数值计算实践实验报告

2024 年 8 月 18 日

1　实验一：幂法及反幂法

幂法及反幂法是求解特征值问题的迭代法。工程实践中有多种振动问题，如桥梁或建筑物的振动，机械机件的振动，飞机机翼的颤动等，还有一些稳定性分析及相关性分析问题，都可以转化为求矩阵特征值与特征向量的问题。

这里我们将介绍反幂法的基本理论及算法实现（以 Matlab 为平台）。

反幂法是计算矩阵最小模的特征值和特征向量的有效方法，它的计算过程需要求解线性方程组，一般采用直接三角分解法，结合原点位移法，它可以修正某一指定的特征值及特征向量。

1.1　问题描述

设 $A \in \mathbb{R}^{n\times n}$ 为非奇异矩阵，A 的特征值次序记作 $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_{n-1}| \geq |\lambda_n|$，相应的特征向量特征值 $x_1, x_2, \ldots, x_n$,

求 $|\lambda_n|$ 及 x_n。

1.2　反幂法公式描述

计算 A 的模最小的特征值 λ_n 的问题就是计算 A^{-1} 的模最大的特征值问题

A^{-1} 的特征值为 $\left|\frac{1}{\lambda_n}\right| \geq \left|\frac{1}{\lambda_{n-1}}\right| \geq \ldots \geq \left|\frac{1}{\lambda_2}\right| \geq \left|\frac{1}{\lambda_1}\right|$, 对应的特征向量为 $x_n, x_{n-1}, \ldots, x_1$。求得矩阵 A^{-1} 的主特征值 $\frac{1}{\lambda_n}$，从而求得 A 的模最小的特征值 λ_n

反幂法迭代公式为任取初始向量 $v_0 = u_0 \neq 0$，构造向量序列

$$\begin{cases} v_k = A^{-1} \cdot u_{k-1} \\ u_k = \dfrac{v_k}{\max(v_k)} \end{cases} (k = 1, 2, \ldots) \tag{1}$$

.

迭代向量 v_k 可以通过解方程组 $A * v_k = u_{k-1}$ 求得

在反幂法中也可以用原点平移法来加速迭代过程或求其他特征值及特征向量

如果矩阵 $(A - pI)^{-1}$ 存在，显然其特征值为 $\frac{1}{\lambda_1 - p}, \frac{1}{\lambda_2 - p}, \ldots \frac{1}{\lambda_n - p}$，对应的特征向量仍然是 $x_1, x_2, \ldots, x_n$.

得到原点位移法反幂法的迭代公式

$$\begin{cases} v_k = (A - pI)^{-1} \cdot u_{k-1} \\ u_k = \dfrac{v_k}{\max(v_k)} \end{cases} (k = 1, 2, \ldots) \tag{2}$$

1.3 算法描述

图 1 描述了反幂法的计算过程。其中 err 表示相邻两次迭代结果的偏差，即

$$err = \left|u_0^{i+1} - u_0^i\right| \tag{3}$$

当偏差 err 小于精度 1e-8 时计算终止。

function [lambda, u0, iter, us, ms, err] = antipow_before(A)
%反幂法的原点位移法只需多输入一个变量p（p为定点）
%function [lambda, u0, iter, us, ms, err] = antipow(A, p)

%程序文件antipow_before.m
%本函数是使用反幂法的原点位移法求矩阵A的最小特征值和特征向量
%输入

```
%A：矩阵
%返回值
%lambda：矩阵特征值
%u0：特征向量
%iter：迭代次数
%us：存储特征向量的迭代序列
%ms：存储特征值的迭代序列
%err：存储误差的迭代序列
    sz = size(A);
    u0 = ones(sz(1), 1); % 初始特征向量
    maxv = max(abs(u0));
    i = 1;
    I = eye(sz); % 单位矩阵

    us = zeros(1000, sz(1)); % 存储特征向量的迭代序列
    ms = zeros(1000, 1); % 存储特征值的迭代序列
    err = zeros(1000, 1); % 存储误差的迭代序列

    u = u0 / maxv;
    while true
        v = A \ u; %原点位移法需将这行更改为v = (A - p * I) \ u;
        maxv = norm(v, inf);
        u0 = v / maxv;

        if norm(u - u0, inf) < 1e-8
            lambda = 1 / maxv; %原点位移法需将这行更改为lambda = p + 1 / maxv;

            us(i, :) = u0;
            ms(i) = lambda;
            err(i) = norm(u - u0, inf);
            break
```

```
        end

        lambda = 1 / maxv; %原点位移法需将这行更改为lambda = p + 1 / maxv;

        us(i, :) = u0;
        ms(i) = lambda;
        err(i) = norm(u − u0, inf);

        u = u0;
        i = i + 1;

        if i > 1000
            break
        end
end
        iter = i;
% 删除剩余的零行
        us = nonzeros(us);
        us = reshape(us, [], sz(1)); % 将非零元素重新形状为原始的行数
        ms = nonzeros(ms);
        ms = reshape(ms, [], 1); % 将非零元素重新形状为列向量形式
        err = err(1:iter);
end
```

