写 之 前 记 得 加 上

clear;clc; !!!!!!!!!!!!!!!!!!!

二分法:

```
function res = division_fun(f, a, b, e)
% f:函数指针
% a:区间左端点
% b:区间右端点
% e:精度
while abs(b - a) > e
    mid = f((a + b) / 2);
    if mid == 0
        res = (a + b) / 2;
        break
    elseif mid * f(a) < 0
            b = (a + b) / 2;
    elseif mid * f(a) > 0
            a = (a + b) / 2;
    end
end
res = (a + b) / 2;
end
```

不动点迭代法:

```
function res = fix_point_fun(f, x0, e)
% f:函数指针
% x0:迭代初值
% e:精度
x1 = f(x0);
while abs(x1 - x0) >= e
    x0 = x1;
    x1 = f(x0);
end
res = x1;
end
```

牛顿迭代法:

```
function res = newton_fun(f, x0, e)
% f:函数
% x0:初始迭代值
% e:精度
x1 = x0 - f(x0) / dif_fun(f, x0);
while abs(x1 - x0) >= e
    x0 = x1;
    x1 = x0 - f(x0) / dif_fun(f, x0);
end
res = x1;
end
```

牛顿下山法:

```
function res = newton_downhill_fun(f, x0, e)
% f:函数
% x0:初始迭代值
% e:精度
x1 = x0 - f(x0) / dif_fun(f, x0);
while abs(x1 - x0) >= e
    lambda = 1;
    while abs(f(x1)) >= abs(f(x0))
        lambda = lambda / 2;
        x1 = x0 - lambda * f(x0) / dif_fun(f, x0);
    end
    lambda = 1;
    x0 = x1;
    x1 = x0 - lambda * f(x0) / dif_fun(f, x0);
end
res = x1;
end
```

牛顿插值:

```
function res = newton_interpolation_fun(X, Y)
% X:横坐标向量
% Y:纵坐标向量
ps = [];
n = length(X);
for i = 1:n
    ps = [ps; poly(X(i))];
end
T = Y;
for i = 1:n-1
    ls = zeros(1, n);
    for j = i:n-1
        ls(j+1) = (T(i, j+1) - T(i, j)) / (X(j+1) - X(j+1-i));
    end
    T = [T; ls];
end

res = zeros(1, n);
for i = 2:n
```

```
    xp = 1;
    for j = 1:i-1
        xp = conv(xp, ps(j, :));
    end
    xpf = zeros(1, n);
    for j = 1:length(xp)
        xpf(n-length(xp)+j) = xp(j);
    end
    res = res + T(i, i) * xpf;
    disp(res)
end
res(n) = res(n) + T(1, 1);
res = res';
end
```

新牛顿插值：

```
function f = new_newton_interpolation_fun(X, Y)
n=length(X);
T=zeros(n,n);
% 对差商表第一列赋值
for k=1:n
    T(k)=Y(k);
end
% 求差商表
for i=2:n
    for k=i:n

T(k,i)=(T(k,i-1)-T(k-1,i-1))/(X(k)-X(k+1-i));
    end
end
f = @(x)(T(1,1)+0*x);
for i = 2:length(X)
    w = @(x)(1+0*x);
    for j = 1:i-1
        w = @(x)(w(x).*(x-X(j)));
    end
    f = @(x)(f(x)+T(i, i).*w(x));
end
end
```

拉格朗日插值：

```
function res = lagrange_interpolation_fun(X, Y)
% X:横坐标向量
% Y:纵坐标向量
ps = [];
for i = 1:length(X)
    ps = [ps; poly(X(i))];
end
ls = [];
for i = 1:length(ps)
    ll = 1;
    for j = 1:length(ps)
        if j == i
            continue
        end
        ll = conv(ll, ps(j, :));
    end
    div_ll = 1;
    for j = 1:length(ps)
        if j == i
            continue
        end
        div_ll = div_ll * (X(i) - X(j));
    end
    ll = ll ./ div_ll;
    ls = [ls; ll];
end
res = 0;
for i = 1:length(ls)
    res = res + ls(i, :) * Y(i);
end
end
```

Jacobi 迭代：

```
function x = jacobi_fun(a, b, x0, e)
% a:系数矩阵
% b:右边向量
% x0:初始向量(列向量)
% e:精度
% 最大迭代次数 M
n = length(b);
M = 100000;
m = 0;
x = zeros(n, 1);

 % 求系数矩阵的对角矩阵
cm_diag = diag(diag(a));
B = cm_diag \ (cm_diag - a);
% 计算谱半径
```

```
R = max(abs(eig(B)));
if R >= 1
    x = zeros(n, 1);
    disp('谱半径不小于 1，无法收敛')
    return
end

while m <= M
    m = m + 1;
    for i = 1:n
        sum_ax = 0;
        for j = 1:n
            if j == i
                continue
            end
            sum_ax = sum_ax + a(i, j) * x0(j);
        end
        x(i) = -(sum_ax - b(i)) / a(i, i);
    end
    if norm(x - x0, 1) < e
        break
    end
    x0 = x;
end
if m > M
    disp('达到最大循环次数')
end
end
```

