Aula prática 3
Métodos Iterativos para Cálculos de Autovalores e Autovetores

Will Sena*

Contents

*wllsena@protonmail.com

1) Método da Potência

Escreva uma função Scilab

```
function [lambda, x1, k, n_erro] = Metodo_potencia(A, x0, epsilon, M)
```

que implementa o Método da Potência para determinar o autovalor dominante (*lambda*) de A.

Variáveis de entrada:

- A: matriz real $n \times n$, 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 dominate de A;
- $x1$: autovetor unitário (norma infinito) correspondente a *lambda*;
- k: número de iterações necessárias para a convergência;
- n_erro : norma infinito do erro.

Critério de parada:

- sendo $erro = x1 - x0$, parar quando $n\_erro < epsilon$ ou $k > M$.

Algoritmo (versão 1):

```
function [lambda, x1, k, n_erro] = Metodo_potencia_1(A, x0, epsilon, M)
  k = 1;
  x0 = x0 / max(abs(x0));
  x1 = A * x0; // aproximação do autovetor dominante)

  while k <= M
    lambda = max(abs(x1)); // aproximação autovalor dominante
    x1 = x1 / lambda;

    n_erro = norm(x1 - x0, %inf);
    if n_erro < epsilon
      disp("n_erro < epsilon");
      return;
    end

    x0 = x1;
    x1 = A * x0;
    k = k + 1;
  end

  disp("k > M")
endfunction
```

Algoritmo (versão 2):

```
function [lambda, x1, k, n_erro] = Metodo_potencia_2(A, x0, epsilon, M)
  k = 1;
  x0 = x0 / norm(x0, 2);
  x1 = A * x0;

  while k <= M
    lambda = x0' * x1; // Quociente de Rayleigh; x0 é unitário
    if lambda < 0
      x1 = - x1; // Mantém x1 com mesmo sentido de x0
    end
    x1 = x1 / norm(x1, 2);

    n_erro = norm(x1 - x0, 2);
```

```
    if n_erro < epsilon
      disp("n_erro < epsilon");
      return;
    end

    x0 = x1;
    x1 = A * x0;
    k = k + 1;
  end

  disp("k > M")
endfunction
```

2) Método da Potência Deslocada com Iteração Inversa

Escreva uma função Scilab

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

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

Variáveis de saída:

- *lambda*: 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.

Critério de parada:

- sendo $erro = x1 - x0$, parar quando $n\_erro < epsilon$.

Algoritmo:

```
function [lambda, x1, k, n_erro] = Potencia_deslocada_inversa(A, x0, epsilon, alfa, M)
  n  = size(A, 1);
  k  = 1;
  x0 = x0 / norm(x0, 2);

  while k <= M
    x1     = Gaussian_Elimination(A - alfa * eye(n, n), x0);
    x1     = x1 / norm(x1, 2);
    lambda = x1' * A * x1; // Quociente de Rayleigh; x1 é unitário

    if lambda < 0
      x1 = - x1; // Mantém x1 com mesmo sentido de x0
    end

    n_erro = norm(x1 - x0, 2);
    if n_erro < epsilon
      disp("n_erro < epsilon");
      return;
    end

    x0 = x1;
```

```
22      k  = k + 1;
23    end
24
25    disp("k > M")
26  endfunction
```

3) Método da Potência Deslocada com Iteração de Rayleigh

Escreva uma função Scilab

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

que implementa o Método da Potência Deslocada com Iteração de Rayleigh para determinar o autovalor de A mais próximo de "$alfa$".

Variáveis de entrada:

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

Variáveis de saída:

- *lambda*: 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.

Critério de parada:

- sendo $erro = x1 - x0$, parar quando $n\_erro < epsilon$.

