数值分析上机实验

Numerical Analysis

20220001 一班 李灵

2022.12

目录

1 实验一误差分析

1.1 病态问题

1.1.1 实验目的

病态问题是指对一个数值问题本身而言，如果输入数据存在微小扰动，那么输出数据的相对误差较大的现象。一般而言，对误差敏感的问题属于坏问题，也即病态问题。对于病态问题的理解与处理是数值分析过程中的重要一环 [3]。

由于存在病态误差是坏问题本身的固有数值属性，通常意义上的误差减小方式对其不适用。以求解线性方程组为例，当系数行列式为零的解附近存在微小扰动时，会对解产生很大的误差，因此需要利用特殊算法解决此类问题。病态问题的求解往往伴随着复杂的算法和多次的迭代过程，因而往往也会伴随着更多的计算资源消耗 [4]。

1.1.2 问题重述

利用 $p(x)$ 探究高次的代数多项式的解对微小扰动的敏感性。

$$p(x) = (x-1)(x-2)...(x-20) = \prod_{k=1}^{20}(x-k) \tag{1}$$

1.1.3 方案设计

考虑对多项式 $p(x)$ 添加一个微小扰动 ϵ:

$$p(x) + \epsilon x^{19} = 0 \tag{2}$$

由于多项式 $p(x)$ 共有 20 个单重根，因此 ϵ 相当于将扰动作用在 x^{19} 的系数上。通过比较1和2二者求出的根，可以得到微小扰动对1的解造成的误差，进一步能够分析该多项式的解对微小扰动的敏感性。

1.1.4 结果分析

（1）利用 MATLAB 程序构造 $p(x)$，选取一个足够小的数 $ess = \epsilon$。考虑计算机的计算精度，取 $ess = 0.000000001$，并将其作用于 x^{19}，添加扰动后的多项式解和原多项式解如图1所示。

可以看到，添加微小扰动后的多项式在解大于 10 时出现了明显的误差，这说明它们对于该种扰动较为敏感。

（2）分别取 x^1、x^{10}、x^{18} 和 x^{20} 为扰动项，重复上述实验。结果如图2所示。

可以看到，在 x^{18} 和 x^{20} 处添加扰动造成的误差较大，而在 x^1 和 x^{10} 处的扰动几乎没有造成误差。

（3）考虑从理论上解释产生（2）中现象的原因。将方程2写成完全展开形式

$$p(x, \alpha) = x^{20} - \alpha x^{19} + ... = 0 \tag{3}$$

显然，如果方程的根存在扰动，那么同样的误差作用的幂次越高，产生的误差就越大。

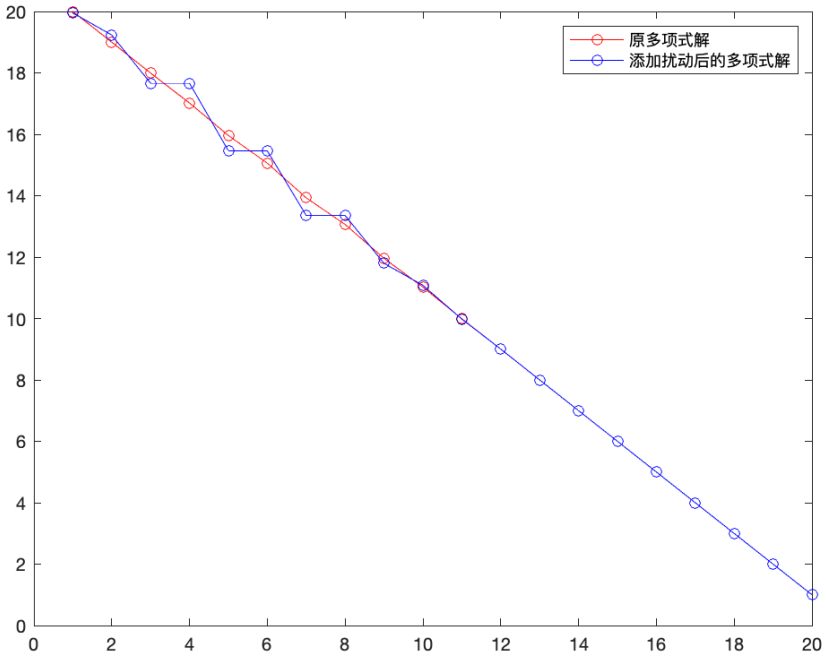
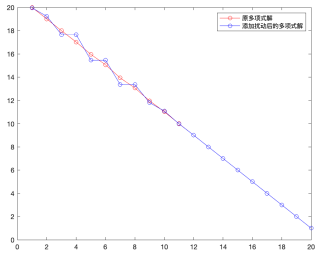
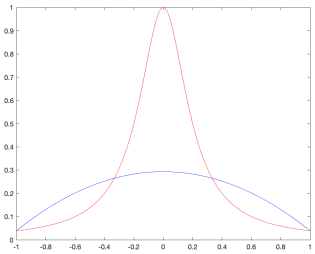


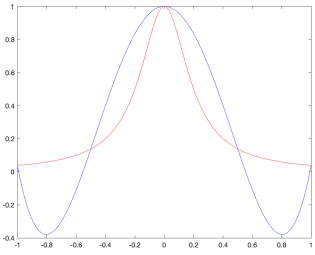
图 1: 原多项式解和添加扰动后的多项式解



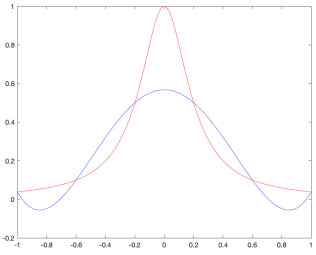
(a) x^1



(b) x^{10}



(c) x^{18}



(d) x^{20}

图 2: 修改扰动项后的多项式解和原多项式解

将方程的解 x 看成是系数 α 的函数，求其关于 α 的导数，由于 $p(x,\alpha)$ 是关于 α 的一次函数，得

$$\frac{\mathrm{d}p(x,\alpha)}{\mathrm{d}\alpha} = -x^{19} \tag{4}$$

* 由于 $p(x,\alpha)$ 关于 α 的导数为恒负值，且 x 值越大下降越快，即 $p(x,\alpha)$ 随 x 的增大对 α 的变化更敏感。

1.1.5 思考题

（1）改进实验

由于 MATLAB 程序中 roots 函数求解多项式方程的精度不高，因此考虑使用 solve 函数以提升计算精度。

重复上述实验（1），得到结果如图4。

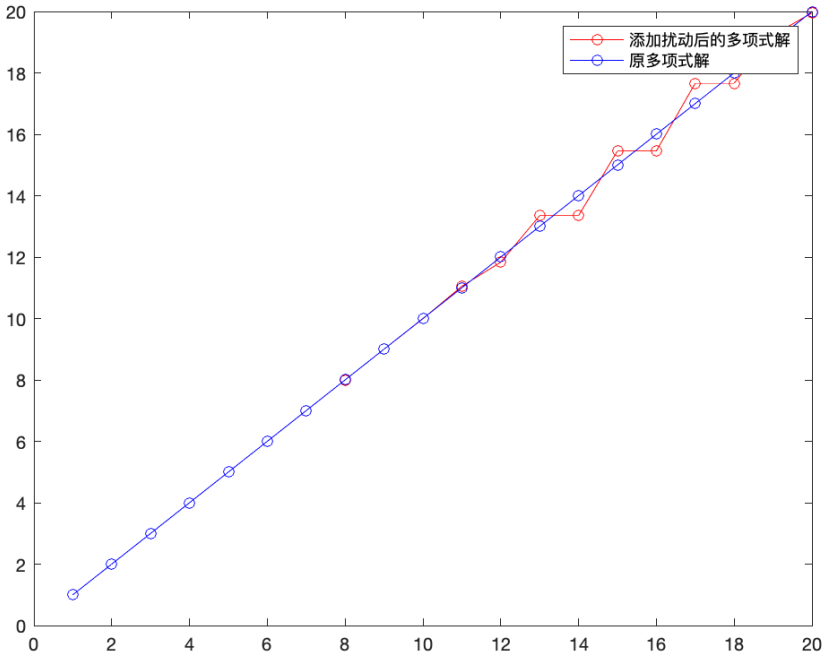


图 3: 更换函数后原多项式解和添加扰动后的多项式解

（2）二进制产生的误差

利用 MATLAB 计算 $\sum_{i=1}^{1000} -100$，结果为 $-1.4069e-12 \neq 0$，这说明由于计算机内部依赖二进制进行存储和计算，因此会存在舍入误差。

（3）简单公式产生的巨大舍入误差

利用式5计算自然对数的底数，此时计算 e 值的精度是随着 n 而改变的。

$$e = e^1 = \lim_{x \to +\infty} (1 + \frac{1}{n})^n \tag{5}$$

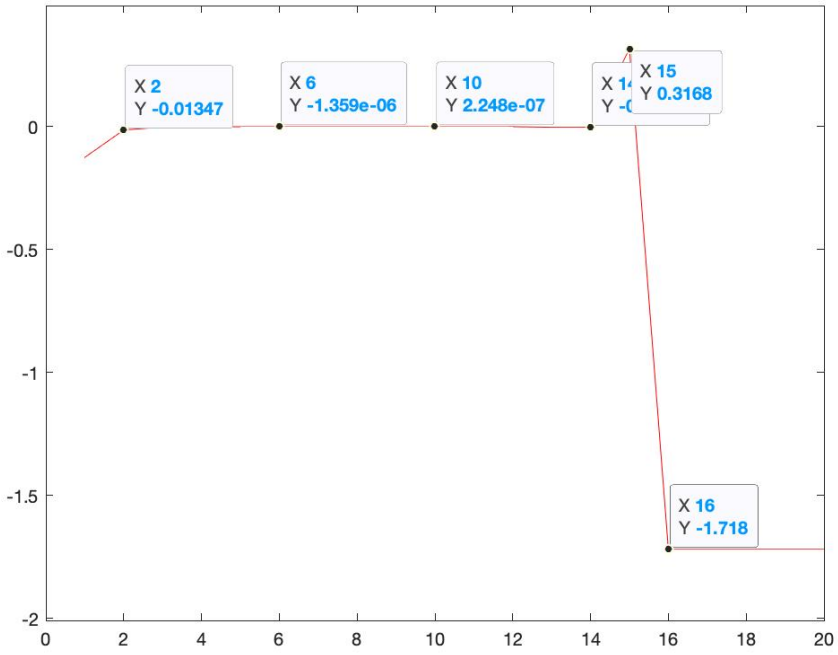


图 4: e 值的精度变化

结果如图5所示，在横坐标（以 10 为底的指数）大于等于 16 时误差的绝对值最大。这是由于 Matlab 中存储的格式为 double，做除法运算后的幂可能出现栈溢出现象，故从 10^{15} 开始会产生较大误差，并在 10^{16} 时达到极大值并保持不变。

1.1.6 程序代码

(1) “test1_1_1.m”

```
clear;                                  % clear the residue

ess = 0.000000001;                      % set the error as ess = 0.000000001
ve = zeros(1, 21);                      % create a 1x21 zero matrix
ve(2) = ess;                            % add the error to the variable of x(19)
a = roots(poly(1: 20) + ve);            % caculate the roots after adding the error
b = roots(poly(1: 20));                 % caculate the roots of original polyminal
x = 1 : 1 : 20;                         % set x from 1 to 10
plot(x, b, '-or', x, a', '-ob');        % plot the diagram
legend({'原多项式解','添加扰动后的多项式解'}) % add the legend
```