G-S 迭代：

```
function x = gauss_seidel_fun(a, b, x0, e)
% a:系数矩阵
% b:右边向量
% x0:初始向量(列向量)
% e:精度
% 最大迭代次数 M
n = length(b);

 % 求系数矩阵的对角矩阵
cm_diag = diag(diag(a));
B = cm_diag \ (cm_diag - a);
% 计算谱半径
R = max(abs(eig(B)));
if R >= 1
    x = zeros(n, 1);
    disp('谱半径不小于 1，无法收敛')
    return
end

M = 100000;
m = 0;
x = zeros(n, 1);
while m <= M
    m = m + 1;
    for i = 1:n
        sum_ax = 0;
        for j = 1:i-1
            sum_ax = sum_ax + a(i, j) * x(j);
        end
        for j = i+1:n
            sum_ax = sum_ax + a(i, j) * x0(j);
        end
        x(i) = -(sum_ax - b(i)) / a(i, i);
    end
    if norm(x - x0, 1) < e
        break
    end
    x0 = x;
end
if m > M
    disp('达到最大循环次数')
end
end
```

高斯消去法：

```
function x = gauss_elimi_fun(a, b)
% a:系数矩阵
% b：右边向量
% n：方程组的阶数
% x: 求解结果列向量
n = length(b);
m = zeros(n, n);
x = zeros(n, 1);
for k = 1:n-1
    if det(a) == 0
        disp('第'+str(k)+'次迭代的矩阵 a 的顺序主子式为 0，无法继续运算');
```

```
            return
        end
        for i = k+1:n
            m(i, k) = a(i, k) / a(k, k);
            for j = k+1:n
                a(i, j) = a(i, j) - m(i, k) * a(k, j);
            end
            b(i) = b(i) - m(i, k) * b(k);
        end
end
x(n) = b(n) / a(n, n);
for i = n-1:-1:1
        sum_of_ax = 0;
        for j = i+1:n
            sum_of_ax = sum_of_ax + a(i, j) *
x(j);
        end
        disp(sum_of_ax)
        x(i) = (b(i) - sum_of_ax) / a(i, i);
end
end
```

列主元高斯消去法：

```
function x = col_pivot_gauss_elimi_fun(a, b)
% a:系数矩阵
% b：右边向量
% n：方程组的阶数
% x：求解结果列向量
n = length(b);
m = zeros(n, n);
x = zeros(n, 1);
for k = 1:n-1
        if det(a) == 0
            disp('第'+str(k)+'次迭代的矩阵 a
的顺序主子式为 0，无法继续运算');
            return
        end
        max_ai = k;
        max_a = a(k, k);
        for i = k:n
            if a(i, k) > max_a
                max_a = a(i, k);
                max_ai = i;
            end
        end
        if max_ai ~= k
            tmp = a(k, :);
            a(k, :) = a(max_ai, :);
            a(max_ai, :) = tmp;
        end
        for i = k+1:n
            m(i, k) = a(i, k) / a(k, k);
            for j = k+1:n
                a(i, j) = a(i, j) - m(i, k) * a(k, j);
            end
            b(i) = b(i) - m(i, k) * b(k);
        end
end
x(n) = b(n) / a(n, n);
for i = n-1:-1:1
        sum_of_ax = 0;
        for j = i+1:n
            sum_of_ax = sum_of_ax + a(i, j) *
x(j);
        end
        x(i) = (b(i) - sum_of_ax) / a(i, i);
end
end
```

三次样条插值：

```
function                res                =
cubic_spline_interpolation_fun(X, Y, condi)
% 目前只能设定自然条件
% X:横坐标向量
% Y:纵坐标向量
% condi:边界条件值
n = length(X);
form = zeros(n,n);
form(:,1)=Y;
M0 = condi(1);
Mn = condi(2);
for i=2:n
        for j=i:n
            form(j,i)                          =
(form(j,i-1)-form(j-1,i-1))/(X(j)-X(j-i+1));
        end
end
h = zeros(n-1,1);
for i=1:n-1
        h(i)=X(i+1)-X(i);
```

```
end
b = zeros(n-2,1);
c = zeros(n-2,n-2);
for i=1:n-2
    c(i,i)=2;
    if (i==1)
        b(i,1)=6 * form(i+2,3)-h(i)/(h(i)+h(i+1))* M0;
    elseif (i==(n-2))
        b(i,1)=6 * form(i+2,3)-(h(i+1)/(h(i)+h(i+1)))* Mn;
    else
        b(i,1)=6 * form(i+2,3);
    end
end
for i=2:n-2
    c(i,i-1)= h(i)/(h(i)+h(i+1));
    c(i-1,i)= h(i)/(h(i-1)+h(i));
end
c(1,n-2) = h(1)/(h(1)+h(2));
c(n-2,1) = h(n-1)/(h(n-2)+h(n-1));
M = c\b;
M = [M0; M; Mn];
res = [];
for i = 1:n-1
    s1 = conv(conv([-1, X(i+1)], [-1, X(i+1)]), [-1, X(i+1)]);
    s1 = s1 * M(i) / (6 * h(i));
    s2 = conv(conv([1, -X(i)], [1, -X(i)]), [1, -X(i)]);
    s2 = s2 * M(i+1) / (6 * h(i));
    s3 = [-1, X(i+1)] * (Y(i) - M(i) * h(i) * h(i) / 6) / h(i);
    s4 = [1, -X(i)] * (Y(i+1) - M(i+1) * h(i) * h(i) / 6) / h(i);
    ss = zeros(1, 4);
    for j = 1:length(s1)
        ss(4-length(s1)+j) = ss(4-length(s1)+j) + s1(j);
    end
    for j = 1:length(s2)
        ss(4-length(s2)+j) = ss(4-length(s2)+j) + s2(j);
    end
    for j = 1:length(s3)
        ss(4-length(s3)+j) = ss(4-length(s3)+j) + s3(j);
    end
    for j = 1:length(s4)
        ss(4-length(s4)+j) = ss(4-length(s4)+j) + s4(j);
    end
    res = [res; ss];
end

end
```

