习题一

第一题

已知区间$[-1,1]$，Runge函数$f(x)=\dfrac{1}{1+25x^2}$，分别取$n=6$和$n=10$。

用区间n等分产生的等距节点对作Newton插值，要求对每个n，画出插值多项式和函数$f(x)$的曲线。

用$n+1$次Chebyshev多项式的零点为插值节点，作Newton插值，要求对每个n，画出插值多项式和函数$f(x)$的曲线。

解：差商公式为

$$f[x_0,\cdots,x_n]=\sum_{i=0}^{n}\frac{f(x_i)}{\prod\limits_{j\neq i}(x_i-x_j)} \tag{1}$$

Newton插值公式为

$$P_n(x)=f(x_0)+\sum_{i=1}^{n}f[x_0,\cdots,x_i]\prod_{j=0}^{i-1}(x-x_j) \tag{2}$$

Chebyshev多项式C_{n+1}的$n+1$个零点为

$$x_k=\cos\frac{2k+1}{2(n+1)}\pi \tag{3}$$

其中$k=0,\cdots,n$。

定义差商函数

```
function result = dividedDifference(fun, points)

    % 名称：差商
    % 输入：
    %       fun:     匿名函数
    %       points：需要求解差商的点
    % 输出：
    %       result：差商值

    %% 函数

    % 初始化结果
    result = 0;

    % 外层循环
    for i = 1: length(points)
        % 初始化积
        product = 1;
        % 内层循环
        for j = 1: length(points)
            if j ~= i
                product = product * (points(i) - points(j));
            end
```

```
        end
        result = result + fun(points(i)) / product;
    end

end

```

Newton插值函数

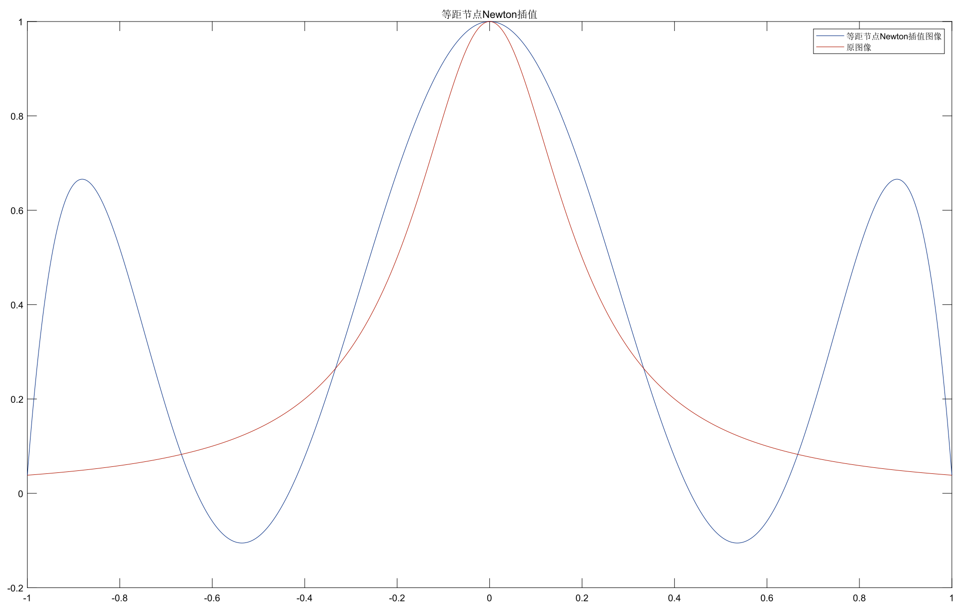
```
function NewtonInterpolationFormula(fun, a, b, points)

    % 名称:          Newton插值公式
    % 输入:
    %       fun:        匿名函数
    %       a:          插值左端点
    %       b:          插值右端点
    %       points:     插值节点
    % 输出:          插值图像

    %% 函数

    % 横坐标
    x = linspace(a, b, 1000);

    % 初始化纵坐标
    y = fun(a);
    % 求和
    for i = 1: length(points) - 1
        % 求解差商
        dividedDif = dividedDifference(fun, points(1: i + 1));
        % 初始化积
        prod = 1;
        % 求积
        for j = 0: i-1
            prod = prod .* (x - points(j + 1));
        end
        y = y + dividedDif .* prod;
    end

    % 绘图
    figure
    plot(x, y, x, fun(x))

end

```

主函数

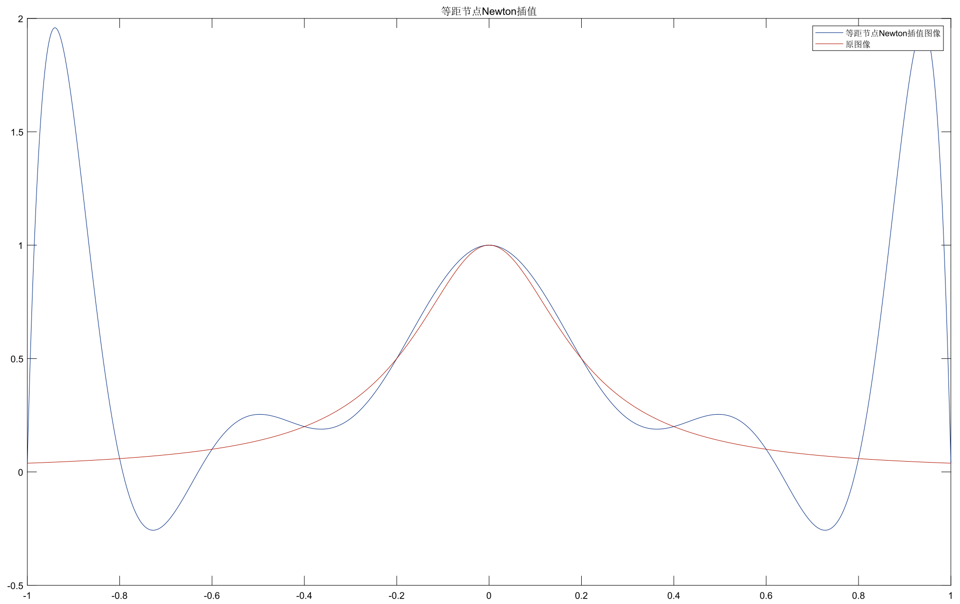
```
clear; clc

% 定义函数
fun = @(x) 1 ./ (1 + 25 .* x .^ 2);
a = -1;
b = 1;
```

```matlab

% n=6时等距节点Newton插值
n = 6;
points = 0: n;
points = a + (b - a) / n .* points;
NewtonInterpolationFormula(fun, a, b, points)
title('等距节点Newton插值')
legend('等距节点Newton插值图像','原图像')

% n=10时等距节点Newton插值
n = 10;
points = 0: n;
points = a + (b - a) / n .* points;
NewtonInterpolationFormula(fun, a, b, points)
title('等距节点Newton插值')
legend('等距节点Newton插值图像','原图像')

% n=6时Chebyshev节点Newton插值
n = 6;
points = 0: n;
points = cos((2 .* points + 1) ./ (2 * (n + 1)) .* pi);
NewtonInterpolationFormula(fun, a, b, points)
title('Chebyshev节点Newton插值')
legend('Chebyshev节点Newton插值图像','原图像')

% n=10时Chebyshev节点Newton插值
n = 10;
points = 0: n;
points = cos((2 .* points + 1) ./ (2 * (n + 1)) .* pi);
NewtonInterpolationFormula(fun, a, b, points)
title('Chebyshev节点Newton插值')
legend('Chebyshev节点Newton插值图像','原图像')

```

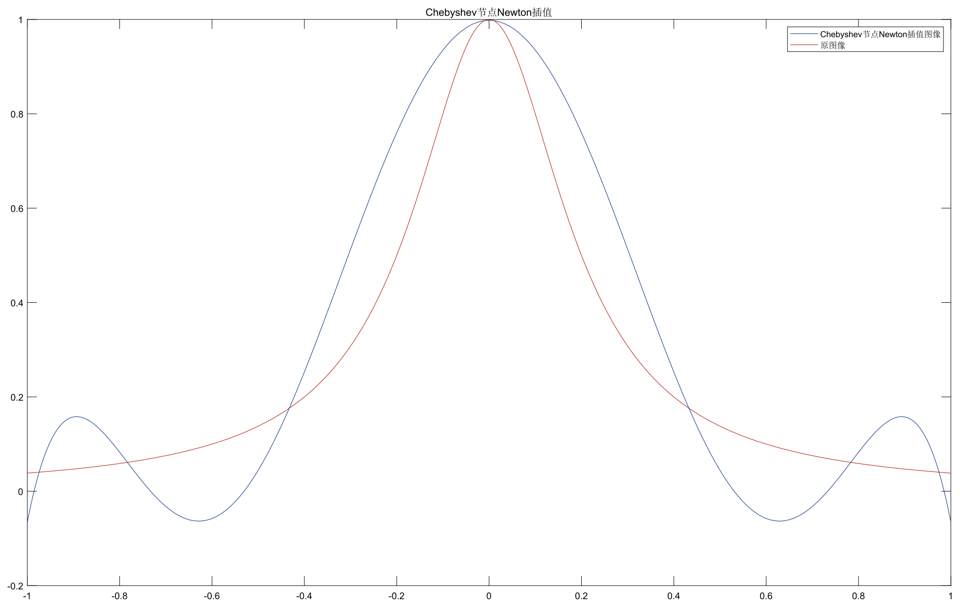
$n=6$时等距节点Newton插值输出图像



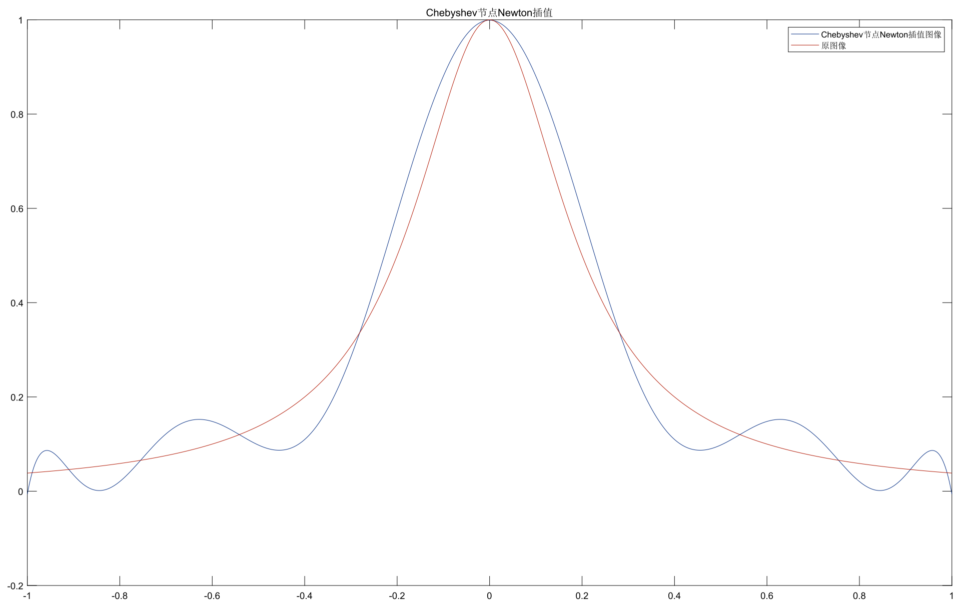
$n = 10$时等距节点Newton插值输出图像



$n = 6$时Chebyshev节点Newton插值输出图像



$n = 10$时等Chebyshev节点Newton插值输出图像



第二题

下列数据点的插值

x_k	0.001	1	8	27	64	125	216
$f(x_k)$	0.1	1	2	3	4	5	6

可以得到立方根函数$f(x)=\sqrt[3]{x}$的近似函数，要求用上述7个点作6次插值多项式$L_6(x)$，画出的曲线$L_6(x)$，并计算$\sqrt[3]{100}$的近似值。

解：定义多项式插值函数

```
function fun = polynomialInterpolationFormula(x0, y0)

    % 名称:            多项式插值公式
    % 输入:
    %       x0:         插值点横坐标
    %       y0:         插值点纵坐标
    % 输出:            多项式插值公式

    %% 函数

    N = length(x0);

    % 初始化系数矩阵
    A = ones(N, N);
    for n = 2: N
        A(n, :) = x0 .^ (n-1);
    end
    A = A';

    % 求解系数
    coefficient = A \ y0';

    % 输出多项式插值函数
```

```
    syms x
    fun = coefficient(1);
    for n = 2: N
        fun = fun + coefficient(n) .* x .^ (n - 1);
    end
    fun = matlabFunction(fun);

end

