目录

《计算机科学计算》

第四章

12 题目

12．用 Newton 法求下列方程的根，要求 $|x_k - x_{k-1}| < 10^{-5}$

（1）$x^3 - x^2 - x - 1 = 0$　　取 $x_0 = 2$

（2）$x = e^{-x}$　　取 $x_0 = 0.6$

Newton 迭代法代码见 [Newton 迭代法]

主函数

```
function C4_12
    format long;
    syms x;

    disp('(1)-----------------------------------------')
    f = x ^ 3 - x ^ 2 - x - 1;
    x0 = 2;
    tol = 1e-5;
    max_iter = 4;
    root = NewtonIterate(f, x, x0, tol, max_iter)

    disp('(2)-----------------------------------------')
    f = x - exp(-x);
    x0 = 0.6;
    tol = 1e-5;
    max_iter = 3;
    root = NewtonIterate(f, x, x0, tol, max_iter)
end
```

（1）结果

```
(1)-----------------------------------------

root =

   1.839286755214163
```

（2）结果

```
(2)----------------------------------------

root =

   0.567143290409784
```

17 题目

17．应用 Newton 法求方程

$$cos(x)\cdot sh(x)-1=0$$

的头五个非零的正根

主函数

```
function C4_17
    format long;
    syms x;
    f = cos(x) * sinh(x) - 1;
    x0_array = [5, 8, 10, 14, 17];
    root_array = x0_array;
    tol = 1e-5;
    max_iter = 100;

    for i = 1:length(x0_array)
        x0 = x0_array(i);
        root = NewtonIterate(f, x, x0, tol, max_iter);
        root_array(i) = root;
    end

    root_array'
end
```

结果

```
ans =

    4.730043447824069
    7.853204623861108
   10.995607838001737
   14.137165491257532
   17.278759657399483
```

第六章

15 题目

13．（数值实验题目）计算 $f(x)=\frac{1}{2}+\frac{1}{\sqrt{2\pi}}\int_0^x e^{-t^2/2}\mathrm{d}t \quad (0\le x\le 3)$ 的函数值 $\{f(0.1k);k=1,2,\cdots,30\}$．计算结果取 7 位有效数字.

主函数

```
function C6_15
    format long;
    syms x t;

    % 被积函数
    gt = exp(-t ^ 2/2);
    gt_numeric = matlabFunction(gt);

    % 求积节点
    t_array = 0:0.1:3;
    gt_array = t_array;
    for i = 1:length(t_array)
        ti = t_array(i);
        gti = gt_numeric(ti);
        gt_array(i) = gti;
    end

    % 求积
    integral_value = t_array;
    for i = 2:length(t_array)
        sum = 0;
        % 偶数个节点，复化梯形
        if 0 == mod(i, 2)
            for j = 1:i - 1
                sum = sum + gt_array(j) + gt_array(j + 1);
            end
            sum = (t_array(i) - t_array(1)) / (2 * (i - 1)) * sum;
        else
        % 奇数个节点，复化 Simpson
            for j = 1:2:i - 2
                sum = sum + gt_array(j) + 4 * gt_array(j + 1) +
gt_array(j + 2);
            end
```

```
            sum = (t_array(i) - t_array(1)) / (6 * (i - 1) / 2) * sum;
        end
        integral_value(i) = sum;
    end

    f = 0.5 + integral_value / sqrt(2 * pi);

    % matlab 计算的值
    true_integral = t_array;
    true_f = t_array;
    for i = 1:length(t_array)
        true_integral(i) = integral(gt_numeric, 0, 0.1 * (i - 1));
        true_f(i) = 0.5 + true_integral(i) / sqrt(2 * pi);
    end

    % 对比
    [integral_value', true_integral']
    [f', true_f']
end
```

结果

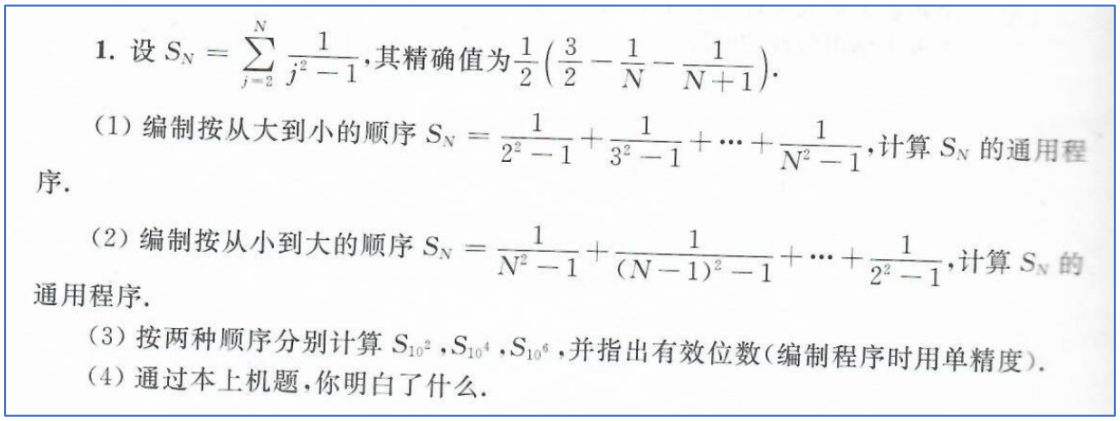
第一列为数值积分结果，第二列为 matlab 的计算值：

```
ans =

   0.500000000000000   0.500000000000000
   0.539794741393922   0.539827837277029
   0.579259838809498   0.579259709439103
   0.617816028938345   0.617911422188953
   0.655421975346860   0.655421741610324
   0.691315700079918   0.691462461274013
   0.725747177021938   0.725746882249926
   0.758541233756651   0.758036347776927
   0.788144906659282   0.788144601416603
   0.815740237815075   0.815939874653241
   0.841345015888470   0.841344746068543
   0.864134181613555   0.864333939053617
   0.884930532142889   0.884930329778292
   0.903013825565894   0.903199515414390
   0.919243461769895   0.919243340766229
   0.933030881515153   0.933192798731142
   0.945200751202520   0.945200708300442
   0.955301298635238   0.955434537241457
   0.964069661285450   0.964069680887074
   0.971179559072905   0.971283440183998
   0.977249807415877   0.977249868051821
   0.982058626261979   0.982135579437184
   0.986096472191914   0.986096552486502
   0.989221617232704   0.989275889978324
   0.991802381328830   0.991802464075404
   0.993753837173947   0.993790334674224
   0.995338738045590   0.995338811976281
   0.996509595862715   0.996533026196959
   0.997444810048069   0.997444869669572
   0.998119814382723   0.998134186699616
   0.998650057703869   0.998650101968370
```

《数值分析方法与应用》

一、基础知识部分

1 题目



1. 设 $S_N=\sum_{j=2}^{N}\frac{1}{j^2-1}$，其精确值为 $\frac{1}{2}\left(\frac{3}{2}-\frac{1}{N}-\frac{1}{N+1}\right)$.

(1) 编制按从大到小的顺序 $S_N=\frac{1}{2^2-1}+\frac{1}{3^2-1}+\cdots+\frac{1}{N^2-1}$，计算 S_N 的通用程序.

(2) 编制按从小到大的顺序 $S_N=\frac{1}{N^2-1}+\frac{1}{(N-1)^2-1}+\cdots+\frac{1}{2^2-1}$，计算 S_N 的通用程序.

(3) 按两种顺序分别计算 S_{10^2}，S_{10^4}，S_{10^6}，并指出有效位数(编制程序时用单精度).

(4) 通过本上机题，你明白了什么.

（1）（2）见主函数

```
function C1_1
    format long;
    disp('(3)-----------------------------------------')
    N = single([1e2, 1e4, 1e6]);
    sum_min_N = N;
    sum_max_N = N;

    for i = 1:length(N)
        sum_min_N(i) = sum_min(N(i));
        sum_max_N(i) = sum_max(N(i));
    end

    [sum_min_N', sum_max_N']

    disp('(4)-----------------------------------------')
    disp('计算机计算存在舍入误差，应避免大数吃小数的情况发生')
end

% disp('(1)-----------------------------------------')
function [sum] = sum_min(n)
    sum = single(0);
    for i = n:-1:2
        sum = sum + 1 / (i * i - 1);
    end
end
```

```
% disp('(2)-----------------------------------------')
function [sum] = sum_max(n)
    sum = single(0);
    for i = 2:n
        sum = sum + 1 / (i * i - 1);
    end
end
```

（3）结果

```
ans =

  3x2 single 矩阵

   0.7400495   0.7400495
   0.7499000   0.7498521
   0.7499990   0.7498521
```

（4）计算机计算存在舍入误差，应避免大数吃小数的情况发生

2 题目

3. 用秦九韶算法编程计算 $f(x)=1+x+x^2+\cdots+x^{50}$ 在 $x=1.000\ 01$ 处的值.

主函数

```
function C1_3
    format long;
    fx = f(1.00001, 50)
end

function [fx] = f(x, n)
    % n 是最高次项系数
    fx = 1;
    for i = 1:n
        fx = x * fx + 1;
    end
end
```

结果

```
fx =

   51.012752082749991
```

二、线性方程组求解

1 题目

1. 分别用 Gauss 消元法和列主元消去法编程求解方程组 $Ax = b$，其中

$$A = \begin{pmatrix} 31 & -13 & 0 & 0 & 0 & -10 & 0 & 0 & 0 \\ -13 & 35 & -9 & 0 & -11 & 0 & 0 & 0 & 0 \\ 0 & -9 & 31 & -10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -10 & 79 & -30 & 0 & 0 & 0 & -9 \\ 0 & 0 & 0 & -30 & 57 & -7 & 0 & -5 & 0 \\ 0 & 0 & 0 & 0 & -7 & 47 & -30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -30 & 41 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5 & 0 & 0 & 27 & -2 \\ 0 & 0 & 0 & -9 & 0 & 0 & 0 & -2 & 29 \end{pmatrix}$$

$$b = (-15, 27, -23, 0, -20, 12, -7, 7, 10)^T$$

并求出矩阵 A 的 LU 分解及列主元的 LU 分解(求出 L,U 和 P)，并用 LU 分解的方法求 A 的逆矩阵及 A 的行列式.

