数值算法 Project

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

1. Helper Functions

(1) Householder 变换

以下算法在给定 $x \in \mathbb{R}^n$ 的条件下，构造 Householder 变换 $H = I_n - \beta v v^{\mathrm{T}} \in \mathbb{R}^{n \times n}$ 使得 $Hx = \|x\|_2 e_1$，
其中 e_1 代表 $\mathbb{R}^n$ 的第 1 个标准正交基向量.

```
function: $[v, \beta] = \text{Householder}(x)$
    $n = \text{length}(x)$
    $x = \frac{x}{\|x\|_\infty}$
    $v(2:n) = x(2:n)$
    $\sigma = x(2:n)^{\mathrm{T}} x(2:n)$
    if $\sigma = 0$
        $\beta = 0$
    else
        $\alpha = \sqrt{x(1)^2 + \sigma}$
        if $x(1) > 0$   (规避相消)
            $v(1) = -\frac{\sigma}{x(1) + \alpha}$
        else   ($x(1) \le 0$ 时无需规避相消)
            $v(1) = x(1) - \alpha$
        end
        $\beta = \frac{2v(1)^2}{v(1)^2 + \sigma}$
        $v = \frac{v}{v(1)}$
    end
```

其 MATLAB 实现为:

```matlab
function [v, beta] = Householder(x)
    % Householder function to compute the Householder vector v and scalar beta

    % Step 1: Get the length of the input vector
    n = length(x);

    % Step 2: Normalize x by its infinity norm
    if norm(x, "inf") == 0
        fprintf("Warning: input vector is zero!\n");
        beta = 0;
        v = zeros(n, 1);
        return
    else
        x = x / norm(x, "inf");
    end

    % Step 3: Initialize the Householder vector v
    v = zeros(n, 1);
    v(2:n) = x(2:n);  % Set v(2:n) = x(2:n)

    % Step 4: Compute the scalar sigma
    sigma = x(2:n)' * x(2:n);  % x(2:n)^{\mathrm T} * x(2:n)

    if sigma == 0
        % If sigma is zero, set beta to zero and return
        beta = 0;
    else
        % Step 5: Compute alpha
        alpha = sqrt(x(1)^2 + sigma);

        if x(1) > 0
            % Avoid cancellation when x(1) is positive
            v(1) = -(sigma / (x(1) + alpha));
        else
            % When x(1) <= 0, no cancellation avoidance needed
            v(1) = x(1) - alpha;
        end

        % Step 6: Compute beta
        beta = 2 * v(1)^2 / (v(1)^2 + sigma);

        % Step 7: Normalize v by its first element
        v = v / v(1);
    end
end
```

(2) Householder 变换的 WY 迭代累积

(Matrix Computation 算法 $5.1.2$)
设有 $r \le n$ 个 Householder 变换 $H_1, \dots, H_r$，其中:

$$H_j = I_n - \beta_j v^{(j)} (v^{(j)})^{\mathrm H}$$
$$v^{(j)} = [\underbrace{0, \dots, 0}_{j-1}, 1, v^{(j)}_{j+1}, \dots, v^{(j)}_n]^{\mathrm T}$$

我们可以计算得到 $W, Y \in \mathbb{C}^{n \times r}$ 满足 $H_1 \cdots H_r = I_n + WY^{\mathrm H}$:

```
$Y = v^{(1)}$
$W = -\beta_1 v^{(1)}$
for $j = 2 : r$
    $z = -\beta_j (I_n + WY^{\mathrm H}) v^{(j)} = -\beta_j [v^{(j)} + W(Y^{\mathrm H} v^{(j)})]$
    $W = [W, z]$
    $Y = [Y, v^{(j)}]$
end
```

(3) 可视化正交性损失的函数

```matlab
function visualize_orthogonality_loss(Q, titleStr)
    % Visualizes the componentwise loss of orthogonality |Q^{\mathrm H} Q - I_n|
    loss = Q' * Q - eye(size(Q, 2)); % Compute the loss
    figure; % Create a new figure window
    imagesc(log10(abs(loss))); % Display the absolute value of the loss
    colorbar; % Add colorbar to indicate scale
    title(titleStr);
    xlabel('Column Index');
    ylabel('Row Index');
    axis square; % Make the axes square for better visualization
end
```

(4) 生成指定条件数的矩阵的函数

```matlab
function A = generate_matrix(m, n, r, desired_cond_num)
    % Generates random complex matrix of size m x n
    % with desired condition number
    %
    % Inputs:
    %   - m: Number of rows
    %   - n: Number of columns
    %   - r: Number of non-zero singular values to consider
    %   - desired_cond_num: Desired condition number for the matrix
    %
    % Outputs:
    %   - A: complex matrix of size m x n with desired condition number

    % Step 1: Limit the number of singular values (r) to be within valid range
    r = max(0, min(r, min(m, n)));  % Ensure r does not exceed matrix dimensions
    % logspace creates values evenly spaced on a logarithmic scale
    % 1 is the lower limit (10^0), and desired_cond_num is the upper limit (10^log10(desired_cond_num))
    % This results in r values ranging from 1 to desired_cond_num, distributed exponentially
    sigma = logspace(0, log10(desired_cond_num), r);  % Generate r singular values

    % Step 2: Generate random unitary matrices U (m x m) and V (n x n)
    % Use QR decomposition on random complex matrices to create unitary matrices.
    % The random matrices are formed by adding real and imaginary parts.
    [U, ~] = qr(randn(m)); % QR decomposition for U
    [V, ~] = qr(randn(n)); % QR decomposition for V

    % Step 3: Construct the diagonal matrix of eigenvalues (D)
    % Initialize an m x n zero matrix and place the eigenvalues
    % (from the sigma vector) on the diagonal.
    D = zeros(m, n);  % Create an m x n matrix filled with zeros
    D(1:min(m,n), 1:min(m,n)) = diag(sigma);  % Place sigma on the diagonal

    % Step 4: Construct the ill-conditioned matrix A
    A = U * D * V';   % U is m x m, D is m x n, and V' is n x n

    % Step 5: Calculate the condition number
    cond_num = cond(A);  % Compute the condition number
    disp(['Condition number of the generated matrix: ', num2str(cond_num, '%.2e')]);
end
```

(5) 求解实系数一元二次方程的函数

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] = solve_quadratic(a, b, c, verbose)

    if nargin < 4
        verbose = false;  % Default: silent mode
    end

    % Check if the equation is actually quadratic
    if a == 0
        error('Coefficient a cannot be zero in a quadratic equation.');
    end

    % Calculate the discriminant
    M = [b, 2*a;
         2*c, b];
    D = det(M); % 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
    if verbose
        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
end
```

(6) 计算实二阶方阵的 Schur 分解

给定实二阶方阵:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

我们只需求解一元二次方程 $(\lambda - a)(\lambda - d) - bc = 0$ 即可得到 A 的特征值:

```
[lambda_1, lambda_2] = solve_quadratic(1, -(A(1,1)+A(2,2)), A(1,1) * A(2,2) - A(1,2) * A(2,1));
```

对于 A 的任意一个特征值 λ，求解以下线性方程组即可得到特征向量: (reference)

$$\begin{bmatrix} \lambda - a & -b \\ -c & \lambda - d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} (\lambda - a)x_1 - bx_2 \\ -cx_1 + (\lambda - d)x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- ① 若 $b = c = 0$，则我们有:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{cases} \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \text{if } \lambda = a \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \text{if } \lambda = d \end{cases}$$

- ② 若 $b \neq 0, c = 0$，则我们有:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b \\ \lambda - a \end{bmatrix}$$

- ③ 若 $b = 0, c \neq 0$，则我们有:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \lambda - d \\ c \end{bmatrix}$$

- ④ 若 $b, c \neq 0$，则上述方程组的信息是冗余的，以下两种选择均是合适的.

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b \\ \lambda - a \end{bmatrix} \text{ or } \begin{bmatrix} \lambda - d \\ c \end{bmatrix}$$

  为避免相消，若 λ 离 a 比离 d 更远，则我们选择前者; 否则选择后者.

MATLAB 代码如下:

```
function [X, lambda] = eig_2x2_real(A)
    % Check if the input matrix is 2x2
    if size(A, 1) ~= 2
        error('Dimension of input matrix should be 2!');
    end

    % Display the input matrix A
    disp("A:");
    disp(A);

    % Extract the elements of the 2x2 matrix A = [a b; c d]
    a = A(1,1); % element a (top-left)
    b = A(1,2); % element b (top-right)
    c = A(2,1); % element c (bottom-left)
    d = A(2,2); % element d (bottom-right)

    % Solve the quadratic equation for eigenvalues:
    [lambda_1, lambda_2] = solve_quadratic(1, -(a+d), a*d - b*c);

