**矩阵的LU分解分析**

矩阵的LU分解是求解线性方程组一种重要的工具，此实践的目的是了解矩阵的LU分解。

LU分解的介绍：

1 给定矩阵为可逆矩阵，LU分解就是将矩阵A分解为一个下三角矩阵L和一个上三角矩阵U的乘积，即，其中矩阵L的对角元全为1.

矩阵的LU分解的方法一般都采用高斯消去的过程来实现

记

**Step 1 假设 记**

**Step 2 假设 记**

**Step k 假设 记**

**Step n-1 假设 记**

矩阵L的存储：由于最终生成的矩阵U的下三角部分都为0，而L的对角元都为1，不需要存储，故可以考虑将L的元素全部存储到U的下三角部分。

由上述过程编写矩阵的LU分解的代码，要求输入参数为矩阵A，输出参数为矩阵L和U

分析LU分解的过程，在算法实现的过程中，导致中断的情况是什么？如果中断，输出相应应的错误信息。

编写程序利用矩阵的LU分解实现求解线性方程组Ax=b;要求输出参数为A,b；输出参数为解向量x；构造数据测试程序的有效性；可以比较x和 A\b 比较【30分】

**代码实现：**

**function main()**

**% 矩阵A和向量b**

**A = [1, 2, -1; 2, 1, -2; -3, 1, 1;];**

**b = [3; 3; -6];**

**try**

**% 进行LU分解**

**[L, U] = LU\_factorization(A);**

**% 显示L和U矩阵**

**disp('矩阵L:');**

**disp(L);**

**disp('矩阵U:');**

**disp(U);**

**% 使用LU分解求解线性方程组**

**x = back\_substitution\_two(L, U, b);**

**% MATLAB直接求解**

**x\_direct = A \ b;**

**% 输出结果**

**disp('输入矩阵A:');**

**disp(A);**

**disp('输入向量b:');**

**disp(b);**

**disp('LU分解求解结果:');**

**disp(x);**

**disp('MATLAB直接求解结果:');**

**disp(x\_direct);**

**% 比较结果**

**disp('解向量x与MATLAB直接求解结果的差异:');**

**disp(x - x\_direct);**

**catch ME**

**% 输出错误信息**

**fprintf('错误: %s\n', ME.message);**

**end**

**end**

**function [L, U] = LU\_factorization(A)**

**% 检查A是否是方阵**

**[m, n] = size(A);**

**if m ~= n**

**error('矩阵不是方阵');**

**end**

**% 初始化L和U**

**L = eye(n);**

**U = zeros(n);**

**for i = 1:n**

**% 检查主元是否为0**

**if A(i, i) == 0**

**error('主元为0，矩阵不可逆');**

**end**

**% 计算U的上三角部分**

**for j = i:n**

**U(i, j) = A(i, j) - L(i, 1:i-1) \* U(1:i-1, j);**

**end**

**% 计算L的下三角部分**

**for j = i+1:n**

**L(j, i) = (A(j, i) - L(j, 1:i-1) \* U(1:i-1, i)) / U(i, i);**

**end**

**end**

**end**

**function [X] = back\_substitution\_two(L, U, b)**

**% Ly = b, Ux = y**

**y = forward\_substitution(L, b);**

**X = back\_substitution(U, y);**

**end**

**function y = forward\_substitution(L, b)**

**% 前向替换解决Ly = b**

**n = length(b);**

**y = zeros(n, 1);**

**for i = 1:n**

**y(i) = (b(i) - L(i, 1:i-1) \* y(1:i-1)) / L(i, i);**

**end**

**end**

**function x = back\_substitution(U, y)**

**% 回代解决Ux = y**

**n = length(y);**

**x = zeros(n, 1);**

**for i = n:-1:1**

**x(i) = (y(i) - U(i, i+1:end) \* x(i+1:end)) / U(i, i);**

**end**

**end**

**main();**

**LU 分解的中断情况**

在实现 LU 分解的过程中，可能会遇到以下中断情况：

1. **矩阵不是方阵**：如果输入的矩阵 AAA 不是方阵，算法会立即中断，并输出错误信息 "矩阵不是方阵"。
2. **主元为 0**：在进行 LU 分解时，如果发现主元为 0，则说明矩阵是奇异的，不可逆，算法会中断并输出错误信息 "主元为0，矩阵不可逆"。

