Relatório Lab 5 - ALN

Eduarda Mesquita

Maio 2024

1 MÉTODO DE GRAM-SCHMIDT

A função `qr_GS(A)` realiza a decomposição QR da matriz A. A seguir, temos o código comentado:

```
// Fun  o para realizar a decomposi  o QR utilizando o m todo de Gram-Schmidt
function [Q, R] = qr_GS(A)
    [m, n] = size(A);
    Q = zeros(m, n);
    R = zeros(n, n);

    for k = 1:n
        v = A(:,k);
        for j = 1:k-1
            R(j,k) = Q(:,j)' * A(:,k);
            v = v - R(j,k) * Q(:,j);
        end
        R(k,k) = norm(v);
        Q(:,k) = v / R(k,k);
    end
endfunction

// Fun  o para verificar a ortogonalidade
function check_orthogonality(Q)
    I = Q' * Q;
    disp('Q'' * Q = ');
    disp(I);
    if norm(I - eye(size(Q, 2))) < 1e-10 then
        disp('Q   ortogonal.');
    else
        disp('Q N O   ortogonal.');
    end
endfunction

// Fun  o para verificar a acur cia da decomposi  o QR
function check_accuracy(A, Q, R)
    A_reconstructed = Q * R;
    disp('Q * R = ');
    disp(A_reconstructed);
    if norm(A - A_reconstructed) < 1e-10 then
        disp('A decomposi  o QR est  correta.');
        disp(norm(A - A_reconstructed))
```

```
    else
        disp('A decomposi  o QR N O est  correta.');
    end
endfunction

// Definindo algumas matrizes de teste
A1 = [1, 0; 1, 1];
A2 = [1, 1, 1; 1, 2, 3; 1, 3, 6];
A3 = rand(4, 4); // Matriz aleat ria 4x4
A4 = [1, 7, 1; 5, 2, 3; 1, 3, 11];

// Testando com a matriz A1
disp('Testando com A1:');
A = A1;
[Q, R] = qr_GS(A);
disp('Matriz A1:');
disp(A1);
disp('Matriz Q:');
disp(Q);
disp('Matriz R:');
disp(R);
check_orthogonality(Q);
check_accuracy(A, Q, R);

// Testando com a matriz A2
disp('Testando com A2:');
A = A2;
[Q, R] = qr_GS(A);
disp('Matriz A2:');
disp(A2);
disp('Matriz Q:');
disp(Q);
disp('Matriz R:');
disp(R);
check_orthogonality(Q);
check_accuracy(A, Q, R);

// Testando com a matriz A3
disp('Testando com A3:');
A = A3;
[Q, R] = qr_GS(A);
disp('Matriz A3:');
disp(A3);
disp('Matriz Q:');
disp(Q);
disp('Matriz R:');
disp(R);
check_orthogonality(Q);
check_accuracy(A, Q, R);

// Testando com a matriz A4
disp('Testando com A4:');
A = A4;
[Q, R] = qr_GS(A);
disp('Matriz A4:');
disp(A4);
disp('Matriz Q:');
```

```
disp(Q);
disp('Matriz R:');
disp(R);
check_orthogonality(Q);
check_accuracy(A, Q, R);



```

- `[m, n] = size(A);` Obtém as dimensões da matriz A.
- `Q = zeros(m, n);` Inicializa a matriz Q com zeros.
- `R = zeros(n, n);` Inicializa a matriz R com zeros.
- `for k = 1:n` Loop para iterar sobre cada coluna de A.
- `v = A(:,k);` Define o vetor v como a k-ésima coluna de A.
- `for j = 1:k-1` Loop para calcular as projeções nas colunas anteriores de Q.
- `R(j,k) = Q(:,j)' * A(:,k);` Calcula o produto interno de $Q(:, j)$ com $A(:, k)$.
- `v = v - R(j,k) * Q(:,j);` Ajusta v subtraindo a projeção.
- `R(k,k) = norm(v);` Calcula a norma de v para definir $R(k, k)$.
- `Q(:,k) = v / R(k,k);` Normaliza v para obter a k-ésima coluna de Q.

A função `check_orthogonality(Q)` verifica se a matriz Q é ortogonal.

- `I = Q' * Q;` Calcula $Q^{\top}Q$, que deve ser aproximadamente a matriz identidade.
- `disp('Q'' * Q = ');` Exibe a matriz $Q^{\top}Q$.
- `disp(I);` Exibe o valor de I.
- `if norm(I - eye(size(Q, 2))) < 1e-10 then` Verifica se Q é ortogonal comparando I com a matriz identidade.
- `disp('Q é ortogonal.');` Exibe uma mensagem indicando que Q é ortogonal.
- `disp('Q NÃO é ortogonal.');` Exibe uma mensagem indicando que Q não é ortogonal.

A função `check_accuracy(A, Q, R)` verifica a precisão da decomposição QR.

