AP3 - Algebra Linear Numérica

Gustavo Ramalho*

08 de Maio de 2022

1 Questão 1

Escreva uma função Scilab que implementa o Método da Potência para determinar o autovalor dominante (lambda) de A.

Variáveis de entrada

- A: : matriz real n x n, diagonalizável, com autovalor dominante (lambda).
- x0: vetor, não nulo, a ser utilizado como aproximação inicial do autovetor dominante.
- epsilon: precisão a ser usada no critério de parada.
- M: Número máximo de iterações

Variáveis de saída

- lambda: autovalor dominante de A.
- x1: autovetor unitário (norma infinito) correspondente a lambda.
- k: número de iterações necessárias para a convergência.
- n_erro: norma infinito do erro.

```
function [lambda,x1,k,n_erro] = Metodo_potencia_1(A,x0,epsilon,M)
    k=0;
    [valor, numero]= max(abs(x0));
    x0=x0/x0(numero);
    x1 = A*x0;

    n_erro = epsilon + 1 //Obriga a entrar no loop
    while k<=M && n_erro>=epsilon
        [valor, numero] = max(abs(x1));
        lambda = x1(numero);
        x1=x1/lambda;
        rest = x1-x0;
        n_erro = norm(rest,%inf);
        x0 = x1;
        x1=A*x0;
        k=k+1;
    end
endfunction


```

*gustavoramalho384@gmail.com

```
    function [lambda,x1,k,n_erro] = Metodo_potencia_2(A,x0,epsilon,M)
        k=0;
        x0=x0/norm(x0);
        x1 = A*x0;                          //aproximacao do autovetor dominante
        n_erro = epsilon + 1                //Obriga a entrar no loop
        while k<=M && n_erro >=epsilon
            lambda = x1'*x0;               //Quociente de Rayleigh; x0 eh unitario
            if lambda<0 then               //Mantem x1 com mesmo sentido de x0
                x1=-x1;
            end
            x1=x1/norm(x1);
            rest = x1-x0;
            n_erro = norm(rest);
            x0 = x1;
            x1=A*x0;
            k=k+1;
        end

    endfunction

```

Aqui, criamos as funções propostas no item 1, com base nos pseudo-códigos disponibilizados no enunciado.

2 Questão 2

Escreva uma função Scilab que implementa o Método da Potência Deslocada com Iteração Inversa para determinar o autovalor de A mais próximo de "alfa".

Variáveis de entrada

- A: : matriz real n x n diagonalizável.
- x0: vetor, não nulo, a ser utilizado como aproximação inicial do autovetor dominante.
- alfa: valor do qual se deseja achar o autovalor de A mais próximo;
- epsilon: precisão a ser usada no critério de parada.
- M: Número máximo de iterações

Variáveis de saída

- lambda1: autovalor de A mais próximo de alfa.
- x1: autovetor unitário (norma 2) correspondente a lambda.
- k: número de iterações necessárias para a convergência.
- n_erro: norma 2 do erro.

```
function [lambda1,x1,k,n_erro] = Potencia_deslocada_inversa (A,x0,epsilon,alfa,M)

    k = 0
    x0 = x0/norm(x0);
    n_erro = epsilon + 1;

    [n]=size(A,1);
    alfaI =A - (alfa*eye(n,n));
    while k<=M && n_erro >=epsilon
        x1=Gaussian_Elimination_4(alfaI,x0);
        x1 = x1/norm(x1);
        lambda1 = x1' * A * x1;
```

```
        if x1' * x0 < 0 then
            x1 = -x1
        end
        rest = x1 - x0;
        n_erro = norm(rest);
    end

endfunction

```

Agora, criamos o método da potência deslocada com iteração inversa, também com base nos pseudo-códigos disponibilizados.

3 Questão 3

Teste suas duas primeiras funções para várias matrizes A, com ordens diferentes e também variando as demais variáveis de entrada de cada função. Use matrizes com autovalores reais (por exemplo, matrizes simétricas ou matrizes das quais você saiba os autovalores). Teste a mesma matriz com os dois primeiros algoritmos, comparando os números de iterações necessárias para convergência e os tempos de execução. Teste com uma matriz em que o autovalor dominante é negativo. Alguma coisa deu errada? Se for o caso, corrija o algoritmo (e a função) correspondente.

