# 云南大学数学与统计学院实验教学中心
# 实验报告

| | | |
|---|---|---|
| **课程名称：** 数学建模实验 | **学期：** 2016~2017 学年下学期 | |
| **指导教师：** 李朝迁 | | |
| **学生：** 刘鹏 20151910042 信计 | **学生：** 王泽坤 20151910011 应数 | **学生：** 段奕臣 20151910002 应数 |
| **实验名称：** **插值与数值积分** & MATLAB 存储方法探究 | | **成绩：** |
| **实验编号：** NO.2 | **实验日期：** 2017 年 3 月 24 日 | **实验学时：** 2 |
| **学院：** 数学与统计学院 | **专业：** 信息与计算科学 | **年级：** 2015 级 |

## 一、实验目的

1. 重点学习 MATALB 的插值函数，参照教科书的分析，自己动手编写拉格朗日插值函数；
2. 学习多种插值方法，然后掌握在 MATLAB 里面调用这些插值方法的函数，理解函数参数的意义。

## 二、实验内容

1.

## 三、实验平台

Windows10 Enterprise 1703 中文版操作系统；
MATLAB R2017a 中文版。

## 四、实验记录与实验结果分析

### 1 题

使用 `help` 学习使用 `polyval` 命令函数

**Solution:**

**polyval**

Polynomial evaluation

**Syntax**

```
y = polyval(p,x)
[y,delta] = polyval(p,x,S)
y = polyval[p,x,[],mu]
[y,delta] = polyval(p,x,S,mu)
```

**Description**

`y = polyval(p,x)` returns the value of a polynomial of degree n evaluated at x. The input argument p is a vector of length n+1 whose elements are the coefficients in descending powers of the polynomial to be evaluated.

*`y = polyval(p,x)`返回 n 次多项式在 x 处的数值。输入参数 p 是一个长度为 n+1 的向量，它的元素是多项式按照降幂顺序排列的项的系数。

$$y = p_1x^n + p_2x^{n-1} + \ldots + p_nx + p_{n+1}$$

x can be a matrix or a vector. In either case, `polyval` evaluates p at each element of x.

*x 可以是一个矩阵也可以是一个向量，无论是哪一种情况，`polyval` 函数都会计算 x 的每一个元素的对应多项式值。

`[y,delta] = polyval(p,x,S)` uses the optional output structure S generated by `polyfit` to generate error estimates `delta`. `delta` is an estimate of the standard deviation of the error in predicting a future observation at x by `p(x)`. If the coefficients in p are least squares estimates computed by `polyfit`, and the errors in the data input to `polyfit` are independent, normal, and have constant variance, then `y±delta` contains at least 50% of the predictions of future observations at x.

* `[y,delta] = polyval(p,x,S)` 利用

polyfit 生成的可选输出结构体 S 来生成误差估计值 delta。delta 是使用 p(x)预测 x 处的未来观测值时的误差标准差的估计值。如果 p 中系数是 polyfit 计算的最小二乘估计，polyfit 数据输入呈独立正态分布，而且拥有常量方差，那么 y±delta 至少包含 x 处 50%的未来观测值。

y = polyval(p,x,[],mu) or [y,delta] = polyval(p,x,S,mu) use $\hat{x}=(x-\mu_1)/\mu_2$ in place of x. In this equation, $\mu_1$=mean($x$) and $\mu_2$=std($x$). The centering and scaling parameters mu = [$\mu_1$,$\mu_2$] are optional output computed by polyfit.

*y = polyval(p,x,[],mu)或[y,delta] = polyval(p,x,S,mu)使用$\hat{x}=(x-\mu_1)/\mu_2$取代 x。在此方程中，$\mu_1$=mean($x$) 且 $\mu_2$=std($x$)。居中和缩放参数 mu = [$\mu_1$,$\mu_2$] 是由 polyfit 计算的可选输出。

**程序代码：**

```
%   filename: myPolyval
%   Tseting function Polyval()
fprintf('y = 9 * x^5 + 4 * x^2 + 7 * x + 9\n');
p = [9 0 0 4 7 9];
myInput = input('Please input a matrix:\n');
polyval(p,myInput)
```