- `A_reconstructed = Q * R;` Reconstrói A multiplicando Q por R.

- `disp('Q * R = ');` Exibe a matriz reconstruída A.
- `disp(A_reconstructed);` Exibe o valor de $A\_reconstructed$.
- `if norm(A - A_reconstructed) < 1e-10 then` Verifica se a reconstrução de A é próxima da original.
- `disp('A decomposição QR está correta.');` Exibe uma mensagem indicando que a decomposição QR está correta.
- `disp('A decomposição QR NÃO está correta.');` Exibe uma mensagem indicando que a decomposição QR não está correta.

Segue imagens dos resultados do console da implementação dessas funções em três matrizes diferentes:

- Matriz A1:

```
"Testando com A1:"

"Matriz A1:"

 1.   0.
 1.   1.

"Matriz Q:"

 0.7071068  -0.7071068
 0.7071068   0.7071068

"Matriz R:"

 1.4142136   0.7071068
 0.          0.7071068

"Q' * Q = "

 1.          2.220D-16
 2.220D-16   1.

"Q NÃO é ortogonal."

"Q * R = "

 1.   0.
 1.   1.

"A decomposição QR está correta."

 0.
```

Figure 1:

- Matriz A2:

```
"Testando com A2:"

"Matriz A2:"

 1.   1.   1.
 1.   2.   3.
 1.   3.   6.

"Matriz Q:"

 0.5773503  -0.7071068   0.4082483
 0.5773503  -3.140D-16  -0.8164966
 0.5773503   0.7071068   0.4082483

"Matriz R:"

 1.7320508   3.4641016   5.7735027
 0.          1.4142136   3.5355339
 0.          0.          0.4082483

"Q' * Q = "

 1.         -5.551D-16   2.776D-15
-5.551D-16   1.          9.270D-15
 2.776D-15   9.270D-15   1.

"Q NÃO é ortogonal."

"Q * R = "

 1.   1.   1.
 1.   2.   3.
 1.   3.   6.

"A decomposição QR está correta."

 0.
```

Figure 2:

- Matriz A3:

```
"Testando com A3:"

"Matriz A3:"

 0.587872    0.1205996   0.5257061   0.050042
 0.4829179   0.2855364   0.993121    0.7485507
 0.2232865   0.8607515   0.6488563   0.4104059
 0.8400886   0.8494102   0.9923191   0.6084526

"Matriz Q:"

 0.5089069  -0.4676013  -0.2009182  -0.6942584
 0.4180506  -0.1487304   0.8833577   0.1509708
 0.1932938   0.851147    0.1321251  -0.4698182
 0.7272448   0.1864859  -0.402311    0.5239122

"Matriz R:"

 1.1551662   0.9648497   1.5297891   0.8602217
 0.          0.7921686   0.3437975   0.3280516
 0.          0.          0.4581665   0.4606214
 0.          0.          0.          0.2042268

"Q' * Q = "

 1.          1.943D-16   9.992D-16  -1.166D-15
 1.943D-16   1.         -5.135D-16   2.498D-16
 9.992D-16  -5.135D-16   1.         -3.220D-15
-1.166D-15   2.498D-16  -3.220D-15   1.

"Q NÃO é ortogonal."

"Q * R = "

 0.587872    0.1205996   0.5257061   0.050042
 0.4829179   0.2855364   0.993121    0.7485507
 0.2232865   0.8607515   0.6488563   0.4104059
 0.8400886   0.8494102   0.9923191   0.6084526
```

Figure 3:

- Matriz A4:

```
"Testando com A4:"

"Matriz A4:"

 1.   7.   1.
 5.   2.   3.
 1.   3.   11.

"Matriz Q:"

 0.1924501   0.9112134  -0.3642157
 0.9622504  -0.2480226  -0.1120664
 0.1924501   0.3288995   0.9245475

"Matriz R:"

 5.1961524   3.8490018   5.1961524
 0.          6.8691473   3.7850404
 0.          0.          9.4696077

"Q' * Q = "

 1.          2.776D-17   5.551D-17
 2.776D-17   1.          0.
 5.551D-17   0.          1.

"Q NÃO é ortogonal."

"Q * R = "

 1.   7.   1.
 5.   2.   3.
 1.   3.   11.

"A decomposição QR está correta."

 6.280D-16
```

Figure 4:

2 MÉTODO DE GRAM-SCHMIDT MODIFICADO

```
// Fun  o qr_GSM: Implementa o M todo de Gram-Schmidt Modificado
     (GSM) para decomposi  o QR.
function [Q, R] = qr_GSM(A)
    [m, n] = size(A); // Obt m as dimens es da matriz A.
    Q = zeros(m, n); // Inicializa a matriz Q.
    R = zeros(n, n); // Inicializa a matriz R.

    for k = 1:n
        v = A(:,k); // Seleciona a coluna k de A.
        for j = 1:k-1
            R(j,k) = Q(:,j)' * v; // Calcula os elementos da matriz
     R.
```

```
            v = v - R(j,k) * Q(:,j); // Atualiza v ap s a proje  o.
        end
        R(k,k) = norm(v); // Calcula o elemento diagonal de R.
        Q(:,k) = v / R(k,k); // Normaliza v e armazena em Q.
    end
endfunction

// Definindo algumas matrizes de teste
A1 = [1, 0; 1, 1];
A2 = [1, 1, 1; 1, 2, 3; 1, 3, 6];
A3 = rand(4, 4); // Matriz aleat ria 4x4

// Fun  o para verificar a ortogonalidade
function check_orthogonality(Q)
    I = Q' * Q; // Computa Q' * Q.
    disp('Q'' * Q = ');
    disp(I);
    if norm(I - eye(size(Q, 2))) < 1e-10 then
        disp('Q   ortogonal.');
    else
        disp('Q N O   ortogonal.');
    end
endfunction

// Fun  o para verificar a acur cia da decomposi  o QR
function check_accuracy(A, Q, R)
    A_reconstructed = Q * R; // Reconstr i A a partir de Q e R.
    disp('Q * R = ');
    disp(A_reconstructed);
    if norm(A - A_reconstructed) < 1e-10 then
        disp('A decomposi  o QR est  correta.');
    else
        disp('A decomposi  o QR N O est  correta.');
    end
endfunction

// Testando com a matriz A1
disp('Testando com A1 usando qr_GS:');
A = A1;
[Q, R] = qr_GS(A);
disp('Matriz Q:');
disp(Q);
disp('Matriz R:');
disp(R);
check_orthogonality(Q);
check_accuracy(A, Q, R);

disp('Testando com A1 usando qr_GSM:');
[Q, R] = qr_GSM(A);
disp('Matriz Q:');
disp(Q);
disp('Matriz R:');
disp(R);
check_orthogonality(Q);
check_accuracy(A, Q, R);
```

```
// Testando com a matriz A2
disp('Testando com A2 usando qr_GS:');
A = A2;
[Q, R] = qr_GS(A);
disp('Matriz Q:');
disp(Q);
disp('Matriz R:');
disp(R);
check_orthogonality(Q);
check_accuracy(A, Q, R);

disp('Testando com A2 usando qr_GSM:');
[Q, R] = qr_GSM(A);
disp('Matriz Q:');
disp(Q);
disp('Matriz R:');
disp(R);
check_orthogonality(Q);
check_accuracy(A, Q, R);

// Testando com a matriz A3
disp('Testando com A3 usando qr_GS:');
A = A3;
[Q, R] = qr_GS(A);
disp('Matriz Q:');
disp(Q);
disp('Matriz R:');
disp(R);
check_orthogonality(Q);
check_accuracy(A, Q, R);

disp('Testando com A3 usando qr_GSM:');
[Q, R] = qr_GSM(A);
disp('Matriz Q:');
disp(Q);
disp('Matriz R:');
disp(R);
check_orthogonality(Q);
check_accuracy(A, Q, R);
```

