数值分析: 实验报告

钟伟　学号: 2021901822　班级: 2021250010

日期：2023 年 5 月 4 日

目录

实验 0 数值稳定性

0.1 问题

讨论两种算法计算积分 $I_k = e^{-1}\int_0^1 x^k e^x \,\mathrm{d}x, (k = 0, 1, \cdots, 22.)$ 的数值稳定性

算法:

(1) $I_k = 1 - kI_{k-1}$,

(2) $I_{k-1} = (1 - I_k)/k$,

0.2 方法

算法 (1) $I_0 = e^{-1}\int_0^1 e^x \,\mathrm{d}x = 1 - \frac{1}{e}$,

算法 (2) $I_k = \int_0^1 x^k e^{-(1-x)} \,\mathrm{d}x < \int_0^1 x^k \,\mathrm{d}x = \frac{1}{k+1} \to 0\ (k \to \infty)$, $I_{k-1} = (1 - I_k)/k$, 取 $I_{22} = 0$.

0.3 程序

Listing 1: Matlab code

```
%% Numerical stability
clear; clc;

MaxN = 22; % 积分序列长度

%% 调用 Matlab 数值积分函数 integral 计算积分序列精确值
fprintf('---- 精确解 begin -----\n')
for k = 0:MaxN
    I = integral(@(x) exp(-1)*x.^k.*exp(x),0,1);
    fprintf('%d: I=%f\n',k,I);
end
fprintf('----- 精确解 end ------\n')

%% Algorithm 1 从前往后计算积分序列
```

```
fprintf('---- Algorithm 1 begin -----\n')
I = 1-1./exp(1);   % 设 I_{0} 的值
fprintf('%d: I=%f\n',0,I);
for k = 1:MaxN
    I = 1 - k*I;
    fprintf('%d: I=%f\n',k,I);
end
fprintf('---- Algorithm 1 end -----\n')

%% Algorithm 2 从后往前计算积分序列
fprintf('---- Algorithm 2 begin -----\n')
I = 0;  % 设 I_{MaxN} 的值为0
vals = zeros(MaxN,1);
vals(MaxN+1) = I;
for k = MaxN:-1:1
    I = (1 - I)./k;
    vals(k) = I;
end

for k = 0:MaxN
    fprintf('%d: I=%f\n',k,vals(k+1));
end
fprintf('---- Algorithm 2 end -----\n')
```

0.4 数值结果

表 1: 迭代结果

k	精确值	算法 1	算法 2
0	0.632121	0.632121	0.632121
1	0.367879	0.367879	0.367879
2	0.264241	0.264241	0.264241
3	0.207277	0.207277	0.207277
4	0.170893	0.170893	0.170893
5	0.145533	0.145533	0.145533
6	0.126802	0.126802	0.126802
7	0.112384	0.112384	0.112384
8	0.100932	0.100932	0.100932
9	0.091612	0.091612	0.091612
10	0.083877	0.083877	0.083877

11	0.077352	0.077352	0.077352
12	0.071773	0.071773	0.071773
13	0.066948	0.066948	0.066948
14	0.062732	0.062731	0.062732
15	0.059018	0.059034	0.059018
16	0.055719	0.055459	0.055719
17	0.052771	0.057192	0.052771
18	0.050120	-0.029454	0.050120
19	0.047723	1.559620	0.047727
20	0.045545	-30.192395	0.045455
21	0.043557	635.040293	0.045455
22	0.041736	-13969.886437	0.000000

0.5 结果分析

- 算法 1 中, 初始步的误差随着计算步数的增加被放大.
- 算法 2 中, 初始步的误差可以再后续的计算中被控制.

因此, 算法 1 是不稳定的, 算法 2 是稳定的.

实验 1　非线性方程求根

1.1　问题

参考例 2.6, 设计相应的计算公式, 用 Newton 迭代法求解下列数的近似, 其中 p_0 为迭代法的初始值.

(1) $p_0 = 3$, 求 $\sqrt{8}$ 的近似值.

(2) $p_0 = 10$, 求 $\sqrt{91}$ 的近似值.

(3) $p_0 = -3$, 求 $-\sqrt{8}$ 的近似值.

(4) $p_0 = 2$, 求 $7^{1/3}$ 的近似值.

(5) $p_0 = 6$, 求 $200^{1/3}$ 的近似值.

(6) $p_0 = -2$, 求 $(-7)^{1/3}$ 的近似值.

1.1.1　方法/算法

构造 Newton 迭代法, 设置为 f, 再计算 f 的导数 df 传入 newton_slove

$Newton$ 迭代法是一种求解方程的数值方法. 给定函数 $f(x)$ 和初始近似解 x_0, 我们可以通过以下的迭代公式来逐步提高解的精度:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \ldots$$

其中 $f'(x)$ 是 $f(x)$ 的一阶导数. 首先计算 $f(x_n)$ 和 $f'(x_n)$, 然后用这两个量来计算 x_{n+1}, 即在当前近似解 x_n 处的切线与 x 轴的交点. 在程序中, 设置最大迭代次数 max_iters 以及误差限 ε, 当达到最大迭代次数或者误差小于误差限 ε 时停止迭代, 输出结果

1.1.2　程序

Listing 2: *Newton Solve*

```
function [x, num_iters] = newton_solve(f, df, x0, tol, max_iters)
% newton_solve 使用牛顿迭代法求解非线性方程f(x)=0。
% code by zhongwei
%   [x, num_iters] = newton_solve(f, df, x0, tol, max_iters)
%   使用初始点X0, 精度TOL和最大迭代次数MAX_ITERS对非线性方程f(x)=0进行求解。
%   F是一个函数句柄, DF是f的导数, 输出X是方程的解, NUM_ITERS是迭代次数。
%
%   示例:
%       f = @(x) x^2 - 2;
%       df = @(x) 2*x;
%       x0 = 1;
%       tol = 1e-6;
%       max_iters = 100;
```

```
%       [x, num_iters] = newton_solve(f, df, x0, tol, max_iters);
%

x = x0;
num_iters = 1;

fprintf('初始值: x0 = %f\n', x0);

while num_iters < max_iters
    % 计算函数值和导数
    f_val = f(x);
    df_val = df(x);

    % 求解牛顿方程
    delta_x = -f_val / df_val;

    % 更新解
    x = x + delta_x;

    % 打印当前迭代的结果
    fprintf('第%d次迭代: x = %.9f, f(x) = %.9f\n', num_iters, x, f_val);

    % 检查是否满足精度要求
    if abs(f_val) < tol
        break
    end

    num_iters = num_iters + 1;
end

fprintf('经过%d次迭代, 得到方程的近似根为: x = %.8f, f(x) = %.8f \n',
    num_iters,x, f(x));

end
```

Listing 3: *Newton Solve* 函数调用

```
% newton_solve函数调用
% code by zhongwei
clear;clc;
%% f = @(x) x^2 - c
% 定义矩阵用于确定方程
X1=[8;91;-8];
% 定义初始值矩阵
```

```
X0_1 = [3;10;-3];
tol = 1e-6;
max_iters = 100;
for k=1:length(X0_1)
    f = @(x) x^2 - abs(X1(k));
    df = @(x) 2*x;
    fprintf("f = x^2 - %d, ",abs(X1(k)));
    [x1(k), numiters_1(k)] = newton_solve(f, df, X0_1(k), tol, max_iters);
end

%% f = @(x) x^3 - c
X2=[7;200;-7];
X0_2 = [2;6;-2];
tol = 1e-6;
max_iters = 100;
for k=1:length(X0_2)
    f = @(x) x^3 - X2(k);
    df = @(x) 3*x^2;
    fprintf("f = x^2 - %d, ",X2(k));
    [x2(k), numiters_2(k)] = newton_solve(f, df, X0_2(k), tol, max_iters);
end
```

1.1.3 数值结果

表 2: $\sqrt{8}$ 的近似值

$iter$	x	$f(x)$
1	2.833333333	1.000000000
2	2.828431373	0.027777778
3	2.828427125	0.000024029
4	2.828427125	0.000000000

表 3: $\sqrt{91}$ 的近似值

$iter$	x	$f(x)$
1	9.550000000	9.000000000
2	9.539397906	0.202500000
3	9.539392014	0.000112404
4	9.539392014	0.000000000

表 4: $-\sqrt{8}$ 的近似值

$iter$	x	$f(x)$
1	-2.833333333	-1.000000000
2	-2.828431373	-0.027777778
3	-2.828427125	-0.000024029
4	-2.828427125	0.000000000

表 5: $7^{\frac{1}{3}}$ 的近似值

$iter$	x	$f(x)$
1	1.916666667	1.000000000
2	1.912938458	0.041087963
3	1.912931183	0.000079871
4	1.912931183	0.000000000

表 6: $200^{\frac{1}{3}}$ 的近似值

$iter$	x	$f(x)$
1	5.851851852	16.000000000
2	5.848037965	0.391810192
3	5.848035476	0.000255303
4	5.848035476	0.000000000

表 7: $(-7)^{\frac{1}{3}}$ 的近似值

$iter$	x	$f(x)$
1	-1.916666667	-1.000000000
2	-1.912938458	-0.041087963
3	-1.912931183	-0.000079871
4	-1.912931183	0.000000000

1.1.4 结果分析

$Newton$ 迭代法具有较好的收敛性, 至少具有二阶收敛速度. 在以上题目中收敛速度较快, 均是 4 次迭代后达到精度要求.