    % Check if the eigenvalues are the same (repeated eigenvalue)
    if lambda_1 == lambda_2
        fprintf("Warning: repeated eigenvalue!\n");
    % Swap eigenvalues to ensure that lambda_1 is the closer one to `a`
    elseif abs(a - lambda_1) > abs(a - lambda_2)
        temp = lambda_1;
        lambda_1 = lambda_2;
        lambda_2 = temp;
    end
    lambda = [lambda_1, lambda_2]';

    % Case 1: if both b = 0 and c = 0, we have trivial eigenvectors
    if b == 0 && c == 0
        x1 = [1, 0]'; % for eigenvalue lambda_1 = a
        x2 = [0, 1]'; % for eigenvalue lambda_2 = d

    % Case 2: if b != 0 and c != 0, solve using a general approach
    elseif b ~= 0 && c ~= 0
        x1 = [lambda_1 - d, c]'; % corresponding to eigenvalue lambda_1
        x2 = [b, lambda_2 - a]'; % corresponding to eigenvalue lambda_2

    % Case 3: if b = 0 but c != 0, solve using the second form
    elseif b == 0
        x1 = [lambda_1 - d, c]'; % corresponding to eigenvalue lambda_1
        x2 = [lambda_2 - d, c]'; % corresponding to eigenvalue lambda_2

    % Case 4: if b != 0 but c = 0, solve using the first form
    else
        x1 = [b, lambda_1 - a]'; % corresponding to eigenvalue lambda_1
        x2 = [b, lambda_2 - a]'; % corresponding to eigenvalue lambda_2
    end

    % Normalize the eigenvectors to make them unit vectors
    x1 = x1 / norm(x1);
    x2 = x2 / norm(x2);
    X = [x1, x2];
end
```

(7) 从 Schur 型中提取特征值

假设已通过 Francis 双位移隐式 QR 算法得到实 Schur 型 $T \in \mathbb{R}^{n\times n}$.
我们需要识别 T 的对角块，并计算其特征值:

- 若 T 的次对角线有两个连续的 0，则可识别出一个 1×1 对角块，它的元素就是 T 的特征值.
  (需要额外检查的边界情况: 左上角的 1×1 块)
- 若 T 的次对角线有两个 0 中间夹了一个非零元，则可识别出一个 2×2 对角块
  我们可以使用 $1.(6)$ 中编写的 `eig_2x2_real` 函数计算其特征值.
  (需要额外检查的边界情况: 左上角的 2×2 块)

MATLAB 代码如下:

```
function eigenvalue = eig_real_Schur(T)
    % This function extracts the eigenvalues from a real Schur form matrix T.
    n = size(T, 1);
    eigenvalue = zeros(n, 1);
    k = n;

    % Loop to extract eigenvalues from the Schur form
    while k >= 1
        if k == 1 || T(k, k-1) == 0 % Aha, a 1x1 diagonal block!
            eigenvalue(k) = T(k, k);
            k = k - 1;
        elseif k == 2 || T(k-1, k-2) == 0 % Aha, a 2x2 diagonal block!
            [~, eigenvalue(k-1:k)] = eig_2x2_real(T([k-1,k],[k-1,k]));
            k = k - 2;
        else % Oops, not convergent! T is not a real Schur form!
            error("Not convergent! T is not a real Schur form!");
        end
    end
    eigenvalue = sort(eigenvalue);
end
```

2. 上 Hessenberg 分解

使用 Householder 变换对 $A \in \mathbb{R}^{n\times n}$ 进行上 Hessenberg 化的算法:

```
Given $A \in \mathbb{R}^{n\times n}$
$W = 0_{n-1,n-2}$
$Y = 0_{n-1,n-2}$
$b = 0_{n-2}$
for $k = 1 : n-2$
    $[v, \beta] = \text{Householder}(A(k+1:n, k))$
    $A(k+1:n, k:n) = A(k+1:n, k:n) - (\beta v)(v^{\mathrm{T}} A(k+1:n, k:n))$
    $A(1:n, k+1:n) = A(1:n, k+1:n) - (A(1:n, k+1:n)v)(\beta v)^{\mathrm{T}}$
    $Y(k:n-1, k) = v$
    $b(k) = \beta$
end
for $k = 1 : n-2$
    $v = Y(k:n-1, k)$
    $\beta = b(k)$
    if $k = 1$
        $W(1:n-1, 1) = -\beta v$
    else
        $W(1:n-1, k) = W(1:n-1, 1:k-1)[Y(k:n-1, 1:k-1)^{\mathrm{T}} v]$
        $W(k:n-1, k) = W(k:n-1, k) + v$
        $W(1:n-1, k) = -\beta W(1:n-1, k)$
    end
end
$H = A$
$Q = \begin{bmatrix} 1 & \\ & I_{n-1} + WY^{\mathrm{T}} \end{bmatrix}$
```

尽管 WY 迭代累积 $n-2$ 个 Householder 变换相比直接累积 $n-2$ 个 Householder 变换并不会减少计算复杂度，但是可以增加 BLAS3 运算的比例，降低通讯开销，从而减少运行时间.
上述算法的 MATLAB 实现为:

```
function [H, Q] = Hessenberg_Reduction_Householder_WY(A)
    % Hessenberg form using WY accumulation of Householder transformations
    % Input:
    % - A: Complex matrix of size n x n
    % Output:
    % - H: Upper Hessenberg form of A
    % - Q: Orthogonal matrix such that A = Q * H * Q'

    % Initialize matrices W and Y for WY accumulation, and vector b for storing betas
    [n, ~] = size(A);
    W = zeros(n-1, n-2);  % Matrix to accumulate transformations for Q
    Y = zeros(n-1, n-2);  % Matrix to store Householder vectors
    b = zeros(1, n-2);    % Vector to store betas
    % Loop through each column (except the last two) for Householder reduction
    for k = 1:n-2
        % Step 1: Compute the Householder vector 'v' and scalar 'beta' for the current column
        [v, beta] = Householder(A(k+1:n, k));

        % Step 2: Apply the Householder transformation to zero out entries below the subdiagonal
        A(k+1:n, k:n) = A(k+1:n, k:n) - beta * v * (v' * A(k+1:n, k:n));
        A(1:n, k+1:n) = A(1:n, k+1:n) - (A(1:n, k+1:n) * v) * (beta * v)';

        % Step 3: Store the Householder vector in Y and the scalar beta in b
        Y(k:n-1, k) = v;  % Store the Householder vector for later use
        b(k) = beta;      % Store the scalar beta for the Householder transformation
    end

    % Loop to compute the W matrix, which accumulates the Householder transformations
    for k = 1:n-2
        % Step 4: Compute the W matrix for WY transformation
        if k == 1
            % For the first iteration, directly compute W
            W(1:n-1, 1) = -b(1) * Y(1:n-1, 1);
        else
            W(1:n-1, k) = W(1:n-1, 1:k-1) * (Y(k:n-1, 1:k-1)' * Y(k:n-1, k));
            W(k:n-1, k) = W(k:n-1, k) + Y(k:n-1, k);
            W(1:n-1, k) = -b(k) * W(1:n-1, k);
        end
    end

    % Step 5: Extract the upper Hessenberg matrix H from the modified matrix A
    H = triu(A,-1);  % The matrix A is now in upper Hessenberg form after applying Householder

    % Step 6: Compute the orthogonal matrix Q as per WY transformation
    Q = eye(n, n);
    Q(2:n, 2:n) = Q(2:n, 2:n) + W * Y';
end
```

3. Francis 双位移隐式 QR 迭代

Francis 双位移的 QR 迭代算法:

```
Given Hessenberg matrix $H \in \mathbb{R}^{n\times n}$ and orthogonal matrix $Q \in \mathbb{R}^{n\times n}$
$t = \mathrm{tr}\left(\begin{bmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{n,n} \end{bmatrix}\right) = h_{n-1,n-1} + h_{n,n}$
$s = \det\left(\begin{bmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{n,n} \end{bmatrix}\right) = h_{n-1,n-1}h_{n,n} - h_{n-1,n}h_{n,n-1}$
$\begin{cases} m_{11} = h_{11}^2 + h_{12}h_{21} - th_{11} + s \\ m_{21} = h_{21}(h_{11} + h_{22} - t) \\ m_{31} = h_{21}h_{32} \end{cases}$ (note that $Me_1 = \begin{bmatrix} m_{11} \\ m_{21} \\ m_{31} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} h_{11}^2 + h_{12}h_{21} - th_{11} + s \\ h_{21}(h_{11} + h_{22} - t) \\ h_{21}h_{32} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ where $M = H^2 - tH + sI$)
$[v, \beta] = \mathrm{Householder}\left(\begin{bmatrix} m_{11} \\ m_{21} \\ m_{31} \end{bmatrix}\right)$ (case of $k = 0$)
$H(1:3, 1:n) = H(1:3, 1:n) - (\beta v)(v^{\mathrm{T}} H(1:3, 1:n))$
if $n = 3$
    $H(1:3, 1:3) = H(1:3, 1:3) - (H(1:3, 1:3)v)(\beta v)^{\mathrm{T}}$
else
    $H(1:4, 1:3) = H(1:4, 1:3) - (H(1:4, 1:3)v)(\beta v)^{\mathrm{T}}$
end
$Q(1:\mathrm{end}, 1:3) = Q(1:\mathrm{end}, 1:3) - (Q(1:\mathrm{end}, 1:3)v)(\beta v)^{\mathrm{T}}$
for $k = 1 : n-4$
    $[v, \beta] = \mathrm{Householder}(H(k+1:k+3, k))$
    $H(k+1:k+3, k:n) = H(k+1:k+3, k:n) - (\beta v)(v^{\mathrm{T}} H(k+1:k+3, k:n))$
    $H(1:k+4, k+1:k+3) = H(1:k+4, k+1:k+3) - (H(1:k+4, k+1:k+3)v)(\beta v)^{\mathrm{T}}$
    $Q(1:\mathrm{end}, k+1:k+3) = Q(1:\mathrm{end}, k+1:k+3) - (Q(1:\mathrm{end}, k+1:k+3)v)(\beta v)^{\mathrm{T}}$
end
if $n \neq 3$
    $[v, \beta] = \mathrm{Householder}(H(n-2:n, n-3))$ (case of $k = n-3$)
    $H(n-2:n, n-3:n) = H(n-2:n, n-3:n) - (\beta v)(v^{\mathrm{T}} H(n-2:n, n-3:n))$
    $H(1:n, n-2:n) = H(1:n, n-2:n) - (H(1:n, n-2:n)v)(\beta v)^{\mathrm{T}}$
    $Q(1:\mathrm{end}, n-2:n) = Q(1:\mathrm{end}, n-2:n) - (Q(1:\mathrm{end}, n-2:n)v)(\beta v)^{\mathrm{T}}$
end
$[v, \beta] = \mathrm{Householder}(H(n-1:n, n-2))$ (case of $k = n-2$)
$H(n-1:n, n-2:n) = H(n-1:n, n-2:n) - (\beta v)(v^{\mathrm{T}} H(n-1:n, n-2:n))$
$H(1:n, n-1:n) = H(1:n, n-1:n) - (H(1:n, n-1:n)v)(\beta v)^{\mathrm{T}}$
$Q(1:\mathrm{end}, n-1:n) = Q(1:\mathrm{end}, n-1:n) - (Q(1:\mathrm{end}, n-1:n)v)(\beta v)^{\mathrm{T}}$
```

上述算法的运算量为 $20n^2$ (包括正交变换的累积)
其 MATLAB 实现为:

```matlab
function [H, Q] = Francis_Double_Shift_QR_Iteration(H, Q)
    % Francis Double Shift QR Iteration
    % Step 1: Compute the trace and determinant for the 2x2 bottom right submatrix
    n = size(H, 1);
    t = H(n-1, n-1) + H(n, n); % Trace of the 2x2 bottom-right block
    s = H(n-1, n-1) * H(n, n) - H(n-1, n) * H(n, n-1); % Determinant of the 2x2 block

    % Step 2: Define the components of Me1 vector
    m11 = H(1, 1)^2 + H(1, 2) * H(2, 1) - t * H(1, 1) + s;
    m21 = H(2, 1) * (H(1, 1) + H(2, 2) - t);
    m31 = H(2, 1) * H(3, 2);

    % Step 3: Apply Householder transformation to Me1 (initial case k=0)
    [v, beta] = Householder([m11; m21; m31]);
    % Apply the transformation to H
    H(1:3, 1:n) = H(1:3, 1:n) - beta * (v * (v' * H(1:3, 1:n)));
    if n == 3
        H(1:3, 1:3) = H(1:3, 1:3) - (H(1:3, 1:3) * v) * (beta * v)';
    else
        H(1:4, 1:3) = H(1:4, 1:3) - (H(1:4, 1:3) * v) * (beta * v)';
    end
    Q(:, 1:3) = Q(:, 1:3) - (Q(:, 1:3) * v) * (beta * v)';

    % Step 4: Iterate for k = 1 to n-4
    for k = 1:n-4
        [v, beta] = Householder(H(k+1:k+3, k));
        H(k+1:k+3, k:n) = H(k+1:k+3, k:n) - beta * (v * (v' * H(k+1:k+3, k:n)));
        H(1:k+4, k+1:k+3) = H(1:k+4, k+1:k+3) - (H(1:k+4, k+1:k+3) * v) * (beta * v)';
        Q(:, k+1:k+3) = Q(:, k+1:k+3) - (Q(:, k+1:k+3) * v) * (beta * v)';
    end

    % Step 5: Handle case for k = n-3
    if n ~= 3
        [v, beta] = Householder(H(n-2:n, n-3));
        H(n-2:n, n-3:n) = H(n-2:n, n-3:n) - beta * (v * (v' * H(n-2:n, n-3:n)));
        H(1:n, n-2:n) = H(1:n, n-2:n) - (H(1:n, n-2:n) * v) * (beta * v)';
        Q(:, n-2:n) = Q(:, n-2:n) - (Q(:, n-2:n) * v) * (beta * v)';
    end

    % Step 6: Handle case for k = n-2
    [v, beta] = Householder(H(n-1:n, n-2));
    H(n-1:n, n-2:n) = H(n-1:n, n-2:n) - beta * (v * (v' * H(n-1:n, n-2:n)));
    H(1:n, n-1:n) = H(1:n, n-1:n) - (H(1:n, n-1:n) * v) * (beta * v)';
    Q(:, n-1:n) = Q(:, n-1:n) - (Q(:, n-1:n) * v) * (beta * v)';
end
```

现在我们对上述算法进行修改，以适配隐式 QR 算法中对它的调用.
假设 $H(l+1:u, l+1:u)$ 是我们实际希望进行 Francis 双位移隐式 QR 迭代的不可约上 Hessenberg 矩阵.
我们针对这个子矩阵计算 Householder 变换，然后应用到 H, Q 整体上:

Given Hessenberg matrix $H \in \mathbb{R}^{n\times n}$, orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ and $0 \le l < u \le n$

$$t = \operatorname{tr}\left(\begin{bmatrix} h_{u-1,u-1} & h_{u-1,u} \\ h_{u,u-1} & h_{u,u} \end{bmatrix}\right) = h_{u-1,u-1} + h_{u,u}$$

$$s = \det\left(\begin{bmatrix} h_{u-1,u-1} & h_{u-1,u} \\ h_{u,u-1} & h_{u,u} \end{bmatrix}\right) = h_{u-1,u-1}h_{u,u} - h_{u-1,u}h_{u,u-1}$$

$$\begin{cases} m_{11} = h_{l+1,l+1}^2 + h_{l+1,l+2}h_{l+2,l+1} - th_{l+1,l+1} + s \\ m_{21} = h_{l+2,l+1}(h_{l+1,l+1} + h_{l+2,l+2} - t) \\ m_{31} = h_{l+2,l+1}h_{l+3,l+2} \end{cases}$$

$$(\text{note that } Me_1 = \begin{bmatrix} m_{11} \\ m_{21} \\ m_{31} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} h_{l+1,l+1}^2 + h_{l+1,l+2}h_{l+2,l+1} - th_{l+1,l+1} + s \\ h_{l+2,l+1}(h_{l+1,l+1} + h_{l+2,l+2} - t) \\ h_{l+2,l+1}h_{l+3,l+2} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \text{ where } M = H_{22}^2 - tH_{22} + sI_{u-l})$$

```
$[v, \beta] = \text{Householder}\left(\begin{bmatrix} m_{11} \\ m_{21} \\ m_{31} \end{bmatrix}\right)$    (case of $k = 0$)
$H(l+1:l+3, l+1:n) = H(l+1:l+3, l+1:n) - (\beta v)(v^{\mathrm{T}}H(l+1:l+3, l+1:n))$
if $u - l = 3$
    $H(1:l+3, l+1:l+3) = H(1:l+3, l+1:l+3) - (H(1:l+3, l+1:l+3))v)(\beta v)^{\mathrm{T}}$
else
    $H(1:l+4, l+1:l+3) = H(1:l+4, l+1:l+3) - (H(1:l+4, l+1:l+3))v)(\beta v)^{\mathrm{T}}$
end
$Q(1:n, l+1:l+3) = Q(1:n, l+1:l+3) - (Q(1:n, l+1:l+3)v)(\beta v)^{\mathrm{T}}$
```

