ALN - Aula Prática 3

Autor: Daniel de Miranda Almeida

Matrícula: 241708065, Curso: Ciência de Dados

Questão 1

```
function [x] = Ajusta_vetor(x)
    // Retorna o vetor ajustado, dividindo-o pelo seu elemento da maior
    // valor absoluto.

    [max_value, max_index] = max(abs(x))
    if x(max_index) < 0 then
        max_value = -max_value
    end
    x = x/max_value
endfunction
****
function [lambda, x1, k, n_erro] = Metodo_potencia_v1(A, x0, epsilon, M)
    // Calcula o autovalor dominante da matriz de maneira iterativa usando o
método
    // da potência.

    // Inicializando contador de iterações.
    k=0

    // Dividindo vetor pela valor da coordenada de maior módulo para que essa
    //seja igual a 1.
    max_value = max(abs(x0))

    x0 = x0/max_value
    // Codigo adicionado pra corrigir erro.

    // Primeira iteração.
    x1 = A*x0

    // Iniciando erro maior que epsilon para que o codigo entre no loop.
    n_erro = epsilon + 1 // Obriga a entrar no loop

    while k<=M && n_erro >= epsilon then
        // A aproximação do autovalor é o elemento de maior valor absoluto
        // do vetor.
        lambda = max(abs(x1))

        // Redução de escala.
        x1 = x1/lambda

        // Erro da iteração atual.
```

```
        n_erro = norm((x1 - x0), %inf)

        // Indo para a próximo iteração
        x0 = x1
        x1 = A*x0
        k = k+1
    end
endfunction
```

Nessa função estamos, a cada iteração, divindo o vetor pela coordenada de maior valor, de forma a manter essa coordenada sempre igual a 1 e em seguida fazendo a iteração multiplicando x0 por A. É necessário sempre fazer uma redução na escala do vetor para evitar que ele tome valores cada vez maiores que possam causar erros de arredondamento. A cada iteração pegamos o lambda, que é o elemento com maior valor absoluto do vetor e novamente dividimos o vetor por esse valor. Obtemos o erro da iteração calculando a norma infinito da diferença do vetor da iteração atual pelo vetor da iteração anterior. Isso se repete até que o erro esteja abaixo de uma tolerância ou tenhamos ultrapassado o número limite de iterações. Conforme fazemos as iterações, o elemento com maior valor absoluto do vetor se aproxima do autovalor dominante da matriz.

```
function [lambda, x1, k, n_erro] = Metodo_potencia_v2(A, x0, epsilon, M)
    // Encontra o autovalor dominante da matriz pelo método da potência,
utilizando
    // o Quociente de Rayleigh.

    // Inicializando contador de iterações.
    k = 0

    // Redução de escala.
    x0 = x0/norm(x0, 2)

    // Primeira iteração.
    x1 = A*x0

    // Inicializando erro.
    n_erro = epsilon+1

    while k<=M && n_erro >= epsilon then
        // Aproximando lambda com o quociente de Rayleigh.
        lambda = x1'*x0

        // Ajuste do vetor.
        if lambda<0 then
            x1 = -x1
        end

        // Redução de escala.
        x1 = x1/norm(x1, 2)

        // Erro da iteração atual.
        n_erro = norm((x1-x0), 2)
```

```
        // Indo para a próxima iteração.
        x0 = x1
        x1 = A*x0
        k = k+1
    end

endfunction
```

Aqui a diferença é que estamos fazendo a redução da escala do vetor normalizando-o. Além disso, estamos obtendo aproximações do autovalor de A utilizando o quociente de Rayleigh.

Questão 2

```
function [x] = Resolve_sistema_4(A, b)
    // Resolve sistema usando a elimincação de Gauss-Seidel com trocas de
    // linha utilizando o pivô de maior módulo.

    [C, P] = Gaussian_Elimination_4(A)
    [x] = Resolve_com_LU_4(C, b, P)
endfunction

function [lambda, x1, k, n_erro] = Potencia_deslocada_inversa(A, x0, epsilon,
alfa, M)
    // Encontra o autovalor da matriz A mais próximo do alfa dado pelo método da
    // potência inversa deslocada.

    // Inicializando contador.
    k = 0

    // Inicializando erro.
    n_erro = epsilon + 1

    // Normalizando o vetor inicial.
    x0 = x0/norm(x0, 2)

    // Criando matriz de deslocamento.
    [n] = size(A, 1)
    I = eye(n, n)
    A_deslocada = A - alfa*I

    while k<=M && n_erro >= epsilon then
        // Resolvendo sistema para encontrar próxima iteração.
        [x1] = Resolve_sistema_4((A_deslocada), x0)

        // Normalizando vetor.
        x1 = x1/norm(x1, 2)

        // Aproximando lambda.
        lambda = x1' * A * x1
```

```
        if (x1'*x0 < 0) then
            x1 = -x1
        end

        // Calculando erro da última iteração.
        n_erro = norm(x1 - x0, 2)

        // Indo para próxima iteração.
        x0 = x1
        k = k+1
    end
endfunction
```