2 由于上述的算法过程可能出现中断，故考虑选主元的的LU分解

选主元的策略常见的有列选主元和全选主元。

列选主元： 把高斯过程中的 **称为主元，由于主元在高斯消去的过程中需要做分母，所以主元不能等于0，从数值稳定的角度出发，主元也不能太小，为了避免主元过小，列选主元的策略就是从选择绝对值最大的元素为新的主元，利用交换行的方式将之换到位置。然后继续消元的过程。**

**全选主元的策略为 假设经过k-1步高斯消去的过程后，从子矩阵**

选择绝对值最大的为新的主元，采用交换行和列的方式来达到此目的。

请根据上述的描述过程，查阅相关文献，编写列选主元的LU分解程序和全选主元的LU分解程序：

#### 实现部分选主元的 LU 分解：

#### PLU Factorization 函数的实现:

#### 首先获取矩阵 AA 的大小 nn。第一次行交换：找出矩阵 AA 的第一列中最大元素的位置，并创建一个单位矩阵 PP，然后交换矩阵 AA 的第一行和包含最大元素的行。接着，将第一列进行归一化处理。

#### 循环进行LU分解和行交换:对每一列，首先找到从当前行到最后一行中的最大元素的位置，并创建一个初等矩阵记录这次交换操作。然后，交换矩阵 AA 的当前行和包含最大元素的行，同时更新置换矩阵 PP。

#### 更新矩阵的第 rr 层：在找到当前列的主元并进行行交换后，更新矩阵 AA 的当前行元素。接着，对当前列以下的每一行，更新其值使其满足LU分解的条件。

#### 提取下三角矩阵 LL 和上三角矩阵 UU：最后，从更新后的矩阵 AA 中提取下三角矩阵 LL 和上三角矩阵 UU。其中 LL 包含 AA 的下三角部分加上单位矩阵，UU 则包含 AA 的上三角部分。

#### back\_substitution\_two 函数的实现

#### 前向替换解决 Ly=bLy=b：首先初始化一个向量 yy，然后使用前向替换法逐步求解 yy。前向替换法的过程是从矩阵 LL 的第一行开始，通过已知的 yy 值计算每一个 yy 元素。

#### 回代解决 Ux=yUx=y：在得到 yy 之后，初始化解向量 xx，并使用回代法求解 xx。回代法的过程是从矩阵 UU 的最后一行开始，通过已知的 xx 值逐步求解每一个 xx 元素。

#### function [L, U, P] = PLU\_factorization(A)

#### % PA = LU分解

#### % Input: A

#### % Output: L, U, P

#### % Version: 1.0

#### % last modified: 09/27/2023

#### n = length(A);

#### % 第一次行交换

#### [~, s] = max(A(1:n, 1)); % s 表示第一列最大元素的位置

#### P = eye(n);

#### P([1, s], :) = P([s, 1], :);

#### A = P \* A; % 用初等矩阵左乘A 对 A 作行交换

#### A([2:n], 1) = A([2:n], 1) \* (1 / A(1, 1)); % 求第一层

#### 

#### for r = 2:1:n

#### % 先有行交换

#### p = eye(n); % 用 p 记录每一次的初等矩阵

#### [~, s] = max(A(r:n, r));

#### s = s + r - 1;

#### p([r, s], :) = p([s, r], :);

#### A = p \* A; % A的改变

#### P = p \* P; % 记录P的变化

#### 

#### % 求第 r 层

#### for k = r:1:n

#### A(r, k) = A(r, k) - A(r, [1:r-1]) \* A([1:r-1], k);

#### end

#### for m = r + 1:1:n

#### A(m, r) = (A(m, r) - A(m, [1:r-1]) \* A([1:r-1], r)) \* (1 / A(r, r));

#### end

#### end

#### 

#### L = tril(A, -1) + eye(n);

#### U = triu(A, 0);

#### end

#### function x = back\_substitution\_two(L, U, b)

#### % Ly = b, Ux = y

#### % b : 列向量

#### % x : 解向量

#### % 前向替换解决Ly = b

#### n = length(b);

#### y = zeros(n, 1);

#### for i = 1:n

#### y(i) = (b(i) - L(i, 1:i-1) \* y(1:i-1)) / L(i, i);

#### end

#### % 回代解决Ux = y

#### x = zeros(n, 1);

#### for i = n:-1:1

#### x(i) = (y(i) - U(i, i+1:end) \* x(i+1:end)) / U(i, i);

#### end

#### end

#### % 测试代码

#### clc; clear all;

#### A = [1 2 -1; 2 1 -2; -3 1 1];

#### b1 = [3 3 -6]';

#### [L, U, P] = PLU\_factorization(A);

#### disp('矩阵L:');

#### disp(L);

#### disp('矩阵U:');

#### disp(U);

#### disp('置换矩阵P:');

#### disp(P);

#### X1 = back\_substitution\_two(L, U, P \* b1);

#### disp('LU分解求解结果:');

#### disp(X1);

#### % MATLAB直接求解结果验证

#### X\_direct = A \ b1;

#### disp('MATLAB直接求解结果:');

#### disp(X\_direct);

给定一个矩阵 AAA，LU 分解将其分解为一个下三角矩阵 LLL 和一个上三角矩阵 UUU 的乘积，即 A=LUA = LUA=LU。在这个过程中，部分选主元和全选主元策略可以显著提高数值稳定性。部分选主元是指在每一列中选择绝对值最大的元素作为主元，全选主元是指在整个子矩阵中选择绝对值最大的元素作为主元。

刻画LU分解是否稳定的一个参考因素为考虑其增长因子的大小，其中增长因子的定义为

即LU分解后U的元素绝对值的最大值与A的元素绝对值的最大值的比值，增长因子越小，相应的算法越稳定；试通过增长因子的大小比较LU分解，列选主元的LU分解，全选主元的LU分解的好坏。

**PLU\_factorization函数**代码首先通过部分选主元方法对矩阵A进行LU分解。使用初等行交换矩阵P来记录行交换操作，并保持A的变化。在每一列中选择绝对值最大的元素作为主元进行行交换。通过前向消去和后向代入来完成LU分解。计算了增长因子，即U的元素绝对值的最大值与A的元素绝对值的最大值的比值，用于评估分解的稳定性。

**back\_substitution\_two函数**:实现了前向替换和回代，解决Ly=b和Ux=y的问题，用于求解线性方程组。

**测试代码**:给定矩阵A和向量b，调用PLU\_factorization函数进行LU分解，并输出L、U和P矩阵及增长因子。使用back\_substitution\_two函数求解线性方程组Ax=b。使用MATLAB自带的直接求解函数（\运算符）进行对比，验证LU分解求解的正确性。

function [L, U, P, growth\_factor] = PLU\_factorization(A)

% PA = LU分解

% Input: A

% Output: L, U, P, growth\_factor

% Version: 1.1

% last modified: 07/04/2024

n = length(A);

max\_A = max(abs(A(:))); % 计算 A 的元素绝对值的最大值

% 第一次行交换

[~, s] = max(A(1:n, 1)); % s 表示第一列最大元素的位置

P = eye(n);

P([1, s], :) = P([s, 1], :);

A = P \* A; % 用初等矩阵左乘A 对 A 作行交换

A(2:n, 1) = A([2:n], 1) \* (1 / A(1, 1)); % 求第一层

for r = 2:1:n

% 先有行交换

p = eye(n); % 用 p 记录每一次的初等矩阵

[~, s] = max(A(r:n, r));

s = s + r - 1;

p([r, s], :) = p([s, r], :);

