華東理工大學

信息科学与工程学院

《计算方法》实验报告

系　　别　计算机科学与工程系

专　　业　计算机科学与技术

年　　级　**2020** 级

姓　　名　刘子言

指导教师　郭卫斌

2021-2022 学年　第 **2** 学期

实验一　插值方法

一. 实验目的

（1）熟悉数值插值方法的基本思想，解决某些实际插值问题，加深对数值插值方法的理解。

（2）熟悉 Matlab 编程环境，利用 Matlab 实现具体的插值算法，并进行可视化。

二. 实验要求

用 Matlab 软件实现 Lagrange 插值、分段线性插值、Hermite 插值、Aitken 逐步插值算法，并用实例在计算机上计算和作图。

三. 实验内容

3-1：已知正弦积分 $f(x)=-\int_x^{\infty}\frac{sint}{t}dt$ 的数据表

x	0.3	0.4	0.5	0.6	0.7
y	0.29850	0.39646	0.49311	0.58813	0.68122

构造适合该数据表的一次、二次和三次 Lagrange 插值公式，计算 *x*=0.358, 0.462, 0.514, 0.635 时 *f*(*x*)的值，比较不同次数的插值公式的计算结果。

1、设计思想

Lagrange 插值方法要求插值函数 p(x)与所逼近的函数 f(x)在一系列的节点上去相同的函数值，实际上是用简单的曲线代替复杂的曲线来得到最终结果。根据节点数，可以分为两点插值、三点插值和多点插值公式，Lagrange 插值公式的形式如下：

$$y=\sum_{i=0}^{n}\left(\prod_{\substack{j=0\\ j\neq i}}^{n}\frac{x-x_j}{x_i-x_j}\right)y_i \qquad (1)$$

Lagrange 插值的算法设计：给定数据表 $(x_i,y_i),i=0,1,\dots,n$ 以及插值节点 x，根据公式（1），求得插值结果 y。

Lagrange 公式（1）具有累乘累加的嵌套结构，在逻辑上表现为二重循环，内循环（j 循环）累乘求得系数，然后通过外循环（i 循环）累加得出插值结果。

2、对应程序

题目 3-1：Lagrange 插值的函数文件名：“liuziyan_3_1.m”，代码如下：

```
function [y0, N] = liuziyan_3_1(X, Y, x0)
%Lagrange 插值算法函数
%X,Y 是已知的插值点坐标，N 是 Language 插值函数的权系数
%x0 是插值点，y0 是 Lagrenge 多项式在 x0 处的值
```

```
m = length(X);
N = zeros(m, 1);
y0 = 0;
for i = 1:m
    N(i) = 1;
    for j = 1:m
        if   j~=i
            N(i) = N(i)*(x0-X(j))/(X(i)-X(j));%迭乘
        end
    end
    y0 = y0+Y(i)*N(i);%迭加
end
```

3、实验结果（下列截图中的函数名为我的名字 liuziyan+题号）

3-1 实验的运行结果截图如下：

（1）一次插值

```
>> X=[0.3,0.4];Y=[0.29850,0.39646];x0=0.358;
>> [y0,N]=liuziyan_3_1(X,Y,x0)

y0 =

    0.3553


N =

    0.4200
    0.5800
```

```
>> X=[0.4,0.5];Y=[0.39646,0.49311];x0=0.462;
>> [y0,N]=liuziyan_3_1(X,Y,x0)

y0 =

    0.4564


N =

    0.3800
    0.6200
```

```
>> X=[0.5,0.6];Y=[0.49311,0.58813];x0=0.514;
>> [y0,N]=liuziyan_3_1(X,Y,x0)

y0 =

    0.5064


N =

    0.8600
    0.1400
```

```
>> X=[0.6,0.7];Y=[0.58813,0.68122];x0=0.635;
>> [y0,N]=liuziyan_3_1(X,Y,x0)

y0 =

    0.6207


N =

    0.6500
    0.3500
```