Primeiro, testei com uma matriz 3x3, que criei manualmente:

Figura 1: console 3.1.1

```
--> A
 A  =

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

--> x0
 x0  =

   1.
   1.
   1.
```

Figura 2: console 3.1.2

```
--> [lambda,x1,k,n_erro] = Metodo_potencia_1(A,x0,10^(-10),50)
 lambda  =

   11.106645
 x1  =

   9.0379377
   6.1818961
   11.106645
 k  =

   17.
 n_erro  =

   9.394D-11

--> [lambda,x1,k,n_erro] = Metodo_potencia_2(A,x0,10^(-10),50)
 lambda  =

   11.106645
 x1  =

   6.4360428
   4.4022153
   7.9091984
 k  =

   17.
 n_erro  =

   5.445D-11
```

Parece que tivemos eficiência semelhante nos dois métodos utilizados. Definimos ϵ como 10^{-10} e o máximo de iterações foi 50. Vale ressaltar que o vetor x1 é o mesmo para os dois métodos, mas multiplicados por escalar diferentes. Verifiquei isso no geogebra, e também dividindo cada componente dos vetores pelo seu respectivo valor no outro, e os resultados foram todos ≈ 1.40.

Agora, vou criar matrizes maiores, e utilizarei comandos do scilab para conseguir criá-las, sem precisar fazer tudo na mão. Começarei com uma 10x10:

Figura 3: console 3.2.1

```
-->   A = ceil(10*rand(10,10));

--> x0 = ceil(10*rand(10,1));

--> A
 A  =

   1.    8.    10.   5.    2.    3.    7.    2.    3.    2.
   6.    6.    6.    3.    1.    6.    8.    10.   10.   9.
   4.    3.    6.    10.   9.    3.    4.    7.    9.    4.
   3.    7.    6.    2.    1.    5.    8.    3.    3.    6.
   6.    1.    6.    5.    9.    10.   6.    6.    9.    6.
   5.    5.    5.    4.    1.    9.    1.    9.    7.    1.
   3.    8.    8.    1.    2.    9.    10.   8.    10.   9.
   1.    9.    8.    6.    5.    3.    3.    2.    1.    6.
   7.    3.    10.   9.    8.    5.    1.    8.    3.    3.
   4.    5.    9.    7.    10.   4.    9.    4.    8.    10.

--> x0
 x0  =

   10.
   4.
   2.
   10.
   8.
   10.
   4.
   6.
   8.
   7.
```

Figura 4: console 3.2.2

```
--> [lambda,x1,k,n_erro] = Metodo_potencia_1(A,x0,10^(-8),100)
 lambda  =

   56.360888
 x1  =

   34.451364
   50.371261
   44.515660
   35.627284
   48.158955
   32.923920
   53.894272
   35.662211
   41.533068
   56.360888
 k  =

   11.
 n_erro  =

   6.702D-09

--> [lambda,x1,k,n_erro] = Metodo_potencia_2(A,x0,10^(-8),100)
 lambda  =

   56.360887
 x1  =

   13.920474
   20.353093
   17.987070
   14.395618
   19.459185
   13.303292
   21.776607
   14.409731
   16.781919
   22.773271
 k  =

   11.
 n_erro  =

   2.504D-09
```

Aqui, defini $\epsilon = 10^{-8}$, e $M = 100$. Assim como na matriz 3x3, as funções foram também igualmente eficientes com a matriz 10x10. Gerei também uma matriz 100x100 aleatória, mas, como $x0$ e A possuem 100 linhas, fica inviável colocar os resultados aqui, mas vale ressaltar que as eficiências foram semelhantes também. Em todas as ocasiões, os vetores $x1$ também foram os mesmos, multiplicados por diferentes escalares.