A = p \* A; % A的改变

P = p \* P; % 记录P的变化

% 求第 r 层

for k = r:1:n

A(r, k) = A(r, k) - A(r, [1:r-1]) \* A([1:r-1], k);

end

for m = r + 1:1:n

A(m, r) = (A(m, r) - A(m, [1:r-1]) \* A([1:r-1], r)) \* (1 / A(r, r));

end

end

L = tril(A, -1) + eye(n);

U = triu(A, 0);

max\_U = max(abs(U(:))); % 计算 U 的元素绝对值的最大值

growth\_factor = max\_U / max\_A; % 计算增长因子

end

function x = back\_substitution\_two(L, U, b)

% Ly = b, Ux = y

% b : 列向量

% x : 解向量

% 前向替换解决Ly = b

n = length(b);

y = zeros(n, 1);

for i = 1:n

y(i) = (b(i) - L(i, 1:i-1) \* y(1:i-1)) / L(i, i);

end

% 回代解决Ux = y

x = zeros(n, 1);

for i = n:-1:1

x(i) = (y(i) - U(i, i+1:end) \* x(i+1:end)) / U(i, i);

end

end

% 测试代码

clc; clear all;

A = [1 2 -1; 2 1 -2; -3 1 1];

b1 = [3 3 -6]';

[L, U, P, growth\_factor] = PLU\_factorization(A);

disp('矩阵L:');

disp(L);

disp('矩阵U:');

disp(U);

disp('置换矩阵P:');

disp(P);

disp('列选主元增长因子:');

disp(growth\_factor);

X1 = back\_substitution\_two(L, U, P \* b1);

disp('LU分解求解结果:');

disp(X1);

% MATLAB直接求解结果验证

X\_direct = A \ b1;

disp('MATLAB直接求解结果:');

disp(X\_direct);

列选主元要求输入为矩阵A，输出为矩阵L和U以及置换矩阵P，其中P记录了交换行的信息。

全选主元列选主元要求输入为矩阵A，输出为矩阵L和U以及置换矩阵P和Q，其中P记录了交换行的信息。Q记录了交换列的信息。

编写代码利用上述两个分解求解线性方程组Ax=b; 构造测试数据测试你的代码的有效性。

**full\_pivot\_LU函数**:代码通过全选主元方法对矩阵A进行LU分解，使用初等行交换矩阵P记录行交换，使用初等列交换矩阵Q记录列交换。选择子矩阵中绝对值最大的元素作为主元，并相应地交换行和列。进行消元操作以构造L和U矩阵返回L、U、P和Q矩阵。

**back\_substitution\_two函数**:实现了前向替换和回代，解决Ly=b和Ux=y的问题，用于求解线性方程组。

**测试代码**:给定矩阵A和向量b，调用full\_pivot\_LU函数进行LU分解，并输出L、U、P和Q矩阵。使用back\_substitution\_two函数求解线性方程组Ax=b,使用MATLAB自带的直接求解函数（\运算符）进行对比，验证LU分解求解的正确性。计算增长因子，即U的元素绝对值的最大值与A的元素绝对值的最大值的比值，用于评估分解的稳定性。

function [L, U, P, Q] = full\_pivot\_LU(A)

% 全选主元 LU分解

% Input: 矩阵 A

% Output: L, U, P, Q

% L: 下三角矩阵

% U: 上三角矩阵

% P: 行交换矩阵

% Q: 列交换矩阵

[n, n2] = size(A);

if n ~= n2

error('矩阵必须是方阵');

end

P = eye(n);

Q = eye(n);

for k = 1:n-1

% 选择主元

[~, idx] = max(abs(A(k:n, k:n)), [], 'all', 'linear');

[i\_max, j\_max] = ind2sub([n-k+1, n-k+1], idx);

i\_max = i\_max + k - 1;

j\_max = j\_max + k - 1;

% 交换行

if i\_max ~= k

A([k, i\_max], :) = A([i\_max, k], :);

P([k, i\_max], :) = P([i\_max, k], :);

end

% 交换列

if j\_max ~= k

A(:, [k, j\_max]) = A(:, [j\_max, k]);

Q(:, [k, j\_max]) = Q(:, [j\_max, k]);

end

% 消元

if A(k, k) == 0

error('矩阵是奇异的');

end

for i = k+1:n

A(i, k) = A(i, k) / A(k, k);

