Atividade prática 5
Álgebra Linear Numérica

Pedro Henrique Coterli

13 de junho de 2024

Exercício 1

Façamos nossa função de decomposição QR com Gram-Schmidt.

```
// Função de decomposição QR de Gram-Schmidt
function [Q, R] = qr_GS(A)
    // Pegando as dimensões da A
    [m, n] = size(A)
    // Inicializando as matrizes Q e R
    Q = zeros(m, n)
    R = zeros(n, n)

    // Para cada coluna...
    for j = 1:n
        // Inicializa o vetor v como a coluna de A
        v = A(:, j)
        // Se não for a primeira coluna...
        if j > 1 then
            // Calcula o termo de projeção de v
            R(1:(j-1), j) = (Q(:, 1:(j-1)))' * A(:, j)
            // Calcula a ortogonalização de v
            v = v - Q(:, 1:(j-1)) * R(1:(j-1), j)
        end
        // Salva o termo da diagonal de R como a norma de v
        R(j, j) = norm(v)
        // Salva a coluna de Q como o vetor v normalizado
        Q(:, j) = v / R(j, j)
    end
endfunction
```

Vamos testá-la com algumas matrizes retangulares, calculando os erros do produto QR com a matriz A e da ortogonalidade da matriz Q.

```
// Gerando uma matriz
A1 = round(20*rand(3, 2) - 10)
// Calculando a decomposição QR
```

```
[Q1, R1] = qr_GS(A1);
disp("norm(Q*R - A)", norm(Q1*R1 - A1))
disp("norm(Q^T*Q - I)", norm(Q1'*Q1 - eye(2, 2)))
```

```
A1  =
  2.   9.
 -2.   7.
  4.  -3.

 "norm(Q*R - A)"

  1.776D-15

 "norm(Q^T*Q - I)"

  2.292D-16
```

```
A2 = round(20*rand(4, 3) - 10)
[Q2, R2] = qr_GS(A2);
disp("norm(Q*R - A)", norm(Q2*R2 - A2))
disp("norm(Q^T*Q - I)", norm(Q2'*Q2 - eye(3, 3)))
```

```
A2  =
  8.   9.    5.
 -8.   10.   1.
  1.   2.    5.
  1.   10.  -9.

 "norm(Q*R - A)"

  1.891D-15

 "norm(Q^T*Q - I)"

  2.562D-16
```

```
A3 = round(20*rand(5, 5) - 10)
[Q3, R3] = qr_GS(A3);
disp("norm(Q*R - A)", norm(Q3*R3 - A3))
disp("norm(Q^T*Q - I)", norm(Q3'*Q3 - eye(5, 5)))
```

```
A3  =
 -2.   8.   6.  -5.   4.
 -8.   5.   4.   4.  -4.
 -5.   1.  -8.   9.  -7.
 -3.  -2.  -5.   0.   4.
 -7.   8.   5.  -5.  -1.
```

```
"norm(Q*R - A)"

 8.882D-16

"norm(Q^T*Q - I)"

 6.279D-16
```

Parece que nossa função está funcionando bem.

Exercício 2

Agora, façamos a função da decomposição QR com Gram-Schmidt modificado. Infelizmente, para essa versão, não é possível calcular o loop interno matricialmente devido à atualização do vetor v, sendo necessário um loop `for`.

```
// Função de decomposição QR de Gram-Schmidt
function [Q, R] = qr_GSM(A)
    // Pegando as dimensões da A
    [m, n] = size(A)
    // Inicializando as matrizes Q e R
    Q = zeros(m, n)
    R = zeros(n, n)

    // Para cada coluna...
    for j = 1:n
        // Inicializa o vetor v como a coluna de A
        v = A(:, j)
        // Para cada linha...
        for i = 1:(j-1)
            // Calcula o fator de projeção do v sobre qi
            R(i, j) = Q(:, i)' * v
            // Ortogonaliza em relação ao qi
            v = v - R(i, j) * Q(:, i)
        end
        // Salva o termo da diagonal de R como a norma de v
        R(j, j) = norm(v)
        // Salva a coluna de Q como o vetor v normalizado
        Q(:, j) = v / R(j, j)
    end
endfunction
```

