Aula prática 5
Decomposição QR

Will Sena*

Contents

*wllsena@protonmail.com

1) Método de Gram-Schmidt

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, n] = size(A);
  Q      = zeros(m, n);
  R      = zeros(n, n);

  for j = 1:n
    v = A(:, j);

    for i = 1:j-1
      R(i, j) = Q(:, i)' * A(:, j);
      v       = v - R(i, j) * Q(:, i);
    end

    R(j, j) = norm(v);
    Q(:, j) = v / R(j, j);
  end
endfunction
```

Teste 1:

```
--> A = rand(3, 3)
A  =

0.1391634   0.780693    0.1110518
0.1121413   0.0620503   0.7938458
0.3280001   0.8427351   0.2010768

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

```
0.3725609   0.8268955   0.4212389
0.3002186  -0.5369026   0.7884189
0.8781043  -0.1672703  -0.4482783
R  =

0.373532   1.0494936   0.4562673
0.         0.4712721  -0.3680238
0.         0.          0.582524

--> norm(Q * R - A)
ans  =

6.939D-18

--> norm(Q' * Q - eye())
ans  =

6.383D-16
```

Teste 2:

```
--> B = rand(5, 5)
B  =

0.025871    0.4236102   0.7064868   0.4498763   0.9677053
0.5174468   0.2893728   0.5211472   0.7227253   0.5068534
0.3916873   0.0887932   0.2870401   0.8976796   0.5232976
0.2413538   0.6212882   0.6502795   0.2427822   0.5596948
0.5064435   0.3454984   0.0881335   0.4337721   0.5617307

--> [Q R] = qr_GS(B)
Q  =

0.0301442   0.6339764   0.4394116   0.3984955   0.4952616
0.6029161  -0.119198    0.4049945  -0.6251949   0.259605
0.4563843  -0.2939525   0.4074913   0.5657128  -0.4682148
0.2812195   0.705044   -0.1749962  -0.220357   -0.587067
0.5900953  -0.0192531  -0.6680001   0.2859097   0.3513522
R  =
```

```
0.8582402   0.6063561   0.7013836   1.1832328   1.0624574
0.          0.6393488   0.7581789   0.0980091   0.7830562
0.          0.          0.4657969   0.5239315   0.3705524
0.          0.          0.          0.3057793   0.4020518
0.          0.          0.          0.          0.2346202

--> norm(Q * R - B)
ans  =

1.614D-16

--> norm(Q' * Q - eye())
ans  =

3.144D-15
```

Teste 3:

```
--> C = rand(50, 50);

--> [Q R] = qr_GS(C);

--> norm(Q * R - C)
ans  =

1.891D-15

--> norm(Q' * Q - eye())
ans  =

7.575D-13
```

A função teve bons resultados nos 3 testes.

2) Método de Gram-Schmidt Modificado

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.

```
function [Q, R] = qr_GSM(A)
  [m, n] = size(A);
  Q      = zeros(m, n);
  R      = zeros(n, n);

  for j = 1:n
    v = A(:, j);

    for i = 1:j-1
      R(i, j) = Q(:, i)' * v;
      v       = v - R(i, j) * Q(:, i);
    end

    R(j, j) = norm(v);
    Q(:, j) = v / R(j, j);
  end
endfunction
```

Teste 1:

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

0.3725609   0.8268955   0.4212389
0.3002186  -0.5369026   0.7884189
0.8781043  -0.1672703  -0.4482783
R  =

0.373532   1.0494936   0.4562673
0.         0.4712721  -0.3680238
0.         0.          0.582524
```

```
--> norm(Q * R - A)
ans  =

6.939D-18

--> norm(Q' * Q - eye())
ans  =

4.658D-16
```

Teste 2:

```
--> [Q R] = qr_GSM(B)
Q  =

0.0301442   0.6339764   0.4394116   0.3984955   0.4952616
0.6029161  -0.119198    0.4049945  -0.6251949   0.259605
0.4563843  -0.2939525   0.4074913   0.5657128  -0.4682148
0.2812195   0.705044   -0.1749962  -0.220357   -0.587067
0.5900953  -0.0192531  -0.6680001   0.2859097   0.3513522
R  =

0.8582402   0.6063561   0.7013836   1.1832328   1.0624574
0.          0.6393488   0.7581789   0.0980091   0.7830562
0.          0.          0.4657969   0.5239315   0.3705524
0.          0.          0.          0.3057793   0.4020518
0.          0.          0.          0.          0.2346202