$Newton$ 迭代法的收敛性可以通过函数的局部性质和 $f'(x_n)$ 不为 0 的条件来保证. 具体来说, 如果 $f(x)$ 是连续可导的, 并且在 x_0 的某个邻域内,$f'(x)$ 不为 0, 那么 $Newton$ 迭代法通常会

收敛到方程的一个根附近. 收敛速度通常是平方级别的, 也就是说, 每次迭代后, 误差的平方会减少一定的比例.

$Newton$ 迭代法也有一些限制. 首先, 如果 $f'(x_n)$ 在某个迭代步骤中变得非常小, 那么分母就会变得非常小, 从而导致迭代结果出现较大误差, 甚至发散. 其次, 如果 x_0 选择不当, 可能会导致迭代跳出函数的定义域, 或者陷入周期性迭代. 此外,$Newton$ 迭代法也无法保证收敛到全局最优解, 因为收敛点可能取决于初始值的选择.

因此,$Newton$ 迭代法在实际应用中需要谨慎使用. 在选择初始值时, 通常需要通过一些启发式方法来选择一个好的初始值, 以确保算法的稳定性和收敛性. 此外, 在实际应用中, 还需要对算法的数值稳定性和收敛速度进行评估和优化, 以便更好地应用该算法来解决实际问题.

实验 2　线性方程组的解法

2.1　高斯顺序消去法

2.1.1　问题

用**高斯顺序消去法**求解以下三个线性方程组:

$$\begin{cases} 4x_1 + 8x_2 + 4x_3 + 0x_4 = 8, \\ x_1 + 5x_2 + 4x_3 - 3x_4 = -4, \\ x_1 + 4x_2 + 7x_3 + 2x_4 = 10, \\ x_1 + 3x_2 + 0x_3 - 2x_4 = -4, \end{cases} \tag{2.1}$$

$$\begin{cases} 2x_1 + 4x_2 - 4x_3 + 0x_4 = 12, \\ x_1 + 5x_2 - 5x_3 - 3x_4 = 18, \\ 2x_1 + 3x_2 + x_3 + 3x_4 = 8, \\ x_1 + 4x_2 - 2x_3 + 2x_4 = 8, \end{cases} \tag{2.2}$$

$$\begin{cases} x_1 + 2x_2 + 0x_3 - x_4 = 9, \\ 2x_1 + 3x_2 - x_3 + 0x_4 = 9, \\ 0x_1 + 4x_2 + 2x_3 - 5x_4 = 26, \\ 5x_1 + 5x_2 + 2x_3 - 4x_4 = 32. \end{cases} \tag{2.3}$$

2.1.2　方法/算法

Gauss 顺序消元法主要有消元和回代两个部分, 将增广矩阵 $[\boldsymbol{A}|\boldsymbol{b}]$ 进行高斯消元, 得到一个阶梯形矩阵, 即 $[\boldsymbol{A'}|\boldsymbol{b'}]$, 其中 $\boldsymbol{A'}$ 为增广矩阵的左侧部分，$\boldsymbol{b'}$ 为增广矩阵的右侧部分. 从最后一行开始, 依次回代求解方程组中的每一个未知量 $\boldsymbol{x}[i], i = n, n-1, ..., 1$. 对于第 i 行, 先将已知的未知量代入方程, 解出 $\boldsymbol{x}[i]$, 然后依次向上回代, 得到剩余未知量的值, 重复上述步骤, 直到求出所有未知量 $\boldsymbol{x}[i]$

2.1.3　程序

Listing 4: *Gauss* 顺序消去法

```
function x = gauss_sequential(A, b)
% GAUSS_SEQUENTIAL 高斯顺序消元法求解线性方程组
% code by zhongwei
%   x = GAUSS_SEQUENTIAL(A, b) 使用高斯顺序消元法求解系数矩阵 A 和
%   常数向量 b 所表示的线性方程组 Ax = b, 并返回方程组的解 x。
%
%   示例:
%       A = [2 1 -1; -3 -1 2; -2 1 2];
```

```
%       b = [8; -11; -3];
%       x = gauss_sequential(A, b);
%       disp(x); % 输出解向量

n = length(b);

% 消元过程
for k = 1:n-1
    % 选取主元
    for i = k+1:n
        factor = A(i,k) / A(k,k);
        A(i,k:n) = A(i,k:n) - factor * A(k,k:n);
        b(i) = b(i) - factor * b(k);
    end
end

% 回代过程
x = zeros(n, 1);
x(n) = b(n) / A(n,n);
for i = n-1:-1:1
    x(i) = (b(i) - A(i,i+1:n)*x(i+1:n)) / A(i,i);
end

% 显示消元后的矩阵
disp('消元后的矩阵: ');
disp(A);
% 输出解向量
disp('解向量为: ')
disp(x);
end
```

Listing 5: *Gauss* 顺序消去法函数调用

```
% 高斯顺序消元法函数调用
% code by zhongwei
clear;clc;
A = [4 8 4 0;
    1 5 4 -3;
    1 4 7 2;
    1 3 0 -2];
b = [8;-4;10;-4];
x = gauss_sequential(A, b);
```

2.1.4 数值结果

(2.1) • 消元后的矩阵为:

$$\begin{pmatrix} 4 & 8 & 4 & 0 \\ 0 & 3 & 3 & -3 \\ 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

• 解向量为:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 3 \\ -1 \\ 1 \\ 2 \end{pmatrix}$$

(2.2) • 消元后的矩阵为:

$$\begin{pmatrix} 2 & 4 & -4 & 0 \\ 0 & 3 & -3 & -3 \\ 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 3 \end{pmatrix}$$

• 解向量为:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 1 \\ -2 \end{pmatrix}$$

(2.3) • 消元后的矩阵为:

$$\begin{pmatrix} 1 & 2 & 0 & -1 \\ 0 & -1 & -1 & 2 \\ 0 & 0 & -2 & 3 \\ 0 & 0 & 0 & \frac{3}{2} \end{pmatrix}$$

• 解向量为:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 2 \\ -2 \end{pmatrix}$$

2.2 直接三角分解法

2.2.1 问题

用 **Doolittle 分解法**求解下面的线性方程组:

$$\begin{cases} x_1 + 2x_2 + 4x_3 + x_4 = 21, \\ 2x_1 + 8x_2 + 6x_3 + 4x_4 = 52, \\ 3x_1 + 10x_2 + 8x_3 + 8x_4 = 79, \\ 4x_1 + 12x_2 + 10x_3 + 6x_4 = 82. \end{cases} \tag{2.4}$$

2.2.2 方法/算法

设 A 是一个 $n \times n$ 的矩阵,Doolittle 分解将 $\boldsymbol{A}$ 分解为 $\boldsymbol{A} = \boldsymbol{LU}$, 其中 $\boldsymbol{L}$ 是一个下三角矩阵, 而 $\boldsymbol{U}$ 是一个上三角矩阵.$\boldsymbol{L}$ 和 $\boldsymbol{U}$ 可以通过以下方式计算得出: 首先对于 $\boldsymbol{A}$ 的第一行和第一列元素有

$$\begin{cases} a_{1j} = u_{1j} & (j = 1, 2, \cdots, n) \\ a_{i1} = l_{i1} \cdot u_{11} & (i = 1, 2, \cdots, n) \end{cases}$$

所以矩阵 $\boldsymbol{U}$ 的第一行和 $\boldsymbol{L}$ 的第一列元素分别为

$$\begin{cases} u_{1j} = a_{1j} & (j = 1, 2, \cdots, n) \\ l_{i1} = \dfrac{a_{i1}}{u_{11}} & (i = 1, 2, \cdots, n) \end{cases}$$

一般地，设矩阵 $\boldsymbol{U}$ 的前 $k-1$ 行和矩阵 $\boldsymbol{L}$ 的前 $k-1$ 列元素已经求出.
则由矩阵乘法, 让第 k 行元素, 第 k 列元素对应相等, 得

$$\begin{cases} a_{kj} = \sum\limits_{m=1}^{k-1} l_{km} u_{mj} + u_{kj} & (j = k, k+1, \cdots, n), \\ a_{ik} = \sum\limits_{m=1}^{k-1} l_{im} u_{mk} + l_{ik} u_{kk} & (i = k+1, k+2, \cdots, n), \end{cases}$$

于是求得矩阵 $\boldsymbol{U}$ 的第 k 行和矩阵 $\boldsymbol{L}$ 的第 k 列元素为

$$\begin{cases} u_{kj} = a_{kj} - \sum\limits_{m=1}^{k-1} l_{km} u_{mj} & (j = k, k+1, \cdots, n), \\ l_{ik} = \dfrac{1}{u_{kk}} (a_{ik} - \sum\limits_{m=1}^{k-1} l_{im} u_{mk}) & (i = k+1, k+2, \cdots, n), \end{cases} \quad (k = 2, 3, \cdots, n).$$