Nessa função fazemos o deslocamento da matriz A subtraindo dela uma matriz identidade multiplicada por α. Isso faz com que o método convirja para o autovalor mais próximo de α. Além disso, com o uso da potência inversa temos que o método converge para o autovalor mais rapidamente. Nessa função, a aproximação do lambda é feita usando o quociente de Rayleigh da mesma forma que na versão 2 do método da potência.

Questão 3

O primeiro teste é com uma matriz bem simples:

```
A = [1 1;
     2 0]

x0 = [1; 0]

[lambda, x1, k, n_erro] = Metodo_potencia_v1(A, x0, 0.001, 100)

disp("Autovalor dominante")
disp(lambda)
disp("Autovetor dominante")
disp(Ajusta_vetor(x1))
disp("Número de iterações")
disp(k)
disp("Erro da última iteração")
disp(n_erro)

[lambda, x1, k, n_erro] = Metodo_potencia_v2(A, x0, 0.001, 100)

disp("Autovalor dominante")
disp(lambda)
disp("Autovetor dominante")
disp(Ajusta_vetor(x1))
disp("Número de iterações")
disp(k)
disp("Erro da última iteração")
disp(n_erro)
```

```
disp("Autovalores pelo scilab")
disp(spec(A))
```

Resultados:

```
"Autovalor dominante"

 2.

"Autovetor dominante"

 0.9999542
 1.

"Número de iterações"

 14.

"Erro da última iteração"

 0.0003662
```

```
"Autovalor dominante"

 1.9996334

"Autovetor dominante"

 0.9998169
 1.

"Número de iterações"

 12.

"Erro da última iteração"

 0.0005493
```

```
"Autovalores pelo scilab"

 2. + 0.i
-1. + 0.i
```

Comentários

O primeiro método parece ter mais precisão na sua aproximação do autovalor dominante da matriz; em contrapartida, o segundo método gastou menos iterações para chegar em um resultado semelhante.

O segundo teste é com uma matriz mais complexa, e testando para vários valores de epsilon.

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

x0 = [1; 0; 0; 0]

for i=1:1:5
    disp("====================================")

    epsilon = 1/(10**i)
    disp("epsilon:")
    disp(epsilon)

    [lambda, x1, k, n_erro] = Metodo_potencia_v1(A, x0, epsilon, 100)

    disp("Aproximação do autovalor")
    disp(lambda)
    disp("Autovetor dominante")
    disp(x1/max(x1))
    disp("Número de iterações")
```

```
    disp(k)
end

for i=1:1:5
    disp("==================================")

    epsilon = 1/(10**i)
    disp("epsilon:")
    disp(epsilon)

    [lambda, x1, k, n_erro] = Metodo_potencia_v2(A, x0, epsilon, 100)

    disp("Aproximação do autovalor")
    disp(lambda)
    disp("Autovetor dominante")
    disp(x1/max(x1))
    disp("Número de iterações")
    disp(k)
end

disp("Autvalor dominante da matriz")
autovalores = spec(A)
[dom, idx] = max(abs(autovalores))
if autovalores(idx) < 0 then
    dom = -dom
end
disp(dom)
```

```
"epsilon:"
 0.1
"Aproximação do autovalor"
 9.9964944
"Autovetor dominante"
 0.5750610
 1.
 0.0315116
 0.7772534
"Número de iterações"
 8.
```

```
"epsilon:"
 0.01
"Aproximação do autovalor"
 10.045726
"Autovetor dominante"
 0.5750599
 1.
 0.0314521
 0.7772306
"Número de iterações"
 10.
```

```
"epsilon:"
 0.001
"Aproximação do autovalor"
 10.048702
"Autovetor dominante"
 0.5750601
 1.
 0.0314500
 0.7772301
"Número de iterações"
 11.
```

```
"epsilon:"
 0.0001
"Aproximação do autovalor"
 10.048141
"Autovetor dominante"
 0.5750600
 1.
 0.0314503
 0.7772301
"Número de iterações"
 12.
```