**程序代码 1**

**运行结果：**

```
命令行窗口
>> myPolyval
y = 9 * x^5 + 4 * x^2 + 7 * x + 9
Please input a matrix:
[1 1 2 3 5 8]

ans =

          29          29         327        2253       28269      295233
```

**运行结果 1**

**代码分析：**

调用函数。

**2 题**

编写程序计算：$\sum_{i=1}^{n} 0.1$，其中$n = 10$，100，1000，要求使用 `for` 循环实现，结果显示用 `format long`。

**程序代码：**

```
%   filename: Sum_Print
%   calculate the sum of QUANTITY NUM
function y = Sum_Print(num,quantity)
    sum_before = num;
    for i = 1: quantity
        sum_before = [sum_before,num];
    end
    y = sum(sum_before);
end
```

程序代码 2

**运行结果：**

```
命令行窗口
>> format LONG
>> Sum_Print(0.1,100)

ans =

   10.099999999999980
```

运行结果 2

**代码分析：**

编写简单的 for 循环。

**3 题**

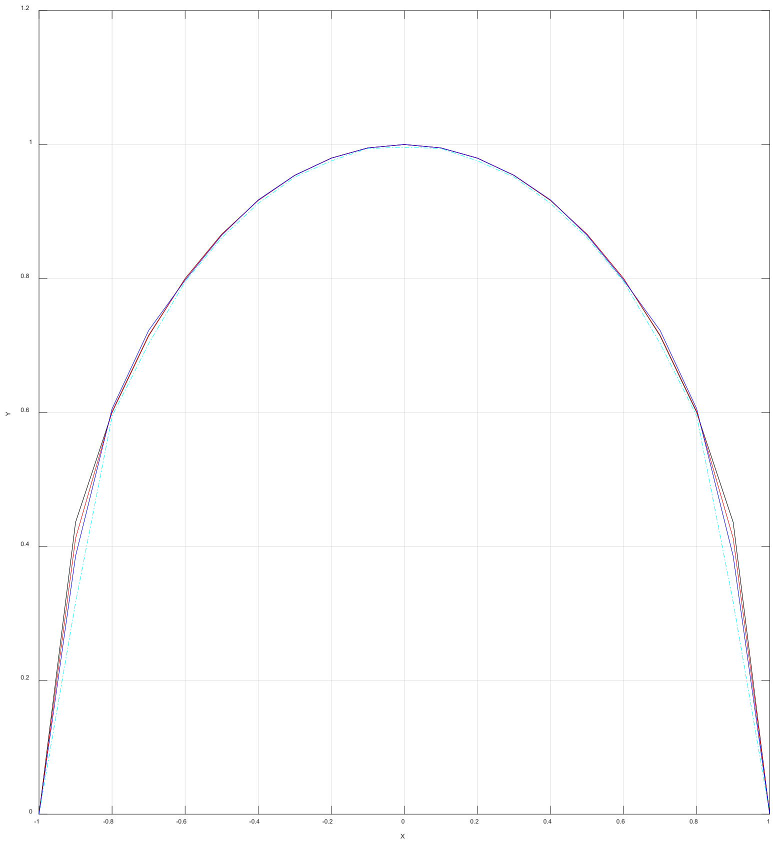
选择一些函数，在$n$个节点上（$n$不要太大，如 5~11），用拉格朗日、分段线性、三次样条三种插值方法，计算$m$个插值点的函数值，（$m$要适中，如 50~100）。通过数值和图形输出，将三种插值结果与精确值进行比较。适当增加$n$，再作比较，由此初步分析。下列函数供选择参考。

（1）$y=\sin(x)$, $0\le x\le 2\pi$ （2）$y=(1-x^2)^{1/2}$
（3）$y=\cos^{10}(x)$, $-2\le x\le 2$ （4）$y=\exp(-x^2)$, $-2\le x\le 2$