2.2.3 程序

Listing 6: *Doolittle* 分解法

```
function [x,y] = doolittle(A, b)
% 使用 Doolittle 分解法求解线性方程组 Ax=b
% code by zhongwei
% 输入: A是一个 n×n 矩阵, b 是一个 n×1 向量
% 输出: x 是一个 n×1 向量, 满足 Ax=b

% 确定矩阵的大小
n = size(A, 1);

% 初始化 L 和 U 矩阵
L = eye(n);
U = zeros(n);

% 进行分解
for k = 1:n
    % 计算 U 的第 k 行
    for j = k:n
        U(k,j) = A(k,j) - L(k,1:k-1)*U(1:k-1,j);
    end

    % 计算 L 的第 k 列
    for i = k+1:n
        L(i,k) = (A(i,k) - L(i,1:k-1)*U(1:k-1,k))/U(k,k);
    end
end

% 解方程组 Ly=b 和 Ux=y
y = zeros(n, 1);
for i = 1:n
    y(i) = b(i) - L(i,1:i-1)*y(1:i-1);
end
x = zeros(n, 1);
for i = n:-1:1
    x(i) = (y(i) - U(i,i+1:n)*x(i+1:n))/U(i,i);
end
disp('L: ');
disp(L);
disp('U: ');
disp(U);
end
```

Listing 7: *Doolittle* 分解法函数调用

```
% doolittle函数调用
% code by zhongwei
% 系数矩阵
A = [1 2 4 1;
     2 8 6 4;
     3 10 8 8;
     4 12 10 6];
b = [21;52;79;82];
doolittle(A,b);
```

2.2.4 数值结果

Doolittle 分解的结果为:

$$\boldsymbol{L}=\begin{pmatrix}1&0&0&0\\2&1&0&0\\3&1&1&0\\4&1&2&1\end{pmatrix},\quad \boldsymbol{U}=\begin{pmatrix}1&2&4&1\\0&4&-2&2\\0&0&-2&3\\0&0&0&-6\end{pmatrix}$$

解向量为:

$$\begin{pmatrix}x_1\\x_2\\x_3\\x_4\end{pmatrix}=\begin{pmatrix}1\\2\\3\\4\end{pmatrix}$$

2.3 平方根方法

2.3.1 问题

用**平方根方法**求解下面的线性方程组:

$$\begin{cases}4x_1-2x_2-4x_3+3x_4=10,\\2x_1-17x_2-10x_3-6x_4=-3,\\4x_1-10x_2-9x_3-2x_4=7,\\3x_1+6x_2+2x_3+15x_4=10.\end{cases}\tag{2.5}$$

2.3.2 方法/算法

注意: 平方根方法的使用前提.

定理 3.2 n 阶 ($n\geq 2$) 对称正定矩阵 $\boldsymbol{A}$ 一定有 Cholesky 分解 $\boldsymbol{A}=\boldsymbol{L}\boldsymbol{L}^{\boldsymbol{T}}$. 当限定 $\boldsymbol{L}$ 的对角元全为正时, 该分解是唯一的.

考虑到平方根法使用的前提, 对系数矩阵进行变换, 使系数矩阵变为对称正定矩阵, 才能满足平方根法的使用条件, 变换后的线性方程组如下:

$$\begin{cases} 4x_1 - 2x_2 - 4x_3 + 3x_4 = 10, \\ -2x_1 + 17x_2 + 10x_3 + 6x_4 = 3, \\ -4x_1 + 10x_2 + 9x_3 + 2x_4 = -7, \\ 3x_1 + 6x_2 + 2x_3 + 15x_4 = 10. \end{cases}$$

由矩阵乘法可知 $a_{11} = l_{11}^2, a_{i1} = l_{i1} \cdot l_{11}$, 进而求得

$$\begin{cases} l_{11} = \sqrt{a_{11}} \\ l_{i1} = \dfrac{a_{i1}}{l_{11}} \end{cases} \qquad (i = 2, 3, \cdots, n)$$

一般地, 设矩阵 $\boldsymbol{L}$ 的前 $k-1$ 列元素已经求出. 则由矩阵乘法知

$$\begin{cases} a_{kk} = \sum\limits_{m=1}^{k-1} l_{km}^2 + l_{kk}^2 \\ a_{ik} = \sum\limits_{m=1}^{k-1} l_{im} l_{km} + l_{ik} l_{kk} \end{cases} \qquad (i = k+1, k+2, \cdots, n).$$

于是

$$\begin{cases} l_{kk} = \sqrt{a_{kk} - \sum\limits_{m=1}^{k-1} l_{km}^2}, \\ l_{ik} = \dfrac{1}{l_{kk}}(a_{ik} - \sum\limits_{m=1}^{k-1} l_{im} l_{km}) \qquad (i = k+1, k+2, \cdots, n) \end{cases} \qquad (k = 2, 3, \cdots, n).$$

2.3.3 程序

Listing 8: *Cholesky* 分解法

```
function [x] = cholesky(A,b)
% 使用 Cholesky 分解法求解线性方程组 Ax=b
%code by zhongwei
% A是一个 n×n 矩阵, b 是一个 n×1 向量
% x - 解向量
% 首先判断输入矩阵是不是对称正定矩阵, 如果不是则输出错误语句
if isequal(A,A') && all(eig(A) > 0)
    n = length(b);
    L = zeros(n,n);
    for j=1:n
        % 此时第j行主对角线元素还未赋值, 但主对角线元素前的元素已被赋值
        L(j,j) = sqrt(A(j,j) - L(j,:)*L(j,:)');
```

```
        for i=j+1:n
            L(i,j) = (A(i,j) - L(i,:)*L(j,:)')/L(j,j);
        end
    end
    disp(L)
    y = zeros(n,1);
    for i=1:n
        y(i) = (b(i) - L(i,:)*y)/L(i,i);
    end
    x = zeros(n,1);
    for i=n:-1:1
        x(i) = (y(i) - sum(L(i+1:n,i).*x(i+1:n)))/L(i,i);
    end
    disp('Y=')
    disp(y)
    disp('X=')
    disp(x)
else
    error('Input matrix is not symmetric positive definite')
end
```

Listing 9: *Cholesky* 分解法函数调用

```
% Cholesky函数调用
% code by zhongwei
clear;clc;
A = [4 -2 -4 3;
    -2 17 10 6;
    -4 10 9 2;
     3 6 2 15];
b = [10;3;-7;10];
cholesky(A,b);
```

2.3.4 数值结果

Cholesky 分解的结果为:

$$\boldsymbol{L} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ -1 & 4 & 0 & 0 \\ -2 & 2 & 1 & 0 \\ \frac{3}{2} & \frac{15}{8} & \frac{5}{4} & \frac{2551}{921} \end{pmatrix}$$

解向量为:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ -1 \\ 0 \end{pmatrix}$$

2.4 线性方程组的迭代法

2.4.1 问题

(1) 实现 Jacobi 迭代法、JGS 迭代法.

(2) 调用上面问题中实现的程序复现教材的例题 3.4、3.5.

(3) 调用上面问题中实现的程序求解习题三第 14 题.

2.4.2 方法/算法

注意: 程序实现建议使用分量形式, 不要使用矩阵形式. Jacobi 迭代法是一种用于解线性方程组的迭代方法, 其基本思想是将线性方程组的系数矩阵分解为对角矩阵和剩余矩阵的和, 然后通过对角矩阵的逆矩阵和剩余矩阵不断交替迭代, 直至收敛于方程组的解.

给定一个 n 元线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$, 其中 $\boldsymbol{A}$ 是一个 $n \times n$ 的矩阵, $\boldsymbol{x}$ 和 $\boldsymbol{b}$ 都是 n 维列向量,Jacobi 迭代法的求解过程如下:

1. 对矩阵 $\boldsymbol{A}$ 进行分解, 得到 $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$, 其中 $\boldsymbol{D}$ 是 $\boldsymbol{A}$ 的对角线矩阵,$\boldsymbol{L}$ 是 $\boldsymbol{A}$ 的严格下三角矩阵, $\boldsymbol{U}$ 是 $\boldsymbol{A}$ 的严格上三角矩阵.

2. 将线性方程组转化为 $\boldsymbol{Dx} = (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x} + \boldsymbol{b}$ 的形式.

3. 初始化迭代向量 $\boldsymbol{x}^{(\boldsymbol{0})}$.

4. 对于迭代次数 $k = 0, 1, 2, ...$, 执行以下操作:

   a. 计算 $\boldsymbol{x}^{(\boldsymbol{k+1})} = \boldsymbol{D}^{-\boldsymbol{1}}(\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}^{(\boldsymbol{k})} + \boldsymbol{D}^{-\boldsymbol{1}}\boldsymbol{b}$.

   b. 判断 $\boldsymbol{x}^{(\boldsymbol{k+1})}$ 是否满足一定的精度要求, 如果满足则输出 $\boldsymbol{x}^{(\boldsymbol{k+1})}$, 否则继续迭代.