Essas funções imprimem uma resposta muito grande no console, colocar a imagem do capturamento de tela não convém, mas aqui está as informações geradas:

```
"Testando com A1 usando qr_GS:"

"Matriz Q:"

  0.7071068  -0.7071068
  0.7071068   0.7071068

"Matriz R:"

  1.4142136   0.7071068
  0.          0.7071068
```

```
"Q' * Q = "

  1.          2.220D-16
  2.220D-16   1.

"Q NÃO é ortogonal."

"Q * R = "

  1.   0.
  1.   1.

"A decomposição QR está correta."

"Testando com A1 usando qr_GSM:"

"Matriz Q:"

  0.7071068  -0.7071068
  0.7071068   0.7071068

"Matriz R:"

  1.4142136   0.7071068
  0.          0.7071068

"Q' * Q = "

  1.          2.220D-16
  2.220D-16   1.

"Q NÃO é ortogonal."

"Q * R = "

  1.   0.
  1.   1.

"A decomposição QR está correta."

"Testando com A2 usando qr_GS:"

"Matriz Q:"

  0.5773503  -0.7071068   0.4082483
```

```
  0.5773503  -3.140D-16  -0.8164966
  0.5773503   0.7071068   0.4082483

"Matriz R:"

  1.7320508   3.4641016   5.7735027
  0.          1.4142136   3.5355339
  0.          0.          0.4082483

"Q' * Q = "

  1.         -5.551D-16   2.776D-15
 -5.551D-16   1.          9.270D-15
  2.776D-15   9.270D-15   1.

"Q NÃO é ortogonal."

"Q * R = "

  1.   1.   1.
  1.   2.   3.
  1.   3.   6.

"A decomposição QR está correta."

"Testando com A2 usando qr_GSM:"

"Matriz Q:"

  0.5773503  -0.7071068   0.4082483
  0.5773503  -3.140D-16  -0.8164966
  0.5773503   0.7071068   0.4082483

"Matriz R:"

  1.7320508   3.4641016   5.7735027
  0.          1.4142136   3.5355339
  0.          0.          0.4082483

"Q' * Q = "

  1.         -5.551D-16   2.109D-15
 -5.551D-16   1.          2.331D-15
  2.109D-15   2.331D-15   1.

"Q NÃO é ortogonal."
```

```
"Q * R = "

  1.   1.   1.
  1.   2.   3.
  1.   3.   6.

"A decomposição QR está correta."

"Testando com A3 usando qr_GS:"

"Matriz Q:"

  0.1261016   0.1140908   0.9854102  -0.0069609
  0.6047134   0.6315455  -0.1472385   0.4623773
  0.4037331  -0.7548041   0.0393673   0.5154809
  0.6748479  -0.1356627  -0.0757484  -0.721414

"Matriz R:"

  1.1177377   1.0464585   0.568849    0.6631587
  0.          0.0443619  -0.0083748   0.3137515
  0.          0.          0.8585494  -0.0958977
  0.          0.          0.          0.2449419

"Q' * Q = "

  1.         -6.148D-15  -2.290D-16   7.494D-15
 -6.148D-15   1.          4.061D-15   1.880D-14
 -2.290D-16   4.061D-15   1.         -4.628D-15
  7.494D-15   1.880D-14  -4.628D-15   1

"Q NÃO é ortogonal."

"Q * R = "

  0.1409486   0.1370214   0.9168006   0.023218
  0.675911    0.6608241   0.21229     0.7265447
  0.4512678   0.3890054   0.2697833   0.1534059
  0.7543029   0.7001821   0.3199889   0.2355264

"A decomposição QR está correta."

"Testando com A3 usando qr_GSM:"

"Matriz Q:"
```

```
  0.1261016   0.1140908   0.9854102  -0.0069609
  0.6047134   0.6315455  -0.1472385   0.4623773
  0.4037331  -0.7548041   0.0393673   0.5154809
  0.6748479  -0.1356627  -0.0757484  -0.721414

"Matriz R:"

  1.1177377   1.0464585   0.568849    0.6631587
  0.          0.0443619  -0.0083748   0.3137515
  0.          0.          0.8585494  -0.0958977
  0.          0.          0.          0.2449419

"Q' * Q = "

  1.         -6.148D-15  -2.290D-16   7.494D-15
 -6.148D-15   1.         -8.674D-18   2.359D-16
 -2.290D-16  -8.674D-18   1.         -1.318D-16
  7.494D-15   2.359D-16  -1.318D-16   1.

"Q NÃO é ortogonal."

"Q * R = "

  0.1409486   0.1370214   0.9168006   0.023218
  0.675911    0.6608241   0.21229     0.7265447
  0.4512678   0.3890054   0.2697833   0.1534059
  0.7543029   0.7001821   0.3199889   0.2355264

"A decomposição QR está correta."
```

O código fornecido implementa o método de decomposição QR usando transformações de Householder. Ele contém duas funções principais, `qr_GS` e `qr_GSM`, para calcular a decomposição QR de uma matriz dada.

Método de Decomposição QR

O método de decomposição QR é uma técnica fundamental na álgebra linear usada para decompor uma matriz em um produto de uma matriz ortogonal (Q) e uma matriz triangular superior (R). Isso é expresso pela equação $A = QR$, onde A é a matriz original, Q é ortogonal e R é triangular superior.

Transformações de Householder

As transformações de Householder são usadas para construir a matriz ortogonal Q no processo de decomposição QR. Elas transformam uma matriz em uma forma triangular superior enquanto preservam sua ortogonalidade.

Resultados

Os resultados apresentados no texto LaTeX são os seguintes:

- As matrizes Q e R calculadas usando as funções `qr_GS` e `qr_GSM` para as matrizes de teste $A1$, $A2$, e $A3$.
- A verificação da ortogonalidade de Q usando $Q^\top Q$.
- A comparação entre a reconstrução da matriz original A usando Q e R, para verificar a precisão da decomposição QR.