Vamos agora testá-la com as mesmas matrizes usadas no exercício anterior, comparando a precisão dos dois algoritmos.

```
// Calculando a decomposição QR pelo GS modificado
[QM1, RM1] = qr_GSM(A1);
```

```
disp("=============== Gram-Schmidt ================")
disp("norm(Q*R - A)", norm(Q1*R1 - A1))
disp("norm(Q^T*Q - I)", norm(Q1'*Q1 - eye(2, 2)))
disp("========== Gram-Schmidt modificado ==========")
disp("norm(Q*R - A)", norm(QM1*RM1 - A1))
disp("norm(Q^T*Q - I)", norm(QM1'*QM1 - eye(2, 2)))
```

```
"=============== Gram-Schmidt ================"

"norm(Q*R - A)"

 1.776D-15

"norm(Q^T*Q - I)"

 2.292D-16

"========== Gram-Schmidt modificado =========="

"norm(Q*R - A)"

 1.776D-15

"norm(Q^T*Q - I)"

 2.292D-16
```

```
[QM2, RM2] = qr_GSM(A2);

disp("=============== Gram-Schmidt ================")
disp("norm(Q*R - A)", norm(Q2*R2 - A2))
disp("norm(Q^T*Q - I)", norm(Q2'*Q2 - eye(3, 3)))
disp("========== Gram-Schmidt modificado ==========")
disp("norm(Q*R - A)", norm(QM2*RM2 - A2))
disp("norm(Q^T*Q - I)", norm(QM2'*QM2 - eye(3, 3)))
```

```
"=============== Gram-Schmidt ================"

"norm(Q*R - A)"

 1.891D-15

"norm(Q^T*Q - I)"

 2.562D-16

"========== Gram-Schmidt modificado =========="
```

```
"norm(Q*R - A)"

 1.891D-15

"norm(Q^T*Q - I)"

 2.562D-16
```

```
[QM3, RM3] = qr_GSM(A3);

disp("=============== Gram-Schmidt ================")
disp("norm(Q*R - A)", norm(Q3*R3 - A3))
disp("norm(Q^T*Q - I)", norm(Q3'*Q3 - eye(5, 5)))
disp("========== Gram-Schmidt modificado ==========")
disp("norm(Q*R - A)", norm(QM3*RM3 - A3))
disp("norm(Q^T*Q - I)", norm(QM3'*QM3 - eye(5, 5)))
```

```
"=============== Gram-Schmidt ================"

"norm(Q*R - A)"

 8.882D-16

"norm(Q^T*Q - I)"

 6.279D-16

"========== Gram-Schmidt modificado =========="

"norm(Q*R - A)"

 1.776D-15

"norm(Q^T*Q - I)"

 5.514D-16
```

Nota-se que, para matrizes comuns, os algoritmos obtiveram desempenhos bem similares, não variando muito o erro numérico tanto da aproximação da QR para a A quanto da ortogonalidade da Q.

Exercício 4

Agora, vamos escrever funções para calcular a decomposição QR de uma matriz com base no refletor de Householder. Primeiro, temos a versão normal do algoritmo escrita abaixo.

```
// Função da versão 1 do algoritmo de Householder para decomposição QR
function [U, R] = qr_House_v1(A)
```

```
    // Pegando as dimensões da A
    [m, n] = size(A)
    // Inicializando a matriz U
    U = zeros(m, n)

    // Para cada coluna...
    for j = 1:n
        // Pega o vetor que queremos projetar sobre um eixo
        x = A(j:m, j)
        // Escolhe a melhor das duas projeções possíveis
        if x(1) > 0 then
            x(1) = x(1) + norm(x)
        else
            x(1) = x(1) - norm(x)
        end
        // Calcula u normalizando x
        u = x/norm(x)
        // Salva u na matriz U
        U(j:m, j) = u
        // Triangulariza a parte da matriz correspondente
        A(j:m, j:n) = A(j:m, j:n) - 2*u*(u' * A(j:m, j:n))
    end
    // A R é a parte triangular superior da A
    R = triu(A)
endfunction
```