1.4 实例分析

1.4.1 例：用反幂法计算特征值

例: 用反幂法求下列矩阵的接近于 p=1.2679 的特征值 (精确特征值入3=3 -√3) 及其特征向量, $A = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 4 \end{bmatrix}$。

解：依题，取 $A=\begin{bmatrix}2&1&0\\1&3&1\\0&1&4\end{bmatrix}$

按式（1）用反幂法计算到第 23 次时，结果收敛（见表 1）。此时最小特征值 λ 为 1.267949197，特征向量为 $(1,-0.732050803,0.267949189)'$。相邻两次迭代偏差 $\mathrm{err}=5.91046723\times10^{-9}$。

表 1: **反幂法结果表**

迭代次数	特征值	特征向量			相邻两次迭代偏差
1	2.25	1	0.25	0.5	7.50000000E-01
5	1.295681063	1	-0.704318937	0.255813953	4.27804753E-02
9	1.268718914	1	-0.731281086	0.267428418	1.08401052E-03
13	1.267973105	1	-0.732026895	0.267931908	3.28371282E-05
17	1.267949952	1	-0.732050048	0.267948637	1.03857596E-06
21	1.267949217	1	-0.732050783	0.267949175	3.30927650E-08
22	1.267949203	1	-0.732050797	0.267949185	1.39852206E-08
23	1.267949197	1	-0.732050803	0.267949189	5.91046723E-09

用反幂法原点位移法的计算到第 3 步时，结果收敛（见表 2）。此时最小特征值值 λ 为 1.267949192，特征向量为 $(1\ -0.732050808\ 0.267949192)'$。相邻两次迭代偏差 $\mathrm{err}=2.09554207\times10^{-9}$。

表 2: **反幂法原点位移法结果表**

迭代次数	特征值	特征向量			相邻两次迭代偏差
1	1.268047571	1	-0.731952429	0.267962373	1.73195243
2	1.267949195	1	-0.732050805	0.267949192	9.83764800E-05
3	1.267949192	1	-0.732050808	0.267949192	2.09554207E-09

结果相比较可以看出，反幂法原点位移法的效果比反幂法要好。调用函数为 [lembda,u0,iter,us,ms,err] = `antipow`(A,p) 和 [lembda,u0,iter,us,ms,err] = `antipow_before`(A)。

在 MATLAB 中运行文件 test_antipow1.

```
% 反幂法原点位移法程序文件为test_antipow1.mlx
clear
close all
clc
format long
A=[2,1,0;1,3,1;0,1,4];
p = 1.2679
[lembda,u0,iter,us,ms,err] = antipow(A,p);
i=1:iter
disp(u0)
disp(lembda)
disp(iter)
T = table(i', ms, us,err, 'VariableNames', {'迭代次数', '特征值', '特征向量','偏差'});
disp(T);
```

```
% 反幂法程序文件为test_antipow_before1.mlx
clear
close all
clc
format long
A=[2,1,0;1,3,1;0,1,4];
p = 1.2679
[lembda,u0,iter,us,ms,err] = antipow_before(A);
i=1:iter
disp(u0)
disp(lembda)
disp(iter)
T = table(i', ms, us,err, 'VariableNames', {'迭代次数', '特征值', '特征向量','偏差'});
```

disp(T);