(2) “test1_1_2.m”

```
clear;                                  %clear the residue

ess = 0.000000001;                      %set the error as ess = 0.000000001
ve = zeros(1, 21);                      %create a 1x21 zero matrix
ve(2) = ess;                            %add the error to the variable of x^(19)
eqn1 = poly2sym(poly(1: 20));           %create original polyminal
eqn2 = poly2sym(poly(1: 20) + ve);      %create polyminal with the error
x = 1 : 1 : 20;                         %set x from 1 to 10

a = solve(eqn1);                        %caculate the roots of original polyminal
b = solve(eqn2);                        %caculate the roots of polyminal after adding
    the error

plot(x, b, '-or', x, a', '-ob');        %plot the diagram
legend({'添加扰动后的多项式解','原多项式解'}) %add the legend
```

(3) “test1_2.m”

```
clear;                  %clear the residue

r = 0;                  %set the sum resulf as r = 0
for i = 1: 1000         %caculate the sum of 1000 0.1
    r = 0.1 + r;
end
s = r - 100;            %minus the sum result with 100
```

2 实验二插值法

2.1 多项式插值的振荡现象

2.1.1 实验目的

在对多项式进行拉格朗日插值的过程中，节点的增加会导致多项式次数的升高 [5]。值得注意的是，高次的插值多项式序列很可能并不收敛，因此插值节点的增多不一定能够使插值多项式更逼近原函数，有时会在两端产生激烈震荡，也就是所谓的“Runge 现象”。

2.1.2 问题重述

考虑 Runge 发表在 1901 年的文章中使用的函数 [1]，

$$f(x) = \frac{1}{1+25x^2} \tag{6}$$

其定义区间为 $[-1,1]$。利用该函数进行多项式插值函数与原函数之间的误差分析。

2.1.3 方案设计

利用等距节点 x_i 进行拉格朗日插值，通过图像比较不同节点个数下的原函数与多项式函数之间的关系。其中，

$$x_i = -1 + \frac{2i}{n}, i = 0, 1, 2, ..., n \tag{7}$$

拉格朗日插值多项式为：

$$L_n(x) = \sum_{i=0}^{n} \frac{1}{1+25x^2} l_i(x) \tag{8}$$

n 为拉格朗日插值基函数的次数。

进一步对函数 $h(x)$ 和 $g(x)$ 重复上述实验，并分析其结果。其中：

$$h(x) = \frac{x}{1+x^4} \tag{9}$$

$$g(x) = arctanx \tag{10}$$

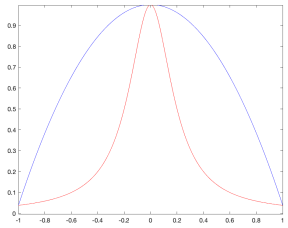
2.1.4 结果分析

（1）利用自行编写的“lagrange.m”作出原函数 $f(x)$ 和插值函数 $L_n(x)$ 在 $[-1,1]$ 上的图像，不同的节点数目 n 得到了不同的结果如图5。

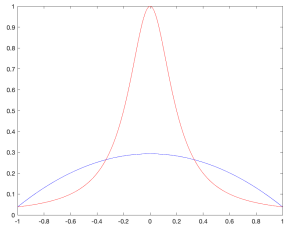
其中，蓝色曲线为插值多项式函数，红色曲线为原函数（下同）。

可以看到，n 越大插值的拟合效果越好，但在端点处的误差（震荡）也越大。

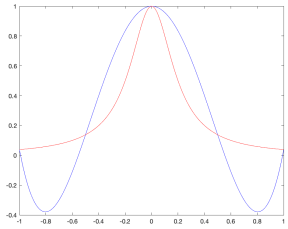
（2）考虑其他函数的情况，分别做出 $h(x)$ 和 $g(x)$ 与其插值函数在 $[-5,5]$ 上的函数图像，如图6、7。



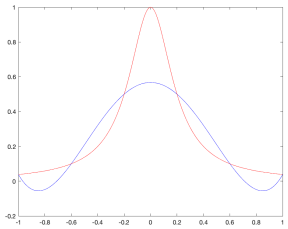
(a) n=2



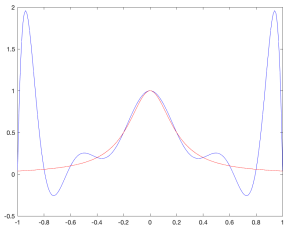
(b) n=3



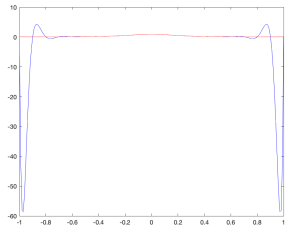
(c) n=4



(d) n=5

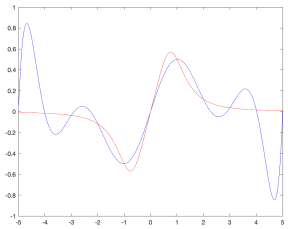


(e) n=10

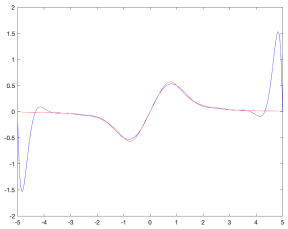


(f) n=20

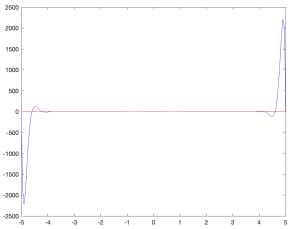
图 5: 不同节点数目下的 $f(x)$ 与 $L_n(x)$ 函数图像



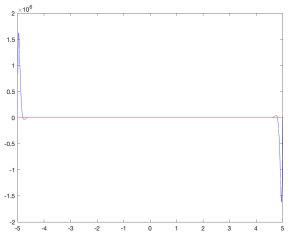
(a) n=2



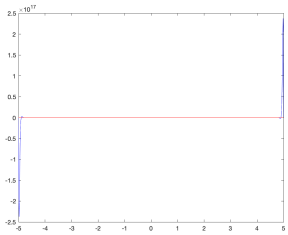
(b) n=3



(c) n=5

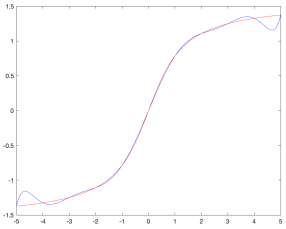


(d) n=10

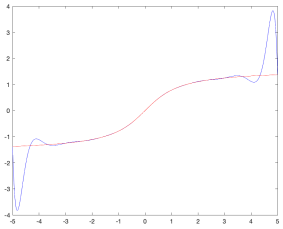


(e) n=20

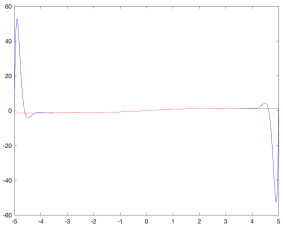
图 6: 不同节点数目下的 $h(x)$ 与 $L_n(x)$ 函数图像



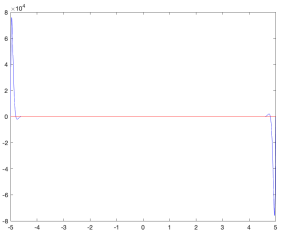
(a) n=2



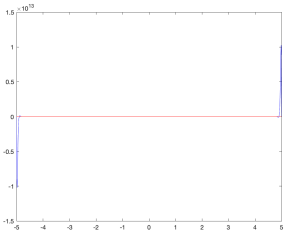
(b) n=3



(c) n=5



(d) n=10



(e) n=20

图 7: 不同节点数目下的 $g(x)$ 与 $L_n(x)$ 函数图像

可以看到，随着 n 的增大，$h(x)$ 和 $g(x)$ 的插值拟合效果都变好了，但从 n=5 开始端点处的震荡变得极为剧烈。

综上所述，对于部分函数，等距拉格朗日节点的增加在优化逼近效果的同时也会无可避免地造成多项式偏离原函数的现象。下面考虑使用其他插值多项式对上述函数进行进一步分析。

2.1.5 实验改进

定义区间 $[a,b]$ 上的切比雪夫点为:

$$x_k = \frac{b+a}{2} + \frac{b-a}{2} cos(\frac{(2k-1)\pi}{2(n+1)}), k = 1, 2, ..., n+1 \tag{11}$$

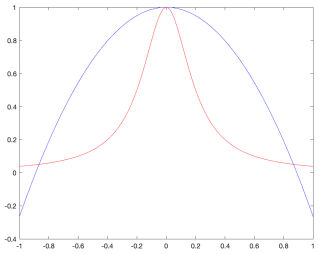
构造以上述 x_k 为插值节点的拉格朗日插值多项式。

分别作出原函数 $f(x)$、$h(x)$、$g(x)$ 和各自插值函数 $L_n(x)$ 在定义域上的图像（如图8-10），定义域分别为 $[-1,1]$、$[-5,5]$ 和 $[-5,5]$。

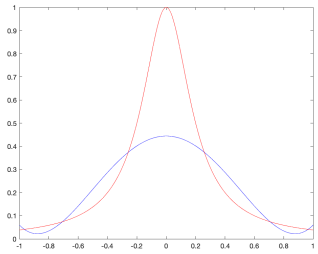
可以看到，随着节点数目 n 的增大，切比雪夫多项式的函数图像越来越逼近原函数的图像且不会出现 Runge 现象。这是由于 Runge 现象产生的根本原因是被插值函数不够光滑 [4]。以 $f(x) = \frac{1}{1+25x^2}$ 为例，尽管 $f(x)$ 在实数域上任意阶可导，但它在 $x = \pm\frac{i}{5}$ 处存在不解析情况，因此会产生 Runge 现象。当利用切比雪夫点进行插值时，函数只需要在插值区间内解析就可以避免 Runge 现象，因此可以实现插值效果随 n 的增大而变好的预期。

2.1.6 程序代码

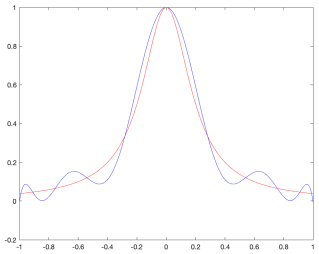
(1) “lagrange.m”