```
"epsilon:"
 0.00001
"Aproximação do autovalor"
 10.048222
"Autovetor dominante"
 0.5750601
 1.
 0.0314503
 0.7772302
"Número de iterações"
 14.
```

```
"epsilon:"
 0.1
"Aproximação do autovalor"
 9.5456504
"Autovetor dominante"
 0.5745481
 1.
 0.0257134
 0.7744544
"Número de iterações"
 5.
```

```
"epsilon:"
 0.01
"Aproximação do autovalor"
 10.012851
"Autovetor dominante"
 0.5750469
 1.
 0.0311471
 0.7771044
"Número de iterações"
 7.
```

```
"epsilon:"
 0.001
"Aproximação do autovalor"
 10.046428
"Autovetor dominante"
 0.5750603
 1.
 0.0314390
 0.7772264
"Número de iterações"
 9.
```

```
"epsilon:"
 0.0001
"Aproximação do autovalor"
 10.048579
"Autovetor dominante"
 0.5750599
 1.
 0.0314521
 0.7772306
"Número de iterações"
 10.
```

```
"epsilon:"
 0.00001
"Aproximação do autovalor"
 10.048159
"Autovetor dominante"
 0.5750601
 1.
 0.0314500
 0.7772301
"Número de iterações"
 11.
```

```
"Autvalor dominante da matriz"

 10.048223
```

Comentários

Novamente podemos ver a relação entre precisão e número de iterações entre as duas versões do algorítmo. Agora, variando a tolerância para o erro da aproximação, podemos ver que conforme colocamos uma tolerância menor, nos aproximamos cada vez mais do valor real do autovalor, mas também gastando cada vez mais iterações para tal, como era de se esperar.

O último teste é com uma matriz que tem como autovalor dominante um número negativo.

```
A = [-2.5 3.5 -2.5;
     -1 2 1;
     -3.5 3.5 -1.5]

x0 = [1; 0; 0]

disp("Autovalores da matriz A")
disp(spec(A))
```

```
"Autovalores da matriz A"

-5. + 0.i
 1. + 0.i
 2. + 0.i
```

Quando tentamos aproximar o autovalor dominante dessa matriz com os métodos:

```
[lambda, x1, k, n_erro] = Metodo_potencia_v1(A, x0, 0.001, 100)

disp("Autovalor dominante")
disp(lambda)
disp("Autovetor dominante")
disp(Ajusta_vetor(x1))
disp("Número de iterações")
disp(k)
disp("Erro da última iteração")
disp(n_erro)
disp("Autovalores pelo scilab")
disp(spec(A))

[lambda, x1, k, n_erro] = Metodo_potencia_v2(A, x0, 0.001, 100)

disp("Autovalor dominante")
disp(lambda)
disp("Autovetor dominante")
```

```
disp(Ajusta_vetor(x1))
disp("Número de iterações")
disp(k)
disp("Erro da última iteração")
disp(n_erro)
disp("Autovalores pelo scilab")
disp(spec(A))
```

```
"Autovalor dominante"

 5.

"Autovetor dominante"

 1.
 0.
 1.

"Número de iterações"

 101.

"Erro da última iteração"

 2.
```

```
"Autovalor dominante"

-4.9981676

"Autovetor dominante"

 1.0000000
 0.0000838
 1.

"Número de iterações"

 10.

"Erro da última iteração"

 0.0005181
```

Podemos ver que o método não converge para a versão 1 do algorítmo, e, portanto, não chega a uma aproximação do autovalor dominante da matriz A. Isso acontece porque quando obtemos o elemento do valor com maior valor absoluto não estamos tomando o cuidado de manter o sinal dele, estamos pegando seu valor absoluto sempre. Podemos corrigir o código da seguinte maneira (as linhas com "//" ao final são as linhas adicionadas):

```
function [lambda, x1, k, n_erro] = Metodo_potencia_v1(A, x0, epsilon, M)
    // Calcula o autovalor dominante da matriz de maneira iterativa usando o
método
    // da potência.

    // Inicializando contador de iterações.
    k=0

    // Dividindo vetor pela valor da coordenada de maior módulo para que essa
    //seja igual a 1.
    [max_value, max_index] = max(abs(x0))
    if x0(max_index) < 0 then //
        lambda = -lambda //
    end //

    x0 = x0/max_value
    // Codigo adicionado pra corrigir erro.

