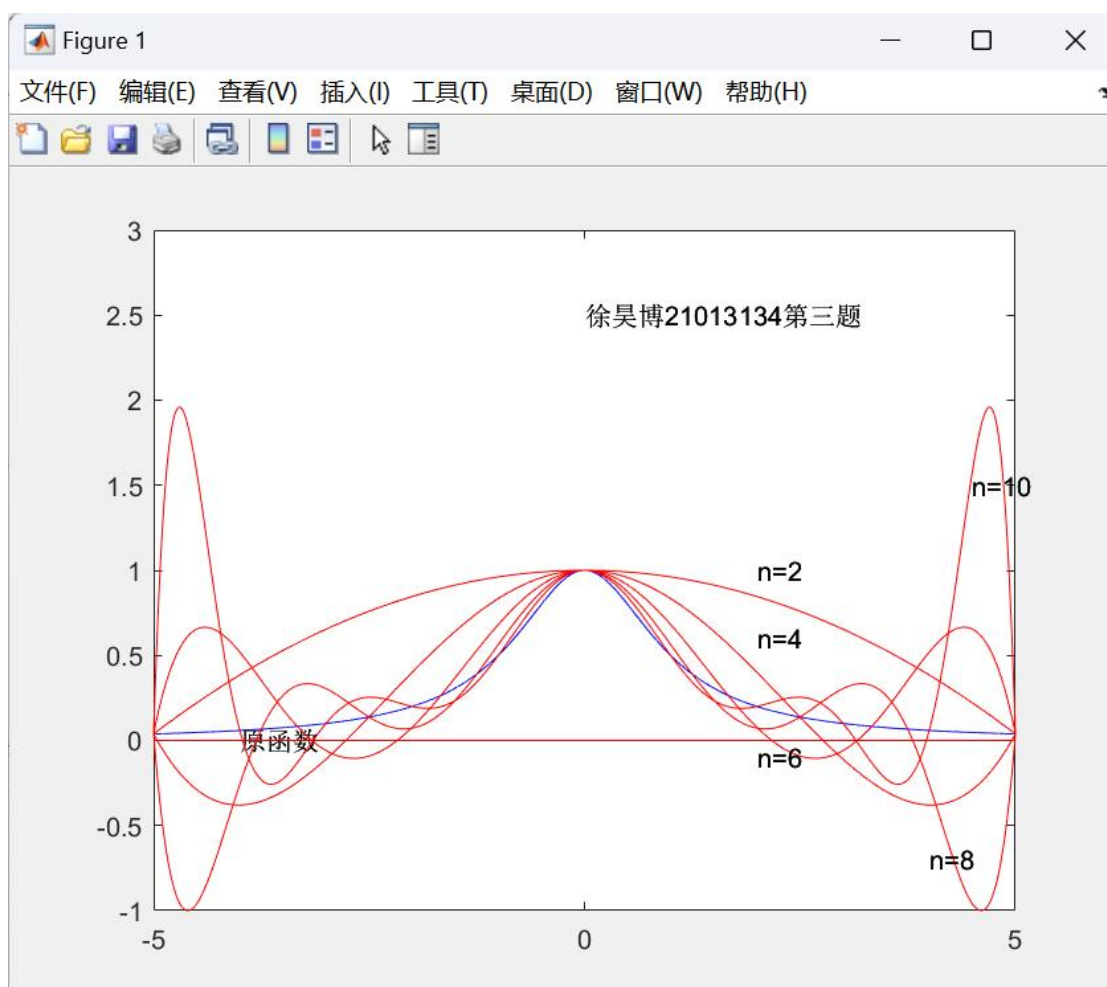
x=linspace(-5,5,7);
y=1./(1+x.^2);
x0=linspace(-5,5,1000);
y0=zeros(1000);
for index=1:1000
    y0(index)=Lagrange(x,y,x0(index));
end
plot(x0,y0,'r')
ylim([-1,3])
text(2,-0.1,'n=6')
hold on

x=linspace(-5,5,9);
y=1./(1+x.^2);
x0=linspace(-5,5,1000);
y0=zeros(1000);
for index=1:1000
    y0(index)=Lagrange(x,y,x0(index));
end
plot(x0,y0,'r')
ylim([-1,3])
text(4,-0.7,'n=8')
hold on