Gauss 消元法

```
function [x] = GaussElimination(A, b)
    %GaussElimination   高斯消元法解方程组
    %   A: 系数矩阵，方阵
    %   b: 常数向量
    % 矩阵的大小
    [n, ~] = size(A);
    Ab = [A, b];

    % 消元，i 为列，j 为行
    for i = 1:n - 1
        for j = i + 1:n
            factor = Ab(j, i) / Ab(i, i);
            Ab(j, i:n + 1) = Ab(j, i:n + 1) - factor * Ab(i, i:n + 1);
        end
    end

    % 回代
    x = zeros(n, 1);
    for i = n:-1:1
        x(i) = (Ab(i, n + 1) - Ab(i, i + 1:n) * x(i + 1:n)) / Ab(i, i);
    end
end
```

Gauss 列主元消去法

```
function [x] = GaussEliminationWithPivoting(A, b)
    %GaussEliminationWithPivoting   高斯列主元消元法解方程组
    %   A: 系数矩阵，方阵
    %   b: 常数向量
    % 矩阵的大小
    [n, ~] = size(A);
    Ab = [A, b];

    % 前向消元
    for i = 1:n - 1
        % 选择列主元
        [~, maxIndex] = max(abs(Ab(i:n, i)));
        maxIndex = maxIndex + i - 1;

        % 交换行
        if i ~= maxIndex
            temp = Ab(i, :);
            Ab(i, :) = Ab(maxIndex, :);
            Ab(maxIndex, :) = temp;
        end

        % 消元
        for j = i + 1:n
            factor = Ab(j, i) / Ab(i, i);
            Ab(j, i:n + 1) = Ab(j, i:n + 1) - factor * Ab(i, i:n + 1);
        end
    end

    % 回代
    x = zeros(n, 1);
    for i = n:-1:1
        x(i) = (Ab(i, n + 1) - Ab(i, i + 1:n) * x(i + 1:n)) / Ab(i, i);
    end
end
```

LU 分解

```
function [L, U] = LUDecomposition(A)
    %LUDecomposition   LU 分解
    %   A: 系数矩阵，方阵
    [n, ~] = size(A);
    L = eye(n);
    U = A;
```

```
    for i = 1:n - 1
        for j = i + 1:n
            L(j, i) = U(j, i) / U(i, i);
            U(j, i:n) = U(j, i:n) - L(j, i) * U(i, i:n);
        end
    end
end
```

带列主元的 LU 分解

```
function [L, U, P] = LUDecompositionWithPivoting(A)
    %LUDecompositionWithPivoting   带列主元的 LU 分解
    %   A: 系数矩阵，方阵
    [n, ~] = size(A);
    L = eye(n);
    U = A;
    P = eye(n);

    for i = 1:n - 1
        % 列主元选择
        [~, maxIndex] = max(abs(U(i:n, i)));
        maxIndex = maxIndex + i - 1;

        % 交换 U 的行
        temp = U(i, :);
        U(i, :) = U(maxIndex, :);
        U(maxIndex, :) = temp;

        % 交换 P 的行
        temp = P(i, :);
        P(i, :) = P(maxIndex, :);
        P(maxIndex, :) = temp;

        % 交换 L 的行和调整 L 的列
        if i > 1
            temp = L(i, 1:i - 1);
            L(i, 1:i - 1) = L(maxIndex, 1:i - 1);
            L(maxIndex, 1:i - 1) = temp;
        end

        % LU 分解
        for j = i + 1:n
            L(j, i) = U(j, i) / U(i, i);
            U(j, i:n) = U(j, i:n) - L(j, i) * U(i, i:n);
        end
```

```
    end
end
```

LU 分解求逆矩阵

```
function [A_Inverse] = InverseByLU(A)
    %InverseByLU   通过 LU 分解求逆矩阵
    %   A: 系数矩阵，方阵

    % LU 分解（带列主元）
    [L, U, P] = LUDecompositionWithPivoting(A);
    n = size(A, 1);
    A_Inverse = zeros(n);

    % 对于 A 的每一列
    for i = 1:n
        e = zeros(n, 1);
        e(i) = 1;

        % 前向替换解 LY = Pe_i
        Y = ForwardSubstitution(L, P * e);

        % 回代解 UX = Y
        X = BackwardSubstitution(U, Y);

        % 构建 A 的逆
        A_Inverse(:, i) = X;
    end
end
```

Gauss 消元法的前向替换

```
function [Y] = ForwardSubstitution(L, B)
    %ForwardSubstitution    前向替换
    %   L: 下三角矩阵
    %   B: 常数矩阵
    n = size(L, 1);
    Y = zeros(n, 1);

    for i = 1:n
        Y(i) = (B(i) - L(i, 1:i - 1) * Y(1:i - 1)) / L(i, i);
    end
end
```

Gauss 消元法的回代

```
function [X] = BackwardSubstitution(U, Y)
```

```
    %BackwardSubstitution    回代
    %   U: 上三角矩阵
    %   Y: 前向替换得到的中间矩阵
    n = size(U, 1);
    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
end
```

LU 分解求行列式见主函数

```
function C2_1
    format short;
    A = [
         31 -13 0 0 0 -10 0 0 0
         -13 35 -9 0 -11 0 0 0 0
         0 -9 31 -10 0 0 0 0 0
         0 0 -10 79 -30 0 0 0 -9
         0 0 0 -30 57 -7 0 -5 0
         0 0 0 0 -7 47 -30 0 0
         0 0 0 0 0 -30 41 0 0
         0 0 0 0 -5 0 0 27 -2
         0 0 0 -9 0 0 0 -2 29
         ];
    b = [-15 27 -23 0 -20 12 -7 7 10]';

    disp('Gauss 消元法----------------------------------------------')
    gauss_x = GaussElimination(A, b)

    disp('Gauss 列主元消元法----------------------------------------')
    gauss_Pivoting_x = GaussEliminationWithPivoting(A, b);

    disp('LU 分解---------------------------------------------------')
    [L1, U1] = LUDecomposition(A);

    disp('列主元 LU 分解--------------------------------------------')
    [L2, U2, P2] = LUDecompositionWithPivoting(A);

    disp('LU 分解求逆矩阵-------------------------------------------')
    A_inv = InverseByLU(A);
    % 对比
    [A_inv, inv(A)]
```

```
    disp('LU 分解求行列式-----------------------------------------')
    n = size(A);
    det_A = 1;

    for i = 1:n
        det_A = det_A * L1(i, i) * U1(i, i);
    end

    % 对比
    [det_A, det(A)]'
end
```

结果

```
Gauss消元法------

gauss_x =

   -0.2892
    0.3454
   -0.7128
   -0.2206
   -0.4304
    0.1543
   -0.0578
    0.2011
    0.2902
```

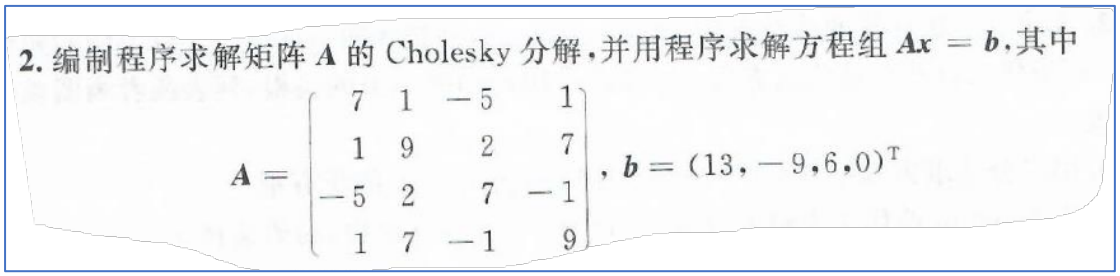
```
LU分解-----------------------------------------------

L1 =

    1.0000         0         0         0         0         0         0         0         0
   -0.4194    1.0000         0         0         0         0         0         0         0
         0   -0.3046    1.0000         0         0         0         0         0         0
         0         0   -0.3539    1.0000         0         0         0         0         0
         0         0         0   -0.3976    1.0000         0         0         0         0
         0         0         0         0   -0.1569    1.0000         0         0         0
         0         0         0         0         0   -0.6540    1.0000         0         0
         0         0         0         0   -0.1121   -0.0175   -0.0246    1.0000         0
         0         0         0   -0.1193   -0.0834   -0.0142   -0.0200   -0.0923    1.0000


U1 =

   31.0000  -13.0000         0         0         0  -10.0000         0         0         0
         0   29.5484   -9.0000         0  -11.0000   -4.1935         0         0         0
         0         0   28.2587  -10.0000   -3.3504   -1.2773         0         0         0
         0         0         0   75.4613  -31.1856   -0.4520         0         0   -9.0000
         0         0         0         0   44.6020   -7.1797         0   -5.0000   -3.5780
         0         0         0         0   -0.0000   45.8732  -30.0000   -0.7847   -0.5615
         0         0         0         0         0   -0.0000   21.3807   -0.5132   -0.3672
         0         0         0         0         0         0         0   26.4131   -2.4200
         0         0         0         0         0         0         0         0   27.3895
```

```
列主元LU分解-----------------------------------------------

L2 =

    1.0000         0         0         0         0         0         0         0         0
   -0.4194    1.0000         0         0         0         0         0         0         0
         0   -0.3046    1.0000         0         0         0         0         0         0
         0         0   -0.3539    1.0000         0         0         0         0         0
         0         0         0   -0.3976    1.0000         0         0         0         0
         0         0         0         0   -0.1569    1.0000         0         0         0
         0         0         0         0         0   -0.6540    1.0000         0         0
         0         0         0         0   -0.1121   -0.0175   -0.0246    1.0000         0
         0         0         0   -0.1193   -0.0834   -0.0142   -0.0200   -0.0923    1.0000


U2 =

   31.0000  -13.0000         0         0         0  -10.0000         0         0         0
         0   29.5484   -9.0000         0  -11.0000   -4.1935         0         0         0
         0         0   28.2587  -10.0000   -3.3504   -1.2773         0         0         0
         0         0         0   75.4613  -31.1856   -0.4520         0         0   -9.0000
         0         0         0         0   44.6020   -7.1797         0   -5.0000   -3.5780
         0         0         0         0   -0.0000   45.8732  -30.0000   -0.7847   -0.5615
         0         0         0         0         0   -0.0000   21.3807   -0.5132   -0.3672
         0         0         0         0         0         0         0   26.4131   -2.4200
         0         0         0         0         0         0         0         0   27.3895


P2 =

     1     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0
     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     0     1
```

```
LU分解求逆矩阵---------------------------------------------

A_inv =

    0.0390    0.0160    0.0058    0.0036    0.0073    0.0176    0.0129    0.0014    0.0012
    0.0159    0.0378    0.0131    0.0065    0.0121    0.0097    0.0071    0.0024    0.0022
    0.0049    0.0116    0.0381    0.0077    0.0069    0.0039    0.0028    0.0015    0.0025
    0.0008    0.0020    0.0064    0.0181    0.0105    0.0033    0.0024    0.0024    0.0058
    0.0005    0.0011    0.0036    0.0101    0.0243    0.0070    0.0051    0.0048    0.0035
    0.0001    0.0003    0.0010    0.0028    0.0068    0.0419    0.0306    0.0013    0.0010
    0.0001    0.0002    0.0007    0.0021    0.0050    0.0306    0.0468    0.0010    0.0007
    0.0001    0.0002    0.0008    0.0023    0.0048    0.0014    0.0010    0.0382    0.0033
    0.0003    0.0006    0.0020    0.0058    0.0036    0.0011    0.0008    0.0034    0.0365
```

```
LU分解求行列式---------

det_A =

   6.1817e+13
```

2 题目



Cholesky 分解

```
function [L] = CholeskyDecomposition(A)
    %CholeskyDecomposition   LL'分解
    %   A: 系数矩阵，对称正定方阵
    [n, ~] = size(A);
    L = zeros(n);

    for i = 1:n
        for j = 1:i
            if i == j
                % 对角元素
                L(i, i) = sqrt(A(i, i) - sum(L(i, 1:i - 1) .^ 2));
            else
                % 非对角元素
                L(i, j) = (A(i, j) - sum(L(i, 1:j - 1) .* L(j, 1:j -
1))) / L(j, j);
            end
        end
    end
end
```

主函数

```
function C2_2
    format short;
    A = [
         7 1 -5 1
         1 9 2 7
         -5 2 7 -1
         1 7 -1 9
         ];
    b = [13 -9 6 0]';

    disp('LLT 分解-----------------------------------------------------')
    L = CholeskyDecomposition(A);
```

```
    LT = L';

    disp('LLT 分解求解方程组----------------------------------------')
    y = ForwardSubstitution(L, b);
    x = BackwardSubstitution(LT, y);
    % 对比
    [x, A \ b]
end
```

结果

```
LLT分解-----------------------------------------------------

L =

    2.6458         0         0         0
    0.3780    2.9761         0         0
   -1.8898    0.9120    1.6115         0
    0.3780    2.3041   -1.4813    1.1636

LLT分解求解方程组-------------------------------------

x =

   19.0780
  -21.8716
   23.2294
   17.4725
```

4 题目

4. 已知

$$A=\begin{pmatrix} 1 & 1 & 0 & 0 \\ -1 & 3 & -\frac{1}{2} & \frac{1}{2} \\ -2 & 2 & \frac{3}{2} & \frac{1}{2} \\ -2 & 2 & -\frac{1}{2} & \frac{5}{2} \end{pmatrix}$$

编程求解矩阵 A 的 QR 分解.