Jacobi 迭代法的收敛性要求矩阵 $\boldsymbol{A}$ 满足对角占优条件, 即对于任意 $i \in \{1, 2, ..., n\}$, 有

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|$$

如果矩阵 $\boldsymbol{A}$ 不满足对角占优条件,Jacobi 迭代法可能无法收敛, 或者收敛速度很慢.

JGS 迭代法是一种用于解线性方程组的迭代方法, 其基本思想是将系数矩阵分解为对称三角矩阵和一个余下部分, 然后通过交替使用两个不同的三角矩阵进行迭代, 直至收敛于方程组的解.

给定一个 n 元线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$, 其中 $\boldsymbol{A}$ 是一个 $n \times n$ 的矩阵, $\boldsymbol{x}$ 和 $\boldsymbol{b}$ 都是 n 维列向量, JGS 迭代法的求解过程如下:

1. 对矩阵 $\boldsymbol{A}$ 进行分解, 得到 $\boldsymbol{A} = \boldsymbol{B} - \boldsymbol{C}$, 其中 $\boldsymbol{B}$ 是 $\boldsymbol{A}$ 的对称三角矩阵, $\boldsymbol{C}$ 是 $\boldsymbol{A}$ 的余下部分.

2. 将线性方程组转化为 $\boldsymbol{Bx} = \boldsymbol{Cx} + \boldsymbol{b}$ 的形式.

3. 初始化迭代向量 $\boldsymbol{x}^{(\boldsymbol{0})}$.

4. 对于迭代次数 $k = 0, 1, 2, ...$, 执行以下操作:

   a. 计算 $\boldsymbol{x}^{(k+1)} = \boldsymbol{B}^{-1}\boldsymbol{C}\boldsymbol{x}^{(k)} + \boldsymbol{B}^{-1}\boldsymbol{b}$.

   b. 判断 $\boldsymbol{x}^{(k+1)}$ 是否满足一定的精度要求, 如果满足则输出 $\boldsymbol{x}^{(k+1)}$, 否则继续迭代.

5. 如果迭代次数达到了最大迭代次数但仍未满足精度要求, 则输出近似解.

JGS 迭代法的收敛性要求矩阵 $\boldsymbol{A}$ 为正定对称矩阵或者严格对角占优的对称矩阵. 如果矩阵 $\boldsymbol{A}$ 不满足这些条件,JGS 迭代法可能无法收敛, 或者收敛速度很慢.

2.4.3 程序

Listing 10: *Jacobi* 迭代法

```
function [x, iter] = jacobi(A, b, x0, epslion, maxiter)
% JACOBI使用Jacobi迭代法解线性方程组Ax = b
% code by zhongwei
%   [x, iter] = JACOBI(A, b, x0, tol, maxiter) 返回解向量x和迭代次数iter,
    直到解向量的误差小于tol,
%   或者迭代次数达到最大迭代次数maxiter为止。
%
%   A是大小为n x n的方阵。
%   b是长度为n的向量。
%   x0是初始解向量。
%   tol是解向量误差的期望容限。
%   maxiter是允许的最大迭代次数。

n = length(b); % 获取向量b的长度
iter = 0; % 将迭代计数器初始化为0
error = 1;
x = zeros(n,1);

while iter < maxiter && error > epslion % 在最大迭代次数内循环
    for i = 1:n % 循环遍历解向量中的每个元素
        s = 0;
        for j = 1:i-1
            s = s + A(i,j) * x0(j);
        end
        for j = i+1:n
            s = s + A(i,j) * x0(j);
        end
        x(i) = (-s + b(i))/A(i,i);
    end
    error = norm(x-x0,'inf');
    iter = iter + 1;
    x0 = x;
```

```
    fprintf('%2d  %.6e  x=(%1.7f %1.7f %1.7f)\n',iter,error,x.');
end
end
```

Listing 11: *JGS* 迭代法

```
function [x, iter] = jgs(A, b, x0, epslion, maxiter)
% JGS使用JGS迭代法解线性方程组Ax = b
% code by zhongwei
%   [x, iter] = JGS(A, b, x0, tol, maxiter) 返回解向量x和迭代次数iter, 直到解向量的误差小于tol,
%   或者迭代次数达到最大迭代次数maxiter为止。
%
%   A是大小为n x n的方阵。
%   b是长度为n的向量。
%   x0是初始解向量。
%   tol是解向量误差的期望容限。
%   maxiter是允许的最大迭代次数。

n = length(b); % 获取向量b的长度
iter = 0; % 将迭代计数器初始化为0
error = 1;
x = zeros(n,1);

while iter < maxiter && error > epslion % 在最大迭代次数内循环
    for i = 1:n % 循环遍历解向量中的每个元素
        s = 0;
        for j = 1:i-1
            s = s + A(i,j) * x(j);
        end
        for j = i+1:n
            s = s + A(i,j) * x0(j);
        end
        x(i) = (-s + b(i))/A(i,i);
    end
    error = norm(x-x0,'inf');
    iter = iter + 1;
    x0 = x;
    fprintf('%2d  %.6e  x=(%1.6f %1.6f %1.6f)\n',iter,error,x.');
end
end
```

Listing 12: *Jacobi&JGS* 迭代法函数调用

```
% Jacobi,JGS函数调用
% code by zhongwei
% 输入系数矩阵
clear;clc;
A = [5 -1 1;
     1 -10 -2;
    -1  2 10];
b = [10; 27; 13];
% 设置迭代初始值
x0 = [0; 0; 0];
epslion = 1e-3;
maxiter = 100;
fprintf("Jacbi迭代法: \n");
[x_jacobi, iter_jacobi] = jacobi(A, b, x0, epslion, maxiter);
fprintf("JGS迭代法: \n");
[x_jgs, iter_jgs] = jgs(A, b, x0, epslion, maxiter);
```

2.4.4 数值结果

表 8: ***Jacobi*** 迭代法的计算结果 (例题 3.4)

k	ε	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	2.700000e+00	2.0000000	-2.7000000	1.3000000
2	8.000000e-01	1.2000000	-2.7600000	2.0400000
3	2.280000e-01	1.0400000	-2.9880000	1.9720000
4	3.200000e-02	1.0080000	-2.9904000	2.0016000
5	9.120000e-03	1.0016000	-2.9995200	1.9988800
6	1.280000e-03	1.0003200	-2.9996160	2.0000640
7	3.648000e-04	1.0000640	-2.9999808	1.9999552

表 9: ***JGS*** 迭代法的计算结果 (例题 3.5)

k	ε	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	2.500000e+00	2.000000	-2.500000	2.000000
2	9.000000e-01	1.100000	-2.990000	2.008000
3	9.960000e-02	1.000400	-3.001560	2.000352
4	1.451360e-03	0.999618	-3.000109	1.999983
5	3.639744e-04	0.999982	-2.999999	1.999998

表 10: $Jacobi$ 迭代法的计算结果 (习题 3.14)

k	ε	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	2.000000e+00	0.3000000	1.5000000	2.0000000
2	6.600000e-01	0.8000000	1.7600000	2.6600000
3	2.040000e-01	0.9180000	1.9260000	2.8640000
4	9.000000e-02	0.9716000	1.9700000	2.9540000
5	2.832000e-02	0.9894000	1.9897200	2.9823200
6	1.144800e-02	0.9961760	1.9961120	2.9937680
7	3.912000e-03	0.9985992	1.9986120	2.9976800
8	1.484640e-03	0.9994904	1.9994878	2.9991646
9	5.285760e-04	0.9998140	1.9998145	2.9996932
10	1.954080e-04	0.9999322	1.9999321	2.9998886
11	7.067328e-05	0.9999753	1.9999753	2.9999593
12	2.588371e-05	0.9999910	1.9999910	2.9999852
13	9.412224e-06	0.9999967	1.9999967	2.9999946

表 11: JGS 迭代法的计算结果 (习题 3.14)

k	ε	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	2.684000e+00	0.300000	1.560000	2.684000
2	5.804000e-01	0.880400	1.944480	2.953872
3	1.038832e-01	0.984283	1.992244	2.993754
4	1.354099e-02	0.997824	1.998940	2.999141
5	1.877959e-03	0.999702	1.999855	2.999882
6	2.569835e-04	0.999959	1.999980	2.999984
7	3.526606e-05	0.999994	1.999997	2.999998
8	4.836669e-06	0.999999	2.000000	3.000000

实验 3　函数插值

3.1　Lagrange 插值逼近

3.1.1　问题

对一下给定函数:

(a) $f(x) = \mathrm{e}^x$,

(b) $f(x) = \sin(x)$,

(c) $f(x) = (x+1)^{x+1}$,

构造 5 次插值多项式 $P(x)$ 过 6 个点 $(0, f(0)), (0.2, f(0.2)), (0.4, f(0.4)), (0.6, f(0.6)), (0.8, f(0.8)), (1, f(1))$. 其中

$$P(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0.$$

利用程序计算以下问题

(1) 确定 $P(x)$ 的系数 $\{a_k\}_{k=0}^{5}$.

