数值算法 Homework 01

Due: Sept. 10, 2024
姓名: 雍崔扬
学号: 21307140051

Problem 1

Let $\hat{x}$ be an approximation to x.
In practice, it is often much easier to estimate $\widetilde{E}_{\text{rel}}(\hat{x}) = \frac{\|x-\hat{x}\|}{\|\hat{x}\|}$ compared to $E_{\text{rel}}(\hat{x}) = \frac{\|x-\hat{x}\|}{\|x\|}$.
What is the relationship between $E_{\text{rel}}(\hat{x})$ and $\widetilde{E}_{\text{rel}}(\hat{x})$?

Solution:
记 $\delta x = \hat{x} - x$，不妨假设 $E_{\text{rel}}(\hat{x}) = \|\delta x\|/\|x\| < 1$ (因而有 $\|\delta x\| < \|x\|$)

$$
\begin{aligned}
|\widetilde{E}_{\text{rel}}(\hat{x}) - E_{\text{rel}}(\hat{x})| &= \left| \frac{\|x-\hat{x}\|}{\|x\|} - \frac{\|x-\hat{x}\|}{\|\hat{x}\|} \right| \\
&= \left| \frac{\|\delta x\|}{\|x\|} - \frac{\|\delta x\|}{\|x+\delta x\|} \right| \\
&= \left| \frac{\|\delta x\|(\|x+\delta x\| - \|x\|)}{\|x\|\|x+\delta x\|} \right| \quad (\text{note that } \|x\| - \|\delta x\| \le \|x+\delta x\| \le \|x\| + \|\delta x\|) \\
&\le \frac{\|\delta x\|(\|x\| + \|\delta x\| - \|x\|)}{\|x\|(\|x\| - \|\delta x\|)} \\
&= \frac{\|\delta x\|^2}{\|x\|(\|x\| - \|\delta x\|)} \\
&= \frac{(\|\delta x\|/\|x\|)^2}{1 - \|\delta x\|/\|x\|} \\
&= \frac{(E_{\text{rel}}(\hat{x}))^2}{1 - E_{\text{rel}}(\hat{x})}
\end{aligned}
$$

当 $E_{\text{rel}}(\hat{x}) = \|\delta x\|/\|x\| \to 0$ 时，误差上界 $\frac{(E_{\text{rel}}(\hat{x}))^2}{1-E_{\text{rel}}(\hat{x})} = \frac{(\|\delta x\|/\|x\|)^2}{1-\|\delta x\|/\|x\|}$ 也趋近于 0，
表明以 $\widetilde{E}_{\text{rel}}(\hat{x})$ 代替 $E_{\text{rel}}(\hat{x})$ 的做法是数值稳定的.
展开绝对值就得到:

$$
\begin{gathered}
\widetilde{E}_{\text{rel}}(\hat{x}) \le \frac{E_{\text{rel}}(\hat{x})}{1 - E_{\text{rel}}(\hat{x})} \\
\Downarrow \\
\frac{\widetilde{E}_{\text{rel}}(\hat{x})}{1 + \widetilde{E}_{\text{rel}}(\hat{x})} \le E_{\text{rel}}(\hat{x})
\end{gathered}
$$

另一种做法:

$$
\begin{aligned}
|E_{\text{rel}}(\hat{x}) - \widetilde{E}_{\text{rel}}(\hat{x})| &= \left| \frac{\|x-\hat{x}\|}{\|x\|} - \frac{\|x-\hat{x}\|}{\|\hat{x}\|} \right| \\
&= \left| \frac{\|\delta x\|}{\|\hat{x}-\delta x\|} - \frac{\|\delta x\|}{\|\hat{x}\|} \right| \\
&= \left| \frac{\|\delta x\|(\|\hat{x}\| - \|\hat{x}-\delta x\|)}{\|\hat{x}-\delta x\|\|\hat{x}\|} \right| \quad (\text{note that } \|\hat{x}\| - \|\delta x\| \le \|\hat{x}-\delta x\| \le \|\hat{x}\| + \|\delta x\|) \\
&\le \frac{\|\delta x\| \cdot \|\delta x\|}{(\|\hat{x}\| - \|\delta x\|)\|\hat{x}\|} \\
&= \frac{(\|\delta x\|/\|x\|)^2}{1 - \|\delta x\|/\|x\|} \\
&= \frac{(\widetilde{E}_{\text{rel}}(\hat{x}))^2}{1 - \widetilde{E}_{\text{rel}}(\hat{x})}
\end{aligned}
$$

展开绝对值就得到:

$$E_{\text{rel}}(\hat{x}) \le \widetilde{E}_{\text{rel}}(\hat{x}) + \frac{(\widetilde{E}_{\text{rel}}(\hat{x}))^2}{1-\widetilde{E}_{\text{rel}}(\hat{x})} = \frac{\widetilde{E}_{\text{rel}}(\hat{x})}{1-\widetilde{E}_{\text{rel}}(\hat{x})}$$

TA: 第一题可以分类讨论，根据 $x, \hat{x}$ 的相对大小，取消绝对值，最终得到 $E_{\text{rel}}(\hat{x})$ 和 $\widetilde{E}_{\text{rel}}(\hat{x})$ 的等式关系.

邵老师的解法:
不妨设 $E_{\text{rel}}(\hat{x}) = \frac{\|x-\hat{x}\|}{\|x\|} < 1$ 和 $\widetilde{E}_{\text{rel}}(\hat{x}) = \frac{\|x-\hat{x}\|}{\|\hat{x}\|} < 1$，则我们有:

$$\begin{aligned}
E_{\text{rel}}(\hat{x}) &= \frac{\|\hat{x}\|}{\|x\|}\widetilde{E}_{\text{rel}}(\hat{x}) \\
&= \frac{\|(\hat{x}-x)+x\|}{\|x\|}\widetilde{E}_{\text{rel}}(\hat{x}) \\
&\le \frac{\|\hat{x}-x\|+\|x\|}{\|x\|}\widetilde{E}_{\text{rel}}(\hat{x}) \qquad \Rightarrow \qquad E_{\text{rel}}(\hat{x}) \le \frac{\widetilde{E}_{\text{rel}}(\hat{x})}{1-\widetilde{E}_{\text{rel}}(\hat{x})} \\
&= (\frac{\|\hat{x}-x\|}{\|x\|}+1)\widetilde{E}_{\text{rel}}(\hat{x}) \\
&= (E_{\text{rel}}(\hat{x})+1)\widetilde{E}_{\text{rel}}(\hat{x})
\end{aligned}$$