--> norm(Q * R - B)
ans  =

1.110D-16

--> norm(Q' * Q - eye())
ans  =

2.478D-15
```

Teste 3:

```
--> [Q R] = qr_GSM(C);

--> norm(Q * R - C)
ans  =

1.941D-15

--> norm(Q' * Q - eye())
ans  =

1.825D-14
```

Os resultados foram levemente melhores aos do método anterior

3) Método de Gram-Schmidt Modificado com Pivoteamento de Colunas

Escreva uma função Scilab *function [Q,R] = qr_ GSP(A)* que implementa o Método de Gram-Schmidt Modificado com Pivoteamento de Colunas. Nesse Método, na primeira iteração, escolhe-se a maior entre as colunas da matriz A para ser a primeira, fazendo a troca necessária, e normalizando para obter q_1. A partir da segunda iteração, a cada iteração subtrai-se das colunas restantes as projeções delas sobre o subespaço gerado pela última coluna ortonormal obtida e escolhe-se aquela de maior norma resultante para ser a próxima coluna processada. Para tal faz-se a troca necessária. O objetivo desse procedimento é escolher sempre a "melhor coluna", isto é, a mais independente das anteriores. Essa função deverá retornar também a matriz de permutação P que contém as trocas de colunas efetuadas, de forma que $AP = QR$. Testar a sua função com as mesmas matrizes usadas nos testes dos itens anteriores. Comparar a precisão e estabilidade dos Métodos.

```
1  function [Q, R, P] = qr_GSP(A)
2    [m, n] = size(A);
```

```
  Q      = zeros(m, n);
  R      = zeros(n, n);
  P      = eye(n, n);

  for j = 1:n
    [_, p] = max(sum(A(:, [j:n]) .^2, 1));
    p = j + p - 1;
    A(:, [j, p]) = A(:, [p, j]);
    R(:, [j, p]) = R(:, [p, j]);
    P(:, [j, p]) = P(:, [p, j]);

    R(j, j) = norm(A(:, j));
    Q(:, j) = A(:, j) / R(j, j);

    for i = j+1:n
      R(j, i) = Q(:, j)' * A(:, i);
      A(:, i) = A(:, i) - R(j, i) * Q(:, j);
    end
  end
endfunction
```

Teste 1:

```
--> [Q R P] = qr_GSP(A)
Q  =

0.6785982  -0.0882889  -0.7291842
0.0539357   0.9960591  -0.0704078
0.7325267   0.0084495   0.6806858
R  =

1.1504496   0.2654704   0.3407533
0.          0.7826117   0.1021842
0.          0.          0.1138936
P  =

0.   0.   1.
1.   0.   0.
0.   1.   0.
```

```
--> norm(Q * R * P' - A)
ans  =

0.

--> norm(Q' * Q - eye())
ans  =

1.399D-15
```

Teste 2:

```
--> [Q R P] = qr_GSP(B)
Q  =

0.668418   -0.4521641  -0.0437027  -0.5250074  -0.266875
0.3500962   0.4682713   0.4228484   0.3194656  -0.6142439
0.3614546   0.7027792  -0.1084764  -0.3758047   0.4716522
0.386595   -0.2867278   0.5255934   0.4038146   0.5735999
0.3880013  -0.0125789  -0.7288853   0.5639269   0.0030846
R  =

1.4477548   1.1403617   1.0440228   0.6298329   0.7907924
0.          0.6908154  -0.0612457   0.4303039  -0.1761209
0.          0.          0.4358972  -0.0671031   0.1689321
0.          0.          0.          0.3875854   0.2823983
0.          0.          0.          0.          0.1085196
P  =

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

--> norm(Q * R * P' - B)
ans  =
```

```
6.622D-17

--> norm(Q' * Q - eye())
ans  =

2.176D-15
```

Teste 3:

```
--> [Q R P] = qr_GSP(C);

--> norm(Q * R * P' - C)
ans  =

1.969D-15

--> norm(Q' * Q - eye())
ans  =

1.827D-14
```

Resultados ligeiramente superiores aos de ambos os métodos de Gram-Schmidt.

4) Método de Householder

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)*.

