**目录**

[《计算机科学计算》 2](#_Toc153740673)

[第四章 2](#_Toc153740674)

[12题目 2](#_Toc153740675)

[17题目 3](#_Toc153740676)

[第六章 4](#_Toc153740677)

[15题目 4](#_Toc153740678)

[《数值分析方法与应用》 6](#_Toc153740679)

[一、基础知识部分 6](#_Toc153740680)

[1题目 6](#_Toc153740681)

[2题目 7](#_Toc153740682)

[二、线性方程组求解 8](#_Toc153740683)

[1题目 8](#_Toc153740684)

[2题目 15](#_Toc153740685)

[4题目 16](#_Toc153740686)

[7题目 18](#_Toc153740687)

[三、非线性方程求解 25](#_Toc153740688)

[2题目 25](#_Toc153740689)

[6题目 28](#_Toc153740690)

[四、插值与逼近 33](#_Toc153740691)

[1题目 33](#_Toc153740692)

[2题目 35](#_Toc153740693)

[6题目 38](#_Toc153740694)

[五、数值积分 41](#_Toc153740695)

[3题目 41](#_Toc153740696)

[六、微分方程数值解 43](#_Toc153740697)

[1题目 43](#_Toc153740698)

# 《计算机科学计算》

## 第四章

### 12题目

文本, 信件

描述已自动生成

#### Newton迭代法代码见[Newton迭代法](#_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;

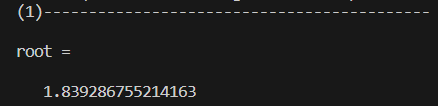
    tol = 1e-5;

    max\_iter = 3;

    root = NewtonIterate(f, x, x0, tol, max\_iter)

end

#### （1）结果



#### （2）结果

图片包含 图形用户界面

描述已自动生成

### 17题目

文本, 信件

描述已自动生成

#### 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);

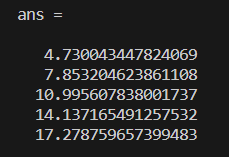
        root\_array(i) = root;

    end

    root\_array'

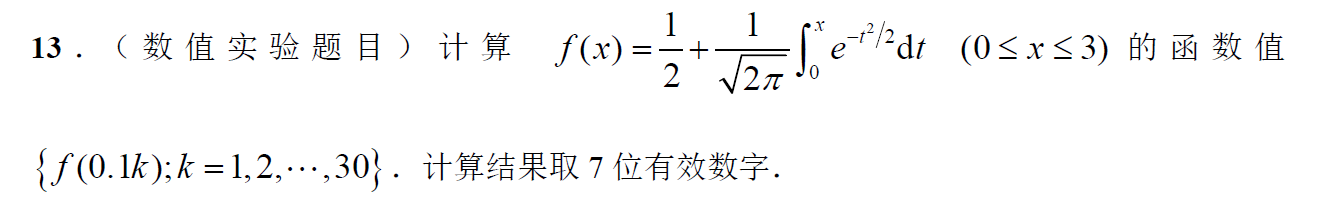
end

#### 结果



## 第六章

### 15题目



#### 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

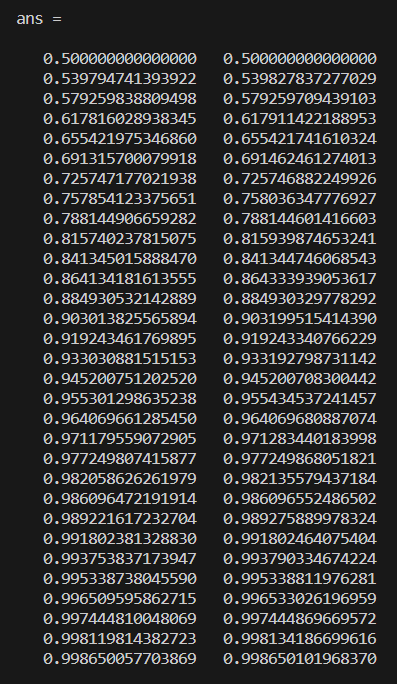
    [integral\_value', true\_integral']

    [f', true\_f']