类似地，我们有:

$$\begin{aligned}
\widetilde{E}_{\text{rel}}(\hat{x}) &= \frac{\|x\|}{\|\hat{x}\|}E_{\text{rel}}(\hat{x}) \\
&= \frac{\|(x-\hat{x})+\hat{x}\|}{\|\hat{x}\|}E_{\text{rel}}(\hat{x}) \\
&\le \frac{\|x-\hat{x}\|+\|\hat{x}\|}{\|\hat{x}\|}E_{\text{rel}}(\hat{x}) \qquad \Rightarrow \qquad E_{\text{rel}}(\hat{x}) \ge \frac{\widetilde{E}_{\text{rel}}(\hat{x})}{1+\widetilde{E}_{\text{rel}}(\hat{x})} \\
&= (\frac{\|\hat{x}-x\|}{\|\hat{x}\|}+1)E_{\text{rel}}(\hat{x}) \\
&= (\widetilde{E}_{\text{rel}}(\hat{x})+1)E_{\text{rel}}(\hat{x})
\end{aligned}$$

总之我们有:

$$\begin{gathered}
\frac{\widetilde{E}_{\text{rel}}(\hat{x})}{1+\widetilde{E}_{\text{rel}}(\hat{x})} \le E_{\text{rel}}(\hat{x}) \le \frac{\widetilde{E}_{\text{rel}}(\hat{x})}{1-\widetilde{E}_{\text{rel}}(\hat{x})} \\
E_{\text{rel}}(\hat{x}) = \widetilde{E}_{\text{rel}}(\hat{x}) + O((\widetilde{E}_{\text{rel}}(\hat{x}))^2)
\end{gathered}$$

我的解法相当于为二阶项提供了估计:
(可以用 $E_{\text{rel}}(\hat{x})$ 也可以用 $\widetilde{E}_{\text{rel}}(\hat{x})$ 表示)

$$|E_{\text{rel}}(\hat{x}) - \widetilde{E}_{\text{rel}}(\hat{x})| \le \frac{(E_{\text{rel}}(\hat{x}))^2}{1-E_{\text{rel}}(\hat{x})} \text{ or } \frac{(\widetilde{E}_{\text{rel}}(\hat{x}))^2}{1-\widetilde{E}_{\text{rel}}(\hat{x})}$$

Problem 2

How to evaluate $f(x) = \tan(x) - \sin(x)$ for $x \approx 0$ so that numerical cancellation is avoided?

Solution:

$$\begin{aligned}
f(x) &= \tan(x) - \sin(x) \\
&= \tan(x)(1-\cos(x)) \\
&= \tan(x)\left(1-\left(1-2\left(\sin\left(\frac{x}{2}\right)\right)^2\right)\right) \\
&= 2\tan(x)\left(\sin\left(\frac{x}{2}\right)\right)^2
\end{aligned}$$

Problem 3

Let A be a square banded matrix with half-bandwidth β (i.e., $a_{ij}=0$ if $|i-j|>\beta$).
Suppose that the LU factorization of A (without pivoting) is $A=LU$.
Show that both L and U are banded matrices with half-bandwidth β.

Solution:

> 以 $n=5,\beta=1$ 的情况为例:
>
> $$\begin{aligned}\begin{bmatrix} \ast & \ast & & & \\ \ast & \ast & \ast & & \\ & \ast & \ast & \ast & \\ & & \ast & \ast & \ast \\ & & & \ast & \ast \end{bmatrix} &= \begin{bmatrix} \ast & & & & \\ \ast & \ast & & & \\ & & \ast & & \\ & & & \ast & \\ & & & & \ast \end{bmatrix}\begin{bmatrix} \ast & \ast & & & \\ & \ast & \ast & & \\ & \ast & \ast & \ast & \\ & & \ast & \ast & \ast \\ & & & \ast & \ast \end{bmatrix} \\ &= \begin{bmatrix} \ast & & & & \\ \ast & \ast & & & \\ & & \ast & & \\ & & & \ast & \\ & & & & \ast \end{bmatrix}\begin{bmatrix} \ast & & & & \\ & \ast & & & \\ & \ast & \ast & & \\ & & & \ast & \\ & & & & \ast \end{bmatrix}\begin{bmatrix} \ast & \ast & & & \\ & \ast & \ast & & \\ & & \ast & \ast & \\ & & \ast & \ast & \ast \\ & & & \ast & \ast \end{bmatrix} \\ &= \begin{bmatrix} \ast & & & & \\ \ast & \ast & & & \\ & & \ast & & \\ & & & \ast & \\ & & & & \ast \end{bmatrix}\begin{bmatrix} \ast & & & & \\ & \ast & & & \\ & \ast & \ast & & \\ & & & \ast & \\ & & & & \ast \end{bmatrix}\begin{bmatrix} \ast & & & & \\ & \ast & & & \\ & & \ast & & \\ & & \ast & \ast & \\ & & & & \ast \end{bmatrix}\begin{bmatrix} \ast & \ast & & & \\ & \ast & \ast & & \\ & & \ast & \ast & \\ & & & \ast & \ast \\ & & & \ast & \ast \end{bmatrix} \\ &= \begin{bmatrix} \ast & & & & \\ \ast & \ast & & & \\ & & \ast & & \\ & & & \ast & \\ & & & & \ast \end{bmatrix}\begin{bmatrix} \ast & & & & \\ & \ast & & & \\ & \ast & \ast & & \\ & & & \ast & \\ & & & & \ast \end{bmatrix}\begin{bmatrix} \ast & & & & \\ & \ast & & & \\ & & \ast & & \\ & & \ast & \ast & \\ & & & & \ast \end{bmatrix}\begin{bmatrix} \ast & & & & \\ & \ast & & & \\ & & \ast & & \\ & & & \ast & \\ & & & \ast & \ast \end{bmatrix}\begin{bmatrix} \ast & \ast & & & \\ & \ast & \ast & & \\ & & \ast & \ast & \\ & & & \ast & \ast \\ & & & & \ast \end{bmatrix} \\ &= \begin{bmatrix} \ast & & & & \\ \ast & \ast & & & \\ & \ast & \ast & & \\ & & \ast & \ast & \\ & & & \ast & \ast \end{bmatrix}\begin{bmatrix} \ast & \ast & & & \\ & \ast & \ast & & \\ & & \ast & \ast & \\ & & & \ast & \ast \\ & & & & \ast \end{bmatrix}\end{aligned}$$