**程序代码：**

```
%   filename: Compare_three_interpolations
%% 这是对函数（2）的三种插值应用
x_1_real = -1 : 0.1 : 1;
y_1_real = (1 - x_1_real.^2).^(1/2);

y_1_lagr = lagr(x_1,y_1,x_1_real);
y_1_interp1 = interp1(x_1,y_1,x_1_real);
y_1_spline = spline(x_1,y_1,x_1_real);

%% 作图比较
figure;
plot(x_1_real,y_1_real,'k-');
text(3,2,'real curve')
hold on
grid on
plot(x_1_real,y_1_lagr,'r-')
plot(x_1_real,y_1_interp1,'c-.');
plot(x_1_real,y_1_spline,'b-')
xlabel('X');
ylabel('Y');
hold off
```

**程序代码 3**




**运行结果 3**

## 代码分析：

调用自己已经编写过的函数。

**4 题**

对 Lagrange 插值方法的程序（P49）进行注释（%），并对

$$g(x)=\frac{1}{1+x^2}, x\in[-5,5]$$

区间的已知点$x=-5:5$进行插值，并画出图像。

**程序代码：**

```
%   filename: lagr
%   lagrange interpolation algorithm
function y = lagr(x_known,y_known,myInput)
    %   x_known, y known, the collection of the points already known
    %   myInput, the variable I need to predict y
    %   y, a vector of the prediction based on muInput
    n = length(x_known);    %   how many of x already known
    len = length(myInput);  %   len, number of the points to calculate
    for i = 1 : len         %   loop i, calculate once in one circle
        z = myInput(i);     %   get one independent variable, naming z
        s = 0;              %   initial value of the prediction
        for k = 1 : n       %   loop k, sum
            p = 1;          %   initial value of l(x)
            for j = 1 : n   %   loop j, continuous multiple
                if j ~= k
                    p = p * (z - x_known(j)) / (x_known(k) - x_known(j));
                end
            end
            s = s + p * y_known(k);
        end
        y(i) = s;
    end
end
%   This function is very easy, just a rewriting of the funtions in book
```

**程序代码 4**

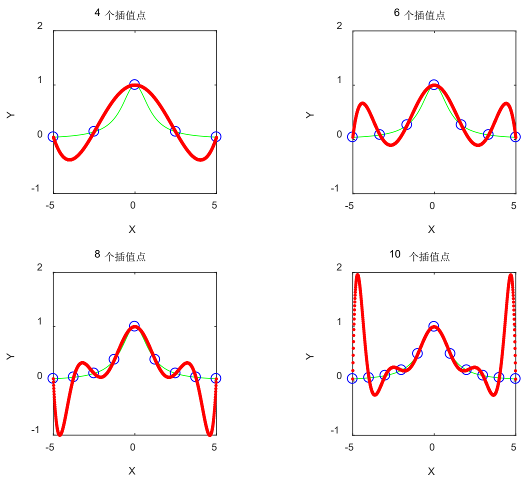
```
%   filename: Test_Lagrange
%   测试拉格朗日插值算法，对比插值产生的图像与实际图像的差别

%% 设置插值的数目为 4，进行计算与画图
x = -5 : 10/4 : 5;                  % 生成 4 组已知点
y = 1 ./ (1 + x.^2);

x_real = -5 : 0.01 : 5;             % 这是比较接近实际的稠密的矩阵
y_real = 1 ./ (1 + x_real.^2);

myInput = x_real;                   % 赋值，利于代码阅读
myOutput = lagr(x,y,x_real);        % 根据算法得出的插值

subplot(2,2,1);
    plot(x_real,y_real,'g -');
```

```
    hold on;
    plot(x,y,'b o');
    plot(myInput,myOutput,'r .');
    xlabel('X');
    ylabel('Y');
    axis([-5 5 -1 2])
    axis('square');
    title('4 个插值点')

%% 设置插值的数目为 6，再次进行计算与画图
x = -5 : 10/6 : 5;                % 生成 6 组已知点
y = 1 ./ (1 + x.^2);

x_real = -5 : 0.01 : 5;            % 这是比较接近实际的稠密的矩阵
y_real = 1 ./ (1 + x_real.^2);

myInput = x_real;                 % 赋值，利于代码阅读
myOutput = lagr(x,y,x_real);      % 根据算法得出的插值

subplot(2,2,2);
    plot(x_real,y_real,'g -');
    hold on;
    plot(x,y,'b o');
    plot(myInput,myOutput,'r .');
    xlabel('X');
    ylabel('Y');
    axis([-5 5 -1 2])
    axis('square');
    title('6 个插值点')

%% 设置插值的数目为 8，再次进行计算与画图
x = -5 : 10/8 : 5;                % 生成 10 组已知点
y = 1 ./ (1 + x.^2);

x_real = -5 : 0.01 : 5;            % 这是比较接近实际的稠密的矩阵
y_real = 1 ./ (1 + x_real.^2);

myInput = x_real;                 % 赋值，利于代码阅读
myOutput = lagr(x,y,x_real);      % 根据算法得出的插值

subplot(2,2,3);
    plot(x_real,y_real,'g -');
    hold on;
    plot(x,y,'b o');
    plot(myInput,myOutput,'r .');
    xlabel('X');
    ylabel('Y');
    axis([-5 5 -1 2])
    axis('square');
```