x=linspace(-5,5,11);
y=1./(1+x.^2);
x0=linspace(-5,5,1000);
y0=zeros(1000);
for index=1:1000
    y0(index)=Lagrange(x,y,x0(index));
end
plot(x0,y0,'r')
ylim([-1,3])
text(4.5,1.5,'n=10')

```

运行代码，得到的结果如下图所示：（蓝色的是原函数）



可以看到在远离原点的两侧，Lagrange 插值函数所带来的图像随着插值次数的升高产生了 Runge 现象，导致插值函数得到的不理想。

(2) 用同样的方法，也对 Hermite 插值方法进行图形化。Hermite 插值方法与 Lagrange 插值方法区别在于 Hermite 不仅要求插值点处函数值要求一致，而且要求函数的导数值也要一致：

Hermite插值公式如下：

$$y(x) = \sum_{i=1}^n h_i [(x_i - x)(2a_i y_i - y'_i) + y_i]$$

上式子中的 h_i 给定如下：

$$h_i = \prod_{j=1, j \neq i}^n \left(\frac{x - x_j}{x_i - x_j} \right)^2$$

a_i 给定如下：

$$a_i = \sum_{j=1, j \neq i}^n \frac{1}{x_i - x_j}$$

根据上述插值公式，我们可以写出求 Hermite 插值的代码：

```
编辑器 - C:\Users\Bo\Desktop\MATLAB\Hermite.m
+1 xuhaobo21013134_3_3_2.m x Lagrange.m x Hermite3.m x xuhaobo2101
1 %y0是函数值, y1是导数值, x0是插值自变量
2 function y=Hermite(x0,y0,y1,x)
3 n=length(x0);
4 m=length(x);
5 for k=1:m
6     yy=0.0;
7     for i=1:n
8         h=1.0;
9         a=0.0;
10        for j=1:n
11            if j~=i
12                h=h*((x(k)-x0(j))/(x0(i)-x0(j)))^2;
13                a=1/(x0(i)-x0(j))+a;
14            end
15        end
16        yy=yy+h*((x0(i)-x(k))*(2*a*y0(i)-y1(i))+y0(i));
17    end
18    y(k)=yy;
19 end
20 end
```

同理得到作图代码:

```
x=-5:0.001:5;
y=1./(1+x.^2);
plot(x,y,'b')
text(-4,0,'原函数')
text(-1.6,0.1,'徐昊博21013134第三题')
hold on
|
x=linspace(-5,5,3);
y=1./(1+x.^2);
y1=2*x./(x.^4+2*x.^2+1);
x0=linspace(-5,5,1000);
y2=Hermite(x,y,y1,x0);
plot(x0,y2,'r')
ylim([0,1])
text(2.2,0.7,'n=2')
hold on

x=linspace(-5,5,5);
y=1./(1+x.^2);
y1=2*x./(x.^4+2*x.^2+1);
x0=linspace(-5,5,1000);
y2=Hermite(x,y,y1,x0);
plot(x0,y2,'r')
ylim([0,1])
text(4.43,0.3,'n=4')
hold on
```



```

x=linspace(-5,5,7);
y=1./(1+x.^2);
y1=2*x./(x.^4+2*x.^2+1);
x0=linspace(-5,5,1000);
y2=Hermite(x,y,y1,x0);
plot(x0,y2,'k')
ylim([0,1])
text(4,0.85,'n=6')
hold on

```

```

x=linspace(-5,5,9);
y=1./(1+x.^2);
y1=2*x./(x.^4+2*x.^2+1);
x0=linspace(-5,5,1000);
y2=Hermite(x,y,y1,x0);
plot(x0,y2,'k')
ylim([0,1])
text(3.2,0.74,'n=8')
hold on