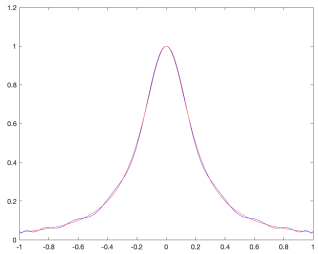
(a) n=2



(b) n=5

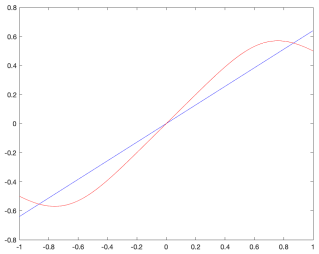


(c) n=10

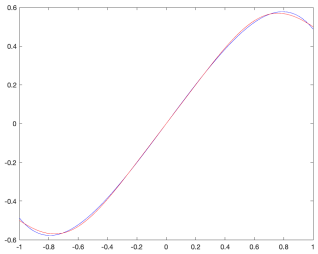


(d) n=20

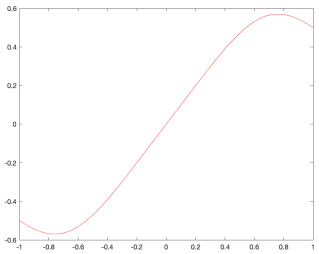
图 8: 不同节点数目下 $f(x)$ 与切比雪夫插值多项式的函数图像



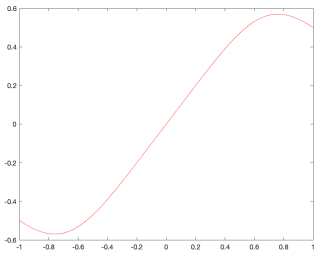
(a) n=2



(b) n=5

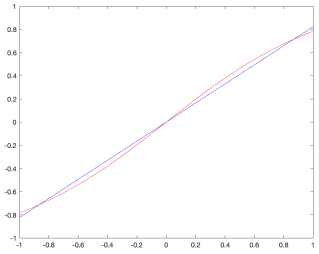


(c) n=10

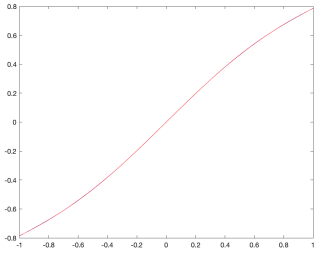


(d) n=20

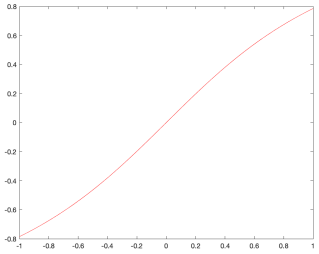
图 9: 不同节点数目下 $h(x)$ 与切比雪夫插值多项式的函数图像



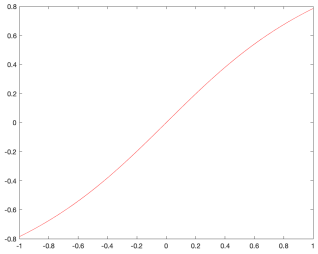
(a) n=2



(b) n=5



(c) n=10



(d) n=20

图 10: 不同节点数目下 $g(x)$ 与切比雪夫插值多项式的函数图像

```
function y = langrange(x0, y0, x)   % name the function

% x : value be used to predict
% y : value to estimate

    n = length(x);              % numbers to predict
    y = zeros(1, n);            % initialize to 0

    for k = 1: length(x0)
        j_nk = find((1: length(x0)) ~= k);
        % find to return the element without subscript k
        % k is from 1 to length(x0)

        y1 = 1;
        for j = 1: length(j_nk)
            y1 = y1 .* (x - x0(j_nk(j)));
            % multiply (x-xj) continuously from 1 to length(j_nk)
        end

        y = y + y1 * y0(k) / prod(x0(k) - x0(j_nk));
```

```
            % prod return to the product of array elements
        end
    end
```

(2) “test2_1.m”

```
clear;                                  % clear the residue

n = 20;                                 % set the node numbers
x0 = (-5: 2/n: 5);                      % set the range of x0
% x0 = cos(x2 .* pi); y0 = atan(x0);
x = (-5: 0.01: 5);                      % set the range of x
y = langrange(x0, y0, x);               % perform Lagrange interpolation

% y1 = arrayfun(@(x) 1./(1 + 25 .* x .* x), x);
y1 = arrayfun(@(x) atan(x), x);         % to draw the original function

plot(x, y,'-b', x, y1,'-r');            % plot the diagram
```

2.2 样条插值的收敛性

2.2.1 实验目的

由上述实验可知，多项式插值可能会存在 Runge 现象，导致插值节点增加而插值效果不一定变好。因此考虑样条函数插值的情况，通过实验证明可以避免理论验证收敛性的困难。

2.2.2 问题重述

利用适当方法选择合适的插值节点，取不同的节点数目分析原函数和插值函数的变化情况，并与拉格朗日插值的结果进行比较。

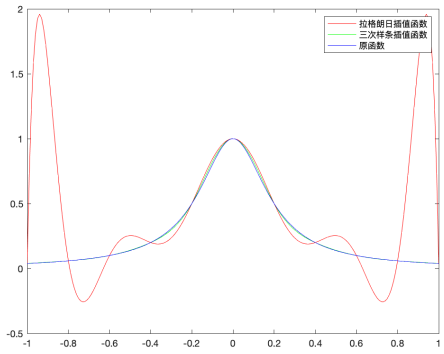
2.2.3 方案设计

经过尝试后选取 $f(x) = \frac{1}{1+25x^2}$ 作为原函数，利用 MATLAB 中的函数作出三次样条插值函数与原函数的图像，并进行比较。

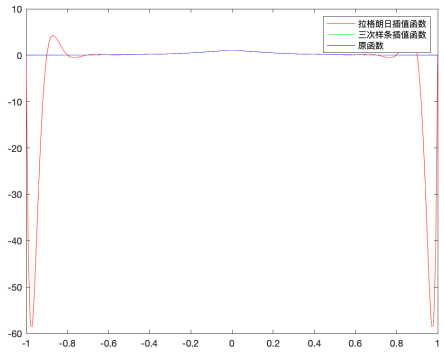
2.2.4 结果分析

结果如图11。

可以看到，在节点数目很大的情况下使用三次样条插值可以相当精确地逼近原函数，而拉格朗日插值则出现了明显的 Runge 现象。通过实验表明，样条插值是具有收敛性的。



(a) n=10



(b) n=20

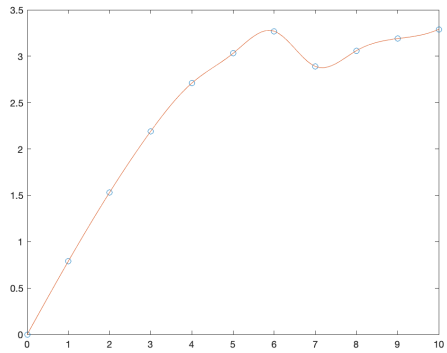
图 11: 不同节点数目下 $f(x)$ 与其三次样条插值、拉格朗日插值的函数图像

2.2.5 工程应用

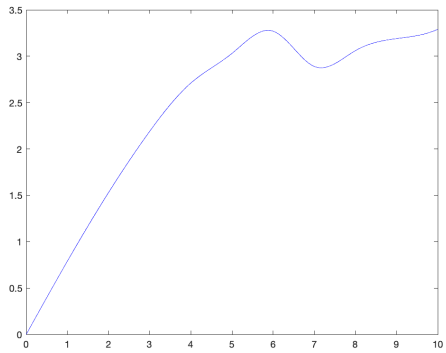
考虑应用于工业部门的样条插值情况。以某汽车制造商设计的车门曲线为例，数据如下：

x_k	0	1	2	3	4	5	6	7	8	9	10
y_k	0.0	0.79	1.53	2.19	2.71	3.03	3.27	2.89	3.06	3.19	3.29
y'_k	0.8										0.2

使用三弯矩方程构造 myspline 函数，利用追赶法求解出三次样条插值函数并作出图像 [2]。



(a) MATLAB 函数 spline



(b) 函数 myspline

图 12: 三转角方法求得的插值结果

由图12可以看到，使用三转角得到的 myspline 函数的插值结果与 MATLAB 自带的函数很接近，逼近原函数的效果较好。

2.2.6 程序代码

(1) “test2_2.m”

```
clear;                                  % clear the residue

n = 2;                                  % set the node numbers
x2 = (1/(2*(n+1)): 1/(n+1): (2*n+1)/(2*(n+1)));  % set the range of x2
% x0 = cos(x2 .* pi); x0 = (-5: 2/n: 5); y0 = atan(x0);
x = (-5: 0.01: 5);                      % set the range of x
% x = (-1: 0.01: 1);
y = langrange(x0, y0, x);               % perform Lagrange interpolation

% y1 = arrayfun(@(x) 1./(1 + 25 .* x .* x), x);
% y1 = arrayfun(@(x) x./(1 + x .* x .* x .* x), x);
y1 = arrayfun(@(x) atan(x), x);         % to draw the original function

yy = spline(x0, y0, x);                 % perform spline interpolation

plot(x, y,'-b', x, yy,'-g', x, y1,'-r');   % plot the diagram
legend({'拉格朗日插值函数','三次样条插值函数','原函数'})   % add the legend
```

(2) “myspline.m”

```
function [si] = myspline(y, y_0, y_10)    % name the function

    syms x;                               % set the parameter

    f = y(2: 11) - y(1: 10);              % f(i) = h_i

    d(1) = 6 * (f(1) - y_0);              % calculate the vector [d]
    d(11) = 6 * (y_10 - f(10));
    d(2: 10) = 3 * (f(2: 10) - f(1: 9));

    b = ones(1, 11) * 2;                  % set the diagonal [b] all to 2
    a = ones(1: 10);                      % set the diagonal [a] and [c]
    a(1: 9) = 1/2;
    c = ones(1, 10);
    c(2: 10) = 1/2;

    B(1) = c(1) / b(1);                   % use formula to calculate the [beta]

```

```
19          for i = 2: 10
20              B(i) = c(i) / (b(i) - a(i-1) * B(i-1));
21              % attention: [a]'s range is [1, 10], but in formula is [2, 11]
22          end
23
24          Y(1) = d(1) / b(1);                         % [Y] is the [y] in formula
25
26          for i = 2: 11
27              Y(i) = (d(i) - a(i-1) * Y(i-1)) / (b(i) - a(i-1) * B(i-1));
28          end
29
30          m(11) = Y(11);                              % [m] is the [M] in formula
31
32          for i = 10: -1: 1
33              m(i) = Y(i) - B(i) * m(i+1);
34          end
35
36          for i = 1: 10                               % use [m] to calculate variables
37              % [ai][bi][ci][di] is the si(x)-parameter, differs from[a][b][c][d]
38              % attention: [y] here is the value of y, not [y] in line 24
39
40              di(i) = y(i);
41              ai(i) = (m(i+1) - m(i)) / 6;
42              bi(i) = m(i) / 2;
43              ci(i) = y(i+1) - y(i) - (m(i+1) + 2 * m(i))/6;
44              % attention: the same as [di]
45
46              si(i) = ai(i) .* (x-i+1).^3 + bi(i) .* (x-i+1).^2 + ci(i) .* (
                    x-i+1) + di(i);
47              % set the si(x)
48          end
49      end
```

(3) "test2_3_1.m"

```
1   clear;                                  % clear the residue
2
3   S = myspline([0.0 0.79 1.53 2.19 2.71 3.03 3.27 2.89 3.06 3.19 3.29],
        0.8, 0.2);
4   % set the variables
5
```

```
    xs = zeros(1, 0);              % initialize to save the coordinate
    ys = zeros(1, 0);

    for i = 1: 1: 10               % draw the function S
        x = i-1: 0.01: i;
        y = subs(S(i), x);
        xs = [xs x];
        ys = [ys y];
    end

    plot(xs, ys, '-b');            % plot the diagram
```