end

#### 结果

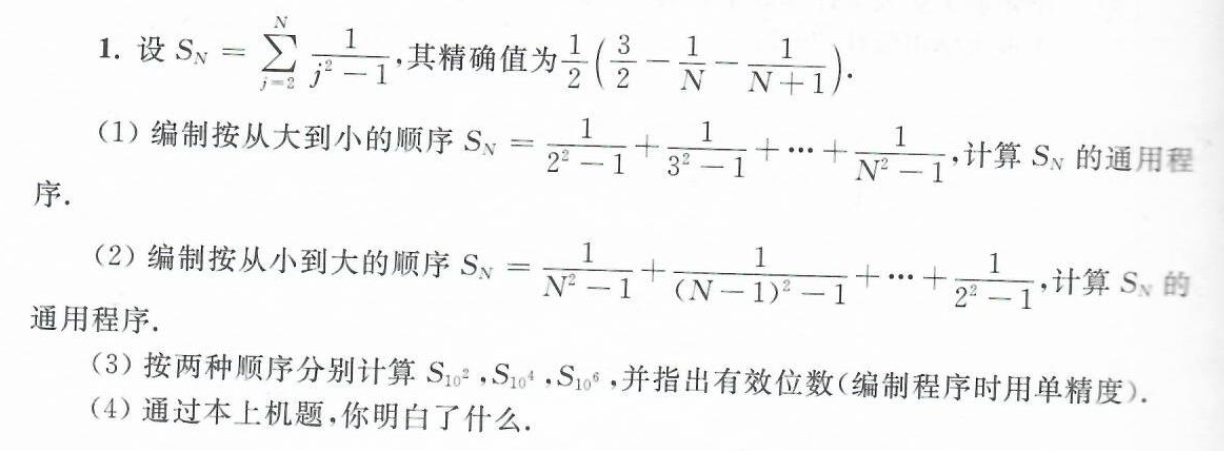
第一列为数值积分结果，第二列为matlab的计算值：



# 《数值分析方法与应用》

## 一、基础知识部分

### 1题目



#### （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

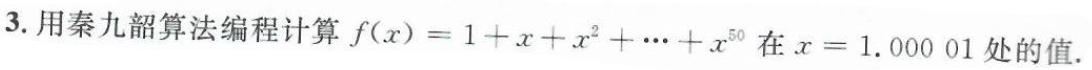
#### （3）结果

文本

描述已自动生成

#### （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



#### 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

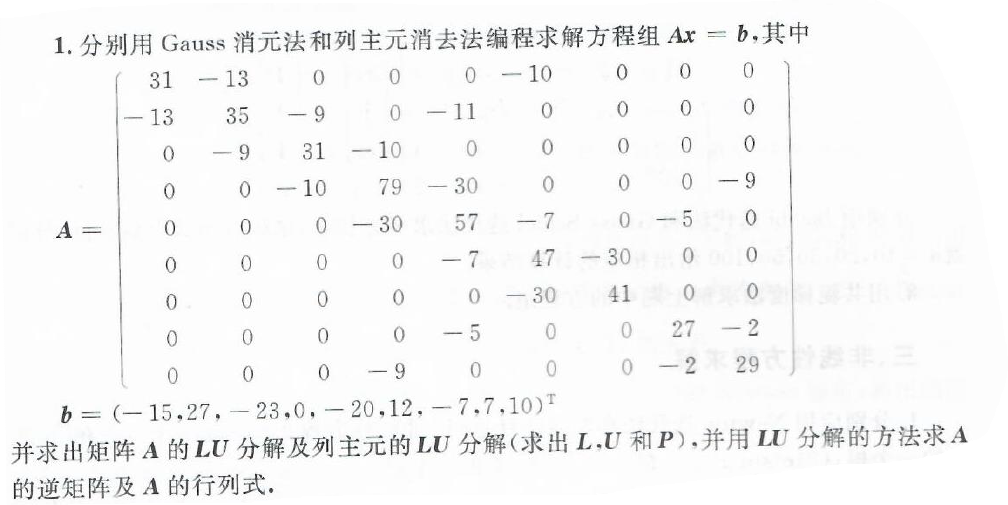
#### 结果

文本

描述已自动生成

## 二、线性方程组求解

### 1题目



#### 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

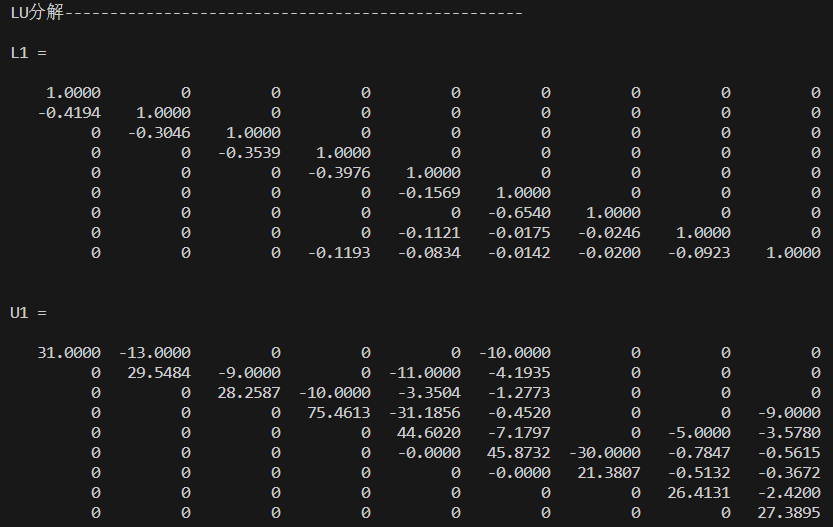
    [det\_A, det(A)]'