QR 分解

```
function [Q, R] = QRDecomposition(A)
    %QRDecomposition   QR 分解
    %   A: 系数矩阵
```

```
    [m, n] = size(A);
    Q = eye(m);
    R = A;

    for k = 1:n
        % 求 Householder 矩阵
        x = R(k:m, k);
        e = zeros(length(x), 1);
        e(1) = 1;
        w = sign(x(1)) * norm(x) * e + x;
        w = w / norm(w);

        H = eye(m);
        H(k:m, k:m) = H(k:m, k:m) - 2 * (w * w');

        % 将 Householder 矩阵乘到 Q 上
        R = H * R;
        Q = Q * H;
    end

    % 取 R 的上三角
    R = triu(R(1:n, :));
    Q = Q(:, 1:n);
end
```

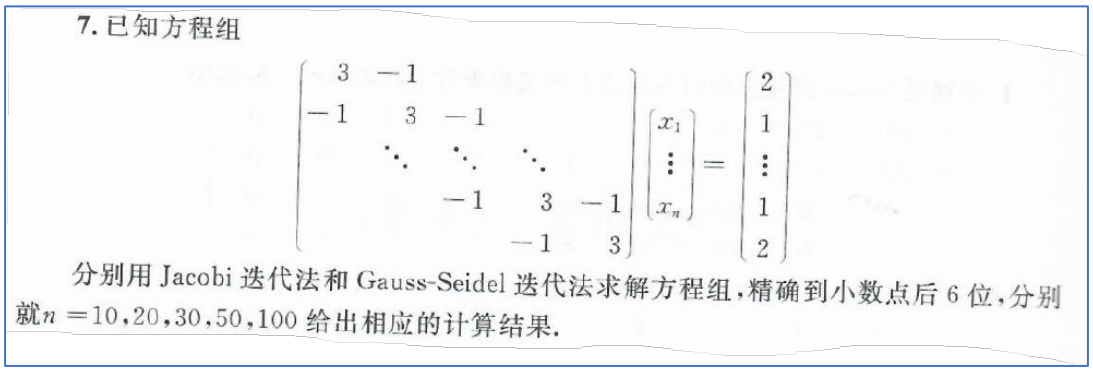
主函数

```
function C2_4
    format short;
    A = [
         1 1 0 0
         -1 3 -1/2 1/2
         -2 2 3/2 1/2
         -2 2 -1/2 5/2
         ];

    disp('QR 分解----------------------------------------------------')
    [Q, R] = QRDecomposition(A);
    [q, r] = qr(A);
    % 对比
    [Q, q]
    [R, r]
end
```

结果

```
QR分解--------------------------------------------------

Q =

   -0.3162   -0.7071   -0.2582   -0.5774
    0.3162   -0.7071    0.2582    0.5774
    0.6325   -0.0000   -0.7746    0.0000
    0.6325   -0.0000    0.5164   -0.5774


R =

   -3.1623    3.1623    0.4743    2.0555
         0   -2.8284    0.3536   -0.3536
         0         0   -1.5492    1.0328
         0         0         0   -1.1547
```

7 题目



7. 已知方程组

$$\begin{pmatrix} 3 & -1 & & & \\ -1 & 3 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 3 & -1 \\ & & & -1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ \vdots \\ 1 \\ 2 \end{pmatrix}$$

分别用 Jacobi 迭代法和 Gauss-Seidel 迭代法求解方程组，精确到小数点后 6 位，分别就 $n=10,20,30,50,100$ 给出相应的计算结果.

Jacobi 迭代法

```m
function [x] = JacobiIterate(A, b, x0, tol, max_iter)
    %JacobiIterate  Jacobi 迭代求解线性方程组
    %   A: 系数矩阵
    %   b: 常数向量
    %   x0: 初始猜测
    %   tol: 容差
    %   max_iter: 最大迭代次数

    n = size(A, 1);
    D = diag(diag(A));
    L = D - A;

    for k = 1:max_iter
        x = D \ (b + L * x0);
        if norm(x - x0, inf) < tol
```

```
            return;
        end
        x0 = x;
    end
    warning('未在最大迭代次数内达到指定容差，返回当前近似值');
end
```

Gauss-Seidel 迭代法

```
function [x] = GaussIterate(A, b, x0, tol, max_iter)
    %GaussIterate  Gauss 迭代求解线性方程组
    %   A: 系数矩阵
    %   b: 常数向量
    %   x0: 初始猜测
    %   tol: 容差
    %   max_iter: 最大迭代次数

    n = size(A, 1);
    L = tril(A);
    U = triu(A, 1);

    for k = 1:max_iter
        x = L \ (b - U * x0);
        if norm(x - x0, inf) < tol
            return;
        end
        x0 = x;
    end
    warning('未在最大迭代次数内达到指定容差，返回当前近似值');
end
```

主函数

```
function C2_7
    format long;

    n = [10 20 30 50 100];
    A = {};
    b = {};

    for i = 1:length(n)
        Ai = 3 .* eye(n(i));
        % 主对角线上、下的对角线设为-1
        for j = 1:n(i) - 1
            Ai(j, j + 1) = -1;
            Ai(j + 1, j) = -1;
```

```
        end

        A{i} = Ai;

        bi = ones(n(i), 1);
        bi(1) = 2;
        bi(n(i)) = 2;
        b{i} = bi;
    end

    disp('Jacobi 迭代----------------------------------------------------
')
    x_Jacobi = {};
    for i = 1:length(n)
        xi = JacobiIterate(A{i}, b{i}, b{i}, 1e-6, 50);
        x_Jacobi{i} = xi;
    end

    disp('Gauss 迭代-----------------------------------------------------')
    x_Gauss = {};
    for i = 1:length(n)
        xi = GaussIterate(A{i}, b{i}, b{i}, 1e-6, 50);
        x_Gauss{i} = xi;
    end

    x_true = {};
    for i = 1:length(n)
        x_true{i} = A{i} \ b{i};
    end

    % 对比
    for i = 1:length(n)
        [x_Jacobi{i}, x_Gauss{i}, x_true{i}]
    end

end
```

结果

n=10 时 Jacobi 迭代、Gauss 迭代和解析解：

```
ans =

   1.000000350854475   1.000000350854475   1.000000000000000
   1.000000874464724   1.000000446406331   1.000000000000000
   1.000000988364520   1.000000414058047   1.000000000000000
   1.000001390660735   1.000000329756499   1.000000000000000
   1.000001398938906   1.000000236235230   1.000000000000000
   1.000001398938906   1.000000154572005   1.000000000000000
   1.000001390660735   1.000000092346840   1.000000000000000
   1.000000988364520   1.000000049572291   1.000000000000000
   1.000000874464724   1.000000022787434   1.000000000000000
   1.000000350854475   1.000000007595811   1.000000000000000
```

n=20 时 Jacobi 迭代、Gauss 迭代和解析解：

```
ans =

   1.000000001580335   1.000000000000000   1.000000000000000
   1.000000350718748   1.000000000000000   1.000000000000000
   1.000000008242764   1.000000007743524   1.000000000000000
   1.000000569917965   1.000000043879971   1.000000000000000
   1.000000028227821   1.000000130779523   1.000000000000000
   1.000000603442551   1.000000272457339   1.000000000000000
   1.000000077635753   1.000000445013654   1.000000000000000
   1.000000491693931   1.000000606066214   1.000000000000000
   1.000000174680739   1.000000714847330   1.000000000000000
   1.000000323482849   1.000000748887679   1.000000000000000
   1.000000323482849   1.000000709173938   1.000000000000000
   1.000000174680739   1.000000614617413   1.000000000000000
   1.000000491693931   1.000000491693931   1.000000000000000
   1.000000077635753   1.000000365045494   1.000000000000000
   1.000000603442551   1.000000252055222   1.000000000000000
   1.000000028227821   1.000000161573860   1.000000000000000
   1.000000569917965   1.000000095405517   1.000000000000000
   1.000000008242764   1.000000050882943   1.000000000000000
   1.000000350718748   1.000000023321349   1.000000000000000
   1.000000001580335   1.000000007773783   1.000000000000000
```

n=30 时 Jacobi 迭代、Gauss 迭代和解析解：

```
ans =

   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000350718748   1.000000000000000   1.000000000000000
   1.000000000000131   1.000000000000000   1.000000000000000
   1.000000569917965   1.000000000000000   1.000000000000000
   1.000000000003409   1.000000000000000   1.000000000000000
   1.000000603442551   1.000000000000000   1.000000000000000
   1.000000000042488   1.000000000000000   1.000000000000000
   1.000000491693931   1.000000000000000   1.000000000000000
   1.000000000337547   1.000000000000000   1.000000000000000
   1.000000323482849   1.000000000000000   1.000000000000000
   1.000000001917883   1.000000000000000   1.000000000000000
   1.000000174680739   1.000000000000000   1.000000000000000
   1.000000008285252   1.000000007743524   1.000000000000000
   1.000000077635884   1.000000043879971   1.000000000000000
   1.000000028231230   1.000000130779523   1.000000000000000
   1.000000028231230   1.000000272457339   1.000000000000000
   1.000000077635884   1.000000445013654   1.000000000000000
   1.000000008285252   1.000000606066214   1.000000000000000
   1.000000174680739   1.000000714847330   1.000000000000000
   1.000000001917883   1.000000748887679   1.000000000000000
   1.000000323482849   1.000000709173938   1.000000000000000
   1.000000000337547   1.000000614617413   1.000000000000000
   1.000000491693931   1.000000491693931   1.000000000000000
   1.000000000042488   1.000000365045494   1.000000000000000
   1.000000603442551   1.000000252055222   1.000000000000000
   1.000000000003409   1.000000161573860   1.000000000000000
   1.000000569917965   1.000000095405517   1.000000000000000
   1.000000000000131   1.000000050882943   1.000000000000000
   1.000000350718748   1.000000023321349   1.000000000000000
   1.000000000000000   1.000000007773783   1.000000000000000
```