(4) “test2_3_2.m”

```
    x = (0: 1: 10);                % set the range of x
    y = [0.0 0.79 1.53 2.19 2.71 3.03 3.27 2.89 3.06 3.19 3.29];
    % set the variables

    cs = spline(x,[0.8 y 0.2]);    % perform the spline interpolation
    xx = linspace(0, 10, 101);     % set the range of xx

    plot(x, y, 'o',xx, ppval(cs, xx), '-');   % plot the diagram
```

2.3 一维插值的应用：画图

2.3.1 实验目的

通过在实际场景下的使用，探究插值函数的现实应用意义。

2.3.2 问题重述

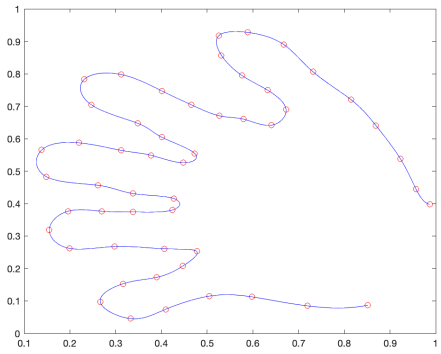
利用 MATLAB 程序进行插值操作，画出手掌的轮廓。

2.3.3 方案设计

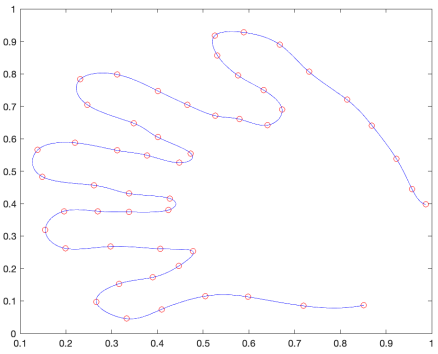
用鼠标在屏幕上勾勒出手掌的大致形状，获得用于插值的 52 个数据点 (x, y)。将 x 和 y 的值作为两个独立变量的函数进行插值，并画出函数图像，就得到了手掌的外形轮廓。

2.3.4 结果分析

插值结果如图13。



(a) interval=0.1



(b) interval=0.01

图 13: 不同作图精度下的手掌插值图像

2.3.5 程序代码

(1) “test2_4.m”

```
clear;                              % clear the residue

figure('position', get(0, 'screensize'));   % to get the figure of hand
axes('position', [0 0 1 1]);
[x, y] = ginput;

% draw the figure of hand
n = (1: 1: 52);                     % set the range of n
pp1 = spline(n, x);                 % perform the spline interpolation for x and y
pp2 = spline(n, y);

x0 = (1: 0.01: 52);                 % set the range of x0
xi = ppval(pp1, x0);
yi = ppval(pp2, x0);

plot(x, y, 'or', xi, yi, '-b');     % plot the diagram
```

(2) “drawhand.m”

```
n = (1: 1: 52);
pp1 = spline(n, x);
pp2 = spline(n, y);
x0 = (1: 0.01: 52);
xi = ppval(pp1, x0);
yi = ppval(pp2, x0);
```

```
7
8     plot(x, y, 'or', xi, yi, '-b');
```

2.4 思考题

2.4.1 实验目的

探究二维插值在实际生活中的应用。

2.4.2 问题重述

利用 MATLAB 程序进行二维插值操作，画出给定数据的地貌图。

2.4.3 方案设计

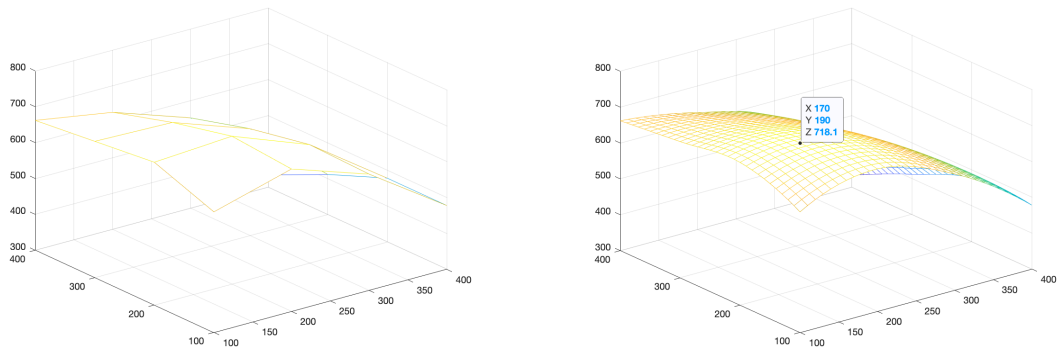
利用给定的高程数据，使用二维插值找到最高点及其高程。数据如下：

y/x	100	200	300	400
100	636	697	624	478
200	698	712	630	478
300	680	674	598	412
400	662	626	552	334

进一步生成 80 阶方阵 x, y, z 作为某个山区的地点三维数据，利用二维插值函数画出山区的地貌图和等值线图。

2.4.4 结果分析

（1）丘陵测高程

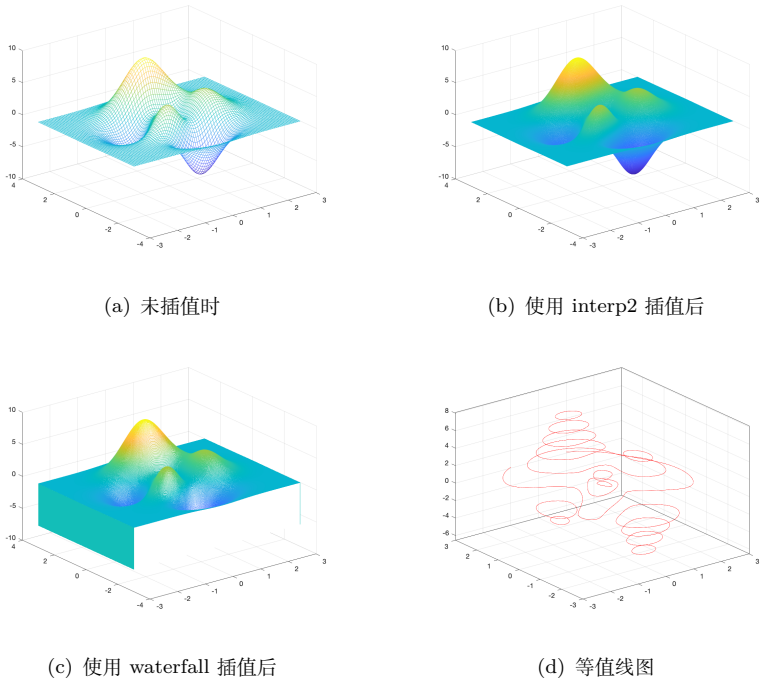


(a) 插值前　　(b) 插值后

图 14: 丘陵高程图

结果如图14所示。最高点在 $(170, 190)$ 处，该点的高程为 718.0618 米。

（2）山区地貌图绘制



(a) 未插值时　(b) 使用 interp2 插值后

(c) 使用 waterfall 插值后　(d) 等值线图

图 15: 山区地貌图

该山区地貌如图15(a)-15(c)，等值线图如图15(d)。

2.4.5 程序代码

(1) “mountain.m”

```
clear;                                  % clear the residue

[x, y, z] = peaks(80);                  % generate the data

xi = -3: 0.01: 3;                       % 2-dimension interpolation
yi = -3: 0.01: 3;
zi = interp2(x, y, z, xi', yi, 'cubic');

% mesh(xi, yi, zi); contour3(xi, yi, zi, 10, 'r');
```

(2) “twodimension.m”

```
clear;                                  % clear the residue

```

```
3     x = 100: 100: 400;                    % set the data
4     y = 100: 100: 400;
5     temps = [636 697 624 478; 698 712 630 478; 680 674 598 412; 662 626
          552 334];
6
7     xi = 100: 10: 400;                    % 2-dimension interpolation
8     yi = 100: 10: 400;
9     zi = interp2(x, y, temps, xi', yi, 'cubic');
10
11    mesh(xi, yi, zi);                     % create a mesh plot
12    max(max(zi));                         % calculate the highest point
```

3 实验三函数逼近与曲线拟合

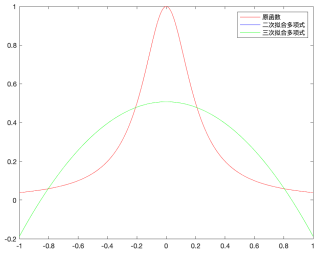
3.1 曲线逼近方法的比较

3.1.1 实验目的

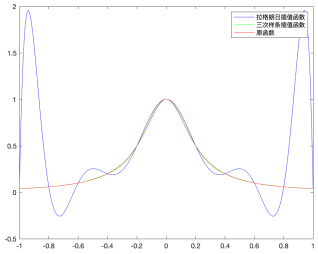
使用曲线逼近原函数通常有插值和拟合两种方法 [4]，这两种方法有各自的适用范围和效果。通过本次实验，需要对这两种方法进行分析比较，总结出各自的特点，方便后续在实际过程中进行应用。

3.1.2 问题重述

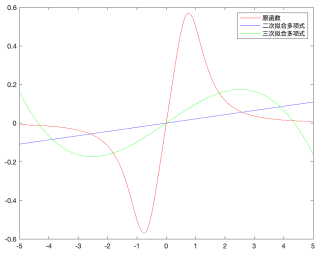
考虑实验 2.1 中的 Runge 问题，分别比较使用拟合和插值得到的结果，归纳得到逼近函数的两种方法的适用范围，进一步考虑其在实际生活中的应用。



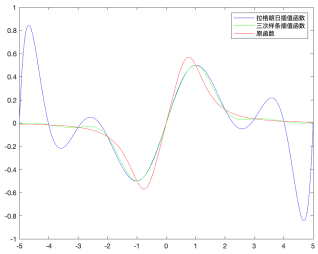
(a) $f(x)$ 的二次、三次多项式



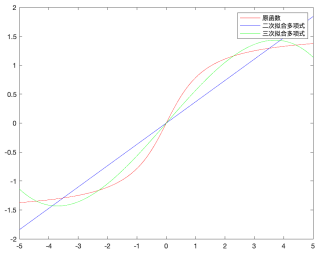
(b) $f(x)$ 的 Lagrange 插值及样条插值



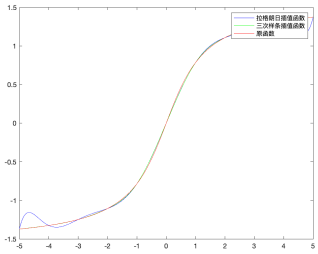
(c) $h(x)$ 的二次、三次多项式



(d) $h(x)$ 的 Lagrange 插值及样条插值



(e) $g(x)$ 的二次、三次多项式



(f) $g(x)$ 的 Lagrange 插值及样条插值

图 16: 拟合多项式与插值多项式的比较

3.1.3 方案设计

分别使用二次和三次多项式对 2.1 中的原函数 $f(x)$、$h(x)$ 和 $g(x)$ 进行拟合，并将结果与 2.1 的插值结果进行比较。

3.1.4 结果分析

（1）使用 polyfit 函数分别对三个原函数进行二次和三次多项式插值，与插值函数的比较结果如图16所示。

可以看到，二次与三次的拟合多项式虽然无法精准地经过每个数值点，但可以在不出现 Runge 现象的同时保持较好的逼近趋势；拉格朗日插值尽管能较为精准地逼近被插值函数，但在函数两端会出现较为明显的 Runge 现象；样条插值能够在逼近原函数的同时避免 Runge 现象，是较好地逼近原函数的方法。