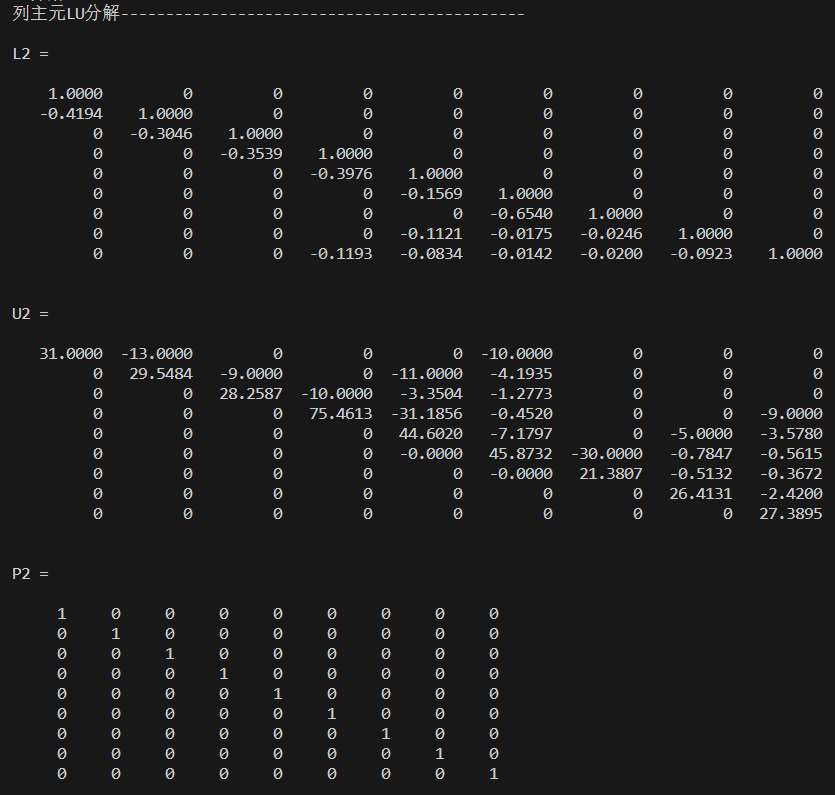
end

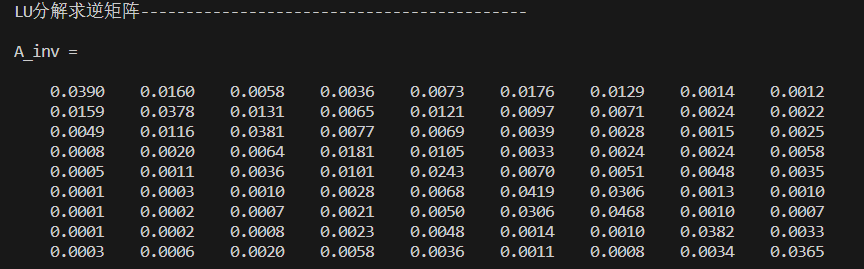
#### 结果

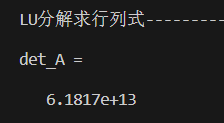
形状

低可信度描述已自动生成









### 2题目

图片包含 图示

描述已自动生成

#### 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);

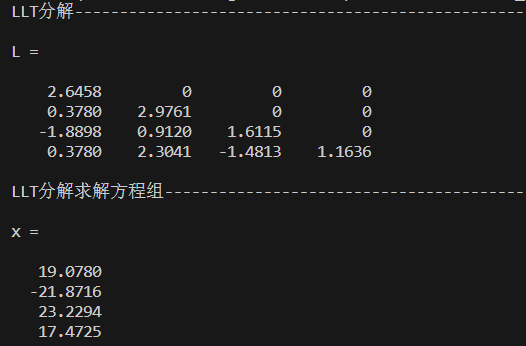
    x = BackwardSubstitution(LT, y);

    % 对比

    [x, A \ b]

end

#### 结果



### 4题目

日历

中度可信度描述已自动生成

#### 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);

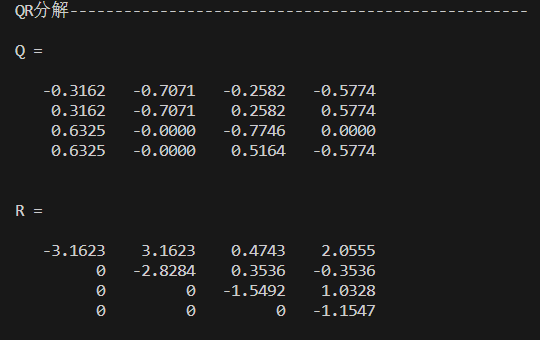
    % 对比

    [Q, q]

    [R, r]

end

#### 结果



### 7题目

图示

低可信度描述已自动生成

#### Jacobi迭代法

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)

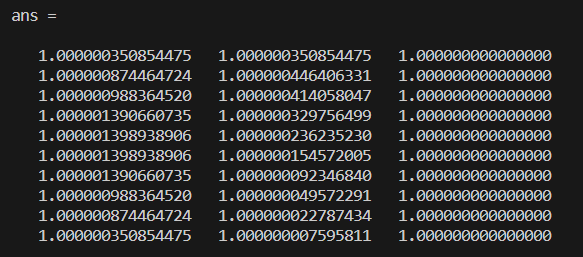
        [x\_Jacobi{i}, x\_Gauss{i}, x\_true{i}]