设 A 的阶数为 n，记 $\mathbb{R}^n$ 的第 k 个标准单位基向量为 e_k.
在不选主元的 Gauss 消去法中，考虑第 k 步的消元个数:

- 若 $k=1,\dots,n-\beta$，则该步消元 β 个，
  构造的 Gauss 变换矩阵 $L_k=I_n-l_ke_k^{\mathrm{T}}$ 仅在第 k 列可能有非零对角元，且从 $(k,k+1)$ 位置到 $(k,k+\beta)$ 位置.
- 在其余 $\beta-1$ 步中，该步消元 $n-k\le\beta-1$ 个
  构造的 Gauss 变换矩阵 $L_k=I_n-l_ke_k^{\mathrm{T}}$ 仅在第 k 列可能有非零对角元，且从 $(k,k+1)$ 位置直到 (k,n) 位置.

显然上述 $n-1$ 步消元得到的上三角阵 U 和 A 一样具有带宽 β.
这是因为上方的行的严格上三角部分具有更多零元，
向下消元时不会增加下方的行的严格上三角部分的非零元个数.

根据 Gauss 变换的性质我们有:

$$\begin{aligned}L&=(L_{n-1}\cdots L_2L_1)^{-1}\\&=L_1^{-1}L_2^{-1}\cdots L_{n-1}^{-1}\\&=(I+l_1e_1^{\mathrm{T}})(I+l_2e_2^{\mathrm{T}})\cdots(I+l_{n-1}e_{n-1}^{\mathrm{T}})\quad(\text{note that }e_j^{\mathrm{T}}l_i=0\text{ for all }j<i)\\&=I+l_1e_1^{\mathrm{T}}+l_2e_2^{\mathrm{T}}+\cdots+l_{n-1}e_{n-1}^{\mathrm{T}}\end{aligned}$$

因此 L 的第 $k=1,\dots,n-\beta$ 列都在 $(k,k+1)$ 位置到 $(k,k+\beta)$ 可能有非零对角元，
而在剩余列中，对角线下方的非对角元都可能非零.
这表明 L 是具有带宽 β 的下三角带状矩阵.

Problem 4

Find the exact LU factorization of the $n\times n$ matrix

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & -1 & \cdots & 1 & 1 \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{bmatrix}$$

Solution:
记 $\mathbb{R}^n$ 的第 k 个标准单位基向量为 e_k.
注意到第 $k=1,\dots,n-1$ 步 Gauss 变换为:

$$L_k = I_n - l_k e_k^{\mathrm{T}} \text{ where } l_k = -e_{k+1} - \cdots - e_n$$

根据 Gauss 变换的性质我们有:

$$\begin{aligned} L &= (L_{n-1}\cdots L_2 L_1)^{-1} \\ &= L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1} \\ &= (I + l_1 e_1^{\mathrm{T}})(I + l_2 e_2^{\mathrm{T}}) \cdots (I + l_{n-1} e_{n-1}^{\mathrm{T}}) \quad (\text{note that } e_j^{\mathrm{T}} l_i = 0 \text{ for all } j < i) \\ &= I + l_1 e_1^{\mathrm{T}} + l_2 e_2^{\mathrm{T}} + \cdots + l_{n-1} e_{n-1}^{\mathrm{T}} \\ &= \begin{bmatrix} 1 & & & & & \\ -1 & 1 & & & & \\ -1 & -1 & 1 & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ -1 & -1 & -1 & \cdots & 1 & \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{bmatrix} \end{aligned}$$

注意到第 k 步总是用第 k 行直接加到后 $n-k$ 行上进行消元，因此得到上三角阵为:

$$U = \begin{bmatrix} 1 & & & & & 1 \\ & 1 & & & & 2 \\ & & 1 & & & 4 \\ & & & \ddots & & \vdots \\ & & & & 1 & 2^{n-2} \\ & & & & & 2^{n-1} \end{bmatrix}$$

因此 A 的 LU 分解为:

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & -1 & \cdots & 1 & 1 \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & & & & \\ -1 & 1 & & & & \\ -1 & -1 & 1 & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ -1 & -1 & -1 & \cdots & 1 & \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & & 1 \\ & 1 & & & & 2 \\ & & 1 & & & 4 \\ & & & \ddots & & \vdots \\ & & & & 1 & 2^{n-2} \\ & & & & & 2^{n-1} \end{bmatrix} = LU$$

其增长因子为:

$$\rho = \frac{\max_{i,j} |u_{ij}|}{\|A\|_\infty} = \frac{2^{n-1}}{1} = 2^{n-1}$$

达到了部分主元 Gauss 消去法的增长因子的上界 2^{n-1}.

Problem 5

Implement Gaussian elimination (without pivoting) for solving nonsingular linear systems.
You may assume that no divide-by-zero error is encountered.
Measure the execution time of your program in terms of matrix dimensions and visualize the result by a log–log scale plot.
(You may generate your test matrices with normally distributed random elements.)