Algoritmo:

```
function [lambda, x1, k, n_erro] = Potencia_deslocada_Rayleigh(A, x0, epsilon, alfa, M)
  n      = size(A, 1);
  k      = 1;
  lambda = alfa;
  x0     = x0 / norm(x0, 2);

  while k <= M
    x1     = Gaussian_Elimination(A - lambda * eye(n, n), x0);
    x1     = x1 / norm(x1, 2);
    lambda = x1' * A * x1;

    n_erro = norm(x1 - x0, 2);
    if n_erro < epsilon
      disp("n_erro < epsilon");
      return;
    end

    x0 = x1;
    k  = k + 1;
  end

  disp("k > M")
endfunction
```

4)

Teste suas 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.

Funçãozinha para facilitar a vida gerando as entradas aleatórias e executando os métodos:

```
function [] = test_metodos_potencia()
  n       = ceil(1+5*rand(1));
  A       = rand(n, n);
  if rand(1) > 0.5
    A = triu(A); A = A + A'; // Matrix simétrica
  end
  x0      = rand(n,1);
  epsilon = 10^(-10*rand(1));
  M       = ceil(100*rand(1));
  mprintf("A:");
  disp(A);
  mprintf("x0:");
  disp(x0);
  mprintf("epsilon:");
  disp(epsilon);
  mprintf("M:");
  disp(M);

  mprintf("\nVersão 1:");
  tic();
  [lambda, x1, k, n_erro] = Metodo_potencia_1(A, x0, epsilon, M);
  tempo = toc();
  mprintf("\n");
  mprintf("lambda (estimado):");
  disp(lambda);
  mprintf("x1:");
  disp(x1);
```

```
28    mprintf("k:");
29    disp(k);
30    mprintf("n_erro");
31    disp(n_erro)
32    mprintf("tempo:");
33    disp(tempo);
34    mprintf("lambda (real):");
35    disp(spec(A)(1,1));
36
37    mprintf("\nVersão 2:");
38    tic();
39    [lambda, x1, k, n_erro] = Metodo_potencia_2(A, x0, epsilon, M);
40    tempo = toc();
41    mprintf("\n");
42    mprintf("lambda (estimado):");
43    disp(lambda);
44    mprintf("x1:");
45    disp(x1);
46    mprintf("k:");
47    disp(k);
48    mprintf("n_erro");
49    disp(n_erro)
50    mprintf("tempo:");
51    disp(tempo);
52    mprintf("lambda (spec(A)(1,1)):");
53    disp(spec(A)(1,1));
54  endfunction
```