```

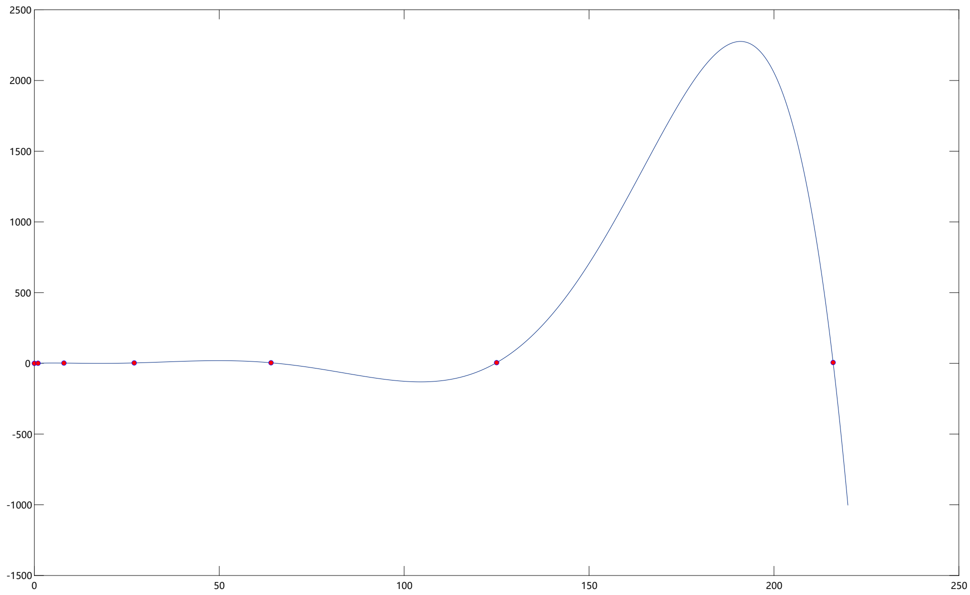
定义主函数

```
clear; clc

% 定义函数
fun = @(x) x .^ (1 / 3);

% 定义插值点
x0 = [0.001, 1, 8, 27, 64, 125, 216];
y0 = fun(x0);

% 求解插值公式
fun = polynomialInterpolationFormula(x0, y0);

% 求解插值
x = linspace(0, 220, 1000);
y = fun(x);

% 求解近似值
fprintf('100^(1/3)的近似值为：%.3f', fun(100 ^ (1 / 3)))

% 绘图
figure
plot(x, y)
hold on
plot(x0, y0, 'bo', 'MarkerSize', 5, 'MarkerFaceColor', 'r')
hold off

```

输出结果

```
100^(1/3)的近似值为：2.373
```

输出图像



第三题

已知函数在下列各点的值为

x_i	0.2	0.4	0.6	0.8	1.0
y_i	0.98	0.92	0.81	0.64	0.39

要求给出在自然边界条件下的三次样条插值多项式$S(x)$的表达式，并由插值多项式分别计算节点$x_k^* = 0.2 + 0.08k$的近似值，其中$k = 1, 3, 7, 9, 11$。

解：MatLab代码如下

```
clear;clc
% 已知数据点
x = [0.2, 0.4, 0.6, 0.8, 1.0];
y = [0.98, 0.92, 0.81, 0.64, 0.39];

% 计算三次样条插值多项式的系数
cubicSplineInterpolation = spline(x, [0, y, 0]);

% 显示插值多项式的系数
disp(round(cubicSplineInterpolation.coefs, 2))

```

输出结果

```
  2.61        -2.02          0         0.98
  0.92        -0.46       -0.5         0.92
- 7.52         0.09       -0.57        0.81
 26.67        -4.42       -1.43        0.64
```

因此三次样条插值函数为

$$S(x)=\begin{cases}2.61-2.02(x-0.2)+0.98(x-0.2)^3, & x<0.4\\ 0.92-0.46(x-0.4)-0.5(x-0.4)^2+0.92(x-0.4)^3, & 0.4\le x<0.6\\ -7.52+0.09(x-0.6)-0.57(x-0.6)^2+0.81(x-0.6)^3, & 0.6\le x<0.8\\ 26.67-4.42(x-0.8)-1.43(x-0.8)+0.64(x-0.8)^3, & x\ge 0.8\end{cases} \tag{4}$$

代入数据

$$S(x_1^*)=2.45,\quad S(x_3^*)=0.90,\quad S(x_7^*)=-7.52,\quad S(x_9^*)=25.97,\quad S(x_{11}^*)=25.05 \tag{5}$$

第四题

下列数据点

x_i	0	1	2	3	4	5	6.2832
y_i	1.0000	0.5403	−0.4161	−0.9900	−0.6536	0.2837	1.0000

是根据$y=\cos x$给出的，要求用上述数据在周期边界条件下作三次样条插值，并计算$x=1.5$和$x=1.8$时的近似值。

解：MatLab代码如下

```
clear; clc
% 给定数据点
x0 = [0, 1, 2, 3, 4, 5, 6.2832];
y0 = [1.0000, 0.5403, -0.4161, -0.9900, -0.6536, 0.2837, 1.0000];

% 为了满足周期边界条件，将第一个点和最后一个点连接起来
x0 = [x0, x0(1) + 2*pi];
y0 = [y0, y0(1)];

% 进行三次样条插值，使用周期边界条件
cubicSplineInterpolation = csape(x0, y0, 'periodic');

% 计算在x=1.5和x=1.8时的近似值
x = [1.5, 1.8];
y = ppval(cubicSplineInterpolation, x);

% 输出结果
fprintf('cos1.5为：%.3f\n', y(1))
fprintf('cos1.8为：%.3f', y(2))

```

输出结果

```
cos1.5为：0.071

cos1.8为：-0.228
```

习题二

第一题

已知函数$y = f(x)$在下列各点的值为

x_i	-1	-0.75	-0.5	0	0.25	0.5	0.75
y_i	1.00	0.8125	0.75	1.00	1.3125	1.75	2.3125

根据最小二乘法，分别用一次、二次、三次多项式拟合上述数据，画出所给数据点和最小二乘拟合多项式的图像。

解：代码如下

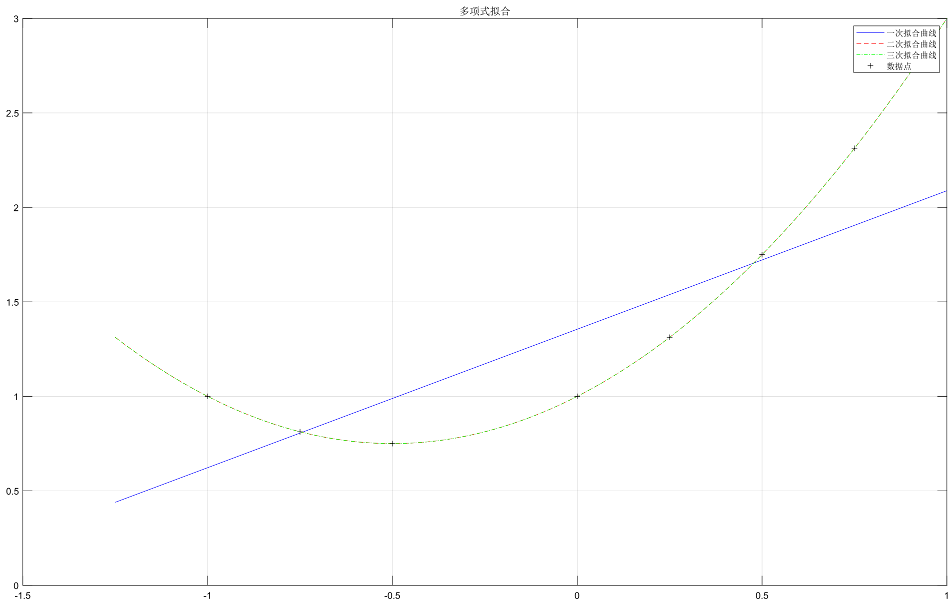
```
clear; clc

%% 准备数据

% 输入原始数据
x0 = [-1, -0.75, -0.5, 0, 0.25, 0.5, 0.75];
y0 = [1.00, 0.8125, 0.75, 1.00, 1.3125, 1.75, 2.3125];

%% 计算最小二乘拟合多项式系数

% 利用polyfit函数，分别用一、二、三次多项式对数据点进行最小二乘拟合
p1 = polyfit(x0, y0, 1); % 一次多项式的系数向量
p2= polyfit(x0, y0, 2); % 二次多项式的系数向量
p3 = polyfit(x0, y0, 3); % 三次多项式的系数向量

% 输出拟合多项式的系数
disp('一次多项式的系数向量为：')
disp(p1)
disp('二次多项式的系数向量为：')
disp(p2)
disp('三次多项式的系数向量为：')
disp(p3)

%%  绘图

% 计算拟合曲线的值
x = linspace(-1.25, 1, 1000);
y1 = polyval(p1, x);
y2 = polyval(p2, x);
y3 = polyval(p3, x);

% 绘制图形
figure
plot(x, y1, 'b-') % 一次拟合曲线，蓝色实线
hold on
plot(x, y2, 'r--') % 二次拟合曲线，红色虚线
plot(x, y3, 'g-.') % 三次拟合曲线，绿色点划线
plot(x0, y0, 'k+') % 原始数据点，黑色加号
hold off
```

```
% 添加图例，标题和网格线
legend('一次拟合曲线', '二次拟合曲线', '三次拟合曲线', '原始数据点')
title('多项式拟合')
grid on

%% 计算误差

% 计算一次拟合曲线的均方误差、最大绝对误差、平均绝对误差
mse1 = mean((y0 - polyval(p1, x0)).^2); % 均方误差
mae1 = max(abs(y0 - polyval(p1, x0))); % 最大绝对误差
mape1 = mean(abs(y0 - polyval(p1, x0))); % 平均绝对误差

% 计算二次拟合曲线的均方误差、最大绝对误差、平均绝对误差
mse2 = mean((y0 - polyval(p2, x0)).^2); % 均方误差
mae2 = max(abs(y0 - polyval(p2, x0))); % 最大绝对误差
mape2 = mean(abs(y0 - polyval(p2, x0))); % 平均绝对误差

% 计算三次拟合曲线的均方误差、最大绝对误差、平均绝对误差
mse3 = mean((y0 - polyval(p3, x0)).^2); % 均方误差
mae3 = max(abs(y0 - polyval(p3, x0))); % 最大绝对误差
mape3 = mean(abs(y0 - polyval(p3, x0))); % 平均绝对误差

% 输出结果
disp('一次拟合曲线的均方误差、最大绝对误差、平均绝对误差为：')
disp([mse1 mae1 mape1])
disp('二次拟合曲线的均方误差、最大绝对误差、平均绝对误差为：')
disp([mse2 mae2 mape2])
disp('三次拟合曲线的均方误差、最大绝对误差、平均绝对误差为：')
disp([mse3 mae3 mape3])
```

输出结果如下

```
一次多项式的系数向量为：
         0.732876712328767          1.35530821917808

二次多项式的系数向量为：
                         1                         1                         1

三次多项式的系数向量为：
     -9.74992473134766e-17                         1                         1                         1

一次拟合曲线的均方误差、最大绝对误差、平均绝对误差为：
        0.0776969178082192         0.407534246575342         0.234344422700587

二次拟合曲线的均方误差、最大绝对误差、平均绝对误差为：
      2.46519032881566e-32      3.33066907387547e-16      9.51619735392991e-17

三次拟合曲线的均方误差、最大绝对误差、平均绝对误差为：
       1.5847652113815e-32      2.22044604925031e-16      7.93016446160826e-17
```

输出图像如下



第二题

已知一组实验数据如下

x_k	0.0	0.2	0.5	0.7	0.85	1.0
y_k	1.000	1.221	1.649	2.014	2.340	2.718
w_k	0.1	0.2	0.3	0.1	0.2	0.1