（2）二次插值

```
>> X=[0.3,0.4,0.5];Y=[0.29850,0.39646,0.49311];x0=0.358;
>> [y0,N] = liuziyan_3_1(X,Y,x0)

y0 =

    0.3555


N =

    0.2982
    0.8236
   -0.1218
```

```
>> X=[0.4,0.5,0.6];Y=[0.39646,0.49311,0.58813];x0=0.462;
>> [y0,N] = liuziyan_3_1(X,Y,x0)

y0 =

    0.4566


N =

    0.2622
    0.8556
   -0.1178
```

```
>> X=[0.5,0.6,0.7];Y=[0.49311,0.58813,0.68122];x0=0.514;
>> [y0,N] = liuziyan_3_1(X,Y,x0)

y0 =

    0.5065


N =

    0.7998
    0.2604
   -0.0602
```

```
>> X=[0.5,0.6,0.7];Y=[0.49311,0.58813,0.68122];x0=0.635;
>> [y0,N] = liuziyan_3_1(X,Y,x0)

y0 =

    0.6209


N =

   -0.1138
    0.8775
    0.2363
```

（3）三次插值

```
>> X=[0.3,0.4,0.5,0.6];Y=[0.29850,0.39646,0.49311,0.58813];x0=0.358;
>> [y0,N] = liuziyan_3_1(X,Y,x0)

y0 =

    0.3555


N =

    0.2405
    0.9966
   -0.2948
    0.0577
```

```
>> X=[0.3,0.4,0.5,0.6];Y=[0.29850,0.39646,0.49311,0.58813];x0=0.462;
>> [y0,N] = liuziyan_3_1(X,Y,x0)

y0 =

    0.4566


N =

   -0.0542
    0.4248
    0.6930
   -0.0636
```

```
>> X=[0.4,0.5,0.6,0.7];Y=[0.39646,0.49311,0.58813,0.68122];x0=0.514;
>> [y0,N] = liuziyan_3_1(X,Y,x0)

y0 =

    0.5065


N =

   -0.0373
    0.9118
    0.1484
   -0.0229
```

```
>> X=[0.4,0.5,0.6,0.7];Y=[0.39646,0.49311,0.58813,0.68122];x0=0.635;
>> [y0,N] = liuziyan_3_1(X,Y,x0)

y0 =

    0.6209


N =

    0.0512
   -0.2673
    1.0311
    0.1851
```

3-2：仿照附录 C 中“文件 1.2 逐步插值”程序（Neville 算法）编写相应的 Aitken 逐步插值算法的程序，根据实验题目 3-1 中所给数据，分别利用上述两种算法求正弦积分 *f*(*x*) 在 *x*=0.358, 0.462, 0.514, 0.635 处的值，比较两种算法的计算结果，并与 3-1 中的计算结果进行比较。

1、设计思想

多点 Lagrange 插值可以划归为两点插值的重复，令每一步增添一个新的节点，直到遍历所有的节点为止，就得到了 Aitken 逐步插值算法。通过 Aitken 逐步插值算法可以生成三角形的逐步插值表。Neville 算法与 Aitken 算法原理类似，也可以生成三角形的逐步插值表，它们都是将高阶插值逐步归结为线性插值最简单、最基本的重复。

Neville 逐步插值算法逐列生成新的值，且每次运算得到的新值为前一列相邻元素及其上一元素间的运算。Aitken 逐步插值算法也是逐列生成新的值，每次运算得到的新值为前一列元素与列首元素间的运算，因此只需将 Neville 插值算法的公式做适当修改即可，运算公式如下，其中对 i=1,2,...计算，k=0,1,2,...,i-1：

$$y_{ki}=\frac{x-x_i}{x_k-x_i}y_{k-1,k}+\frac{x-x_k}{x_i-x_k}y_{k-1,i} \qquad (2)$$

Aitken 逐步插值的算法设计：依据公式（2）逐行生成插值表；检查计算误差，对于给定精度 ε，当 $|y_{i,i+1}-y_{i-1,i}|<\varepsilon$ 时计算终止，并输出 $y_{i,i+1}$ 作为插值结果；自然停机，当 $i=n$ 时输出 $y_{n-1,n}$ 作为插值结果。

逐步插值公式（2）在具体实现过程中可以将系数合并化简，在逻辑上也表现为二重循环，内循环（j 循环）每一列中单个 y 的值，外循环（i 循环）表示逐列生成，i 每加 1 就进行下一列 y 的计算。