（2）综上所述，插值方法由于需要经过已知观测点，因此得到简单的解析函数较为准确，但需要合理选择插值方法和插值节点以避免 Runge 现象；拟合方法由于需要使误差的平方和最小，因此常用于数据较多的情况，可以得到比较好的整体逼近效果，但也在一定程度上存在不可避免的误差。

3.1.5 程序代码

(1) “test3_1_1.m”

```
clear;                                  % clear the residue

% x = -1: 0.01: 1; x = -5: 0.01: 5;
% y = 1./(1+25.*x.*x); y = atan(x);
xx = -5: 0.01: 5;                       % actually I make the xx = x to simplify the
    compare
% xx = -1: 0.01: 1;

p2 = polyfit(x, y, 2);                  % draw the curve of polygon in degree n (2)
yy = polyval(p2, xx);

p3 = polyfit(x, y, 3);
yy2 = polyval(p3, xx);

xlabel('x');                            % plot the diagram
ylabel('y');
plot(x, y, '-r', xx, yy, '-b', xx, yy2, '-g');
legend({'原函数', '二次拟合多项式','三次拟合多项式'});   % add the legend
```

3.2 最小二乘拟合的经验公式和模型

3.2.1 实验目的

最小二乘作为一种简便易行的曲线拟合方法，被广泛应用于现实生活中 [6]。考虑使用最小二乘方法，分别应用于拟合模型已知与未知的两种实际场景，进一步探究其数学应用价值。

3.2.2 问题重述

（1）拟合模型已知场景

某种疾病的年龄段 x（一段：五年）与发病率 $y‰$ 的数据如下：

x	1	2	3	4	5	6	7	8	9
y	0.898	2.38	3.07	1.84	2.02	1.94	2.22	2.77	4.02

x	10	11	12	13	14	15	16	17	18	19
y	4.76	5.46	6.53	10.9	16.5	22.5	35.7	50.6	61.6	81.8

已知二者间存在 $y = ae^{bx}$ 的经验关系，试通过最小二乘方法确定其参数并进行后续分析。

（2）拟合模型未知场景

调查某年的美国汽车价格，数据如下表所示：

x_i	1	2	3	4	5	6	7	8	9	10
y_i	2615	1943	1494	1087	765	538	484	290	226	204

其中，x_i 表示汽车的使用年份数，y_i 表示对应的汽车平均价格，试分析拟合数据的曲线形式，并对使用 4.5 年后汽车的平均价格进行预测。

3.2.3 方案设计

（1）针对已知拟合模型的问题，使用最小二乘法求出该模型中的未知参数 a 和 b，画出离散数据与拟合函数的图像并计算该拟合函数在离散点处的误差大小和均方误差。

（2）针对未知拟合模型的问题，通过计算不同用于拟合的曲线的误差选择结果较好的拟合函数，并用于之后的数据预测。

3.2.4 结果分析

（1）使用 lsqcurvefit 函数，求得 a 为 0.2369，b 为 0.3090。作出图像如图17。

可以看到，虽然在个别点处该拟合结果存在一定误差，但整体的直观逼近效果较好。进一步对各个离散点处的误差进行分析，得到各点处数据误差如表1：

误差分布如图21：

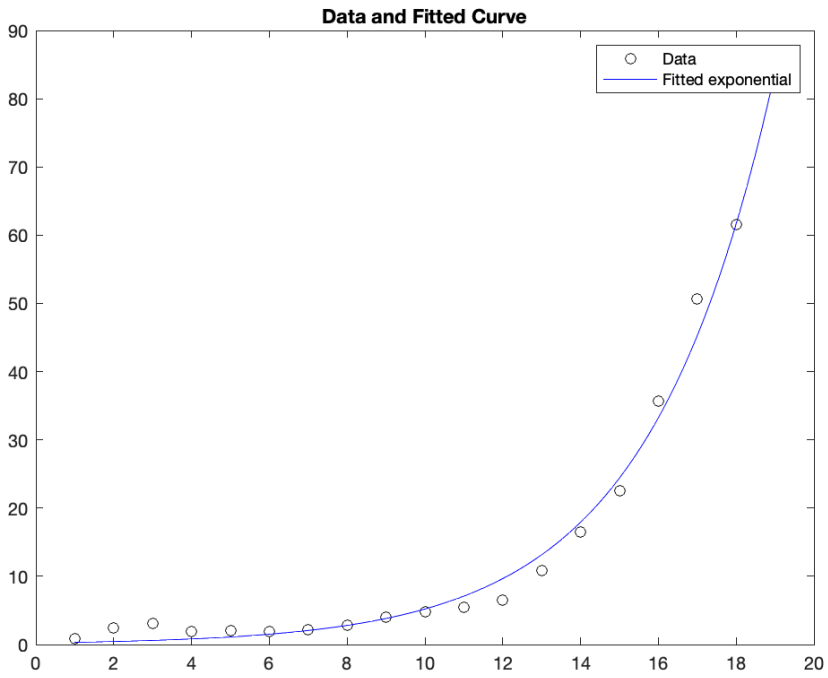


图 17: 疾病发病率与年龄段的关系拟合图

x	1	2	3	4	5	6	7	8	9	10
e	0.575	1.941	2.471	1.025	0.910	0.428	0.160	0.036	0.199	0.445
e^2	0.331	3.766	6.108	1.050	0.827	0.183	0.026	0.001	0.0395	0.198
x	11	12	13	14	15	16	17	18	19	sum
e	1.629	3.126	2.251	1.412	1.897	2.470	5.340	0.046	2.164	28.524
e^2	2.654	9.770	5.068	1.995	3.600	6.101	28.512	0.002	4.682	74.914

表 1: 各点处数据误差及误差平方

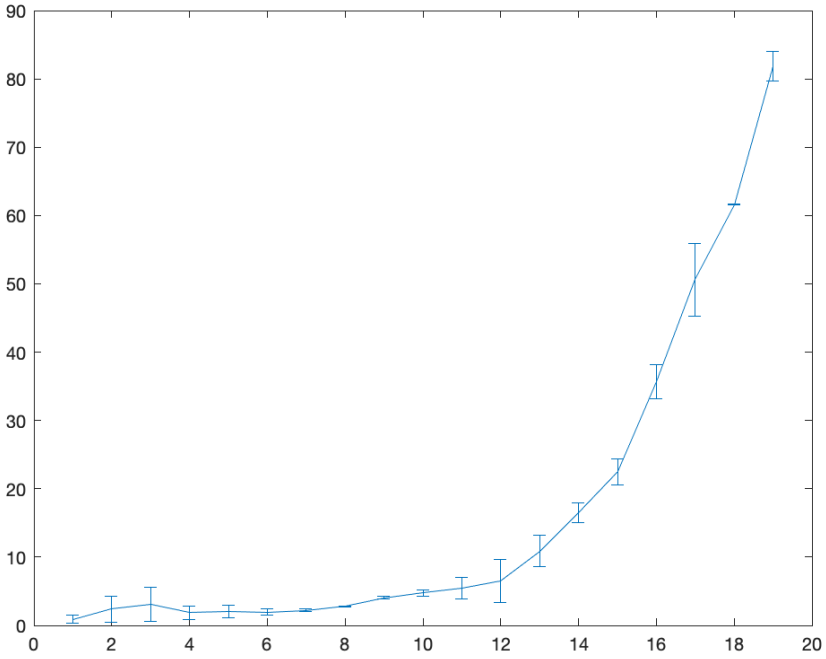


图 18: 上述最小二乘拟合在离散点处的误差图

整体离散点处的误差大小差别较大，需要利用均方误差得到误差分布。计算得该值约为 3.9428，说明该最小二乘的拟合效果较好。

（2）由于该场景下的函数模型未知，因此尝试不同模型进行拟合。

首先进行多项式拟合。分别选取次数为 3、4、5、10 的多项式进行拟合，拟合和误差结果如图19、20。不难看出，次数为 10 时的拟合误差最小，但出现了过拟合现象，不予考虑；分别计算次数为 3、4、5 的误差，结果如表。

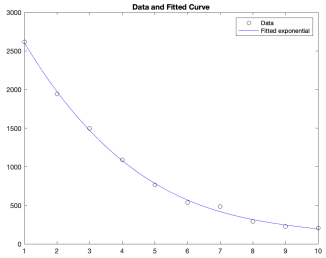
x_i	1	2	3	4	5	6	7	8	9	10	SSE
e_3	108.89	900.33	600.60	79.25	320.10	896.7	4444.9	630.83	375.44	139.70	8496.80
e_4	25.04	546.02	878.35	63.95	543.96	1251.5	4325.1	399.46	162.25	40.81	8236.39
e_5	61.98	903.72	906.92	175.76	418.35	1462.7	3660.8	418.81	36.52	12.38	8058.02

表 2: 不同次数多项式各点处数据误差及 SSE

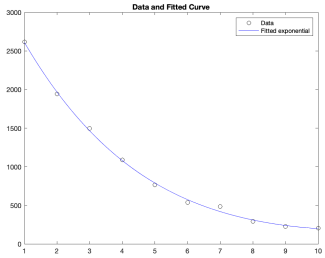
经过尝试，发现多项式次数为 5 时的 MSE 最小。

再考虑指数形式的情况，假设模型为 $y = a * e^{bx}$，代入数据拟合得到，此时的误差结果为表。

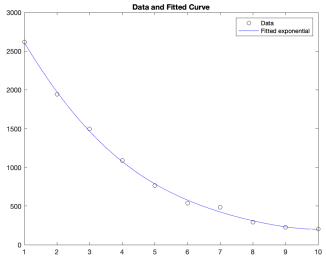
可以看到，此时的 SSE 远大于上面多项式的 SSE，因此最终选择利用 5 次多项式进行拟合。此时的相对误差平方和为 0.0282，可以认为拟合的精确度较好。此时的拟合公式为：



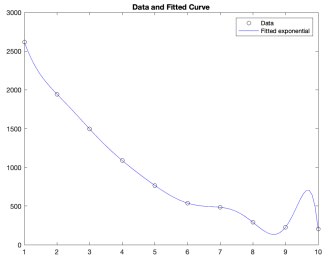
(a) n = 3



(b) n = 4

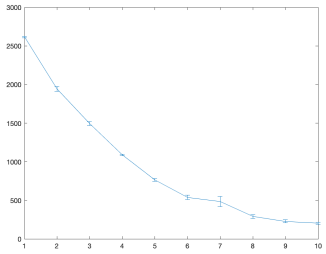


(c) n = 5

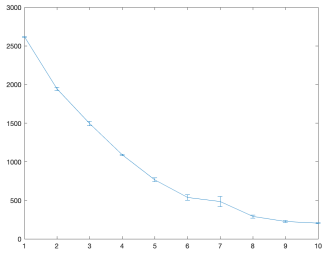


(d) n = 10

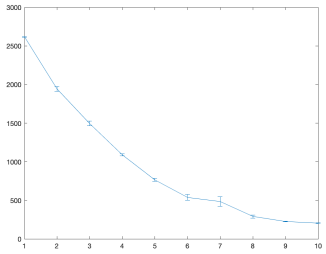
图 19: 多项式拟合结果图



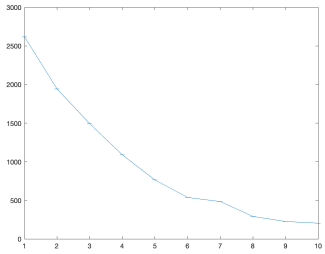
(a) n = 3



(b) n = 4



(c) n = 5



(d) n = 10

图 20: 多项式拟合误差分布图

x_i	1	2	3	4	5	6	7	8	9	10	SSE
e_1	13.104	5.648	49.146	15.691	29.338	50.974	47.296	33.801	14.087	25.984	10577.25