(1) Gauss 消去法

不选主元的 Gauss 消去法的算法如下:
(Gauss 消去法, 数值线性代数, 算法 $1.1.3$**)**

$$
\begin{aligned}
&\text{function: } [L, U] = \text{Gaussian\_Elimination}(A) \\
&\quad n = \dim(A) \\
&\quad \text{for } k = 1 : n-1 \\
&\quad\quad A(k+1:n, k) \leftarrow A(k+1:n, k)/A(k,k) \\
&\quad\quad A(k+1:n, k+1:n) \leftarrow A(k+1:n, k+1:n) - A(k+1:n, k)A(k, k+1:n) \\
&\quad \text{end} \\
&\quad L = I_n + A \odot (\text{strictly lower triangular matrix with all ones}) \\
&\quad U = A \odot (\text{upper triangular matrix with all ones}) \\
&\quad \text{return } [L, U]
\end{aligned}
$$

总浮点运算数为:

$$
\begin{aligned}
\sum_{k=1}^{n-1}((n-k) + 2(n-k)^2) &= \sum_{i=1}^{n-1}(i + 2i^2) \\
&= \frac{1}{2}(n-1)n + 2 \cdot \frac{1}{6}(n-1)n(2n-1) \\
&= \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n \\
&= \Theta\left(\frac{2}{3}n^3\right)
\end{aligned}
$$

MATLAB 代码如下:

```
function [L, U] = Gaussian_Elimination(A)
    % Input:
    % A - An n x n matrix
    %
    % Output:
    % L - Lower triangular matrix
    % U - Upper triangular matrix

    % Get the size of the matrix A
    [n, ~] = size(A);

    % Perform Gaussian Elimination
    for k = 1:n-1
        % Update column elements below the diagonal
        A(k+1:n, k) = A(k+1:n, k) / A(k, k);

        % Update the remaining submatrix
        A(k+1:n, k+1:n) = A(k+1:n, k+1:n) - A(k+1:n, k) * A(k, k+1:n);
    end

    % Construct the lower triangular matrix L
    L = eye(n) + tril(A, -1);

    % Construct the upper triangular matrix U
    U = triu(A);

    % Return the results
    return;
end
```

(2) 回代法 & 前代法

根据 Gauss 消元法得到 $A = LU$ 后，我们按如下步骤求解线性方程组 $Ax = b$:

- 用前代法求解 $Ly = b$ 得到 y
- 用回代法求解 $Ux = y$ 得到 x

MATLAB 代码如下:

```
% 示例: 求解线性方程组 Ax = b
function x = Solve_Linear_System(A, b)
    % 使用 Gaussian 消去法计算 A = LU
    [L, U] = Gaussian_Elimination(A);

    % 使用前代法求解 Ly = b
    y = Forward_Sweep(L, b);

    % 使用回代法求解 Ux = y
    x = Backward_Sweep(U, y);
end
```

(前代法, 数值线性代数, 算法 1.1.1)

$$\begin{aligned}
&\text{function : } y = \text{Forward_Sweep}[L, b] \\
&\quad n \leftarrow \text{length}(b) \\
&\quad \text{for } i = 1 : n-1 \\
&\quad\quad b(i) \leftarrow b(i)/L(i,i) \\
&\quad\quad b(i+1:n) \leftarrow b(i+1:n) - b(i)L(i+1:n, i) \\
&\quad \text{end} \\
&\quad b(n) \leftarrow b(n)/L(n,n) \\
&\quad \text{return } b
\end{aligned}$$

最终 $Ly = b$ 的解 y 存储在 b 中.
第 $1 \leq i \leq n-1$ 步浮点运算次数为 $1 + (n-i) + (n-i) = 2(n-i) + 1$,
最后一步浮点运算次数为 1.
总浮点运算次数为:

$$\begin{aligned}
\sum_{i=1}^{n-1} (2(n-i)+1) + 1 &= \sum_{k=1}^{n-1} (2k+1) + 1 \\
&= \frac{1}{2}(n-1)(3+2n-1) + 1 \\
&= n^2 - 1 + 1 \\
&= n^2
\end{aligned}$$

MATLAB 代码如下:

```
function y = Forward_Sweep(L, b)
    % 前代法求解 Ly = b
    n = length(b);
    for i = 1:n-1
        b(i) = b(i) / L(i, i);  % 对角线归一化
        b(i+1:n) = b(i+1:n) - b(i) * L(i+1:n, i);  % 消去
    end
    b(n) = b(n) / L(n, n);  % 处理最后一行
    y = b;  % 返回结果
end
```

(回代法, 数值线性代数, 算法 1.1.2)

$$\begin{aligned}
&\text{function: } x = \text{Backward_Sweep}[U, y] \\
&\quad n \leftarrow \text{length}(y) \\
&\quad \text{for } i = n : -1 : 2 \\
&\quad\quad y(i) \leftarrow y(i)/U(i,i) \\
&\quad\quad y(1:i-1) \leftarrow y(1:i-1) - y(i)U(1:i-1, i) \\
&\quad \text{end} \\
&\quad y(1) \leftarrow y(1)/U(1,1) \\
&\quad \text{return } y
\end{aligned}$$

最终 $Ux = y$ 的解 x 存储在 y 中.
第 $2 \leq i \leq n$ 步浮点运算次数为 $1 + (i-1) + (i-1) = 2i - 1$,
最后一步浮点运算次数为 1.
总浮点运算次数为:

$$
\begin{aligned}
\sum_{i=2}^{n}(2i-1)+1 &= \sum_{k=1}^{n-1}(2k+1)+1 \\
&= \frac{1}{2}(n-1)(3+2n-1)+1 \\
&= n^2-1+1 \\
&= n^2
\end{aligned}
$$

MATLAB 代码如下:

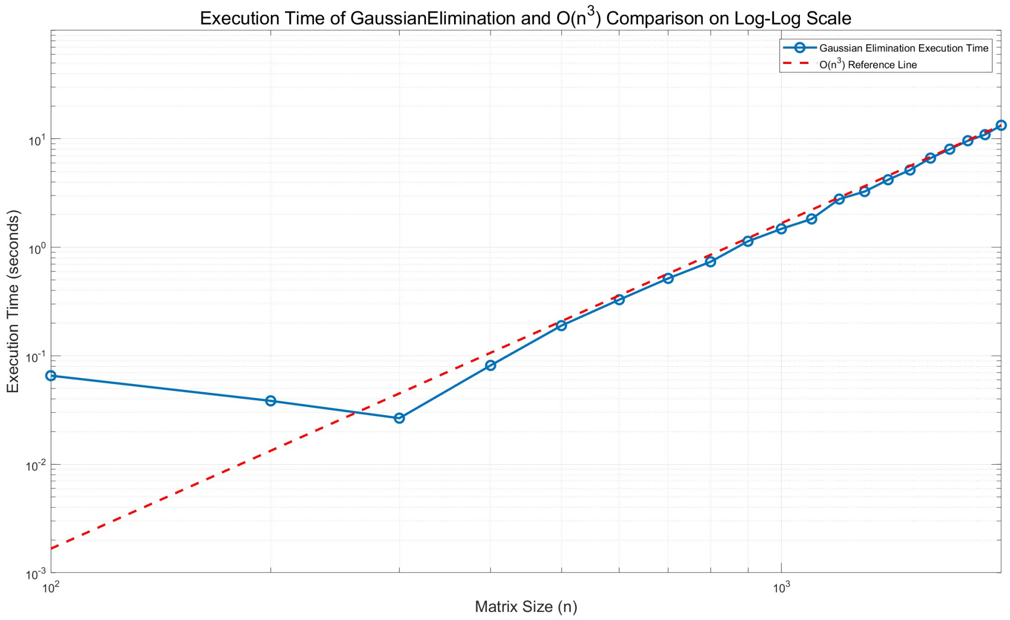
```
function x = Backward_Sweep(U, y)
    % 回代法求解 Ux = y
    n = length(y);
    for i = n:-1:2
        y(i) = y(i) / U(i, i);  % 对角线归一化
        y(1:i-1) = y(1:i-1) - y(i) * U(1:i-1, i);  % 消去
    end
    y(1) = y(1) / U(1, 1);  % 处理第一行
    x = y;  % 返回结果
end
```

(3) 函数调用

```
% 定义不同 n 值的范围 (100到2000, 每步100)
n_values = 100:100:2000;
execution_times = zeros(size(n_values));

% 遍历每个 n 值
for i = 1:length(n_values)
    n = n_values(i);

    % 生成随机 nxn 矩阵和随机右侧向量 b
    A = randn(n, n);
    b = randn(n, 1);

    % 记录 Gauss 消去法求解线性方程组 Ax = b 的执行时间
    tic;  % 开始计时

    % 使用 Gauss 消去法结合前代法和回代法求解线性方程组 Ax = b
    Solve_Linear_System(A, b)

    execution_times(i) = toc;  % 停止计时并记录时间

    % 输出当前维度和执行时间
    fprintf('Matrix size: %d x %d, Execution time: %.4f seconds\n', n, n, execution_times(i));
end

% 绘制执行时间的 log-log 图
figure;
loglog(n_values, execution_times, '-o', 'LineWidth', 2, 'MarkerSize', 8);
hold on;

% 绘制 n^3 比较线（归一化以匹配执行时间的尺度）
normalized_n_cubed = (n_values.^3) * (execution_times(end) / n_values(end)^3);
loglog(n_values, normalized_n_cubed, '--r', 'LineWidth', 2);

% 添加标签和标题
xlabel('Matrix Size (n)', 'FontSize', 14);
ylabel('Execution Time (seconds)', 'FontSize', 14);
title('Execution Time of GaussianElimination and O(n^3) Comparison on Log-Log Scale', 'FontSize',
16);
legend('Gaussian Elimination Execution Time', 'O(n^3) Reference Line');
grid on;
hold off;
```

输出结果:

```
Matrix size: 100 x 100, Execution time: 0.0099 seconds
Matrix size: 200 x 200, Execution time: 0.0156 seconds
Matrix size: 300 x 300, Execution time: 0.0286 seconds
Matrix size: 400 x 400, Execution time: 0.0840 seconds
Matrix size: 500 x 500, Execution time: 0.1986 seconds
Matrix size: 600 x 600, Execution time: 0.3693 seconds
Matrix size: 700 x 700, Execution time: 0.5889 seconds
Matrix size: 800 x 800, Execution time: 0.8541 seconds
Matrix size: 900 x 900, Execution time: 1.2081 seconds
Matrix size: 1000 x 1000, Execution time: 1.7111 seconds
Matrix size: 1100 x 1100, Execution time: 2.3695 seconds
Matrix size: 1200 x 1200, Execution time: 3.4898 seconds
Matrix size: 1300 x 1300, Execution time: 4.2514 seconds
Matrix size: 1400 x 1400, Execution time: 5.5133 seconds
Matrix size: 1500 x 1500, Execution time: 6.6729 seconds
Matrix size: 1600 x 1600, Execution time: 7.1258 seconds
Matrix size: 1700 x 1700, Execution time: 9.3520 seconds
Matrix size: 1800 x 1800, Execution time: 12.8476 seconds
Matrix size: 1900 x 1900, Execution time: 11.9220 seconds
Matrix size: 2000 x 2000, Execution time: 13.9834 seconds
```



这验证了 Gauss 消元法 $O(n^3)$ 级别的时间复杂度.

Problem 6 (optional)

Write a program to solve the quadratic equation $ax^2 + bx + c = 0$ with real coefficients. Describe how to avoid cancellation when the equation has a tiny root.

Solution:
实系数一元二次方程 $ax^2 + bx + c = 0\ (a \neq 0)$ 的解为:

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$
$$x_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

当 $4ac > b^2$ 时方程具有一对共轭复根，不存在相消问题.
当 $4ac \approx b^2$ 时相消问题并不严重 (判别式的相消问题与根号外的加减法的相消问题相比并不严重).
当 $4ac \ll b^2$ 时:

- 若 $b > 0$，则 x_2 的分子计算时会出现相消，因此应用 $x_2 = \frac{2c}{-b-\sqrt{b^2-4ac}}$ 计算 x_2

- 若 $b < 0$，则 x_1 的分子计算时会出现相消，因此应用 $x_1 = \frac{2c}{-b+\sqrt{b^2-4ac}}$ 计算 x_1

总之我们有如下 MATLAB 代码:

```
function [x1, x2] = solveQuadratic(a, b, c)
    % Check if the equation is actually quadratic
    if a == 0
        error('Coefficient a cannot be zero in a quadratic equation.');
    end

    % Calculate the discriminant
    D = b^2 - 4*a*c;

    % Compute the roots
    if D >= 0
        % Real roots
        if b >= 0
            x1 = (-b - sqrt(D)) / (2*a);
            x2 = (2*c) / (-b - sqrt(D));
        else
            x1 = (-b + sqrt(D)) / (2*a);
            x2 = (2*c) / (-b + sqrt(D));
        end
    else
        % Complex roots
        realPart = -b / (2*a);
        imagPart = sqrt(-D) / (2*a);
        x1 = realPart + 1i*imagPart;
        x2 = realPart - 1i*imagPart;
    end

    % Display the results
    fprintf('The roots of the quadratic equation are:\n');
    fprintf('x1 = %.12f + %.12fi\n', real(x1), imag(x1));
    fprintf('x2 = %.12f + %.12fi\n', real(x2), imag(x2));
end
```