2、对应程序

（1）题目 3-2：Neville 插值的函数文件名：“liuziyan_3_2_Neville.m”，代码如下：

```
function y0 = liuziyan_3_2_Neville(X, Y, x0)
%Neville 逐步插值算法，逐列生成新的值，且每个新值为前一列相邻两元素间的运算
%X,Y 是已知插值点的坐标点，x0 是插值点，y0 是多项式在 x0 处的值
m = length(X);
P = zeros(m, 1);
P1 = zeros(m, 1);
P = Y;
for i=1:m
    P1 = P;
    k = 1;
    for j = i+1:m
```

```
        k = k+1;
        P(j) = P1(j-1)+(P1(j)-P1(j-1))*(x0-X(k-1))/(X(j)-X(k-1));
    end
    if abs(P(m)-P(m-1)) < 10^-6;
        y0 = P(m);
        return;
    end
end
y0 = P(m);
```

（2）题目 3-2：Aitken 插值的函数文件名：“liuziyan_3_2_Aitken.m”，代码如下：

```
function y0 = liuziyan_3_2_Aitken(X, Y, x0)
%Aitken 逐步插值算法，逐列生成新的值，每个新值为前一列元素与列首元素间的运算
%把 Neville 算法代码改写为 Aitken 算法
%关键在于将 j-1,k-1 改成每列第一个元素对应的值 i,k
%X,Y 是已知插值点的坐标点，x0 是插值点，y0 是多项式在 x0 处的值
m = length(X);
P = zeros(m, 1);
P1 = zeros(m, 1);
P = Y;
for i=1:m
    P1 = P;
    k = 1;
    for j = i+1:m
        P(j) = P1(i)+(P1(j)-P1(i))*(x0-X(k))/(X(j)-X(k));
    end
    k = k+1;
    if abs(P(m)-P(m-1)) < 10^-6;
        y0 = P(m);
        return;
    end
end
y0 = P(m);
```

3、实验结果

3-2 实验的运行结果截图如下：

（下列截图中的函数名是我的名字 liuziyan+题号+插值算法名称）

（1）Neville 逐步插值算法

```
>> X=[0.3,0.4,0.5,0.6,0.7];Y=[0.29850,0.39646,0.49311,0.58813,0.68122];x0=0.358;
>> y0 = liuziyan_3_2_Neville(X, Y, x0)

y0 =

    0.3555
```

```
>> X=[0.3,0.4,0.5,0.6,0.7];Y=[0.29850,0.39646,0.49311,0.58813,0.68122];x0=0.462;
>> y0 = liuziyan_3_2_Neville(X, Y, x0)

y0 =

    0.4566
```

```
>> X=[0.3,0.4,0.5,0.6,0.7];Y=[0.29850,0.39646,0.49311,0.58813,0.68122];x0=0.514;
>> y0 = liuziyan_3_2_Neville(X, Y, x0)

y0 =

    0.5065
```

```
>> X=[0.3,0.4,0.5,0.6,0.7];Y=[0.29850,0.39646,0.49311,0.58813,0.68122];x0=0.635;
>> y0 = liuziyan_3_2_Neville(X, Y, x0)

y0 =

    0.6209
```

（2）Aitken 逐步插值算法

```
>> X=[0.3,0.4,0.5,0.6,0.7];Y=[0.29850,0.39646,0.49311,0.58813,0.68122];x0=0.358;
>> y0 = liuziyan_3_2_Aitken(X, Y, x0)

y0 =

    0.3552
```

```
>> X=[0.3,0.4,0.5,0.6,0.7];Y=[0.29850,0.39646,0.49311,0.58813,0.68122];x0=0.462;
>> y0 = liuziyan_3_2_Aitken(X, Y, x0)

y0 =

    0.4561
```

```
>> X=[0.3,0.4,0.5,0.6,0.7];Y=[0.29850,0.39646,0.49311,0.58813,0.68122];x0=0.514;
>> y0 = liuziyan_3_2_Aitken(X, Y, x0)

y0 =

    0.5061
```