```
for $k = 1 : u - l - 4$
    $[v, \beta] = \text{Householder}(H(l+k+1:l+k+3, l+k))$
    $H(l+k+1:l+k+3, l+k:n) = H(l+k+1:l+k+3, l+k:n) - (\beta v)(v^{\mathrm{T}}H(l+k+1:l+k+3, l+k:n))$
    $H(1:l+k+4, l+k+1:l+k+3) = H(1:l+k+4, l+k+1:l+k+3) - (H(1:l+k+4, l+k+1:l+k+3)v)(\beta v)^{\mathrm{T}}$
    $Q(1:n, l+k+1:l+k+3) = Q(1:n, l+k+1:l+k+3) - (Q(1:n, l+k+1:l+k+3)v)(\beta v)^{\mathrm{T}}$
end
```

```
if $u - l \ne 3$
    $[v, \beta] = \text{Householder}(H(u-2:u, u-3))$    (case of $k = u - l - 3$)
    $H(u-2:u, u-3:n) = H(u-2:u, u-3:n) - (\beta v)(v^{\mathrm{T}}H(u-2:u, u-3:n))$
    $H(1:u, u-2:u) = H(1:u, u-2:u) - (H(1:u, u-2:u)v)(\beta v)^{\mathrm{T}}$
    $Q(1:n, u-2:u) = Q(1:n, u-2:u) - (Q(1:n, u-2:u)v)(\beta v)^{\mathrm{T}}$
end
```

```
$[v, \beta] = \text{Householder}(H(u-1:u, u-2))$    (case of $k = u - l - 2$)
$H(u-1:u, u-2:n) = H(u-1:u, u-2:n) - (\beta v)(v^{\mathrm{T}}H(u-1:u, u-2:n))$
$H(1:u, u-1:u) = H(1:u, u-1:u) - (H(1:u, u-1:u)v)(\beta v)^{\mathrm{T}}$
$Q(1:n, u-1:u) = Q(1:n, u-1:u) - (Q(1:n, u-1:u)v)(\beta v)^{\mathrm{T}}$
```

MATLAB 代码为:

```matlab
function [H, Q] = Generalized_Francis_Double_Shift_QR_Iteration(H, Q, l, u)
    % Generalized Francis Double Shift QR Iteration
    % Inputs:
    %   H - Hessenberg matrix (n x n)
    %   Q - Orthogonal matrix (n x n)
    %   l, u - Indices defining the block for the Householder transformation
    % Output:
    %   H - Updated Hessenberg matrix
    %   Q - Updated orthogonal matrix

    % Step 1: Compute the trace and determinant for the 2x2 bottom-right submatrix
    n = size(H, 1);
    t = H(u-1, u-1) + H(u, u);  % Trace of the 2x2 bottom-right block
    s = H(u-1, u-1) * H(u, u) - H(u-1, u) * H(u, u-1);  % Determinant of the 2x2 block

    % Step 2: Define the components of Me1 vector
    m11 = H(l+1, l+1)^2 + H(l+1, l+2) * H(l+2, l+1) - t * H(l+1, l+1) + s;
    m21 = H(l+2, l+1) * (H(l+1, l+1) + H(l+2, l+2) - t);
    m31 = H(l+2, l+1) * H(l+3, l+2);

    % Step 3: Apply Householder transformation to Me1 (initial case k=0)
    Me1 = [m11; m21; m31];
    [v, beta] = Householder(Me1);

    % Apply the transformation to H
    H(l+1:l+3, l+1:n) = H(l+1:l+3, l+1:n) - beta * (v * (v' * H(l+1:l+3, l+1:n)));
    if u - l == 3
        H(1:l+3, l+1:l+3) = H(1:l+3, l+1:l+3) - (H(1:l+3, l+1:l+3) * v) * (beta * v)';
    else
        H(1:l+4, l+1:l+3) = H(1:l+4, l+1:l+3) - (H(1:l+4, l+1:l+3) * v) * (beta * v)';
    end
    Q(:, l+1:l+3) = Q(:, l+1:l+3) - (Q(:, l+1:l+3) * v) * (beta * v)';

    % Step 4: Iterate for k = 1 to u-l-4
    for k = 1:u-l-4
        % Compute Householder for each block
        [v, beta] = Householder(H(l+k+1:l+k+3, l+k));

        % Apply the transformation to H
        H(l+k+1:l+k+3, l+k:n) = H(l+k+1:l+k+3, l+k:n) - beta * (v * (v' * H(l+k+1:l+k+3, l+k:n)));
        H(1:l+k+4, l+k+1:l+k+3) = H(1:l+k+4, l+k+1:l+k+3) - (H(1:l+k+4, l+k+1:l+k+3) * v) * (beta * v)';
        Q(:, l+k+1:l+k+3) = Q(:, l+k+1:l+k+3) - (Q(:, l+k+1:l+k+3) * v) * (beta * v)';
    end

    % Step 5: Handle case for k = u-l-3
    if u - l ~= 3
        [v, beta] = Householder(H(u-2:u, u-3));
        H(u-2:u, u-3:n) = H(u-2:u, u-3:n) - beta * (v * (v' * H(u-2:u, u-3:n)));
        H(1:u, u-2:u) = H(1:u, u-2:u) - (H(1:u, u-2:u) * v) * (beta * v)';
        Q(:, u-2:u) = Q(:, u-2:u) - (Q(:, u-2:u) * v) * (beta * v)';
    end

    % Step 6: Handle case for k = u-l-2
    [v, beta] = Householder(H(u-1:u, u-2));
    H(u-1:u, u-2:n) = H(u-1:u, u-2:n) - beta * (v * (v' * H(u-1:u, u-2:n)));
    H(1:u, u-1:u) = H(1:u, u-1:u) - (H(1:u, u-1:u) * v) * (beta * v)';
    Q(:, u-1:u) = Q(:, u-1:u) - (Q(:, u-1:u) * v) * (beta * v)';

end
```

4. 隐式 QR 算法

隐式 QR 算法用于计算实方阵 $A \in \mathbb{R}^{n\times n}$ 的实 Schur 分解 $A = QTQ^{\mathrm{T}}$,
其中实 Schur 型 $T \in \mathbb{R}^{n\times n}$ 为分块上三角阵.
其对角块要么是 1×1 的 (对应 A 的一个实特征值)，要么是 2×2 的 (对应 A 的一对共轭的复特征值).

隐式 QR 算法:
给定方阵 $A \in \mathbb{R}^{n\times n}$.

- (1) **上 Hessenberg 化:**
  对 $A \in \mathbb{R}^{n\times n}$ 应用 Householder 变换得到上 Hessenberg 分解 $A = QHQ^{\mathrm{T}}$，记 $H := [h_{ij}]_{i,j=1}^{n}$.

```
[H, Q] = Hessenberg_Reduction_Householder_WY(A);
```

- (2) **迭代步骤:**
  - ① 将所有满足条件 $|h_{i+1,i}| \leq (|h_{i,i}| + |h_{i+1,i+1}|)\text{eps}$ 的次对角元 $h_{i+1,i}$ $(i = 1, \ldots, n-1)$ 置为零，其中 eps 代表机器精度.

```
for i = 1:n-1
    if abs(H(i+1, i)) <= eps * (abs(H(i, i)) + abs(H(i+1, i+1)))
        H(i+1, i) = 0;  % Zero out subdiagonal elements that are sufficiently small
    end
end
```

  - ② 将 H 划分为:

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ & H_{22} & H_{23} \\ & & H_{33} \end{bmatrix},$$

其中:

$$\begin{cases} H_{11} \in \mathbb{R}^{l\times l} \\ H_{22} \in \mathbb{R}^{(u-l)\times(u-l)} \\ H_{33} \in \mathbb{R}^{(n-u)\times(n-u)}. \end{cases}$$

最小化 u 使得 $H_{33} \in R^{(n-u)\times(n-u)}$ 为对角块为 1×1 或 2×2 方阵的块上三角阵.
最小化 l 使得 $H_{22} \in \mathbb{R}^{(u-l)\times(u-l)}$ 为不可约的上 Hessenberg 矩阵.

```
[l, u] = Partition_Blocks(diag(H, -1), u);  % Find the indices l and u for partitioning
```

  - ③ 收敛性检查:
若 $u = 0$，则迭代终止; 否则移步 ④.

```
if u == 0
    break;  % If u == 0, convergence is achieved
end
```

  - ④ 若 $u - l \neq 2$，则移步 ⑤.
若 $u - l = 2$，则需要处理一个具有实特征值的 2×2 块.
我们需要计算一个正交变换将其转换为一个 2 阶上三角阵.
处理完成后，将 u 减去 2.
再次判断 u 是否为零 (若 $u = 0$，则迭代终止; 否则移步 ①).