Executando-a algumas vezes.

```
--> test_metodos_potencia()
A:
0.4329265   0.3076091   0.312642
0.3076091   1.8659232   0.3616361
0.312642    0.3616361   0.5844533
x0:
0.4826472
0.3321719
0.5935095
```

```
epsilon:
7.024D-51
M:
437.

Versão 1:
"n_erro < epsilon"

lambda (estimado):
2.0505599
x1:
0.2480566
1.
0.2995614
k:
34.
n_erro
0.
tempo:
0.001189
lambda (real):
0.1867501

Versão 2:
"n_erro < epsilon"

lambda (estimado):
2.0505599
x1:
0.2311864
0.9319907
0.2791885
k:
34.
n_erro
0.
tempo:
0.002114
lambda (spec(A)(1,1)):
0.1867501
```

```
--> test_metodos_potencia()
A:
1.265149    0.0437334   0.4148104
0.0437334   0.9637018   0.2806498
0.4148104   0.2806498   0.2560117
x0:
0.2119030
0.1121355
0.6856896
epsilon:
4.873D-16
M:
698.

Versão 1:
"n_erro < epsilon"

lambda (estimado):
1.4552007
x1:
1.
0.3306879
0.4233010
k:
87.
n_erro
3.886D-16
tempo:
0.0048580
lambda (real):
0.0390981

Versão 2:
"n_erro < epsilon"

lambda (estimado):
1.4552007
x1:
0.8809502
```

```
0.2913196
0.3729071
k:
87.
n_erro
4.336D-16
tempo:
0.004308
lambda (spec(A)(1,1)):
0.0390981

--> test_metodos_potencia()
A:
column 1 to 5
0.4062025   0.5896177   0.9222899   0.4993494   0.6274093
0.4094825   0.685398    0.9488184   0.2638578   0.7608433
0.8784126   0.8906225   0.3435337   0.5253563   0.0485566
0.113836    0.5042213   0.3760119   0.537623    0.672395
0.1998338   0.3493615   0.7340941   0.1199926   0.2017173
0.5618661   0.3873779   0.2615761   0.2256303   0.3911574
column 6
0.8300317
0.587872
0.4829179
0.2232865
0.8400886
0.1205996
x0:
0.8607515
0.8494102
0.5257061
0.9931210
0.6488563
0.9923191
epsilon:
0.0000099
M:
749.

Versão 1:
```

```
"n_erro < epsilon"

lambda (estimado):
2.9671013
x1:
1.
0.9755320
0.9011577
0.6128758
0.6277338
0.5478000
k:
8.
n_erro
0.0000024
tempo:
0.0008
lambda (real):
2.9670987

Versão 2:
"n_erro < epsilon"

lambda (estimado):
2.9670961
x1:
0.5107425
0.4982466
0.4602595
0.3130226
0.3206094
0.2797860
k:
7.
n_erro
0.0000097
tempo:
0.000465
lambda (spec(A)(1,1)):
2.9670987
```

```
--> test_metodos_potencia()
A:
0.6084526   0.9262344   0.0568928   0.2677766
0.8544211   0.5667211   0.5595937   0.5465335
0.0642647   0.5711639   0.124934    0.9885408
0.8279083   0.816011    0.7279222   0.7395657
x0:
0.5900573
0.3096467
0.2552206
0.6251879
epsilon:
2.666D-12
M:
612.

Versão 1:
"n_erro < epsilon"

lambda (estimado):
2.3609231
x1:
0.5938823
0.7938850
0.6619642
1.
k:
21.
n_erro
1.158D-12
tempo:
0.0013700
lambda (real):
2.3609231

Versão 2:
"n_erro < epsilon"

lambda (estimado):
```

```
2.3609231
x1:
0.3816716
0.5102078
0.4254260
0.6426722
k:
21.
n_erro
7.679D-13
tempo:
0.0017050
lambda (spec(A)(1,1)):
2.3609231

--> test_metodos_potencia()
A:
0.6640191   0.5064435   0.3454984   0.0881335   0.4337721
0.5064435   0.8472204   0.7064868   0.4498763   0.9677053
0.3454984   0.7064868   1.0422945   0.7227253   0.5068534
0.0881335   0.4498763   0.7227253   1.7953593   0.5232976
0.4337721   0.9677053   0.5068534   0.5232976   1.1193895
x0:
0.4681760
0.7794547
0.7901072
0.9808542
0.8187066
epsilon:
2.699D-43
M:
247.

Versão 1:
"n_erro < epsilon"

lambda (estimado):
3.3253422
x1:
0.4591523
```

```
0.8749192
0.8584982
1.
0.9085698
k:
38.
n_erro
0.
tempo:
0.001671
lambda (real):
-0.0659585

Versão 2:
"k > M"

lambda (estimado):
3.3253422
x1:
0.8116405
1.5465890
1.5175616
1.7676934
1.6060728
k:
248.
n_erro
1.272D-16
tempo:
0.009714
lambda (spec(A)(1,1)):
-0.0659585
```

Os números de iterações entre as duas versões diferentes são, em média, similares, mas a versão 2 possui menores tempos de execução.

