Fundação Getúlio Vargas
Escola de Matemática Aplicada
Álgebra Linear Numérica

Métodos Iterativos para Cálculo de Autovalores e Autovetores

Aluno:
- Jeann da Rocha Silva
Professor:
- Antonio Carlos Saraiva Branco

Abril
2024

1) MÉTODO DA POTÊNCIA - versão 1 e 2
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.

No arquivo em anexo, encontram-se as duas versões para funções que implementam o Método da Potência, sendo a primeira (*Metodo_potencia_1*) com o método de **mudança de escala** e a segunda (*Metodo_potencia_2*) com o método do **quociente de Rayleigh**. Na prática, ambas as funções começam com $k = 0$, indicando o início das iterações, tomam x_0 dividido por sua norma do máximo (infinito) (no caso da versão 1) e x_0 dividido por sua norma euclidiana (2) (vetor normalizado em $\mathbb{R}^n$) (no caso da versão 2). Tomamos uma aproximação inicial $x_1 = Ax_0$ para o autovetor dominante e criamos um erro $\varepsilon > 0$ para o qual desejamos que o resultado final se mantenha na faixa $[0, \varepsilon]$. Com isso, iniciamos as iterações, tomando λ_k como a coordenada de maior absoluto de x_k (no caso da versão 1) e λ_k como o produto interno $\langle x_k, x_{k-1} \rangle$, verificando (se necessário), se $\lambda_k < 0$ e invertendo o sentido do vetor x_k caso isto ocorra (no caso da versão 2). Para o método da **mudança de escala**, isto significa que $\lambda = \lim \lambda_k$ é aproximadamente a coordenada de maior módulo de x_k (pois, sendo $|x_{k-1}|_\infty = 1$ no decorrer das iterações, temos $x_k = Ax_{k-1} \approx \lambda x_{k-1} \Leftrightarrow \lambda \approx \dfrac{|x_k|_\infty}{|x_{k-1}|_\infty} = |x_k|_\infty$). Para o método do **quociente de Rayleigh**, isto significa que $\lambda = \dfrac{\lambda\langle x, x\rangle}{\langle x, x\rangle} \approx \dfrac{\lambda\langle x_k, x_{k-1}\rangle}{|x_{k-1}|} = \langle \lambda x_k, x_{k-1}\rangle = \langle Ax_{k-1}, x_{k-1}\rangle$. Repetimos até que as iterações excedam o limite permitido ou a faixa de erro $[0, \varepsilon]$ seja satisfeita e esta faixa é calculada tomando a norma (que for conveniente para o método) da diferença entre os dois últimos vetores consecutivos. Por fim, retornamos também o tempo de execução de cada método a fim de comparar a eficácia de cada um. Para mais detalhes, consulte o arquivo *Metodo_potencia.sci* em anexo.

2) MÉTODO DA POTÊNCIA DESLOCADA com ITERAÇÃO INVERSA

Escreva uma função Scilab

```
function [lambda1,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".

No arquivo em anexo, encontra-se a função que implementa o Método da Potência Inversa Deslocada, denotada por *Potencia_deslocada_inversa*. A forma desta função é similar a do **coeficiente de Rayleigh**, mas calculamos x_k a partir de $x_k = (A - \alpha I)^{-1} x_{k-1}$, resolvendo a Eliminação Gaussiana $(A - \alpha I)x_k = x_{k-1}$ e tomamos λ do mesmo modo. Por fim, ao final do loop, tomamos $\lambda_1 = \alpha + \frac{1}{\lambda}$, pois se λ é autovalor de $(A - \alpha I)^{-1}$, então $\frac{1}{\lambda}$ é autovalor de $A - \alpha I$, donde $\frac{1}{\lambda} + \alpha$ é autovalor de A e este é o autovalor de A mais próximo de α. Também retornamos como variável o tempo de execução da função. Há também no arquivo a função de Eiminação Gaussiana utiizada na aula prática 1. Para mais detalhes, consulte o arquivo *Potencia_deslocada_inversa.sci* em anexo.

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.

Vamos considerar $\varepsilon = 10^{-10}$ e $M = 500$ para os testes abaixos. As matrizes A e os respectivos valores x_0 estão especificados em cada teste. Para as matrizes a seguir, sempre teremos um autovalor real dominante. O caso de autovalores repetidos e/ou complexos será tratado no item 5.

Teste 1 - $A = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}, x_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ **(VERSÃO 1)**

