AP5 - Algebra Linear Numérica

Gustavo Ramalho*

5de junho de 2022

1 Questão 1

Escreva uma função Scilab function [Q,R] = $qr\_GS(A)$ que implementa o Método de Gram-Schmidt para determinar a decomposição QR de uma matriz A com colunas linearmente independentes. Testar a sua função com algumas matrizes de ordens diferentes. Para cada uma delas, testar a precisão do método (por exemplo, teste a ortogonalidade da matriz Q obtida calculando Q^TQ).

```
function [Q,R]=qr_GS(A)

m = size(A,1);
n = size(A,2);
Q = eye(m,n);
R = eye(n,n);

for j=1:n
    a_j = A(:,j);
    v = a_j;
    for i=1:j-1
        q_i = Q(:,i);
        r_ij = q_i' * a_j;
        R(i,j) = r_ij;
        v = v - r_ij * q_i;
    end
    r_jj = norm(v);
    R(j,j) = r_jj;
    q_j = v / r_jj;
    Q(:,j) = q_j;
end

endfunction
```

No começo do meu código, inicio a matriz Q como uma matriz identidade de mesmas dimensões que A. Depois, R é uma matriz identidade nxn. Após isso, faço um for para encontrar Q e R.

Agora, vou testar o código com algumas matrizes independentes. Testarei com matrizes de ordem 3, 4 e 5.

1.1 Ordem 3

Primeiro, testo com a matriz $\begin{bmatrix} 2 & 1 & 2 \\ 5 & -2 & 3 \\ 1 & 0 & 4 \end{bmatrix}$, cuja colunas são independentes. Escolhi essa matriz aleatoriamente. Assim, obtenho o resultado:

*gustavoramalho384@gmail.com

Figura 1: Matriz 3x3

```
--> A = [2 1 2; 5 -2 3; 1 0 4]
 A  =

   2.    1.    2.
   5.   -2.    3.
   1.    0.    4.

--> [Q,R]=qr_GS(A)
 Q  =

   0.3651484    0.9056241  -0.2156655
   0.9128709   -0.3937496  -0.1078328
   0.1825742    0.1574998   0.970495
 R  =

   5.4772256  -1.4605935   4.1992063
   0.          1.6931233   1.2599988
   0.          0.          3.1271504

--> |
```

Calculando a decomposição QR, vi que o resultado é realmente o que a função nos deu. Apesar disso, poderia conferir essa informação olhando a precisão da matriz, que vou calcular agora:

Figura 2: Matriz 3x3 - Precisão

```
--> Q' * Q
 ans  =

   1.          -3.175D-16   5.526D-16
  -3.175D-16    1.          3.554D-16
   5.526D-16    3.554D-16   1.
```

A precisão do método está ótima, pois o resultado foi uma matriz identidade com pequenos erros de arredondamento nas outras células que não fazem parte da diagonal. Ou seja, se aproximam de 0!

Agora, vou testar com uma 4x4, também independente:

$$A = \begin{bmatrix} 3 & 2 & 7 & -2 \\ 5 & -1 & 2 & 5 \\ 2 & -7 & -4 & 12 \\ 8 & 10 & 3 & 1 \end{bmatrix}$$

Figura 3: Matriz 4x4

```
--> A = [3 2 7 -2; 5 -1 2 5; 2 -7 -4 12; 8 10 3 1]
 A  =

   3.    2.    7.   -2.
   5.   -1.    2.    5.
   2.   -7.   -4.    12.
   8.    10.   3.    1.

--> [Q,R]=qr_GS(A)
 Q  =

   0.2970443   0.0028044   0.8608718   0.413106
   0.4950738  -0.4085115   0.1873317  -0.7435907
   0.1980295  -0.792718   -0.3015453   0.4913787
   0.792118    0.4524475  -0.3645229   0.1869848
 R  =

   10.099505   6.6339885   4.6536935   5.0497525
   0.          10.487621   3.7308223  -11.108334
   0.          0.          6.5133787  -4.7681519
   0.          0.          0.          1.5393632
```