n=50 时 Jacobi 迭代、Gauss 迭代和解析解：

```
ans =

   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000350718748   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000569917965   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000603442551   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000491693931   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000323482849   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000174680739   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000077635884   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000028231230   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000008285252   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000001917883   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000337547   1.000000000000000   1.000000000000000
   1.000000000000131   1.000000000000000   1.000000000000000
   1.000000000042488   1.000000000000000   1.000000000000000
   1.000000000003409   1.000000000000000   1.000000000000000
   1.000000000003409   1.000000000000000   1.000000000000000
   1.000000000042488   1.000000000000000   1.000000000000000
   1.000000000000131   1.000000000000000   1.000000000000000
   1.000000000337547   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000001917883   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000008285252   1.000000007743524   1.000000000000000
   1.000000000000000   1.000000043879971   1.000000000000000
   1.000000028231230   1.000000130779523   1.000000000000000
   1.000000000000000   1.000000272457339   1.000000000000000
   1.000000077635884   1.000000445013654   1.000000000000000
   1.000000000000000   1.000000606066214   1.000000000000000
   1.000000174680739   1.000000714847330   1.000000000000000
   1.000000000000000   1.000000748887679   1.000000000000000
   1.000000323482849   1.000000709173938   1.000000000000000
   1.000000000000000   1.000000614617413   1.000000000000000
   1.000000491693931   1.000000491693931   1.000000000000000
   1.000000000000000   1.000000365045494   1.000000000000000
   1.000000603442551   1.000000252055222   1.000000000000000
   1.000000000000000   1.000000161573860   1.000000000000000
   1.000000569917965   1.000000095405517   1.000000000000000
   1.000000000000000   1.000000050882943   1.000000000000000
   1.000000350718748   1.000000023321349   1.000000000000000
   1.000000000000000   1.000000007773783   1.000000000000000
```

n=100 时 Jacobi 迭代、Gauss 迭代和解析解：

```
ans =

   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000350718748   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000569917965   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000603442551   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000491693931   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000323482849   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000174680739   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000077635884   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000028231230   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000008285252   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000001917883   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000337547   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000042488   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000003409   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000131   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000000131   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000003409   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000042488   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000000337547   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000001917883   1.000000000000000   1.000000000000000
   1.000000000000000   1.000000000000000   1.000000000000000
   1.000000008285252   1.000000007743524   1.000000000000000
   1.000000000000000   1.000000043879971   1.000000000000000
   1.000000028231230   1.000000130779523   1.000000000000000
   1.000000000000000   1.000000272457339   1.000000000000000
   1.000000077635884   1.000000445013654   1.000000000000000
   1.000000000000000   1.000000606066214   1.000000000000000
   1.000000174680739   1.000000714847330   1.000000000000000
   1.000000000000000   1.000000748887679   1.000000000000000
   1.000000323482849   1.000000709173938   1.000000000000000
   1.000000000000000   1.000000614617413   1.000000000000000
   1.000000491693931   1.000000491693931   1.000000000000000
   1.000000000000000   1.000000365045494   1.000000000000000
   1.000000603442551   1.000000252055222   1.000000000000000
   1.000000000000000   1.000000161573860   1.000000000000000
   1.000000569917965   1.000000095405517   1.000000000000000
   1.000000000000000   1.000000050882943   1.000000000000000
   1.000000350718748   1.000000023321349   1.000000000000000
   1.000000000000000   1.000000007773783   1.000000000000000
```

三、非线性方程求解

2 题目

2. 采用二分法计算非线性方程 $x\cos x+2=0$，查找区间为$[-4,4]$. 取不同的初值用 Newton 迭代法以及弦截法求方程 $x^3+2x^2+10x-100=0$ 的实根，列表或者画图说明收敛速度.

二分查找

```
function [root] = BinaryIterate(f, x0, x1, tol)
    %BinaryIterate  二分查找求根
    %   f: 目标函数
    %   x0: 区间左端点
    %   x1: 区间右端点
    %   tol: 容差
    f_numeric = matlabFunction(f);

    while abs(x1 - x0) > tol
        fx0 = f_numeric(x0);
        fx1 = f_numeric(x1);
        xm = (x0 + x1) / 2;
        fxm = f_numeric(xm);

        if fxm == 0
            root = xm;
            return
        else
            if fx0 * fxm < 0
                x1 = xm;
            else
                x0 = xm;
            end
        end
    end
    root = (x0 + x1) / 2;
end
```

Newton 迭代法

```
function [root] = NewtonIterate(f, x, x0, tol, max_iter)
    %NewtonIterate  Newton 迭代求根
    %   f: 目标函数
    %   x: 函数变量
```

```
    %   x0: 初始值
    %   tol: 容差
    %   max_iter: 最大迭代次数

    df = diff(f, x);
    f_numeric = matlabFunction(f);
    df_numeric = matlabFunction(df);

    for i = 1:max_iter
        x1 = x0 - f_numeric(x0) / df_numeric(x0);
        if abs(x1 - x0) < tol
            root = x1;
            return
        end
        x0 = x1;
    end

    root = x1;
    warning('未在最大迭代次数内达到指定容差，返回当前近似值');
end
```

Newton 弦截法（割线法）

```
function [root] = NewtonSecantIterate(f, x, x0, x1, tol, max_iter)
    %NewtonSecantIterate  Newton 弦截法迭代求根
    %   f: 目标函数
    %   x: 函数变量
    %   x0: 初始值
    %   tol: 容差
    %   max_iter: 最大迭代次数

    f_numeric = matlabFunction(f);

    for i = 1:max_iter
        xt = x1 - f_numeric(x1) / ((f_numeric(x1) - f_numeric(x0)) / (x1
- x0));
        x0 = x1;
        x1 = xt;
        if abs(x1 - x0) < tol
            root = x1;
            return
        end
    end

    root = x1;
```

```
    warning('未在最大迭代次数内达到指定容差，返回当前近似值');
end
```

主函数

```
function C3_2
    format long;
    syms x;

    disp('二分查找-----------------------------------------')
    f = x * cos(x) + 2;
    x0 = -4;
    x1 = 4;
    tol = 1e-6;
    root = BinaryIterate(f, x0, x1, tol)

    disp('Newton 迭代-----------------------------------------')
    f = x ^ 3 + 2 * x ^ 2 + 10 * x - 100;
    x0 = 0;
    tol = 1e-6;
    max_iter = 10;
    root = NewtonIterate(f, x, x0, tol, max_iter)

    disp('弦截法-----------------------------------------')
    f = x ^ 3 + 2 * x ^ 2 + 10 * x - 100;
    x0 = 0;
    x1 = 1;
    tol = 1e-6;
    max_iter = 10;
    root = NewtonSecantIterate(f, x, x0, x1, tol, max_iter)

    disp('迭代值列表-----------------------------------------')
    Newton = [];
    Secant = [];
    tol = 1e-16; % double 的最大精度
    max_iter = 12;

    for i = 1:max_iter
        Newton(i) = NewtonIterate(f, x, x0, tol, i);
        Secant(i) = NewtonSecantIterate(f, x, x0, x1, tol, i);
    end

    [Newton', Secant']
    disp('Newton 迭代法收敛更快-----------------------------')
```

```
end
```

结果

```
二分查找----------------------------------------

root =

   2.498755931854248

Newton迭代----------------------------------------

root =

   3.460586726723288

弦截法----------------------------------------

root =

   3.460586726718538
```

第一列为 Newton 迭代法的迭代根，第二列为 Newton 弦截法的迭代根：

```
迭代值列表----------------------------------------

ans =

  10.000000000000000   7.692307692307693
   6.571428571428571   1.913400012424675
   4.546183326887240   2.536639505029803
   3.650755382642228   3.879130592057319
   3.467732584608362   3.375777456579033
   3.460597285868235   3.453539648150463
   3.460586726746385   3.460711985526595
   3.460586726723288   3.460586543702102
   3.460586726723287   3.460586726718538
   3.460586726723288   3.460586726723288
   3.460586726723287   3.460586726723287
   3.460586726723288   3.460586726723288

Newton迭代法收敛更快------------------------------
```

6 题目

6. 设 $f(x)=54x^6+45x^5-102x^4-69x^3+35x^2+16x-4$. 在区间$[-2,2]$上画出函数，(1) 使用 Newton 迭代法找出该区间上的 5 个根，并计算 e_{i+1}/e_i^2 和 e_{i+1}/e_i，由此判断哪个根是 1 阶收敛，哪个根是 2 阶收敛？(2) 使用割线法计算这 5 个根，并判断哪个根是线性收敛，哪个是超线性收敛？