    end

end

#### 结果

n=10时Jacobi迭代、Gauss迭代和解析解：



n=20时Jacobi迭代、Gauss迭代和解析解：

图形用户界面

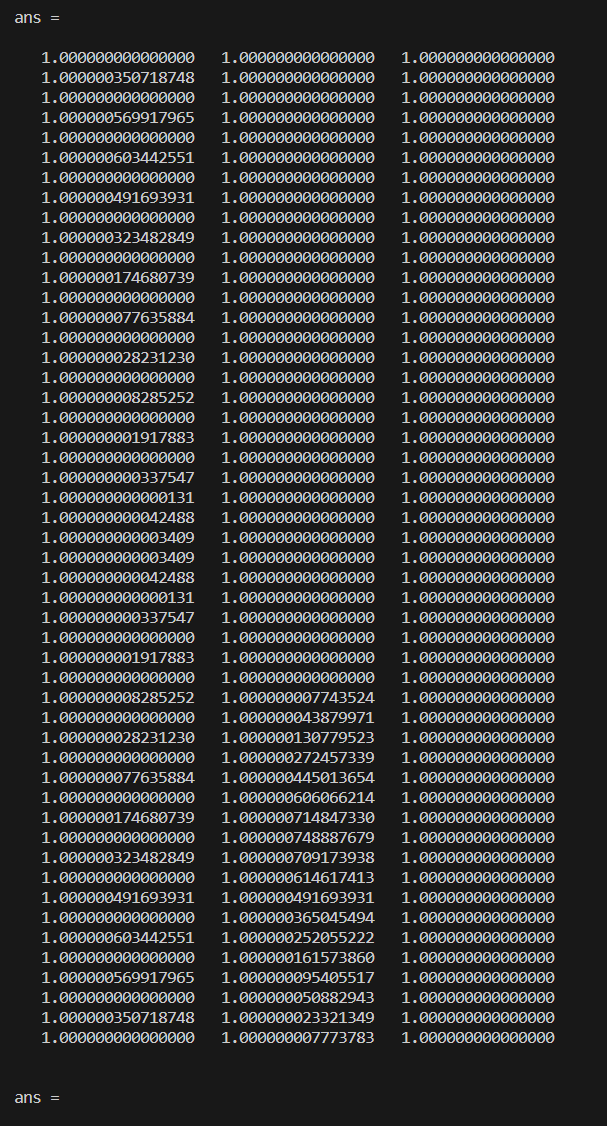
描述已自动生成

n=30时Jacobi迭代、Gauss迭代和解析解：

图片包含 表格

描述已自动生成

n=50时Jacobi迭代、Gauss迭代和解析解：



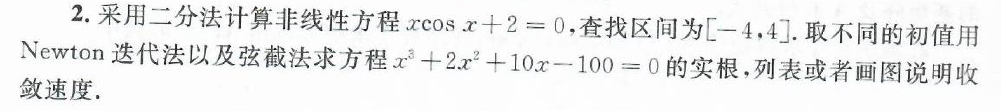
n=100时Jacobi迭代、Gauss迭代和解析解：

建筑与房屋的城市空拍图

中度可信度描述已自动生成

## 三、非线性方程求解

### 2题目



#### 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);

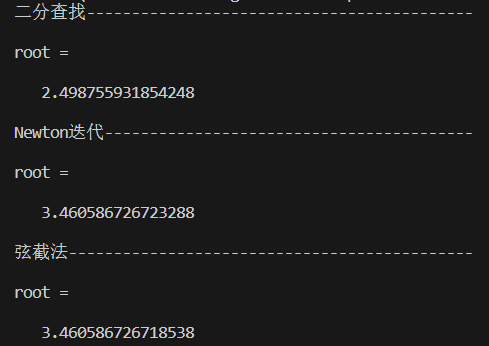
    end

    [Newton', Secant']

    disp('Newton迭代法收敛更快-------------------------------')

end

#### 结果

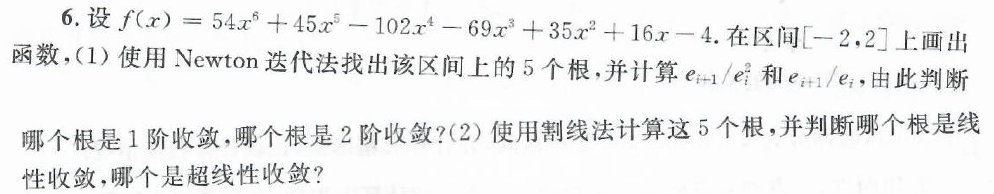


第一列为Newton迭代法的迭代根，第二列为Newton弦截法的迭代根：

文本

描述已自动生成

### 6题目



#### 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-----------------------')

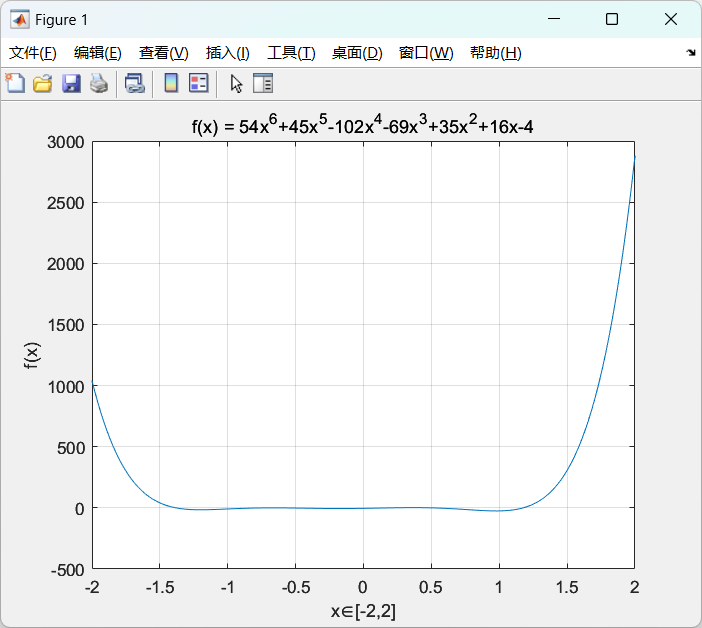
    disp(' 0.2附近的根，e1减小，e2增大，超线性收敛---------------------')