Agora, vou realizar o teste com uma matriz simétrica 3x3:

Figura 5: console 3.3.1

```
--> A = [1 6 5; 6 2 9; 5 9 1]
 A  =

   1.    6.    5.
   6.    2.    9.
   5.    9.    1.

--> x0 = [1;1;1]
 x0  =

   1.
   1.
   1.
```

Figura 6: console 3.3.2

```
--> [lambda,xl,k,n_erro] = Metodo_potencia_1(A,x0,10^(-6),100)
 lambda  =

   14.891192
 xl  =

   11.378799
   14.891183
   13.743583
 k  =

   19.
 n_erro  =

   0.0000007

--> [lambda,xl,k,n_erro] = Metodo_potencia_2(A,x0,10^(-6),100)
 lambda  =

   14.891186
 xl  =

   7.2909548
   9.5415139
   8.8061863
 k  =

   18.
 n_erro  =

   0.0000007
```

Dessa vez, $\epsilon = 10^{-6}$. Pela primeira vez, tivemos k com valores diferentes, apesar de ser apenas um de diferença. O segundo método foi mais rápido com essa matriz simétrica. Obviamente, obtivemos um autovalor positivo e vetores iguais multiplicados por escalares diferentes.

Por último, testaremos uma matriz com autovalor negativo:

Figura 7: console 3.4.1

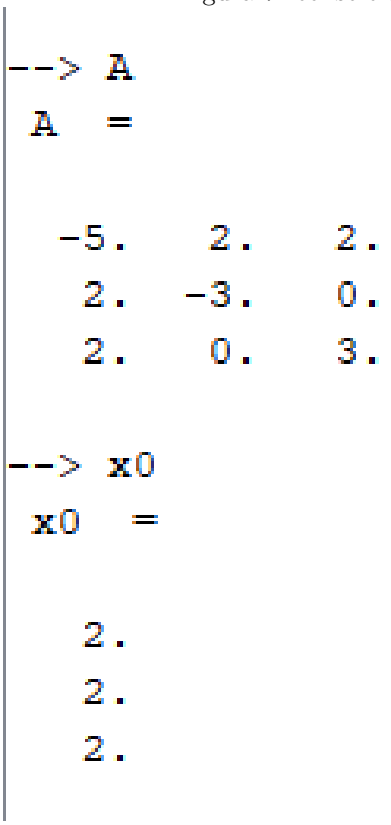


Figura 8: console 3.4.2

```
--> [lambda,k,n_erro] = Metodo_potencia_1(A,x0,10^(-9),70)
 lambda  =

  -6.5467748
 k  =

  -6.5467748
   3.6916777
   1.3715155
 n_erro  =

   39.

--> [lambda,k,n_erro] = Metodo_potencia_2(A,x0,10^(-9),70)
 lambda  =

  -6.5467748
 k  =

   5.6099721
  -3.1634217
  -1.1752602
 n_erro  =

   38.
```

Agora, utilizei $\epsilon = 10^{-9}$ e $M = 70$. Apesar de diversas tentativas para verificar se acharia algum erro com o autovalor λ, parece estar tudo normal com os dois métodos, pois chegamos no valor que queriamos. Um fato curioso é que, apesar dos autovetores ainda serem os mesmos, o escalar que é multiplicado nos dois tem sentido contrário, situação que não tinha acontecido ainda.