    // Primeira iteração.
    x1 = A*x0

    // Iniciando erro maior que epsilon para que o codigo entre no loop.
    n_erro = epsilon + 1 // Obriga a entrar no loop
```

```
    while k<=M && n_erro >= epsilon then
        // A aproximação do autovalor é o elemento de maior valor absoluto
        // do vetor.

        // Apesar de precisarmos do elemento com maior valor absoluto
        // a operação não deve ser feita com o valor absoluto.
        [lambda, max_index] = max(abs(x1)) //
        if x1(max_index) < 0 then //
            lambda = -lambda //
        end //

        // Redução de escala.
        x1 = x1/lambda

        // Erro da iteração atual.
        n_erro = norm((x1 - x0), %inf)

        // Indo para a próximo iteração
        x0 = x1
        x1 = A*x0
        k = k+1
    end
endfunction
```

Agora, rodando novamente:

```
[lambda, x1, k, n_erro] = Metodo_potencia_v1(A, x0, 0.001, 100)

disp("Autovalor dominante")
disp(lambda)
disp("Autovetor dominante")
disp(Ajusta_vetor(x1))
disp("Número de iterações")
disp(k)
disp("Erro da última iteração")
disp(n_erro)
```

```
"Autovalor dominante"

-4.9981665

"Autovetor dominante"

 1.0000000
 0.0000838
 1.

"Número de iterações"

 10.

"Erro da última iteração"

 0.0007327
```

Questão 4

O primeiro teste foi feite com a seguinte matriz:

```
A = [-2.5 3.5 -2.5;
     -1 2 1;
     -3.5 3.5 -1.5]
//

AtA = A'*A
```

Para encontrar os autovalores eu utilizei o método da potência inversa deslocada colocando como alfa os centros dos discos (os elementos da diagonal de A), uma vez que estes são as melhores aproximações para os autovalores da matriz - os autovalores são todos reais e, como o disco define um intervalo onde o autovalor possa estar, a melhor estimativa é o centro desse intervalo. Contudo, se houver uma sobreposição dos discos, pode ocorrer uma situação em que utilizar o centro de dois discos diferentes acaba fazendo o método convergir para o mesmo valor:

```
// Centros dos discos.
centros = diag(AtA)
disp("Centros dos discos: ")
disp(centros)

autovalores = zeros(n)

for i=1:n
    [lambda1, x1, k, n_erro] = Potencia_deslocada_inversa(AtA, x0, 0.001,
centros(i), 100)

    autovalores(i) = lambda1
end

disp("Autovalores encontrados: ")
disp(autovalores)
```

```
disp("Autovalores dados pelo scilab: ")
disp(spec(AtA))
```

```
"Centros dos discos: "

 19.5
 28.5
 9.5

"Autovalores encontrados: "

 3.7562388
 51.688235
 3.7563071

"Autovalores dados pelo scilab: "

 0.5000000
 3.7563139
 53.243686
```

Nesses resultados podemos ver que dois dos centros (19.5 e 9.5) convergiram para o autovalor 3.75, enquanto o centro 28.5 convergiu para o autovalor 51.68. Para que o método seja capaz de encontrar todos os autovalores de uma matriz precisamos que seus discos sejam disjuntos. Podemos ter uma matriz com discos disjuntos se tivermos seus centros bastante distantes entre si com raios não muito grandes. Um exemplo de matriz com discos disjuntos seria:

```
A = [10 1 1/2;
     1 20 2;
     1/2 2 30]
```

Calculando a aproximação dos autovalores dessa matriz:

```
centros = diag(A)
disp("Centros dos discos: ")
disp(centros)

autovalores = zeros(n)

for i=1:n
    [lambda1, x1, k, n_erro] = Potencia_deslocada_inversa(A, x0, 0.001,
centros(i), 100)

    autovalores(i) = lambda1
end

disp("Autovalores encontrados: ")
disp(autovalores)

disp("Autovalores dados pelo scilab: ")
disp(spec(A))
```

```
"Centros dos discos: "

 10.
 20.
 30.

"Autovalores encontrados: "

 9.8963960
 19.695694
 30.407910

"Autovalores dados pelo scilab: "

 9.8963960
 19.695694
 30.407910
```

E aqui conseguimos uma aproximação de todos os autovalores.

Questão 5

Minha primeira ideia foi criar uma matriz que tem como autovalores dominantes dois valor que são um o negativo do outro.