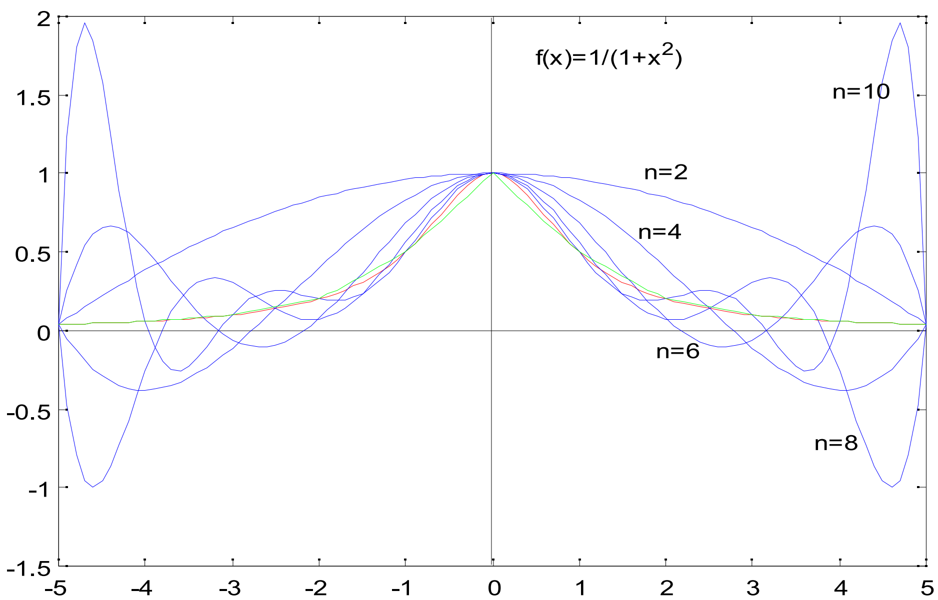
```
>> X=[0.3,0.4,0.5,0.6,0.7];Y=[0.29850,0.39646,0.49311,0.58813,0.68122];x0=0.635;
>> y0 = liuziyan_3_2_Aitken(X, Y, x0)

y0 =

  0.6208
```

综上所述，结合实验 3-1，并比较 3-2 中的两种插值算法的计算结果，可以得到：

在精确性上，对 Lagrange 插值算法，插值次数越高，计算结果越精确；逐步差值算法中，相同节点数的情况下，Neville 算法的计算结果比 Aitken 更加精确，因为 Neville 算法采用的是上一列相邻两个 y 值之间运算得到新的结果，能够更大效率地运用各节点的值，更快达到精度要求；Neville 逐步插值算法的结果准确性与 Lagrange 三次插值得到的结果相似；

在优越性上，逐步插值算法比 Lagrange 算法更优越，因为增加节点时并不需要全部重新计算。

3-3：对于函数$f(x)=\frac{1}{1+x^2}$，在利用 Lagrange 插值方法进行插值时，随着插值次数的增大，会出现如下图中所示的 Runge 现象：



要求：

（1）利用 Lagrange 插值方法验证 Runge 现象；

（2）将区间[-5,5]分为 n 等份（n=5,10,20），做$f(x)=\frac{1}{1+x^2}$的 Lagrange 分段线性插值函数 $L_5(x)$、$L_{10}(x)$、$L_{20}(x)$，考察上述三种插值在 x=-4.8、4.8 处的误差，并分析。

1、设计思想

（1）验证 Runge 现象

对于代数插值，插值多项式的次数随着节点个数的增加而升高，然而高次插值的逼近效果往往是不理想的，次数越高，会带来舍入误差的增大，引起计算失真。

本实验我们采用 Lagrange 插值方法验证 Runge 现象，Lagrange 插值算法依旧按照实验 3-1 设计的算法来进行实现，利用输入的阶数 n 计算步长 h，x 自取 1000 个点，调用 Lagrange 插值算法函数计算每个 x 对应的 y 值，在主函数中绘制出插值函数的图像。

（2）Lagrange 分段线性插值函数

分段线性插值，就是将被插值函数逐段多项式（一次）化，对于函数 f(x)，如果在每个子段上用线性插值，即用连接相邻节点的折线逼近所考察的曲线，就能保证一定的逼近效果。

具体过程是先将区间[-5,5]作一分划，将等分的份数作为参数 n（n=5,10,20）输入，并在每个子区间$[x_i, x_{i+1}]$上利用实验 3-1 设计的 Lagrange 插值算法构造插值多项式（阶数为 2），最后再将每个子段上的插值多项式装配拼接在一起，作为整个区间[-5,5]上的插值函数。再求这三个插值函数在 x=-4.8、4.8 处的值，最后分析误差。