Em seguida, vamos escrever uma versão modificada desse algoritmo, na qual as iterações sobre as colunas de uma matriz m x n acontecem apenas até $m - 1$ colunas se m $\leq$ n. A explicação para isso é esta: a cada iteração, o algoritmo seleciona a coluna seguinte e zera todos os valores abaixo da diagonal da matriz nessa coluna, ou seja, abaixo do elemento $A(j, j)$ da coluna j. No entanto, caso a matriz seja quadrada ou possua mais colunas que linhas, na m-ésima iteração, esse elemento da diagonal será o último elemento daquela coluna, não havendo nada abaixo dele para ser zerado (para iterações seguintes, ele apenas não selecionará nada). Portanto, podemos simplesmente desconsiderar essas últimas iterações.

Feita a explicação, abaixo está a função dessa 2° versão do algoritmo.

```
// Função da versão 2 do algoritmo de Householder para decomposição QR
function [U, R] = qr_House_v2(A)
    // Pegando as dimensões da A
    [m, n] = size(A)
    // Inicializando a matriz U
    k = min(m-1, n)
    U = zeros(m, k)

    // Para cada coluna...
    for j = 1:k
        // Pega o vetor que queremos projetar sobre um eixo
```

```
        x = A(j:m, j)
        // Escolhe a melhor das duas projeções possíveis
        if x(1) > 0 then
            x(1) = x(1) + norm(x)
        else
            x(1) = x(1) - norm(x)
        end
        // Calcula u normalizando x
        u = x/norm(x)
        // Salva u na matriz U
        U(j:m, j) = u
        // Triangulariza a parte da matriz correspondente
        A(j:m, j:n) = A(j:m, j:n) - 2*u*(u' * A(j:m, j:n))
    end
    // A R é a parte triangular superior da A
    R = triu(A)
endfunction
```

Por fim, vamos escrever uma função para construir a matriz Q dessa decomposição com base nos vetores presentes nas colunas da matriz U fornecida pelo algoritmo.

```
// Função para construir a matriz Q
function [Q] = constroi_Q_house(U)
    // Pegando as dimensões da U
    [m, n] = size(U)

    // Inicializando Q como a identidade
    Q = eye(m, m)

    // Para cada vetor em U...
    for j = 1:n
        // Pega a coluna da matriz
        uj = U(:, j)
        // Calcula a matriz de Householder
        Qj = eye(m, m) - 2*uj*uj'
        // Multiplica à direita da Q
        Q = Q * Qj
    end
endfunction
```

Item 4.1)