求二次最小二乘拟合多项式，并计算均方误差。

解：定义加权最小二乘函数

```
function c = weightedLeastSquaresFit(x, y, w, n)

    % 名称:    加权最小二乘拟合
    % 输入:
    %       x: 拟合点横坐标
    %       y: 拟合点纵坐标
    %       w: 拟合权重
    %       n: 拟合多项式次数
    % 输出:
    %       c: 拟合多项式系数

    %% 函数

    % 计算系数矩阵
    A = zeros(n + 1, n + 1);
    b = zeros(n + 1, 1);
    for i = 1: n + 1
        b(i) = sum(w .* y .* x .^ (i - 1));
        for j = 1: n + 1
            A(i, j) = sum(w .* x .^ (i + j - 2));
        end
```

```
    end
    % 求解多项式系数
    c = A \ b;

end

```

主函数

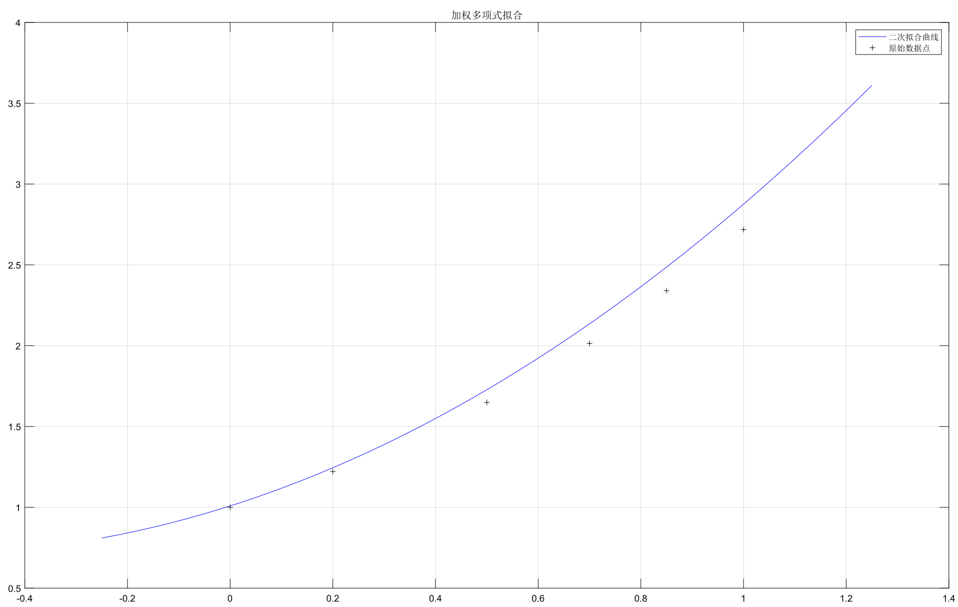
```
clear; clc

%% 准备数据

% 输入原始数据
x0 = [0.0, 0.2, 0.5, 0.7, 0.85, 1.0];
y0 = [1.000, 1.221, 1.649, 2.014, 2.340, 2.718];
w = [0.1, 0.2, 0.3, 0.1, 0.2, 0.1];

%%  计算加权最小二乘拟合多项式系数
n = 2; % 拟合多项式次数
c = weightedLeastSquaresFit(x0, y0, w, n);
% 输出拟合多项式的系数
disp('二次多项式的系数向量为：')
disp(c)

%%  绘图

% 计算拟合曲线的值
x = linspace(-0.25, 1.25, 1000);
y = c(1) * ones(1, 1000);
for k = 1: n
    y = y + c(k) * x .^ k;
end

% 绘制图形
figure
plot(x, y, 'b-') % 二次拟合曲线，蓝色实线
hold on
plot(x0, y0, 'k+') % 原始数据点，黑色加号
hold off

% 添加图例，标题和网格线
legend('二次拟合曲线', '原始数据点')
title('加权多项式拟合')
grid on

%% 计算误差

% 计算二次拟合曲线的均方误差、最大绝对误差、平均绝对误差
m = size(x0, 2);
y = c(1) * ones(1, m);
for k = 1: n
    y = y + c(k) * x0 .^ k;
end
mse2 = mean((y0 - y.^2)); % 均方误差
```

```
mae2 = max(abs(y0 - y)); % 最大绝对误差
mape2 = mean(abs(y0 - y)); % 平均绝对误差

% 输出结果
disp('二次拟合曲线的均方误差、最大绝对误差、平均绝对误差为：')
disp([mse2 mae2 mape2])

```

输出结果如下

```
二次多项式的系数向量为：
          1.00876388907147
         0.858131088382626
         0.840708191462776

二次拟合曲线的均方误差、最大绝对误差、平均绝对误差为：
         -2.27010629254655         0.157658866525572
 0.0894231704314411
```

输出图像如下



第三题

设$f(x) = \sin \pi x$，利用Legendre多项式分别求次数为$2, 3, 4$的多项式$p(x)$，使得$\int_0^1 (f(x) - p(x))^2 \mathrm{d}x$达到最小，并画出$f(x)$和$p(x)$的曲线进行比较。

解：定义加权平方逼近多项式拟合函数

```
function c = weightedSquaresApproximatePolynomialFit(fun, rho, n, a, b)

    % 名称：   加权平方逼近多项式拟合
    % 输入：
    %       fun：拟合函数
    %       rho：拟合权重
    %       n：拟合多项式次数
```

```
    %       a: 拟合左边界
    %       b: 拟合右边界
    % 输出:
    %       c: 拟合多项式系数

    %% 函数

    % 计算系数矩阵
    A = zeros(n + 1, n + 1);
    B = zeros(n + 1, 1);
    for i = 1: n + 1
        B(i) = integral(@(x) rho(x) .* fun(x) .* x .^ (i - 1), a, b);
        for j = 1: n + 1
            A(i, j) = integral(@(x) rho(x) .* x .^ (i + j - 2), a, b);
        end
    end
    % 求解多项式系数
    c = A \ B;

end

```

主函数

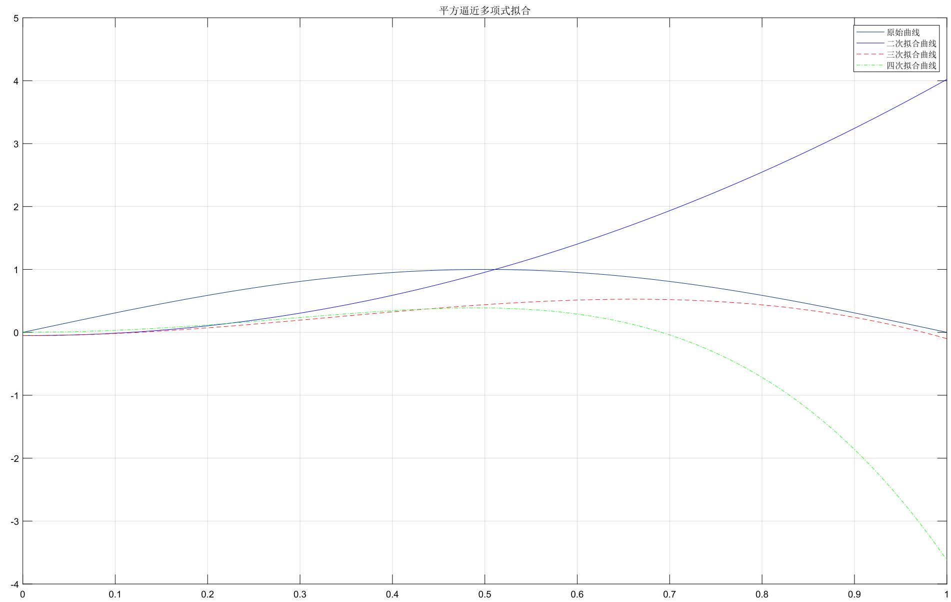
```
clear; clc

%% 准备数据

% 输入原始函数
fun = @(x) sin(pi * x);
rho = @(x) 1;
a = 0;
b = 1;

%% 计算拟合多项式系数
c2 = weightedSquaresApproximatePolynomialFit(fun, rho, 2, a, b);
c3 = weightedSquaresApproximatePolynomialFit(fun, rho, 3, a, b);
c4 = weightedSquaresApproximatePolynomialFit(fun, rho, 4, a, b);

% 输出拟合多项式的系数
disp('二次多项式的系数向量为: ')
disp(c2)
disp('三次多项式的系数向量为: ')
disp(c3)
disp('四次多项式的系数向量为: ')
disp(c4)

%%  绘图

% 计算拟合曲线的值
x = linspace(0, 1, 1000);

y2 = c2(1) * ones(1, 1000);
for k = 1: 2
    y2 = y2 + c2(k) * x .^ k;
```

```
end

y3 = c3(1) * ones(1, 1000);
for k = 1: 3
    y3 = y3 + c3(k) * x .^ k;
end

y4 = c4(1) * ones(1, 1000);
for k = 1: 4
    y4 = y4 + c4(k) * x .^ k;
end

% 绘制图形
figure
plot(x, fun(x))    % 原始曲线
hold on
plot(x, y2, 'b-')  % 二次拟合曲线，蓝色实线
plot(x, y3, 'r--') % 三次拟合曲线，红色虚线
plot(x, y4, 'g-.') % 四次拟合曲线，绿色点划线
hold off

% 添加图例，标题和网格线
legend('原始曲线', '二次拟合曲线', '三次拟合曲线', '四次拟合曲线')
title('平方逼近多项式拟合')
grid on

```

输出结果

```
二次多项式的系数向量为：
       -0.0504654977784496
          4.12251162087619
         -4.12251162087619

三次多项式的系数向量为：
       -0.0504654977784651
          4.12251162087637
         -4.12251162087665
      3.06005221162348e-13

四次多项式的系数向量为：
         0.001313455897898
          3.08693254734936
         0.537594209994186
         -7.24905351468689
          3.62452675734332
```

输出图像


平方逼近多项式拟合
原始曲线
二次拟合曲线
三次拟合曲线
四次拟合曲线
5
4
3
2
1
0
-1
-2
-3
-4
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

习题三

第一题

已知

$$\int_0^1 \frac{4}{1+x^2}\mathrm{d}x = \pi \tag{6}$$

因此可以通过数值积分来计算π的近似值。

第一问

分别用四点、六点Newton-Cotes公式计算近似值。

解：Cotes系数为

$$C_k^{(n)} = \frac{(-1)^{n-k}}{nk!(n-k)!}\int_0^n \prod_{\substack{0\le i\le n\\ i\ne k}}(x-i)\mathrm{d}x \tag{7}$$

定义Cotes系数函数

```
function result = CotesCoefficient(n, k)

    % 名称：Cotes系数
    % 输入：
    %       n
    %       k
    % 输出：
    %      result：Cotes系数C_k^n

    %% 函数
    syms x;

    result = (- 1) ^ (n - k) / (n * factorial(k) * factorial(n - k));

    % 定义被积函数
    integrand = 1;
    for i = 0: n
        if i ~= k
            integrand = integrand * (x - i);
        end
    end

    % 计算积分
    result = result * int(integrand, 0, n);

end

```

对于等距节点$x_k = a + \frac{b-a}{n}k$，Newton-Cotes公式为

$$\int_a^b f(x)\mathrm{d}x \approx (b-a)\sum_{k=0}^{n} C_k^{(n)} f(x_k) \tag{8}$$

定义Newton-Cotes公式函数

```
function result = NewtonCotesFormula(fun, n, a, b)

    % 名称：Newton-Cotes公式
    % 输入：
    %       fun:    积分函数
    %       n:      积分节点数
    %       a:      积分左边界
    %       b:      积分右边界
    % 输出：
    %      result：Newton-Cotes公式积分值

    %% 函数

    result = 0;
    for k = 0: n
        result = result + CotesCoefficient(n, k) * f(a + (b - a) * k / n);
    end
    result = (b - a) * result;

end
```

主函数

```
clear; clc

% 定义积分函数
fun = @(x) 4 ./ (1 + x .^ 2);

% 计算积分值
int4 = double(NewtonCotesFormula(fun, 3, 0, 1)); % 四点Newton-Cotes公式近似值
int6 = double(NewtonCotesFormula(fun, 5, 0, 1)); % 六点Newton-Cotes公式近似值

% 输出结果
fprintf('四点Newton-Cotes公式近似值为：%.4f\n', int4)
fprintf('六点Newton-Cotes公式近似值为：%.4f\n', int6)
```