2、对应程序

（1）题目 3-3：Lagrange 插值验证 Runge 现象的函数文件名：“liuziyan_3_3_Runge.m”，代码如下：

```
function y0 = liuziyan_3_3_Runge(n)
%插值节点选择取值范围上的等距节点，只输入阶数 n，每条线 x 取 1000 个点
for i=1:1001
    x(i)=-5+(i-1)*0.01;
    y(i)=lagrange(x(i),n);
end
plot(x,y)
function y1=lagrange(x,n)   %以下为插值多项式计算
h = 10/n;
y1 = 0;
N = 1;
for i=1:(n+1)
    for j=1:(n+1)
        if j==i
            N = 1*N;
        else
            N = N*(x-(-5+(j-1)*h))/((i-j)*h);%迭乘
        end
```

```
        end
        y1 = y1+N*(1/(1+(-5+(i-1)*h)^2));%迭加
        N = 1;
end
```

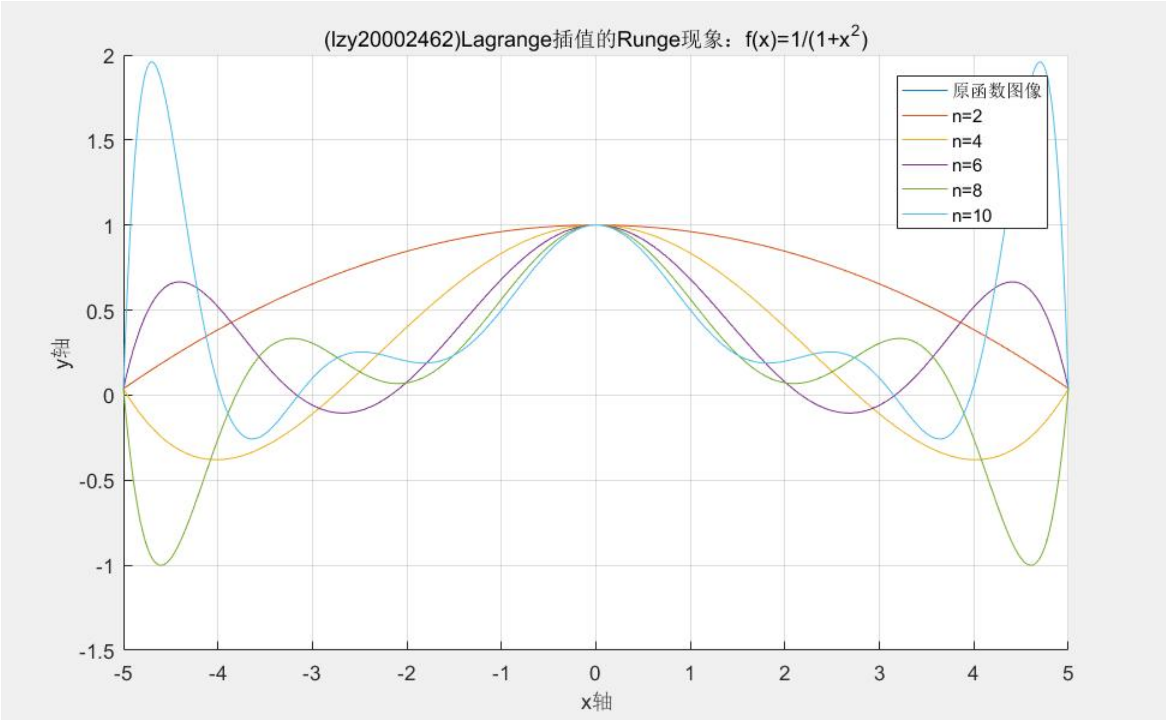
（3）题目 3-3：Lagrange 分段线性插值函数文件名：“liuziyan_3_3_Piece_Lagrange.m”，代码如下：

```
function y0 = liuziyan_3_3_Piece_Lagrange(x0,n)
%Lagrange 分段线性插值函数，n 是分段的段数，每段均线性插值
for i=1:1001
        x(i) = -5+(i-1)*0.01;
        y(i) = lagrange(x(i),n);
        if   x0 == x(i)
             y0 = y(i);
        end
end
plot(x,y)
hold on
grid on  %添加网格线
xlabel('x 轴');
ylabel('y 轴');
function y1=lagrange(x,n)   %以下为插值多项式计算
H = 10/n;%每段的长度
h = H/1;%每段中每步的长度
y1 = 0;
N = 1;
k = floor((x-(-5))/H)+1;%判断当前的 x 属于第几段
for i=1:2
        for j=1:2
                if j==i
                        N = 1*N;
                else
                        N = N*(x-(-5+(k-1)*H+(j-1)*h))/((i-j)*h);%迭乘
                end
          end
          y1 = y1+N*(1/(1+(-5+(k-1)*H+(i-1)*h)^2));%迭加
          N = 1;
end
```