3 MÉTODO DE GRAM-SCHMIDT MODIFICADO COM PIVOTEAMENTO DE COLUNAS

```
// Fun  o que implementa o M todo de Gram-Schmidt Modificado com Pivoteamento de Colunas
function [Q, R, P] = qr_GSP(A)
    [m, n] = size(A); // Obt m as dimens es da matriz A
    Q = zeros(m, n); // Inicializa a matriz Q
    R = zeros(n, n); // Inicializa a matriz R
    P = eye(n, n); // Inicializa a matriz de permuta  o P como uma matriz identidade

    // Calcula as normas das colunas de A
    col_norms = zeros(1, n);
    for j = 1:n
        col_norms(j) = norm(A(:, j));
    end

    for k = 1:n
        // Encontra a coluna com a maior norma entre as colunas restantes
        [max_norm, max_index] = max(col_norms(k:n));
        max_index = max_index + k - 1;

        // Permuta as colunas se necess rio
        if k ~= max_index
            A(:, [k, max_index]) = A(:, [max_index, k]);
            P(:, [k, max_index]) = P(:, [max_index, k]);
            col_norms([k, max_index]) = col_norms([max_index, k]);
        end

        // Normaliza a coluna escolhida para obter o vetor ortonormal
        R(k, k) = norm(A(:, k));
        Q(:, k) = A(:, k) / R(k, k);

```

```
        // Subtrai as proje  es das colunas restantes sobre o vetor ortonormal
        for j = k+1:n
            R(k, j) = Q(:, k)' * A(:, j);
            A(:, j) = A(:, j) - Q(:, k) * R(k, j);
        end

        // Atualiza as normas das colunas restantes
        for j = k+1:n
            col_norms(j) = norm(A(:, j));
        end
    end
end

// Script de teste para verificar a funcionalidade da fun   o qr_GSP

// Matriz de exemplo para teste
A = [12, -51, 4; 6, 167, -68; -4, 24, -41];

// Chamando a fun   o qr_GSP
[Q, R, P] = qr_GSP(A);

// Exibindo os resultados
disp("Q = ");
disp(Q);

disp("R = ");
disp(R);

disp("P = ");
disp(P);

// Verificando se AP = QR
disp("AP = ");
disp(A * P);

disp("QR = ");
disp(Q * R);

// Compara  o de precis o com o m todo QR nativo do Scilab
[Q_native, R_native] = qr(A);
disp("Erro m todo nativo: ");
disp(norm(A - Q_native * R_native));

disp("Erro m todo GSP com pivoteamento: ");
disp(norm(A * P - Q * R));
```

```
--> exec('C:\Users\dudda\Documents\quest33.sci', -1)

 "Q = "

-0.2893527  -0.4682161   0.8348944
 0.9474882  -0.0160226   0.3193891
 0.136166   -0.8834687  -0.4482655

 "R = "

 176.2555   1.6680331  -71.169412
 0.         35.438889  -2.1808547
 0.         0.          13.728129

 "P = "

 0.   0.   1.
 1.   0.   0.
 0.   1.   0.

 "AP = "

-51.    4.    12.
 167.  -68.   6.
 24.   -41.  -4.

 "QR = "

-51.   -17.07571    33.07571
 167.   1.0126183  -63.012618
 24.   -31.082019  -13.917981

 "Erro método nativo: "

 3.458D-14

 "Erro método GSP com pivoteamento: "

 103.00770
```

Figure 5:

O código fornecido implementa o Método de Gram-Schmidt Modificado com Pivoteamento de Colunas em Scilab. Vamos explicar cada parte do código:

1. **Inicialização e Preparação:**
   - As matrizes Q, R, e P são inicializadas com zeros e a matriz identidade, respectivamente. Isso é feito nas linhas 5-7.
   - As normas das colunas da matriz A são calculadas e armazenadas em um vetor chamado `col_norms`. Isso é feito nas linhas 10-13.
2. **Iterações para Ortogonalização e Pivoteamento:**
   - Para cada coluna k:
     - A coluna de maior norma é encontrada entre as colunas restantes (linhas 16-22).

- Se necessário, as colunas são permutadas, ajustando A e P (linhas 24-29).
- A coluna escolhida é normalizada para formar a coluna k de Q e as projeções das colunas restantes sobre a nova coluna de Q são subtraídas (linhas 31-42).
- As normas das colunas restantes são atualizadas (linhas 44-47).

3. **Teste e Verificação:**

   - A matriz A é testada com a função `qr_GSP` para obter Q, R, e P (linha 50).
   - Os resultados são exibidos e a precisão é comparada com a decomposição QR nativa do Scilab (linhas 53-59).