```
--> [lambda, x1, k, n_erro, time] = Metodo_potencia_1(A, x0, epsilon, M)
 lambda  =

   3.4494897
 x1  =

   2.8164966
   3.4494897
 k  =

   29.
 n_erro  =

   6.641D-11
 time  =

   0.000408
```

Para verificar se o resultado fornecido é correto, podemos olhar para o raio espectral da matriz e confirmar que, de fato, trata-se do autovalor dominante de A. Analogamente, com λ e x_0, podemos efetuar Ax_0 e λx_0 e verificar que, de fato, os resultados coincidem para confirmar que se trata de um autovetor relativo à λ.

```
--> spec(A)
 ans  =

   3.4494897 + 0.i
  -1.4494897 + 0.i

--> A*x1
 ans  =

   9.7154761
   11.898979

--> lambda*x1
 ans  =

   9.7154761
   11.898979
```

Teste 1 - $A = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}, x_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ **(VERSÃO 2)**

```
--> [lambda, x1, k, n_erro, time] = Metodo_potencia_2(A, x0, epsilon, M)
 lambda  =

   3.4494897
 x1  =

   2.1816489
   2.6719633
 k  =

   28.
 n_erro  =

   9.482D-11
 time  =

   0.0004318
```

```
--> spec(A)
 ans  =

   3.4494897 + 0.i
  -1.4494897 + 0.i

--> A*x1
 ans  =

   7.5255754
   9.2169099

--> lambda*x1
 ans  =

   7.5255754
   9.2169099
```

Teste 2 - $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 4 \end{bmatrix}, x_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ **(VERSÃO 1)**

```
--> [lambda, x1, k, n_erro, time] = Metodo_potencia_1(A, x0, epsilon, M)
 lambda  =

   6.5160459
 x1  =

   2.0482406
   4.7821432
   6.5160459
 k  =

   13.
 n_erro  =

   1.124D-11
 time  =

   0.0004044
```

```
--> spec(A)
 ans  =

  -0.2184795
   0.7024336
   6.5160459

--> A*x1
 ans  =

   13.346430
   31.160665
   42.458854

--> lambda*x1
 ans  =

   13.346430
   31.160665
   42.458854
```

Teste 2 - $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 4 \end{bmatrix}, x_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ **(VERSÃO 2)**

```
--> [lambda, x1, k, n_erro, time] = Metodo_potencia_2(A, x0, epsilon, M)
 lambda  =

   6.5160459
 x1  =

   1.6006661
   3.7371656
   5.0921818
 k  =

   12.
 n_erro  =

   7.792D-11
 time  =

   0.0001726
```

```
--> spec(A)
 ans  =

  -0.2184795
   0.7024336
   6.5160459

--> A*x1
 ans  =

   10.430013
   24.351543
   33.180890

--> lambda*x1
 ans  =

   10.430013
   24.351543
   33.180890
```

Os dois testes acima revelam que, **aparentemente**, os algoritmos funcionam e os números de iterações necessárias são quase os mesmos (nos casos acima, diferem apenas por 1 unidade). Vamos efetuar os mesmos testes, mas para vetores x_0 diferentes a fim de verificar o número de iterações e os tempos de execução.

Teste 1' - $A = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}, x_0 = \begin{bmatrix} 42 \\ -13 \end{bmatrix}$ **(VERSÃO 1)**

```
 k  =

   30.
 time  =

   0.0004414
```

Teste 1' - $A = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}, x_0 = \begin{bmatrix} 42 \\ -13 \end{bmatrix}$ **(VERSÃO 2)**

```
 k  =

   29.
 time  =

   0.0004464
```

Teste 2' - $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 4 \end{bmatrix}, x_0 = \begin{bmatrix} 111 \\ -5 \\ 2.7 \end{bmatrix}$ **(VERSÃO 1)**