    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---------------------')

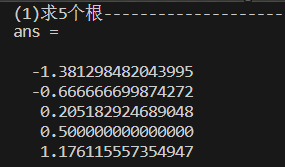
end

#### 图像

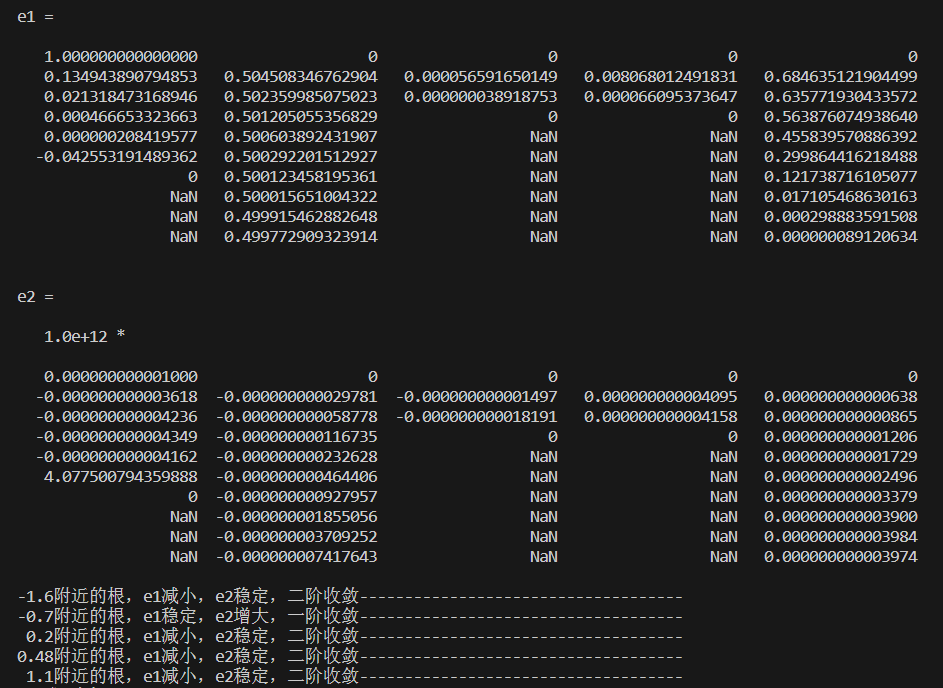


#### 结果

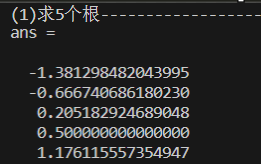
Newton迭代法求5个根：



判断收敛阶：



Newton弦截法求5个根：



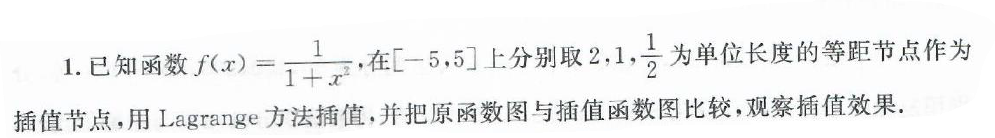
判断收敛阶：

图形用户界面, 文本

描述已自动生成

## 四、插值与逼近

### 1题目



#### Lagrange插值

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);