表 3: $y = ae^{bx}$ 各点处数据误差及 SSE

$$y = 0.0478x^5 - 1.189x^4 + 7.809x^3 + 41.017x^2 - 795.424x + 3354.867 \tag{12}$$

代入 $x = 4.5$ 即得到使用 4.5 年后的汽车的预计平均价格，为 918.226 元。

3.2.5 程序代码

(1) "test3_2_1.m"

```
clear;                                          %clear the residue

%input the data
xdata = [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19];
ydata = [0.898 2.38 3.07 1.84 2.02 1.94 2.22 2.77 4.02 4.76 5.46 6.53
    10.9 16.5 22.5 35.7 50.6 61.6 81.8];

fun = @(x, xdata) x(1)*exp(x(2)*xdata);   %set the function
x0 = [1 0.898];                           %choose two original parameters
x = lsqcurvefit(fun, x0, xdata, ydata);   %fitting to calculate the parameters

times = linspace(xdata(1), xdata(end));   %plot the diagram
plot(xdata, ydata, 'ko', times, fun(x,times), 'b-');
legend('Data', 'Fitted exponential');     %add the legend
title('Data and Fitted Curve');
hold on;

figure;                                   %plot the errorbar
y = fun(x, xdata);
e = abs(ydata - y);
errorbar(xdata, ydata, e);
```

(2) "test3_2_2.m"

```
clear;                                          %clear the residue

%input the data
xdata = [1 2 3 4 5 6 7 8 9 10];
ydata = [2615 1943 1494 1087 765 538 484 290 226 204];
```

```
6
7       % fun = @(x, xdata) x(1).*exp(x(2).*xdata);
8       times = linspace(xdata(1), xdata(end));
9       p = polyfit(xdata, ydata, 5);
10      yy = polyval(p, times);
11
12      % plot(xdata, ydata, 'ko', times, yy, 'b-');
13      % y = fun(x, xdata);
14      % figure;
15      %
16      % figure;
17      y = polyval(p, 4.5);          % calculate the average value after 4.5 years
```

3.3 研究最佳平方逼近多项式的收敛性质

3.3.1 实验目的

对于某一给定的函数 $f(x)$，能用简单函数 $\{\phi_0(x)\ \phi_1(x)\ldots\ldots\phi_n(x)\}$ 的线性组合逼近该函数，使得 $||f(x)-S(x)||_2^2$ 的值最小，则称 $S(x)$ 为 $f(x)$ 在 ϕ 中的最佳平方逼近多项式。

通过该实验，能够更直观地理解最佳平方逼近多项式的构造方法，也方便后续进行误差计算等工作。

3.3.2 问题重述

令 $[-1,1]$ 上的 Legendre 多项式为基函数，对 $n\in(0,10)$ 构造函数 $f(x)=e^x$ 的最佳平方逼近多项式 $P_n(x)$。分析该多项式的逼近误差，并探究 n 与收敛性之间的关系。

3.3.3 方案设计

画出 $\epsilon_n(x)=|f(x)-P_n(x)|$ 关于 x 的曲线。进一步作出 $E_n=max|f(x)-P_n(x)|$ 关于 n 的最小二乘曲线，探究 E_n 关于 n 的收敛性关系。

3.3.4 结果分析

可以看出，拟合多项式的次数越高，拟合误差 $\epsilon_n(x)$ 就越小，且从次数为 5 开始误差就已小于 10^{-4}，可以认为拟合的效果很好。从最大误差随次数的分布图中也能看出，随着次数 n 的增大，拟合多项式与原函数之间的最大误差近似呈指数下降分布。

综上所述，可以认为最佳平方逼近是收敛的。

3.3.5 程序代码

(1) “test3_3_1.m”

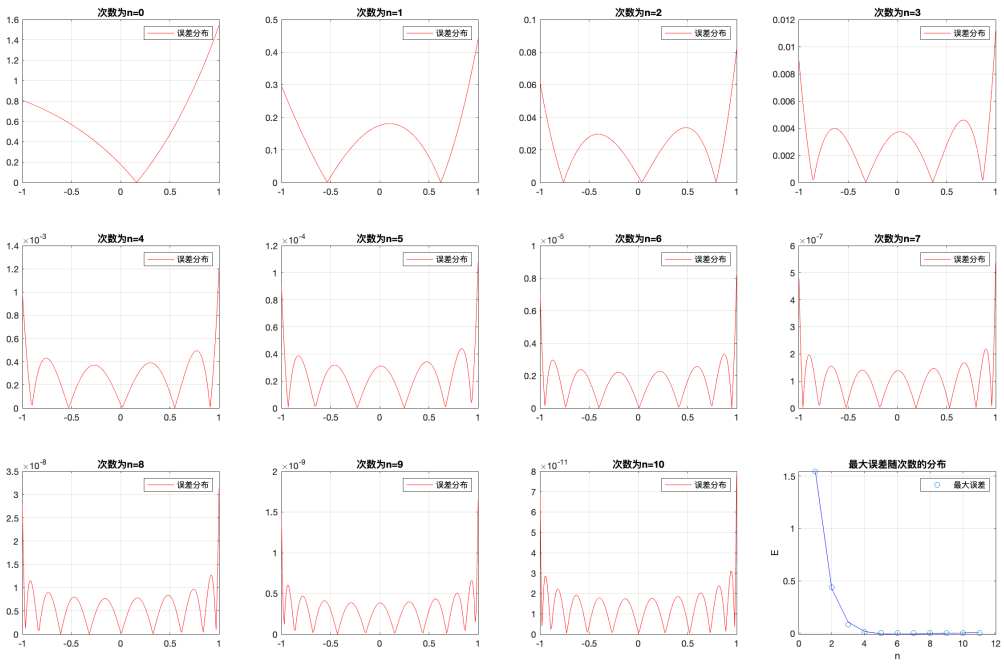


图 21: n 从 0-10 的 $P_n(x)$ 及 E_n 关于 n 的最小二乘拟合曲线

```
clear;                          %clear the residue

s = legendre(10);               %set the n in P_n(x)with10
```

(2) “legendre.m”

```
function s = legendre(n)        %name the function

syms x;                         %define symbol x

s = sym(zeros(n+1, 1));         %define symbol array s, p, a, e
p = sym(zeros(n+1, 1));
a = sym(zeros(n+1, 1));
e = sym(zeros(n+1, 1));

f = exp(x);                     %set the function f

p(1) = 1;                       %set order 1 Legendre function
p(2) = x;                       %set order 1 Legendre function

```

```
    for i = 3 : n+1                    %calculate order 3-n Legendre function
        p(i) = ((2 * i - 3) * x * p(i-1) - (i-2) * p(i-2))/(i-1);
    end

    for j = 1 : n+1                    %calculate orthogonal polynomial coefficients
        a(j) = (2 * j - 1)/2 * (int(f * p(j), -1, 1));
    end

    s(1) = a(1) * p(1);
    e(1) = abs(f - s(1));

    for j = 1 : n                      %find the Best-Square-Approximation Polynomial
        s(j+1) = s(j) + a(j+1) * p(j+1);
        e(j+1) = abs(f - s(j+1));
    end

    E = zeros(n+1, 1);

    for i = 1 : n+1                    %plot the diagram from n=0 to n=10
        subplot(3,4,i);
        x1 = -1 : 0.01 : 1;
        e1 = subs(e(i), x1);
        plot(x1, e1, 'r-');
        title(['次数为n=', int2str(i-1)]);
        legend('误差分布');
        grid on;
        E(i) = max(double(e1));
    end

    %plot the-least-square curve between En-n
    N = 1 : n+1;
    a0=[1 -1 1 1];
    fun = @(a, xdata) a(1) .* exp(a(2) .* xdata) + a(3) .* xdata + a(4);

    a = lsqcurvefit(fun, a0, N, E');

    subplot(3,4,12);
    y = a(1) .* exp(a(2) .* N) + a(3) .* N + a(4);
```

```
54      plot(N, E, 'o', N, y, '-b');
55      title('最大误差随次数的分布');
56      xlabel('n');
57      ylabel('E');
58      legend('最大误差');
59      grid on;
```

3.4 思考题 1：病态问题

3.4.1 实验目的

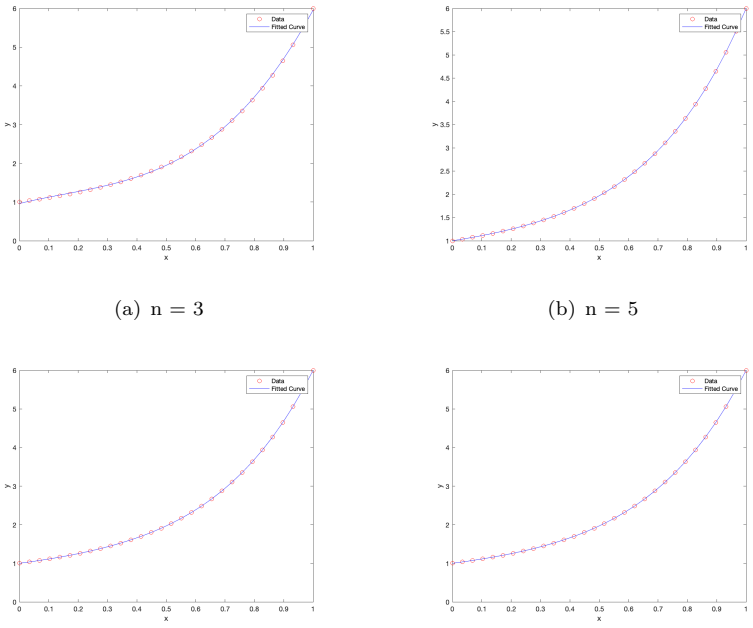
探究 polyfit 拟合多项式时产生误差的原因。

3.4.2 问题重述

将 $[0,1]$ 等距划分为 30 个节点，使用多项式 $y = 1 + x + x^2 + x^3 + x^4 + x^5$ 生成 30 组 (x, y) 数据。利用 polyfit 求其不同次数下的拟合多项式并加以分析。

3.4.3 结果分析

分别取次数 n 为 3、5、10、15，求 (x, y) 数据的 n 次多项式，结果如图22。



(a) n = 3　　(b) n = 5

(c) n = 10　　(d) n = 15

图 22: n 次多项式拟合结果图

在进行 15 次多项式拟合的时候，出现了 “Polynomial is badly conditioned” 的 Warning。这意味着数据点太少，用于拟合的条件不足。由多项式的性质可知，n 次多项式有 $n+1$ 个系数 $\{a_0, a_1 \ldots a_n\}$。