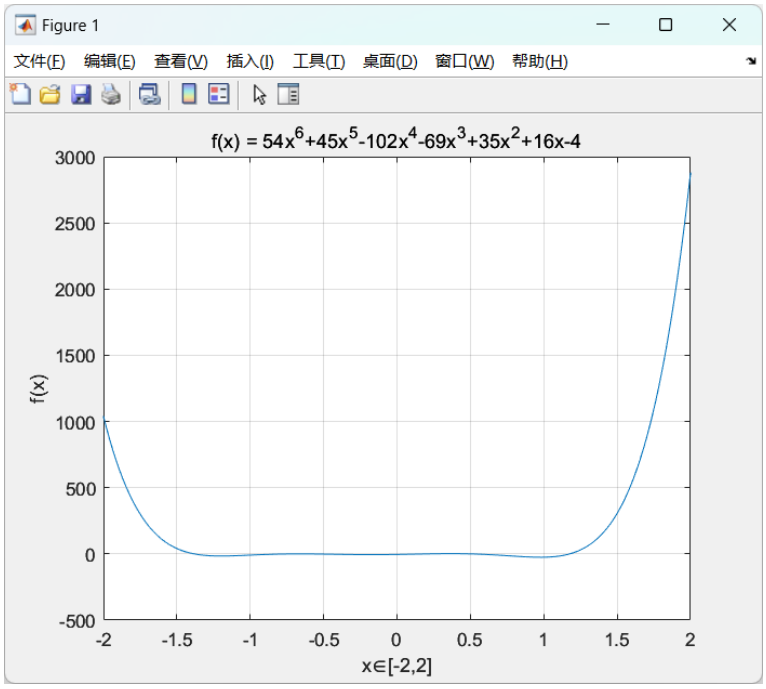
主函数

```
function C3_6
    format long;
    syms x;

    disp('函数图像----------------------------------------')
    f = 54 * x ^ 6 + 45 * x ^ 5 - 102 * x ^ 4 - 69 * x ^ 3 + 35 * x ^ 2
+ 16 * x - 4;
    f_numeric = matlabFunction(f);
    x_value = -2:0.01:2;
    fx_value = f_numeric(x_value);

    plot(x_value, fx_value);
    xlabel('x∈[-2,2]');
    ylabel('f(x)');
    grid on;
    title('f(x) = 54x^6+45x^5-102x^4-69x^3+35x^2+16x-4');

    disp('(1)求5个根----------------------------------------')
    x0 = [-1.5 -0.7 0.2 0.48 1];
    tol = 1e-16;
    max_iter = 20;
    roots = [];

    for i = 1:length(x0)
        roots(i) = NewtonIterate(f, x, x0(i), tol, max_iter);
    end

    roots'

    disp('(1)判断收敛阶数----------------------------------')
    all_roots = [];
    e1 = ones(1, length(x0));
    e2 = ones(1, length(x0));
    max_iter = 10;

    for j = 1:max_iter
        for i = 1:length(x0)
            all_roots(i, j) = NewtonIterate(f, x, x0(i), tol, j);

            if j > 1
                e1(i, j) = (all_roots(i, j) - roots(i)) / (all_roots(i,
j - 1) - roots(i));
```

```
                e2(i, j) = (all_roots(i, j) - roots(i)) / (all_roots(i,
j - 1) - roots(i)) ^ 2;
            end
        end
    end

    e1'
    e2'
    disp('-1.6 附近的根，e1 减小，e2 稳定，二阶收敛----------------------')
    disp('-0.7 附近的根，e1 稳定，e2 增大，一阶收敛----------------------')
    disp(' 0.2 附近的根，e1 减小，e2 稳定，二阶收敛----------------------')
    disp('0.48 附近的根，e1 减小，e2 稳定，二阶收敛----------------------')
    disp(' 1.1 附近的根，e1 减小，e2 稳定，二阶收敛----------------------')

    disp('(1)求 5 个根-------------------------------------')
    x0 = [-1.5 -0.7 0.2 0.48 1];
    x1 = [-1.4 -0.6 0.3 0.6 1.1];
    tol = 1e-16;
    max_iter = 20;
    roots = [];

    for i = 1:length(x0)
        roots(i) = NewtonSecantIterate(f, x, x0(i), x1(i), tol,
max_iter);
    end

    roots'

    disp('(2)判断收敛阶数----------------------------------')
    all_roots = [];
    e1 = ones(1, length(x0));
    e2 = ones(1, length(x0));
    max_iter = 10;

    for j = 1:max_iter
        for i = 1:length(x0)
            all_roots(i, j) = NewtonSecantIterate(f, x, x0(i), x1(i),
tol, j);

            if j > 1
                e1(i, j) = (all_roots(i, j) - roots(i)) / (all_roots(i,
j - 1) - roots(i));
                e2(i, j) = (all_roots(i, j) - roots(i)) / (all_roots(i,
j - 1) - roots(i)) ^ 2;
```

```
            end
        end
    end

    e1'
    e2'
    disp('-1.5 附近的根，e1 减小，e2 增大，超线性收敛----------------------')
    disp('-0.7 附近的根，e1 稳定，e2 增大，线性收敛------------------------')
    disp(' 0.2 附近的根，e1 减小，e2 增大，超线性收敛----------------------')
    disp('0.48 附近的根，e1 减小，e2 增大，超线性收敛----------------------')
    disp(' 1.1 附近的根，e1 减小，e2 增大，超线性收敛----------------------')
end
```

图像



结果

Newton 迭代法求 5 个根：

```
(1)求5个根--------------------
ans =

   -1.381298482043995
   -0.666666699874272
    0.205182924689048
    0.500000000000000
    1.176115557354947
```

判断收敛阶：

```
e1 =

   1.000000000000000                   0                   0                   0                   0
   0.134943890794853   0.504508346762904   0.000056591650149   0.008068012491831   0.684635121904499
   0.021318473168946   0.502359985075023   0.000000038918753   0.000066095373647   0.635771930433572
   0.000466653323663   0.501205055356829                   0                   0   0.563876074938640
   0.000000208419577   0.500603892431907                 NaN                 NaN   0.455839570886392
  -0.042553191489362   0.500292201512927                 NaN                 NaN   0.299864416218488
                   0   0.500123458195361                 NaN                 NaN   0.121738716105077
                 NaN   0.500015651004322                 NaN                 NaN   0.017105468630163
                 NaN   0.499915462882648                 NaN                 NaN   0.000298883591508
                 NaN   0.499772909323914                 NaN                 NaN   0.000000089120634


e2 =

   1.0e+12 *

   0.000000000001000                   0                   0                   0                   0
  -0.000000000003618  -0.000000000029781  -0.000000000001497   0.000000000004095   0.000000000000638
  -0.000000000004236  -0.000000000058778  -0.000000000018191   0.000000000004158   0.000000000000865
  -0.000000000004349  -0.000000000116735                   0                   0   0.000000000001206
  -0.000000000004162  -0.000000000232628                 NaN                 NaN   0.000000000001729
   4.077500794359888  -0.000000000464406                 NaN                 NaN   0.000000000002496
                   0  -0.000000000927957                 NaN                 NaN   0.000000000003379
                 NaN  -0.000000001855056                 NaN                 NaN   0.000000000003900
                 NaN  -0.000000003709252                 NaN                 NaN   0.000000000003984
                 NaN  -0.000000007417643                 NaN                 NaN   0.000000000003974

-1.6附近的根，e1减小，e2稳定，二阶收敛----------------------------------
-0.7附近的根，e1稳定，e2增大，一阶收敛----------------------------------
 0.2附近的根，e1减小，e2稳定，二阶收敛----------------------------------
0.48附近的根，e1减小，e2稳定，二阶收敛----------------------------------
 1.1附近的根，e1减小，e2稳定，二阶收敛----------------------------------
```

Newton 弦截法求 5 个根：

```
(1)求5个根------------------
ans =

  -1.381298482043995
  -0.666740686180230
   0.205182924689048
   0.500000000000000
   1.176115557354947
```

判断收敛阶：

```
e1 =

   1.000000000000000                   0                   0                   0                   0
   0.076054840058795  -5.811295913494275  -0.255821461112363   0.299538652959934  -0.249865721313480
   0.029703539258492  -0.272550818704181  -0.001923455894840  -0.026811664914900   0.430053497254591
   0.002301892293159   2.197858558565303   0.000488333726637  -0.007831482465008  -0.159126897752024
   0.000068472077347   0.319499420848889  -0.000000813994114   0.000208432279853  -0.061855052631219
                   0   0.774195449709394  -0.111111111111111                   0   0.009482967544312
                 NaN   0.579168335043193                   0                 NaN  -0.000590076700610
                 NaN   0.645559331824936                 NaN                 NaN  -0.000005630179262
                 NaN   0.614887567810348                 NaN                 NaN                   0
                 NaN   0.622672848549346                 NaN                 NaN                 NaN


e2 =

   1.0e+14 *

   0.000000000000010                   0                   0                   0                   0
  -0.000000000000109   0.000000000000800  -0.000000000002003  -0.000000000000480  -0.000000000000018
  -0.000000000000561  -0.000000000000006   0.000000000000059   0.000000000000143  -0.000000000000123
  -0.000000000001463  -0.000000000000191   0.000000000007771  -0.000000000001562   0.000000000000106
  -0.000000000018903  -0.000000000000013  -0.000000000026525  -0.000000000005309  -0.000000000000259
                   0  -0.000000000000096   4.447999631970860                   0  -0.000000000000641
                 NaN  -0.000000000000093                   0                 NaN   0.000000000004204
                 NaN  -0.000000000000178                 NaN                 NaN  -0.000000000067981
                 NaN  -0.000000000000263                 NaN                 NaN                   0
                 NaN  -0.000000000000433                 NaN                 NaN                 NaN

-1.5附近的根，e1减小，e2增大，超线性收敛----------------------------------
-0.7附近的根，e1稳定，e2增大，线性收敛------------------------------------
 0.2附近的根，e1减小，e2增大，超线性收敛----------------------------------
0.48附近的根，e1减小，e2增大，超线性收敛----------------------------------
 1.1附近的根，e1减小，e2增大，超线性收敛----------------------------------
```

四、插值与逼近

1 题目

1. 已知函数 $f(x)=\frac{1}{1+x^2}$，在$[-5,5]$上分别取 $2,1,\frac{1}{2}$ 为单位长度的等距节点作为插值节点，用 Lagrange 方法插值，并把原函数图与插值函数图比较，观察插值效果.

Lagrange 插值

```m
function [f] = LagrangeInterpolation(x, xi, yi)
    %LagrangeInterpolation    Lagrange 插值
    %   x: 函数变量
    %   xi, yi: 插值节点
    n = length(xi);
    f = zeros(size(x));

    for i = 1:n
        Li = ones(size(x));
        for j = [1:i - 1, i + 1:n]
            Li = Li .* (x - xi(j)) / (xi(i) - xi(j));
        end
        f = f + yi(i) * Li;
    end
```