(2) 计算内插值 $P(0.3), P(0.4)$ 和 $P(0.5)$, 并与 $f(0.3), f(0.4)$ 和 $f(0.5)$ 比较.

(3) 计算外插值 $P(-0.1)$ 和 $P(1.1)$, 并与 $f(-0.1)$ 和 $f(1.1)$ 比较.

3.1.2　方法

Lagrange 插值是一种基于多项式的逼近方法, 用于构造一个函数 $f(x)$, 它经过给定的 $n+1$ 个点 $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, 即满足:

$$f(x_i) = y_i, \quad i = 0, 1, \ldots, n.$$

Lagrange 插值多项式的形式为:

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

其中,

$$l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

Lagrange 插值法的步骤如下:

1. 给定 $n+1$ 个点 $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$.

2. 构造 $n+1$ 个基函数 $l_i(x)$, 其中 $i = 0, 1, \ldots, n$, 并计算 $l_i(x)$ 在点 x_j 处的取值, 即 $l_i(x_j)$, 其中 $j = 0, 1, \ldots, n$.

3. 根据基函数 $l_i(x)$ 和点 (x_i, y_i), 构造插值多项式 $L_n(x)$.

4. 计算插值多项式 $L_n(x)$ 在给定点 x 处的值 $f(x)$.

3.1.3 程序

Listing 13: *Lagrange* 插值

```
function [L,P] = lagrange_interpolation(x, y)
% LAGRANGE_INTERPOLATIOPN 计算使用Lagrange插值方法得到的一组数据点的插值多项式。
% code by zhongwei
% the last edition
% x是一列x坐标，y是相应的y坐标。
% P用来存储lagrange基函数
% [L,P] = LAGRANGE_INTERPOLATIOPN(x, y) 返回Lagrange插值多项式L,P。
%   示例:
%       x = [144;169;196];
%       y = [12;13;14];
% 计算Lagrange插值多项式和点值多项式
%       [L,P] = lagrange_interpolation(x, y);
% 显示Lagrange插值多项式
%       disp(L);
%展示未化简的P,即未化简的Lagrange插值多项式
%       disp(P)
% 将X的值代入求值
%       X = 165;
%       Y = subs(L,X);
%       fprintf('f(x) = %.6f\n', Y);

% 获取数据点的数量
n = length(x);
% 计算点值多项式
syms X;
P = 0;
for k = 1:n
    Lk = 1;
    for j = 1:n
        if j ~= k
            Lk = Lk * (X - x(j)) / (x(k) - x(j));
        end
    end
    P = P + y(k) * Lk;
end
% 展示未化简的P,即未化简的Lagrange插值多项式
%disp(P)
% 返回插值多项式
L = simplify(P);
```

```
% 以小数形式显示
% L = vpa(L, 6);
end
```

Listing 14: lagrange_interpolation 函数调用

```
clear;clc;
format long;
x = linspace(0,1,6);
y = (x+1).^(x+1);
% 计算Lagrange插值多项式和点值多项式
[L,P] = lagrange_interpolation(x, y);
% 显示Lagrange插值多项式
% 以小数形式显示
L = expand(L);
L = vpa(L, 8);
disp(L);
%展示未化简的P,即未化简的Lagrange插值多项式
%disp(P)
% 在X处评估插值多项式
X = [-0.1,0.3,0.4,0.5,1.1];
% Y0用来存放f(x)的值,Y用来存放P(x)的值
Y = [];
Y0 = [];
for k =1:length(X)
    Y(k) = subs(L,X(k));
    Y0(k) = (X(k)+1).^(X(k)+1);
    fprintf('x=%f,P(x) = %.8f\n', X(k),Y(k));
    fprintf('x=%f,f(x) = %.8f\n', X(k),Y0(k));
end
```

3.1.4 数值结果

(a) $f(x) = \mathrm{e}^x$

(1) $P(x) = 0.01385428x^5 + 0.03486645x^4 + 0.170410x^3 + 0.4990688x^2 + 1.000083x^1 + 1$

(2) $f(x) = \mathrm{e}^x$ 内插值的结果如下表所示:

表 12: $f(x) = \mathrm{e}^x$ 内插值

x	$P(x)$	$f(x)$
0.3	1.349858102	1.349858808
0.4	1.491824698	1.491824698
0.5	1.648721789	1.648721271

(3) $f(x) = \mathrm{e}^x$ 外插值的结果如下表所示:

表 13: $f(x) = \mathrm{e}^x$ 外插值

$\boldsymbol{x}$	$\boldsymbol{P(x)}$	$\boldsymbol{f(x)}$
-0.1	0.904815372	0.904837418
1.1	3.004139857	3.004166024

(b) $f(x) = \sin(x)$,

(1) $P(x) = 0.0072524778x^5 + 0.0016129607x^4 - 0.16761639x^3 + 0.00024391539x^2 + 0.99997802x$

(2) $f(x) = \sin(x)$ 内插值的结果如下表所示:

表 14: $f(x) = \sin(x)$ 内插值

$\boldsymbol{x}$	$\boldsymbol{P(x)}$	$\boldsymbol{f(x)}$
0.3	0.295520404	0.295520207
0.4	0.389418342	0.389418342
0.5	0.479425390	0.479425539

(3) $f(x) = \sin(x)$ 外插值的结果如下表所示:

表 15: $f(x) = \sin(x)$ 外插值

$\boldsymbol{x}$	$\boldsymbol{P(x)}$	$\boldsymbol{f(x)}$
-0.1	-0.099827658	-0.099833417
1.1	0.891215270	0.891207360

(c) $f(x) = (x+1)^{x+1}$,

(1) $P(x) = 0.39453072x^5 - 0.07171214x^4 + 0.73040131x^3 + 0.94153747x^2 + 1.0052426x + 1$

(2) $f(x) = (x+1)^{x+1}$ 内插值的结果如下表所示:

表 16: $f(x) = (x+1)^{x+1}$ 内插值

$\boldsymbol{x}$	$\boldsymbol{P(x)}$	$\boldsymbol{f(x)}$
0.3	1.406409840	1.406456673
0.4	1.601692898	1.601692898
0.5	1.837152927	1.837117307

(3) $f(x) = (x+1)^{x+1}$ 外插值的结果如下表所示:

表 17: $f(x) = (x+1)^{x+1}$ 外插值

x	$P(x)$	$f(x)$
-0.1	0.908149593	0.909532576
1.1	4.747593312	4.749638092

3.1.5 结果分析

内插值的所得结果与真实值之间的差异与外插值所得结果与真实值之间的差异相比, 内插值结果差异较小

3.2 Newton 插值逼近

3.2.1 问题

利用 Newton 插值法计算习题四第 3 题.

3.2.2 方法

Newton 插值是一种基于差商的逼近方法, 用于构造一个函数 $f(x)$, 它经过给定的 $n+1$ 个点 $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, 即满足:

$$f(x_i) = y_i, \quad i = 0, 1, \ldots, n.$$

Newton 插值多项式的形式为:

$$N_n(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \cdots + b_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}),$$

其中,

$$b_0 = y_0, \quad b_k = \frac{\Delta^k y_0}{k!}, \quad k = 1, 2, \ldots, n,$$

其中 $\Delta^k y_0$ 表示从 y_0 开始的 k 阶差分, 即

$$\Delta y_0 = y_1 - y_0, \quad \Delta^2 y_0 = \Delta(\Delta y_0) = \Delta y_1 - \Delta y_0 = y_2 - 2y_1 + y_0, \quad \ldots$$

Newton 插值法的步骤如下:

1. 给定 $n+1$ 个点 $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$.
2. 计算并存储 $n+1$ 个差商 $\Delta^k y_0$, 其中 $k = 0, 1, \ldots, n$.
3. 根据差商 $\Delta^k y_0$ 和点 (x_0, y_0), 构造插值多项式 $N_n(x)$.
4. 计算插值多项式 $N_n(x)$ 在给定点 x 处的值 $f(x)$.

此题中将 $y = f(x)$ 转化为 $x = f^{-1}(y)$, 进而求 $f(x) = 0$ 的近似根就可以转化成 $x = f^{-1}(0)$ 的近似值, 进一步构建差商表, 利用差商表可以确定 $Newton$ 插值多项式的系数, 最终得到 $Newton$ 插值多项式.

