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 |

# 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* = *p1xn* + *p2xn-1* + … + *pnx* + *pn*+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 =(*x*−*μ1*)/*μ2* in place of x. In this equation, *μ*1=mean(*x*) and *μ*2=std(*x*). The centering and scaling parameters mu = [*μ*1,*μ*2] are optional output computed by polyfit.

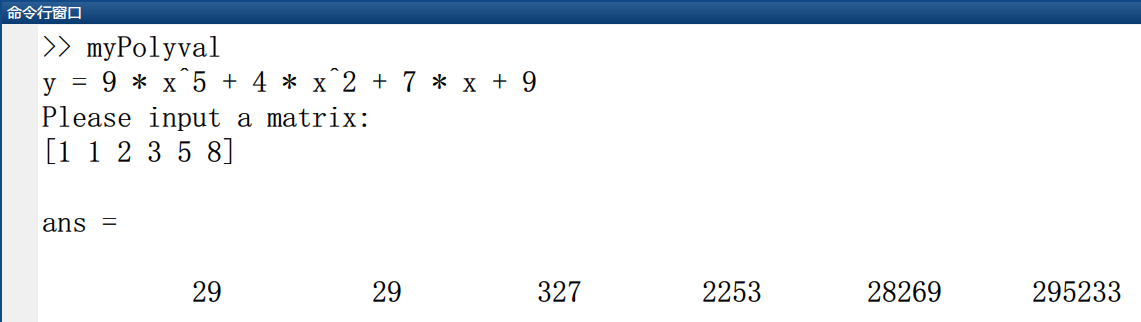
＊y = polyval(p,x,[],mu)或[y,delta] = polyval(p,x,S,mu)使用=(*x*−*μ1*)/*μ2*取代x。在此方程中，*μ*1=mean(*x*)且*μ*2=std(*x*)。居中和缩放参数mu = [*μ*1,*μ*2]是由polyfit计算的可选输出。

程序代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | % 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：



运行结果 1

代码分析：

调用函数。

2题

编写程序计算：，其中，要求使用for循环实现，结果显示用

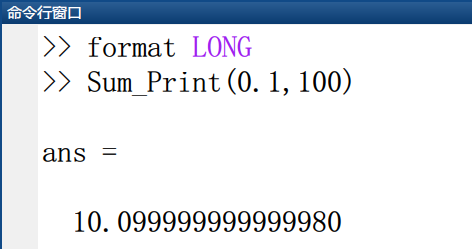
format long。

程序代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | % 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：



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，在个节点上（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，计算个插值点的函数值，（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，再作比较，由此初步分析。下列函数供选择参考。

（1） （2）

（3） （4）

程序代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | % 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



区间的已知点进行插值，并画出图像。

程序代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | % 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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107 | % 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：







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。

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

**表 3.6 数值积分数据**

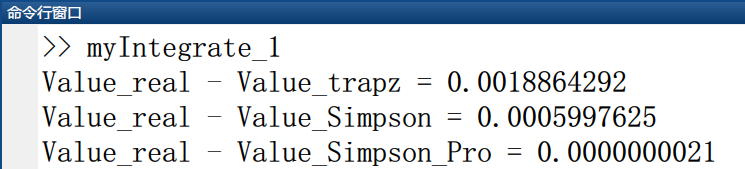
已知该表数据为函数所产生，将计算值与精确值作比较。

程序代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | % 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：



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。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，第个区间组包含三个端点，而且分别是，，，，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

.

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）； （2）；

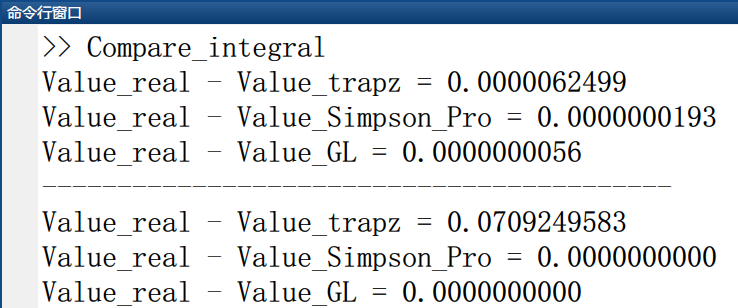
（3）； （4）。

程序代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | % 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：



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，数据位于机翼剖面的轮廓线上，和分别对应轮廓的上下线。假设需要得到坐标每改变时的坐标。试完成加工所需数据，画出曲线，求机翼剖面的面积。

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

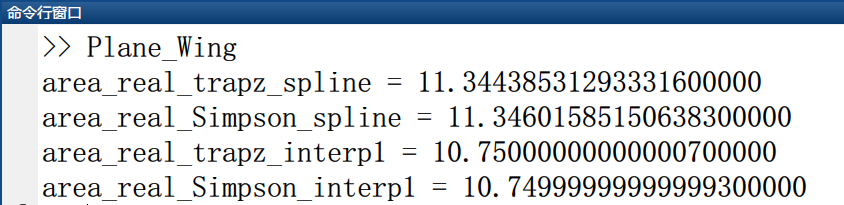
**表 3.6 机翼剖面轮廓线数据**

程序代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105 | %% 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：



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：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **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