```
1 function [U, R] = qr_House(A)
2   [m, n] = size(A);
```

```
  U      = zeros(m, n);

  for k = 1:n
    x = A(k:m, k)

    if x(1) < 0
      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, n] = size(U);
  Q      = eye(m, m);

  for i = 1:n
    Q = Q * (eye(m, m) - 2 * U(:, i) * U(:, i)');
  end
endfunction
```

4.1)

Testar as suas funções com as mesmas matrizes usadas nos testes dos itens anteriores. Comparar a precisão dos Métodos.

Teste 1:

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

0.8284204   0.          0.
0.1811994  -0.9267601   0.
```

```
0.5299871  -0.3756538  -1.
R  =

-0.373532  -1.0494936  -0.4562673
0.         0.4712721  -0.3680238
0.         0.          0.582524

--> Q = constroi_Q_House(U)
Q  =

-0.3725609   0.8268955   0.4212389
-0.3002186  -0.5369026   0.7884189
-0.8781043  -0.1672703  -0.4482783

--> norm(Q * R - A)
ans  =

4.781D-16

--> norm(Q' * Q - eye())
ans  =

4.454D-16
```

Teste 2:

```
--> [U R] = qr_House(B)
U  =

0.7176852   0.          0.          0.          0.
0.4200422  -0.8632055   0.          0.          0.
0.3179557  -0.3329582   0.760198    0.          0.
0.1959212   0.308139   -0.1592904  -0.8615823   0.
0.4111101  -0.2215072  -0.6298616   0.5076179  -1.
R  =

-0.8582402  -0.6063561  -0.7013836  -1.1832328  -1.0624574
0.          0.6393488   0.7581789   0.0980091   0.7830562
0.          0.         -0.4657969  -0.5239315  -0.3705524
```

```
0.          0.          0.          0.3057793   0.4020518
0.          0.          0.          0.          0.2346202

--> Q = constroi_Q_House(U)
Q  =

-0.0301442   0.6339764  -0.4394116   0.3984955   0.4952616
-0.6029161  -0.119198   -0.4049945  -0.6251949   0.259605
-0.4563843  -0.2939525  -0.4074913   0.5657128  -0.4682148
-0.2812195   0.705044    0.1749962  -0.220357   -0.587067
-0.5900953  -0.0192531   0.6680001   0.2859097   0.3513522

--> norm(Q * R - B)
ans  =

2.234D-15

--> norm(Q' * Q - eye())
ans  =

1.529D-15
```

Teste 3:

```
--> [U R] = qr_House(C);

--> Q = constroi_Q_House(U);

--> norm(Q * R - C)
ans  =

1.771D-14

--> norm(Q' * Q - eye())
ans  =

4.411D-15
```

Este foi o melhor método para obter matrizes Q "mais ortogonais", isto

é Q^TQ mais próximo da matrix identidade, porem foi o pior método para encontrar matrizes Q e R tal que $QR = A$.

4.2)

Testar as suas funções com a matriz $A = \begin{bmatrix} 0.70000 & 0.70711 \\ 0.70001 & 0.70711 \end{bmatrix}$. Comparar a ortogonalidade das matrizes Q produzidas pelos Métodos.

```
--> A = [0.70000 0.70711; 0.70001 0.70711]
A  =

0.7       0.70711
0.70001   0.70711

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

--> norm(Q' * Q - eye())
ans  =

2.301D-11

--> [Q R] = qr_GSM(A);

--> norm(Q' * Q - eye())
ans  =

2.301D-11

--> [Q R P] = qr_GSP(A);

--> norm(Q' * Q - eye())
ans  =

2.220D-11

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

```
--> Q = contri_Q_House(U);

--> norm(Q' * Q - eye())
ans  =

1.111D-16
```

Equivalente aos resultados das questões anteriores, o método de Householder produz matrizes Q "mais ortogonais", em seguida o método de Gram-Schmidt Modificado com Pivoteamento de Coluna.

4.3)

Seja a matriz $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 7 \\ 4 & 2 & 3 \\ 4 & 2 & 2 \end{bmatrix}$**. Calcule a decomposição** QR **(reduzida) usando os métodos de Gram-Schmidt, Householder e a função "qr" do Scilab. Compare os três resultados. Comente.**

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

1.   2.   3.
4.   5.   6.
7.   8.   7.
4.   2.   5.
4.   2.   2.
```

Método de Gram-Schmidt Modificado com Pivoteamento de Coluna (o melhor entre os métodos de Gram-Schmidt):

```
--> tic(); [Q R P] = qr_GSP(A),
printf("\nTime: " + string(toc()));
Q  =
```

```
0.2705009  -0.4843384   0.0295965
0.5410018  -0.3382681   0.1061991
0.6311687   0.3408307   0.5379595
0.4508348  -0.0717538  -0.7938819
0.1803339   0.727789   -0.2611454
R  =

11.090537   9.3773642   9.5576981
0.          3.1725448   1.3786988
0.          0.          2.7838096
P  =

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

Time: 0.003871

--> norm(Q * R * P' - A)
ans  =

4.441D-16

--> norm(Q' * Q - eye())
ans  =

2.777D-16
```