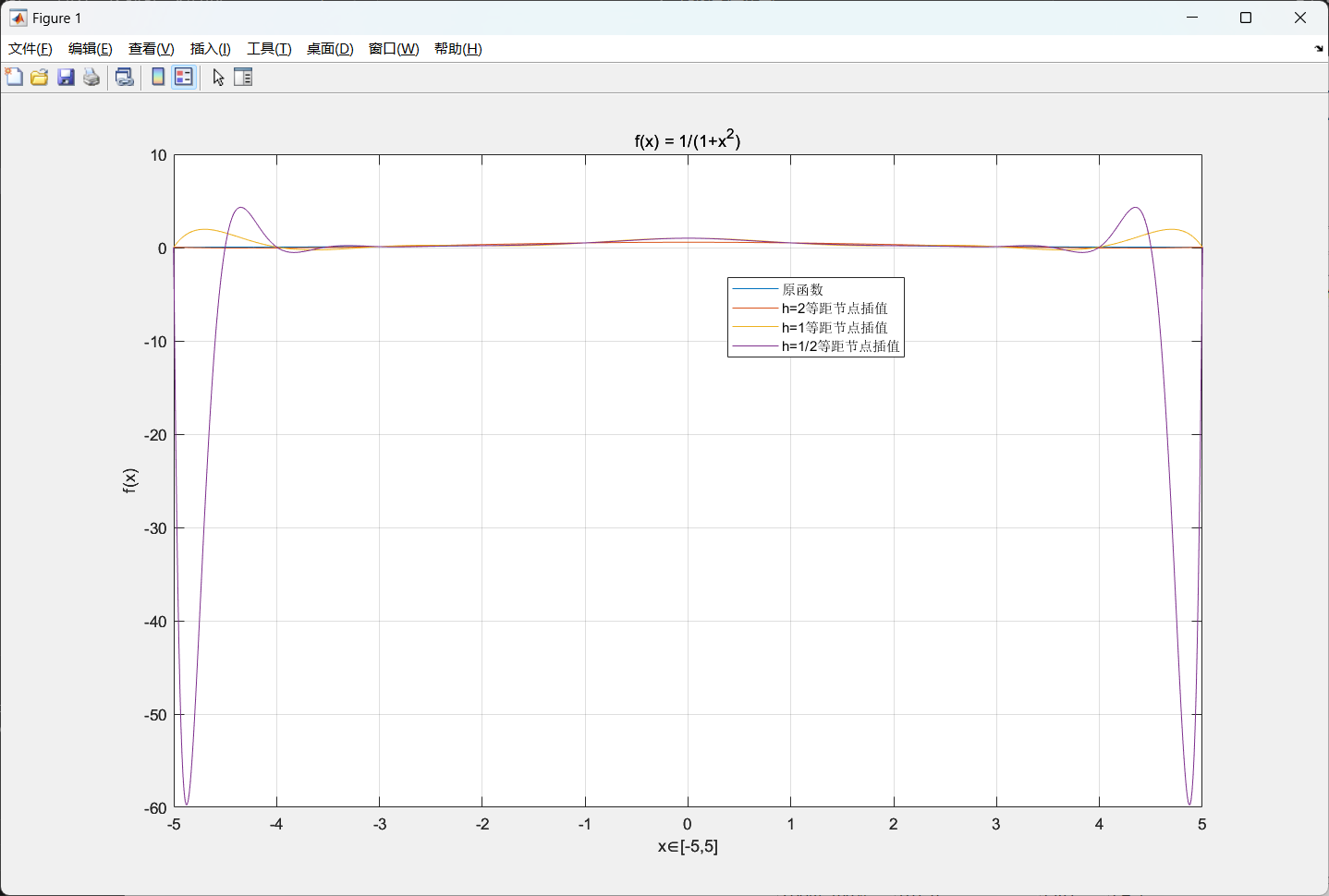
    plot(x\_value, fx\_lagrange\_value);

    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

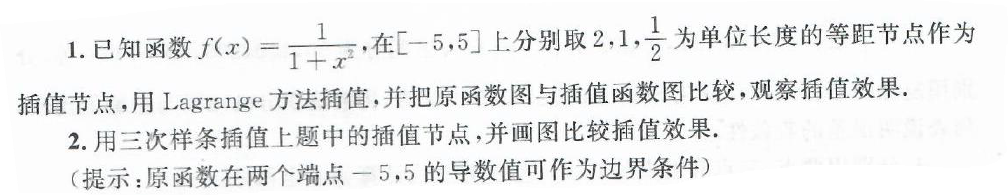
#### 函数图像



#### 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



#### 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);

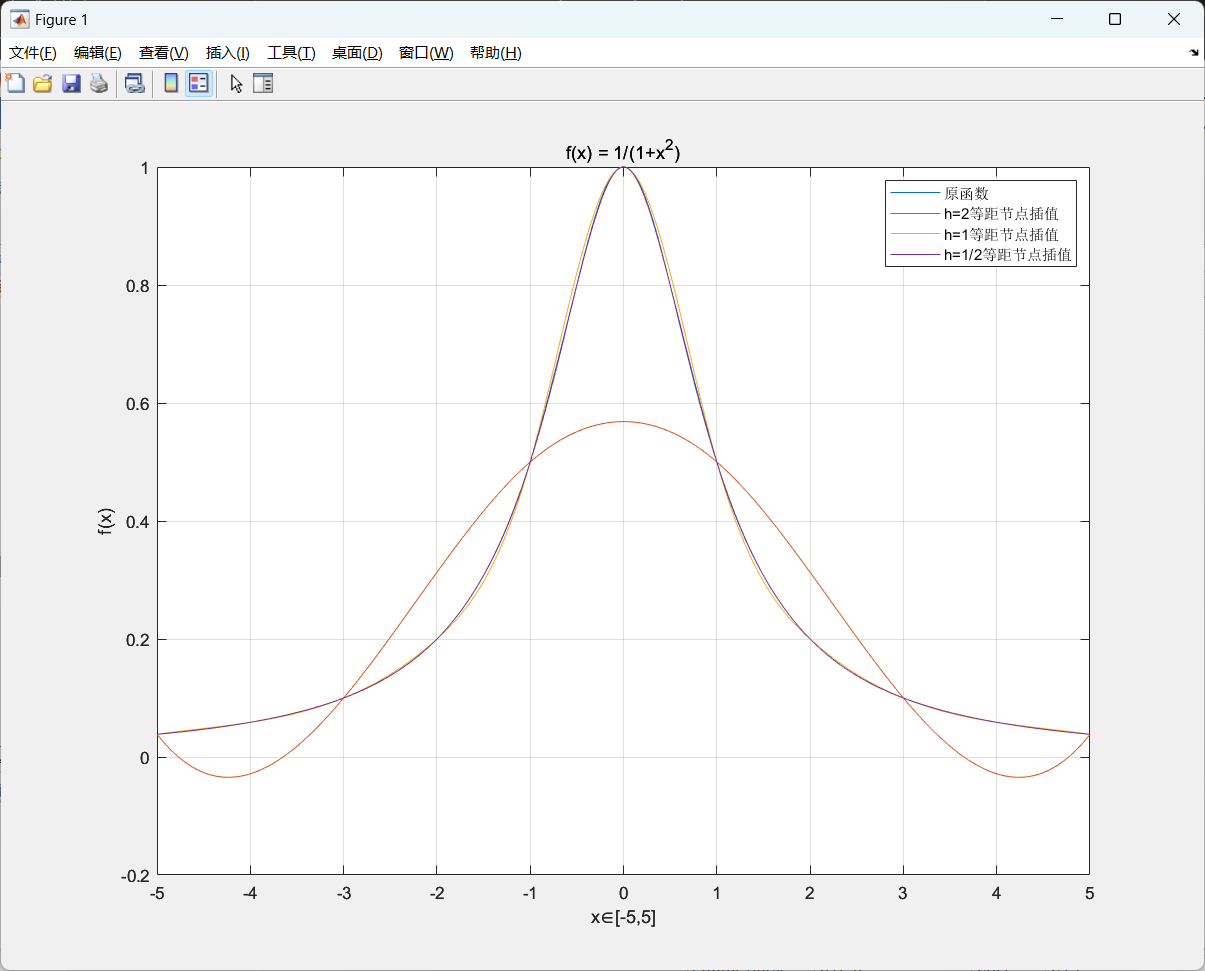
    plot(x\_value, fx\_spline\_value);