输出结果

```
四点Newton-Cotes公式近似值为：3.1385
六点Newton-Cotes公式近似值为：3.1419
```

第二问

分别取$h=0.1$和$h=0.2$，利用复合梯形公式和复合Simpson公式计算π的近似值。

解：等距节点$x_k=a+\frac{b-a}{n}k$的复合梯形公式为

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2n}\left(f(a)+2\sum_{k=1}^{n-1} f(x_k)+f(b)\right) \tag{9}$$

等距节点$x_k=a+\frac{b-a}{n}k$的复合Simpson公式为

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{6n}\left(f(a)+2\sum_{k=1}^{n-1} f(x_k)+4\sum_{k=1}^{n} f\left(\frac{x_{k-1}+x_k}{2}\right)+f(b)\right) \tag{10}$$

定义复合梯形公式函数

```matlab
function result = compoundTrapezoidalFormula(fun, n, a, b)

    % 名称：复合梯形公式
    % 输入：
    %       fun:    积分函数
    %       n:      积分节点数
    %       a:      积分左边界
    %       b:      积分右边界
    % 输出：
    %      result：复合梯形公式积分值

    %% 函数

    result = fun(a) + fun(b);
    for k = 1: n-1
        result = result + 2 * fun(a + (b - a) * k / n);
    end
    result = (b - a) / (2 * n) * result;

end

```

定义复合Simpson公式函数

```matlab
function result = compoundSimpsonFormula(fun, n, a, b)

    % 名称：复合Simpson公式
    % 输入：
    %       fun:    积分函数
    %       n:      积分节点数
    %       a:      积分左边界
    %       b:      积分右边界
    % 输出：
    %      result：复合Simpson公式积分值

    %% 函数

    result = fun(a) + fun(b);
    for k = 1: n-1
```

```
        result = result + 2 * fun(a + (b - a) * k / n);
    end
    for k = 1: n
        result = result + 4 * fun(a + (b - a) * (k - 1 / 2) / n);
    end
    result = (b - a) / (6 * n) * result;

end

```

主函数

```
clear; clc

% 定义积分函数
fun = @(x) 4 ./ (1 + x .^ 2);

% 计算积分值
trapezoidal1 = compoundTrapezoidalFormula(fun, 10, 0, 1); % 间距为0.1的复合梯形公式近似值
trapezoidal2 = compoundTrapezoidalFormula(fun, 5, 0, 1); % 间距为0.2的复合梯形公式近似值
Simpson1 = compoundSimpsonFormula(fun, 10, 0, 1); % 间距为0.1的复合Simpson公式近似值
Simpson2 = compoundSimpsonFormula(fun, 5, 0, 1); % 间距为0.2的复合Simpson公式近似值

% 输出结果
fprintf('间距为0.1的复合梯形公式近似值为：%.5f\n', trapezoidal1)
fprintf('间距为0.2的复合梯形公式近似值为：%.5f\n', trapezoidal2)
fprintf('间距为0.1的复合Simpson公式近似值为：%.10f\n', Simpson1)
fprintf('间距为0.2的复合Simpson公式近似值为：%.10f\n', Simpson2)

```

输出结果

```
间距为0.1的复合梯形公式近似值为：3.13993
间距为0.2的复合梯形公式近似值为：3.13493
间距为0.1的复合Simpson公式近似值为：3.1415926530
间距为0.2的复合Simpson公式近似值为：3.1415926139
```

第三问

把区间$[0,1]$进行n等分，利用复合梯形公式和复合Simpson公式计算π的近似值。若要求误差不超过0.5×10^{-6}，问需要把区间$[0,1]$划分成多少等份。

解：复合梯形公式函数和复合Simpson公式函数见上。

主函数

```
clear; clc

% 定义积分函数
fun = @(x) 4 ./ (1 + x .^ 2);

```

```

trapezoidalNumber = 2;
while abs(compoundTrapezoidalFormula(fun, trapezoidalNumber, 0, 1) - pi) > 0.5 * 10 ^ (-6)
    trapezoidalNumber = trapezoidalNumber + 1;
end

SimpsonNumber = 2;
while abs(compoundSimpsonFormula(fun, SimpsonNumber, 0, 1) - pi) > 0.5 * 10 ^ (-6)
    SimpsonNumber = SimpsonNumber + 1;
end

% 输出结果
fprintf('复合梯形公式需要把区间[0,1]划分成等份%.0f等份\n', trapezoidalNumber)
fprintf('复合Simpson公式需要把区间[0,1]划分成等份%.0f等份\n', SimpsonNumber)

```

输出结果

```
复合梯形公式需要把区间[0,1]划分成等份578等份
复合Simpson公式需要把区间[0,1]划分成等份4等份
```

第四问

选择不同的h，对两种复合求积公式，试将误差描述为h的函数，输出函数表达式。

解：复合梯形公式的积分余项的绝对值为

$$R[f] = \frac{1}{12n^2} f''(\xi), \qquad \xi \in (0,1) \tag{11}$$

使用拟合求出$f''(\xi)$的拟合值

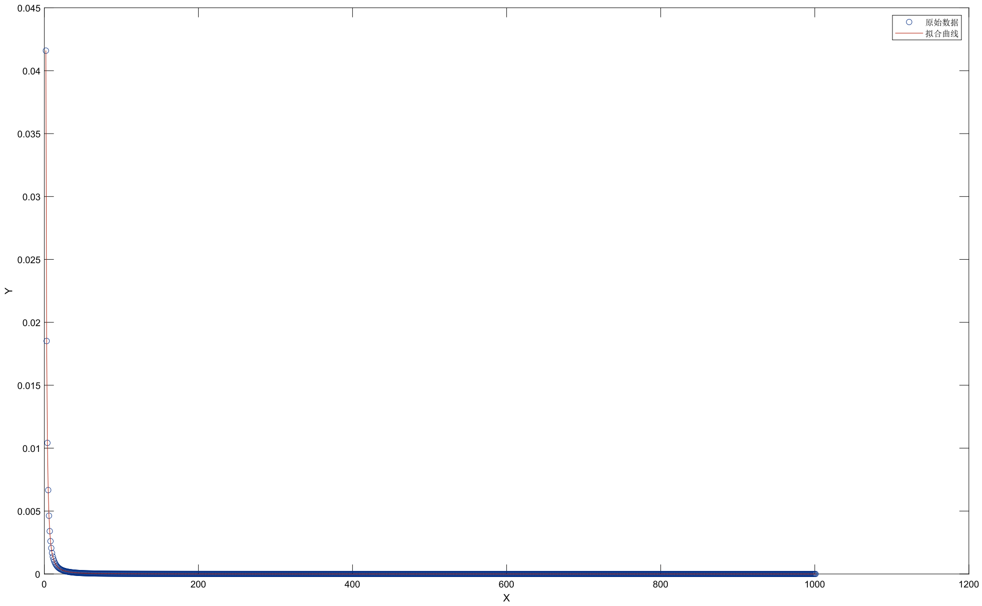
```
clear; clc

% 定义积分函数
fun = @(x) 4 ./ (1 + x .^ 2);

compoundTrapezoidalFormulaError = zeros(1, 1000);
for n = 2: 1001
    compoundTrapezoidalFormulaError(n - 1) = abs(compoundTrapezoidalFormula(fun, n, 0, 1) - pi);
end

X = 2: 1001;
Y = compoundTrapezoidalFormulaError;

% 定义函数模型
model = fittype(@(a, x) a./(12 * x.^2), 'independent', 'x', 'dependent', 'y');

% 初始参数猜测
initialGuess = 1;

```

```matlab
% 进行非线性拟合
fitResult = fit(X', Y', model, 'StartPoint', initialGuess);

% 获取拟合后的参数
a_fit = fitResult.a;

% 计算拟合后的Y
Y_fit = a_fit./(12 * X.^2);

% 计算R方
R_squared = 1 - sum((Y - Y_fit).^2) / sum((Y - mean(Y)).^2);

% 计算RMSE
RMSE = sqrt(sum((Y - Y_fit).^2) / n);

% 计算SSE
SSE = sum((Y - Y_fit).^2);

% 输出结果
fprintf('拟合值: %f\n', a_fit);
fprintf('R方: %f\n', R_squared);
fprintf('RMSE: %f\n', RMSE);
fprintf('SSE: %f\n', SSE);

% 绘制拟合曲线
figure;
plot(X, Y, 'o', X, Y_fit, '-')
legend('原始数据', '拟合曲线');
xlabel('X');
ylabel('Y');
```

输出结果

```
拟合值: 1.997252
R方: 0.999999
RMSE: 0.000001
SSE: 0.000000
```

输出图像



因此复合梯形公式的积分余项的绝对值为

$$R[f] = \frac{1}{6n^2} = \frac{h^2}{6} \tag{12}$$

复合Simpson公式的积分余项的绝对值为

$$R[f] = \frac{1}{2880n^4} f^{(4)}(\xi), \qquad \xi \in (0,1) \tag{13}$$

使用拟合求出$f^{(4)}(\xi)$的拟合值

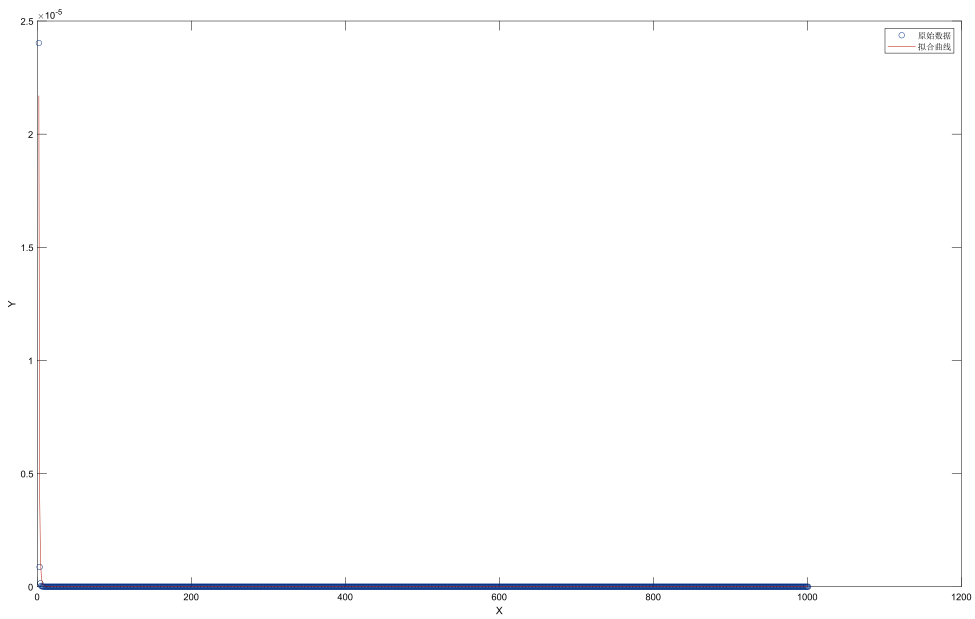
```
clear; clc

% 定义积分函数
fun = @(x) 4 ./ (1 + x .^ 2);

compoundSimpsonFormulaError = zeros(1, 1000);
for n = 2: 1001
    compoundSimpsonFormulaError(n - 1) = abs(compoundSimpsonFormula(fun, n, 0, 1) - pi);
end

X = 2: 1001;
Y = compoundSimpsonFormulaError;

% 定义函数模型
model = fittype(@(a, x) a./(2880 * x.^4), 'independent', 'x', 'dependent', 'y');

% 初始参数猜测
initialGuess = 1;

% 进行非线性拟合
fitResult = fit(X', Y', model, 'StartPoint', initialGuess);

% 获取拟合后的参数
```

```matlab
a_fit = fitResult.a;

% 计算拟合后的Y
Y_fit = a_fit./(2880 * X.^4);

% 计算R方
R_squared = 1 - sum((Y - Y_fit).^2) / sum((Y - mean(Y)).^2);

% 计算RMSE
RMSE = sqrt(sum((Y - Y_fit).^2) / n);

% 计算SSE
SSE = sum((Y - Y_fit).^2);

% 输出结果
fprintf('拟合值: %f\n', a_fit);
fprintf('R方: %f\n', R_squared);
fprintf('RMSE: %f\n', RMSE);
fprintf('SSE: %f\n', SSE);

% 绘制拟合曲线
figure;
plot(X, Y, 'o', X, Y_fit, '-')
legend('原始数据', '拟合曲线');
xlabel('X');
ylabel('Y');
```