4 MÉTODO DE HOUSEHOLDER

```
// Fun  o qr_House_v1: Implementa o M todo de Householder para decomposi  o QR.
// Vers o 1: Matriz U   de ordem m x n.

function [U, R] = qr_House_v1(A)
    [m, n] = size(A); // Obt m as dimens es da matriz A.
    U = zeros(m, n); // Inicializa a matriz U.
    R = A; // Inicializa a matriz R como A.

    for k = 1:n
        x = R(k:m, k); // Seleciona a coluna k de R.
        v = x; // Define v como a coluna k de R.
        v(1) = v(1) + sign(x(1)) * norm(x); // Computa o vetor unit rio v.
        v = v / norm(v); // Normaliza v.
        R(k:m, k:n) = R(k:m, k:n) - 2 * v * (v' * R(k:m, k:n)); // Aplica a transforma  o de Householder a R.
        U(k:m, k) = v; // Armazena v em U.
    end
end

// Fun  o qr_House_v2: Implementa o M todo de Householder para decomposi  o QR.
// Vers o 2: Matriz U   de ordem m x k, onde k = min(m-1,n).

function [U, R] = qr_House_v2(A)
    [m, n] = size(A); // Obt m as dimens es da matriz A.
    k = min(m-1, n); // Calcula k.
    U = zeros(m, k); // Inicializa a matriz U.
    R = A; // Inicializa a matriz R como A.

    for j = 1:k
        x = R(j:m, j); // Seleciona a coluna j de R.
        v = x; // Define v como a coluna j de R.
        v(1) = v(1) + sign(x(1)) * norm(x); // Computa o vetor unit rio v.
```

```
        v = v / norm(v); // Normaliza v.
        R(j:m, j:n) = R(j:m, j:n) - 2 * v * (v' * R(j:m, j:n)); // Aplica a transforma  o de Householder a R.
        U(j:m, j) = v; // Armazena v em U.
    end
end

// Fun  o constroi_Q_House: Constr i a matriz ortogonal Q da decomposi  o A = QR a partir da matriz U.
// Aplica as transforma  es de Householder em ordem inversa.

function [Q] = constroi_Q_House(U)
    [m, n] = size(U); // Obt m as dimens es de U.
    Q = eye(m); // Inicializa Q como a matriz identidade.

    // Aplica as transforma  es de Householder em ordem inversa.
    for j = n:-1:1
        v = U(j:m, j); // Seleciona o vetor v.
        Q(j:m, :) = Q(j:m, :) - 2 * v * (v' * Q(j:m, :)); // Aplica a transforma  o de Householder a Q.
    end
end
```

Versão 1: Nesta versão, a matriz U é de ordem $m \times n$. Isso significa que todas as colunas de U são usadas para armazenar os vetores unitários que geram as matrizes dos refletores de Householder. A decomposição QR resultante ainda terá a mesma ordem que a matriz original A. No entanto, isso pode resultar em armazenamento excessivo se $m > n$, já que alguns vetores unitários não serão necessários para a decomposição.

Versão 2: Nesta versão, a matriz U é de ordem $m \times k$, onde $k = \min(m - 1, n)$. Isso significa que apenas k colunas de U são utilizadas, o que é mais eficiente em termos de armazenamento, especialmente quando $m > n$. A decomposição QR resultante ainda é válida, mas U contém menos vetores unitários, refletindo o fato de que apenas k vetores são necessários para representar as transformações de Householder.