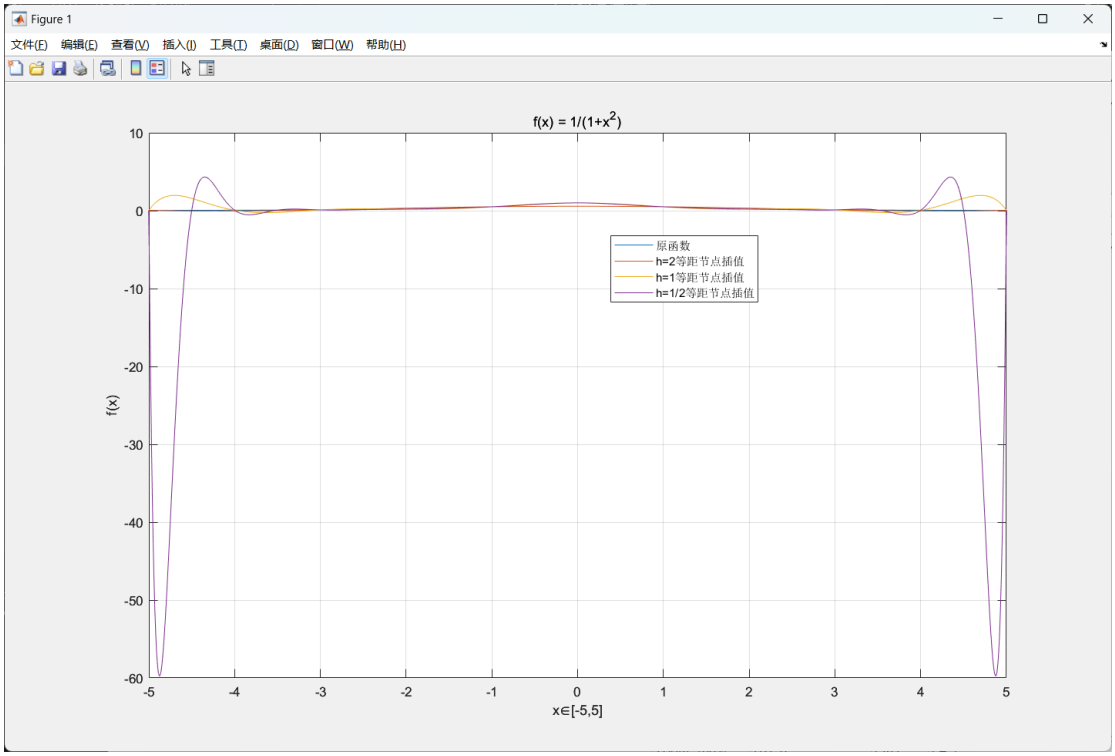
```
end
```

主函数

```
function C4_1
    format long;
    syms x;

    f = 1 / (1 + x ^ 2);
    f_numeric = matlabFunction(f);

    disp('蓝色原函数图像-------------------------------------------')
    x_value = -5:0.01:5;
    fx_value = f_numeric(x_value);
    plot(x_value, fx_value);
    xlabel('x∈[-5,5]');
    ylabel('f(x)');
    grid on;
    title('f(x) = 1/(1+x^2)');
    hold on;

    disp('橙色 h=2 等距节点插值-------------------------------------')
    x_interpolation = -5:2:5;
    fx_interpolation = f_numeric(x_interpolation);
    f_lagrange = LagrangeInterpolation(x, x_interpolation,
fx_interpolation);
    f_lagrange_numeric = matlabFunction(f_lagrange);
    fx_lagrange_value = f_lagrange_numeric(x_value);
    plot(x_value, fx_lagrange_value);
    hold on;

    disp('黄色 h=1 等距节点插值-------------------------------------')
    x_interpolation = -5:1:5;
    fx_interpolation = f_numeric(x_interpolation);
    f_lagrange = LagrangeInterpolation(x, x_interpolation,
fx_interpolation);
    f_lagrange_numeric = matlabFunction(f_lagrange);
    fx_lagrange_value = f_lagrange_numeric(x_value);
    plot(x_value, fx_lagrange_value);
    hold on;

    disp('紫色 h=1/2 等距节点插值-----------------------------------')
    x_interpolation = -5:1/2:5;
    fx_interpolation = f_numeric(x_interpolation);
```

```
    f_lagrange = LagrangeInterpolation(x, x_interpolation,
fx_interpolation);
    f_lagrange_numeric = matlabFunction(f_lagrange);
    fx_lagrange_value = f_lagrange_numeric(x_value);
    plot(x_value, fx_lagrange_value);

    legend('原函数', 'h=2 等距节点插值', 'h=1 等距节点插值', 'h=1/2 等距节点
插值');
    disp('插值节点越多，插值多项式次数越高，在某些局部拟合效果越好，但在其它局
部可能出现振荡')
end
```

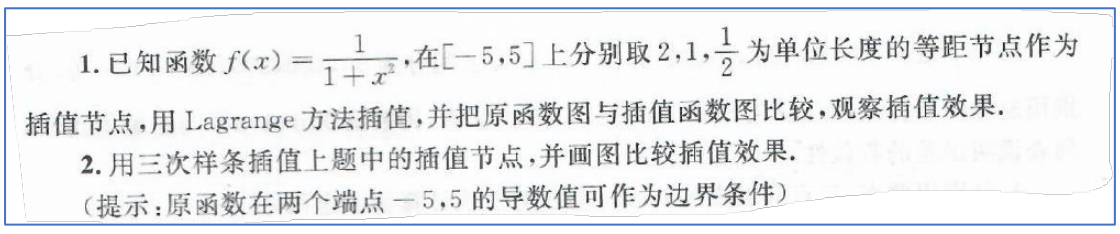
函数图像



插值效果

插值节点越多，插值多项式次数越高，在某些局部拟合效果越好，但在其它局部可能出现振荡。

2 题目



1. 已知函数 $f(x)=\frac{1}{1+x^2}$，在$[-5,5]$上分别取 $2,1,\frac{1}{2}$ 为单位长度的等距节点作为插值节点，用 Lagrange 方法插值，并把原函数图与插值函数图比较，观察插值效果.

2. 用三次样条插值上题中的插值节点，并画图比较插值效果.

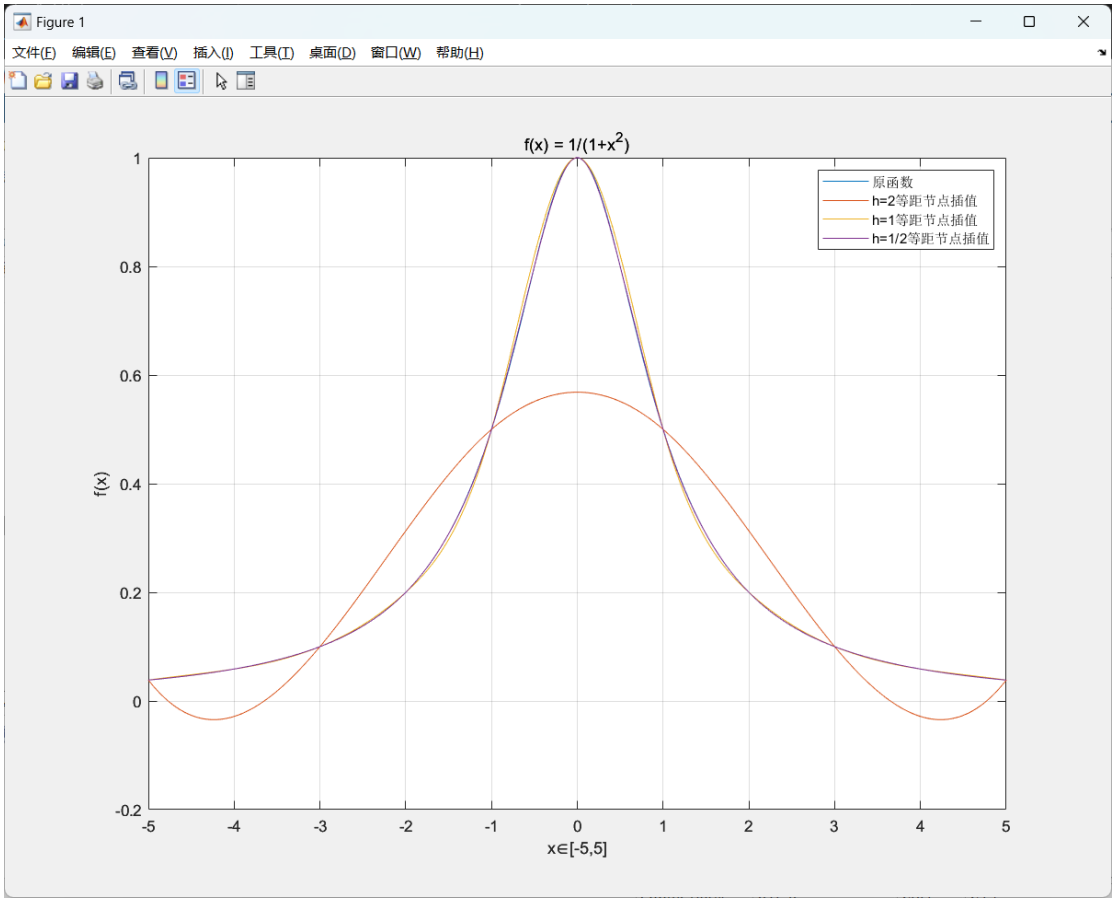
（提示：原函数在两个端点 $-5,5$ 的导数值可作为边界条件）

主函数

```
function C4_2
    format long;
    syms x;

    f = 1 / (1 + x ^ 2);
    f_numeric = matlabFunction(f);

    disp('蓝色原函数图像----------------------------------------------')
    x_value = -5:0.01:5;
    fx_value = f_numeric(x_value);
    plot(x_value, fx_value);
    xlabel('x∈[-5,5]');
    ylabel('f(x)');
    grid on;
    title('f(x) = 1/(1+x^2)');
    hold on;

    disp('橙色 h=2 等距节点插值---------------------------------------')
    x_interpolation = -5:2:5;
    fx_interpolation = f_numeric(x_interpolation);
    fx_spline_value = spline(x_interpolation, fx_interpolation,
x_value);
    plot(x_value, fx_spline_value);
    hold on;

    disp('黄色 h=1 等距节点插值---------------------------------------')
    x_interpolation = -5:1:5;
    fx_interpolation = f_numeric(x_interpolation);
    fx_spline_value = spline(x_interpolation, fx_interpolation,
x_value);
    plot(x_value, fx_spline_value);
    hold on;

    disp('紫色 h=1/2 等距节点插值-------------------------------------')
    x_interpolation = -5:1/2:5;
    fx_interpolation = f_numeric(x_interpolation);
    fx_spline_value = spline(x_interpolation, fx_interpolation,
x_value);
    plot(x_value, fx_spline_value);

    legend('原函数', 'h=2 等距节点插值', 'h=1 等距节点插值', 'h=1/2 等距节点
插值');
```

```
    disp('三次样条插值，插值节点越多，拟合效果越好----------------------')
end
```

函数图像



插值效果

三次样条插值，插值节点越多，拟合效果越好。

6 题目

6. 全世界石油产量(单位:百万桶/日)见表 2 所示,确定并画出经过这些点的 9 阶多项式,并使用该多项式估计 2010 年的石油产量. Runge 现象在这个例子中出现了吗? 以你的观点,插值多项式是描述这些数据好的模型吗? 请解释.

表 2 全世界石油产量

年	产量/(百万桶/日)	年	产量/(百万桶/日)
1994	67.052	1999	72.063
1995	68.008	2000	74.669
1996	69.803	2001	74.487
1997	72.024	2002	74.065
1998	73.400	2003	76.777

用最小二乘法拟合题目中的 10 个数据点,拟合曲线为(1)直线;(2)抛物线,以及(3)三次曲线,并计算它们的均方误差. 使用所得到的拟合曲线估计 2010 年的产量,在均方误差意义下,哪个拟合最好?