二阶导:

```
function res = ddif_fun(f, x)
% 求函数数值二阶导
e = 0.0001;
left = diff([f(x-e), f(x)])/e;
right = diff([f(x), f(x+e)])/e;
res = diff([left, right])/e;
end
```

一阶导:

```
function res = dif_fun(f, x)
% 求函数数值一阶导
% f:需要求导的函数
% x:求 x 处的导数值
e = 0.00000001;
res = diff([f(x), f(x+e)])/e;
end
```

展示多项式:

```
function disp_fun(X)
% X:系数矩阵
% 系数个数 n
n = length(X);
if n == 1
    fprintf('%.2f', X(1))
elseif n > 1
    res = '';
    for i = 1:n
        if i == 1
            if X(i) == -1
                res = [res, '-'];
            elseif X(i) == 1.0
                res = res;
            else
```

```
                res = [res, num2str(X(i))];
            end
            res = [[res, 'x^'], num2str(n-i)];
        elseif i < n - 1
            if X(i) > 0
                res = [[[[res, '+'], num2str(X(i))], 'x^'], num2str(n-i)];
            elseif X(i) < 0
                res = [[[res, num2str(X(i))], 'x^'], num2str(n-i)];
            end
        elseif i == n - 1
            if X(i) > 0
                res = [[[res, '+'], num2str(X(i))], 'x'];
            elseif X(i) < 0
                res = [[res, num2str(X(i))], 'x'];
            end
        else
            if X(i) > 0
                res = [[res, '+'], num2str(X(i))];
            elseif X(i) < 0
                res = [res, num2str(X(i))];
            end
        end
    end
end
disp(res)
end
```

句柄函数绘图:

```
function f_plot_fun(f, X, Y, e, x0, y0)
% f:函数句柄
% X:插值节点横坐标
% Y:插值节点纵坐标
% left:左边界
% right:右边界
% e:绘制图像中的两点间隔
% x:（可选）要预测的点横坐标
% y:（可选）要预测的点纵坐标
if nargin > 4
    left = min([x0 - 2 * e, X(1) - 2 * e]);
    right = max([x0 + 2 * e, X(length(X)) + 2 * e]);
else
    left = X(1) - 2 * e;
    right = X(length(X)) + 2 * e;
end
x = left:e:right;
y = zeros(1, length(x));
for i = 1:length(x)
    y(i) = f(x(i));
end
plot(x, y, 'b');
hold on
plot(X, Y, 'ro');
if nargin > 4
    hold on
    plot(x0, y0, '*');
end
end
```

三次埃尔米特插值:

```
function res = hermite_fun(X,Y,y0,yn)
% y0:左边界导数值
% yn:右边界导数值
x_input = X;
y_input = Y;
n = length(X);
y_0 = y0;
y_n = yn;
[~,number] = size(x_input);
delta_h = zeros(1,number-1);
delta_f = zeros(1,number-1);
lambda_ = zeros(1,number-2);
miu = zeros(1,number-2);
e = zeros(1,number-2);
for i = 1:(number-1)
    delta_h(i) = x_input(i+1) - x_input(i);
    delta_f(i) = (y_input(i+1) - y_input(i))/ delta_h(i);
end
for i=1:number-2
    lambda_(1,i) = delta_h(1,i+1) / (delta_h(1,i+1) + delta_h(1,i));
     miu(1,i) = 1 - lambda_(1,i);
     e(1,i) = 3*(lambda_(1,i)*delta_f(1,i) +
```

```
miu(1,i)*delta_f(1,i+1));
end
A = zeros(number-2,number-2);
B = zeros(number-2,1);
A(1,1) = 2;
A(1,2) = miu(1,1);
B(1,1) = e(1,1) - lambda_(1,1) * y_0;
for i = 2:number-3
    B(i,1) = e(1,i);
    A(i,i-1) = lambda_(1,i);
    A(i,i) = 2;
    A(i,i+1) = miu(1,i);
end
A(number-2,number-3) = lambda_(1,number-2);
A(number-2,number-2) = 2;
B(number-2,1) = e(1,number-2) - miu(1,number-2)*y_n;
m_matrix = A\B;
m = zeros(1,number);
m(1) = y_0;
m(number) = y_n;
for i = 2:number-1
    m(i) = m_matrix(i-1,1);
end
res = [];
for i = 1:n-1
    s1 = conv([1/(X(i)-X(i+1)), -X(i+1)/(X(i)-X(i+1))], [1/(X(i)-X(i+1)), -X(i+1)/(X(i)-X(i+1))]);
    s1 = conv(s1, [2/(X(i+1)-X(i)), 1-2*X(i)/(X(i+1)-X(i))]);
    s1 = s1 * Y(i);
    s2 = conv([1/(X(i+1)-X(i)), -X(i)/(X(i+1)-X(i))], [1/(X(i+1)-X(i)), -X(i)/(X(i+1)-X(i))]);
    s2 = conv(s2, [2/(X(i)-X(i+1)), 1-2*X(i+1)/(X(i)-X(i+1))]);
    s2 = Y(i+1) * s2;
    s3 = conv([1/(X(i)-X(i+1)), -X(i+1)/(X(i)-X(i+1))], [1/(X(i)-X(i+1)), -X(i+1)/(X(i)-X(i+1))]);
    s3 = m(i) * conv(s3, [1, -X(i)]);
    s4 = conv([1/(X(i+1)-X(i)), -X(i)/(X(i+1)-X(i))], [1/(X(i+1)-X(i)), -X(i)/(X(i+1)-X(i))]);
    s4 = m(i+1) * conv(s4, [1, -X(i+1)]);
    res = [res; s1+s2+s3+s4];
end
end
```