```
% Handle the case of a 2x2 real eigenvalue block
if u - l == 2
    % Construct rotation matrix using the first eigenvector
    [X, ~] = eig_2x2_real(H(u-1:u, u-1:u));
    Q0 = [X(1,1), -X(2,1);
    X(2,1), X(1,1)];

    % Apply the rotation matrix
    H(u-1:u, u-1:n) = Q0' * H(u-1:u, u-1:n);
    H(1:u, u-1:u) = H(1:u, u-1:u) * Q0;
    H(u, u-1) = 0;
    Q(1:n, u-1:u) = Q(1:n, u-1:u) * Q0;
    u = u - 2;

    % Convergence check
    if u == 0
        break   % If u == 0, convergence is achieved
    end
end
```

- ⑤ 对 H_{22} 进行 Francis 双位移隐式 QR 迭代:

```
[H, Q] = Generalized_Francis_Double_Shift_QR_Iteration(H, Q, l, u);
```

实际计算的经验表明:
隐式 QR 算法每分离出一个 1×1 或 2×2 子矩阵平均需要 2 次迭代，因此总运算量约为 $25n^3$.

其中 `Partition_Blocks` 函数的实现如下:

```
function [l, u] = Partition_Blocks(H, u)
    % Partition_Blocks determines the partition of the Hessenberg matrix block
    % and identifies the block indices based on the subdiagonal elements of H.
    % This function updates the indices `l` and `u` which correspond to
    % the partition of H into the blocks H_{11}, H_{22}, and H_{33}.
    %
    % Input:
    %   - H: A Hessenberg Matrix
    %   - u: The end index of the block H_{22} from the previous iteration.
    %
    % Output:
    %   - l: The end index of the block H_{11}.
    %   - u: The updated end index of the block H_{22}

    while true
        % Check if u is less than or equal to 1, if so, we set both l and u to 0.
        if u <= 1 % base case to stop the iteration
            u = 0;
            l = 0;
            return
        % If the element H(u, u-1) is zero, move u left by one position.
        elseif H(u, u-1) == 0 % Aha, a 1x1 diagonal block!
            u = u - 1;
        % If the element H(u-1, u-2) is zero
        % check whether the 2x2 diagonal block has complex eigenvalues
        elseif u == 2 || H(u-1, u-2) == 0
            % Compute Delta
            b = H(u-1, u-1) + H(u, u);
            ac = H(u-1, u-1) * H(u, u) - H(u-1, u) * H(u, u-1);
            Delta = b^2 - 4 * ac;

            if Delta < 0 % It has complex eigenvalues, life will be much easier!
                u = u - 2;
            else % It has real eigenvalues, we should transform it into a 2x2 upper-triangle matrix!
                l = u - 2;
                return
            end
        % If neither of the above conditions is true, exit the loop.
        else % Oops, neither 1x1 nor 2x2 diagonal block is found!
            break;
        end
    end

    % Set l to u - 3, since h(u-2) and h(u-1) are all zero, guaranteed by the previous loop
    l = u - 3;
    % Find the indices of the zeros in h(1:l) and update l accordingly.
    h = diag(H, -1);
    zero_indices = find(h(1:l) == 0);
    % If there are no zeros in this range, set l to 0.
    if isempty(zero_indices)
        l = 0;
    else
        % Otherwise, set l to the index of the last zero element in h(1:l).
        l = zero_indices(end);
    end
end
```

隐式 QR 算法的实现如下:

```
function [T, Q] = Implicit_QR_Algorithm(A)
    % Implicit QR Algorithm for computing real Schur decomposition
    % Inputs:
    %   A - Real square matrix (n x n)
    % Outputs:
    %   T - Schur matrix (upper triangular with 1x1 or 2x2 blocks)
    %   Q - Orthogonal matrix (n x n) such that A = Q*T*Q'

    n = size(A, 1);
    max_iter = 5 * n;  % Set the maximum number of iterations
    u = n;             % Initialize the end index of the unreduced submatrix H_{22} to be n

    % Step 1: Reduce A to Hessenberg form using Householder transformations
    [H, Q] = Hessenberg_Reduction_Householder_WY(A);

    % Step 2: Francis double shift implicit QR iteration
    for iteration = 1:max_iter
        fprintf("Iteration %d:\n", iteration);
        % Substep 2.1: Convergence condition: |h_{i+1,i}| <= eps * (|h_{i,i}| + |h_{i+1,i+1}|)
        for i = 1:n-1
            if H(i+1, i) == 0
                continue
            elseif abs(H(i+1, i)) <= eps * (abs(H(i, i)) + abs(H(i+1, i+1)))
                fprintf("zero out subdiagonal: (%d, %d)\n", i+1, i);
                H(i+1, i) = 0;  % Zero out subdiagonal elements that are sufficiently small
            end
        end

        % Substep 2.2: Partition H into blocks
        [l, u] = Partition_Blocks(H, u);  % Find the indices l and u for partitioning
        fprintf("Partition [l, u] = [%d, %d]\n", l, u);

        % Substep 2.3: Convergence check
        if u == 0
            fprintf("Converged on after %d iterations!\n", iteration);
            break;  % If u == 0, convergence is achieved

        % Substep 2.4: Handle the case of a 2x2 real eigenvalue block
        elseif u - l == 2
            fprintf("Handling 2x2 real eigenvalue block!\n");

            % Construct rotation matrix using the first eigenvector
            [X, ~] = eig_2x2_real(H(u-1:u, u-1:u));
            Q0 = [X(1,1), -X(2,1);
                  X(2,1), X(1,1)];

            % Apply the rotation matrix
            H(u-1:u, u-1:n) = Q0' * H(u-1:u, u-1:n);
            H(1:u, u-1:u) = H(1:u, u-1:u) * Q0;
            H(u, u-1) = 0;
            Q(1:n, u-1:u) = Q(1:n, u-1:u) * Q0;
            u = u - 2;

            % Convergence check
            if u == 0
                fprintf("Converged on after %d iterations!\n", iteration);
                break   % If u == 0, convergence is achieved
            end

        % Substep 2.5: Apply Francis Double Shift QR iteration to the submatrix H_{22}
        else
            [H, Q] = Generalized_Francis_Double_Shift_QR_Iteration(H, Q, l, u);
        end
        fprintf("\n");
    end

    % Step 3: Output the Schur matrix T
    T = triu(H, -1);
end
```

5. 验证

(1) 正确性验证

我们需要验证:

- ① 正交性损失 $\|Q^{\mathrm{T}}Q - I\|_{\mathrm{F}}$ 是机器精度级别的
- ② 向后误差 $\frac{\|Q^{\mathrm{T}}AQ-T\|_{\mathrm{F}}}{\|A\|_{\mathrm{F}}}$ 是机器精度级别的
- ③ 矩阵 T 具有实 Schur 型的形式 (即分块上三角阵，对角分块要么是 1×1 的，要么是 2×2 的)

我们首先使用教材中给出的 5 阶方阵进行验证:

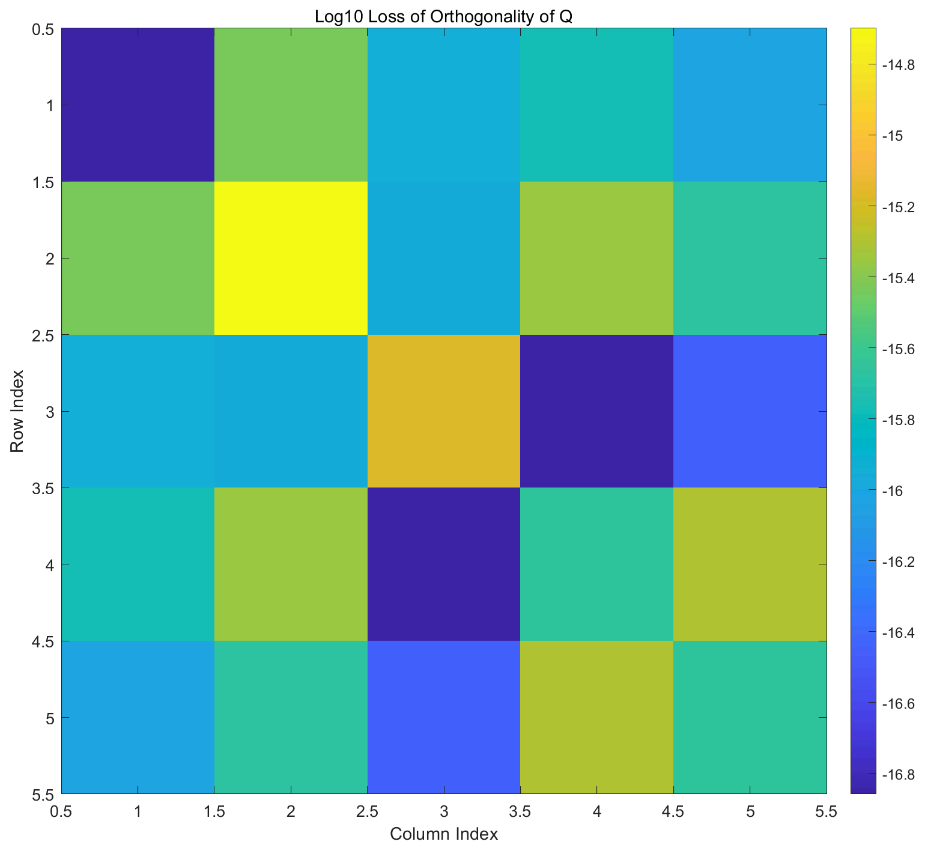
```
clc; clear;
format short e
A = [2, 3, 4, 5, 6;
     4, 4, 5, 6, 7;
     0, 3, 6, 7, 8;
     0, 0, 2, 8, 9;
     0, 0, 0, 1, 10];