3、实验结果（下列截图中的出现的函数名是我的名字 liuziyan+题号+插值算法名称）

3-2 实验的运行结果截图如下：

（1）Lagrange 插值验证 Runge 现象，绘制出的 Runge 现象图像如下：

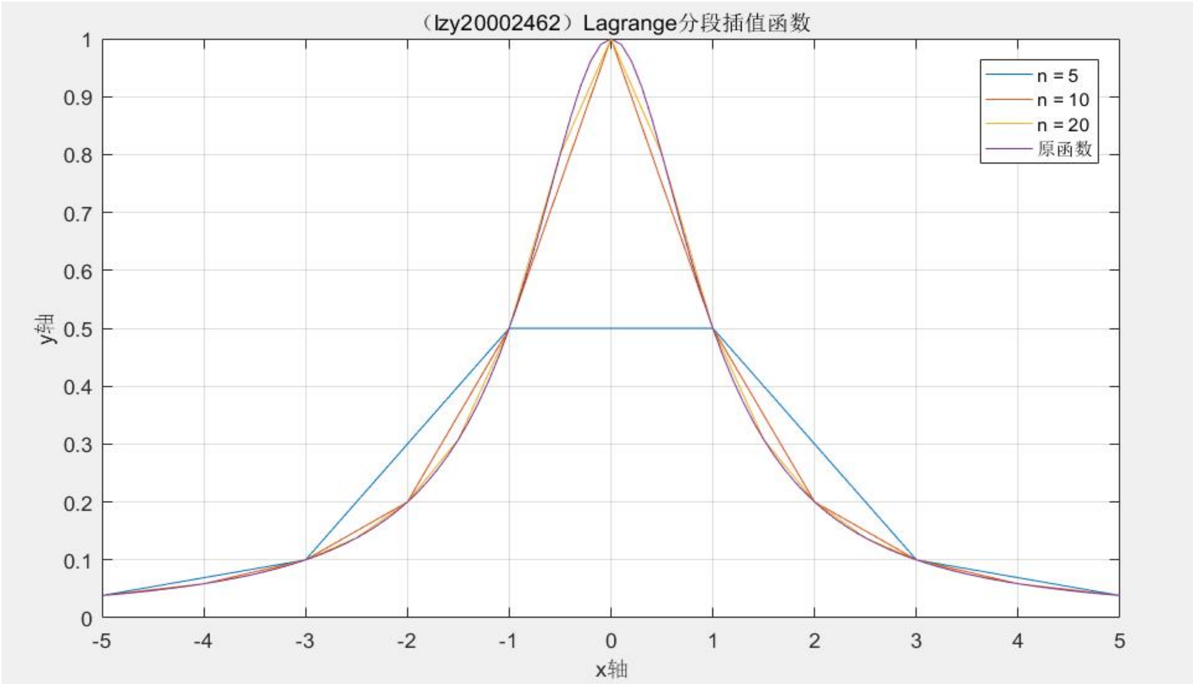


命令窗口输入的代码截图如下：

```
>> legend('原函数图像','n=2','n=4','n=6','n=8','n=10')
>> xlabel('x轴');
>> ylabel('y轴');
>> title('(lzy20002462)Lagrange插值的Runge现象：f(x)=1/(1+x^2)');
>> grid on
>> x=[-5:0.1:5];y=(1+x.^2).\1;
>> plot(x,y,':r')
>> hold on
>> liuziyan_3_3_Runge(2);
>> hold on
>> liuziyan_3_3_Runge(4);
>> hold on
>> liuziyan_3_3_Runge(6);
>> hold on
>> liuziyan_3_3_Runge(8);
>> hold on
>> liuziyan_3_3_Runge(10);
```