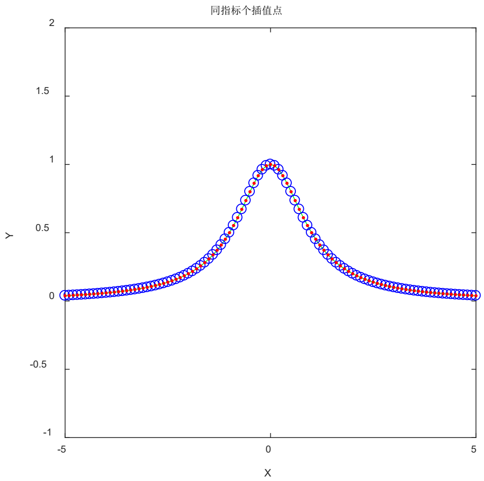
```
    title('8 个插值点')

%% 设置插值的数目为 10，再次进行计算与画图
x = -5 : 5;                          % 生成 10 组已知点
y = 1 ./ (1 + x.^2);

x_real = -5 : 0.01 : 5;               % 这是比较接近实际的稠密的矩阵
y_real = 1 ./ (1 + x_real.^2);

myInput = x_real;                    % 赋值，利于代码阅读
myOutput = lagr(x,y,x_real);          % 根据算法得出的插值

subplot(2,2,4);
    plot(x_real,y_real,'g -');
    hold on;
    plot(x,y,'b o');
    plot(myInput,myOutput,'r .');
    xlabel('X');
    ylabel('Y');
    axis([-5 5 -1 2])
    axis('square');
    title('10 个插值点')

%% 设置插值的数目与实际已知点数目相同，再次进行计算与画图
x = -5 : 0.1 : 5;                    % 生成 100 组已知点
y = 1 ./ (1 + x.^2);

x_real = -5 : 0.1 : 5;               % 这是比较接近实际的稠密的矩阵
y_real = 1 ./ (1 + x_real.^2);

myInput = x_real;                    % 赋值，利于代码阅读
myOutput = lagr(x,y,x_real);          % 根据算法得出的插值

figure
    plot(x_real,y_real,'g -');
    hold on;
    plot(x,y,'b o');
    plot(myInput,myOutput,'r .');
    xlabel('X');
    ylabel('Y');
    axis([-5 5 -1 2])
    axis('square');
    title('同指标个插值点')
```

**程序代码 5**

**运行结果：**




**运行结果 4**




**运行结果 5**

**代码分析：**

$$L_n(x)=y_j,\ j=0,1,\cdots,n.$$

$$l_i(x)=\frac{x-x_0}{x_i-x_0}\cdot\cdots\cdot\frac{x-x_{i-1}}{x_i-x_{i-1}}\cdot\frac{x-x_{i+1}}{x_i-x_{i+1}}\cdot\cdots\cdot\frac{x-x_n}{x_i-x_n}$$

$$L_n(x)=\sum_{i=0}^{n}y_i\ l_i(x)$$

可以看出，算法已经在书中给出了，而 `lagr()`函数的主要任务就是带参数地实现这个算法，得到目的插值点的数值解。由于暂时没有办法对字符串进行下推自动机解读或者输出，所以这里只能根据插值的这一系列目的自变量值，输出数值解。其实更理想的做法是输出一个字符串，该字符串是一个整系数多项式。