Figura 4: Matriz 4x4 - Precisão

```
--> Q' * Q
 ans  =

   1.          -9.837D-17  -4.043D-17  -1.427D-15
  -9.837D-17    1.          1.462D-16   2.182D-16
  -4.043D-17    1.462D-16   1.          1.817D-15
  -1.427D-15    2.182D-16   1.817D-15   1.

-->
```

Aqui eu já calculei a decomposição QR, assim como também a sua precisão, e chegamos em resultados tão satisfatórios quanto os resultados da matriz 3x3. Ou seja, a precisão está muito boa e o resultado que o código nos deu, correto. Por fim, testarei com uma matriz 5x5:

$$A = \begin{bmatrix} 5 & -2 & 1 & 8 & 3 \\ -1 & 6 & 5 & 3 & 9 \\ 6 & 5 & 4 & 2 & 1 \\ 4 & 2 & 6 & 1 & 2 \\ 7 & 5 & 6 & 3 & 2 \end{bmatrix}$$

Figura 5: Matriz 5x5

```
--> A = [5 -2 1 8 3; -1 6 5 3 9; 6 5 4 2 1; 4 2 6 1 2; 7 5 6 3 2]
 A  =

   5.  -2.   1.   8.   3.
  -1.   6.   5.   3.   9.
   6.   5.   4.   2.   1.
   4.   2.   6.   1.   2.
   7.   5.   6.   3.   2.

--> [Q,R]=qr_GS(A)
 Q  =

   0.4436783  -0.5131      0.141416    0.7171545   0.0746249
  -0.0887357   0.7796454   0.2735953   0.5552194   0.0340007
   0.5324139   0.2789208  -0.5098169  -0.0926129   0.6084802
   0.3549426   0.0247506   0.7991115  -0.3894635   0.2883437
   0.6211496   0.2246597  -0.0815756  -0.1310031  -0.7347686
 R  =

   11.269428   5.0579321   7.9862086   6.5664382   3.0170121
   0.          8.2714765   5.9972726  -0.5092922   6.2552505
   0.          0.          3.7753402   1.4868645   3.8118602
   0.          0.          0.          6.4351958   6.0148919
   0.          0.          0.          0.          0.2455116
```

Figura 6: Matriz 5x5 - Precisão

```
--> Q' * Q
 ans  =

   1.          -4.233D-17  -4.082D-18  -1.717D-16   3.022D-15
  -4.233D-17    1.          -5.771D-16   1.750D-16  -3.185D-15
  -4.082D-18  -5.771D-16    1.          -3.046D-17   1.610D-14
  -1.717D-16   1.750D-16  -3.046D-17    1.          -3.200D-15
   3.022D-15  -3.185D-15   1.609D-14  -3.214D-15    1.
```

Como podemos observar mais uma vez, a função chega em um resultado correto. Em geral, podemos dizer que a função funciona bem e prevê resultados corretos, uma vez que todos nossos testes cumpriram nossas espectativas. As matrizes foram escolhidas totalmente aleatórias, mas, obviamente, com o cuidado de conferir se são linearmente independentes.

2 Questão 2

Escreva uma função Scilab function [Q,R] = qr_GSM(A) que implementa o Método de Gram-Schmidt Modificado. Testar a sua função com as mesmas matrizes usadas nos testes do item anterior. Comparar a precisão dos dois Métodos.

Primeiramente, crio a função que implementa o Método de Gram-Schmidt Modificado:

```
function [Q,R]=qr_GSM(A)

m = size(A,1);
n = size(A,2);
Q = eye(m,n);
R = eye(n,n);

for j=1:n
    a_j = A(:,j);
    v = a_j;
    for i=1:j-1
        q_i = Q(:,i);
        r_ij = q_i' * v;
        R(i,j) = r_ij;
        v = v - r_ij * q_i;
    end
    r_jj = norm(v);
    R(j,j) = r_jj;
    q_j = v / r_jj;
    Q(:,j) = q_j;

end
endfunction
```

Vamos conferir o resultado com esse método, assim como a precisão:

Figura 7: Matriz 3x3

```
--> [Q,R]=qr_GSM(A)
 Q  =

   0.3651484   0.9056241  -0.2156655
   0.9128709  -0.3937496  -0.1078328
   0.1825742   0.1574998   0.970495
 R  =

   5.4772256  -1.4605935   4.1992063
   0.          1.6931233   1.2599988
   0.          0.          3.1271504

--> Q' * Q
 ans  =

   1.          -3.175D-16   5.526D-16
  -3.175D-16    1.          -5.375D-18
   5.249D-16   -5.375D-18    1.
```

Figura 8: Matriz 4x4

```
--> [Q,R]=qr_GSM(A)
 Q  =

   0.2970443   0.0028044   0.8608718   0.413106
   0.4950738  -0.4085115   0.1873317  -0.7435907
   0.1980295  -0.792718   -0.3015453   0.4913787
   0.792118    0.4524475  -0.3645229   0.1869848
 R  =

   10.099505   6.6339885   4.6536935   5.0497525
   0.          10.487621   3.7308223  -11.108334
   0.          0.          6.5133787  -4.7681519
   0.          0.          0.          1.5393632

--> Q' * Q
 ans  =

   1.          -9.837D-17  -4.043D-17  -1.280D-15
  -9.837D-17    1.          9.067D-17   6.225D-17
  -4.043D-17    9.067D-17   1.          9.219D-16
  -1.280D-15    6.225D-17   9.219D-16   1.
```

Figura 9: Matriz 5x5

```
--> [Q,R]=qr_GSM(A)
 Q  =

   0.4436783  -0.5131      0.141416    0.7171545   0.0746249
  -0.0887357   0.7796454   0.2735953   0.5552194   0.0340007
   0.5324139   0.2789208  -0.5098169  -0.0926129   0.6084802
   0.3549426   0.0247506   0.7991115  -0.3894635   0.2883437
   0.6211496   0.2246597  -0.0815756  -0.1310031  -0.7347686
 R  =

   11.269428   5.0579321   7.9862086   6.5664382   3.0170121
   0.          8.2714765   5.9972726  -0.5092922   6.2552505
   0.          0.          3.7753402   1.4868645   3.8118602
   0.          0.          0.          6.4351958   6.0148919
   0.          0.          0.          0.          0.2455116

--> Q' * Q
 ans  =

   1.          -4.233D-17  -4.082D-18  -1.578D-16   3.811D-15
  -4.233D-17    1.          -5.771D-16   1.785D-16  -4.670D-15
  -4.082D-18   -5.771D-16   1.          1.464D-17   9.550D-16
  -1.578D-16    1.785D-16   1.464D-17   1.          3.292D-15
   3.811D-15   -4.697D-15   9.411D-16   3.292D-15   1.
```

Em todas os testes com as matrizes, podemos ver que os resultados foram iguais aos da função do item 1. Como concluido anteriormente, os resultados estão corretos. $Q^TQ = I$, assim como

esperávamos, com as pequenas diferenças de arredondamento. Esse método feito agora é uma "melhoria"do método da questão 1, sendo mais preciso computacionalmente. Parece que isso não ficou tao destacado nos exemplos que dei, mas é um método mais estável e com menos erros de arredondamento.

3 Questão 4

Escreva uma função Scilab function [U,R] = qr_House(A) que implementa o Método de Householder para determinar a decomposição QR de uma matriz A. A matriz U, triangular inferior, deve conter em suas colunas os vetores unitários que geraram as matrizes dos refletores de Householder usadas para gerar a decomposição QR. Escreva também uma função Scilab function [Q] = constroi_Q_House(U) que constrói a matriz ortogonal Q da decomposição A = QR a partir da matriz U retornada pela função function [U,R] = qr_House(A).