分段插值函数绘图：

```
function multi_poly_plot_fun(param, X, Y, e, x0, y0)
% param:多项式系数向量
% X:插值节点横坐标
% Y:插值节点纵坐标
% left:左边界
% right:右边界
% e:绘制图像中的两点间隔
% x:（可选）要预测的点横坐标
% y:（可选）要预测的点纵坐标
[n, m] = size(param);
for i = 1:n
    if i == 1
        if nargin > 4
            left = min([x0 - 3 * e, X(i) - 3 * e]);
            x = left:e:X(i+1);
        else
            x = (X(i)-3 * e):e:X(i+1);
        end
    elseif i == n
        if nargin > 4
            right = max([x0 + 3 * e, X(i+1) + 3 * e]);
            x = X(i):e:right;
        else
            x = X(i):e:X(i+1) + 3 * e;
        end
    else
        x = X(i):e:X(i+1);
    end
    y = zeros(1, length(x));
    for j = 1:m
        y = y + param(i, j) * x .^ (m-j);
    end
    plot(x, y, 'b');
    hold on
```

```
end
plot(X, Y, 'ro');
if nargin > 6
    hold on
    plot(x0, y0, '*');
end
end
```

牛顿前向插值

```
function [res, param] = newton_forward_interpolation_fun(X, Y, x)
% X:横坐标向量
% Y:纵坐标向量
% x:插值节点
n = length(X);
inter = X(2) - X(1);
dt = Y;
for i = 2:n
    dl = zeros(1, n);
    for j = i:n
        dl(j) = dt(i-1, j) - dt(i-1, j-1);
    end
    dt = [dt; dl];
end
t = (x - X(1)) / inter;
param = zeros(1, n);
pse = zeros(1, n);
pse(n) = dt(1, 1);
param = param + pse;
ps = 1;
for i = 2:n
    ps = conv(ps, [1, 2-i]);
    for j = 1:length(ps)
        pse(n-length(ps)+j) = ps(j);
    end
    pse = pse * dt(i, i);
    pse = pse ./ factorial(i-1);
    param = param + pse;
end
res = 0;
for i = 1:n
    res = res + param(i) * t^(n-i);
end
end
```

已知重根数的牛顿迭代法:

```
function res = newton_know_root_fun(f, x0, m, e)
% f:函数
% x0:初始迭代值
% m:m 重根
% e:精度
x1 = x0 - f(x0) / dif_fun(f, x0);
while abs(x1 - x0) >= e
    x0 = x1;
    x1 = x0 - m * f(x0) / dif_fun(f, x0);
end
res = x1;
end
```

未知重根数的牛顿迭代法:

```
function res = newton_unknow_root_fun(f, x0, e)
% f:函数
% x0:初始迭代值
% e:精度
x1 = x0 - f(x0) / dif_fun(f, x0);
while abs(x1 - x0) >= e
    x0 = x1;
    x1 = x0 - dif_fun(f, x0) * f(x0) / (dif_fun(f, x0) * dif_fun(f, x0) - f(x0) * ddif_fun(f, x0));
end
res = x1;
end
```

线性插值:

```
function res = piecewise_linear_interpolation_fun(X, Y)
% X:横坐标向量
% Y:纵坐标向量
n = length(X);
res = zeros(n-1, 2);
for i = 1:n-1
    res(i, 1) = Y(i) / (X(i) - X(i+1)) + Y(i+1) / (X(i+1) - X(i));
    res(i, 2) = -X(i+1) * Y(i) / (X(i) - X(i+1)) - X(i) * Y(i+1) / (X(i+1) - X(i));
end
```

多项式求值:

```
function y = poly_value_fun(param, x)
y = 0;
for j = 1:length(param)
```

```
    y = y + param(j) * x ^ (length(param)-j);
end

end
```

SOR 迭代法:

```
function x = sor_fun(a, b, n, x0, e, w)
% a:系数矩阵
% b:右边向量
% x0:初始向量(列向量)
% e:精度
% w:松弛因子
% 最大迭代次数 M
M = 100000;
m = 0;
x = zeros(n, 1);
while m <= M
    m = m + 1;
    for i = 1:n
        sum_ax = 0;
        for j = 1:i-1
            sum_ax = sum_ax + a(i, j) * x(j);
        end
        for j = i+1:n
            sum_ax = sum_ax + a(i, j) * x0(j);
        end
        if i == 0
            x(i) = -(sum_ax - b(i)) / a(i, i);
        elseif i > 0
            x(i) = -w * (sum_ax - b(i)) / a(i, i) + (1 - w) * x0(i);
        end
    end
    if norm(x - x0, 1) < e
        break
    end
    x0 = x;
end
if m > M
    disp('达到最大循环次数')
end
end
```

牛顿法

牛顿迭代公式:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

算法:

- Step 0: 给定初始估计 x_0，以及预设精度 ε
- Step 1: 计算$x_1 = x_0 - f(x_0)/f'(x_0)$
- Step 2: 若$|x_1 - x_0| < \varepsilon$，则停止，输出近似解 x_1;否则，令 $x_0 = x_1$，返回Step 1。

解决初值问题—牛顿下山法

- 如何保证单调性呢?
- 将牛顿迭代公式改为

$$x_{k+1} = x_k - \lambda\frac{f(x_k)}{f'(x_k)}$$

- 其中，λ是下山因子。
- 选择合适的下山因子以保证单调性。
- 可以采取逐步搜索的方式，从$\lambda = 1$开始，逐次取前一次的一半，直到单调性满足。

解决重根问题

- 当 m 已知时，由于x^*是方程$f(x)^{1/m} = 0$的单根，对此方程应用牛顿迭代公式，有

$$x_{k+1} = x_k - m\frac{f(x_k)}{f'(x_k)}, k = 0,1,2, \dots$$

- 当 m 未知时，令$u(x) = f(x)/f'(x)$，则 x^*是方程$u(x) = 0$的单根。对$u(x)$用牛顿法进行求解，其迭代公式如下

$$x_{k+1} = x_k - m\frac{f'(x_k)f(x_k)}{f'(x_k)^2 - f(x_k)f''(x_k)}, k = 0,1,2, \dots$$

Gauss 消去法的算法

Step 0: 输入方程组的阶数 n，系数矩阵 A 和右边向量 b

Step 1: 对 $k = 1, 2, \dots, n-1$，$i, j = k+1, k+2, \cdots, n$，假设 $a_{kk} \neq 0$，计算

$$\begin{cases} m_{ik} = a_{ik}/a_{kk} \\ a_{ij} = a_{ij} - m_{ik}a_{kj} \\ b_i = b_i - m_{ik}b_k \end{cases}$$

Step 2： 对 $i = n-1, n-2, ..., 1$，计算

$$\begin{cases} x_n = b_n / a_{nn} \\ x_i = \dfrac{b_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}} \end{cases}$$

列主元 Gauss 消去法的算法

Step 0： 输入方程组的阶数 n，系数矩阵 A 和右边向量 b

Step 1： 对 $k = 1, 2, ..., n-1$，

计算 $|a_{i_k k}| = \max_{k \le i \le n} \{|a_{ik}|\}$；

如果 $|a_{i_k k}| = 0$，则停止；否则，

若 $i_k \neq k$，则交换 A 和 b 的第 i_k 行与第 k 行；

列主元 Gauss 消去法的算法

Step 1： 对 $i, j = k+1, k+2, \cdots, n$，计算

$$\begin{cases} m_{ik} = a_{ik} / a_{kk} \\ a_{ij} = a_{ij} - m_{ik} a_{kj} \\ b_i = b_i - m_{ik} b_k \end{cases}$$

Step 2： 对 $i = n-1, n-2, ..., 1$，计算

$$\begin{cases} x_n = b_n / a_{nn} \\ x_i = \dfrac{b_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}} \end{cases}$$

Jacobi 迭代的算法

Step 0: 输入方程组的阶数 n，系数矩阵 A 和右边向量 b，初始向量 x_0，误差要求 e，最大迭代次数 M，$m = 0$

Step 1: 对 $i = 1, 2, ..., n$，计算

$$x_i = -\frac{1}{a_{ii}}\left(\sum_{j=1, j\neq i}^{n} a_{ij} x_{0j} - b_i\right), m = m+1 \quad (4)$$

Step 2: 若 $||x - x_0|| < e$，则计算停止，输出 x；否则若 $m > M$，则终止，输出“达到最大循环次数”；否则令 $x_0 = x$，返回到 Step 1。

- 把系数矩阵 A 写成

$$A = D + L + U \quad (5)$$

的形式。其中，D 是由 A 的对角线元素组成的对角矩阵，L 和 U 分别为的严格下三角和严格上三角部分构成的严格三角形矩阵。

- 从而，公式 (7) 的矩阵形式为

$$x^{(k+1)} = -D^{-1}(L+U)x^{(k)} + D^{-1}b, k = 0, 1, 2, ... \quad (6)$$

- Jacobi 迭代矩阵为

$$B = -D^{-1}(L+U) \quad (7)$$

Gauss-Seidel 迭代的算法

Step 0: 输入方程组的阶数 n，系数矩阵 A 和右边向量 b，初始向量 x_0，误差要求 e，最大迭代次数 M，$m = 0$

Step 1: 对 $i = 1, 2, ..., n$，计算

$$x_i = -\frac{1}{a_{ii}}\left(\sum_{j=1}^{i-1} a_{ij} x_j + \sum_{j=i+1}^{n} a_{ij} x_{0j} - b_i\right), m = m+1 \quad (8)$$

Step 2: 若 $||x - x_0|| < e$，则计算停止，输出 x；否则若 $m > M$，则终止，输出“达到最大循环次数”；否则 $x_0 = x$，返回到 Step 1。

- Gauss-Seidel 迭代格式的矩阵形式为

$$x^{(k+1)} = -(D+L)^{-1}Ux^{(k)} + (D+L)^{-1}b, k = 0, 1, 2, ... \quad (9)$$

- 迭代矩阵 B 为

$$B = -(D+L)^{-1}U \quad (10)$$

拉格朗日插值:

- $l_0(x), l_1(x), ..., l_n(x)$ 构成不超过 n 次多项式集合的一组基。
- $l_0(x), l_1(x), ..., l_n(x)$ 只与插值节点有关，和 $f(x)$ 的值无关。
- 引入函数 $w_{n+1}(x) = (x - x_0)(x - x_1)\cdots(x - x_n)$，则拉格朗日插值基函数可以表示为

$$l_k(x) = \frac{w_{n+1}(x)}{(x - x_k)w'_{n+1}(x_k)}, k = 0, 1, ..., n \quad (5)$$

- 已知函数 $y = f(x)$ 在 $x_0 = -2$，$x_1 = -1$，$x_2 = 0$，$x_3 = 1$ 处的值分别为 $y_0 = 3$，$y_1 = 1$，$y_2 = 1$，$y_3 = 6$，据此构造拉格朗日插值多项式并求 $f(0.5)$ 的近似值。

解：有之前的定义可以构造

$$\begin{aligned} L_3(x) &= y_0 l_0(x) + y_1 l_1(x) + y_2 l_2(x) + y_3 l_3(x) \\ &= 3 * \frac{(x+1)(x-0)(x-1)}{(-2+1)(-2-0)(-2-1)} + 1 * \frac{(x+2)(x-0)(x-1)}{(-1+2)(-1-0)(-1-1)} \\ &+ 1 * \frac{(x+2)(x+1)(x-1)}{(0+2)(0+1)(0-1)} + 6 * \frac{(x+2)(x+1)(x-0)}{(1+2)(1+1)(1-0)} \\ &= 0.5x^3 + 2.5x^2 + 2x + 1 \end{aligned}$$

给定函数 $f(x)$ 在 (a, b) 上 $n+1$ 个互异节点 $x_0, x_1, ..., x_n$ 处的函数值 $f(x_i), i = 0, 1, 2, ..., n$，称

$$f(x_i, x_j) = \frac{f(x_i) - f(x_j)}{x_i - x_j} \quad (10)$$

为 $f(x)$ 关于点 x_i 及 x_j 的一阶差商；

$$f(x_i, x_j, x_k) = \frac{f(x_i, x_j) - f(x_j, x_k)}{x_i - x_k} \quad (11)$$

为 $f(x)$ 关于点 x_i、x_j 及 x_k 的二阶差商。

■ 记公式（13）为

$$f(x) = N_n(x) + R_n(x) \tag{14}$$

其中，

$$\begin{aligned} N_n(x) = & f(x_0) + f(x_0, x_1)(x - x_0) + f(x_0, x_1, x_2)(x - x_0)(x - x_1) + \cdots \\ & + f(x_0, x_1, ..., x_n)(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \end{aligned} \tag{15}$$

$$R_n(x) = f(x, x_0, x_1, ..., x_n)(x - x_0)(x - x_1)\cdots(x - x_n) \tag{16}$$

牛顿前插：

■ 已知在等距节点 $x_k = x_0 + kh$，$k = 0, 1, ..., n$ 处的函数值 $f_k = f(x_k)$

■ 对于点 x，可令 $x = x_0 + th$，牛顿前插公式为

$$\begin{aligned} N_n(x) = & f_0 + t \triangle f_0 + \frac{t(t-1)}{2!} \triangle^2 f_0 + \cdots \\ & + \frac{t(t-1)\cdots(t-n+1)}{n!} \triangle^n f_0 \end{aligned} \tag{3}$$

■ 插值余项为

$$R_n(x) = \frac{t(t-1)\cdots(t-n)}{(n+1)!} h^{n+1} f^{(n+1)}(\xi), \xi \in (x_0, x_n) \tag{4}$$

■ 给出 $f(x) = cosx$ 在等距节点 $0 : 0.1 : 0.5$ 处的函数值，试用 4 次 Newton 前插公式计算 $f(0.048)$ 的近似值，并估计误差。

■ 去等距节点 $0, 0.1, 0.2, 0.3, 0.4$，做查分表

x_k	$f(x_k)$	$\triangle f$	$\triangle^2 f$	$\triangle^3 f$	$\triangle^4 f$
0.0	1.00000				
0.1	0.99500	-0.00500			
0.2	0.98007	-0.01493	-0.00993		
0.3	0.95534	-0.02473	-0.00980	-0.00013	
0.4	0.92106	-0.03428	-0.00955	-0.00025	-0.00012

■ 通过插值点 $x = 0.048$，计算出 $t = (x - x_0)/0.1 = 0.48$。

■ 代入公式（3），可以计算得到结果，约等于 0.99884。

分段线性插值：

■ 容易求得，在每个区间 (x_i, x_{i+1}) 上，

$$I_h(x) = \frac{x - x_{i+1}}{x_i - x_{i+1}} y_i + \frac{x - x_i}{x_{i+1} - x_i} y_{i+1} \tag{5}$$

■ 令 $M_2 = \max\limits_{a \le x \le b} |f''(x)|, h = \max\limits_{0 \le n-1} (x_{i+1} - x_i)$，则对于任意的 $x \in (a, b)$，插值余项满足

$$|R(x)| = |f(x) - I_h(x)| \le \frac{M_2}{8} h^2 \tag{6}$$

埃尔米特插值：

■ 利用构造法，可以得到基函数 $\alpha_k(x)$、$\alpha_{k+1}(x)$、$\beta_k(x)$ 和 $\beta_{k+1}(x)$ 的具体形式。