我们可以看到运行结果 5，我将插值点与已知坐标设置为一样的，结果惊奇地发现所有的插值点都在实际曲线上。这个结果可以从高等代数的知识中得到解答，即拟合的多项式曲线与原曲线有$n$个交点，交点数目与已知点数目相同。所以这个结果是显然的。

**5 题**

用梯形公式和辛普森公式计算由表 3.6 数据给出的积分$\int_{0.3}^{1.5} y(x)\mathrm{d}x$。

**表 3.6 数值积分数据**

| $k$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $x_k$ | 0.3 | 0.5 | 0.7 | 0.9 | 1.1 | 1.3 | 1.5 |
| $y_k$ | 0.3985 | 0.6598 | 0.9147 | 1.1611 | 1.3971 | 1.6212 | 1.8325 |

已知该表数据为函数$y = x + \sin(\frac{x}{3})$所产生，将计算值与精确值作比较。

**程序代码：**

```
%   filename: myIntegrate_1
%   Page 64
%% 数据输入
x = [0.3 : 0.2 : 1.5];
y = [0.3895 0.6598 0.9147 1.1611 1.3971 1.6212 1.8325];
%% 经过手工求定积分公式的精确值
Value_real = 1/2 * (1.5^2 - 0.3^2) + 1/3 * ( cos(0.3) - cos(1.5) );
%% 梯形公式
Value_trapz = trapz(x,y);
%% 辛普森公式
len = length(y);
point_middle = [ y(2:2:len-1) ];
sum_point_middle = sum(point_middle);

point_double_edge = [ y(3:2:len-1) ];
sum_point_double_edge = sum(point_double_edge);

Value_Simpson = ...
    ( y(1)+y(len)...
        + 4 * sum_point_middle ...
        + 2 * sum_point_double_edge ) * 0.2 / 3;
%% 自适应辛普森公式
Value_Simpson_Pro = quad('x+sin(x)/3',0.3,1.5);

%% 不同类型的数值积分对比
fprintf('Value_real - Value_trapz = %1.10f\n',...
    Value_real - Value_trapz);
fprintf('Value_real - Value_Simpson = %1.10f\n',...
    Value_real - Value_Simpson);
fprintf('Value_real - Value_Simpson_Pro = %1.10f\n',...
    Value_real - Value_Simpson_Pro);
```

**程序代码 6**

**运行结果：**

```
命令行窗口
>> myIntegrate_1
Value_real - Value_trapz = 0.0018864292
Value_real - Value_Simpson = 0.0005997625
Value_real - Value_Simpson_Pro = 0.0000000021
```

**运行结果 6**

**代码分析：**

为了提高精度采用分段二次差值函数代替分段线性函数$f(x)$。因为每一段都要用到相邻的两个小区间，所以小区间的数目必须是偶数。我们根据课本上的零开头方法进行分析。假设区间左端点是第0个端点，每两个区间视为一组，且开头组是第0组。那么我们会发现，第$k$个区间组包含三个端点，而且分别是$2k$，$2k+1$，$2k+2$，，根据书中所给的插值积分公式，知道在这个区间组上面积分所得为

$$\int_{2k}^{2k+2} S_k(x)\mathrm{d}x = \frac{h}{3}(f_{2k}+4f_{2k+1}+f_{2k+2}).$$

抛开符号看函数，可以知道这个最终的数值积分就是：所有区间组的中点对应函数值的 4 倍与非首、尾端点的 2 倍（因为前后加了两次），最后加上首尾端点对应的函数值；该数值乘以单个区间长度的三分之一。