A(i, k+1:n) = A(i, k+1:n) - A(i, k) \* A(k, k+1:n);

end

end

L = tril(A, -1) + eye(n);

U = triu(A);

end

function x = back\_substitution\_two(L, U, b)

% Ly = b, Ux = y

% b : 列向量

% x : 解向量

% 前向替换解决Ly = b

n = length(b);

y = zeros(n, 1);

for i = 1:n

y(i) = (b(i) - L(i, 1:i-1) \* y(1:i-1)) / L(i, i);

end

% 回代解决Ux = y

x = zeros(n, 1);

for i = n:-1:1

x(i) = (y(i) - U(i, i+1:end) \* x(i+1:end)) / U(i, i);

end

end

% 测试代码

clc; clear all;

A = [1 2 -1; 2 1 -2; -3 1 1];

b1 = [3 3 -6]';

[L, U, P, Q] = full\_pivot\_LU(A);

disp('矩阵L:');

disp(L);

disp('矩阵U:');

disp(U);

disp('置换矩阵P:');

disp(P);

disp('置换矩阵Q:');

disp(Q);

X1 = back\_substitution\_two(L, U, P \* b1);

disp('LU分解求解结果:');

disp(X1);

% MATLAB直接求解结果验证

X\_direct = A \ b1;

disp('MATLAB直接求解结果:');

disp(X\_direct);

% 计算增长因子

growth\_factor = max(max(abs(U))) / max(max(abs(A)));

disp('全选主元增长因子:');

disp(growth\_factor);

3 考虑新的选主元的策略(随机选主元)

由于主元的选择每次都需要搜索比较主元所在的列从k到n所有的元素，当n比较大时，搜索量比较大，随机列选主元的方式就是在k到n中所有的元素随机选择一个为新的主元，然后进行高斯消去的过程，请编写相应的程序。

通过增长因子说明随机选主元是否具有优越性。

现在考虑主元所在的列从k到n中随机选择r个，然后在这个r个中选择绝对值最大的一个为新的主元，然后进行高斯消去的过程，请编写相应的程序。分析r对增长因子的影响。【40分】

**random\_pivot\_LU函数**： 这个函数实现了随机列选主元的LU分解。首先，检查输入矩阵是否为方阵。然后在每一列中随机选择一个列，再在该列中选择绝对值最大的元素作为主元。通过交换行来保持矩阵A的变化，并记录行交换操作在矩阵P中。之后进行高斯消去，构造下三角矩阵L和上三角矩阵U。

**random\_subset\_pivot\_LU函数**： 这个函数实现了随机列子集选主元的LU分解。首先检查输入矩阵是否为方阵。然后在每一列中随机选择r个列，再在这些列中选择绝对值最大的元素作为主元。通过交换行和列来保持矩阵A的变化，并记录行交换操作在矩阵P中，列交换操作在矩阵Q中。之后进行高斯消去，构造下三角矩阵L和上三角矩阵U。

**back\_substitution\_two函数**： 该函数用于解决线性方程组Ly=b和Ux=y。通过前向替换解决Ly=b，通过回代解决Ux=y。此函数被用于验证LU分解求解线性方程组的正确性。

最终定义了一个矩阵A和向量b，分别使用随机列选主元和随机列子集选主元的LU分解来求解线性方程组Ax=b，并输出结果。随后，使用MATLAB自带的直接求解函数（\运算符）进行对比，验证LU分解求解的正确性。最后，计算增长因子，以评估两种策略的数值稳定性。

function [L, U, P] = random\_pivot\_LU(A)

% 随机列选主元 LU分解

% Input: 矩阵 A

% Output: L, U, P

% L: 下三角矩阵

% U: 上三角矩阵

% P: 行交换矩阵

[n, n2] = size(A);

if n ~= n2

error('矩阵必须是方阵');

end

P = eye(n);

for k = 1:n-1

% 随机选择一个列

col\_indices = k:n;

rand\_col = col\_indices(randi(length(col\_indices)));