1.5 进一步分析

1.5.1 例：用反幂法计算最小特征值

例: 求下列矩阵的最小特征值及其特征向量。$A = \begin{bmatrix} 5 & 2 \\ 2 & 3 \end{bmatrix}$。

解：依题，取 $A = \begin{bmatrix} 5 & 2 \\ 2 & 3 \end{bmatrix}$。

用反幂法计算到第 17 步时，结果收敛（见表 3 ）。此时特征值 λ 为 1.76393202806178 相邻两次迭代偏差 err $= 7.05018476931940 \times 10^{-9}$ 。

表 3: **反幂法结果表**

迭代次数	特征值	特征向量		相邻两次迭代偏差
1	3.666666667	0.333333333	1	6.66666667E-01
4	1.837647059	-0.581176471	1	8.82187241E-02
7	1.765627619	-0.61718619	1	2.14656384E-03
10	1.763970411	-0.618014795	1	4.86616205E-05
13	1.763932891	-0.618033554	1	1.10132654E-06
16	1.763932042	-0.618033979	1	2.49246741E-08
17	1.763932028	-0.618033986	1	7.05018477E-09

由于矩阵定点未知，原点位移法中设置已知特征值为 0，这种情况下原点位移法的计算次数、计算结果与反幂法相同。

由公式（1）与（2）可得，当不知道所求矩阵的定点时，即定点设置为 0 时，二者迭代公式相同，故结果相同。

若将 1.4.1 例题中已知条件“p=1.2679”改为“p=4.7320”（矩阵的另一个特征值），则反幂法仍可求出矩阵的最小特征值（结果见表 3），而原点位移法求得的特征值则为定点精度更高的特征值 λ4.732050808，相邻两次迭代偏差 err $= 3.45876106 \times 10^{-10}$。

表 4: 原点位移法结果表

迭代次数	特征值	特征向量			相邻两次迭代偏差
1	4.732040841	0.267935578	0.732040841	1	7.32064422E-01
2	4.732050808	0.267949193	0.732050808	1	1.36144461E-05
3	4.732050808	0.267949192	0.732050808	1	3.45876106E-10

1.6 本实验心得体会

反幂法不需要已知矩阵的特征值即可求出矩阵的最小特征值，而原点位移法则需已知矩阵其中一个特征值，并根据已知特征值得到对应的特征值，而非最小特征值。

2 实验二：线性方程组的求解

解线性代数方程组是科学研究和工程计算中一个非常重要的问题。无论是在数学理论上的插值公式（包括样条插值）、求积公式的建立、常微分方程的差分格式构造，还是在工程科学中的诸如电路分析、分子构造、大地测量等方面，以及在一些经济学科和其他社会科学学科中的数量研究中，经常会遇到求解线性方程组。线性方程组的解法大致分为直接法与迭代法两大类。每一类方法中都有很多经典的方法，各有特色。这里我们将介绍 Cholesky 分解法的基本理论及算法实现（以 Matlab 为平台）。Cholesky 分解法的一个突出优点是针对大型矩阵具有更高的计算效率。但是此法也有缺点，它要求方程组的系数矩阵具有某种特殊性质（比如是正定矩阵），且缺乏通用性：Cholesky 分解仅提供一个分解结果，无法在多个线性方程组之间重复使用。对于不同的方程组，需要重新进行 Cholesky 分解。

2.1 问题描述

设 $A \in \mathbb{R}^{n\times n}$，$b \in \mathbb{R}^n$，A 非奇异，$x \in \mathbb{R}^n$ 满足

$$Ax = b \tag{2.1}$$

如果能找到一个主对角线元素都取正值的下三角阵 L，使得

$$A = LL^T \tag{2.2}$$

则此时线性方程组 $Ax = b$ 可以写成 $LL^Tx = b$ 从而推出

$$\begin{cases} Ly = b \\ L^Tx = y \end{cases} \tag{2.3}$$

现在问题转化为如何求出一个主对角线元素都取正值的下三角阵 L。

2.2 Cholesky 分解法公式描述

设 L 的元素为 l_{ij}，$i \geq j$，则

$$A = \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ & l_{22} & \cdots & l_{n2} \\ & & \ddots & \vdots \\ & & & l_{nn} \end{pmatrix} \tag{4}$$

显然

$$\begin{cases} l_{11} = \sqrt{a_{11}} \\ l_{i1} = \dfrac{a_{i1}}{l_{11}} \qquad (i = 2, 3, \ldots) \end{cases} \tag{2.4}$$