经过这一番分析，我们知道了具体在 MATLAB 里面如何写代码。毕竟 MATLAB 是从 1 开始的，单纯从公式去写代码可能会乱。

为了更加细致观察插值积分的结果，我们把数值积分结果与实际结果做差，并且保留 10 位小数，发现自适应的辛普森方法结果是最精确的。

**6 题**

选择一些函数用梯形、辛普森和 Gauss-Lobatto 三种方法计算积分。改变步长（对梯形），改变精度要求（对辛普森和 Gauss-Lobatto），进行比较、分析。如下函数供选择参考：

（1）$y=\frac{1}{x+1}, 0\le x\le 1$　　　　（2）$y=e^{3x}\sin(2x)$，$0\le x\le 2$

（3）$y=\sqrt{1+x^2}$，$0\le x\le 2$　　　　（4）$y=\frac{1}{\sqrt{(2\pi)}}e^{-\frac{x^2}{2}}, -2\ \le x\ \le 2$

**程序代码：**

```
%   filename: Compare_integral
%   compare the influence of STEP and TOL
x = 0 : 0.01 : 1;
y = 1 ./ (x + 1);
Value_real = log(2);
Value_trapz = trapz(x,y);
Value_Simpson_Pro =  quad('1./(x+1)',0,1);
%   Gauss-Lobatto(GL)
Value_GL = quadl('1./(x+1)',0,1);
%%  不同类型的数值积分对比
fprintf('Value_real - Value_trapz = %1.10f\n',...
    abs(Value_real - Value_trapz));
fprintf('Value_real - Value_Simpson_Pro = %1.10f\n',...
    abs(Value_real - Value_Simpson_Pro));
fprintf('Value_real - Value_GL = %1.10f\n',...
    abs(Value_real - Value_GL));
fprintf('------------------------------------------\n');
%%  New
x = 0 : 0.8 : 1;
y = 1 ./ (x + 1);
Value_real = log(2);
Value_trapz_wide = trapz(x,y);
Value_Simpson_Pro_wide =  quad('1./(x+1)',0,1,1e-12);
%   Gauss-Lobatto(GL)
Value_GL_wide = quadl('1./(x+1)',0,1,1e-12);
%%  不同类型的数值积分对比
fprintf('Value_real - Value_trapz = %1.10f\n',...
    abs(Value_real - Value_trapz_wide));
fprintf('Value_real - Value_Simpson_Pro = %1.10f\n',...
    abs(Value_real - Value_Simpson_Pro_wide));
fprintf('Value_real - Value_GL = %1.10f\n',...
    abs(Value_real - Value_GL_wide));
```

程序代码 7

**运行结果：**

```
命令行窗口
>> Compare_integral
Value_real - Value_trapz = 0.0000062499
Value_real - Value_Simpson_Pro = 0.0000000193
Value_real - Value_GL = 0.0000000056
------------------------------------------
Value_real - Value_trapz = 0.0709249583
Value_real - Value_Simpson_Pro = 0.0000000000
Value_real - Value_GL = 0.0000000000
```

运行结果 7

**代码分析：**

比较不同算法的精度。

**7 题**

表 3.7 给出的$x$，$y$数据位于机翼剖面的轮廓线上，$y_1$和$y_2$分别对应轮廓的上下线。假设需要得到$x$坐标每改变0.1 时的$y$坐标。试完成加工所需数据，画出曲线，求机翼剖面的面积。

| $x$ | 0 | 3 | 5 | 7 | 9 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|
| $y_1$ | 0 | 1.8 | 2.2 | 2.7 | 3.0 | 3.1 | 2.9 | 2.5 | 2.0 | 1.6 |
| $y_2$ | 0 | 1.2 | 1.7 | 2.0 | 2.1 | 2.0 | 1.8 | 1.2 | 1.0 | 1.6 |