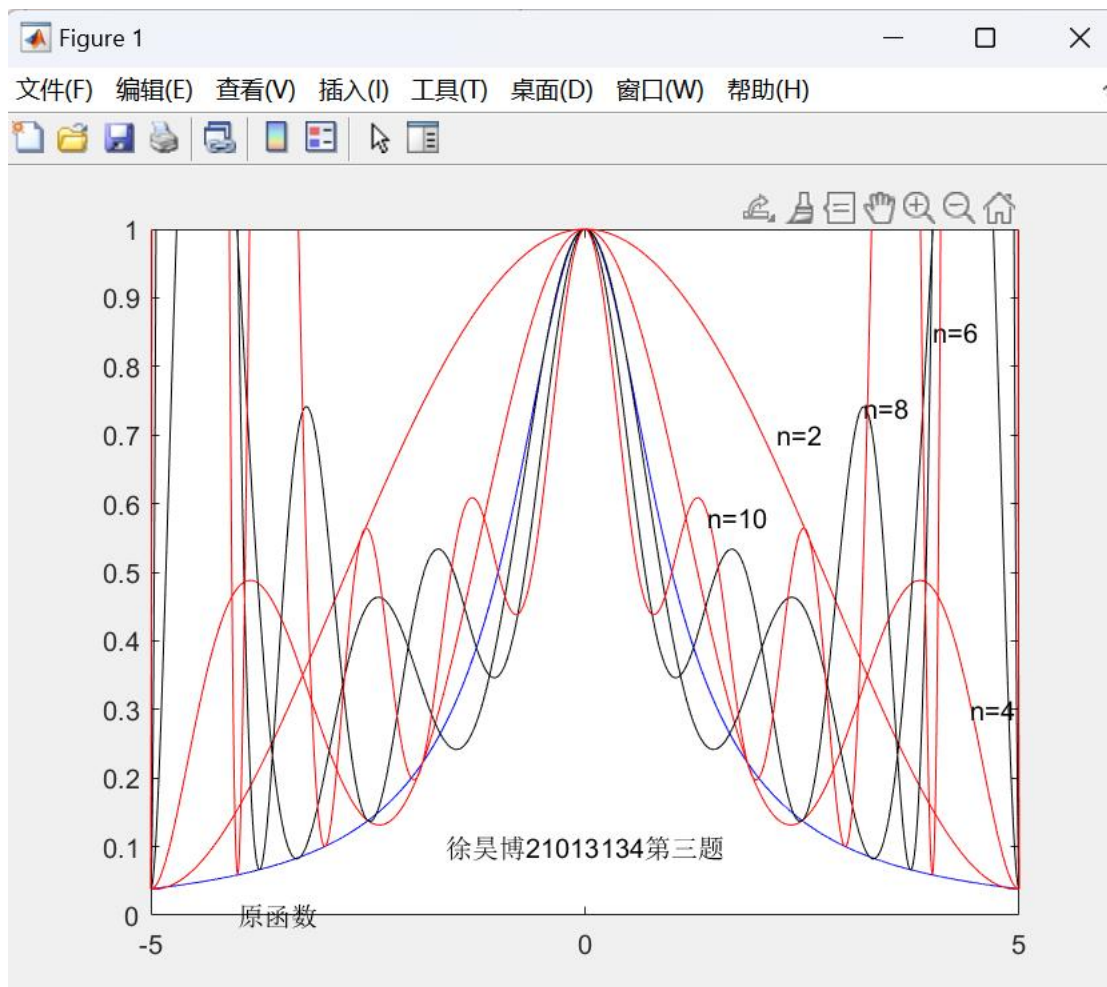
```

```

x=linspace(-5,5,11);
y=1./(1+x.^2);
y1=2*x./(x.^4+2*x.^2+1);
x0=linspace(-5,5,1000);
y2=Hermite(x,y,y1,x0);
plot(x0,y2,'r')
ylim([0,1])
text(1.41,0.58,'n=10')
hold on

```

运行代码，得到的结果如下图所示：（蓝色的是原函数）



说明：我们可以从上图看到，Hermite 插值函数也会随着插值次数的提高导致出现 Runge 现象，而且比 Lagrange 插值函数更为严重。

(3) 在 MATLAB2023a 版本当中，系统内部有分段线性插值函数，因而在生成图形时便可以调用这个函数，名为 `interp1`，而 Hermite 分段三次插值函数通过求解在特定条件下的函数的解来得到更精确的插值函数，一般地，在每个分段中，通常要求函数值和导数值都相等，其实现代码如下：

```

1 function y0=Hermite3(X,Y,DY,x0)
2     N=length(X);
3     for i=1:N
4         if x0>=X(i)&&X(i)<=X(i+1)
5             k=i;break;
6         end
7     end
8     a1=x0-X(k+1);
9     a2=x0-X(k);
10    a3=X(k)-X(k+1);
11    y0=(a1/a3)^2*(1-2*a2/a3)*Y(k)+(-a2/a3)^2*(1+2*a1/a3)*Y(k+1)+(a1/a3)^2*a2*DY(k)+(-a2/a3)^2*a1*DY(k+1);
12 end

```

最后将线性分段插值函数与 Hermite 三次分段插值函数形成图形放在一起，代码如下：