Vamos testar a precisão desse método em comparação com os anteriores.

```
// Calculando a U e a R do método de Householder
[U1HH1, R1HH1] = qr_House_v1(A1);
[U1HH2, R1HH2] = qr_House_v2(A1);
```

```
// Calculando a matriz Q de Householder
Q1HH1 = constroi_Q_house(U1HH1);
Q1HH2 = constroi_Q_house(U1HH2);

disp("=============== Q*R - A ================")
disp("Gram-Schmidt", norm(Q1*R1 - A1))
disp("Gram-Schmidt modificado", norm(QM1*RM1 - A1))
disp("Householder 1", norm(Q1HH1*R1HH1 - A1))
disp("Householder 2", norm(Q1HH2*R1HH2 - A1))
disp("=============== Q^T*Q - I ===============")
disp("Gram-Schmidt", norm(Q1'*Q1 - eye(2, 2)))
disp("Gram-Schmidt modificado", norm(QM1'*QM1 - eye(2, 2)))
disp("Householder 1", norm(Q1HH1'*Q1HH1 - eye(3, 3)))
disp("Householder 2", norm(Q1HH2'*Q1HH2 - eye(3, 3)))
```

```
"=============== Q*R - A ================"

"Gram-Schmidt"

 1.776D-15

"Gram-Schmidt modificado"

 1.776D-15

"Householder 1"

 1.111D-14

"Householder 2"

 1.111D-14

"=============== Q^T*Q - I ==============="

"Gram-Schmidt"

 2.292D-16

"Gram-Schmidt modificado"

 2.292D-16

"Householder 1"

 1.882D-15

"Householder 2"
```

```
1.882D-15
```

```
[U2HH1, R2HH1] = qr_House_v1(A2);
[U2HH2, R2HH2] = qr_House_v2(A2);

Q2HH1 = constroi_Q_house(U2HH1);
Q2HH2 = constroi_Q_house(U2HH2);

disp("=============== Q*R - A ================")
disp("Gram-Schmidt", norm(Q2*R2 - A2))
disp("Gram-Schmidt modificado", norm(QM2*RM2 - A2))
disp("Householder 1", norm(Q2HH1*R2HH1 - A2))
disp("Householder 2", norm(Q2HH2*R2HH2 - A2))
disp("=============== Q^T*Q - I ===============")
disp("Gram-Schmidt", norm(Q2'*Q2 - eye(3, 3)))
disp("Gram-Schmidt modificado", norm(QM2'*QM2 - eye(3, 3)))
disp("Householder 1", norm(Q2HH1'*Q2HH1 - eye(4, 4)))
disp("Householder 2", norm(Q2HH2'*Q2HH2 - eye(4, 4)))
```

```
"=============== Q*R - A ================"

"Gram-Schmidt"

 1.891D-15

"Gram-Schmidt modificado"

 1.891D-15

"Householder 1"

 2.722D-14

"Householder 2"

 2.722D-14

"=============== Q^T*Q - I ==============="

"Gram-Schmidt"

 2.562D-16

"Gram-Schmidt modificado"

 2.562D-16
```

```
"Householder 1"

 2.144D-15

"Householder 2"

 2.144D-15
```

```
[U3HH1, R3HH1] = qr_House_v1(A3);
[U3HH2, R3HH2] = qr_House_v2(A3);

Q3HH1 = constroi_Q_house(U3HH1);
Q3HH2 = constroi_Q_house(U3HH2);

disp("=============== Q*R - A ================")
disp("Gram-Schmidt", norm(Q3*R3 - A3))
disp("Gram-Schmidt modificado", norm(QM3*RM3 - A3))
disp("Householder 1", norm(Q3HH1*R3HH1 - A3))
disp("Householder 2", norm(Q3HH2*R3HH2 - A3))
disp("=============== Q^T*Q - I ===============")
disp("Gram-Schmidt", norm(Q3'*Q3 - eye(5, 5)))
disp("Gram-Schmidt modificado", norm(QM3'*QM3 - eye(5, 5)))
disp("Householder 1", norm(Q3HH1'*Q3HH1 - eye(5, 5)))
disp("Householder 2", norm(Q3HH2'*Q3HH2 - eye(5, 5)))
```

```
"=============== Q*R - A ================"

"Gram-Schmidt"

 8.882D-16

"Gram-Schmidt modificado"

 1.776D-15

"Householder 1"

 1.308D-14

"Householder 2"

 1.308D-14

"=============== Q^T*Q - I ==============="

"Gram-Schmidt"

 6.279D-16
```

```
"Gram-Schmidt modificado"

 5.514D-16

"Householder 1"

 1.404D-15

"Householder 2"

 1.404D-15
```

Assim, observa-se que o método de decomposição QR de Householder tem uma precisão bem semelhante aos métodos de Gram-Schmidt.