主函数

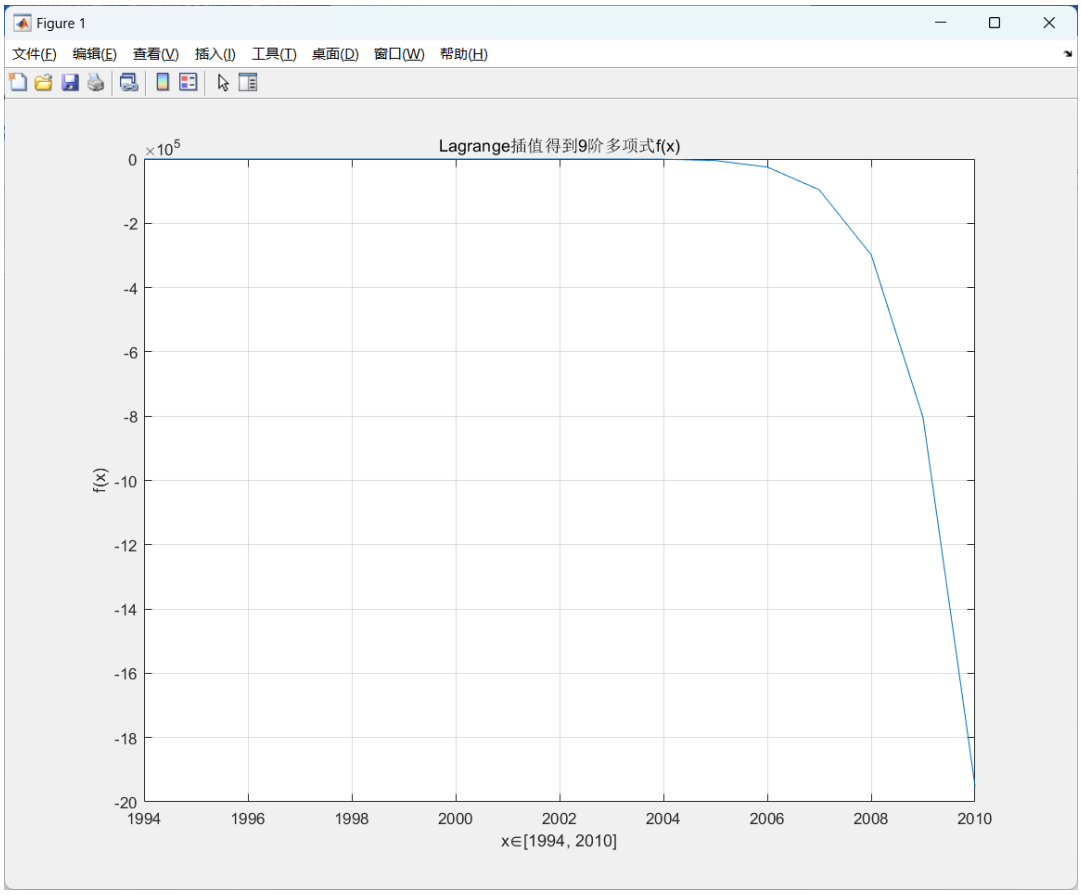
```
function C4_6
    format long;
    syms x;

    x_value = 1994:2003;
    fx_value = [67.052 68.008 69.803 72.024 73.400 72.063 74.669 74.487
74.065 76.777];

    disp('Lagrange 插值求 9 阶多项式--------------------------------------')
    f_lagrange = LagrangeInterpolation(x, x_value, fx_value);
    f_lagrange_numeric = matlabFunction(f_lagrange);
    fx_lagrange_value = f_lagrange_numeric(2010)

    x_interpolation = 1994:2010;
    fx_interpolation_value = f_lagrange_numeric(x_interpolation);
    plot(x_interpolation, fx_interpolation_value);
    xlabel('x∈[1994, 2010]');
    ylabel('f(x)');
    grid on;
    title('Lagrange 插值得到 9 阶多项式 f(x)');
    disp('2005 年起产量预测下降，2010 年产量预测为负，Runge 现象出现，插值多项
式不是好的模型')

    disp('(1)橙色最小二乘直线拟合--------------------------------------')
    p1 = polyfit(x_value, fx_value, 1);
    p1_value = polyval(p1, x_interpolation);
    figure;
```

```
    plot(x_value, fx_value, 'o', x_interpolation, p1_value, '-');
    grid on;
    hold on;

    disp('(2)紫色最小二乘抛物线拟合-----------------------------------')
    p2 = polyfit(x_value, fx_value, 2);
    p2_value = polyval(p2, x_interpolation);
    plot(x_value, fx_value, 'o', x_interpolation, p2_value, '-');
    grid on;
    hold on;

    disp('(3)蓝色最小二乘三次拟合-----------------------------------')
    p3 = polyfit(x_value, fx_value, 3);
    p3_value = polyval(p3, x_interpolation);
    plot(x_value, fx_value, 'o', x_interpolation, p3_value, '-');
    grid on;

    legend('原始数据', '最小二乘直线拟合', '最小二乘抛物线拟合', '最小二乘三
次拟合');

    disp('计算均方误差-------------------------------------------------')
    p1_mean = mean((fx_value - p1_value(1:length(fx_value))) .^ 2)
    p2_mean = mean((fx_value - p2_value(1:length(fx_value))) .^ 2)
    p3_mean = mean((fx_value - p3_value(1:length(fx_value))) .^ 2)
    disp('在均方误差意义下，最小二乘三次拟合最好------------------------')
end
```

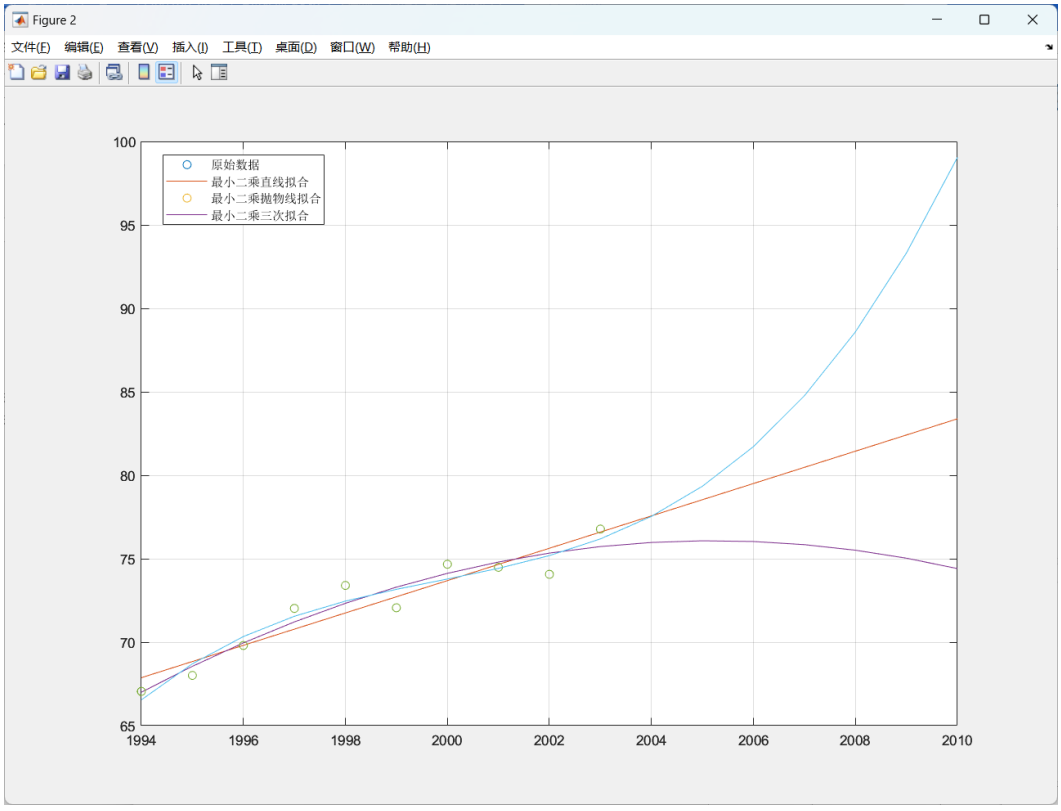
Lagrange 插值结果

```
Lagrange插值求9阶多项式-----------------------------------

fx_lagrange_value =

    -1.951646134001173e+06

2005年起产量预测下降，2010年产量预测为负，Runge现象出现，插值多项式不是好的模型
```

Lagrange 插值 9 阶多项式图像



最小二乘拟合图像



最小二乘均方误差

```
计算均方误差-----------------------------------

p1_mean =

   0.953197025454521


p2_mean =

   0.676796873180650


p3_mean =

   0.574336190032745

在均方误差意义下，最小二乘三次拟合最好----------
```

五、数值积分

3 题目

3. 令 $f(x)=\mathrm{e}^{3x}\cos \pi x$，考虑积分$\int_0^{2\pi} f(x)\mathrm{d}x$. 区间分为 50，100，200，500，1 000 等，分别用复化梯形公式以及复化 Simpson 公式计算积分值，将数值积分的结果与精确值比较，列表说明误差的收敛性.

主函数

```
function C5_3
    format long;
    syms x;

    f = exp(3 * x) * cos(pi * x);
    f_numeric = matlabFunction(f);

    h = 2 * pi ./ [50 100 200 500 1000];
    x_array = {};
    fx_array = {};
    for i = 1:length(h)
        x_array{i} = 0:h(i):2 * pi;
        fx_array{i} = f_numeric(x_array{i});
    end

    sum_T = [];
    sum_S = [];
```

```
    for i = 1:length(h)
        ni = length(x_array{i});
        % 复化梯形
        sum = 0;
        for j = 1:ni - 1
            sum = sum + fx_array{i}(j) + fx_array{i}(j + 1);
        end
        sum = (x_array{i}(ni) - x_array{i}(1)) / (2 * (ni - 1)) * sum;
        sum_T(i) = sum;

        % 复化 Simpson
        sum = 0;
        for j = 1:2:ni - 2
            sum = sum + fx_array{i}(j) + 4 * fx_array{i}(j + 1) +
fx_array{i}(j + 2);
        end
        sum = (x_array{i}(ni) - x_array{i}(1)) / (6 * (ni - 1) / 2) *
sum;
        sum_S(i) = sum;
    end

    % 对比
    f_integral = exp(3 * x) * (3 * cos(pi * x) + pi * sin(pi * x)) / (9
+ pi ^ 2);
    f_integral_numeric = matlabFunction(f_integral);
    integral_value = f_integral_numeric(2 * pi) - f_integral_numeric(0);
    sum_true = ones(1, 5) * integral_value;
    [sum_T', sum_S', sum_true'];
    % 误差
    [sum_true' - sum_T', sum_true' - sum_S']
    disp('复化 Simpson 公式的误差比复化梯形公式小------------------------
')
end
```

结果

第一列为复化梯形求积公式的误差，第二列为复化 Simpson 求积公式的误差：

```
ans =

   1.0e+05 *

   1.071421668633372   0.172912433976457
   0.275921618148983   0.010754934654236
   0.069483834370822   0.000671239778176
   0.011139958760813   0.000017176738158
   0.002785794802308   0.000001073483154

复化Simpson公式的误差比复化梯形公式小------------------------
```

六、微分方程数值解

1 题目

1. 已知常微分方程

$$\begin{cases}\dfrac{du}{dx}=\dfrac{2}{x}u+x^2e^x\\ x\in[1,2],u(1)=0\end{cases}$$

分别用 Euler 法，改进的 Euler 法，Runge-Kutta 法去求解该方程，步长选为 0.1，0.05，0.01. 画图观察求解效果.