■ 最终，$H_3(x)$ 在 (x_k, x_{k+1}) 的表达式为

$$\begin{aligned} H_3(x) = & y_k(1 + 2\frac{x - x_k}{x_{k+1} - x_k})(\frac{x - x_{k+1}}{x_k - x_{k+1}})^2 \\ & + y_{k+1}(1 + 2\frac{x - x_{k+1}}{x_k - x_{k+1}})(\frac{x - x_k}{x_{k+1} - x_k})^2 \\ & + m_k(x - x_k)(\frac{x - x_{k+1}}{x_k - x_{k+1}})^2 + m_{k+1}(x - x_{k+1})(\frac{x - x_k}{x_{k+1} - x_k})^2 \end{aligned} \tag{10}$$

三次样条插值：

■ 利用插值条件 $S(x_i) = y_i$，$S(x_{i+1}) = y_{i+1}$，得到

$$\begin{aligned} S_i(x) = & \frac{(x_{i+1} - x)^3}{6h_i} M_i + \frac{(x - x_i)^3}{6h_i} M_{i+1} \\ & + (y_i - \frac{M_i h_i^2}{6}) \frac{x_{i+1} - x}{h_i} + (y_{i+1} - \frac{M_{i+1} h_i^2}{6}) \frac{x - x_i}{h_i} \end{aligned} \tag{4}$$

■ 针对第二类边界条件：$S''(x_0) = M_0$，$S''(x_n) = M_n$，等于只有 n-1 个未知数，则直接用三弯矩方程

$$\begin{bmatrix} 2 & \lambda_1 & & & \\ \mu_2 & 2 & \lambda_2 & & \\ & \cdots & \cdots & \cdots & \\ & & \mu_{n-2} & 2 & \lambda_{n-2} \\ & & & \mu_{n-1} & 2 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \cdots \\ M_{n-2} \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} d_1 - \mu_1 M_0 \\ d_2 \\ \cdots \\ d_{n-2} \\ d_{n-1} - \lambda_{n-1} M_n \end{bmatrix} \tag{8}$$