3.2.3 程序

Listing 15: *Newton* 插值

```
function [D, N] = newton_interpolation(x, y)
% NEWTON_INTERPOLATIOPN计算使用差商方法得到的一组数据点的Newton插值多项式。
% code by zhongwei
% the last edition
% x是一列x坐标，y是相应的y坐标。
%
% [D, N] = NEWTON_INTERPOLATIOPN(x, y) 返回差分表D和插值多项式N。
%   示例:
%       x = [-1.1;-0.5;0.9;1.7];
%       y = [-1.12;0;1.80;2.20];
%       [D, N] = newton_interpolation(x, y);
%       disp(D);
%       disp(N);
%       X = 0;
%       Y = subs(N,X);
%       fprintf('f(x) = %.6f\n', Y);

% 获取数据点的数量
n = length(x);
% 将差分表初始化为零
D = zeros(n);
% 使用y坐标填充差分表的第一列
D(:,1) = y;
% 使用递推公式计算差商
for j = 2:n
    for i = j:n
        D(i,j) = (D(i,j-1) - D(i-1,j-1)) / (x(i) - x(i-j+1));
    end
end
% 使用差商计算插值多项式
N = D(1,1);
syms X;
for k = 2:n
    c = prod(X - x(1:k-1));
    N = N + D(k,k) * c;
end
% 以小数形式显示
% N = vpa(N, 6);
end
```

Listing 16: newton_interpolation 函数调用

```
x = [-1.15;-0.87;1.0;2.21];
y = [-0.8;0;1.2;1.8];
[D, N] = newton_interpolation(x, y);
disp(D);
% 以小数形式显示,保留6为小数
N = vpa(N, 6);
disp(N);
% 为Newton插值多项式赋值
X = 0;
Y = subs(N,X);
fprintf('f(x) = %.6f\n', Y);
```

3.2.4 数值结果

差商表如下表所示:

表 18: 差商表

$f(x)$	x_k	一阶差商	二阶差商	三阶差商
-1.150000000000000	0.800000000000000	0	0	0
-0.870000000000000	0	2.857142857142858	0	0
1.000000000000000	1.200000000000000	0.641711229946524	-1.030433314975039	0
2.210000000000000	1.800000000000000	0.495867768595041	-0.047351773166066	0.292583792205052

$x = f^{-1}(y)$ 的三次 $Newton$ 插值多项式为:

$$N_3(y) = 2.85714y - 1.03043(y+1.15)(y+0.87) + 0.292584(y+1.15)(y+0.87)(y-1.0) + 2.48571$$

3.2.5 结果分析

3.3 Runge 现象

利用 Lagrange 插值或者 Newton 插值实现 Runge 现象, 即对函数 $f(x) = \dfrac{1}{1+x^2}$ 在区间 $[-5,5]$ 上构造等距节点的高次插值函数, 并作图.

3.3.1 程序

Listing 17: $Runge$ 现象

```
%利用lagrange插值多项式绘图
%code by zhongwei
```

```
%a, b为左右区间端点
clear;
a = -5;
b = 5;
X = a:0.001:b;
%绘制函数图像
plot(X,1./(1+X.^2),'b','LineWidth',3);
hold on;
% 生成用于插值的点
n = [2,4,6,8,10];
for i=1:size(n,2)
    h = (b-a)/n(i);
    xx = a:h:b;
    for j=1:size(xx,1)
        yy=1./(1+xx.^2);
    end
    L = lagrange(xx, yy, X);
    plot( X, L, '-','LineWidth',3);
    idx = i;
    % 设置图例矩阵
    legend_str{idx} = ['$P_{', num2str(n(i)), '}(x)$'];
end
%添加图例
legend(legend_str, 'Interpreter', 'latex');
%添加图表标题
t = '$Plot$ $of$ $Interpolation$ $polynomial$($Plot$ $by$ $zhongwei$)';
title(t,'interpreter','latex')
```

Listing 18: *lagrange* 插值函数

```
function L = lagrange(x, y, xx)
%lagrange插值
%the first edition
%code by zhongwei
% x: 数据点的x坐标
% y: 数据点的y坐标
% xx: 需要计算插值的点的x坐标
% L: 对应的插值多项式值

n = length(x); % 数据点数量
L = zeros(size(xx)); % 初始化插值多项式值

% 遍历每个插值点
for k = 1:length(xx)
```

```
    % 计算插值多项式在xx(k)处的值
    for i = 1:n
        % 计算插值基函数的值
        Lk = 1;
        for j = 1:n
            if j ~= i
                Lk = Lk * (xx(k) - x(j)) / (x(i) - x(j));
            end
        end
        % 累加插值多项式的值
        L(k) = L(k) + y(i) * Lk;
    end
end
end
```

3.3.2 可视化结果

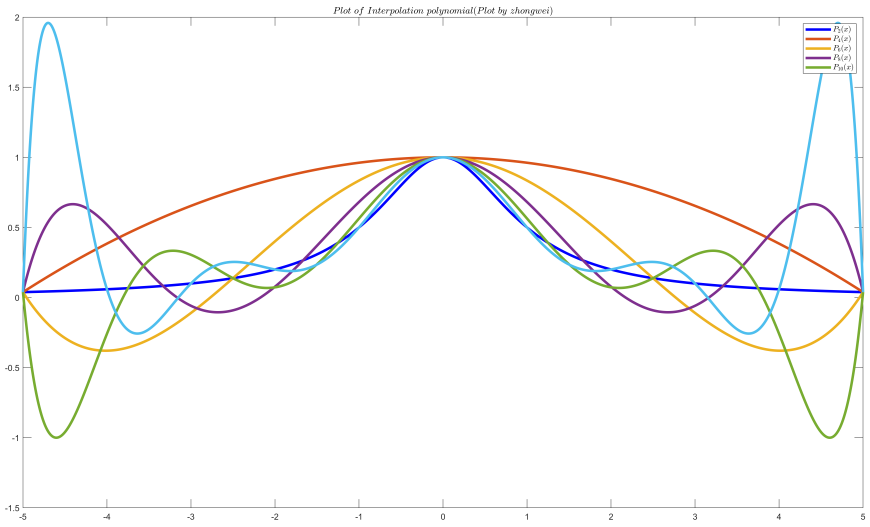

图 1: *Runge* 现象

3.3.3 结果分析

从图像中可以看出插值次数并不是越高越好, 随着插值次数的增大, 可以发现在端点处的震荡越来越大, 拟合效果不佳, 产生了 *Runge* 现象: 使用均匀节点构造高次多项式差值时, 在插值区间的边缘的误差可能很大的现象.

实验 4　曲线拟合

4.1　曲线拟合

4.1.1　问题

(课件例题) 试用最小二乘法确定经验公式 $f(x)=\dfrac{cx}{1+ax+bx^2}$ 中的参数，使之与如下数据拟合.

x_i	0.1	0.2	0.3	0.4	0.5	0.6
y_i	0.172	0.323	0.484	0.690	1.000	1.579

4.1.2　方法

曲线最小二乘拟合是指给定一组二维平面上的数据点 (x_i, y_i), 通过拟合一个函数 $y=f(x)$, 使得该函数与数据点的距离最小. 同样地, 我们可以将拟合函数写成一组基函数的线性组合形式：

$$y=c_0\phi_0(x)+c_1\phi_1(x)+\cdots+c_m\phi_m(x)=\sum_{i=1}^{m}c_i\phi_i(x)\quad(m<n)$$

其中 $\phi_0(x),\phi_1(x),\ldots,\phi_m(x)$ 是给定的基函数,$c_0,c_1,\ldots,c_m$ 是待确定的拟合参数. 我们的任务是确定这些参数的值, 以使得拟合曲线与数据点的距离最小.

$\phi(x_i)=y_i(i=0,1,2,\cdots,n)$ 可以得到以下方程组

$$\begin{cases} c_0\phi_0(x_0)+c_1\phi_1(x_0) & +\cdots+c_m\phi_m(x_0)=y_0,\\ c_0\phi_0(x_1)+c_1\phi_1(x_1) & +\cdots+c_m\phi_m(x_1)=y_1,\\ \quad\cdots\cdots & \\ c_0\phi_0(x_n)+c_1\phi_1(x_n) & +\cdots+c_m\phi_m(x_n)=y_n. \end{cases}$$

写成矩阵形式 $\boldsymbol{Ax=b}$, 其中

$$\boldsymbol{A}=\begin{pmatrix} \phi_0(x_0) & \phi_1(x_0) & \cdots & \phi_m(x_0)\\ \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_m(x_1)\\ \vdots & \vdots & & \vdots\\ \phi_0(x_n) & \phi_1(x_n) & \cdots & \phi_m(x_n) \end{pmatrix},$$

$$\boldsymbol{x}=(c_0,c_1,\cdots,c_m)^T,\quad \boldsymbol{b}=(y_0,y_1,\cdots,y_n)^T.$$

对于曲线拟合问题, 拟合函数可以是任意的形式, 但是目标函数仍然是要最小化下面的距离平方和:

$$S=\sum_{i=0}^{n}(y_i-\sum_{j=0}^{m}c_j\phi_j(x_i))^2$$

这个目标函数可以通过求解一组偏导数方程组的解来确定参数 $c_0,c_1,\ldots,c_m$ 的值，即

$$\frac{\partial S}{\partial c_j}=-2\sum_{i=0}^{n}\phi_j(x_i)(y_i-\sum_{j=0}^{m}c_j\phi_j(x_i))=0,\quad j=0,1,\ldots,m.$$