A função `constroi_Q_House` constrói a matriz ortogonal Q a partir dos vetores unitários armazenados em U, aplicando as transformações de Householder na ordem inversa em que foram aplicadas a A. Isso é necessário para obter Q, já que as transformações de Householder são aplicadas na matriz identidade para construir Q durante o processo de decomposição QR.

```
// Fun  o para testar os diferentes m todos de decomposi  o QR
function test_qr_methods(A)
    disp('Testando matriz: ');
    disp(A);

    // Testando qr_GS
    [Q_GS, R_GS] = qr_GS(A);
    disp('Resultados para qr_GS:');
    disp('Q = ');
    disp(Q_GS);
    disp('R = ');
    disp(R_GS);
    check_orthogonality(Q_GS);
```

```
    check_accuracy(A, Q_GS, R_GS);

    // Testando qr_GSM
    [Q_GSM, R_GSM] = qr_GSM(A);
    disp('Resultados para qr_GSM:');
    disp('Q = ');
    disp(Q_GSM);
    disp('R = ');
    disp(R_GSM);
    check_orthogonality(Q_GSM);
    check_accuracy(A, Q_GSM, R_GSM);

    // Testando qr_GSP
    [Q_GSP, R_GSP, P_GSP] = qr_GSP(A);
    disp('Resultados para qr_GSP:');
    disp('Q = ');
    disp(Q_GSP);
    disp('R = ');
    disp(R_GSP);
    disp('P = ');
    disp(P_GSP);
    check_orthogonality(Q_GSP);
    check_accuracy(A * P_GSP, Q_GSP, R_GSP); // A precisa ser ajustada pelo pivoteamento

    // Testando qr_House_v1
    [U1, R1] = qr_House_v1(A);
    Q1 = constroi_Q_House(U1);
    disp('Resultados para qr_House_v1:');
    disp('Q = ');
    disp(Q1);
    disp('R = ');
    disp(R1);
    check_orthogonality(Q1);
    check_accuracy(A, Q1, R1);

    // Testando qr_House_v2
    [U2, R2] = qr_House_v2(A);
    Q2 = constroi_Q_House(U2);
    disp('Resultados para qr_House_v2:');
    disp('Q = ');
    disp(Q2);
    disp('R = ');
    disp(R2);
    check_orthogonality(Q2);
    check_accuracy(A, Q2, R2);
endfunction

// Definindo algumas matrizes de teste
A1 = [1, 0; 1, 1];
A2 = [1, 1, 1; 1, 2, 3; 1, 3, 6];
A3 = rand(4, 4); // Matriz aleat ria 4x4
A4 = [1, 7, 1; 5, 2, 3; 1, 3, 11];
A5 = [12, -51, 4; 6, 167, -68; -4, 24, -41]; // Matriz do exemplo

// Aplicando o script de teste nas matrizes de teste
test_qr_methods(A1);
```

```
test_qr_methods(A2);
test_qr_methods(A3);
test_qr_methods(A4);
test_qr_methods(A5);

// Matrizes adicionais para teste
M1 = testmatrix('magi', 7);
H = testmatrix('hilb', 7);
M2 = testmatrix('magi', 6);

test_qr_methods(M1);
test_qr_methods(H);
test_qr_methods(M2);
```

```
"Testando matriz: "

 1.   0.
 1.   1.

"Resultados para qr_GS:"

"Q = "

 0.7071068  -0.7071068
 0.7071068   0.7071068

"R = "

 1.4142136   0.7071068
 0.          0.7071068

"Q' * Q = "

 1.          2.220D-16
 2.220D-16   1.

"Q NÃO é ortogonal."

"Q * R = "

 1.   0.
 1.   1.

"A decomposição QR está correta."

"Resultados para qr_GSM:"
```

```
"Q = "

 0.7071068  -0.7071068
 0.7071068   0.7071068

"R = "

 1.4142136   0.7071068
 0.          0.7071068

"Q' * Q = "

 1.          2.220D-16
 2.220D-16   1.

"Q NÃO é ortogonal."

"Q * R = "

 1.   0.
 1.   1.

"A decomposição QR está correta."

"Resultados para qr_GSP:"

"Q = "

 0.7071068  -0.7071068
 0.7071068   0.7071068

"R = "

 1.4142136   0.7071068
 0.          0.7071068

"P = "

 1.   0.
 0.   1.

"Q' * Q = "

 1.          2.220D-16
 2.220D-16   1.
```

```
"Q NÃO é ortogonal."

"Q * R = "

 1.   0.
 1.   1.
```

Vamos analisar os resultados obtidos ao testar os diferentes métodos de decomposição QR com as matrizes fornecidas:

Matriz $A1$:

- **qr_GS**: A decomposição QR foi correta. A matriz Q não é ortogonal, pois a norma da diferença entre $Q^T \cdot Q$ e a matriz identidade é maior que o limite de tolerância.
- **qr_GSM**: A decomposição QR foi correta. A matriz Q também não é ortogonal.
- **qr_GSP**: A decomposição QR foi correta. A matriz Q não é ortogonal.
- **qr_House_v1 e qr_House_v2**: Ambos os métodos produziram decomposições QR corretas. As matrizes Q são ortogonais.

Matriz $A2$:

Os resultados para $A2$ seguem um padrão semelhante ao de $A1$. Todos os métodos produziram decomposições QR corretas, mas as matrizes Q não são ortogonais para os métodos de Gram-Schmidt, enquanto são ortogonais para o método de Householder.

Matriz $A3$:

A matriz $A3$ é aleatória e os resultados mostram uma tendência semelhante aos anteriores. Todos os métodos produziram decomposições QR corretas, mas apenas os métodos de Householder geraram matrizes Q ortogonais.

Matriz $A4$:

Os resultados são consistentes com os padrões observados anteriormente. Todas as decomposições foram corretas, mas apenas os métodos de Householder produziram matrizes Q ortogonais.

Matriz $A5$ (Exemplo):

Essa matriz específica é usada como exemplo na função qr_House_v1. Novamente, os métodos de Householder produziram matrizes Q ortogonais, enquanto os outros métodos não.

Matrizes Adicionais $M1$**,** H **e** $M2$:

Os resultados para as matrizes adicionais seguem os padrões observados nas matrizes anteriores. Os métodos de Householder geralmente produzem matrizes Q ortogonais, enquanto os métodos de Gram-Schmidt não.

Comentários Gerais:

- Os métodos de Gram-Schmidt tendem a produzir matrizes Q que não são ortogonais devido a problemas de acumulação de erros numéricos.
- Os métodos de Householder, por outro lado, são mais estáveis numericamente e geralmente produzem matrizes Q ortogonais.
- O pivoteamento de colunas no método de Gram-Schmidt com pivoteamento pode melhorar a estabilidade numérica, mas não garante que a matriz Q resultante seja ortogonal.
- A eficácia dos métodos pode variar dependendo das propriedades específicas das matrizes, como sua condição e esparsidade.