因此，在用多项式进行 n 次数据拟合时，被拟合数据点的个数必须大于等于 $n+1$ 个，不然将缺少条件，无法解出多项式系数。

3.4.4 程序代码

(1) “test3_4_1.m”

```
clear;                                  %clear the residue

x = linspace(0, 1, 30);                 %set the data
y = 1 + x + x.*x + x.^3 + x.^4 +x.^5;
p = polyfit(x, y, 15);                  %perform the polyfit interpolation

y0 = polyval(p, x);                     %calculate the fitting function

plot(x, y, 'ro', x, y0, '-b');          %plot the diagram
xlabel('x');
ylabel('y');
legend('Data', 'Fitted Curve');
```

3.5 思考题 2：MATLAB 调研

3.5.1 拟合工具箱 Curve Fitting ToolBox

可以使用 cftool 在命令行中快速打开 MATLAB 的拟合工具箱。选择 workshop 中的待拟合数据或 load census 就可以将数据导入工具箱，在右侧的下拉选项可以选择拟合方式（如 Polynomial 多项式拟合、Fourier 傅立叶拟合等），也可以在 fit options 里选择次数、稳定性和周期数等参数。使用 autofit 或 fit/stop 进行拟合。Result 窗口有拟合的数据结果和曲线示意图，可以比较直观地看到拟合的效果。TableofFits 给出了对该拟合的误差参数（SSE、RMSE 等）的计算，可以比较方便地比较多次拟合的误差大小，便于选定更优的拟合方法。

3.5.2 CF toolbox 提供的拟合曲线

函数名称	逼近方式	拟合函数类型
Exponential	指数逼近	ae^{bx} 和 $ae^{bx}+ce^{dx}$ 两种
Fourier	傅立叶逼近	$a_0+a_1cos(xw)+b_1sin(xw)$ 等七种
Gaussian	高斯逼近	$a_1e^{c_1-(\frac{x-b_1}{c_1})^2}$ 等八种
Interpolant	插值逼近	cubic spline、linear、nearest neighbor 等四种
Polynomial	多形式逼近	linear、cubic 等九种
Power	幂逼近	ax^b 和 ax^b+c 两种
Rational	有理数逼近	/
Smoothing Spline	平滑样条逼近	/
Sum of Sin Functions	正弦曲线逼近	$a_1sin(b_1x+c_1)$ 等八种
Weibull	韦伯分布逼近	只有 $abx^{b-1}e^{-ax^b}$ 一种

4 实验五解线性方程组的直接方法

4.1 主元的选取与算法的稳定性

4.1.1 实验目的

Gauss 消去法是线性代数计算中一种重要的方法，经由计算机从理论到实践会产生舍入误差，而该误差会导致 Gauss 消去算法的不稳定性。选择合适的主元可以在一定程度上减小这种算法的不稳定性，如何选择主元也是数值分析中一个重要的问题。

4.1.2 问题重述

设计一个能控制如何选取主元的 Gauss 消去程序，用于求解线性方程组：

$$Ax=b, A\in R^{n\times n}, b\in R^n \tag{13}$$

4.1.3 方案设计

（1）Gauss 消去过程

将矩阵 A 与 b 合并拓展成矩阵 B，在第 i 步利用消去因子 $l_{j,i}=\frac{B_{j,i}}{B_{i,i}}$ 对矩阵 B 进行 Gauss 消去，将矩阵 B 中第 i 列 a_{ii} 以下的元素全变为 0，最终得到便于计算的上三角矩阵。

（2）计算矩阵 A 的条件数，分别选取顺序消元、按模最小消元、按模最大消元对线性方程组进行求解，分析不同矩阵条件数下选取不同主元的 Gauss 消去法得到的结果的不同。

4.1.4 结果分析

（1）利用顺序选主元方法，计算解已知的线性方程组 $Ax=b, A\in R^{nxn}, b\in R^n$ 的解。其中，

显然，方程的解 $x^*=(1,1,...,1)^T$。取 n=10，利用 *cond* 函数计算得此时矩阵 A 的 1-范数、2-范数、∞-范数的条件数分别为 2557.52、1727.556 和 2557.5。

$$A=\begin{bmatrix} 6 & 1 & & & \\ 8 & 6 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 8 & 6 & 1 \\ & & & 8 & 6 \end{bmatrix}, b=\begin{bmatrix} 7 \\ 15 \\ \vdots \\ 15 \\ 14 \end{bmatrix}$$

计算得到的结果与 x^* 之间的 ∞-范数，为 $7.5717e^{-14}$。可以发现，在 n 取 10 时，尽管三类条件数均较大，但使用顺序选主元的计算方法得到的结果和实际值误差并不大。

（2）利用手动选主元方法。若选取每次模最小的元素作为主元进行计算，得到 n=10 时的结果与实际值的 ∞-范数为 $7.5717e^{-14}$，误差较小；若选取每次模最大的元素作为主元进行计算，得到 n=10 时的结果与实际值的 ∞-范数为 0，误差不存在。

（3）上述前两种方法相较于最大选主元方法的结果差别不大，因此考虑选取大于 10 的 n 值，重复进行上述计算。

当 n=20 时，计算得上述三种方法结果误差的 ∞-范数分别为 $7.7606e^{-11}$、$7.7606e^{-11}$ 和 0，此时矩阵 A 的 1-范数、2-范数、∞-范数的条件数分别为 2621437.5、1789670.5658 和 2621437.5。可以看到，此时的条件数已经非常大，结果误差的 ∞-范数的数量级也增大了。此时利用顺序选主元和手动选最小主元的方法均存在误差，而手动选最大主元的方法仍然不存在误差。

（4）利用三对角矩阵 A，分别选取 n=30、50、100 重复上述实验。结果如下：

n	顺序选主元	手动选最小主元	手动选最大主元	∞-范数的条件数
30	$7.9468e^{-8}$	$7.9468e^{-8}$	0	$2.68e^{9}$
50	0.083328	0.083328	0	$2.81e^{15}$
100	$9.38e^{13}$	$9.38e^{13}$	6.4978	$3.17e^{30}$

此时可以看到，手动选最大主元的方法在条件数极大的条件下仍然能够保持较好的计算精确度，而其余两种方法在三对角矩阵 A 的维度达到一定程度时会产生巨大的计算误差。

继续考虑矩阵 A 为随机矩阵的情况。取 n=20，此时 A 的 ∞-范数的条件数为 144.9943，顺序选主元、手动选最小主元、手动选最大主元方法得到的误差范数分别为 424.3655、424.3655、424.3655。可以看到，当矩阵为随机矩阵时，三种方法的优劣性就没有绝对的规律可循了。

一般来说，对于低阶的矩阵，三种方法均可以得到较为精确的结果；而对于高阶矩阵，采取手动选取最大主元的方法进行计算可以得到较好的结果。

4.1.5 程序代码

(1) “gausselimination.m”

```
function x = gausselimination(A, n, b, sw)
    %combine the A and b as B
```

```
3           l = diag(ones(1, n));          % create the elimination factor
4           x = zeros(1, n);
5           B = [A, b];
6
7           % choose sw as 1 to use the sequential elimination
8           if sw == 1
9               disp('use the sequential elimination');
10              for i = 1 : n-1
11                  if B(i, i) == 0
12                      disp("The pivot is 0, Gauss elimination cannot
                            continue.");
13                      break
14                  end
15
16                  for j = i+1 : n
17                      l(j, i) = B(j, i) / B(i, i);
18                      B(j, :) = B(j, :) - B(i, :) * l(j, i);
19  % b(j) = b(j) - b(i) * l(j, i);
20                  end
21              end
22
23              x(n) = B(n, n+1) / B(n , n);
24              x(n-1) = (B(n-1, n+1) - B(n-1 , n) .* x(n)) / B(n-1 , n-1);
25
26              for p = n-2 : -1 : 1
27                  x(p) = (B(p, n+1) - B(p , p+1 : n) * x(p+1 : n)') / B(p, p
                        );
28              end
29  % disp(["the answer of sequential elimination is: ", num2str(x)]);
30
31          % choose sw as 2 to use the minimum modulus elimination
32          elseif sw == 2
33              disp('use the minimum modulus elimination');
34              for i = 1 : n-1
35                  E = abs(B(i : n, i)); % the element of the list i
36                  [~, k] = min(E(E > 0));
37                  a = B(k+i-1, i);
38                  B([i k+i-1], :) = B([k+i-1 i], :);
39
```

```
            if B(i, i) == 0
                disp("The pivot is 0, Gauss elimination cannot
                    continue.");
                break
            end

            for j = i+1 : n
                l(j, i) = B(j, i) / a;
                B(j, :) = B(j, :) - B(i, :) * l(j, i);
            end
        end

        x(n) = B(n, n+1) / B(n , n);
        x(n-1) = (B(n-1, n+1) - B(n-1 , n) .* x(n)) / B(n-1 , n-1);

        for p = n-2 : -1 : 1
            x(p) = (B(p, n+1) - B(p , p+1 : n) * x(p+1 : n)') / B(p, p
                );
        end

    % choose sw as 3 to use the maximum modulus elimination
    elseif sw == 3
        disp('use the maximum modulus elimination');
        for i = 1 : n-1
            E = abs(B(i : n, i)); % the element of the list i
            [~, k] = max(E(E > 0));
            a = B(k+i-1, i);
            B([i k+i-1], :) = B([k+i-1 i], :);

            if B(i, i) == 0
                disp("The pivot is 0, Gauss elimination cannot
                    continue.");
                break
            end

            for j = i+1 : n
                l(j, i) = B(j, i) / a;
                B(j, :) = B(j, :) - B(i, :) * l(j, i);
            end
```

```
            end

            x(n) = B(n, n+1) / B(n , n);
            x(n-1) = (B(n-1, n+1) - B(n-1 , n) .* x(n)) / B(n-1 , n-1);

            for p = n-2 : -1 : 1
                x(p) = (B(p, n+1) - B(p , p+1 : n) * x(p+1 : n)') / B(p, p
                    );
            end
        end
    end
```

(2) "test5_1.m"

```
clear;                          %clear the residue

n = 100;                        %set the dimension n

%create the matrix A
%A = rand(n);
A = diag(ones(1, n)*6) + diag(ones(1, n-1), 1) + diag(ones(1, n-1)*8,
    -1);
b = [7 ones(1, n-2)*15 14]';
X = A\b;                        %calculate the X

%output the cond
c1 = cond(A, 1);
c2 = cond(A, 2);
c3 = cond(A, Inf);
disp(['1 cond = ', num2str(c1), ' 2 cond = ', num2str(c2), ' Infinity
    cond = ', num2str(c3)]);

%1:sequential elimination 2:minimum modulus elimination 3:maximum modulus elimination
y1 = gausselimination(A, n, b, 1);
error1 = norm(y1 - X, Inf);
disp(['the sequential elimination error is: ', num2str(error1)]);

y2 = gausselimination(A, n, b, 2);
error2 = norm(y2 - X, Inf);
disp(['the minimum elimination error is: ', num2str(error2)]);