这个方程组可以写成矩阵形式：

$$\begin{pmatrix}\sum\phi_0(x_i)^2 & \sum\phi_0(x_i)\phi_1(x_i) & \cdots & \sum\phi_0(x_i)\phi_m(x_i)\\ \sum\phi_1(x_i)\phi_0(x_i) & \sum\phi_1(x_i)^2 & \cdots & \sum\phi_1(x_i)\phi_m(x_i)\\ \vdots & \vdots & & \vdots\\ \sum\phi_m(x_i)\phi_0(x_i) & \sum\phi_m(x_i)\phi_1(x_i) & \cdots & \sum\phi_m(x_i)^2\end{pmatrix}\begin{pmatrix}c_0\\c_1\\\vdots\\c_m\end{pmatrix}=\begin{pmatrix}\sum y_i\phi_0(x_i)\\\sum y_i\phi_1(x_i)\\\vdots\\\sum y_i\phi_m(x_i)\end{pmatrix}$$

求解线性方程组 $\boldsymbol{Ax=b}$ 的正规方程组 $\boldsymbol{A^TAx=A^Tb}$ 可表示为

$$\begin{pmatrix}(\phi_0,\phi_0) & (\phi_0,\phi_1) & \cdots & (\phi_0,\phi_m)\\ (\phi_1,\phi_0) & (\phi_1,\phi_1) & \cdots & (\phi_1,\phi_m)\\ \vdots & \vdots & & \vdots\\ (\phi_m,\phi_0) & (\phi_m,\phi_1) & \cdots & (\phi_m,\phi_m)\end{pmatrix}\begin{pmatrix}c_0\\c_1\\\vdots\\c_m\end{pmatrix}=\begin{pmatrix}(\phi_0,b)\\(\phi_0 1,b)\\\vdots\\(\phi_m,b)\end{pmatrix}$$

其中

$$(\phi_k,\phi_j)=\sum_{i=0}^{n}\phi_k(x_i)\phi_j(x_i),\quad (\phi_k,\boldsymbol{b})=\sum_{i=0}^{n}\phi_k(x_i)y_i.$$

对于本题, 首先对经验公式进行变形, 转换成一组已知函数的线性组合, 如下:

$$g(x)=\tfrac{1}{f(x)}=\tfrac{a}{c}+\tfrac{1}{cx}+\tfrac{b}{c}x\xlongequal{\text{记为}}c_0\tfrac{1}{x}+c_1+c_2x,\quad g(x)\in\boldsymbol{span}\{\tfrac{1}{x},1,x\}.$$

函数表如下:

x_i	0.1	0.2	0.3	0.4	0.5	0.6
$g(x_i)$	5.81395	3.09598	2.06612	1.44928	1.00000	0.63331

4.1.3 程序

Listing 19: 最小二乘法曲线拟合

```
% 输入数据点
x = linspace(0.1,0.6,6);
y = [0.172,0.323,0.484,0.69,1,1.579];
n = 6;
```

```
Y = 1./y;
% 构建设计矩阵
X = [1./x',ones(n,1), x'];
% 使用最小二乘法求解参数
beta = (X'*X)\(X'*Y');
M = X'*X;
disp(M);
N = X'*Y';
disp(N);
% 提取参数估计值
c0 = beta(1);
c1= beta(2);
c2 = beta(3);
fprintf('c0=%.6f,c1=%.6f,c2=%.6f\n',c0,c1,c2);
c = 1./c0;
a = c1.*c;
b = c2.*c;
fprintf('c=%.6f,a=%.6f,b=%.6f\n',c,a,b);
```

4.1.4 数值结果

最小二乘曲线拟合的法方程组为 $\boldsymbol{A^TAC = A^Tg}$, 即

$$\begin{pmatrix} 149.139 & 24.50 & 6 \\ 24.5 & 6 & 2.1 \\ 6 & 2.1 & 0.91 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 87.185172 \\ 14.058632 \\ 3.280123 \end{pmatrix}$$

解方程组得

$$\begin{cases} c_0 = 0.503375 \\ c_1 = 0.976071 \\ c_2 = -1.966900 \end{cases}$$

故

$$\begin{cases} c = \dfrac{1}{c_0} & = 1.986590 \\ a = c_1 c & = 1.939053 \\ b = c_2 c & = -\,3.907422 \end{cases}$$

4.2 程序验证

确定经验公式 $f(x) = ae^{bx}$ 中的参数 a 和 b, 使之与如下数据拟合.

x	1.0	1.25	1.50	1.75	2.00
$f(x)$	5.10	5.79	6.53	7.45	8.48

4.2.1 数值结果

最小二乘曲线拟合的法方程组为 $\boldsymbol{A^T AC = A^T g}$, 即

$$\begin{pmatrix} 5. & 7.5 \\ 7.5 & 11.875 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 9.407704 \\ 14.428812 \end{pmatrix}$$

解方程组得

$$\begin{cases} a = 3.065247 \\ b = 0.507609 \end{cases}$$

故

$$\boldsymbol{y = e^{3.065247+0.507609x}}$$

实验 5　数值积分

5.1　复化梯形公式

5.1.1　问题

已知 $f(x) = 2 + \sin(2\sqrt{x})$ 的不定积分为

$$F(x) = 2x - \sqrt{x}\cos(2\sqrt{x}) + \frac{\sin(2\sqrt{x})}{2}.$$

在区间 $[1, 6]$ 上, 对 $f(x)$ 使用复化梯形公式计算积分, 计算子区间数分别为 $10, 20, 40, 80$ 和 160 时的误差且验证收敛阶为 $\boldsymbol{O}(h^2)$, 即补充下表的第 3-5 列. 其中误差和收敛阶的计算公式为

$$E_T(f,h) = \int_1^6 f(x)\,\mathrm{d}x - T(f,h), \quad \text{Rate} = \frac{\ln\left(E_T(f,h_1)/E_T(f,h_2)\right)}{\ln\left(h_1/h_2\right)}.$$

表 19: $f(x) = 2 + \sin(2\sqrt{x})$ 在 $[1, 6]$ 上的复化梯形公式

n	h	$T(f,h)$	$E_T(f,h)$	Rate
10	0.5			–
20	0.25			
40	0.125			
80	0.0625			
160	0.03125			

5.1.2　方法

- 根据不定积分计算精确积分 $\int_1^6 f(x)\,\mathrm{d}x$,
- 写出复化梯形公式 $T(f,h)$ 及其算法, 算法包括输入参数和输出参数.
- 提示使用 `feval()` 来计算函数 f 在某些点处的值, 将函数名 f 作为输入参数.

复化梯形公式是一种数值积分方法, 用于计算在一个区间上的定积分. 它的基本思想是将区间分成若干个小区间, 然后在每个小区间上用梯形面积来近似被积函数的面积.

在区间 $[a, b]$ 上, 假设有 n 个等距分割点 $x_0 = a, x_1, x_2, \ldots, x_n = b$, 其中梯形宽度为 $h = \frac{b-a}{n}$, 则复化梯形公式的近似积分公式为:

$$I \approx \frac{h}{2}\left[f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)\right]$$

其中 $f(x)$ 是被积函数. 需要注意的是, 在每个小区间内使用梯形面积进行近似.

复化梯形公式的精度与分割数 n 的选择有关. 当 n 越大时, 精度越高, 但计算量也越大. 一般来说, 需要根据被积函数的性质和精度要求选择合适的分割数.

在 MATLAB 中, 可以使用类似于复化梯形公式的程序结构来实现复化梯形公式的计算. 需要先计算分割点和函数值, 然后按照梯形面积的权重对函数值进行加权, 并计算加权和的积分近似值.

5.1.3　程序

Listing 20: 复化梯形公式函数定义

```
function [T] = composite_trapezoidal_rule(f,a,b,n)
% 使用复化梯形公式计算不定积分
% code by zhongwei
% 输入参数:
% f - 被积函数
% a - 积分区间的下限
% b - 积分区间的上限
% n - 分割数
% 输出参数:
% T - 积分的近似值

h = (b-a)/n; % 梯形宽度
sum = 0;
for i = 1:1:n-1
    x = a + i*h;
    sum = sum + 2*f(x);
end
T = h*(f(a) + sum + f(b))/2; % 计算积分值
end
```