过去题目代码：

```
function res = division(fun, precision, b, e)
   res = 0;
   max_iter = 2000;
   iter = 1;
   while(abs(res - (b + e) / 2) >= precision && iter <= max_iter)
      res = (b + e) / 2;
      if fun(b) * fun(res) <= 0
         e = res;
      else
         b = res;
      end
      iter = iter + 1;
   end
   res = (b + e) / 2;
end
```

```
function res = fixed_point(fun, precision, begin)
   res = eval(fun(begin));
   max_iter = 2000;
   iter = 1;
   while(abs(res - eval(fun(res))) >= precision && iter <= max_iter)
      res = eval(fun(res));
      iter = iter + 1;
   end
   res = eval(fun(res));
end
```

```
function res = newton(fun, precision, begin)
   syms x;
   diff_fun(x) = diff(fun(x));
   res = begin;
   iter = 1;
   max_iter = 2000;
   while(abs(res - (res - eval(fun(res)) ./ eval(diff_fun(res)))) >= ...
      precision && iter <= max_iter)
      res = res - eval(fun(res)) ./ eval(diff_fun(res));
      iter = iter + 1;
   end
   res = res - eval(fun(res)) ./ eval(diff_fun(res));
end
```

```
%% 用二分法、不动点迭代（与牛顿法不一样）、牛顿法求解以下非线性方程。
% （1）sin x = 6x + 5
% （2）lnx + x^2 = 3
```

```
% （3）e^x + x = 7
clear;clc;
% 终止条件为前后两次近似解之差小于10^?3
precision = 0.001;
% 声明自变量 x
syms x;
%% sin x = 6x + 5
disp('方程一：sin x = 6x + 5');

% 二分法
fun(x) = sin(x) - 6*x - 5;
division_res = division(fun, precision, -1, 0);
disp('二分法：');
disp('观察可知该方程在-1 和 0 间有解');
disp('求解结果：');
disp(division_res);

% 不动点法
fun(x) = (sin(x) - 5) / 6;
fixed_point_res = fixed_point(fun, precision, -1);
disp('不动点迭代法：');
disp('将方程变形为 x = (sin(x) - 5) / 6, 取 x = -1 作为初始迭代解');
disp('求解结果：');
disp(fixed_point_res);

% 牛顿法
fun(x) = sin(x) - 6*x - 5;
newton_res = newton(fun, precision, -1);
disp('牛顿法：');
disp('取 x = -1 作为初始迭代解');
disp('求解结果：');
disp(newton_res);

%% ln x + x^2 = 3
disp('方程二：lnx + x^2 = 3');

% 二分法
fun(x) = log(x) + x^2 - 3;
division_res = division(fun, precision, 1, 2);
disp('二分法：');
disp('观察可知该方程在 1 和 2 间有解');
disp('求解结果：');
disp(division_res);

% 不动点法
fun(x) = sqrt(3 - log(x));
fixed_point_res = fixed_point(fun, precision, 2);
disp('不动点迭代法：');
disp('将方程变形为 x = (3 - ln(x))^0.5，取 x = 2 作为初始迭代解');
disp('求解结果：');
disp(fixed_point_res);

% 牛顿法
fun(x) = log(x) + x^2 - 3;
newton_res = newton(fun, precision, 2);
disp('牛顿法：');
disp('取 x = 2 作为初始迭代解');
disp('求解结果：');
disp(newton_res);

%% e^x + x = 7
disp('方程三：e^x + x = 7');

% 二分法
fun(x) = exp(x) + x - 7;
division_res = division(fun, precision, 1, 2);
disp('二分法：');
disp('观察可知该方程在 1 和 2 间有解');
disp('求解结果：');
disp(division_res);

% 不动点法
fun(x) = log(7 - x);
fixed_point_res = fixed_point(fun, precision, 2);
disp('不动点迭代法：');
disp('将方程变形为 x = ln(7 - x)，取 x = 2 作为初始迭代解');
disp('求解结果：');
disp(fixed_point_res);

% 牛顿法
fun(x) = exp(x) + x - 7;
```

```
newton_res = newton(fun, precision, 2);
disp('牛顿法：');
disp('取 x = 2 作为初始迭代解');
disp('求解结果：');
disp(newton_res);
```

```
function res = gauss_elimination(cm, bm)
% input:
%  cm: 系数矩阵:n*n
%  bm：常数项矩阵:n*1
% output:
%  res：求解结果:n*1

  [rcm, ccm] = size(cm);
  [rbm, ~] = size(bm);
  res = zeros(rbm, 1);
  if (rcm ~= ccm) || (rcm ~= rbm)
    disp('输入矩阵格式错误');
  else
    for i = 1:rcm-1
      if cm(i, i) == 0
        disp('主对角线元素错误');
      else
        for j = i+1:rcm
          ratio = cm(j, i) / cm(i, i);
          for k = i+1:ccm
            cm(j, k) = cm(j, k) - ratio * cm(i, k);
          end
          bm(j) = bm(j) - ratio * bm(i);
          cm(j, 1) = 0;
        end
      end
    end
  end
  res(rcm) = bm(rcm) / cm(rcm, ccm);
  for i = rcm-1:-1:1
    tmp = 0;
    for j = i+1:ccm
      tmp = tmp + cm(i, j) * res(j);
    end
    res(i) = (bm(i) - tmp) / cm(i, i);
  end
end
```

```
function [convergence, res] = GS(cm, bm, precision)
  iter = 1;
  convergence = true;
  max_iter = 10000;
  [rbm, ~] = size(bm);
  % 求系数矩阵的对角矩阵
  cm_diag = diag(diag(cm));
  % 求系数矩阵的下三角矩阵
  low_diag = -tril(cm, -1);
  % 求系数矩阵的下三角矩阵
  up_diag = -triu(cm, 1);

  B = (cm_diag - low_diag) \ up_diag;
  % 计算谱半径
  R = max(abs(eig(B)));
  if R >= 1
    convergence = false;
    res = zeros(rbm, 1);
    return
  end
  f = (cm_diag - low_diag) \ bm;
  res = zeros(rbm, 1);
  while (norm(res - (B * res + f)) >= precision && iter <= max_iter)
    res = B * res + f;
    iter = iter + 1;
  end
end
```

```
function [convergence, res] = jacobi(cm, bm, precision)
  iter = 1;
  convergence = true;
  max_iter = 2000;
  [rbm, ~] = size(bm);
  % 求系数矩阵的对角矩阵
  cm_diag = diag(diag(cm));

  B = cm_diag \ (cm_diag - cm);
  % 计算谱半径
  R = max(abs(eig(B)));
  if R >= 1
    convergence = false;
    res = zeros(rbm, 1);
    return
  end
  f = cm_diag \ bm;
  res = zeros(rbm, 1);
  while (norm(res - (B * res + f)) >= precision && iter <= max_iter)
    res = B * res + f;
    iter = iter + 1;
  end
end
```

```
%% 用高斯消去法、Jacobi 迭代、G-S 迭代
求解以下线性方程组。
clear;clc;
precision = 0.001;
%% 第一问
% 2x - 2y - z = ?2
% 4x + y - 2z = 1
% -2x + y - z = ?3

disp('第一问：');

% 系数矩阵 cm
cm = [2, -2, -1;4, 1, -2;-2, 1, -1];
% 常数项矩阵
bm = [-2;1;-3];

% 高斯消去法
gauss_res = gauss_elimination(cm, bm);
disp('高斯消去法结果：');
disp(gauss_res);
% Jacobi 迭代
[convergence, jacobi_res] = jacobi(cm, bm,
precision);
disp('Jacobi 迭代结果：');
if convergence
    disp(jacobi_res);
else
    disp('谱半径不小于 1，迭代不收敛');
end
% G-S 迭代
```

```
[convergence, GS_res] = GS(cm, bm, precision);
disp('G-S 迭代结果：');
if convergence
    disp(GS_res);
else
    disp('谱半径不小于 1，迭代不收敛');
end
%% 第二问

disp('第二问：');

% 系数矩阵 cm
cm = zeros(100, 100);
for i = 1:100
    cm(i, i) = 3;
    if i < 100
        cm(i+1, i) = -1;
        cm(i, i+1) = -1;
    end
end
% 常数项矩阵
bm = ones(100, 1);
bm(1) = 2;
bm(100) = 2;

% 高斯消去法
gauss_res = gauss_elimination(cm, bm);
disp('高斯消去法结果：');
disp(gauss_res);
% Jacobi 迭代
[convergence, jacobi_res] = jacobi(cm, bm, precision);
disp('Jacobi 迭代结果：');
if convergence
    disp(jacobi_res);
else
    disp('谱半径不小于 1，迭代不收敛');
end
% G-S 迭代
[convergence, GS_res] = GS(cm, bm, precision);
disp('G-S 迭代结果：');
if convergence
    disp(GS_res);
else
    disp('谱半径不小于 1，迭代不收敛');
end
```