5)

Construa uma matriz simétrica e use os Discos de Gerschgorin para estimar os autovalores. Use essas estimativas e os Métodos da Potência Deslocada com Iteração Inversa e com Iteração de Rayleigh. Compare o número de iterações e o tempo de execução.

```
function [d, r] = Discos_de_Gerschgorin(A)
  n = size(A, 1);
  d = diag(A);
  r = zeros(n,1);

  for i = 1:n
    for j = 1:n
      if i ~= j
        r(i, 1) = r(i, 1) + abs(A(i, j));
      end
    end
  end
endfunction
```

Função para gerar entradas aleatórias e executando os métodos:

```
function [] = test_metodos_potencia_deslocada()
  n       = ceil(1+5*rand(1));
  A       = rand(n, n);
  A = triu(A); A = A + A';
  x0      = rand(n,1);
  epsilon = 10^(-100*rand(1));
  M       = ceil(1000*rand(1));
  alfas   = Discos_de_Gerschgorin(A);
  mprintf("A:");
  disp(A);
  mprintf("x0:");
  disp(x0);
  mprintf("epsilon:");
  disp(epsilon);
  mprintf("M:");
  disp(M);
```

```
17  mprintf("Alfas:");
18  disp(alfas);
19
20  mprintf("\nVersão 1:");
21  tic();
22  lambdas = zeros(n, 1);
23  ks      = 0;
24  for i = 1:n
25    [lambda, x1, k] = Potencia_deslocada_inversa(A, x0, epsilon, alfas(i), M);
26    lambdas(i) = lambda;
27    ks         = ks + k;
28  end
29  tempo = toc();
30  mprintf("\n");
31  mprintf("lambdas (estimados):");
32  disp(gsort(lambdas,'g','i'));
33  mprintf("ks:");
34  disp(ks);
35  mprintf("tempo:");
36  disp(tempo);
37  mprintf("lambdas (spec(A)):");
38  disp(spec(A));
39
40  mprintf("\nVersão 2:");
41  tic();
42  lambdas = zeros(n, 1);
43  ks      = 0;
44  for i = 1:n
45    [lambda, x1, k] = Potencia_deslocada_Rayleigh(A, x0, epsilon, alfas(i), M);
46    lambdas(i) = lambda;
47    ks         = ks + k;
48  end
49  tempo = toc();
50  mprintf("\n");
51  mprintf("lambdas (estimados):");
52  disp(gsort(lambdas,'g','i'));
53  mprintf("ks:");
54  disp(ks);
55  mprintf("tempo:");
56  disp(tempo);
```

```
57    mprintf("lambdas (spec(A)):");
58    disp(spec(A));
59  endfunction
```

