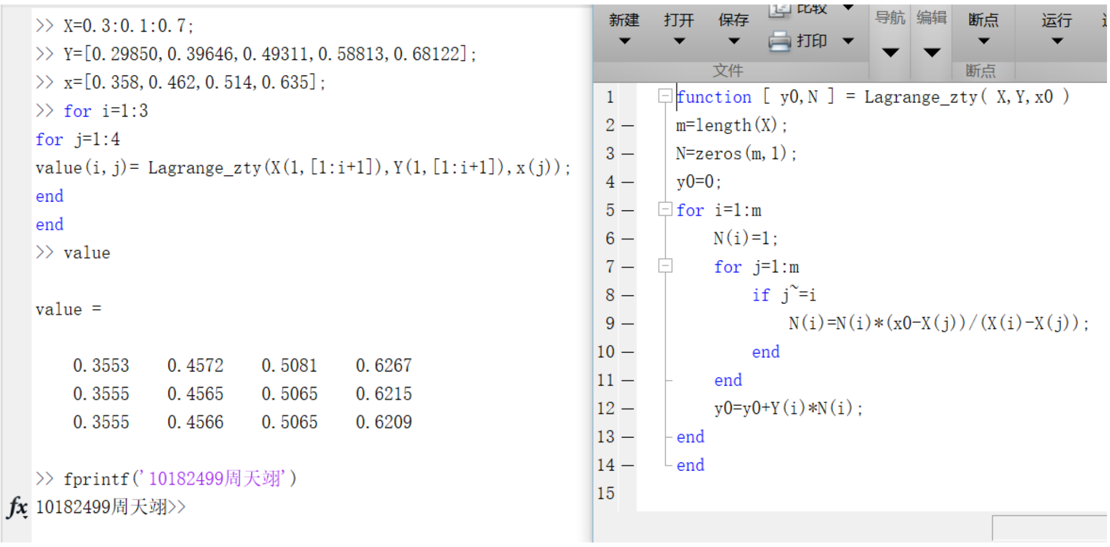
3-1

设原问题中函数自变量矩阵为 X，因变量矩阵为 Y，插值点矩阵为 x，用前 i+1 个插值点作 Lagrange 插值多项式如下图所示，于是矩阵中元素 value（i，j）的值表示 i+1 次 Lagrange 插值函数在插值点 x（j）处的计算值。

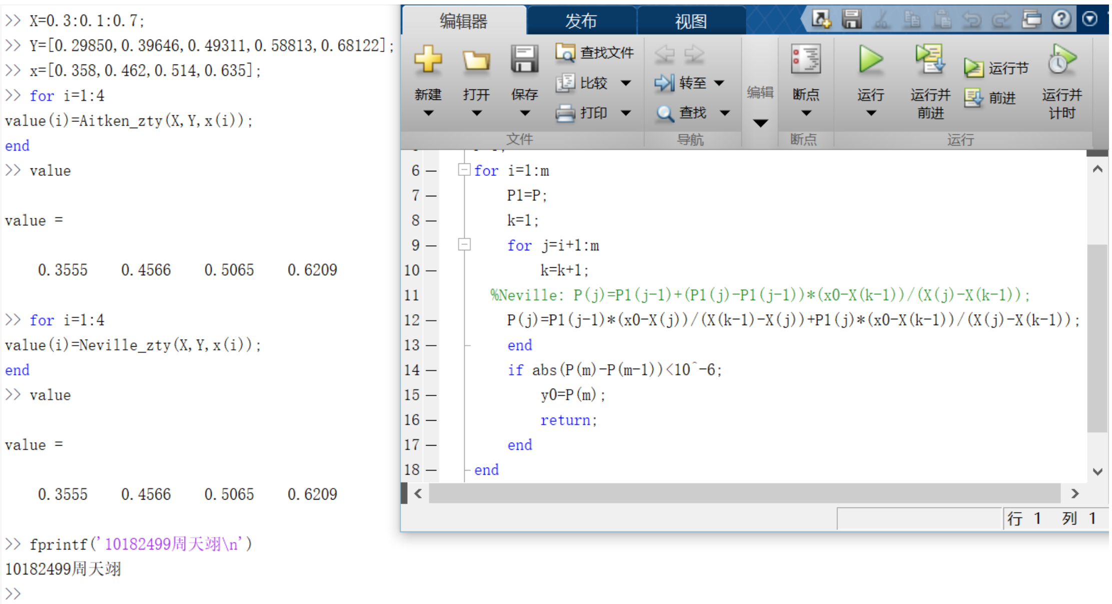


3-2

Neville 算法和 Aitken 算法流程大致相同，只需将课本 Neville 算法中的通项改作：

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

如此即得到 Aitken 算法代码。分别用 Neville 算法和 Aitken 算法计算 3-1 的问题，结果如图，在四位精度下两种算法的结果和 3 次 Lagrange 插值公式一致。