**表 3.6 机翼剖面轮廓线数据**

**程序代码：**

```
%% 实际数据录入
x = [0 3 5 7 9 11 12 13 14 15];
y_upside = [0 1.8 2.2 2.7 3.0 3.1 2.9 2.5 2.0 1.6];
y_downside = [0 1.2 1.7 2.0 2.1 2.0 1.8 1.2 1.0 1.6];

%% 目标插值点
x_real = [0 : 0.1 : 15];

y_upside_spline = spline(x,y_upside,x_real);
y_dowmside_spline = spline(x,y_downside,x_real);

y_upside_interp1 = interp1(x,y_upside,x_real);
y_dowmside_interp1 = interp1(x,y_downside,x_real);

%% 三次样条插值与梯形积分
area_big_trapz = trapz(x_real,y_upside_spline);
area_small_trapz = trapz(x_real,y_dowmside_spline);
fprintf('area_real_trapz_spline = %2.20f\n',...
    area_big_trapz - area_small_trapz);

%% 三次样条插值与辛普森积分
% 大面积
len = length(y_upside_spline);

point_middle = [ y_upside_spline(2:2:len-1) ];
sum_point_middle = sum(point_middle);

point_double_edge = [ y_upside_spline(3:2:len-1) ];
sum_point_double_edge = sum(point_double_edge);

area_Simpson_big = ...
    ( y_upside_spline(1) + y_upside_spline(len) ...
     + 4 * sum_point_middle ...
     + 2 * sum_point_double_edge ) * 0.1 / 3;
% 小面积
len = length(y_dowmside_spline);

point_middle = [ y_dowmside_spline(2:2:len-1) ];
sum_point_middle = sum(point_middle);

```

```
point_double_edge = [ y_dowmside_spline(3:2:len-1) ];
sum_point_double_edge = sum(point_double_edge);

area_Simpson_small = ...
    ( y_dowmside_spline(1) + y_dowmside_spline(len) ...
    + 4 * sum_point_middle ...
    + 2 * sum_point_double_edge ) * 0.1 / 3;
% 实际面积
fprintf('area_real_Simpson_spline = %2.20f\n',...
    area_Simpson_big - area_Simpson_small);

%% 分段线性插值与梯形积分
y_upside_interp1 = interp1(x,y_upside,x_real);
y_dowmside_interp1 = interp1(x,y_downside,x_real);

area_big_trapz = trapz(x_real,y_upside_interp1);
area_small_trapz = trapz(x_real,y_dowmside_interp1);
fprintf('area_real_trapz_interp1 = %2.20f\n',...
    area_big_trapz - area_small_trapz);

%% 分段线性插值与辛普森积分
% 大面积
len = length(y_upside_interp1);

point_middle = [ y_upside_interp1(2:2:len-1) ];
sum_point_middle = sum(point_middle);

point_double_edge = [ y_upside_interp1(3:2:len-1) ];
sum_point_double_edge = sum(point_double_edge);

area_Simpson_big = ...
    ( y_upside_interp1(1) + y_upside_interp1(len) ...
    + 4 * sum_point_middle ...
    + 2 * sum_point_double_edge ) * 0.1 / 3;
% 小面积
len = length(y_dowmside_interp1);

point_middle = [ y_dowmside_interp1(2:2:len-1) ];
sum_point_middle = sum(point_middle);

point_double_edge = [ y_dowmside_interp1(3:2:len-1) ];
sum_point_double_edge = sum(point_double_edge);

area_Simpson_small = ...
    ( y_dowmside_interp1(1) + y_dowmside_interp1(len) ...
    + 4 * sum_point_middle ...
    + 2 * sum_point_double_edge ) * 0.1 / 3;
% 实际面积
fprintf('area_real_Simpson_interp1 = %2.20f\n',...
```

```
    area_Simpson_big - area_Simpson_small);

%% 作图
subplot(2,1,1);
plot(x_real,y_upside_spline,'k-');
hold on;
plot(x_real,y_dowmside_spline,'k-');

plot(x_real,y_upside_interp1,'r-');
hold on;
plot(x_real,y_dowmside_interp1,'r-');
xlabel('X');
ylabel('Y');
text(6.5,1.3,'red:interp1');
text(6.5,1,'black:spline');
hold off;
```

**程序代码 8**

**运行结果：**

命令行窗口

```
>> Plane_Wing
area_real_trapz_spline = 11.3443853129333160000
area_real_Simpson_spline = 11.3460158515063830000
area_real_trapz_interp1 = 10.7500000000000070000
area_real_Simpson_interp1 = 10.7499999999999930000
```