Listing 21: 复化梯形公式函数调用

```
%% 定义输入及基本量
%code by zhongwei
clear;clc;
% 定义被积函数
f = @(x) 2 + sin(2.*sqrt(x));
F = @(x) 2.*x -sqrt(x) .* cos(2.*sqrt(x)) + (sin(2.*sqrt(x)))/2
% 定义积分区间和分割数
a = 1;
b = 6;
% 定义分割数矩阵
N = [10,20,40,80,160];
% 计算积分的精确值
I = F(b) - F(a);

%% 复化梯形公式的调用
% 定义积分值矩阵,用于存放复化梯形积分结果
T=[];
% 定义矩阵E,用于存储复化梯形积分误差
E_T = [];
% 定义矩阵h,用于存储区间长度
h = [];
```

```
fprintf('------------复化梯形公式------------\n')
% 调用复化梯形公式计算积分值,循环计算不同分割数的积分值
for k =1:length(N)
    n = N(k);
    % 计算区间长度
    h(k) = (b-a)./n;
    T(k) = composite_trapezoidal_rule(f,a,b,n);
    % 计算和精确值之间的误差
    E_T(k) = T(k) - I;
    % 输出积分值
    fprintf('复化梯形公式:n=%d,The approximate value of the integral is %.6
   f,the error is %.10f\n',n,T(k),E_T(k));
end

%% 复化Simpson公式的调用
% 定义积分值矩阵,用于存放复化Simpson积分结果
S=[];
% 定义矩阵E_S,用于存储复化Simpson积分误差
E_S = [];
% 定义矩阵h,用于存储区间长度
h = [];
fprintf('------------复化Simpson公式------------\n')
% 调用复化梯形公式计算积分值,循环计算不同分割数的积分值
for k =1:length(N)
    n = N(k);
    % 计算区间长度
    h(k) = (b-a)./n;
    S(k) = composite_simpson_rule(f,a,b,n);
    % 计算和精确值之间的误差
    E_S(k) = S(k) - I;
    % 输出积分值
    fprintf('复化Simpson公式:n=%d,The approximate value of the integral is
   %.6f,the error is %.10f\n',n,S(k),E_S(k));
end

%% 最终输出
% 输出积分精确值
fprintf('The exact value of the integral is %.10f\n',I);
% 合成输出总表
Table_T = [N',h',T',E_T'];
disp('复化梯形公式Table:');
disp(Table_T);
Table_S = [N',h',S',E_S'];
```

```
63 disp('复化Simpson公式Table:');
64 disp(Table_S);
```

5.1.4 数值结果

表 20: $f(x)=2+\sin(2\sqrt{x})$ 复化梯形公式求积结果

n	h	$S(f,h)$	$E_S(f,h)$	Rate
10	0.5	8.19385456517	0.010375358	–
20	0.25	8.18604926377	2.57E-03	2.01
40	0.125	8.18412019179	6.41E-04	2.00
80	0.0625	8.18363935732	1.60E-04	2.00
160	0.03125	8.18351923904	4.00E-05	2.00

5.1.5 结果分析

随着子区间数 n 的不断增大, 复化梯形公式所计算的积分近似值与精确值之间的误差越来越小. 复化梯形公式的收敛阶为 $\boldsymbol{O}(h^2)$.

5.2 复化 Simpson 公式

5.2.1 问题

已知 $f(x)=2+\sin(2\sqrt{x})$ 的不定积分为

$$F(x)=2x-\sqrt{x}\cos(2\sqrt{x})+\frac{\sin(2\sqrt{x})}{2}.$$

在区间 $[1,6]$ 上, 对 $f(x)$ 使用复化复化 Simpson 公式计算积分, 计算子区间数分别为 $10,20,40,80$ 和 160 时的误差且验证收敛阶为 $\boldsymbol{O}(h^4)$, 即补充下表的第 3-5 列. 其中误差和收敛阶的计算公式为

$$E_S(f,h)=\int_1^6 f(x)\,\mathrm{d}x-S(f,h),\quad \text{Rate}=\frac{\ln\left(E_S(f,h_1)/E_S(f,h_2)\right)}{\ln\left(h_1/h_2\right)}.$$

表 21: $f(x)=2+\sin(2\sqrt{x})$ 在 $[1,6]$ 上的复化 Simpson 公式

n	h	$S(f,h)$	$E_S(f,h)$	Rate
10	0.5			–
20	0.25			
40	0.125			
80	0.0625			
160	0.03125			

5.2.2 方法

- 根据不定积分计算精确积分 $\int_1^6 f(x)\,\mathrm{d}x$,
- 写出复化梯形公式 $S(f,h)$ 及其算法, 算法包括输入参数和输出参数.
- 提示使用 `feval()` 来计算函数 f 在某些点处的值, 将函数名 f 作为输入参数.

复化 Simpson 公式是一种数值积分方法, 用于计算在一个区间上的定积分. 与梯形法类似,Simpson 公式是一种基于插值的方法, 但它使用二次多项式来逼近被积函数而不是一次多项式.

在区间 $[a,b]$ 上, 假设有一个偶数个的分割点 $x_0=a,x_1,x_2,\ldots,x_n=b$, 其中 n 是偶数, 且梯形宽度为 $h=\frac{b-a}{n}$,h 表示相邻节点的距离, 故为 $\frac{h}{3}$, 则复化 Simpson 公式的近似积分公式为:

$$I \approx \frac{h}{3}\left[f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+\cdots+4f(x_{n-1})+2f(x_n)+f(x_{n+1})\right]$$

其中 $f(x)$ 是被积函数. 需要注意的是, 每三个分割点组成一个小区间, 在每个小区间内使用 Simpson 公式进行近似, 因此称为复化 Simpson 公式.

复化 Simpson 公式的精度比复合梯形公式高一个阶, 即如果被积函数具有足够的光滑性, 复合 Simpson 公式的收敛速度比复化梯形公式更快.

在 MATLAB 中, 可以使用类似于复合梯形公式的程序结构来实现复化 Simpson 公式的计算. 需要先计算分割点和函数值, 然后按照 Simpson 公式的权重对函数值进行加权, 并计算加权和的积分近似值.

5.2.3 程序

Listing 22: 复化 *Simpson* 公式函数定义

```
function [S] = composite_simpson_rule(f,a,b,n)
% 使用复合Simpson公式计算不定积分
% code by zhongwei
% 输入参数:
% f - 被积函数
% a - 积分区间的下限
% b - 积分区间的上限
% n - 分割数（偶数）
% 输出参数:
% S - 积分的近似值

h = (b-a)/n; % 梯形宽度
sum = 0;
for i = 0:1:n-1
    x = a + i*h;
    sum = sum + f(x) + 4*f(x+0.5*h) + f(x+h);
end
S = sum*h/6; % 计算积分值
end
```

Listing 23: 复化 *Simpson* 公式函数调用

```
%% 定义输入及基本量
%code by zhongwei
clear;clc;
% 定义被积函数
f = @(x) 2 + sin(2.*sqrt(x));
F = @(x) 2.*x -sqrt(x) .* cos(2.*sqrt(x)) + (sin(2.*sqrt(x)))/2
% 定义积分区间和分割数
a = 1;
b = 6;
% 定义分割数矩阵
N = [10,20,40,80,160];
% 计算积分的精确值
I = F(b) - F(a);

%% 复化梯形公式的调用
% 定义积分值矩阵,用于存放复化梯形积分结果
T=[];
% 定义矩阵E,用于存储复化梯形积分误差
E_T = [];
% 定义矩阵h,用于存储区间长度
h = [];
fprintf('------------复化梯形公式------------\n')
% 调用复化梯形公式计算积分值,循环计算不同分割数的积分值
for k =1:length(N)
    n = N(k);
    % 计算区间长度
    h(k) = (b-a)./n;
    T(k) = composite_trapezoidal_rule(f,a,b,n);
    % 计算和精确值之间的误差
    E_T(k) = T(k) - I;
    % 输出积分值
    fprintf('复化梯形公式:n=%d,The approximate value of the integral is %.6
   f,the error is %.10f\n',n,T(k),E_T(k));
end

%% 复化Simpson公式的调用
% 定义积分值矩阵,用于存放复化Simpson积分结果
S=[];
% 定义矩阵E_S,用于存储复化Simpson积分误差
E_S = [];
% 定义矩阵h,用于存储区间长度
h = [];
```

```
fprintf('------------复化Simpson公式------------\n')
% 调用复化梯形公式计算积分值,循环计算不同分割数的积分值
for k =1:length(N)
    n = N(k);
    % 计算区间长度
    h(k) = (b-a)./n;
    S(k) = composite_simpson_rule(f,a,b,n);
    % 计算和精确值之间的误差
    E_S(k) = S(k) - I;
    % 输出积分值
    fprintf('复化Simpson公式:n=%d,The approximate value of the integral is
    %.6f,the error is %.10f\n',n,S(k),E_S(k));
end

%% 最终输出
% 输出积分精确值
fprintf('The exact value of the integral is %.10f\n',I);
% 合成输出总表
Table_T = [N',h',T',E_T'];
disp('复化梯形公式Table:');
disp(Table_T);
Table_S = [N',h',S',E_S'];
disp('复化Simpson公式Table:');
disp(Table_S);
```

5.2.4 数值结果

表 22: $f(x) = 2 + \sin(2\sqrt{x})$ 复化 *Simpson* 公式求积结果

n	h	$S(f,h)$	$E_S(f,h)$	Rate
10	0.5	8.18344749664	-3.17E-05	–
20	0.25	8.18347716780	-2.04E-06	3.96
40	0.125	8.18347907916	-1.29E-07	3.99
80	0.0625	8.18347919962	-8.05E-09	4.00
160	0.03125	8.18347920716	-5.03E-10	4.00

5.2.5 结果分析

随着子区间数 n 的不断增大, 复化 Simpson 公式所计算的积分近似值与精确值之间的误差越来越小. 复化 Simpson 公式的收敛阶为 $\boldsymbol{O}(h^4)$, 相较于复化梯形公式有着更好的收敛性.