3-3

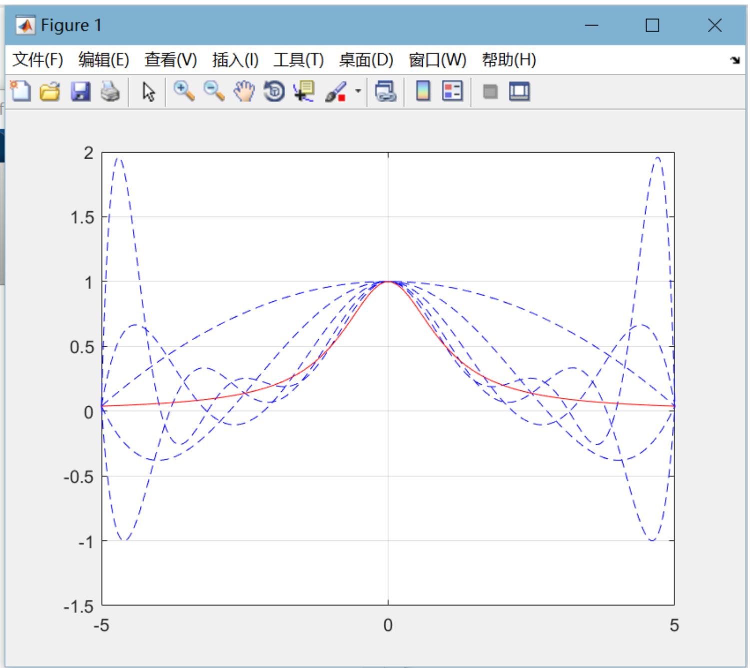
如图，Runge 现象得到验证；对分段区间中的每一段函数线性拟合然后计算，插值函数

值和误差值如图所示。由于函数 $f(x)=\dfrac{1}{1+x^2}$ 是偶函数，所以函数值和误差在 $x_1=-4.8$、$x_2=4.8$ 两处相等；而分段越精细，误差越小，所以第三列的 error_value 误差值最小。

```
>> RungePhenomenon_zty
周天翊
>>
```

```
function RungePhenomenon_zty( )

for i=3:2:11
x=linspace(-5,5,i);
y=1./(1+x.^2);
p=polyfit(x,y,i-1);
xx=-5:0.01:5;
yy=polyval(p,xx);
plot(xx,yy,'--b');
hold on;
grid on;
end;
x=linspace(-5,5);
plot(x,1./(1+x.^2),'r');
fprintf('周天翊\n');
end
```




```
>> N=[5,10,20];
>> x=[-4.8,4.8];
>> [value,error_value]=Problem_3(N,x)

value =

    0.0446    0.0425    0.0419
    0.0446    0.0425    0.0419


error_value =

    0.0030    0.0009    0.0003
    0.0030    0.0009    0.0003

>> fprintf('10182499周天翊\n')
10182499周天翊
>>
```

```
function [ value,error_value ] = Problem_3( N,x )
for i=1:2
    for j=1:3
        X=-5:10/N(j):5;
        Y=1./(X.^2+1);
        value(i,j)=LagrangeLine_zty(X,Y,x(i));
        error_value(i,j)=value(i,j)-1/(x(i)^2+1);
    end
end
```

```
function [ value ] = LagrangeLine_zty( X,Y,x0 )

m=length(X);
for j=1:m-1
    if x0>=X(j) && x0<=X(j+1)
        p=polyfit([X(j),X(j+1)],[Y(j),Y(j+1)],1);
        value=polyval(p,x0);
        return
    end
end
end
```

实验心得体会：

3-1：Lagrange 插值公式形式简洁，逻辑结构简单易于编写程序，在采用高次插值时精度优秀，但在加入新插值点时有计算量浪费的问题；

3-2：Aitken 逐步插值算法可以解决此问题，在相同精度的情况下可以复用之前的计算结果，但逻辑结构比 Lagrange 算法稍难，同时应注意终止条件，即在精度满足条件或所有插值点都已迭代的情况下可以终止计算；

3-3：Runge 现象表明对函数进行多项式拟合时并非次数越高越好，非常有趣。在分段线性拟合时，对函数区间划分越精细得到的精确度越高。