% Apply the Hessenberg reduction
[T, Q] = Implicit_QR_Algorithm(A);

disp("T:")
disp(T);

% Compare the difference of eigenvalue with eig() function
disp("Difference of eigenvalue with eig() function:");
eig_T = sort(eig_real_Schur(T));
eig_A = sort(eig(A));
disp(norm(abs(eig_A - eig_T) ./ eig_A, "inf"))

% Display the Frobenius norm of Q' * A * Q - H
disp("Frobenius norm of Q' * A * Q - T:");
disp(norm(Q' * A * Q - T, 'fro') / norm(A, 'fro'));  % Compute Frobenius norm for backward error

% Display the Frobenius norm of Q' * Q - In
disp("Frobenius norm of Q' * Q - In:");
disp(norm(Q' * Q - eye(size(A,1)), 'fro')); % Check orthogonality of Q
```

```
% Visualize the log10 loss of orthogonality of Q using the custom function
visualize_orthogonality_loss(Q, 'Log10 Loss of Orthogonality of Q');
```

运行结果: (附加详细调试信息)

```
Iteration 1:
Partition [l, u] = [0, 5]

Iteration 2:
Partition [l, u] = [0, 5]

Iteration 3:
Partition [l, u] = [0, 5]

Iteration 4:
Partition [l, u] = [0, 5]

Iteration 5:
Partition [l, u] = [0, 5]

Iteration 6:
Partition [l, u] = [0, 5]

Iteration 7:
Partition [l, u] = [0, 5]

Iteration 8:
zero out subdiagonal: (4, 3)
Partition [l, u] = [3, 5]
Handling 2x2 real eigenvalue block!

Iteration 9:
Partition [l, u] = [0, 3]

Iteration 10:
Partition [l, u] = [0, 3]

Iteration 11:
Partition [l, u] = [0, 3]

Iteration 12:
zero out subdiagonal: (2, 1)
Partition [l, u] = [1, 3]
Handling 2x2 real eigenvalue block!

Iteration 13:
Partition [l, u] = [0, 0]
Converged after 13 iterations!
T:
  -3.3542e-01   2.0716e+00   9.7846e-01  -3.4348e-01   4.0987e-01
            0   1.5014e+00   1.6432e+00   2.1778e+00   9.3169e-01
            0            0   1.4154e+01   1.5675e+01   5.6250e+00
            0            0            0   9.5248e+00   5.3666e+00
            0            0            0            0   5.1552e+00

Difference of eigenvalue with eig() function:
   3.6972e-15

Frobenius norm of Q' * A * Q - T:
   5.8410e-16
```

```
Frobenius norm of Q' * Q - In:
   2.4330e-15
```

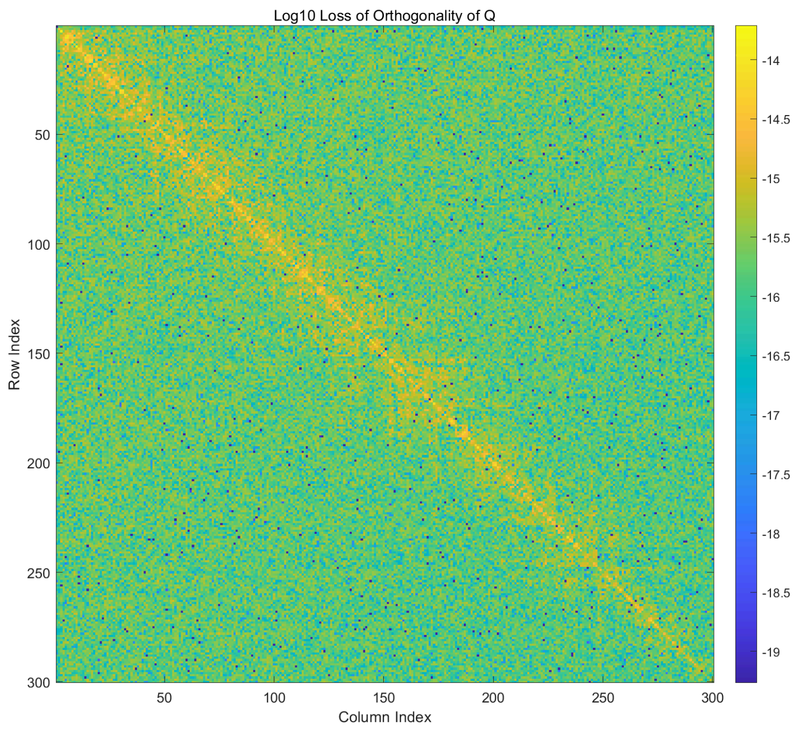


现在使用一个随机生成的 300 阶方阵进行验证:

```
clc; clear;
format short e
rng(51);
n = 300;
A = rand(n, n);

% Apply the Hessenberg reduction
[T, Q] = Implicit_QR_Algorithm(A);

% Compare the difference of eigenvalue with eig() function
disp("Difference of eigenvalue with eig() function:");
eig_T = sort(eig_real_Schur(T));
eig_A = sort(eig(A));
disp(norm(abs(eig_A - eig_T) ./ eig_A, "inf"))

% Display the Frobenius norm of Q' * A * Q - H
disp("Frobenius norm of Q' * A * Q - T:");
disp(norm(Q' * A * Q - T, 'fro') / norm(A, 'fro'));  % Compute Frobenius norm for backward error

% Display the Frobenius norm of Q' * Q - In
disp("Frobenius norm of Q' * Q - In:");
disp(norm(Q' * Q - eye(size(A,1)), 'fro')); % Check orthogonality of Q

% Visualize the log10 loss of orthogonality of Q using the custom function
visualize_orthogonality_loss(Q, 'Log10 Loss of Orthogonality of Q');
```

运行结果:

```
Converged after 557 iterations!
Difference of eigenvalue with eig() function:
   2.3682e-14

Frobenius norm of Q' * A * Q - T:
   5.2877e-15

Frobenius norm of Q' * Q - In:
   1.3890e-13
```



(2) 稳定性验证

我们使用 `generate_matrix()` 函数生成给定条件数的实方阵来测试隐式 QR 算法的稳定性.
我们发现该算法向后稳定，向前不稳定 (但对于条件数小于 10^{11} 的 100 阶病态矩阵仍有不错的稳定性).

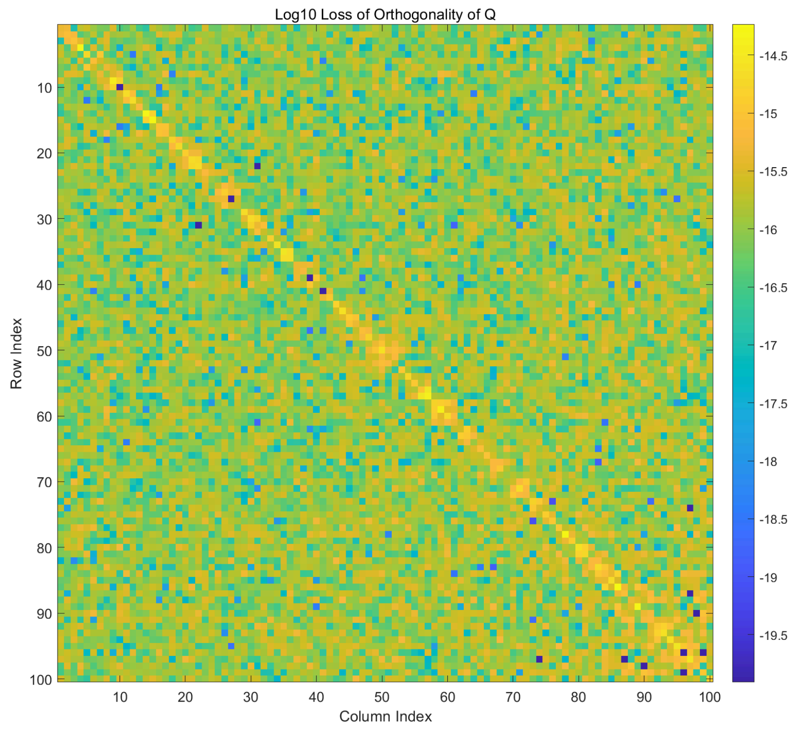
首先测试条件数为 10^{11} 的 100 阶病态矩阵:

```
clc; clear;
format short e
rng(51);
n = 100;
desired_cond_num = 1e11;
A = generate_matrix(n, n, n, desired_cond_num);

% Apply the Hessenberg reduction
[T, Q] = Implicit_QR_Algorithm(A);

% Compare the difference of eigenvalue with eig() function
disp("Difference of eigenvalue with eig() function:");
eig_T = sort(eig_real_Schur(T));
eig_A = sort(eig(A));
disp(norm(abs(eig_A - eig_T) ./ eig_A, "inf"))

% Display the Frobenius norm of Q' * A * Q - H
```

```
disp("Frobenius norm of Q' * A * Q - T:");
disp(norm(Q' * A * Q - T, 'fro') / norm(A, 'fro'));  % Compute Frobenius norm for backward error

% Display the Frobenius norm of Q' * Q - In
disp("Frobenius norm of Q' * Q - In:");
disp(norm(Q' * Q - eye(size(A,1)), 'fro')); % Check orthogonality of Q

% Visualize the log10 loss of orthogonality of Q using the custom function
visualize_orthogonality_loss(Q, 'Log10 Loss of Orthogonality of Q');
```

运行结果:

```
Condition number of the generated matrix: 1.00e+11
Converged on after 124 iterations!
Difference of eigenvalue with eig() function:
   1.2351e-06

Frobenius norm of Q' * A * Q - T:
   2.3789e-15

Frobenius norm of Q' * Q - In:
   2.8717e-14
```



其次绘制向前误差、向后误差和正交性损失关于条件数的 log - log 图像:

```
clc; clear;
rng(51);
n = 100;
condition_numbers = logspace(1, 15, 15);  % Logarithmic range of condition numbers

% Pre-allocate arrays to store results
eig_diff = zeros(size(condition_numbers));
backward_error = zeros(size(condition_numbers));
orthogonality_loss = zeros(size(condition_numbers));

% Loop over different condition numbers
```

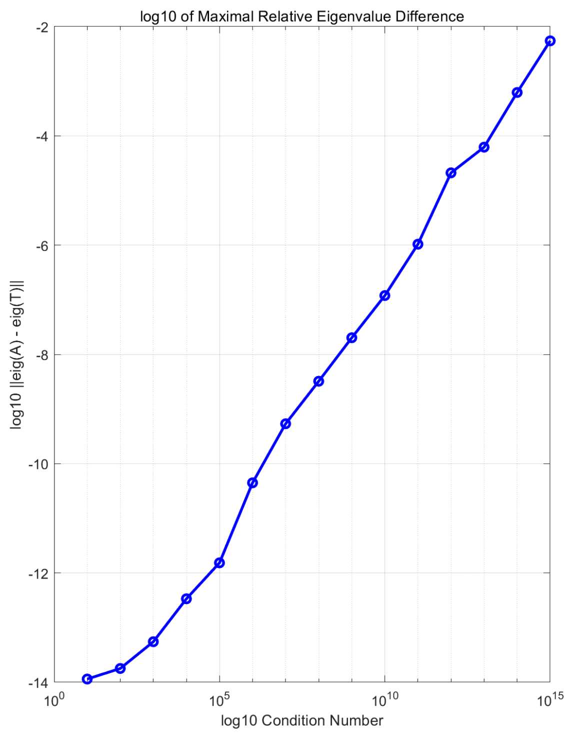
```
for i = 1:length(condition_numbers)
    desired_cond_num = condition_numbers(i);

    % Generate matrix with the desired condition number
    A = generate_matrix(n, n, n, desired_cond_num);

    % Apply Implicit QR Algorithm
    [T, Q] = Implicit_QR_Algorithm(A);

    % Compute the eigenvalue difference
    eig_T = sort(eig_real_Schur(T));
    eig_A = sort(eig(A));
    eig_diff(i) = norm(abs(eig_A - eig_T)./eig_A, "inf");

    % Compute the Frobenius norm of the backward error
    backward_error(i) = norm(Q' * A * Q - T, 'fro') / norm(A, 'fro');

    % Compute the Frobenius norm of orthogonality loss
    orthogonality_loss(i) = norm(Q' * Q - eye(n), 'fro');
end

% Plot the results
figure;

% Plot the eigenvalue difference
subplot(1, 2, 1);  % Create a subplot for eigenvalue difference
semilogx(condition_numbers, log10(eig_diff), 'b-o', 'LineWidth', 2);
title('log10 of Maximal Relative Eigenvalue Difference');
xlabel('log10 Condition Number');
ylabel('log10 ||eig(A) - eig(T)||');
grid on;

% Plot the backward error and orthogonality loss
subplot(1, 2, 2);  % Create a subplot for backward error and orthogonality loss
semilogx(condition_numbers, log10(backward_error), 'r-o', 'LineWidth', 2);
hold on;
semilogx(condition_numbers, log10(orthogonality_loss), 'g-o', 'LineWidth', 2);
title('log10 of Backward Error and Orthogonality Loss');
xlabel('log10 Condition Number');
ylabel('log10 Error');
legend('Backward Error', 'Orthogonality Loss');
grid on;

% Adjust figure layout
set(gcf, 'Position', [100, 100, 1200, 500]);
```

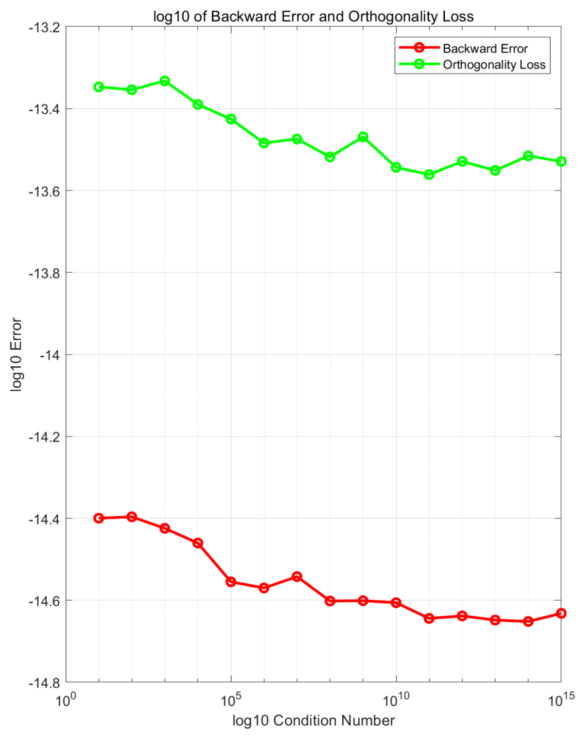
运行结果:

```
Condition number of the generated matrix: 1.00e+01
Converged after 187 iterations!
Condition number of the generated matrix: 1.00e+02
Converged after 162 iterations!
Condition number of the generated matrix: 1.00e+03
Converged after 155 iterations!
Condition number of the generated matrix: 1.00e+04
Converged after 127 iterations!
Condition number of the generated matrix: 1.00e+05
Converged after 139 iterations!
Condition number of the generated matrix: 1.00e+06
Converged on after 113 iterations!
Condition number of the generated matrix: 1.00e+07
```