4 Questão 4

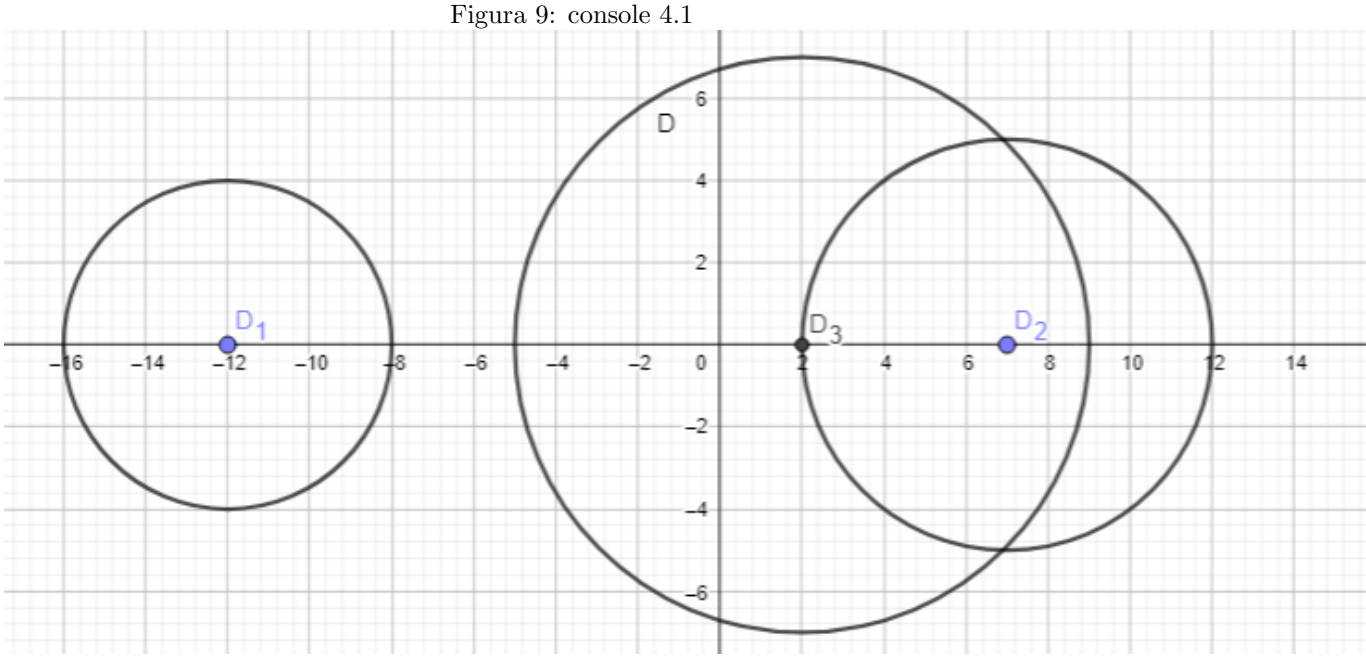
Construa uma matriz simétrica e use os Discos de Gerschgorin para estimar os autovalores. Use essas estimativas e o Método da Potência Deslocada com Iteração Inversa para calcular os autovalores.

Para essa questão, irei utilizar a matriz simétrica $\begin{bmatrix} -12 & 1 & 3 \\ 1 & 7 & 4 \\ 3 & 4 & 2 \end{bmatrix}$.

Primeiro, encontro os centros e raios dos Discos de Gerschgorin:

$$D_1 \Rightarrow C_1 = -12, r_1 = 4$$
$$D_2 \Rightarrow C_2 = 7, r_2 = 5$$
$$D_3 \Rightarrow C_3 = 2, r_3 = 7$$

Figura 9: console 4.1



Agora, com auxílio dos Discos de Gerschgorin, conseguimos saber em que intervalo estão os autovalores da matriz. Temos a certeza de que são 3 autovalores, além de também sabermos que um está entre -16 e 8 (pois o primeiro disco é disjunto) e outros dois autovalores estão entre -5 e 12, correspondente aos dois discos que compartilham espaços.

A seguir, utilizaremos o método da potência deslocada para encontrar os autovalores dessa matriz:

Figura 10: console 4.2

```
--> A
 A  =

  -12.    1.    3.
   1.     7.    4.
   3.     4.    2.

--> x0
 x0  =

   1.
   1.
   1.
```

Figura 11: console 4.3

```
--> [lambda1,x1,k,n_erro] = Potencia_deslocada_inversa (A,x0,10^(-10),-12,50)
 lambda1  =

  -12.617434
 x1  =

   0.9800457
  -0.0094742
  -0.1985465
 k  =

   9.
 n_erro  =

   2.994D-11
```

Figura 12: console 4.4

```
--> [lambda1,x1,k,n_erro] = Potencia_deslocada_inversa (A,x0,10^(-10),7,50)
 lambda1  =

   9.4752380
 x1  =

   0.1101778
   0.8572691
   0.5029418
 k  =

   23.
 n_erro  =

   8.351D-11
```

Figura 13: console 4.5

```
--> [lambda1,x1,k,n_erro] = Potencia_deslocada_inversa (A,x0,10^(-10),2,50)
 lambda1  =

   0.1421961
 x1  =

  -0.1654428
   0.5147814
  -0.8412068
 k  =

   19.
 n_erro  =

   4.883D-11
```

Defini ϵ como 10^{-10} e M em 50 para todas as iterações. Alfa é o centro de cada disco que procuro o autovalor referente, portanto, cada resultado acima diz respeito a um alfa diferente.