假设 L 的第 k-1 列元素已经求得，下面求 L 第 k 列元素 $l_{ik}, (i = k, k+1, \ldots, n)$，注意到

$$a_{ik} = \sum_{j=1}^{k-1} l_{ij} \cdot l_{kj} + l_{ik} \cdot l_{kk} \tag{2.5}$$

得

$$l_{kk} = \left(a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2 \right)^{\frac{1}{2}} \tag{2.6}$$

$$l_{ik} = \frac{(a_{ik} - \sum_{j=1}^{k-1} l_{ij} \cdot l_{kj})}{l_{kk}} \quad (i = k+1, \ldots, n) \tag{2.7}$$

2.3 算法描述

相应的Cholesky算法函数[L, x] = Cholesky_sll(A, b)源代码如下：

```
{
    function [L, x] = Cholesky_sll(A, b)
    %程序文件 Cholesky_sll.m
    % 本函数是用Cholesky分解法解线性方程组
    %          Ax = b
    % 输入
    % A:  方程组系数矩阵
    % b:  方程组右端向量
    % 返回值
    % L:  主对角线元素都取正值的下三角阵
    % x:  方程组的解
    sz = size(A);
    L = zeros(sz);

    L(1,1) = sqrt(A(1,1));
    L(2:sz, 1) = A(2:sz,1) / L(1,1);

    for j = 2:sz
        for i = j:sz
            L(j,j) = sqrt(A(j,j) - sum(L(j,1:j-1).^2));
            L(i, j) = (A(i,j) - sum(L(i,1:j-1).*L(j,1:j-1))) / L(j,j);
        end
    end

    y = L \ b;
    x = L' \ y;
}
```

2.4 实例分析

2.4.1 例：Cholesky 解方程组

随机生成一个较大的正定矩阵 A 和一个对应的列向量 b，用 Cholesky 分解法解决。

解：

分别用 Cholesky 分解法和 LU 分解法计算方程组，并比较运行时间。

Cholesky 运行时间	LU 运行时间
0.000225	0.0004129
0.000174	0.0002229
0.000189	0.0002112
0.000101	0.0001922
0.000313	0.0005813
9.26E-05	0.0002348
6.35E-05	0.0001419
4.72E-05	0.0001292
4.39E-05	0.0001257
4.45E-05	0.000128

Cholesky 平均运行时间	**LU 平均运行时间**
1.29E-04	2.38E-04

结果相比较可以看出，求解较大的正定矩阵方程组时，Cholesky 分解法运行效率比 LU 分解法运行效率高。实现调用函数为在 MATLAB 中运行文件 test_cholesky1.

```
% 程序文件为test_cholesky1.mlx
N = 10;
M = diag(rand(N,1));
Z = orth(rand(N,N));
A = Z' * M * Z;    % A为N阶正定矩阵
```

```
b = rand(N,1);

time_cho = zeros(1, N);
time_LU = zeros(1, N);

for i = 1:N
tic;
[L, x1] = Cholesky_sll(A, b);
time_cho(i) = toc;

tic;
[L, U, x2] = lusll_0614(A, b);
time_LU(i) = toc;
end
format short

avg_time_cho = mean(time_cho);
avg_time_LU = mean(time_LU);

tb = array2table([time_cho', time_LU'], 'VariableNames', {'Cholesky运行时间', 'L
avg_tb = array2table([avg_time_cho, avg_time_LU], 'VariableNames', {'Cholesky
```

2.5 进一步分析

从结果我们可以看出求解较大的正定矩阵方程组时，Cholesky 分解法运行效率比 LU 分解法运行效率高。但是，这并不意味着 LU 分解法完全比 Cholesky 分解法差。下面通过一个例子进一步分析，请读者体会两种解法的使用技巧和适用范围。

2.5.1 例：Cholesky 解方程组

$$\begin{pmatrix} 4 & 1 & -2 & 3 \\ 1 & 2 & 0 & 1 \\ -2 & 0 & 3 & -2 \\ 3 & 1 & -2 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \tag{5}$$

解：依题，取

$$A = \begin{pmatrix} 4 & 1 & -2 & 3 \\ 1 & 2 & 0 & 1 \\ -2 & 0 & 3 & -2 \\ 3 & 1 & -2 & 0 \end{pmatrix} b = \begin{pmatrix} 1 \\ 2 \\ -1 \\ 3 \end{pmatrix}$$