Item 4.2)

1. Matriz mágica 7x7

A seguir, testaremos nossas funções com algumas matrizes especiais, geralmente instáveis. Começaremos com uma matriz mágica 7x7, na qual as somas dos elementos de cada linha, coluna e diagonal é a mesma.

```
M1 = testmatrix('magi', 7)
```

```
 M1  =
   30.   39.   48.   1.    10.   19.   28.
   38.   47.   7.    9.    18.   27.   29.
   46.   6.    8.    17.   26.   35.   37.
   5.    14.   16.   25.   34.   36.   45.
   13.   15.   24.   33.   42.   44.   4.
   21.   23.   32.   41.   43.   3.    12.
   22.   31.   40.   49.   2.    11.   20.
```

```
// Calculando a decomposição QR pelos métodos de Gram-Schmidt
[QM1GS, RM1GS] = qr_GS(M1);
[QM1GSM, RM1GSM] = qr_GSM(M1);

// Calculando a decomposição QR pelo método de Householder
[UM1HH, RM1HH] = qr_House_v2(M1);
QM1HH = constroi_Q_house(UM1HH);

// Calculando a decomposição QR com a função do Scilab
[QM1Sci, RM1Sci] = qr(M1);

disp("============== Q*R - A ===============")
disp("Gram-Schmidt", norm(QM1GS*RM1GS - M1))
disp("Gram-Schmidt modificado", norm(QM1GSM*RM1GSM - M1))
```

```
disp("Householder", norm(QM1HH*RM1HH - M1))
disp("Scilab", norm(QM1Sci*RM1Sci - M1))
disp("=============== Q^T*Q - I ===============")
disp("Gram-Schmidt", norm(QM1GS'*QM1GS - eye(7, 7)))
disp("Gram-Schmidt modificado", norm(QM1GSM'*QM1GSM - eye(7, 7)))
disp("Householder", norm(QM1HH'*QM1HH - eye(7, 7)))
disp("Scilab", norm(QM1Sci'*QM1Sci - eye(7, 7)))
```

```
"=============== Q*R - A ================"

"Gram-Schmidt"

 7.105D-15

"Gram-Schmidt modificado"

 1.029D-14

"Householder"

 5.506D-14

"Scilab"

 9.110D-14

"=============== Q^T*Q - I ==============="

"Gram-Schmidt"

 1.574D-15

"Gram-Schmidt modificado"

 1.017D-15

"Householder"

 6.063D-16

"Scilab"

 6.171D-16
```

Parece que todos os métodos obtiveram resultados bem precisos para ortogonalizar essa matriz mágica.

2. Matriz de Hilbert 7x7

Agora, vamos testar com a inversa de uma matriz de Hilbert, cuja definição é a seguinte para cada elemento (i, j):

$$H_{i,j} = \frac{1}{i + j - 1}$$

```
H = testmatrix('hilb', 7)
```

```
 H  =
   49.     -1176.      8820.      -29400.      48510.     -38808.       12012.
  -1176.    37632.    -317520.     1128960.    -1940400.    1596672.    -504504.
   8820.   -317520.    2857680.   -10584000.   18711000.   -15717240.   5045040.
  -29400.   1128960.  -10584000.   40320000.   -72765000.   62092800.  -20180160.
   48510.  -1940400.   18711000.  -72765000.   1.334D+08  -1.153D+08    37837800.
  -38808.   1596672.  -15717240.   62092800.  -1.153D+08   1.006D+08  -33297264.
   12012.  -504504.    5045040.   -20180160.   37837800.  -33297264.   11099088.
```

```
// Calculando a decomposição QR pelos métodos de Gram-Schmidt
[QHGS, RHGS] = qr_GS(H);
[QHGSM, RHGSM] = qr_GSM(H);

// Calculando a decomposição QR pelo método de Householder
[UHHH, RHHH] = qr_House_v2(H);
QHHH = constroi_Q_house(UHHH);

// Calculando a decomposição QR com a função do Scilab
[QHSci, RHSci] = qr(H);

disp("=============== Q*R - A ================")
disp("Gram-Schmidt", norm(QHGS*RHGS - H))
disp("Gram-Schmidt modificado", norm(QHGSM*RHGSM - H))
disp("Householder", norm(QHHH*RHHH - H))
disp("Scilab", norm(QHSci*RHSci - H))
disp("=============== Q^T*Q - I ===============")
disp("Gram-Schmidt", norm(QHGS'*QHGS - eye(7, 7)))
disp("Gram-Schmidt modificado", norm(QHGSM'*QHGSM - eye(7, 7)))
disp("Householder", norm(QHHH'*QHHH - eye(7, 7)))
disp("Scilab", norm(QHSci'*QHSci - eye(7, 7)))
```

```
  "=============== Q*R - A ================"

  "Gram-Schmidt"

   0.

  "Gram-Schmidt modificado"
```

```
 8.975D-09

"Householder"

 0.0000001

"Scilab"

 3.688D-08

"============== Q^T*Q - I =============="

"Gram-Schmidt"

 0.9852109

"Gram-Schmidt modificado"

 1.933D-09

"Householder"

 1.447D-15

"Scilab"

 3.746D-16
```