```
Converged after 127 iterations!
Condition number of the generated matrix: 1.00e+08
Converged after 136 iterations!
Condition number of the generated matrix: 1.00e+09
Converged after 131 iterations!
Condition number of the generated matrix: 1.00e+10
Converged after 138 iterations!
Condition number of the generated matrix: 1.00e+11
Converged after 141 iterations!
Condition number of the generated matrix: 1.00e+12
Converged on after 142 iterations!
Condition number of the generated matrix: 1.00e+13
Converged after 129 iterations!
Condition number of the generated matrix: 1.00e+14
Converged after 128 iterations!
Condition number of the generated matrix: 1.00e+15
Converged after 121 iterations!
```

至于向后误差为什么下降，邵老师如是说:
问题越病态，越容易找到一个小扰动来 "解释" 误差，而良态问题没有这个机会 (或者说很小)





(3) 时间复杂度验证

我们使用 `Implicit_QR_Algorithm()` 函数计算不同阶数的随机矩阵的实 Schur 型，
并记录运行时间，绘制图像以验证其 $O(n^3)$ 的时间复杂度.

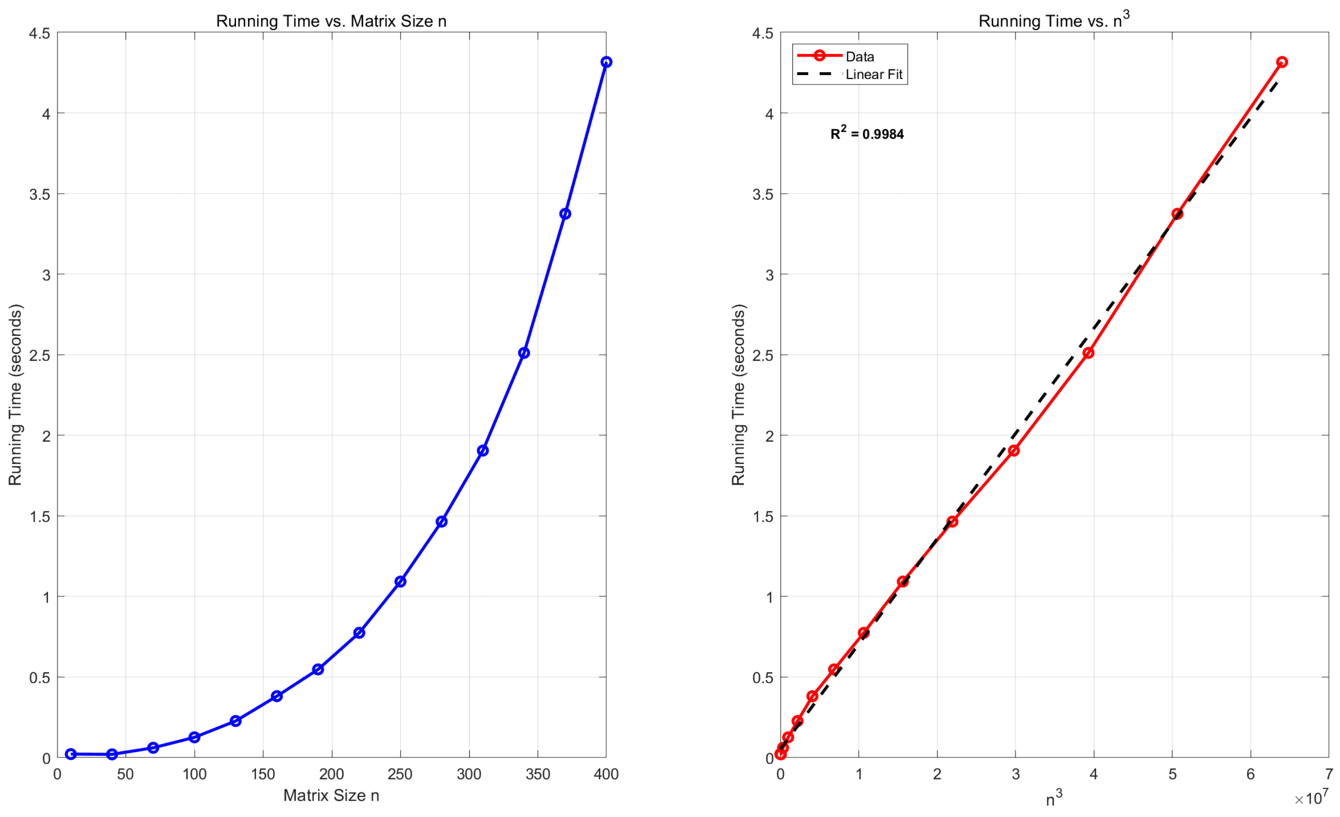
```
clc; clear;
format short e
rng(51);
% Set parameters for different matrix sizes
n_values = 10:30:400;
running_times = zeros(size(n_values));  % To store the running times

% Loop through different matrix sizes
for i = 1:length(n_values)
```

```
    n = n_values(i);

    % Generate a random matrix A of size n x n
    A = randn(n, n);

    % Start measuring time
    tic;

    % Apply the Implicit QR Algorithm
    [T, Q] = Implicit_QR_Algorithm(A);

    % Store the running time
    running_times(i) = toc;
    fprintf("The %d-th example (n = %d) running time: %.4fs\n", i, n, running_times(i));
end

% Create a new figure for plotting
figure;

% Subplot 1: Plot running time vs. n
subplot(1, 2, 1);
plot(n_values, running_times, 'bo-', 'LineWidth', 2);
title('Running Time vs. Matrix Size n');
xlabel('Matrix Size n');
ylabel('Running Time (seconds)');
grid on;

% Subplot 2: Plot running time vs. n^3
subplot(1, 2, 2);
n_cubed = n_values.^3;  % n^3
plot(n_cubed, running_times, 'ro-', 'LineWidth', 2);
title('Running Time vs. n^3');
xlabel('n^3');
ylabel('Running Time (seconds)');
grid on;

% Add a linear fit to the second plot
p = polyfit(n_cubed, running_times, 1);
hold on;
fitted_values = polyval(p, n_cubed);
plot(n_cubed, fitted_values, 'k--', 'LineWidth', 2);
legend('Data', 'Linear Fit', 'Location', 'NorthWest');

% Calculate R^2
SS_res = sum((running_times - fitted_values).^2); % Residual sum of squares
SS_tot = sum((running_times - mean(running_times)).^2); % Total sum of squares
R_squared = 1 - (SS_res / SS_tot); % R^2 calculation

% Display R^2 on the plot
text(0.1 * max(n_cubed), 0.9 * max(running_times), sprintf('R^2 = %.4f', R_squared), ...
    'FontSize', 9, 'Color', 'k', 'FontWeight', 'bold');
hold off;
```

运行结果:

```
Converged after 21 iterations!
The 1-th example (n = 10) running time: 0.0221s
Converged after 77 iterations!
The 2-th example (n = 40) running time: 0.0205s
Converged after 149 iterations!
The 3-th example (n = 70) running time: 0.0613s
```

```
Converged after 205 iterations!
The 4-th example (n = 100) running time: 0.1262s
Converged after 250 iterations!
The 5-th example (n = 130) running time: 0.2274s
Converged after 307 iterations!
The 6-th example (n = 160) running time: 0.3815s
Converged after 356 iterations!
The 7-th example (n = 190) running time: 0.5475s
Converged after 408 iterations!
The 8-th example (n = 220) running time: 0.7745s
Converged after 460 iterations!
The 9-th example (n = 250) running time: 1.0925s
Converged after 511 iterations!
The 10-th example (n = 280) running time: 1.4647s
Converged after 570 iterations!
The 11-th example (n = 310) running time: 1.9051s
Converged after 639 iterations!
The 12-th example (n = 340) running time: 2.5112s
Converged after 679 iterations!
The 13-th example (n = 370) running time: 3.3745s
Converged after 712 iterations!
The 14-th example (n = 400) running time: 4.3160s
```



6. An Afterthought

Francis 的这项工作是如此之好，几乎对所有的矩阵每个特征值平均只需二次迭代就能收敛.
因此将 Francis 双位移隐式 QR 迭代算法称为 "直接法" 从某种意义上来说是恰当的.

不过使用 Francis 位移的 QR 迭代收敛可能失败，例如它会保持如下的矩阵不变:

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

```
A = [0, 0, 1;
     1, 0, 0;
     0, 1, 0];

[T, Q] = Implicit_QR_Algorithm(A);

disp("A:");
disp(A);

disp("T:");
disp(T);

disp("Q:");
disp(Q);
```

运行结果:

```
A:
     0     0     1
     1     0     0
     0     1     0

T:
     0     0     1
     1     0     0
     0     1     0

Q:
     1     0     0
     0     1     0
     0     0     1
```

因此每十次 Francis 位移若不出现收敛，就会有一个特殊的位移.
(对于酉矩阵和秩一矩阵还可能有特殊的 QR 算法)
但找到一个对一切矩阵都能保证快速收敛的位移策略仍是一个悬而未决的问题.

此外，G. Golub 在 Matrix Computation (3rd Edition) $7.5.7$ 节中指出:
若 $A \in \mathbb{R}^{n\times n}$ 的元素的数量级变化很大 $(\geq 10^{10})$，则在应用 Francis 隐式 QR 算法之前，应对 A 进行平衡.
具体来说，可以计算一个对角阵 D 使得 $D^{-1}AD$ 是一个行列平衡的矩阵:

$$D^{-1}AD = [c_1, \dots, c_n] = \begin{bmatrix} r_1^{\text{T}} \\ \vdots \\ r_n^{\text{T}} \end{bmatrix},$$

其中 $c_1, \dots, c_n$ 和 $r_1, \dots, r_n$ 满足 $\|c_i\|_\infty \approx \|r_i\|_\infty \ (i = 1, \dots, n)$.
对角阵 D 的形式可取为 $D = \text{diag}\{\beta^{i_1}, \dots, \beta^{i_n}\}$ (即元素为 β 的幂次),
其中 β 是浮点基数 (通常是 2)，这样做可以避免在计算 $D^{-1}AD$ 时引入舍入误差.
当 A 被平衡后 (需要 $O(n^2)$ 的计算量)，使用 Francis 隐式 QR 算法计算的特征值通常会更加精确.

The End