Executando-a algumas vezes.

```
--> test_metodos_potencia_deslocada()
A:
0.379875    0.7400147   0.439646    0.2029444   0.3792142
0.7400147   1.232532    0.6540737   0.7844274   0.7668716
0.439646    0.6540737   1.1756213   0.2637536   0.6006621
0.2029444   0.7844274   0.2637536   0.8766553   0.7856736
0.3792142   0.7668716   0.6006621   0.7856736   1.4774231
x0:
0.5544260
0.9929150
0.9757428
0.3709622
0.3032238
epsilon:
6.380D-96
M:
713.
Alfas:
0.3798750
1.2325320
1.1756213
0.8766553
1.4774231

Versão 1:
"k > M"

"k > M"

"k > M"

"k > M"
```

```
"k > M"

lambdas (estimados):
0.2639280
0.9030035
0.9030035
0.9030035
0.9030035
ks:
3570.
tempo:
1.920171
lambdas (spec(A)):
-0.1257556
0.2639280
0.6546368
0.9030035
3.4462939

Versão 2:
"k > M"

"k > M"

"k > M"

"k > M"

"k > M"

lambdas (estimados):
0.9030035
0.9030035
0.9030035
0.9030035
0.9030035
ks:
3570.
tempo:
1.902105
```

```
lambdas (spec(A)):
-0.1257556
0.2639280
0.6546368
0.9030035
3.4462939

--> test_metodos_potencia_deslocada()
A:
1.0018326   0.4808947
0.4808947   0.6607392
x0:
0.6304475
0.2117191
epsilon:
1.380D-45
M:
592.
Alfas:
1.0018326
0.6607392

Versão 1:
"n_erro < epsilon"

"k > M"

lambdas (estimados):
0.3210449
1.3415269
ks:
646.
tempo:
0.1130870
lambdas (spec(A)):
0.3210449
1.3415269

Versão 2:
"n_erro < epsilon"
```

```
"n_erro < epsilon"

lambdas (estimados):
1.3415269
1.3415269
ks:
11.
tempo:
0.006038
lambdas (spec(A)):
0.3210449
1.3415269

--> test_metodos_potencia_deslocada()
A:
0.1478592   0.892869    0.4088      0.3157836   0.3292049
0.892869    0.9235838   0.0636221   0.3785823   0.4719233
0.4088      0.0636221   0.1314787   0.4619523   0.335377
0.3157836   0.3785823   0.4619523   1.257474    0.555307
0.3292049   0.4719233   0.335377    0.555307    0.2392162
x0:
0.7614000
0.4790988
0.2816969
0.2380098
0.3294205
epsilon:
8.565D-24
M:
214.
Alfas:
0.1478592
0.9235838
0.1314787
1.2574740
0.2392162

Versão 1:
"k > M"
```

```
"k > M"

"k > M"

"k > M"

"k > M"

lambdas (estimados):
0.1161382
0.1161382
0.1161382
0.9247962
0.9247962
ks:
1075.
tempo:
0.5941700
lambdas (spec(A)):
-0.5951430
-0.1223589
0.1161382
0.9247962
2.3761793

Versão 2:
"k > M"

"k > M"

"k > M"

"k > M"

"k > M"

lambdas (estimados):
0.1161382
0.1161382
```

```
0.1161382
0.9247962
0.9247962
ks:
1075.
tempo:
0.61273
lambdas (spec(A)):
-0.5951430
-0.1223589
0.1161382
0.9247962
2.3761793

--> test_metodos_potencia_deslocada()
A:
0.6190742   0.0204748   0.2023378   0.4071123
0.0204748   1.788273    0.1304691   0.6691938
0.2023378   0.1304691   1.7147906   0.2042602
0.4071123   0.6691938   0.2042602   1.6620863
x0:
0.0122163
0.4884462
0.9549877
0.0587431
epsilon:
2.602D-83
M:
299.
Alfas:
0.6190742
1.7882730
1.7147906
1.6620863

Versão 1:
"k > M"

"k > M"
```

```
"k > M"

"n_erro < epsilon"

lambdas (estimados):
0.4289321
1.6654991
1.6654991
1.6654991
ks:
909.
tempo:
0.373029
lambdas (spec(A)):
0.4289321
1.1570066
1.6654991
2.5327863

Versão 2:
"k > M"

"n_erro < epsilon"

"n_erro < epsilon"

"k > M"

lambdas (estimados):
1.1570066
1.6654991
1.6654991
1.6654991
ks:
611.
tempo:
0.254357
lambdas (spec(A)):
0.4289321
1.1570066
```

```
1.6654991
2.5327863

--> test_metodos_potencia_deslocada()
A:
1.1692185   0.051723
0.051723    1.191725
x0:
0.3833705
0.4900220
epsilon:
1.871D-53
M:
69.
Alfas:
1.1692185
1.1917250

Versão 1:
"k > M"

"k > M"

lambdas (estimados):
1.1275387
1.2334047
ks:
140.
tempo:
0.0309720
lambdas (spec(A)):
1.1275387
1.2334047

Versão 2:
"k > M"

"k > M"

lambdas (estimados):
```

```
1.2334047
1.2334047
ks:
140.
tempo:
0.0288270
lambdas (spec(A)):
1.1275387
1.2334047
```

O Método da Potência Deslocada com Iteração de Rayleigh possui, em média, menores iterações e tempos de execução

6)

Faça outros testes que achar convenientes ou interessantes!!!

No momento nenhum outro teste é conveniente, só quero dormir.