输出结果

```
拟合值: 1.000000
R方: 0.967311
RMSE: 0.000000
SSE: 0.000000
```

输出图像



因此复合Simpson公式的积分余项的绝对值为

$$R[f]=\frac{1}{2280n^4}=\frac{h^4}{2280} \tag{14}$$

第二题

分别用三点和五点Gauss-Legendre公式计算积分

$$\int_0^1 \frac{xe^x}{(1+x)^2}\mathrm{d}x=\frac{\mathrm{e}}{2}-1\approx 0.3591409142295 \tag{15}$$

解：区间$[-1,1]$上关于权$\rho=1$的Gauss型求积公式为Gauss-Legendre求积公式

$$\int_{-1}^1 f(x)\mathrm{d}x\approx\sum_{k=1}^n A_k f(x_k) \tag{16}$$

其中求积节点$\{x_k\}_{k=1}^n$为n次Legendre多项式$L_n(x)$的零点，且

$$L_n(x)=\frac{1}{2^n n!}\frac{\mathrm{d}^n}{\mathrm{d}x^n}(x^2-1)^n \tag{17}$$

同时

$$\int_{-1}^1 x^m\mathrm{d}x=\sum_{k=1}^n A_k x_k^m,\qquad 0\le m\le 2n-1 \tag{18}$$

一般的

$$\int_a^b f(x)\mathrm{d}x=\frac{b-a}{2}\int_{-1}^1 f\left(\frac{b-a}{2}x+\frac{b+a}{2}\right)\mathrm{d}x \tag{19}$$

定义Gauss-Legendre求积公式函数

```
function result = GaussLegendreIntegralFormula(fun, n, a, b)

    % 名称：Gauss-Legendre求积公式
    % 输入：
    %       fun:    积分函数
    %       n:      积分节点数
    %       a:      积分左边界
    %       b:      积分右边界
    % 输出：
    %       result: 积分值

    %% 函数

    % 求解Legendre多项式的零点
    syms x
    L = diff((x^2-1)^n, x, n) / (2^n * factorial(n)); % Legendre多项式
    root = solve(L);                                  % Legendre多项式的根

    % 求解权重
    A = zeros(2 * n, n);
    B = zeros(2 * n, 1);
    for k = 0: 2 * n - 1
        A(k + 1, :) = transpose(root .^ k);
```

```
        B(k + 1) = int(x .^ k, -1, 1);
    end
    w = A \ B;

    % 求解积分值
    f = @(x) fun((b - a) / 2 .* x + (b + a) / 2);
    result = (b - a) / 2 * sum(w .* f(root));

end

```

主函数

```
clear; clc

% 定义函数
fun = @(x) x .* exp(x) ./ (1 + x) .^ 2;
% 计算积分值
int3 = GaussLegendreIntegralFormula(fun, 3, 0, 1);
int5 = GaussLegendreIntegralFormula(fun, 5, 0, 1);
% 输出结果
fprintf('三点Gauss-Legendre公式积分值为: %.10f\n', int3)
fprintf('五点Gauss-Legendre公式积分值为: %.10f\n', int5)

```

输出结果

```
三点Gauss-Legendre公式积分值为: 0.3591871703
五点Gauss-Legendre公式积分值为: 0.3591409792
```

第三题

分别用三点和四点Gauss-Lagurre公式计算积分

$$\int_0^{\infty} \mathrm{e}^{-10x} \sin x \mathrm{d}x = \frac{1}{101} \approx 0.00990099 \tag{20}$$

解:

$$\int_0^{\infty} \mathrm{e}^{-10x} \sin x \mathrm{d}x = \int_0^{\infty} \mathrm{e}^{-x} \frac{1}{10} \sin \frac{x}{10} \mathrm{d}x \tag{21}$$

区间$[0,\infty)$上关于权$\rho = \mathrm{e}^{-x}$的Gauss型求积公式为Gauss-Laguerre求积公式

$$\int_0^{\infty} \mathrm{e}^{-x} f(x) \mathrm{d}x \approx \sum_{k=1}^{n} A_k f(x_k) \tag{22}$$

其中求积节点$\{x_k\}_{k=1}^n$为n次Laguerre多项式$L_n(x)$的零点，且

$$L_n(x) = \mathrm{e}^x \frac{\mathrm{d}}{\mathrm{d}x^n} x^n \mathrm{e}^{-x} \tag{23}$$

同时

$$A_k = \frac{((n+1)!)^2}{x_k (L'_{n+1}(x_k))^2} \tag{24}$$

定义Gauss-Laguerre求积公式函数

```
function result = GaussLaguerreIntegralFormula(fun, n)

    % 名称：Gauss-Laguerre求积公式
    % 输入：
    %       fun:    积分函数
    %       n:      积分节点数
    % 输出：
    %       result: 积分值

    %% 函数
    syms x
    L = exp(x) * diff(x^n * exp(-x), x, n); % Laguerre多项式
    root = solve(L);                          % Laguerre多项式的根
    DL = matlabFunction(diff(L, x));
    result = 0;
    for k = 1: n
        result = result + (factorial(n))^2 / root(k) / (DL(root(k)))^2 * fun(root(k));
    end

end

```

主函数

```
clear; clc

% 定义函数
fun = @(x) sin(x / 10) / 10;
% 计算积分值
int3 = GaussLaguerreIntegralFormula(fun, 3);
int4 = GaussLaguerreIntegralFormula(fun, 4);
% 输出结果
fprintf('三点Gauss-Lagurre公式积分值为：%.10f\n', int3)
fprintf('四点Gauss-Lagurre公式积分值为：%.10f\n', int4)

```

输出结果

```
三点Gauss-Lagurre公式积分值为：0.0099009918
四点Gauss-Lagurre公式积分值为：0.0099009901
```

第四题

设$f(x)=\ln x$，分别取$h=10^{-n}$，其中$n=1,2,3,4$，用以下三个公式计算$f'(0.7)$的近似值。

$$f'(x)=\frac{f(x+h)-f(x)}{h} \tag{25}$$

$$f'(x)=\frac{f(x+h)-f(x-h)}{2h} \tag{26}$$

$$f'(x)=\frac{f(x-2h)-8f(x-h)+8f(x+h)-f(x+2h)}{12h} \tag{27}$$

列表比较三个公式的计算误差，从误差可以得出什么结论?

解：主函数

```
clear; clc

% 定义函数 f(x) = ln(x)
f = @(x) log(x);

% 待计算的点
x = 0.7;

% 求导数的准确值
exactDerivative = 1 / x;

% 不同的步长
H = transpose(10 .^ (-1: -1: -4));

% 初始化误差矩阵
errors = zeros(numel(H), 3);

% 计算误差
for k = 1: numel(H)
    h = H(k);

    % 使用第一个公式计算近似值
    derivative1 = (f(x + h) - f(x)) / h;
    errors(k, 1) = abs(exactDerivative - derivative1);

    % 使用第二个公式计算近似值
    derivative2 = (f(x + h) - f(x - h)) / (2 * h);
    errors(k, 2) = abs(exactDerivative - derivative2);

    % 使用第三个公式计算近似值
    derivative3 = (f(x - 2 * h) - 8 * f(x - h) + 8 * f(x + h) - f(x + 2 * h)) / (12 * h);
    errors(k, 3) = abs(exactDerivative - derivative3);
end

% 创建表格
variable_names = {'步长', '公式1', '公式2', '公式3'};
T = table(H, errors(:, 1), errors(:, 2), errors(:, 3), 'VariableNames', variable_names);
% 显示表格
format short e
disp(T);

```

输出结果

```
        步长          公式1          公式2          公式3
    ___________   ___________   ___________   ___________

    1.0000e-01    9.3258e-02    9.8389e-03    5.1317e-04
    1.0000e-02    1.0108e-02    9.7194e-05    4.7634e-08
    1.0000e-03    1.0194e-03    9.7182e-07    4.7569e-12
    1.0000e-04    1.0203e-04    9.7180e-09    3.1619e-13
```

横向比较：同一步长，公式1误差>公式2误差>公式3误差。

纵向比较：同一公式，步长越小误差越小。

第五题

对于积分

$$\int_0^1 \frac{4}{1+x^2}\mathrm{d}x = \pi \tag{28}$$

，取$h=0.1$和$h=0.2$，分别用复合两点Gauss-Legendre公式和复合三点Gauss-Legendre公式计算π的近似值。

解：定义Gauss-Legendre求积公式函数

```
function result = GaussLegendreIntegralFormula(fun, n, a, b)

    % 名称：Gauss-Legendre求积公式
    % 输入：
    %       fun:     积分函数
    %       n:       积分节点数
    %       a:       积分左边界
    %       b:       积分右边界
    % 输出：
    %       result：积分值

    %% 函数

    % 求解Legendre多项式的零点
    syms x
    L = diff((x^2-1)^n, x, n) / (2^n * factorial(n)); % Legendre多项式
    root = solve(L);                                  % Legendre多项式的根

    % 求解权重
    A = zeros(2 * n, n);
    B = zeros(2 * n, 1);
    for k = 0: 2 * n - 1
        A(k + 1, :) = transpose(root .^ k);
        B(k + 1) = int(x .^ k, -1, 1);
    end
    w = A \ B;

    % 求解积分值
    f = @(x) fun((b - a) / 2 .* x + (b + a) / 2);
    result = (b - a) / 2 * sum(w .* f(root));

```

```
end

```

定义复合Gauss-Legendre求积公式函数

```
function result = CompoundGaussLegendreIntegralFormula(fun, n, k, a, b)

    % 名称：复合Gauss-Legendre求积公式
    % 输入：
    %       fun:     积分函数
    %       n:       积分区间数
    %       k:       区间积分节点数
    %       a:       积分左边界
    %       b:       积分右边界
    % 输出：
    %       result: 积分值

    %% 函数
    result = 0;
    x = @(i) a + (b - a) / n * i;
    for i = 1: n
        result = result + GaussLegendreIntegralFormula(fun, k, x(i - 1), x(i));
    end

end

```

主函数

```
clear; clc

% 定义函数
fun = @(x) 4 ./ (1 + x .^ 2);

% 计算积分值
a = 0;
b = 1;
h = [0.1; 0.2];
int12 = CompoundGaussLegendreIntegralFormula(fun, (b - a) / h(1), 2, a, b);
int13 = CompoundGaussLegendreIntegralFormula(fun, (b - a) / h(1), 3, a, b);
int22 = CompoundGaussLegendreIntegralFormula(fun, (b - a) / h(2), 2, a, b);
int23 = CompoundGaussLegendreIntegralFormula(fun, (b - a) / h(2), 3, a, b);
int = [int12, int13; int22, int23];

% 创建表格
variable_names = {'步长', '两点', '三点'};
precision = 15;  % 设置精度
T = table(vpa(h, 2), vpa(int(:, 1), precision), vpa(int(:, 2), precision), 'VariableNames', variable_names);
% 显示表格
disp(T);

```

输出结果

```
步长          两点                  三点
_____   ________________   ________________

0.1     3.14159265403069   3.14159265356003
0.2     3.14159268178543   3.14159265168714
```

习题四

第一题

对于如下方程组

$$\begin{pmatrix} 1 & -1 & 2 & 1 \\ -1 & 3 & 0 & -3 \\ 2 & 0 & 9 & -6 \\ 1 & -3 & -6 & 19 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 5 \\ 7 \end{pmatrix} \tag{29}$$

判断用Jacobi迭代、Gauss-Seidel 迭代、SOR迭代（分别取$\omega = 0.8, 1.2, 1.3, 1.6$）解上述方程组的收敛性。

若收敛，再用Jacobi迭代、Gauss-Seidel迭代、SOR迭代（分别取$\omega = 0.8, 1.2, 1.3, 1.6$）分别解上述方程组，若迭代终止条件为$\left\|b - Ax^{(n)}\right\|_2 \leq 10^{-6}$，写出数值解。