用 Cholesky 分解法计算出方程组的解 x_1 为

$$x_1 = \begin{bmatrix} -0.6779 \\ 0.9952 \\ -0.3269 \\ 0.6875 \end{bmatrix}$$

用 LU 分解法计算出方程组的解 x_2 为

$$x_2 = \begin{bmatrix} 0.0625 \\ 1.13125 \\ -0.7500 \\ -0.6875 \end{bmatrix}$$

将计算结果乘上 A，与 b 进行比较发现，用 Cholesky 分解法算出的 b 为 $\begin{pmatrix} 1 \\ 2 \\ -1 \\ -0.3846 \end{pmatrix}$，与原题中给出的不同。实现调用函数为在 `MATLAB` 中运行文件 `test_LUsll`.

%程序文件为test_LUsll.mlx

```
A = [4, 1, −2, 3;
       1, 2, 0, 1;
      −2, 0, 3, −2;
       3, 1, −2, 0];
b = [1; 2; −1; 3];

[L, x1] = Cholesky_sll(A, b);
[L,U,x2] = lusll_0614(A,b);
disp(x1);
disp(x2);
b_cho = A ∗ x1
b_lu = A ∗ x2
```

对于非正定矩阵，其特征值可能存在非正值或者零，这会导致 Cholesky 分解过程中出现负数或者零的平方根，从而无法进行分解。

2.6 本实验心得体会

LU 分解适用于一般的方阵，包括非对称矩阵和非正定矩阵, 但计算量比 Cholesky 分解大，因此不适用于大型矩阵计算。而 Cholesky 分解利用了正定矩阵的特性，可以提供更快速和稳定的计算结果，但仅局限于计算正定矩阵。

3 实验三：非线性方程的求解

非线性科学是当今科学发展的一个重要研究方向, 其中非线性方程的求根是一个不可或缺的研究内容。非线性方程的解法主要为迭代法。这里我们将介绍牛顿迭代法的基本理论及算法实现（以 Matlab 为平台）。牛顿迭代法的突出优点为迭代速度快，但此方法也有缺点，它对初始点的选择相对敏感，并且可能收敛到不同的解。

3.1 问题描述

设 $f(x)$ 为连续的非线性函数，满足 $f(x)=0$，若存在 x* 使得 $f(x*)=0$，则 x* 为方程的根。

3.2 牛顿迭代法公式描述

设 $f(x)=0$ 有近似根 x_k，将函数 $f(x)$ 在点 x_k 处一阶泰勒展开有

$$f(x)=f(x_k)+f'(x_k)\cdot(x-x_k)$$

于是，方程 $f(x)=0$ 就可近似地表示为

$$f(x_k)+f'(x_k)\cdot(x-x_k)=0$$

记该方程的根为 x_{k+1}，则 x_{k+1} 的计算公式为

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

相应的牛顿迭代法算法函数`[x, iter,err] = NNnewton_0608(F, x0)`源代码如下：

```
function [x, iter,err] = NNnewton_0608(F, x0)

switch class(F)
case 'sym'
syms x;
dF = diff(F, x);
F = matlabFunction(F);
dF = matlabFunction(dF);
case 'function_handle'
syms x;
F_sym = F(x);
dF_sym = diff(F_sym, x);
F = matlabFunction(F_sym);
dF = matlabFunction(dF_sym);
```

```
otherwise
error('第一个参数必须为符号函数或函数句柄！');
end

xk = x0;
iter = 0;
for i = 1:100
xk_1 = xk − dF(xk) \ F(xk);

iter = iter + 1;
err(iter) = xk_1 −xk;
if abs(xk_1 − xk)<1e−10
break
end

xk = xk_1;
end
x = xk;
```

3.3 实例分析

3.3.1 例：牛顿迭代法解非线性方程

利用牛顿迭代法求解函数 $f(x) = e^{-x} - x$ 在 0.1 附近的零点。

解：依题，取 $f(x) = e^{-x} - x$，用牛顿迭代法的计算到第 5 步时，结果收敛。此时 x=0.567143290409781。相邻两次迭代偏差 err $= 1.11022302462516 \times 10^{-16}$.