Agora, implementarei o Método de Householder para determinar a decomposição QR de uma matriz A e a função que constrói a matriz ortogonal Q da decomposição A:

```
function [U,R]=qr_House(A)

m = size(A,1);
n = size(A,2);
U = zeros(m,n);
R = eye(m,n);

for k=1:n
    x = A(k:m,k);
    if x(1) < 0 then
        x(1) = x(1) - norm(x);
    else
        x(1) = x(1) + norm(x);
    end
    u = x / norm(x);
    U(k:m,k) = u;
    A(k:m,k:n) = A(k:m,k:n) - 2 * u * (u' * A(k:m,k:n));
end

R = triu(A);

endfunction
```

```
function [Q]=constroi_Q_House(U)

m = size(U,1);
n = size(U,2);
u_1 = U(:,1);
Q = eye(m,m) - 2 * u_1 * u_1';

for k=2:n
    u_k = U(:,k);
    Q_k = eye(m,m) - 2 * u_k * u_k';
    Q = Q * Q_k;
end

endfunction
```

3.1 4.1)

Irei testar com as funções que criei anteriormente: $\begin{bmatrix} 2 & 1 & 2 \\ 5 & -2 & 3 \\ 1 & 0 & 4 \end{bmatrix}$, $\begin{bmatrix} 3 & 2 & 7 & -2 \\ 5 & -1 & 2 & 5 \\ 2 & -7 & -4 & 12 \\ 8 & 10 & 3 & 1 \end{bmatrix}$, $\begin{bmatrix} 5 & -2 & 1 & 8 & 3 \\ -1 & 6 & 5 & 3 & 9 \\ 6 & 5 & 4 & 2 & 1 \\ 4 & 2 & 6 & 1 & 2 \\ 7 & 5 & 6 & 3 & 2 \end{bmatrix}$

3.1.1 3x3

Figura 10: Matriz 3x3 - item 4

```
--> [U,R]=qr_House(A)
 U  =

   0.8261805   0.           0.
   0.5524646  -0.9998345    0.
   0.1104929   0.0181941    1.
 R  =

  -5.4772256   1.4605935  -4.1992063
   0.          1.6931233   1.2599988
   0.          0.         -3.1271504

--> [Q]=constroi_Q_House(U)
 Q  =

  -0.3651484   0.9056241   0.2156655
  -0.9128709  -0.3937496   0.1078328
  -0.1825742   0.1574998  -0.970495

--> Q' * Q
 ans  =

   1.          2.123D-16   5.971D-17
   2.123D-16   1.         -6.532D-18
   8.747D-17  -6.532D-18   1.
```

Como vimos, a precisão do modelo ainda é muito boa, assim como nas funções implementadas anteriormente. Mas essa precisão se mostra ainda melhor porque esse método não faz a ortogonalização de vetores, que pode fazer com que o modelo não seja tão estável quando trabalhamos com vetores que possam ser "quase"múltiplos um do outro. As matrizes Q e R resultantes são diferentes das encontradas nos outros métodos, mas ainda satisfazem $A = QR$ e $Q^TQ = I$.

3.1.2 4x4

Figura 11: Matriz 4x4 - item 4

```
--> [U,R]=qr_House(A)
 U  =

   0.8053087   0.          0.          0.
   0.3073814  -0.8395183   0.          0.
   0.1229525  -0.4723817  -0.8226475   0.
   0.4918102   0.2684485  -0.5685517  -1.
 R  =

  -10.099505  -6.6339885  -4.6536935  -5.0497525
   0.          10.487621   3.7308223  -11.108334
   0.          0.          6.5133787  -4.7681519
   0.          0.          0.          1.5393632

--> [Q]=constroi_Q_House(U)
 Q  =

  -0.2970443   0.0028044   0.8608718   0.413106
  -0.4950738  -0.4085115   0.1873317  -0.7435907
  -0.1980295  -0.792718   -0.3015453   0.4913787
  -0.792118    0.4524475  -0.3645229   0.1869848

--> Q' * Q
 ans  =

   1.         -5.662D-17  -2.047D-16  -1.451D-16
  -5.662D-17   1.          -6.306D-17   1.653D-16
  -2.047D-16  -6.306D-17   1.          -2.622D-16
  -1.451D-16   1.653D-16  -2.622D-16   1.
```

Assim com na matriz 3x3, esse exemplo funcionou da mesma forma que as implementações anteriores, além dos detalhes que comentei em relação a matriz 3x3.