调用函数:

```
% Test case with large b and small discriminant
solveQuadratic(1, 1e10, 1);

% Test case with complex root
solveQuadratic(1, 2, 3);
```

输出结果:

```
The roots of the quadratic equation are:
x1 = -10000000000.000000000000 + 0.000000000000i
x2 = -0.000000000100 + 0.000000000000i
The roots of the quadratic equation are:
x1 = -1.000000000000 + 1.414213562373i
x2 = -1.000000000000 + -1.414213562373i
```

Problem 7 (optional)

The 32-bit floating-point format discussed in the lecture is the IEEE-754 single precision format,
consisting of 1 sign bit, 8 bits of exponent, and 23 bits of significand.
Estimate the maximum finite floating-point number,
the minimum positive (normal) floating-point number,
as well as a tight upper bound on the relative representation error for the IEEE-754 single precision format.

What is the hexadecimal representation of $3/7$ in IEEE-754 single/double precision floating point format?
Explain how the number is encoded.

What about the IEEE-754 double precision format,
consisting of 1 sign bit, 11 bits of exponent, and 52 bits of significand?

Suppose that you are evaluating the harmonic series,
using IEEE 754 single/double precision floating-point numbers and obtained a "converged" result.
Make an estimate on when the computation converges, and what is the final result.

(1) 论文阅读

Read the paper *What every computer scientist should know about floating-point arithmetic* by David Goldberg.

It is a tutorial that focuses on the often non-intuitive behavior of floating-point arithmetic,
how errors arise, how the IEEE standard works,
and what system designers can / should do to support predictable, reliable floating-point computation.
It is organized roughly in three main parts:

- Rounding Error & Basic Properties
  - why floating-point arithmetic can behave oddly
  - what kinds of error measures we care about
- IEEE-754 Standard
  - formats and representations
  - rounding modes
  - special values and exceptional quantities
- Systems Aspects
  - how floating point affects hardware, languages, compilers, and software systems

(2) IEEE-754 浮点数

考虑 IEEE-754 单精度浮点数,
它由**符号位** (Sign)、**指数位** (Exponent)、**尾数位** (Fraction) 构成:

- 符号位: 占 1 位，用于表示数值的正负;
- 指数位: 占 8 位，使用偏移量为 $2^8-1=127$ 的偏移表示法;
- 尾数位: 占 23 位，用于表示数值的小数部分，隐含一个前导的 1.

当指数位全为 0 时:

- 若尾数位全为 0，则当符号位为 0 时代表 $+0$，当符号位为 0 时代表 -0.
- 否则代表非规格化浮点数.

当指数位全为 1 时:

- 若尾数位全为 0，则当符号位为 0 时代表 $+\infty$，当符号位为 0 时代表 $-\infty$.
- 否则代表 NaN (Not a Number).

当指数为 $1\sim 254$ 时，代表规格化浮点数，其数值为:

$$\text{Double Float}=(-1)^{\text{Sign}}\times 1.\text{Fraction}\times 2^{\text{Exponent}-127}$$

其精度范围约为:

$$\pm[1\times 2^{-126},(2-2^{-23})\times 2^{+127}]\approx\pm[1.18\times 10^{-38},3.40\times 10^{+38}]$$

如果考虑非规格化的情况，则精度范围约为:

$$\pm[2^{-23}\times 2^{-126},(2-2^{-23})\times 2^{+127}]\approx\pm[1.40\times 10^{-45},3.40\times 10^{+38}]$$

机器精度 $\text{eps}=2^{-23}\approx 1.19\times 10^{-7}$.
对于最邻近舍入法 (round-to-nearest)，相对舍入误差上界为:

$$\frac{|\text{fl}(x)-x|}{|x|}\leq 2^{-23}/2=2^{-24}\approx 5.96\times 10^{-8}$$

考虑 IEEE-754 双精度浮点数,
它由**符号位** (Sign)、**指数位** (Exponent)、**尾数位** (Fraction) 构成:

- 符号位: 占 1 位，用于表示数值的正负;

- 指数位: 占 11 位，使用偏移量为 $2^{10}-1=1023$ 的偏移表示法;
- 尾数位: 占 52 位，用于表示数值的小数部分，隐含一个前导的 1.

当指数位全为 0 时:

- 若尾数位全为 0，则当符号位为 0 时代表 $+0$，当符号位为 0 时代表 -0.
- 否则代表非规格化浮点数.

当指数位全为 1 时:

- 若尾数位全为 0，则当符号位为 0 时代表 $+\infty$，当符号位为 0 时代表 $-\infty$.
- 否则代表 NaN (Not a Number).

当指数为 $1\sim 2046$ 时，代表规格化浮点数，其数值为:

$$\text{Double Float}=(-1)^{\text{Sign}}\times 1.\text{Fraction}\times 2^{\text{Exponent}-1023}$$

其精度范围约为:

$$\pm[1\times 2^{-1022},(2-2^{-52})\times 2^{+1023}]\approx\pm[2.23\times 10^{-308},1.78\times 10^{+308}]$$

如果考虑非规格化的情况，则精度范围约为:

$$\pm[2^{-52}\times 2^{-1022},(2-2^{-52})\times 2^{+1023}]\approx\pm[4.94\times 10^{-324},1.78\times 10^{+308}]$$

机器精度 $\text{eps}=2^{-52}\approx 2.22\times 10^{-16}$.
对于最邻近舍入法 (round-to-nearest)，相对舍入误差上界为:

$$\frac{|\text{fl}(x)-x|}{|x|}\leq 2^{-52}/2=2^{-53}\approx 1.11\times 10^{-16}$$

(3) $3/7$ 的浮点数表示

注意到 $3/7=0.428571\ldots$ 是无限循环小数.
现考虑其位表示:

- $2\times 0.428571\cdots=0.857142\ldots$，得到位 0.
- $2\times 0.857142\cdots=1.714285\ldots$，得到位 1.
- $2\times 0.714285\cdots=1.428571\ldots$，得到位 1 (又回到了刚开始的情况)

因此 $3/7$ 的位表示为 $\text{0b0.011\_011\_011}\ldots=2^{-2}\times\text{0b1.101\_101\_101}\ldots$

- 符号位为 0
- 指数为 -2:
  对于单精度浮点数，指数位为 $-2+127=125=\text{0b0111\_1101}$;
  对于双精度浮点数，指数位为 $-2+1023=1021=\text{0b011\_1111\_1101}$;
- 尾数为 $\text{101\_101\_101}\ldots$ (去除了前导的 1):
  对于单精度浮点数，截断并舍入得到的 23 位尾数为:
  (截断的第 24 位为 1，而后续的位也出现了 1，因此向上舍入，第 23 位原本取 0，截断后取 1)

  $$\text{0b101\_1011\_0110\_1101\_1011\_0111}\ldots$$

  对于双精度浮点数，截断并舍入得到的 52 位尾数为:
  (截断的第 53 位为 0，直接截断即可，第 52 位取 1)

$$\text{0b1011\_0110\_1101\_1011\_0110\_1101\_1011\_0110\_1101\_1011\_0110\_1101\_1011}$$

因此 $3/7$ 的单精度浮点数表示为:

$$\begin{aligned}3/7&=\text{0b0011\_1110\_1101\_1011\_0110\_1101\_1011\_0111}\\&=\text{0x3EDB6DB7}\end{aligned}$$

而 $3/7$ 的双精度浮点数表示为:

$$\begin{aligned}3/7&=\text{0b0011\_1111\_1101\_1011\_0110\_1101\_1011\_0110\_1101\_1011\_0110\_1101\_1011\_0110\_1101\_1011}\\&=\text{0x3FDB6DB6DB6DB6DB}\end{aligned}$$

Python 代码验证:

```
import struct

x = 3/7

# 单精度 (32-bit)
single_hex = f"0x{struct.unpack('>I', struct.pack('>f', x))[0]:08X}"
print("single precision hex:", single_hex)

# 双精度 (64-bit)
double_hex = f"0x{struct.unpack('>Q', struct.pack('>d', x))[0]:016X}"
print("double precision hex:", double_hex)
```

运行结果:

```
single precision hex: 0x3EDB6DB7
double precision hex: 0x3FDB6DB6DB6DB6DB
```

(4) 调和级数

理论上，调和级数 $\sum_{n=1}^{\infty} 1/n$ 是发散的 (这又不得不提起那篇文章了),
而数值计算中发生的 "收敛" 是由浮点数的舍入误差造成的.

考虑以下算法:

$$\begin{aligned}
&\text{whlie } H_n \neq H_n + \frac{1}{n} \text{ do} \\
&\qquad H_{n+1} = H_n + \frac{1}{n} \\
&\qquad n = n+1 \\
&\text{end}
\end{aligned}$$

当计算机在计算 $H_n + 1/n$ 时,
它首先会将两个浮点数 H_n 和 $1/n$ 的指数部分对齐,
此时 $1/n$ 作为较小的数，其尾数部分会向右移动，超出的部分会被舍弃,
当 n 足够大时，H_n 和 $1/n$ 的指数部分的差距会足够大，使得在对齐过程中 $1/n$ 的尾数部分全部被舍弃.
这样 $H_n + 1/n$ 的结果就是 H_n，于是循环条件 $H_n \neq H_n + 1/n$ 判错，迭代终止.
这就是调和级数在数值计算中产生 "收敛" 现象的原因.

那么 H_n 收敛时的 n 的大致是多少呢?
要找 n 使得单精度浮点数下 $H_n = H_n + 1/n$ 成立,
即要找 n 使得单精度浮点数 H_n 和 $1/n$ 的指数部分至少相差 $23+2=25$ 位.
我们知道调和级数的增长速度类似于自然对数，即 $H_n \approx \ln(n) + \gamma$
(其中 Euler 常数 $\gamma = \lim_{n\to\infty}(H_n - \ln(n)) \approx 0.5772156649$)

因此 H_n 的指数部分 $\text{Exponent}(H_n) \approx \text{Floor}\{\log_2(\ln(n)+\gamma)\}$,
而 $1/n$ 的指数部分 $\text{Exponent}(1/n) = \text{Floor}\{-\log_2(n)\} = -\text{Ceil}\{\log_2(n)\}$,
其中 Ceil、Floor 分别代表上、下取整.

我们令:

$$\begin{aligned}
&\text{Exponent}(H_n) - \text{Exponent}(1/n) \\
&= \text{Floor}\{\log_2(\ln(n)+\gamma)\} + \text{Ceil}\{\log_2(n)\} \\
&\geq 24
\end{aligned}$$

通过数值方法解得 $n \approx 2.097 \times 10^6$.

- 对于双精度浮点数，将 24 替换为 $52+2=54$，解得 $n \approx 1.407 \times 10^{14}$.

MATLAB 代码如下:

```
result24 = find_min_n_converged(1e6, 1e3, 25);
H24_approx = log(result24) + 0.5772156649; % Euler 近似
fprintf('Estimation for single precision: n = %.0f, H_n ≈ %.8f\n', result24, H24_approx);

result54 = find_min_n_converged(1e14, 1e11, 54);
H54_approx = log(result54) + 0.5772156649;
```

```
fprintf('Estimation for double precision: n = %.0f, H_n ≈ %.16f\n', result54, H54_approx);

function min_n = find_min_n_converged(n_init, step, p)
    % find_min_n_converged searches for minimal n
    % n_init = 起始值 (linear search 的起点)
    % step   = 每次增加的步长 (coarse search 的步长)
    % p      = mantissa 位数 (e.g., 25 for single, 54 for double)

    gamma = 0.5772156649; % Euler-Mascheroni 常数
    n = n_init;

    % ----------- 粗搜索 (linear search) -----------
    while true
        term1 = floor(log2(log(n) + gamma));
        term2 = ceil(log2(n));
        if term1 + term2 >= p
            break;  % 找到一个上界
        end
        n = n + step;
    end

    % 此时 [n - step, n] 是包含解的区间
    low = max(n - step, 1);  % 避免 < 1
    high = n;

    % ----------- 二分搜索 (binary search) -----------
    while low < high
        mid = floor((low + high) / 2);
        term1 = floor(log2(log(mid) + gamma));
        term2 = ceil(log2(mid));

        if term1 + term2 >= p
            high = mid;   % 缩小到左半区间
        else
            low = mid + 1; % 缩小到右半区间
        end
    end

    min_n = low - 1;  % low == high, 再减去 1 就是最小的 n
end
```