从图中可以看出，随着节点的加密采用高次插值，虽然插值函数会在更多的点上与所逼近的函数取相同的值，但整体上不一定能改善逼近的效果。事实上，当 n 增大时，所得的插值函数在两端会发生激烈的振荡，这就是 Runge 现象。Runge 现象说明，在大范围内使用高次插值，逼近的效果往往是不理想的。

（2）Lagrange 分段线性插值函数（n=5,10,20）的图像如下：



$L_5(x)$、$L_{10}(x)$、$L_{20}(x)$分别在-4.8,4.8 处的值如下：

```
>> liuziyan_3_3_Piece_Lagrange(-4.8,5)

ans =

    0.0446

>> liuziyan_3_3_Piece_Lagrange(-4.8,10)

ans =

    0.0425

>> liuziyan_3_3_Piece_Lagrange(-4.8,20)

ans =

    0.0419
```

```
>> liuziyan_3_3_Piece_Lagrange(4.8,5)

ans =

    0.0446

>> liuziyan_3_3_Piece_Lagrange(4.8,10)

ans =

    0.0425

>> liuziyan_3_3_Piece_Lagrange(4.8,20)

ans =

    0.0419
```

经过计算，$f(-4.8)=f(4.8)=0.04159734\approx 0.0416$，相对误差：

$$\varepsilon_5=\frac{|0.0446-0.0416|}{0.0416}\approx 0.0721，\quad \varepsilon_{10}=\frac{|0.0425-0.0416|}{0.0416}\approx 0.0216，$$

$$\varepsilon_{20}=\frac{|0.0419-0.0416|}{0.0416}\approx 0.0072$$

由此可见，分段插值函数的分段数越多，插值函数的曲线越趋近平滑，误差越小，精度越高。Lagrange 分段插值函数能够有效避免 Runge 现象造成的计算失真。

四. 实验体会

对实验过程进行总结，分析比较各插值算法的效率和精度差异，指出每种算法的设计要点及应注意的事项，以及自己通过实验所获得的对插值方法的理解。

通过以上三个实验，我有以下几点总结和收获：

1、对于 Lagrange 插值方法：

• 它的计算公式是一个累乘累加的二重算式，结构紧凑，其中各个节点地位对等，形式上很对称，符合数学公式的美学；

• 但是这个插值方法也有缺点，在实际运用时，如果临时需要增添一个节点，则其所有的系数都需要重新计算，造成了计算量上的浪费；

• 在计算精度上，一次（线性）插值即用直线来代替原曲线，往往精度不足，而用更高次数（二次、三次）的简单曲线来代替原曲线，往往能够得到更高精度的结果。

2、对两种逐步插值方法：

• Aitken 和 Neville 逐步差值方法，在设计算法的过程中，我们选择逐列生成的方式，每增加一列，插值节点数减少一个。如果将插值节点数定义为插值问题的规模，那么逐步插值算法的加工过程就是个规模逐次减 1 的规模缩减过程；

• 这两个逐步差值算法中，相同节点数的情况下，Neville 算法的计算结果比 Aitken 更易精确，Neville 算法采用的是上一列相邻两个 y 值之间运算得到新的结果，能更充分运用各节点的值，结果更准确；Neville 逐步插值算法得到的结果值与 Lagrange 三次插值得到的结果相似；

• 与 Lagrange 插值方法对比，Neville 逐步插值比 Lagrange 插值更优越，对 Lagrange 插值来说，临时增加一个节点需要全部重新计算，而 Neville 逐步插值仅需在原计算基础上做些许修改即可，更便于实际运用。

3、对于 Runge 现象和分段插值的运用：

• 对于代数插值，插值多项式的次数随着节点个数的增加而升高，然而高次插值的逼近效果往往是不理想的，次数越高，同时也会带来舍入误差的增大，引起计算失真，这就造成了 Runge 现象。

• 利用分段线性插值，就是将被插值函数逐段多项式（一次）化，对于函数 f(x)，如果在每个子段上用线性插值，即用连接相邻节点的折线逼近所考察的曲线，就能保证一定的逼近效果。

•通过计算结果我们也能得到，Lagrange 分段插值函数能够有效避免 Runge 现象造成的计算失真，并且分段插值函数的分段数 n 越大，插值函数的曲线就越趋近平滑，产生的误差越小。