```
A = [-2 -3 -3;
     -4 -2 -4;
     4 3 5]
//

[R, diagvalues] = spec(A)
```

Autovalores

```
 -2. + 0.i
  2. + 0.i
  1. + 0.i
```

Autovetores

```
  0.5773503 + 0.i   -4.314D-16 + 0.i   -0.7071068 + 0.i
  0.5773503 + 0.i   -0.7071068 + 0.i   -1.015D-15 + 0.i
 -0.5773503 + 0.i    0.7071068 + 0.i    0.7071068 + 0.i
```

Tentando calcular uma aproximação para o autovalor dominante usando as duas versões do método da potência:

```
x0 = [0; 1; 1]

[lambda, x1, k, n_erro] = Metodo_potencia_v1(A, x0, 0.001, 100)

disp("Autovalor dominante")
disp(lambda)
disp("Autovetor dominante")
disp(Ajusta_vetor(x1))
disp("Número de iterações")
disp(k)

[lambda, x1, k, n_erro] = Metodo_potencia_v2(A, x0, 0.001, 100)

disp("Autovalor dominante")
disp(lambda)
disp("Autovetor dominante")
disp(Ajusta_vetor(x1))
disp("Número de iterações")
disp(k)
```

```
"Autovalor dominante"
-3.0000000
"Autovetor dominante"
 1.
 0.5000000
-0.5000000
"Número de iterações"
 101.
```

```
"Autovalor dominante"
-3.3333333
"Autovetor dominante"
 1.
 0.5000000
-0.5000000
"Número de iterações"
 101.
```

Se colocarmos o autovetor inical como `x0 = [1; 0; 0]`:

```
"Autovalor dominante"
-4.
"Autovetor dominante"
 1.
 0.
 0.
"Número de iterações"
 101.
```

```
"Autovalor dominante"
-2.0000000
"Autovetor dominante"
 1.
 0.
 1.665D-16
"Número de iterações"
 101.
```

E para `x0 = [1; 1; 1]`:

```
"Autovalor dominante"    "Autovalor dominante"

 3.3333333               -3.4545455

"Autovetor dominante"    "Autovetor dominante"

 1.                       1.
 0.3333333                0.3333333
-0.3333333               -0.3333333

"Número de iterações"    "Número de iterações"

 101.                     101.
```

Podemos ver que nenhum dos dois algorítmos consegue chegar em uma aproximação, porque a matriz não tem um autovalor dominante para o qual o método possa convergir. Mudar os vetores iniciais claramente muda o resultado final, e em um dos casos o algoritmo inclusive se aproximou de fato de um dos autovalores "dominantes".

O outro teste que fiz foi ver como o algoritmo se comportava com uma matriz que tivesse um autovalor muito próximo do autovalor dominante.

```
A = [499/100 399/100 399/100;
     -1/100 499/100 -1/100;
     1/100 -399/100 101/100]
```

Autovalores

```
4.99 + 0.i
5.   + 0.i
1.   + 0.i
```

Autovetores

```
  0.5773503 + 0.i    1.830D-13 + 0.i   -0.7071068 + 0.i
  0.5773503 + 0.i   -0.7071068 + 0.i    2.136D-16 + 0.i
 -0.5773503 + 0.i    0.7071068 + 0.i    0.7071068 + 0.i
```

Usando os dois algoritmos com `x0 = [1; 0; 0]`:

```
"Autovalor dominante"
 4.99
"Autovetor dominante"
 1.
-0.2265487
 0.2265487
"Número de iterações"
 101.
```

```
"Autovalor dominante"
 4.9949310
"Autovetor dominante"
 1.
-0.2265487
 0.2265487
"Número de iterações"
 101.
```

Com `x0 = [0; 1; 1]`:

```
"Autovalor dominante"
 4.99
"Autovetor dominante"
 1.
 0.3867257
-0.3867257
"Número de iterações"
 101.
```

```
"Autovalor dominante"
 4.9879378
"Autovetor dominante"
 1.
 0.4929501
-0.4929371
"Número de iterações"
 6.
```

Com `x0 = [1; 1; 1]`:

```
"Autovalor dominante"
 4.99
"Autovetor dominante"
 1.
 0.1823009
-0.1823009
"Número de iterações"
 101.
```

```
"Autovalor dominante"
 4.9871721
"Autovetor dominante"
 1.
 0.1823009
-0.1823009
"Número de iterações"
 101.
```

Podemos ver que o primeiro método nunca converge de fato e nós três testes retorna a aproximação que não é do autovalor dominante da matriz. A segunda versão do método retornou aproximações que ficam mais próximas do menor autovalor entre os dois maiores. Além disso, se observamos a aproximação do autovetor retornado vemos que ele se parece com uma mistura dos dois autovetores relacionados aos maiores autovalores da matriz.