用不动点迭代法的计算到第 42 步时，结果收敛。此时 x $=$ 0.567143290452107。相邻两次迭代偏差 err $= -6.632616678103886 \times 10^{-11}$.

表 5: 牛顿迭代法结果表

xk	相邻两次迭代偏差
0.100000000000000	0.422522893773166
0.522522893773166	0.0442552640130278
0.566778157786194	0.000365108496569566
0.567143266282763	2.41270203815702e-08
0.567143290409784	1.11022302462516e-16

表 6: 不动点迭代法结果表

迭代次数	xk	相邻两次迭代偏差
1	0.10000000	8.04837E-01
5	0.5131235920	8.54992E-02
10	0.5703707028	-5.05487E-03
15	0.5669541237	2.96461E-04
20	0.5671543908	-1.73958E-05
25	0.5671426391	1.02072E-06
30	0.5671433286	-5.98924E-08
35	0.5671432882	3.51427E-09
40	0.5671432905	-2.06205E-10
42	0.5671432905	-6.63262E-11

在 MATLAB 中运行文件 `test_NNnewton_0608`.

```
clear
format long
f =@(x) x − exp(−x) ;   %test_fixpt1中改为f =@(x)  exp(−x) ;
x0 = 0.1;
[x, iter,err,xk_history] = NNnewton_0608(f, x0);

%test_fixpt1中改为 [x, iter,err,xk_history] = fixpt(f, x0);

i = 1:iter
```

T = table(i',xk_history', err', 'VariableNames', {'迭代次数','xk','相邻两次迭代偏差'});
disp(T);

3.4 进一步分析

从结果我们可以看出求解非线性方程时，牛顿迭代法运行效率比不动点迭代法运行效率高。但是，这并不意味着不动点迭代法完全比牛顿迭代法差。下面通过一个例子进一步分析，请读者体会两种解法的使用技巧和适用范围。

3.4.1 例：分别用牛顿迭代法和不动点迭代法解方程

分别用牛顿迭代法和不动点迭代法解方程 $f(x) = x^3 - 2x - 5 = 0$
解：依题取 $f(x) = x^3 - 2x - 5 = 0$，$x_0 = 0$
发现牛顿迭代法结果不收敛（这里表格只取前 5 次）

迭代次数	xk	相邻两次偏差
1	0	-2.50000000000000
2	-2.50000000000000	0.932835820895523
3	-1.56716417910448	1.06457173401780
4	-0.502592445086680	-3.31811402261265
5	-3.82070646769933	1.27131307633872

在 MATLAB 中运行文件 test_newton2.

clear
format long
f(x) =@(x) x^3 − 2∗x − 5 = 0 ; *%test_fixpt1中改为f = @(x) (x^3 − 5) / 2 ;*
x0 = 0.1;
[x, iter,err,xk_history] = NNnewton_0608(f, x0);

%test_fixpt1中改为 [x, iter,err,xk_history] = fixpt(f, x0);

```
i = 1:iter
T = table(i',xk_history', err', 'VariableNames', {'迭代次数','xk','相邻两次迭代偏差'});
disp(T);
```

牛顿迭代法不收敛原因为牛顿迭代法依赖于 x_0 的选取，如果初始点 x_0 选取得不够接近解 x*，那么牛顿法的局部线性近似就会变得不准确，导致迭代点的更新方向和步长不正确。这可能会导致迭代发散，即迭代点越来越远离解，无法收敛到方程的解。
因此，牛顿法的收敛性依赖于初始点 x_0 的选取，需要选择一个足够接近解的初始点，以保证牛顿法的局部线性近似有效，并使迭代能够逐渐接近方程的解。如果选择的初始点与解之间的距离过大，牛顿法可能会出现发散的情况。

3.5 本实验心得体会

牛顿法收敛速度快，通常具有二次收敛性，因此在接近解时收敛迅速。但比较依赖于 x_0 的取值，牛顿法可能会收敛到局部最小值而不是全局最小值。此外，该方法需要计算函数的一阶和二阶导数，对于一些复杂的函数可能比较困难。
不动点迭代法：相对于牛顿法，不动点迭代法的实现相对简单，不需要计算导数，仅需对方程进行简单的变形即可。但是收敛速度较慢，通常具有线性收敛性，因此需要更多的迭代步骤才能达到一定的精度。此外，对于某些问题，不动点迭代法可能无法收敛。