Os resultados foram satisfatórios e concluimos que nossa função funcionou. obtivemos $\lambda_1 \approx -12.61$, $\lambda_2 \approx 9.47$ e $\lambda_3 \approx 0.14$. Além disso, todos esses valores estão nos intervalos de tempo que definimos com os Discos de Gerschgorin, assim como esperávamos.

5 Questão 5

Faça outros testes que achar convenientes ou interessantes!!!

Para essa questão, resolvi fazer outros testes com as minhas duas funções usadas no item (1). Agora, vou criar matrizes bem maiores e verificar se a eficiência permanece igual, dado que na questão 3 concluimos que as duas parecem igualmente úteis até com matrizes 100x100, mas ainda não sei se o aumento exponencial desse tamanho irá afetar em algo. Como irei falar de eficiência, é válido ver também o tempo gasto por cada função para chegar ao resultado desejado, ou seja, vou utilizar minha Função Tempo, feita na Atividade Prática 2, para este exercício.

Primeiro, testei com uma matriz aleatória 500x500:

Figura 14: console 5.1.1

```
--> A = ceil(10*rand(500,500));

--> x0 = ceil(10*rand(500,1));

--> [lambda,k,n_erro] = Metodo_potencia_1(A,x0,10^(-10),300)
 lambda  =

   2751.6383
 k  =

   8.
 n_erro  =

   7.405D-12

--> [lambda,k,n_erro] = Metodo_potencia_2(A,x0,10^(-10),300)
 lambda  =

   2751.6383
 k  =

   7.
 n_erro  =

   9.973D-11

-->
```

Figura 15: console 5.1.2

```
--> [tempo]=tempoexecucao(500)
 tempo  =


    0.0018541    0.0017985
```

Aqui, conseguimos ver que k teve um valor baixo, além de também notarmos que o tempo de execução das duas funções é mínimo. O primeiro valor no tempo de execução representa o método 1, enquanto o 2^{o} representa o método 2. Quando tentamos usar essa função na AP2, para análise de Gauss-Seidel, as iterações eram muito mais custosas e esse tempo era bem maior, o que nos diz que agora podemos "arriscar"mais com matrizes bem maiores! Portanto, tentaremos com uma 5000x5000:

Figura 16: console 5.2.1

```
--> A = ceil(10*rand(5000,5000));

--> x0 = ceil(10*rand(5000,1));

--> [lambda,k,n_erro] = Metodo_potencia_1(A,x0,10^(-10),300)
 lambda  =

   27500.796
 k  =

   6.
 n_erro  =

   4.547D-11

--> [lambda,k,n_erro] = Metodo_potencia_2(A,x0,10^(-10),300)
 lambda  =

   27500.796
 k  =

   6.
 n_erro  =

   1.108D-11
```

Figura 17: console 5.2.2

```
--> [tempo]=tempoexecucao(5000)
 tempo  =

   0.2889775    0.2824672
```

Nosso tempo de execução ainda é bastante baixo. Um fato curioso é que parece que o λ assume valor pouco maior que $5 \cdot n$, onde n representa o tamanho da matriz. Isso aconteceu na 500x500, e agora repetiu-se na 5000x5000. Irei conferir se esse padrão permanece, agora com uma matriz 20000x20000:

Figura 18: console 5.3.1

```
--> A = ceil(10*rand(20000,20000));

--> x0 = ceil(10*rand(20000,1));

--> [lambda,k,n_erro] = Metodo_potencia_1(A,x0,10^(-10),300)
 lambda  =

   110003.80
 k  =

   6.
 n_erro  =

   1.526D-12

--> [lambda,k,n_erro] = Metodo_potencia_2(A,x0,10^(-10),300)
 lambda  =

   110003.80
 k  =

   5.
 n_erro  =

   9.628D-11
```

Figura 19: console 5.3.2

```
--> [tempo]=tempoexecucao(20000)
 tempo  =

    4.4741671    4.3386632
```

Consegui ver que o tempo de execução ainda é pequeno e o padrão do λ se manteve, agora em ≈ 110000. Tentei fazer essas mesmas sequências com matrizes ainda maiores, mas quando cheguei em 40000, se tornou muito demorado gerar uma matriz aleatória A, consumindo toda a RAM do meu notebook. No final, parece que as duas funções são bastante eficientes para os nossos cálculos, mesmo com testes de matrizes bem maiores!