运行结果:

```
Estimation for single precision: n = 2097152, H_n ≈ 15.13330646
Estimation for double precision: n = 281474976710656, H_n ≈ 33.8482803317773744
```

通过计算几何级数直接验证:

```
% 验证调和级数在单精度浮点数运算中的"收敛"现象
H_single = single(0.0);
n_single = single(1.0);   % 用 single 而不是 int32

while true
    if H_single == H_single + 1.0/n_single
        break;
    end
    H_single = H_single + 1.0./n_single;
    n_single = n_single + 1.0;
end

fprintf('Single precision stops at n = %.0f\n', n_single);
fprintf('H_n = %.8f\n', H_single);
```

运算结果:

```
Single precision stops at n = 2097152
H_n = 15.40368271
```

Problem 8

Let $U \in \mathbb{R}^{n\times n}$ be upper triangular and nonsingular.
Provide two different implementations for solving the linear system $Ux = b$, where $b \in \mathbb{R}^n$ is a given vector.

Solution:

- **解法 1 (回代法的原始形式):**

$$
\begin{aligned}
&\text{function: } x = \text{Backward\_Sweep\_Raw}[U, b]\\
&\qquad n \leftarrow \text{length}(b)\\
&\qquad b(n) \leftarrow b(n)/U(n,n)\\
&\qquad \text{for } i = n-1 : -1 : 1\\
&\qquad\qquad b(i) \leftarrow \frac{b(i) - U(i, i+1:n)\, b(i+1:n)}{U(i,i)}\\
&\qquad \text{end}\\
&\qquad \text{return } b
\end{aligned}
$$

  最终 $Ux = b$ 的解 x 存储在 b 中.

- **解法 2 (回代法的标准形式, 数值线性代数, 算法 $1.1.2$):**

$$
\begin{aligned}
&\text{function: } x = \text{Backward\_Sweep}[U, b]\\
&\qquad n \leftarrow \text{length}(b)\\
&\qquad \text{for } i = n : -1 : 2\\
&\qquad\qquad b(i) \leftarrow b(i)/U(i,i)\\
&\qquad\qquad b(1:i-1) \leftarrow b(1:i-1) - b(i)U(1:i-1, i)\\
&\qquad \text{end}\\
&\qquad b(1) \leftarrow b(1)/U(1,1)\\
&\qquad \text{return } b
\end{aligned}
$$

  最终 $Ux = b$ 的解 x 存储在 b 中.

(存疑: 哪种解法的舍入误差通常会更小一些?)
MATLAB 代码验证:

```matlab
% 生成测试数据
rng(51);
n = 1000;
U = triu(rand(n)*sqrt(n)) + 2*sqrt(n)*eye(n); % 保证随机生成的上三角阵的条件数适中
b = rand(n,1);

% 解法 1: Backward_Sweep_Raw
x1 = Backward_Sweep_Raw(U,b);

% 解法 2: Backward_Sweep
x2 = Backward_Sweep(U,b);

% 验证是否正确
fprintf('||U*x1 - b||_inf = %.2e\n', norm(U*x1 - b, Inf));
fprintf('||U*x2 - b||_inf = %.2e\n', norm(U*x2 - b, Inf));

% ======== 方法 1 =========
function b = Backward_Sweep_Raw(U,b)
    n = length(b);
    b(n) = b(n)/U(n,n);
    for i = n-1:-1:1
        b(i) = (b(i) - U(i,i+1:n) * b(i+1:n)) / U(i,i);
    end
end

% ======== 方法 2 =========
function b = Backward_Sweep(U,b)
```

```
    n = length(b);
    for i = n:-1:2
        b(i) = b(i)/U(i,i);
        b(1:i-1) = b(1:i-1) - b(i) * U(1:i-1,i);
    end
    b(1) = b(1)/U(1,1);
end
```

运行结果:

```
||U*x1 - b||_inf = 5.12e-13
||U*x2 - b||_inf = 5.62e-13
```

(Variant) Let $L \in \mathbb{R}^{n\times n}$ be Lower triangular and nonsingular.
Provide two different implementations for solving the linear system $Lx = b$, where $b \in \mathbb{R}^n$ is a given vector.

Solution:

- **解法 1 (前代法的原始形式):**

$$\begin{array}{l} \text{function: } x = \text{Forward\_Sweep\_Raw}[L, b] \\ \qquad n \leftarrow \text{length}(b) \\ \qquad b(1) \leftarrow b(1)/L(1,1) \\ \qquad \text{for } i = 1 : n-1 \\ \qquad\qquad b(i) \leftarrow \dfrac{b(i) - L(i,1:i-1)\, b(1:i-1)}{L(i,i)} \\ \qquad \text{end} \\ \qquad \text{return } b \end{array}$$

  最终 $Lx = b$ 的解 x 存储在 b 中.

- **解法 2 (前代法的标准形式, 数值线性代数, 算法 $1.1.1$):**

$$\begin{array}{l} \text{function} : y = \text{Forward\_Sweep}[L, b] \\ \qquad n \leftarrow \text{length}(b) \\ \qquad \text{for } i = 1 : n-1 \\ \qquad\qquad b(i) \leftarrow b(i)/L(i,i) \\ \qquad\qquad b(i+1:n) \leftarrow b(i+1:n) - b(i)L(i+1:n, i) \\ \qquad \text{end} \\ \qquad b(n) \leftarrow b(n)/L(n,n) \\ \qquad \text{return } b \end{array}$$

  最终 $Lx = b$ 的解 x 存储在 b 中.

The End