```

```
    y3 = gausselimination(A, n, b, 3);
    error3 = norm(y3 - X, Inf);
    disp(['the maximum elimination error is: ', num2str(error3)]);
```

4.2 线性代数方程组的性态与条件数的估计

4.2.1 实验目的

探究线性代数方程组的病态性与条件数估计值之间的联系，通过估计计算对该联系进行一定的分析。

4.2.2 问题重述

矩阵的病态与否可以根据矩阵的条件数大小进行一定的判断。从理论上对 $\delta x/x$ 进行计算，得到如下公式：

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{cond(A)}{1-\|A^{-1}\|\|\Delta A\|}\left(\frac{\|\Delta A\|}{\|A\|}+\frac{\|\Delta b\|}{\|b\|}\right)$$

可以利用 MATLAB 内置的“condest”函数来计算该不等式的右端项，从而近似地估计出左端的误差结果。

4.2.3 方案设计

（1）利用方程 $(A+\triangle A)x=b+\triangle b$，计算出 $\frac{\|\triangle x\|_1}{\|x\|_1}$ 的结果，比较不同维度 n 下 MATLAB 函数“condest”的计算所需时间的差别。

（2）进一步利用“eig”函数，给出 2-范数的条件数的计算估计值，与 MATLAB 函数“cond”所得的精确值进行比较，最终给出利用“condest”估计出 $\frac{\|\triangle x\|_1}{\|x\|_1}$ 的理论值，分析得到的结果。

（3）利用上述方法对 Hilbert 矩阵的条件数进行估计，并加以分析。

4.2.4 结果分析

（1）随机生成维度为 10 的方阵 A 和 $n\times 1$ 的矩阵 x，利用 A*x 生成矩阵 b。考虑在 A 和 b 上均加上微小扰动 $\triangle A$ 和 $\triangle b$，利用右除运算符计算出添加扰动后的 x 值。使用 norm 对 1-范数进行精确计算，计算结果如下：

$$\frac{\|\triangle x\|_1}{\|x\|_1}=2.9317e^{-5} \tag{14}$$

（2）分别选取 n 为 10、20、30，利用“condest”函数进行条件数的计算，通过“tic-toc”指令得到它们运行的机器时间分别为 0.000680、0.000507 和 0.000521 秒。可以发现，整体的计算时间均较小，

且通常情况下 n=20 时的运行时间最小。

（3）考虑利用“eig”函数计算 $cond_2(A)$ 的值，将其与函数“cond”的计算结果作差后取 ∞-范数，得到的结果如下：

n	10	20	30
$error$	$3.5527e^{-13}$	$4.7805e^{-8}$	$3.45112e^{-8}$

可以看到，使用“eig”得到的条件数的精确度较好。

（4）使用理论公式对 $\frac{\|\triangle x\|_1}{\|x\|_1}$ 进行估计，结果如下表：

n	$\frac{\|\triangle x\|_1}{\|x\|_1}$	估计值
10	$4.5236e^{-5}$	$8.4873e^{-4}$
20	$3.2611e^{-4}$	$1.89e^{-2}$
30	$6.4385e^{-5}$	$7.5e^{-3}$

可以发现，结果的估计值均大于等于精确解，且估计值的精确度与矩阵的选取有关，整体表现为 n 越小估计的精度越高。

（5）分别利用“cond”和“condest”函数对 Hilbert 矩阵的条件数进行估计。

n	1-范数	2-范数	∞-范数	1-范数的估计值
1	1	1	1	1
2	27	19.2815	27	27
3	748	524.0568	748	748
4	28375	15513.7387	28375	28375
5	$9.4366e^5$	$4.7661e^5$	$9.4366e^5$	$9.4366e^5$
7	$9.8519e^8$	$47537e^8$	$9.8519e^8$	$9.8519e^8$
10	$3.5353e^{13}$	$1.6025e^{13}$	$3.5353e^{13}$	$3.5353e^{13}$

可以看到，随着 Hilbert 矩阵维度的增加，矩阵的条件数呈指数上升。由于矩阵的条件数越大，该矩阵的病态性越严重，因此在实际求解线性代数方程组的过程中需要避免对 Hilbert 矩阵的求解。

4.2.5 程序代码

（1）“test5_2_1.m”

```
clear;                              %clear the residue

%create the random matrix
```

```
n = 10;
A = rand(n);
x = rand(n, 1);
b = A * x;

%add the error
dA = 0.00001 * rand(n);
db = 0.00001 * rand(n, 1);

%calculate the x
A1 = A + dA;
b1 = b + db;
x1 = A1\b1;

%calculate the relative error
dx = x1 - x;
r = norm(dx, 1)/norm(x, 1);
```

(2) “test5_2_2.m”

```
clear;                          %clear the residue

%create the random matrix
n1 = 10;
A1 = rand(n1);
n2 = 20;
A2 = rand(n2);
n3 = 30;
A3 = rand(n3);

%write down the time of calculation
disp('n = 10 时运行时间: ');
tic
    c1 = condest(A1);
toc

disp('n = 20 时运行时间: ');
tic
    c2 = condest(A2);
toc

```

```
    disp('n = 30 时运行时间: ');
    tic
        c3 = condest(A3);
    toc
```

(3) “test5_2_3.m”

```
clear;

n = 40;
A = rand(n);
%B = inv(A);

B = A'*A;
E = eig(B);

c1 = sqrt(max(E)/min(E));
c2 = cond(A, 2);

d = abs(c1 - c2);
```

(4) “test5_2_4.m”

```
clear;                          %clear the residue

n = 20;                         %set the dimension n
A = rand(n);                    %create the equation
x = rand(n, 1);
b = A * x;

%add the error
dA = 0.00001 * rand(n);
db = 0.00001 * rand(n, 1);

%calculate the x
A1 = A + dA;
b1 = b + db;
x1 = A1\b1;

%calculate the relative error
dx = x1 - x;
r1 = norm(dx, 1)/norm(x, 1);
```

```
20
21      c = condest(A);
22      r2 = (norm(dA, 1)/norm(A, 1) + norm(db, 1)/norm(b, 1)) * c/(1 - c *
            norm(dA, 1)/norm(A, 1));
23      % r3 = (norm(dA, 1)/norm(A, 1) + norm(db, 1)/norm(b, 1)) * c/(1 - norm(inv(A), 1) *
            norm(dA, 1));
24
25      s = abs(r1 - r2);
```

(5) “test5_2_5.m”

```
1       clear;                          % clear the residue
2
3       n = 7;                          % create the Hilbert matrix
4       H = hilb(n);
5
6       % H = zeros(n, n);
7       % for i = 1 : 1 : n
8       % for j = 1 : 1 : n
9       % H(i, j) = 1/(i + j - 1);
10      % end
11      % end
12
13      % output the cond
14      c1 = cond(H, 1);
15      c2 = cond(H, 2);
16      c3 = cond(H, Inf);
17      disp(['1 cond = ', num2str(c1), ' 2 cond = ', num2str(c2), ' Infinity
            cond = ', num2str(c3)]);
18
19      % estimate the condest
20      c = condest(H);
21      disp(['1 cond evaluate = ', num2str(c)]);
```

4.3 思考题

4.3.1 实验目的

通过上述方法探究 Vadermonde 矩阵的病态性与条件数估计值之间的联系。

4.3.2 问题重述

已知 Vadermonde 矩阵如下：

$$A=\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}$$

计算 n 取不同值时 A 的条件数和 Ax=b 的解，观察其中的规律。其中，

$$b=\begin{bmatrix} \sum_{i=0}^{n} x_0^i \\ \sum_{i=0}^{n} x_1^i \\ \sum_{i=0}^{n} x_2^i \\ \vdots \\ \sum_{i=0}^{n} x_n^i \end{bmatrix}$$

$x_k = 1 + 0.1k, k = 0, 1, ..., n$，$Ax = b$。

4.3.3 方案设计

选取 n 的值为 2、5、8，分别计算 A 的条件数，解出 n=5 时的方程的解并考虑其在收到扰动时的误差大小。

4.3.4 结果分析

（1）计算所得的条件数的结果为：

n	1-范数的条件数	2-范数的条件数	∞-范数的条件数
2	44.1	42.0762	44.1
5	$6.5120e^5$	$3.5740e^5$	$6.2755e^5$
8	$2.9273e^9$	$1.3926e^9$	$2.8128e^9$

可以发现，随着 n 的增加，矩阵 A 的条件数迅速上升，矩阵逐渐变得非常病态。

（2）取 n=5，解方程得 $x = [11111]^T$。在矩阵 A 的最后一个元素上添加 10^{-4} 大小的扰动，重新求解 Ax=b，得到 $x_1 = [0.9314, 1.2410, 0.6836, 1.1840, 0.9600]^T$。二者结果作差后的 ∞-范数为 0.3164，误

差不大但也不可忽略。

（3）计算上述扰动的相对误差为 $\frac{dA}{A(n,n)} = 2.6030e^{-5}$，解的相对误差为 $\frac{|x_1-x|}{|x|} = [0.0686, 0.2410, 0.3164, 0.1840, 0.0400]^T$。此时 1-范数、2-范数、$\infty$-范数的条件数分别为 651203.784、357402.357、627547.733。

可以看到，相对误差的大小仍然处在不可忽略的范围内，而该矩阵的条件数量级在 10^5，存在相当的病态性。

（4）利用上述实验分析求解插值函数时使用 Lagerange 和 Newton 插值的原因。

如果使用插值函数存在定理直接求解插值函数，则在计算过程中会产生不可避免的误差，且该误差会随着迭代的次数增多而迅速增长；而使用 Lagerange 和 Newton 插值法的话，可以对该误差起到一定的补偿修正作用，最终多次迭代后的误差也不会过分大。

4.3.5 程序代码

(1) “test5_3.m”

```
clear;                              % clear the residue

n = 5;

A = zeros(n, n);
for i = 1 : 1 : n
    for j = 1 : 1 : n
        A(i, j) = (1 + 0.1 * (i - 1))^(j - 1);
    end
end

% output the cond
c1 = cond(A, 1);
c2 = cond(A, 2);
c3 = cond(A, Inf);
disp(['1 cond = ', num2str(c1), ' 2 cond = ', num2str(c2), ' Infinity
    cond = ', num2str(c3)]);

% calculate the x
b = sum(A, 2);
x = A\b;

% add the error
dA = 0.0001;
A(n, n) = A(n, n) + dA;
```

```
25
26      x1 = A\b;
27      % error = norm(x - x1, Inf);
28
29      dx = (x1 - x)./x;
```

参考文献

[1] John Charles Butcher. A history of runge-kutta methods. *Applied numerical mathematics*, 20(3):247–260, 1996.

[2] 于振廷. 二次样条插值的 matlab 实现. 科技风, (2):166–166, 2017.

[3] 李庆扬. 数值分析. 清华大学出版社有限公司, 2001.

[4] 程鹏飞. 多项式拟合模型病态性问题的分析与应用研究. 测绘通报, (7):35–38, 2012.

[5] 郭永杰. 拉格朗日插值公式病态现象的研究. 水文, (3):31–35, 1996.

[6] 陈光. 最小二乘曲线拟合及 matlab 实现. 兵工自动化, 24(3):107–108, 2005.