    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

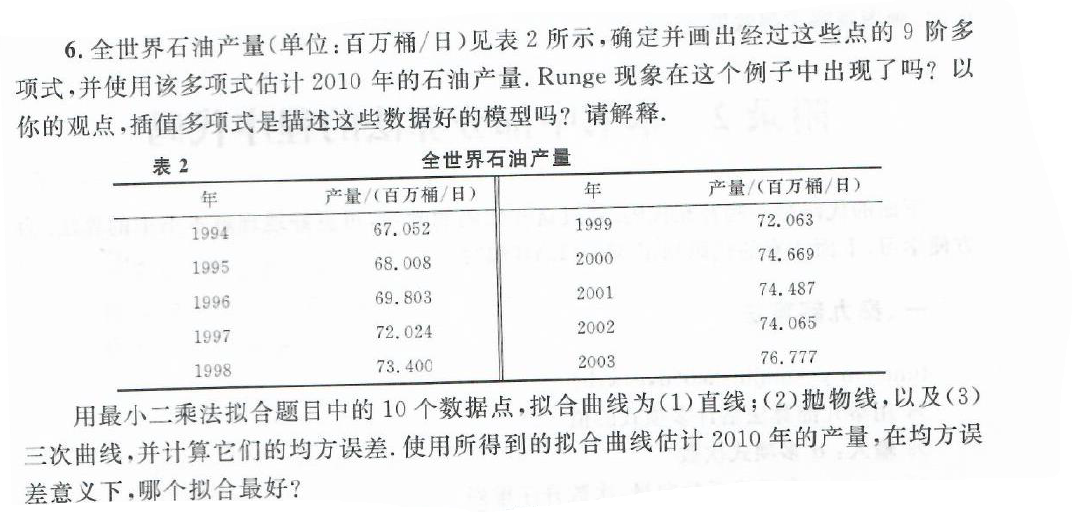
#### 函数图像



#### 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



#### 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)

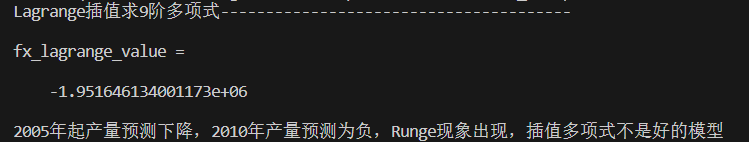
    p2\_mean = mean((fx\_value - p2\_value(1:length(fx\_value))) .^ 2)