**运行结果 8**

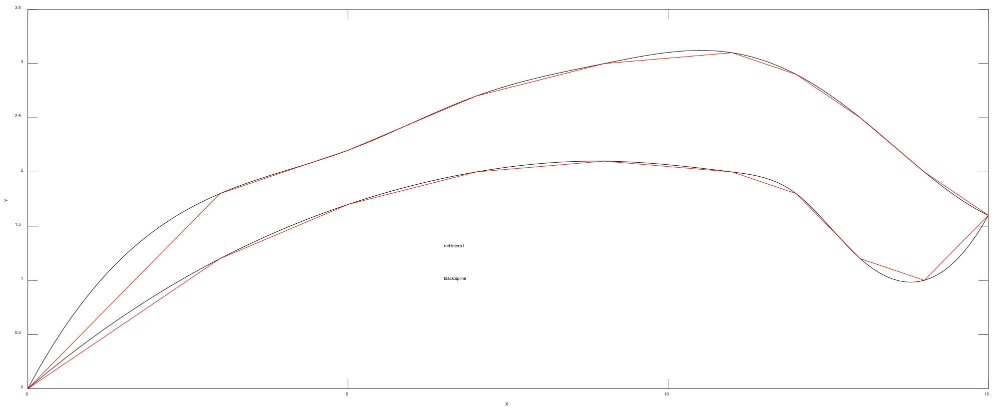


**Figure 1**

从运行结果可以看出，黑色曲线，即三次样条插值得出的曲线，比较光滑，与实际体验相吻合。而分段线性插值就不太好了。两种插值方法，两种积分方法，唯三次样条与辛普森公式法比较精确，因为已知数据比较少。

**代码分析：**

表中给出了一些轮廓曲线的样本点，那么我们考虑先通过插值函数，进行精细刻画，得到插值函数之后，利用积分公式进行积分。

**8 题**

辛普森公式的 MATLAB 函数构建。

**程序代码：**

```
function y = simp(x_input,y_input)
%   Simpson method for integration
    len = length(y_input);
    point_middle = [ y_input( 2 : 2 : len-1) ];
    sum_point_middle = sum(point_middle);

    point_double_edge = [ y_input( 3 : 2 : len-1) ];
    sum_point_double_edge = sum(point_double_edge);

    Value_Simpson = ...
        ( y_input(1) + y_input(len)...
            + 4 * sum_point_middle ...
            + 2 * sum_point_double_edge ) * (x_input(2) - x_input(1)) / 3;
    y = Value_Simpson;
end
```

注意：给出的 x_input 必须是等距的。

**代码分析：**

编写辛普森函数，比较简单，关键是理解数学原理。

## 五、实验体会

通过对 MATLAB 的交互式操作与程序设计，掌握了对于输入输出函数的应用。

其次，还有对于计算机数值计算的内在数值存储形式，因为很多有限小数不能用有限位的二进制表示，所以采取了有限位的存储方式之后，计算机计算某些数值就会出现误差。但是 MATLAB 是如何避免这种误差的我目前还想不明白，因为我在实验中，用了乘法和循环加法，按照乘法就是循环相加的实质，两者的结果应该是一样的，但是实际并不是这样。可能 MATLAB 采用了一种算法，避免了这种可怕的事情发生。但是网络上没有找到相应案例。

最后，按照课堂讲授的拉格朗日常插值算法，编写 MATLAB 程序进行计算插值。`Lagr()`函数，无非就是对算法的语言转化。

值得说明的一点是，自适应的辛普森公式，在尚未注意到题目 4 已经给出公式的条件下，我自己写了一个集合对应的向量函数，命名为 `Dict()`，然而我用这个函数句柄进行操作却提示矩阵维数超界。应该是连续的函数才有可能被系统调用，因为连续函数基本上是初等函数复合或者复杂而来，进行符号运算是简单的。

## 六、参考文献

[1] 大学数学实验/姜启源，谢金星，邢文训，张立平，北京：清华大学出版社，2010.12
[2] MATLAB 教程/张志涌，杨祖樱，北京：北京航空航天大学出版社，2015.1

## 七、教师评语