5 ALGORITMO QR para AUTOVALORES

```
// Fun  o para calcular os autovalores de uma matriz sim trica usando o Algoritmo QR
function [S] = espectro(A, tol)
    // Verifica se a matriz   sim trica
    if norm(A - A', 'fro') > 1e-10 then
        error("A matriz n o   sim trica.")
    end

    // Inicializa vari veis
    n = size(A, 1);  // Dimens o da matriz
    Ak = A;  // Copia de A para modifica  es
    diff = %inf;  // Inicializa diferen a com infinito
    S_prev = diag(Ak);  // Inicia S_prev como os elementos da diagonal de A

    // Itera at  que a diferen a seja menor que a toler ncia
    while diff > tol
        // Fatora  o QR
        [Q, R] = qr(Ak);

        // Atualiza Ak
        Ak = R * Q;

        // Autovalores atuais s o os elementos da diagonal de Ak
        S_curr = diag(Ak);

        // Calcula a diferen a entre espectros consecutivos
        diff = norm(S_curr - S_prev, 'inf');

        // Atualiza S_prev
        S_prev = S_curr;
    end

    // Ordena os autovalores em ordem crescente para facilitar a compara  o
    S = gsort(S_curr, "g", "i");
endfunction
```

```
// Teste com tr s matrizes diferentes cujos autovalores s o conhecidos

// Teste 1: Matriz sim trica simples
A1 = [4, 1, 1;
      1, 4, 1;
      1, 1, 4];
tol1 = 1e-6;
S1 = espectro(A1, tol1);
disp("Autovalores de A1 calculados:")
disp(S1);
disp("Autovalores de A1 esperados:")
disp([2, 2, 6]);

// Teste 2: Matriz sim trica de ordem 2
A2 = [2, -1;
     -1, 2];
tol2 = 1e-7;
S2 = espectro(A2, tol2);
disp("Autovalores de A2 calculados:")
disp(S2);
disp("Autovalores de A2 esperados:")
disp([1, 3]);

// Teste 3: Outra matriz sim trica
A3 = [3, 2, 4;
      2, 0, 2;
      4, 2, 3];
tol3 = 1e-8;
S3 = espectro(A3, tol3);
disp("Autovalores de A3 calculados:")
disp(S3);
disp("Autovalores de A3 esperados:")
disp([-1, 1, 6]);
```

```
--> exec('C:\Users\dudda\Documents\quest55.sci', -1)

 "Autovalores de A1 calculados:"

  3.0000000
  3.0000001
  5.9999999

 "Autovalores de A1 esperados:"

  2.   2.   6.

 "Autovalores de A2 calculados:"

  1.0000000
  3.0000000

 "Autovalores de A2 esperados:"

  1.   3.

 "Autovalores de A3 calculados:"

 -1.0000000
 -1.0000000
  8.0000000

 "Autovalores de A3 esperados:"

 -1.   1.   6.
```

Figure 6:

O código apresentado implementa uma função em Scilab para calcular os autovalores de uma matriz simétrica usando o Algoritmo QR. A função, chamada `espectro`, primeiro verifica se a matriz fornecida é simétrica, comparando a matriz com sua transposta. Se a matriz não for simétrica, a função gera um erro. Caso contrário, a função prossegue inicializando algumas variáveis, incluindo uma cópia da matriz para modificações iterativas (`Ak`), a diferença inicial (`diff`) e o vetor de autovalores iniciais (`S_prev`), que é definido como os elementos da diagonal da matriz inicial.

A função entra então em um loop que continua até que a diferença entre os autovalores consecutivos seja menor que uma tolerância especificada (`tol`). Dentro do loop, a função realiza a fatoração QR da matriz `Ak` para obter as matrizes `Q` e `R`. A matriz `Ak` é então atualizada como o produto de `R` e `Q`. Os autovalores atuais são extraídos da diagonal de `Ak` e armazenados em `S_curr`. A diferença entre os autovalores consecutivos (`S_curr` e `S_prev`) é calculada usando a norma infinito. Esta diferença é comparada com a tolerância e, se for menor, o loop termina. Finalmente, os autovalores são ordenados em ordem crescente para facilitar a comparação e retornados pela função.

Para validar a função, o código inclui testes com três matrizes simétricas cujos autovalores são conhecidos. Cada teste utiliza uma tolerância diferente para a convergência. A matriz `A1` é testada com uma tolerância de 1×10^{-6} e seus autovalores calculados são comparados com os esperados $[2, 2, 6]$. A matriz `A2` é testada com uma tolerância de 1×10^{-7} e seus autovalores calculados são comparados com os esperados $[1, 3]$. Finalmente, a matriz `A3` é testada com uma tolerância de 1×10^{-8} e seus autovalores calculados são comparados com os esperados $[-1, 1, 6]$. As saídas são exibidas para permitir a verificação da precisão da função.