    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

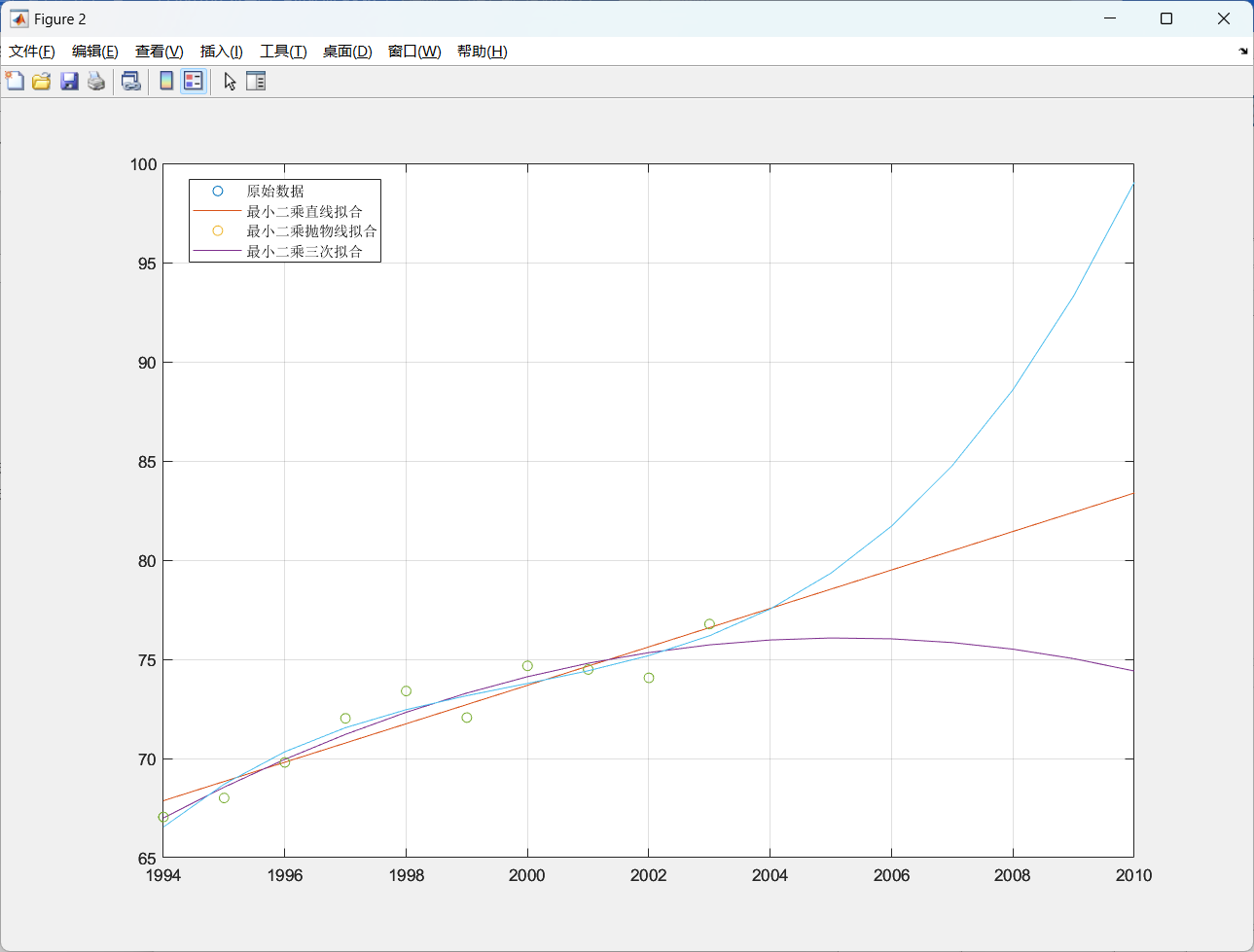


#### Lagrange插值9阶多项式图像

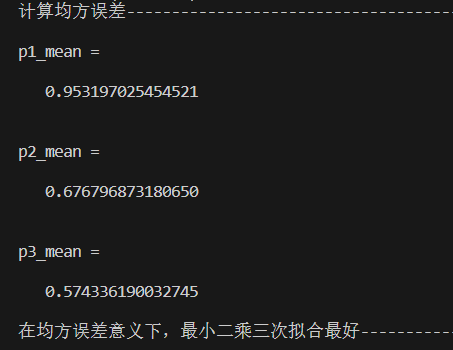
图形用户界面, 图表

描述已自动生成

#### 最小二乘拟合图像



#### 最小二乘均方误差



## 五、数值积分

### 3题目

文本

描述已自动生成

#### 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

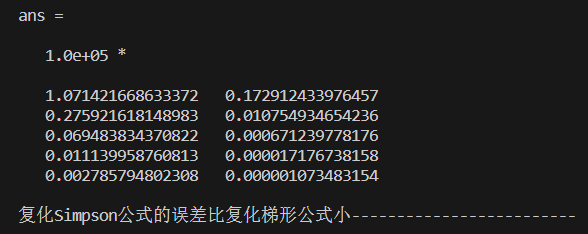
    [sum\_true' - sum\_T', sum\_true' - sum\_S']

    disp('复化Simpson公式的误差比复化梯形公式小-------------------------')

end

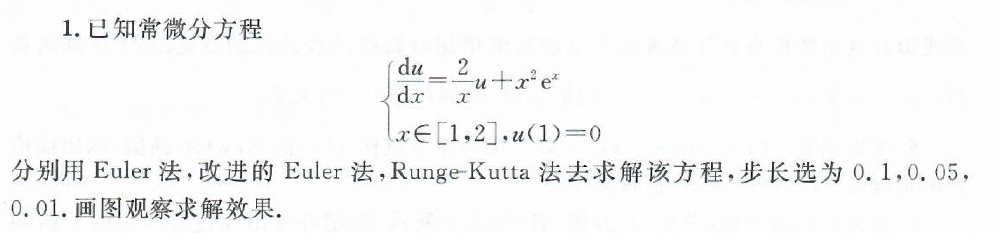
#### 结果

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：



## 六、微分方程数值解

### 1题目



#### 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：

图表, 折线图

描述已自动生成

步长为0.05时：

图表, 折线图

描述已自动生成

步长为0.01时：