比较上述各种迭代方法的收敛速度。

解：首先进行DLU分解，将$A = \{a_{ij}\}_{n\times n} \in \mathbb{R}^{n\times n}$分裂为$D - L - U$：

$$\begin{pmatrix} a_{11} & & & & \\ & a_{22} & & & \\ & & \ddots & & \\ & & & a_{n-1,n-1} & \\ & & & & a_{nn} \end{pmatrix} - \begin{pmatrix} 0 & & & & \\ -a_{21} & 0 & & & \\ \vdots & \vdots & \ddots & & \\ -a_{n-1,1} & a_{n-1,2} & \cdots & 0 & \\ -a_{n1} & -a_{n2} & \cdots & -a_{n,n-1} & 0 \end{pmatrix} - \begin{pmatrix} 0 & -a_{21} & \cdots & -a_{1,n-1} & -a_{1n} \\ & 0 & \cdots & -a_{2,n-1} & -a_{2n} \\ & & \ddots & \vdots & \vdots \\ & & & 0 & -a_{n-1,n} \\ & & & & 0 \end{pmatrix} \tag{30}$$

定义DLU分解函数

```
function [D, L, U] = DLUDecomposition(A)

    % 名称：
    % 输入：
    %       A: 欲分解矩阵
    % 输出：
    %       D: 对角矩阵
    %       L: 下三角矩阵
    %       U: 上三角矩阵

    %% 函数

    order = size(A, 1);
    D = zeros(size(A));
    L = zeros(size(A));
    U = zeros(size(A));
    for i = 1: order
        D(i, i) = A(i, i);
        for j = 1: order
            if i > j
                L(i, j) = -A(i, j);
            elseif i < j
                U(i, j) = -A(i, j);
            end
        end
    end

```

```
end

```

Jacobi迭代：如果$\det D \neq 0$，那么

$$\begin{gathered}Ax=b \iff x=(I-D^{-1}A)x+D^{-1}b \iff x=B_Jx+f_J \\ x_i^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j\neq i}a_{ij}x_j^{(k)}\right), \qquad 1\leq i\leq n, k\in\mathbb{N}\end{gathered} \tag{31}$$

Gauss-Seidel迭代：如果$\det D \neq 0$，那么

$$\begin{gathered}Ax=b \iff x=(I-(D-L)^{-1}A)x+(D-L)^{-1}b \iff x=B_Gx+f_G \\ x_i^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}x_j^{(k+1)}-\sum_{j=i+1}^{n}a_{ij}x_j^{(k)}\right), \qquad 1\leq i\leq n, k\in\mathbb{N}\end{gathered} \tag{32}$$

逐次超松弛迭代(SOR)迭代：选择松弛因子$w>0$，那么

$$\begin{gathered}Ax=b \iff x=(I-w(D-wL)^{-1}A)x+w(D-wL)^{-1}b \iff x=B_wx+f_w \\ x_i^{(k+1)}=x_i^k+\frac{w}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}x_j^{(k+1)}-\sum_{j=i}^{n}a_{ij}x_j^{(k)}\right), \qquad 1\leq i\leq n, k\in\mathbb{N}\end{gathered} \tag{33}$$

一阶线性定常迭代的基本定理：对于任意初始向量$x^{(0)}$，一阶线性定常迭代$x^{(n+1)}=Bx^{(n)}+f$收敛的充分必要条件为

$$\lim_{n\to\infty}B^n=O \iff \rho(B)<1 \iff \exists\|\cdot\|, \quad \|B\|<1 \tag{34}$$

分别定义迭代函数

```matlab
function [judge, root] = JacobiIteration(A, b, x0, n)

    % 名称:        Jacobi迭代
    % 输入:
    %       A:      系数矩阵
    %       b:      右侧矩阵
    %       x0:     初始解
    %       n:      迭代次数
    % 输出:
    %       judge:  是否收敛
    %       root:   迭代解

    %% 函数

    % DLU分解
    D = DLUDecomposition(A);

    % Jacobi矩阵
    BJ = eye(size(A)) - D \ A;

    % 计算特征值
    eigenvalues = eig(BJ);

    % 判断是否收敛
    if max(abs(eigenvalues)) < 1
        judge = 1;
```

```
        root = x0;
        for k = 1: n
            root = BJ * root + D \ b;
        end
    else
        judge = 0;
        root = [];
    end

end

```

```
function [judge, root] = GaussSeidelIteration(A, b, x0, n)

    % 名称:       Gauss-Seidel迭代
    % 输入:
    %       A:      系数矩阵
    %       b:      右侧矩阵
    %       x0:     初始解
    %       n:      迭代次数
    % 输出:
    %       judge:  是否收敛
    %       root:   迭代解

    %% 函数

    % DLU分解
    [D, L, ~] = DLUDecomposition(A);

    % Gauss-Seidel矩阵
    BG = eye(size(A)) - (D - L) \ A;

    % 计算特征值
    eigenvalues = eig(BG);

    % 判断是否收敛
    if max(abs(eigenvalues)) < 1
        judge = 1;
        root = x0;
        for k = 1: n
            root = BG * root + (D - L) \ b;
        end
    else
        judge = 0;
        root = [];
    end

end

```

```
function [judge, root] = SORIteration(A, b, w, x0, n)

    % 名称:       SOR迭代
    % 输入:
```

```
    %       A:      系数矩阵
    %       b:      右侧矩阵
    %       w:      松弛因子
    %       x0:     初始解
    %       n:      迭代次数
    % 输出:
    %       judge: 是否收敛
    %       root:  迭代解

    %% 函数

    % DLU分解
    [D, L, ~] = DLUDecomposition(A);

    % 松弛矩阵
    Bw = eye(size(A)) - (D - w * L) \ A * w;

    % 计算特征值
    eigenvalues = eig(Bw);

    % 判断是否收敛
    if max(abs(eigenvalues)) < 1
        judge = 1;
        root = x0;
        for k = 1: n
            root = Bw * root + (D - w * L) \ b * w;
        end
    else
        judge = 0;
        root = [];
    end

end

```

定义主函数

```
clear; clc

% 定义系数矩阵与初始解
A = [1, -1, 2, 1;
    -1, 3, 0, -3;
    2, 0, 9, -6;
    1, -3, -6, 19];
b = [1; 3; 5; 7];
x0 = [0; 0; 0; 0];

% Jacobi迭代
JacobiRoot = x0;
JacobiNumber = 0;
while norm(b - A * JacobiRoot) > 1e-6
    JacobiNumber = JacobiNumber + 1;
    [JacobiJudge, JacobiRoot] = JacobiIteration(A, b, x0, JacobiNumber);
end

```

```
% Gauss-Seidel迭代
GaussSeidelRoot = x0;
GaussSeidelNumber = 0;
while norm(b - A * GaussSeidelRoot) > 1e-6
    GaussSeidelNumber = GaussSeidelNumber + 1;
    [GaussSeidelJudge, GaussSeidelRoot] = GaussSeidelIteration(A, b, x0, GaussSeidelNumber);
end

% SOR迭代
SORRootMatrix = [];
SORNumberMatrix = [];
SORJudgeMatrix = [];
for w = [0.8, 1.2, 1.3, 1.6]
    SORRoot = x0;
    SORNumber = 0;
    while norm(b - A * SORRoot) > 1e-6
        SORNumber = SORNumber + 1;
        [SORJudge, SORRoot] = SORIteration(A, b, w, x0, SORNumber);
    end
    SORRootMatrix = [SORRootMatrix, SORRoot];
    SORNumberMatrix = [SORNumberMatrix, SORNumber];
    SORJudgeMatrix = [SORJudgeMatrix, SORJudge];
end

%  创建表格
iterationName = {'Jacobi'; 'Gauss-Seidel'; 'SOR(w=0.8)'; 'SOR(w=1.2)'; 'SOR(w=1.3)'; 'SOR(w=1.6)'};
judge = [JacobiJudge; GaussSeidelJudge; SORJudgeMatrix'];
number = [JacobiNumber; GaussSeidelNumber; SORNumberMatrix'];
root = [JacobiRoot'; GaussSeidelRoot'; SORRootMatrix'];
variableNames = {'迭代方法', '是否收敛', '迭代次数', '迭代解'};
T = table(iterationName, int16(judge), int16(number), vpa(root, 3), 'VariableNames', variableNames);
% 显示表格
disp(T)

```

输出结果

```
        迭代方法           是否收敛     迭代次数                 迭代解
    ________________    ________    ________    ______________________________

    {'Jacobi'      }       1          417       -8.0    0.333    3.67    2.0
    {'Gauss-Seidel'}       1          204       -8.0    0.333    3.67    2.0
    {'SOR(w=0.8)'  }       1          309       -8.0    0.333    3.67    2.0
    {'SOR(w=1.2)'  }       1          136       -8.0    0.333    3.67    2.0
    {'SOR(w=1.3)'  }       1          110       -8.0    0.333    3.67    2.0
    {'SOR(w=1.6)'  }       1           35       -8.0    0.333    3.67    2.0
```

通过输出结果，我们可知这五种迭代方法均收敛，且数值解为

$$x_1 = -8, \qquad x_2 = 0.333, \qquad x_3 = 3.67, \qquad x_4 = 2 \tag{35}$$

迭代次数如结果所示，迭代次数越少，迭代速度越快。

第二题

用共轭梯度法求解方程组$Ax = b$，其中

$$A = \begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 99 & -1 \\ & & & -1 & 100 \end{pmatrix}, \qquad b = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ \vdots \\ 96 \\ 97 \\ 99 \end{pmatrix} \tag{36}$$

若迭代终止条件为$\left\| b - Ax^{(n)} \right\|_2 \leq 10^{-8}$，分别给出数值近似解，迭代步数和计算时间，并计算误差$\left\| x^{(n)} - x^* \right\|_2$，其中$x^*$为方程组的精确解

$$x^* = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \tag{37}$$

解：**共轭梯度法(CG方法)**：

$$\begin{cases} p^{(0)} = r^{(0)} = b - Ax^{(0)} \\ \rho^{(0)} = (r^{(0)}, r^{(0)}) \\ \alpha_0 = \frac{\rho^{(0)}}{(Ap^{(0)}, p^{(0)})} \\ x^{(1)} = x^{(0)} + \alpha_0 p^{(0)} \end{cases}, \qquad \begin{cases} r^{(n)} = b - Ax^{(n)} \\ \rho^{(n)} = (r^{(n)}, r^{(n)}) \\ \beta_n = \frac{\rho^{(n)}}{\rho^{(n-1)}} \\ p^{(n)} = r^{(n)} + \beta_n p^{(n-1)} \\ \alpha_n = \frac{\rho^{(n)}}{(Ap^{(n)}, p^{(n)})} \\ x^{(n+1)} = x^{(n)} + \alpha_n p^{(n)} \end{cases} \tag{38}$$

定义共轭梯度函数

```
function root = conjugateGradient(A, b, x0, n)

    % 名称:         共轭梯度算法
    % 输入:
    %       A:       系数矩阵
    %       b:       右侧矩阵
    %       x0:      初始解
    %       n:       迭代次数
    % 输出:
    %       root:    迭代解

    %% 函数

    p = b - A * x0;
    r = b - A * x0;
    rho = dot(r, r);
    alpha = rho / dot(A * p, p);
    root = x0 + alpha * p;
    if n >= 2
        for k = 2: n
            r = b - A * root;
```

```
            rho0 = rho;
            rho = dot(r, r);
            beta = rho / rho0;
            p = r + beta * p;
            alpha = rho / dot(A * p, p);
            root = root + alpha * p;
        end
    end

end

```

定义主函数

```
clear; clc

% 定义系数矩阵
A = zeros(100, 100);
b = transpose([0, 0, 1: 97, 99]);
for n = 1: 100
    A(n, n) = n;
    if n == 1
        A(n, n + 1) = -1;
    elseif n == 100
        A(n, n - 1) = -1;
    else
        A(n, n + 1) = -1;
        A(n, n - 1) = -1;
    end
end

% 精确根
exactRoot = A \ b;

% 迭代求解近似根
x0 = zeros(100, 1);   % 初始根
approximateRoot = x0; % 近似根
n = 0;
tic % 启动计时器
while norm(b - A * approximateRoot) > 1e-8
    n = n + 1;
    approximateRoot = conjugateGradient(A, b, x0, n);
end
runTime = toc; % 计算时间
error = norm(exactRoot - approximateRoot); % 计算误差

% 输出结果
disp('数值近似解为：')
disp(approximateRoot)
fprintf('迭代步数为：%d步\n', n);
fprintf('计算时间：%f秒\n', runTime)
fprintf('误差为：%e\n', error)