3.1.3 5x5

Figura 12: Matriz 5x5 - item 4

```
--> [U,R]=qr_House(A)
 U  =

   0.8496112   0.          0.          0.          0.
  -0.0522213   0.9349085   0.          0.          0.
   0.313328    0.2503704  -0.9048665   0.          0.
   0.2088853   0.0807038   0.4088864  -0.9621331   0.
   0.3655493   0.2382177  -0.1184419  -0.2725801  -1.
 R  =

  -11.269428  -5.0579321  -7.9862086  -6.5664382  -3.0170121
   0.         -8.2714765  -5.9972726   0.5092922  -6.2552505
   0.          0.          3.7753402   1.4868645   3.8118602
   0.          0.          0.          6.4351958   6.0148919
   0.          0.          0.          0.          0.2455116

--> [Q]=constroi_Q_House(U)
 Q  =

  -0.4436783   0.5131      0.141416    0.7171545   0.0746249
   0.0887357  -0.7796454   0.2735953   0.5552194   0.0340007
  -0.5324139  -0.2789208  -0.5098169  -0.0926129   0.6084802
  -0.3549426  -0.0247506   0.7991115  -0.3894635   0.2883437
  -0.6211496  -0.2246597  -0.0815756  -0.1310031  -0.7347686

--> Q' * Q
 ans  =

   1.          1.557D-16   3.752D-17   1.784D-16   2.272D-18
   1.557D-16   1.          7.957D-18  -8.325D-17   3.692D-17
   3.752D-17   7.957D-18   1.         -3.094D-16   2.050D-16
   1.784D-16  -8.325D-17  -3.094D-16   1.          1.454D-16
   5.778D-17   9.167D-18   2.258D-16   1.593D-16   1.
```

A mesma situação se repete na matriz 5x5.

3.2 4.2)

Agora, vou testar os meus métodos com a matriz $\begin{bmatrix} 0.7 & 0.70711 \\ 0.70001 & 0.70711 \end{bmatrix}$.

A seguir, os resultados:

Figura 13: GS

```
--> [Q,R]=qr_GS(A)
 Q   =

    0.7071017    0.7071118
    0.7071118   -0.7071017
 R   =

    0.9899566    1.0000046
    0.           0.0000071

--> Q'*Q
 ans   =

    1.           2.301D-11
    2.301D-11    1.
```

Figura 14: GSM

```
--> [Q,R]=qr_GSM(A)
 Q  =

   0.7071017   0.7071118
   0.7071118  -0.7071017
 R  =

   0.9899566   1.0000046
   0.          0.0000071

--> Q' * Q
 ans  =

   1.          2.301D-11
   2.301D-11   1.
```

Figura 15: Holseholder

```
--> [U,R]=qr_House(A)
 U  =

   0.9238782   0.
   0.3826867  -1.
 R  =

  -0.9899566  -1.0000046
   0.          0.0000071

--> [Q]=constroi_Q_House(U)
 Q  =

  -0.7071017   0.7071118
  -0.7071118  -0.7071017

--> Q' * Q
 ans  =

   1.    0.
   0.    1.
```

Todas os métodos satisfazem a condição $A = QR$ e $Q^TQ = I$. Entretanto, os vetores colunas são muito próximos, e, como comentado na questão anterior, Gram-Schmidt e Gram-Schmidt Modificado não são tão estáveis quando trabalhamos com vetores quase múltiplos uns dos outros. Isso não vai ocorrer na aplicação do método de Householder, e podemos notar isso ao testarmos a ortogonalidade das matrizes Q que obtivemos.