Curiosamente, o método de Gram-Schmidt normal obteve precisão máxima no quesito da decomposição da matriz H, mas ficou com um erro muito grande na ortogonalidade da matriz Q. Isso provavelmente ocorre devido a sua abordagem direta e simples na decomposição da matriz e ao fato de ela não preservar exatamente a ortogonalidade entre as colunas de Q, especialmente em matrizes mal-condicionadas como essa.

Com relação ao método modificado, seus resultados ficaram com uma precisão razoável, melhor que a do método normal. Por ser uma melhoria, o algoritmo atualizado geralmente mantém a precisão da decomposição da matriz e visa melhorar a ortogonalidade de Q.

Olhando para o Householder, ele obteve o resultado inverso do Gram-Schmidt: sua precisão na decomposição da H não foi tão boa, mas a ortogonalidade da Q obtida ficou muito alta. Uma possível razão para isso é que esse método preserva a ortogonalidade exata entre as colunas de Q, mesmo com a decomposição não sendo tão precisa.

Por fim, a função do Scilab ficou com resultados bons nos dois quesitos.

3. Matriz mágica 6x6

Por fim, vamos testar com outra matriz mágica, mas agora 6x6.

```
M2 = testmatrix('magi', 6)
```

```
M2  =
  35.   1.    6.    26.   19.   24.
  3.    32.   7.    21.   23.   25.
  31.   9.    2.    22.   27.   20.
  8.    28.   33.   17.   10.   15.
  30.   5.    34.   12.   14.   16.
  4.    36.   29.   13.   18.   11.
```

```
// Calculando a decomposição QR pelos métodos de Gram-Schmidt
[QM2GS, RM2GS] = qr_GS(M2);
[QM2GSM, RM2GSM] = qr_GSM(M2);

// Calculando a decomposição QR pelo método de Householder
[UM2HH, RM2HH] = qr_House_v2(M2);
QM2HH = constroi_Q_house(UM2HH);

// Calculando a decomposição QR com a função do Scilab
[QM2Sci, RM2Sci] = qr(M2);

disp("=============== Q*R - A ================")
disp("Gram-Schmidt", norm(QM2GS*RM2GS - M2))
disp("Gram-Schmidt modificado", norm(QM2GSM*RM2GSM - M2))
disp("Householder", norm(QM2HH*RM2HH - M2))
disp("Scilab", norm(QM2Sci*RM2Sci - M2))
disp("=============== Q^T*Q - I ===============")
disp("Gram-Schmidt", norm(QM2GS'*QM2GS - eye(6, 6)))
disp("Gram-Schmidt modificado", norm(QM2GSM'*QM2GSM - eye(6, 6)))
disp("Householder", norm(QM2HH'*QM2HH - eye(6, 6)))
disp("Scilab", norm(QM2Sci'*QM2Sci - eye(6, 6)))
```

```
 "=============== Q*R - A ================"

 "Gram-Schmidt"

  7.957D-15

 "Gram-Schmidt modificado"

  7.960D-15

 "Householder"

  1.750D-14

 "Scilab"
```

```
 4.101D-14

"=============== Q^T*Q - I ==============="

"Gram-Schmidt"

 0.9965096

"Gram-Schmidt modificado"

 0.9374063

"Householder"

 1.166D-15

"Scilab"

 6.629D-16
```