```

输出结果

```
数值近似解为：
    0.999999999999984
    1.000000000000031
    0.999999999999846
    1.000000000000559
    0.999999999998171
    1.000000000005283
    0.999999999986591
    1.000000000029690
    0.999999999943325
    1.000000000091597
    0.999999999878557
    1.000000000123857
    0.999999999919209
    1.000000000001305
    1.000000000068709
    0.999999999924879
    1.000000000011854
    1.000000000056110
    0.999999999946979
    0.999999999985556
    1.000000000055845
    0.999999999984342
    0.999999999956661
    1.000000000030604
    1.000000000029485
    0.999999999964283
    0.999999999980812
    1.000000000035984
    1.000000000013048
    0.999999999965696
    0.999999999989694
    1.000000000031962
    1.000000000010028
    0.999999999970667
    0.999999999988592
    1.000000000026371
    1.000000000013778
    0.999999999977110
    0.999999999983449
    1.000000000018731
    1.000000000019184
    0.999999999986142
    0.999999999978817
    1.000000000008394
    1.000000000022148
    0.999999999997375
    0.999999999978180
    0.999999999996951
    1.000000000020129
    1.000000000008188
    0.999999999982792
    0.999999999987597
    1.000000000013362
    1.000000000015433
```

```
    0.999999999990992
    0.999999999982809
    1.000000000004592
    1.000000000017776
    0.999999999999489
    0.999999999982565
    0.999999999997054
    1.000000000016509
    1.000000000005616
    0.999999999984629
    0.999999999992559
    1.000000000014369
    1.000000000008426
    0.999999999986209
    0.999999999991443
    1.000000000013831
    1.000000000007716
    0.999999999985467
    0.999999999994404
    1.000000000015639
    1.000000000001666
    0.999999999983732
    1.000000000004588
    1.000000000014370
    0.999999999987387
    0.999999999993515
    1.000000000017980
    0.999999999990537
    0.999999999991136
    1.000000000021119
    0.999999999978363
    1.000000000014988
    0.999999999992294
    1.000000000003009
    0.999999999999124
    1.000000000000177
    0.999999999999982
    0.999999999999999
    1.000000000000001
    1.000000000000000
    1.000000000000000
    1.000000000000000
    1.000000000000001
    1.000000000000001
    1.000000000000001
    0.999999999999996

迭代步数为：65步
计算时间：0.020306秒
误差为：3.004497e-10
```

第三题

已知方程

$$x^3-3x-1=0 \tag{39}$$

分别用不动点迭代（取迭代函数为$\varphi(x)=\sqrt[3]{3x+1}$）、Steffensen迭代法（其中不动点迭代的迭代函数仍为$\varphi(x)=\sqrt[3]{3x+1}$)、Newton迭代法、Newton下山法求方程的根，其中除Newton下山法初值为$x_0=0.6$外，其余初值为$x_0=2$。迭代终止条件为$|x_{n+1}-x_n|<10^{-6}$，并分别输出方程的近似根和每种迭代的次数。

解：**不动点迭代**：

$$x_{n+1}=\varphi(x_n) \tag{40}$$

Steffensen迭代：

$$y_n=\varphi(x_n),\qquad z_n=\varphi(y_n),\qquad x_{n+1}=x_n-\frac{(y_n-x_n)^2}{z_n-2y_n+x_n} \tag{41}$$

Newton法：方程$f(x)=0$的迭代

$$x_{n+1}=\varphi(x_n),\qquad \varphi(x)=x-\frac{f(x)}{f'(x)} \tag{42}$$

Newton下山法：方程$f(x)=0$的迭代

$$x_{n+1}=x_n-\lambda_n\frac{f(x_n)}{f'(x_n)} \tag{43}$$

其中下山因子

$$\lambda_n=\max\left\{\frac{1}{2^r}:\left|f\left(x_n-\frac{f(x_n)}{2^rf'(x_n)}\right)\right|<|f(x_n)|,r\in\mathbb{N}\right\} \tag{44}$$

分别定义迭代函数

```
function root = fixedPointIteration(phi, x0, n)

    % 名称:        不动点迭代
    % 输入:
    %       phi:    迭代函数
    %       x0:     初始解
    %       n:      迭代次数
    % 输出:
    %       root:   迭代解

    %% 函数
    root = x0;
    for k = 1: n
        root = phi(root);
    end

end

```

```
function root = SteffensenIteration(phi, x0, n)

    % 名称:        Steffensen迭代
    % 输入:
    %       phi:    迭代函数
```

```
    %       x0:     初始解
    %       n:      迭代次数
    % 输出:
    %       root:   迭代解

    %% 函数
    root = x0;
    for k = 1: n
        y = phi(root);
        z = phi(y);
        root = root - (y - z)^2 / (z - 2 * y + root);
    end

end

```

```
function root = NewtonIteration(fun, x0, n)

    % 名称:        Newton迭代
    % 输入:
    %       fun:    函数
    %       x0:     初始解
    %       n:      迭代次数
    % 输出:
    %       root:   迭代解

    %% 函数
    syms x
    phi = matlabFunction(x - fun(x) ./ diff(fun(x)));
    root = x0;
    for k = 1: n
        root = phi(root);
    end

end

```

```
function root = NewtonDescentIteration(fun, x0, n)

    % 名称:        Newton下山迭代
    % 输入:
    %       fun:    函数
    %       x0:     初始解
    %       n:      迭代次数
    % 输出:
    %       root:   迭代解

    %% 函数
    syms x
    phi = matlabFunction(fun(x) ./ diff(fun(x)));
    root = x0;
    for k = 1: n
        lambda = 1;
        A = abs(fun(root - phi(root) / 2^lambda));
```

```
        B = abs(fun(root));
        while A > B
            lambda = lambda + 1;
            A = abs(fun(root - phi(root) / 2^lambda));
            B = abs(fun(root));
        end
        root = root - lambda * phi(root);
    end

end

```

定义主函数

```
clear; clc

% 不动点迭代
phi = @(x) (3 * x + 1) .^ (1 / 3);
x0 = 2;
fixedPointRoot = fixedPointIteration(phi, x0, 1);
fixedPointRootMatrix = [x0, fixedPointRoot];
fixedPointNumber = 1;
while abs(fixedPointRootMatrix(end) - fixedPointRootMatrix(end - 1)) >= 1e-6
    fixedPointNumber = fixedPointNumber + 1;
    fixedPointRoot = fixedPointIteration(phi, x0, fixedPointNumber);
    fixedPointRootMatrix = [fixedPointRootMatrix, fixedPointRoot];
end

% Steffensen迭代
phi = @(x) (3 * x + 1) .^ (1 / 3);
x0 = 2;
SteffensenRoot = SteffensenIteration(phi, x0, 1);
SteffensenRootMatrix = [x0, SteffensenRoot];
SteffensenNumber = 1;
while abs(SteffensenRootMatrix(end) - SteffensenRootMatrix(end - 1)) >= 1e-6
    SteffensenNumber = SteffensenNumber + 1;
    SteffensenRoot = SteffensenIteration(phi, x0, SteffensenNumber);
    SteffensenRootMatrix = [SteffensenRootMatrix, SteffensenRoot];
end

% Newton迭代
fun = @(x) x^3 - 3*x - 1;
x0 = 2;
NewtonRoot = NewtonIteration(fun, x0, 1);
NewtonRootMatrix = [x0, NewtonRoot];
NewtonNumber = 1;
while abs(NewtonRootMatrix(end) - NewtonRootMatrix(end - 1)) >= 1e-6
    NewtonNumber = NewtonNumber + 1;
    NewtonRoot = NewtonIteration(fun, x0, NewtonNumber);
    NewtonRootMatrix = [NewtonRootMatrix, NewtonRoot];
end

% Newton下山迭代
fun = @(x) x^3 - 3*x - 1;
x0 = 0.6;
```

```
NewtonDescentRoot = NewtonDescentIteration(fun, x0, 1);
NewtonDescentRootMatrix = [x0, NewtonDescentRoot];
NewtonDescentNumber = 1;
while abs(NewtonDescentRootMatrix(end) - NewtonDescentRootMatrix(end - 1)) >= 1e-6
    NewtonDescentNumber = NewtonDescentNumber + 1;
    NewtonDescentRoot = NewtonDescentIteration(fun, x0, NewtonDescentNumber);
    NewtonDescentRootMatrix = [NewtonDescentRootMatrix, NewtonDescentRoot];
end

% 精确解
root = roots([1, 0, -3, -1]);

% 输出结果
disp('精确解为：')
disp(root)
disp('--------------------------------------------')
disp(' ')
%  创建表格
iterationName = {'不动点迭代'; 'Steffensen迭代'; 'Newton迭代'; 'Newton下山迭代'};
number = [fixedPointNumber; SteffensenNumber; NewtonNumber; NewtonDescentNumber];
root = [fixedPointRoot; SteffensenRoot; NewtonRoot; NewtonDescentRoot];
variableNames = {'迭代方法', '迭代次数', '迭代解'};
T = table(iterationName, int16(number), vpa(root, 5), 'VariableNames', variableNames);
% 显示表格
disp(T)

```

输出结果

```
精确解为：
    1.8794
   -1.5321
   -0.3473

--------------------------------------------

         迭代方法          迭代次数      迭代解
    __________________    _______    _______

    {'不动点迭代'      }       10       1.8794
    {'Steffensen迭代'}      112       1.8794
    {'Newton迭代'     }        4       1.8794
    {'Newton下山迭代' }        6      -0.3473
```

第四题

已知$x^* = \sqrt{2}$为方程$x^4 - 4x^2 + 4 = 0$的二重根，分别用重根Newton迭代、求重根的含参数的Newton迭代、改进Newton迭代法求该方程的的近似值，其中初始解为$x_0 = 1.5$，迭代终止条件为$|x_{n+1} - x_n| < 10^{-6}$，给出几种方法的具体迭代步数。

解：**重根Newton法**：如果x^*为方程$f(x) = 0$的m重根，那么迭代

$$x_{n+1} = \varphi(x_n), \qquad \varphi(x) = x - \frac{f(x)}{f'(x)} \tag{45}$$

含参m的Newton迭代法：如果x^*为方程$f(x)=0$的m重根，那么迭代

$$x_{n+1} = \varphi(x_n), \qquad \varphi(x) = x - m\frac{f(x)}{f'(x)} \tag{46}$$

改进Newton迭代法：如果x^*为方程$f(x)=0$的m重根，那么迭代

$$x_{n+1} = \varphi(x_n), \qquad \varphi(x) = x - \frac{\mu(x)}{\mu'(x)}, \qquad \mu(x) = \frac{f(x)}{f'(x)} \tag{47}$$

分别定义迭代函数

```
function root = reRootsNewtonIteration(fun, x0, n)

    % 名称：       重根Newton迭代
    % 输入：
    %       fun:    函数
    %       x0:     初始解
    %       n:      迭代次数
    % 输出：
    %       root:   迭代解

    %% 函数
    syms x
    phi = matlabFunction(x - fun(x) ./ diff(fun(x)));
    root = x0;
    for k = 1: n
        root = phi(root);
    end

end

```

```
function order = orderOfRoot(fun, x0)

    % 名称：       求解函数零点的阶
    % 输入：
    %       fun:    函数
    %       x0:     初始解
    % 输出：
    %       order: x0附近零点的阶

    %% 函数
    syms x
    % 找到最近的根
    roots = solve(fun, x);
    [~, index] = min(abs(roots - x0));
    exactRoot = roots(index);

    % 求解精确根的阶
    order = 1;
    Df = matlabFunction(diff(fun(x)));
    while abs(Df(exactRoot)) < 1e-3
```

```
        order = order + 1;
        Df = matlabFunction(diff(Df(x)));
    end

end

```

```
function root = NewtonIterationWithParameter(fun, x0, n)

    % 名称:        含参Newton迭代
    % 输入:
    %       fun:    函数
    %       x0:     初始解
    %       n:      迭代次数
    % 输出:
    %       root:   迭代解

    %% 函数
    syms x
    order = orderOfRoot(fun, x0);
    phi = matlabFunction(x - order .* fun(x) ./ diff(fun(x)));
    root = x0;
    for k = 1: n
        root = phi(root);
    end

end