% 选择该列中的最大元素作为主元

[~, i\_max] = max(abs(A(k:n, rand\_col)));

i\_max = i\_max + k - 1;

% 交换行

if i\_max ~= k

A([k, i\_max], :) = A([i\_max, k], :);

P([k, i\_max], :) = P([i\_max, k], :);

end

% 高斯消去

for i = k+1:n

A(i, k) = A(i, k) / A(k, k);

A(i, k+1:n) = A(i, k+1:n) - A(i, k) \* A(k, k+1:n);

end

end

L = tril(A, -1) + eye(n);

U = triu(A);

end

function [L, U, P, Q] = random\_subset\_pivot\_LU(A, r)

% 随机列子集选主元 LU分解

% Input: 矩阵 A

% Output: L, U, P, Q

% L: 下三角矩阵

% U: 上三角矩阵

% P: 行交换矩阵

% Q: 列交换矩阵

[n, n2] = size(A);

if n ~= n2

error('矩阵必须是方阵');

end

P = eye(n);

Q = eye(n);

for k = 1:n-1

% 随机选择 r 列

col\_indices = k:n;

rand\_cols = col\_indices(randperm(length(col\_indices), r));

% 选择这些列中的最大元素作为主元

[~, idx] = max(abs(A(k:n, rand\_cols)), [], 'all', 'linear');

[i\_max, j\_max] = ind2sub([n-k+1, r], idx);

i\_max = i\_max + k - 1;

j\_max = rand\_cols(j\_max);

% 交换行

if i\_max ~= k

A([k, i\_max], :) = A([i\_max, k], :);

P([k, i\_max], :) = P([i\_max, k], :);

end

% 交换列

if j\_max ~= k

A(:, [k, j\_max]) = A(:, [j\_max, k]);

Q(:, [k, j\_max]) = Q(:, [j\_max, k]);

end

% 高斯消去

for i = k+1:n

A(i, k) = A(i, k) / A(k, k);

A(i, k+1:n) = A(i, k+1:n) - A(i, k) \* A(k, k+1:n);

end

end

L = tril(A, -1) + eye(n);

U = triu(A);

end

% 测试代码

clc; clear all;

% 定义矩阵A和向量b

A = [1 2 -1; 2 1 -2; -3 1 1];

b = [3; 3; -6];

% 随机列选主元

[L\_rand, U\_rand, P\_rand] = random\_pivot\_LU(A);

X\_rand = back\_substitution\_two(L\_rand, U\_rand, P\_rand \* b);

disp('随机列选主元 LU分解结果:');

disp(X\_rand);

% 随机列子集选主元

r = 2; % 子集大小

[L\_sub, U\_sub, P\_sub, Q\_sub] = random\_subset\_pivot\_LU(A, r);

X\_sub = back\_substitution\_two(L\_sub, U\_sub, P\_sub \* b);

disp(['随机列子集选主元 (r=', num2str(r), ') LU分解结果:']);

disp(X\_sub);

% MATLAB 直接求解结果验证

X\_direct = A \ b;

disp('MATLAB直接求解结果:');

disp(X\_direct);

% 计算增长因子

growth\_factor\_rand = max(max(abs(U\_rand))) / max(max(abs(A)));

growth\_factor\_sub = max(max(abs(U\_sub))) / max(max(abs(A)));

disp('随机列选主元增长因子:');

disp(growth\_factor\_rand);

disp(['随机列子集选主元 (r=', num2str(r), ') 增长因子:']);

disp(growth\_factor\_sub);

function x = back\_substitution\_two(L, U, b)

% Ly = b, Ux = y

% b : 列向量

% x : 解向量

% 前向替换解决Ly = b

n = length(b);

y = zeros(n, 1);

for i = 1:n

y(i) = (b(i) - L(i, 1:i-1) \* y(1:i-1)) / L(i, i);

end

% 回代解决Ux = y

x = zeros(n, 1);

for i = n:-1:1

x(i) = (y(i) - U(i, i+1:end) \* x(i+1:end)) / U(i, i);

end

end