3.2.1 4.3)

Agora, vou calcular a matriz proposta com o Método de Gram-Schmidt, HouseHolder e com a função QR do scilab:

Figura 16: 4.3

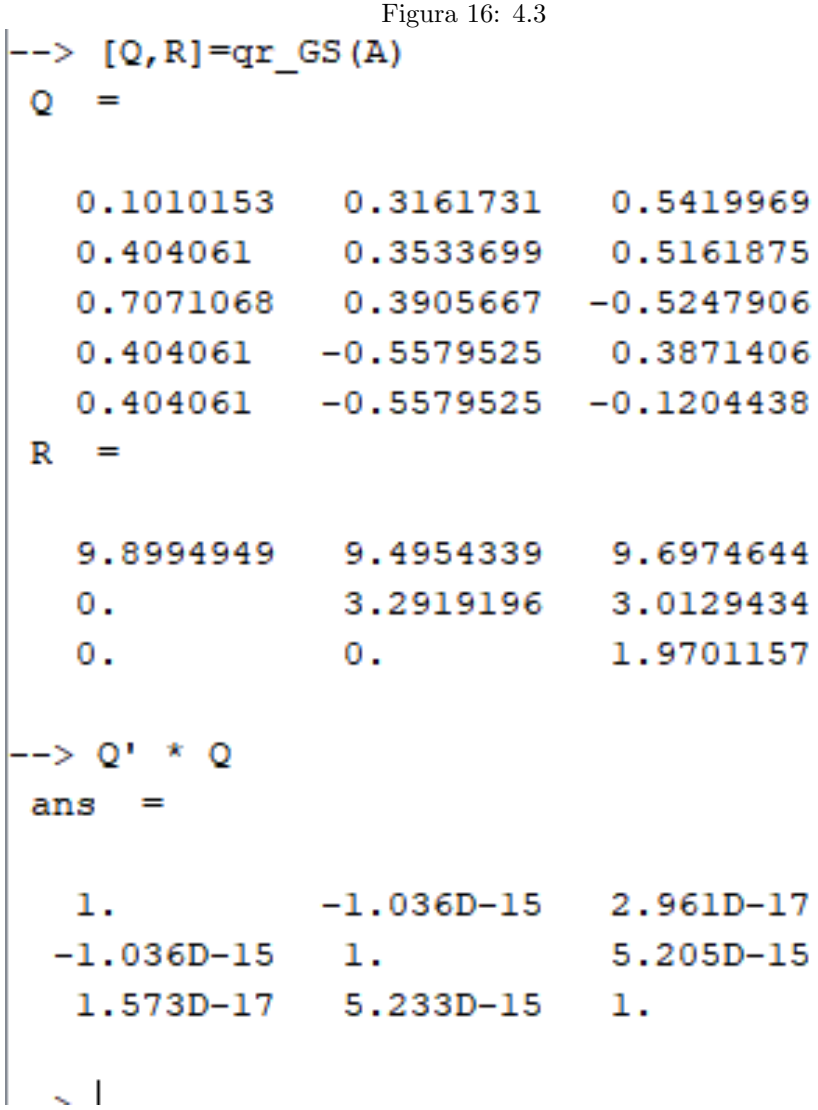

Figura 17: 4.3

```
--> [U,R]=qr_House(A)
 U  =

   0.741962    0.          0.
   0.2722923   0.7865551   0.
   0.4765114   0.1191972  -0.9800408
   0.2722923  -0.4284409   0.1842055
   0.2722923  -0.4284409  -0.0747553
 R  =

  -9.8994949  -9.4954339  -9.6974644
   0.         -3.2919196  -3.0129434
   0.          0.          1.9701157
   0.          0.          0.
   0.          0.          0.

--> [Q]=constroi_Q_House(U)
 Q  =

  -0.1010153  -0.3161731   0.5419969  -0.6842085  -0.3576711
  -0.404061   -0.3533699   0.5161875   0.3280084   0.5812274
  -0.7071068  -0.3905667  -0.5247906   0.0093972  -0.2682612
  -0.404061    0.5579525   0.3871406   0.3655973  -0.4918175
  -0.404061    0.5579525  -0.1204438  -0.5389987   0.4694651

--> Q'*Q
 ans  =

   1.          6.216D-17  -3.187D-17   7.261D-17  -5.803D-17
   6.216D-17   1.         -1.685D-16  -1.677D-16  -3.222D-16
  -3.187D-17  -1.685D-16   1.         -1.333D-16   1.389D-16
   7.261D-17  -1.677D-16  -1.333D-16   1.          1.674D-16
  -8.579D-17  -3.222D-16   1.805D-16   2.229D-16   1.
```