Diferentemente da matriz mágica 7x7 testada anteriormente, essa 6x6 trouxe alguns problemas para nossos algoritmos. Isso se deve à peculiar característica dessa classe de matrizes de que seus espécimes de ordem ímpar são bem-condicionados, enquanto que os de ordem par são mal-condicionados.

Todos os métodos obtiveram aproximações boas na decomposição da matriz em Q e R.

No entanto, o Gram-Schmidt não obteve sucesso na ortogonalidade da Q, com ambas as versões ficando com erros altos. Uma possível causa para isso é a falta de robustez do método de Gram-Schmidt para garantir a ortogonalidade exata entre as colunas de Q em comparação com os outros métodos, especialmente com matrizes mal-condicionadas, introduzindo erros numéricos que afetam essa ortogonalidade.

Por outro lado, tanto o método de Householder quanto o do Scilab conseguiram resultados muito bons nesse quesito.

Exercício 5

Por último, vamos usar a decomposição QR para encontrar os autovalores de uma matriz simétrica. A função para isso está a seguir.

```
// Função para calcular os autovalores de uma matriz por meio da decomposição QR
function [S] = espectro(A, tol)
    // Inicializando os vetores atual e anterior distantes para garantir a␣
 ↪entrada no loop
    S = [%inf]
    S0 = [0]

    // Enquanto o vetor não convergir...
    while norm(S - S0, 'inf') > tol
```

```
        // Salva o valor anterior
        S0 = S
        // Calcula a decomposição QR da matriz
        [Q, R] = qr_GSM(A)
        // Substitui ela por RQ
        A = R*Q
        // Pega a diagonal dessa matriz como os autovalores
        S = diag(A)
    end
endfunction
```

Vamos testar com algumas matrizes.

```
// Gerando uma matriz simétrica
A4 = 20 * rand(3, 3) - 10;
A4 = round((A4 + A4')/2)

// Vendo quais são seus autovalores
autovalores = spec(A4)

// Calculando esses autovalores com nossa função
S4 = espectro(A4, 10^(-8))
```

```
 A4  =
   5.  -5.  -1.
  -5.   4.   2.
  -1.   2.  -4.
 autovalores  =
  -4.4768213
  -0.3632067
   9.8400280
 S4  =
   9.8400280
  -4.4768213
  -0.3632067
```

```
A5 = 20 * rand(4, 4) - 10;
A5 = round((A5 + A5')/2)

autovalores = spec(A5)

S5 = espectro(A5, 10^(-7))
```

```
 A5  =
  -10.  -2.  -7.  -1.
  -2.    4.   1.   1.
  -7.    1.  -7.  -3.
  -1.    1.  -3.  -5.
```

```
autovalores  =
 -16.316859
 -5.5016052
 -0.9370026
  4.7554671
S5  =
 -16.316859
 -5.5016049
  4.7554669
 -0.9370026
```

```
A6 = 20 * rand(5, 5) - 10;
A6 = round((A6 + A6')/2)

autovalores = spec(A6)

S6 = espectro(A6, 10^(-6))
```

```
A6  =
  9.   1.   5.   2.   5.
  1.  -9.   1.   4.  -1.
  5.   1.   5.   4.  -1.
  2.   4.   4.  -7.  -3.
  5.  -1.  -1.  -3.  -6.
autovalores  =
 -12.599706
 -9.3512946
 -4.0616507
  4.2589108
  13.753740
S6  =
  13.753740
 -12.599706
 -9.3512946
  4.2589013
 -4.0616413
```

Nossa função está funcionando perfeitamente!