```

```
function root = improvingNewtonIteration(fun, x0, n)

    % 名称:        改进Newton迭代
    % 输入:
    %       fun:    函数
    %       x0:     初始解
    %       n:      迭代次数
    % 输出:
    %       root:   迭代解

    %% 函数
    syms x
    mu = matlabFunction(fun(x) ./ diff(fun(x)));
    phi = matlabFunction(x - mu(x) ./ diff(mu(x)));
    root = x0;
    for k = 1: n
        root = phi(root);
    end

end

```

定义主函数

```
clear; clc
```

```
% 重根Newton迭代
fun = @(x) x^4 - 4*x^2 + 4;
x0 = 1.5;
reRootsNewtonRoot = reRootsNewtonIteration(fun, x0, 1);
reRootsNewtonRootMatrix = [x0, reRootsNewtonRoot];
reRootsNewtonNumber = 1;
while abs(reRootsNewtonRootMatrix(end) - reRootsNewtonRootMatrix(end - 1)) >= 1e-6
    reRootsNewtonNumber = reRootsNewtonNumber + 1;
    reRootsNewtonRoot = reRootsNewtonIteration(fun, x0, reRootsNewtonNumber);
    reRootsNewtonRootMatrix = [reRootsNewtonRootMatrix, reRootsNewtonRoot];
end

% 含参Newton迭代
fun = @(x) x^4 - 4*x^2 + 4;
x0 = 1.5;
NewtonWithParameterRoot = NewtonIterationWithParameter(fun, x0, 1);
NewtonWithParameterRootMatrix = [x0, NewtonWithParameterRoot];
NewtonWithParameterNumber = 1;
while abs(NewtonWithParameterRootMatrix(end) - NewtonWithParameterRootMatrix(end - 1)) >= 1e-6
    NewtonWithParameterNumber = NewtonWithParameterNumber + 1;
    NewtonWithParameterRoot = NewtonIterationWithParameter(fun, x0, NewtonWithParameterNumber);
    NewtonWithParameterRootMatrix = [NewtonWithParameterRootMatrix, NewtonWithParameterRoot];
end

% 改进Newton迭代
fun = @(x) x^4 - 4*x^2 + 4;
x0 = 1.5;
improvingNewtonRoot = improvingNewtonIteration(fun, x0, 1);
improvingNewtonRootMatrix = [x0, improvingNewtonRoot];
improvingNewtonNumber = 1;
while abs(improvingNewtonRootMatrix(end) - improvingNewtonRootMatrix(end - 1)) >= 1e-6
    improvingNewtonNumber = improvingNewtonNumber + 1;
    improvingNewtonRoot = improvingNewtonIteration(fun, x0, improvingNewtonNumber);
    improvingNewtonRootMatrix = [improvingNewtonRootMatrix, improvingNewtonRoot];
end

% 输出结果
%  创建表格
iterationName = {'重根Newton迭代'; '含参Newton迭代'; '改进Newton迭代'};
number = [reRootsNewtonNumber; NewtonWithParameterNumber; improvingNewtonNumber];
root = [reRootsNewtonRoot; NewtonWithParameterRoot; improvingNewtonRoot];
variableNames = {'迭代方法', '迭代次数', '迭代解'};
T = table(iterationName, int16(number), vpa(root, 5), 'VariableNames', variableNames);
% 显示表格
disp(T)
```

输出结果

```
             迭代方法                  迭代次数        迭代解
    ____________________    ________     _______

    {'重根Newton迭代'}         17          1.4142
    {'含参Newton迭代'}          8          1.4142
    {'改进Newton迭代'}          4          1.4142
```

第五题

用Euler公式、改进Euler公式、经典四阶Runge-Kutta 方法解下列初值问题

$$\begin{cases} y'(x) = \dfrac{2}{x}y + x^2\mathrm{e}^x, \qquad 1 \le x \le 2 \\ y(1) = 0 \end{cases} \tag{48}$$

为使计算量相当，步长比为$1:2:4$，即三种方法的步长分别为$0.05, 0.1, 0.2$，计算在$x = 1.2, 1.4, 1.8, 2.0$点处的数值解，并与精确解比较误差，其中精确解为

$$y(x) = x^2(\mathrm{e}^x - \mathrm{e}) \tag{49}$$

解：**Euler公式**：

$$y_{n+1} = y_n + hf(x_n, y_n), \qquad x_n = x_0 + nh \tag{50}$$

改进Euler法：

$$y_{n+1} = y_n + \frac{h}{2}(f(x_n, y_n) + f(x_n + h, y_n + hf(x_n, y_n))) \tag{51}$$

经典四阶Runge-Kutta方法：

$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\ K_1 = f(x_n, y_n) \\ K_2 = f\left(x_n + \dfrac{h}{2}, y_n + \dfrac{h}{2}K_1\right) \\ K_3 = f\left(x_n + \dfrac{h}{2}, y_n + \dfrac{h}{2}K_2\right) \\ K_4 = f\left(x_n + h, y_n + hK_3\right) \end{cases} \tag{52}$$

分别定义函数

```matlab
function matrix = EulerFormula(fun, h, x0, xend, y0)

    % 名称:          Euler公式
    % 输入:
    %       fun:      函数
    %       h:        步长
    %       x0:       初始x值
    %       xend:     终止x值
    %       y0:       初始y值
    % 输出:
    %       matrix:   近似解

```

```

    %% 函数
    n = length(x0: h: xend);
    matrix = [x0: h: xend; y0, zeros(1, n-1)];
    for k = 1: n-1
        matrix(2, k+1) = matrix(2, k) + h * fun(matrix(1, k), matrix(2, k));
    end

end

```

```
function matrix = improvingEulerFormula(fun, h, x0, xend, y0)

    % 名称:         改进Euler公式
    % 输入:
    %       fun:      函数
    %       h:        步长
    %       x0:       初始x值
    %       xend:     终止x值
    %       y0:       初始y值
    % 输出:
    %       matrix:   近似解

    %% 函数
    n = length(x0: h: xend);
    matrix = [x0: h: xend; y0, zeros(1, n-1)];
    for k = 1: n-1
        matrix(2, k+1) = matrix(2, k) ...
            + h * fun(matrix(1, k), matrix(2, k)) / 2 ...
            + h * fun(matrix(1, k) + h, matrix(2, k) + h * fun(matrix(1, k), matrix(2, k))) / 2;
    end

end

```

```
function matrix = Classic4RungeKuttaMethod(fun, h, x0, xend, y0)

    % 名称:         经典四阶Runge-Kutta方法
    % 输入:
    %       fun:      函数
    %       h:        步长
    %       x0:       初始x值
    %       xend:     终止x值
    %       y0:       初始y值
    % 输出:
    %       matrix:   近似解

    %% 函数
    n = length(x0: h: xend);
    matrix = [x0: h: xend; y0, zeros(1, n-1)];
    for k = 1: n-1
        K1 = fun(matrix(1, k), matrix(2, k));
        K2 = fun(matrix(1, k) + h/2, matrix(2, k) + h*K1/2);
```

```
        K3 = fun(matrix(1, k) + h/2, matrix(2, k) + h*K2/2);
        K4 = fun(matrix(1, k) + h, matrix(2, k) + h*K3);
        matrix(2, k+1) = matrix(2, k) + h / 6 * (K1 + 2 * K2 + 2 * K3 + K4);
    end

end

```

定义主函数

```
clear; clc

% 定义函数
fun = @(x, y) 2 .* y ./ x + x .^ 2 .* exp(x);
x0 = 1;
xend = 2;
y0 = 0;

% Euler法
EulerMatrix05 = EulerFormula(fun, 0.05, x0, xend, y0);
EulerMatrix1 = EulerFormula(fun, 0.1, x0, xend, y0);
EulerMatrix2 = EulerFormula(fun, 0.2, x0, xend, y0);

% 改进Euler法
improvingEulerMatrix05 = improvingEulerFormula(fun, 0.05, x0, xend, y0);
improvingEulerMatrix1 = improvingEulerFormula(fun, 0.1, x0, xend, y0);
improvingEulerMatrix2 = improvingEulerFormula(fun, 0.2, x0, xend, y0);

% 经典四阶Runge-Kutta方法
RungeKuttaMatrix05 = Classic4RungeKuttaMethod(fun, 0.05, x0, xend, y0);
RungeKuttaMatrix1 = Classic4RungeKuttaMethod(fun, 0.1, x0, xend, y0);
RungeKuttaMatrix2 = Classic4RungeKuttaMethod(fun, 0.2, x0, xend, y0);

% 精确解
exactFunction = @(x) x .^ 2 .* (exp(x) - exp(1));

% 比较结果
matrix = [];
for x = [1.2, 1.4, 1.8, 2.0]
    matrix0 = [0.05, exactFunction(x), ...
    EulerMatrix05(2, EulerMatrix05(1, :) == x),...
    improvingEulerMatrix05(2, improvingEulerMatrix05(1, :) == x),...
    RungeKuttaMatrix05(2, RungeKuttaMatrix05(1, :) == x);
    0.1, exactFunction(x), ...
    EulerMatrix1(2, EulerMatrix1(1, :) == x),...
    improvingEulerMatrix1(2, improvingEulerMatrix1(1, :) == x),...
    RungeKuttaMatrix1(2, RungeKuttaMatrix1(1, :) == x);
    0.2, exactFunction(x), ...
    EulerMatrix2(2, EulerMatrix2(1, :) == x),...
    improvingEulerMatrix2(2, improvingEulerMatrix2(1, :) == x),...
    RungeKuttaMatrix2(2, RungeKuttaMatrix2(1, :) == x)];
    matrix = [matrix; matrix0];
end
matrix12 = matrix(1: 3, :);
matrix14 = matrix(4: 6, :);
```

```matlab
matrix18 = matrix(7: 9, :);
matrix20 = matrix(10: 12, :);

% 输出结果

%  创建表格
variableNames = {'x', '步长', '精确解', 'Euler法', 'Euler法误差', '改进Euler法', '改进Euler法误差', '经典四阶Runge-Kutta方法', 'Runge-Kutta方法误差'};
num = 8;
X = [1.2; 1.2; 1.2; 1.4; 1.4; 1.4; 1.8; 1.8; 1.8; 2.0; 2.0; 2.0];
T = table(X, matrix(:, 1), vpa(matrix(:, 2), num), ...
    vpa(matrix(:, 3), num), vpa(abs(matrix(:, 3) - matrix(:, 2)), num), ...
    vpa(matrix(:, 4), num), vpa(abs(matrix(:, 4) - matrix(:, 2)), num), ...
    vpa(matrix(:, 5), num), vpa(abs(matrix(:, 5) - matrix(:, 2)), num), ...
    'VariableNames', variableNames);
% 显示表格
disp(T)

```

输出结果

```
     x     步长       精确解          Euler法        Euler法误差       改进Euler法      改进Euler法误差     经典四阶Runge-Kutta方法     Runge-Kutta方法误差
    ___    ____    __________    __________    ___________    __________    ______________    _____________________    __________________

    1.2    0.05    0.86664254    0.769696      0.096946536    0.86429069    0.0023518451      0.86664107    0.000014660831
    1.2     0.1    0.86664254    0.68475558    0.18188696     0.85831454    0.0083279984      0.86662169    0.000020843031
    1.2     0.2    0.86664254    0.54365637    0.32298617     0.84053441    0.026108122       0.86637911    0.00026342379
    1.4    0.05    2.6203596     2.3402236     0.28013595     2.6141742     0.0061853358      2.6203562     0.0000033682149
    1.4     0.1    2.6203596     2.0935477     0.52681186     2.5982982     0.022061312       2.6203113     0.000048245364
    1.4     0.2    2.6203596     1.6810688     0.93929072     2.5502404     0.070119148       2.6197405     0.00061903077
    1.8    0.05    10.793625     9.7434894     1.0501353      10.774418     0.019206872       10.793616     0.0000084984631
    1.8     0.1    10.793625     8.8091197     1.984505       10.724467     0.0691576         10.793502     0.00012287684
    1.8     0.2    10.793625     7.2247183     3.5689063      10.569818     0.22380681        10.792018     0.001607063
      2    0.05    18.683097     16.949013     1.7340838      18.654245     0.028851759       18.683085     0.000011755683
      2     0.1    18.683097     15.398236     3.2848614      18.578882     0.10421463        18.682927     0.00017051423
      2     0.2    18.683097     12.750383     5.9327142      18.343834     0.33926303        18.680852     0.0022447174
```