Figura 18: 4.3

```
--> [Q,R]=qr(A)
 Q  =

 -0.1010153  -0.3161731   0.5419969  -0.6842085  -0.3576711
 -0.404061   -0.3533699   0.5161875   0.3280084   0.5812274
 -0.7071068  -0.3905667  -0.5247906   0.0093972  -0.2682612
 -0.404061    0.5579525   0.3871406   0.3655973  -0.4918175
 -0.404061    0.5579525  -0.1204438  -0.5389987   0.4694651
 R  =

 -9.8994949  -9.4954339  -9.6974644
  0.         -3.2919196  -3.0129434
  0.          0.          1.9701157
  0.          0.          0.
  0.          0.          0.

--> Q'*Q
 ans  =

  1.          4.407D-17   5.254D-18   1.913D-17   7.260D-17
  4.407D-17   1.          1.353D-17   2.488D-16   1.971D-16
  5.254D-18   1.353D-17   1.         -1.242D-16  -5.542D-17
  1.913D-17   2.488D-16  -1.242D-16   1.         -7.352D-17
  7.260D-17   1.971D-16  -5.542D-17  -7.352D-17   1.
```

Assim como nos exemplos de outras questões, todos os métodos satisfizeram as condições $A = QR$ e $Q^TQ = I$, apesar do Método de Gram-Schmidt nos retornar matrizes diferentes do método de householder e da função qr do scilab. A função do scilab nos dá valores mais próximos da matriz identidade, enquanto o método de householder também é muito próximo, mas não tanto quanto a função do scilab. Por último, o método de Gram-Schmidt é o "pior dos melhores", pois é o menos estável dessas 3 funções.

4 Questão 5

Escreva uma função Scilab function [S] = espectro(A, tol) que calcula os autovalores de uma matriz simétrica A usando o Algoritmo QR. Os autovalores calculados devem ser devolvidos no vetor S. Use como critério de parada a norma infinito da diferença entre dois espectros consecutivos menor do que uma tolerância tol dada ($10^{-3}, 10^{-4}, 10^{-5}$, ...). Teste a sua função com matrizes simétricas das quais você saiba quais são os autovalores.

Primeiro, mostro a função criada para a questão:

```
function [S] = espectro(A, tol)

[Q_inicial,R_inicial]=qr_GSM(A);
A_prox = R_inicial * Q_inicial;
e = norm(diag(A_prox) - diag(A), %inf);

while e > tol
    A = A_prox;
    [Q_prox,R_prox]=qr_GSM(A);
    A_prox = R_prox * Q_prox;
    e = norm(diag(A_prox) - diag(A), %inf);
end

S = diag(A_prox);
endfunction
```

Primeiro, vamos testar com a matriz $\begin{bmatrix} 4 & 6 & 3 \\ 6 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$. Agora, usarei o código com essa matriz, sabendo que os seus autovalores dela são $\lambda = -2.916$, $\lambda = 1.403$, $\lambda = 10.513$.

Figura 19: 5.1

```
--> A = [4 6 3; 6 3 1;3 1 2]
 A  =

    4.    6.    3.
    6.    3.    1.
    3.    1.    2.

--> [S] = espectro(A, 10^(-10))
 S  =

    10.513314
   -2.9159584
    1.4026441
```

Como podemos ver, os resultados dos autovalores conferem, com um pequeno erro de arredondamento. Agora, vou testar com uma matriz 4x4:

Figura 20: 5.2

```
 A  =

    4.    3.    2.    1.
    3.    5.    2.    1.
    2.    2.    9.    2.
    1.    1.    2.    2.

--> [S] = espectro(A, 10^(-10))
 S  =

    11.796686
    5.3671131
    1.4897797
    1.3464211

--> spec(A)
 ans  =

    1.3464211
    1.4897797
    5.3671131
    11.796686
```

,

Como podemos ver, os resultados bateram. Para conferir isso, usei a função spec() do scilab que calcula os autovalores da matriz.