主函数

```
function C6_1
    format long;
    syms x u;

    f = 2 * u / x + x ^ 2 * exp(x);
    f_numeric = matlabFunction(f); % f(u,x)
    u1 = 0;

    h = [0.1 0.05 0.01];
    x_array = {};
    for i = 1:length(h)
        x_array{i} = 1:h(i):2;
    end

    ux_value_Euler = {};
    ux_value_Euler_improved = {};
    ux_value_Runge = {};
    for i = 1:length(h)
        ni = length(x_array{i});
        ux_value_Euler{i}(1) = u1;
        ux_value_Euler_improved{i}(1) = u1;
        ux_value_Runge{i}(1) = u1;

        % Euler
        for j = 1:ni - 1
            uj = ux_value_Euler{i}(j);
            xj = x_array{i}(j);
            ux_value_Euler{i}(j + 1) = uj + h(i) * f_numeric(uj, xj);

            % 改进的 Euler
```

```
            uj_1 = ux_value_Euler{i}(j + 1);
            xj_1 = x_array{i}(j + 1);
            ux_value_Euler_improved{i}(j + 1) = uj + 0.5 * h(i) *
(f_numeric(uj, xj) + f_numeric(uj_1, xj_1));
        end

        % Runge-Kutta
        for j = 1:ni - 1
            uj = ux_value_Runge{i}(j);
            xj = x_array{i}(j);
            k1 = f_numeric(uj, xj);
            xj_h_2 = xj + h(i) / 2;
            k2 = f_numeric(uj + h(i) * k1 / 2, xj_h_2);
            k3 = f_numeric(uj + h(i) * k2 / 2, xj_h_2);
            k4 = f_numeric(uj + h(i) * k3, xj + h(i));
            ux_value_Runge{i}(j + 1) = uj + h(i) * (k1 + 2 * k2 + 2 * k3
+ k4) / 6;
        end
    end

    % 对比
    for i = 1:length(h)
        figure;
        [x_array{i}', ux_value_Euler{i}', ux_value_Euler_improved{i}',
ux_value_Runge{i}']
        disp('蓝色 Euler 法----------------------------------------------')
        plot(x_array{i}, ux_value_Euler{i});
        hold on;
        disp('橙色改进的 Euler 法----------------------------------------')
        plot(x_array{i}, ux_value_Euler_improved{i});
        hold on;
        disp('黄色 Runge-Kutta 法----------------------------------------')
        plot(x_array{i}, ux_value_Runge{i});
        grid on;
        legend('Euler', 'Euler improved', 'Runge-Kutta');
    end
end
```

函数图像

步长为 0.1 时：

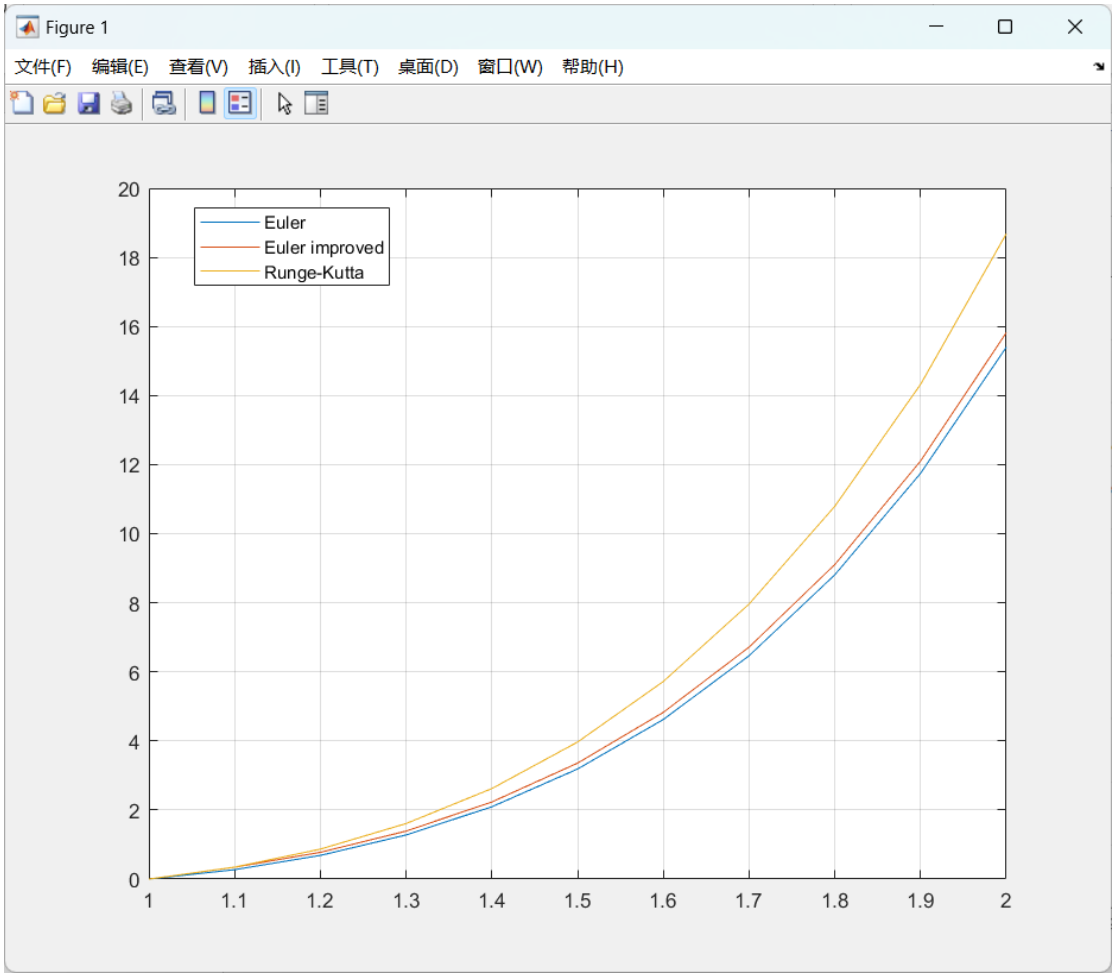


Figure 1
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Euler
Euler improved
Runge-Kutta
20
18
16
14
12
10
8
6
4
2
0
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2

步长为 0.05 时：

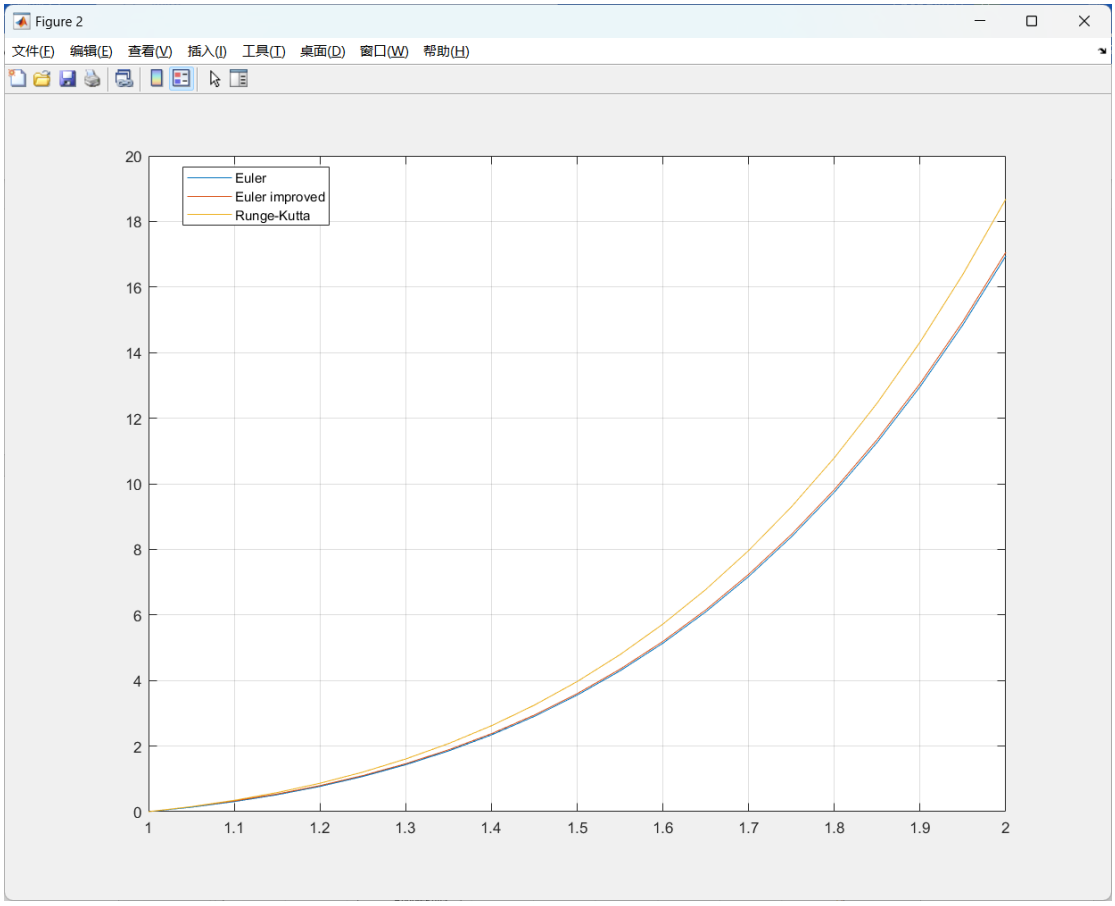


Figure 2
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Euler
Euler improved
Runge-Kutta
20
18
16
14
12
10
8
6
4
2
0
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2

步长为 0.01 时：

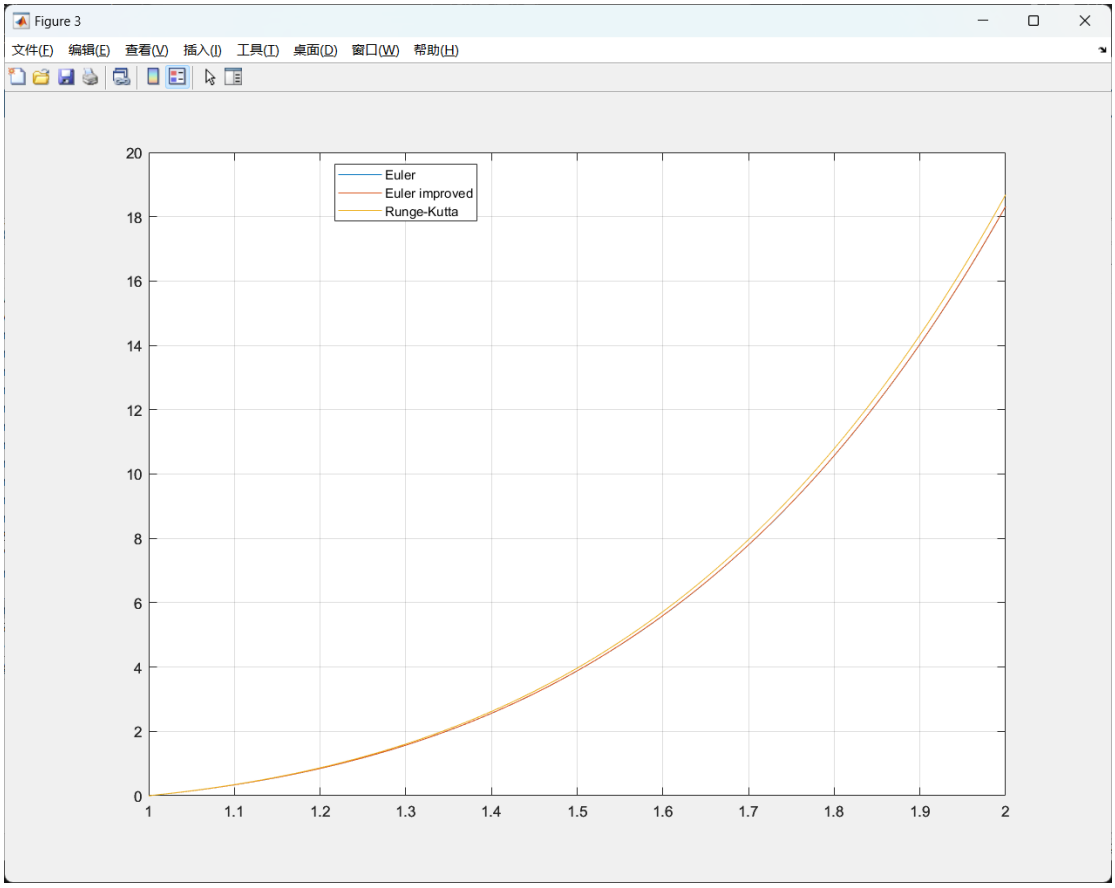


Figure 3
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Euler
Euler improved
Runge-Kutta
20
18
16
14
12
10
8
6
4
2
0
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2