```

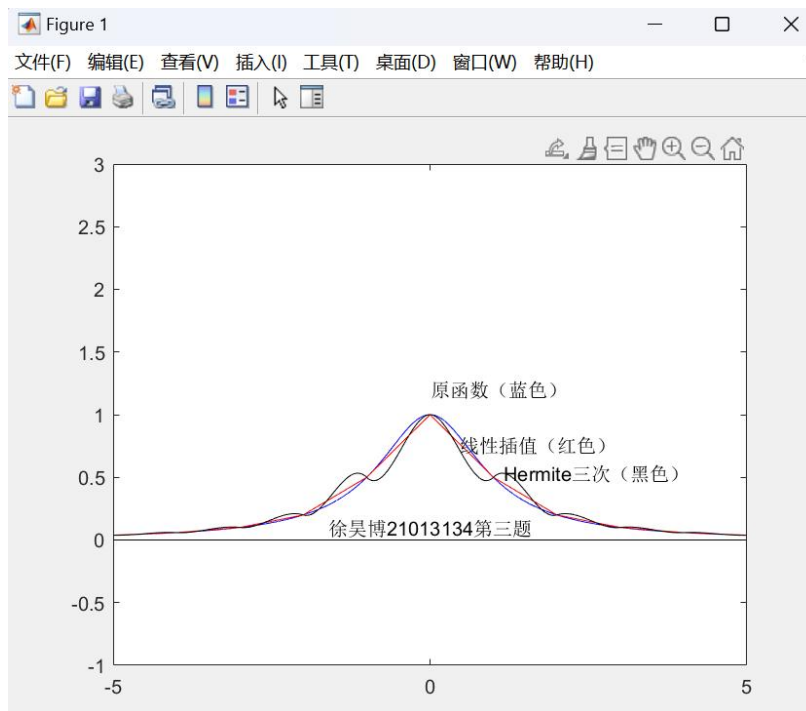
x=-5:0.001:5;
y=1./(1+x.^2);
plot(x,y,'b')
text(-1.6,0.1,'徐昊博21013134第三题')
text(0,1.2,'原函数（蓝色）')
hold on

x=linspace(-5,5,1000);
y=1./(1+x.^2);
x0=linspace(-5,5,11);
y0=1./(1+x0.^2);
y1=interp1(x0,y0,x);
plot(x,y1,'r');
text(0.475,0.762,'线性插值（红色）')
hold on

x0=linspace(-5,5,11);
y0=1./(1+x0.^2);
y1=2*x0./(x0.^4+2*x0.^2+1);
x=linspace(-5,5,1000);
y=zeros(1000);
for index=1:1000
    y(index)=Hermite3(x0,y0,y1,x(index));
end
plot(x,y,'k')
text(1.16,0.53,'Hermite三次（黑色）')
ylim([-1,3])

```

得到的两个插值函数图像与原函数对比如下：



说明：从上述的结果可以看出，使用分段线性插值函数得到的结果更加精确，得出的函数图形也与原函数更加吻合。

实验体会

通过本次插值实验的学习与实际动手操作，我更加清晰深刻地认识到了常见的插值方法，例如 **Lagrange** 插值方法，**Aitken** 逐步插值方法，**Neville** 逐步插值方法，分段插值方法等等，这些插值方法都是为了求出未知函数函数值而服务的。在插值效率与实际效果方面，对于单个插值点的求解，两点的 **Lagrange** 插值方法最为直接简单，而精度方面却不仅如人意，为此可以通过增加插值点，提高插值函数次数来逼近准确的函数值。而 **Aitken** 与 **Neville** 插值方法都是通过多点插值转化为单点插值来实现，在具体实现时通常代码上要多加注意循环位置相较于 **Lagrange** 插值，这两个方法便捷之处在于可以随时添加插值点，计算实现上更加便捷。而对于整个插值函数的求解，上述方法在函数图像上都会在远离原点处产生 **Runge** 现象。因而就有了分段插值法，通过将所求区间分成较小的部分，对每一个区间都进行按照求单个点的方法来进行插值，这样做可以使得插值函数与原函数更加吻合，但是求解过程通常较为繁琐。

在本次插值实验结束后，我也对插值有了更新的理解，插值本质上是通过几个离散的点，通过已知的几个函数点，通过不同拟合方法拟合出了一个尽可能贴近原函数的插值函数，最后将所求得的点带入拟合的函数得到插值结果，这对于一些表达式较为复杂或者甚至没有表达式的函数的未知函数值求解有了很大的帮助。