Método de Householder:

```
--> tic(); [U R] = qr_House(A), Q = constroi_Q_House(U),
printf("\nTime: " + string(toc()));
U  =

0.741962    0.          0.
0.2722923   0.7865551   0.
0.4765114   0.1191972  -0.9233424
0.2722923  -0.4284409   0.2964374
0.2722923  -0.4284409  -0.2440566
```

```
R  =

-9.8994949  -9.4954339  -10.505586
0.          -3.2919196  -1.8970384
0.           0.          3.0056444
0.           0.          0.
0.           0.          0.
Q  =

-0.1010153  -0.3161731   0.4454895  -0.817697   -0.1508018
-0.404061   -0.3533699   0.3609029   0.2503195   0.7203844
-0.7071068  -0.3905667  -0.3890984   0.105686   -0.429989
-0.404061    0.5579525   0.6033845   0.2243545  -0.3331905
-0.404061    0.5579525  -0.3947375  -0.4552002   0.4029873

Time: 0.00336

--> norm(Q * R - A)
ans  =

3.441D-15

--> norm(Q' * Q - eye())
ans  =

8.169D-16
```

Função "qr" do Scilab:

```
--> tic(); [Q R] = qr(A),
printf("\nTime: " + string(toc()));
Q  =

-0.1010153  -0.3161731   0.4454895  -0.817697   -0.1508018
-0.404061   -0.3533699   0.3609029   0.2503195   0.7203844
-0.7071068  -0.3905667  -0.3890984   0.105686   -0.429989
-0.404061    0.5579525   0.6033845   0.2243545  -0.3331905
-0.404061    0.5579525  -0.3947375  -0.4552002   0.4029873
R  =
```

```
 -9.8994949  -9.4954339  -10.505586
 0.          -3.2919196  -1.8970384
 0.           0.          3.0056444
 0.           0.          0.
 0.           0.          0.

Time: 0.002297

--> norm(Q * R - A)
ans  =

2.997D-15

--> norm(Q' * Q - eye())
ans  =

1.895D-16
```

A função "qr" do Scilab é mais rápida e com melhores resultados, porém retorna Q e R similares aos do Método de Householder, provavelmente é uma implementação deste método com otimizações.

5) Algoritmo QR para autovalores

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.

```
function [S] = espectro(A, tol)
  d = diag(A);
  while %t
    [Q R] = qr_GSM(A);
```

```
    d_    = d;
    A     = R * Q;
    if norm(d_ - d, %inf) < tol
      S = gsort(d, 'g','i');
      return;
    end
  end
endfunction
```

Teste 1:

```
--> A = rand(3, 3); A = triu(A) + triu(A, 1)'
A  =

0.7770124   0.6051367   0.0312945
0.6051367   0.7429886   0.8922392
0.0312945   0.8922392   0.010713

--> espectro(A, 10^(-3))
ans  =

-0.6617782
0.5301518
1.6623404

--> spec(A)
ans  =

-0.6617906
0.5301642
1.6623404
```

Teste 2:

```
--> A = rand(5, 5); A = triu(A) + triu(A, 1)'
A  =

0.9864259   0.0479586   0.6881744   0.6855798   0.3935583
```

```
0.0479586   0.7368848   0.9940326   0.6273061   0.97193
0.6881744   0.9940326   0.4526969   0.3971971   0.4964329
0.6855798   0.6273061   0.3971971   0.10704     0.1112268
0.3935583   0.97193     0.4964329   0.1112268   0.8805518

--> espectro(A, 10^(-4))
ans  =

-0.8416772
-0.1919457
0.2818675
1.0342720
2.8810829

--> spec(A)
ans  =

-0.8416773
-0.1919457
0.2818675
1.0342720
2.8810829
```

Teste 3:

```
--> A = rand(10, 10); A = triu(A) + triu(A, 1)';

--> espectro(A, 10^(-5))
ans  =

-1.2238885
-1.0757530
-0.9420898
-0.6958281
-0.2771740
0.2059337
0.6952506
0.8435077
1.1240412
```

```
4.7391524

--> spec(A)
ans  =

-1.2238885
-1.0757530
-0.9420898
-0.6959288
-0.2771740
0.2059337
0.6953514
0.8435077
1.1240412
4.7391524
```

Não tem muito o que dizer, os resultados são muito bons.