```
k  =

  13.
time  =

  0.0002282
```

Teste 2' - $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 4 \end{bmatrix}, x_0 = \begin{bmatrix} 111 \\ -5 \\ 2.7 \end{bmatrix}$ **(VERSÃO 2)**

```
k  =

  12.
time  =

  0.0002464
```

Como se observa, o número de iterações permanecem os mesmos (diferindo por uma unidade relativamente ao método) e os tempos de execução são aproximadamente iguais. Logo, ambos os métodos **aparentemente** parecem ser equivalentes (no sentido de resultado e número de iterações).

Vamos agora considerar o caso de uma matriz com autovalor dominante negativo.

Teste 3 - $A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -1 \\ 0 & 0 & 0.1 \end{bmatrix}, x_0 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ **(VERSÃO 1 e 2)**

Para esta matriz, os autovalores são $1, -2$ e 0.1, Pois a matriz é triangular superior com estes termos na diagonal, donde o autovalor dominante é -2. Aplicando o algoritmo, temos

```
--> [lambda, x1, k, n_erro, time] = Metodo_potencia_1(A, x0, epsilon, M)
 lambda  =

   2.
 x1  =

  -1.3333333
   2.
   0.
 k  =

   501.
 n_erro  =

   2.
 time  =

   0.0052383

--> [lambda, x1, k, n_erro, time] = Metodo_potencia_2(A, x0, epsilon, M)
 lambda  =

  -2.0000000
 x1  =

  -1.1094004
   1.6641006
   0.
 k  =

   36.
 n_erro  =

   5.037D-11
 time  =

   0.0005161
```

Como se observa, o autovalor dominante (em módulo) é obtido pelo método do **quociente de Rayleigh**, mas o método da **mudança de escala** retorna o autovalor em sua forma positiva (ou seja, seu módulo). Isso se deve, pois tomamos o módulo da maior coordenada para a aproximação do autovalor. Ademais, isto implica também o número de iterações ter sido excedido, pois o sinal sempre vai estar errado (positivo) neste caso. Para corrigir isto, podemos ir na função e alterar a variável λ para que ela tenha o mesmo sinal da coordenada de maior módulo. Com isto, segue em anexo o arquivo *Metodo_potencia_1_corrigido.sci*, com a versão da função *Metodo_potencia_1* do arquivo *Metodo_potencia.sci* corrigida neste aspecto. Realizando os testes novamente, temos

```
--> [lambda,x1,k,n_erro,time]=Metodo_potencia_1_corrigido(A,x0,epsilon,M)
 lambda  =

   2.
 x1  =

   1.3333333
  -2.
   0.
 k  =

   36.
 n_erro  =

   7.276D-11
 time  =

   0.0013426
```

Para fins de sequência cronológica, vamos manter o método de **mudanca de escala** antigo nos arquivos em anexo. Mas, a partir de agora, só utilizaremos o método corrigido.

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. Alguma coisa deu errada? Comente!

Segue em anexo o arquivo *Gerschgorin.sci* que contém as funções *Gerschgorin_real* que retorna os discos de Gerschgorin no caso de autovalores reais e *Find_eigenvalue* que encontra os autovalores mais próximos dos centros encontrados, utilizando a função *Potencia_deslocada_inversa*.

Teste 1 - $A = \begin{bmatrix} 1 & 0.1 & 0.9 \\ 0.1 & 4 & 0.9 \\ 0.1 & 0.9 & -2 \end{bmatrix}, x_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

```
--> [C]=Gerschgorin_Real(A)
 C  =

   1.    1.
   4.    1.
  -2.    1.

--> Find_eigenvalues(A,x0,epsilon,C(:,1),M)
 ans  =

   1.0151366
   4.1404197
  -2.1555562
```

Para verificar que, de fato, estes são os autovalores, podemos consultar o espectro da matriz e visualizar como se distribuem os discos de Gerschgorin no plano.

```
--> spec(A)
 ans  =

   1.0151366 + 0.i
   4.1404197 + 0.i
  -2.1555562 + 0.i
```

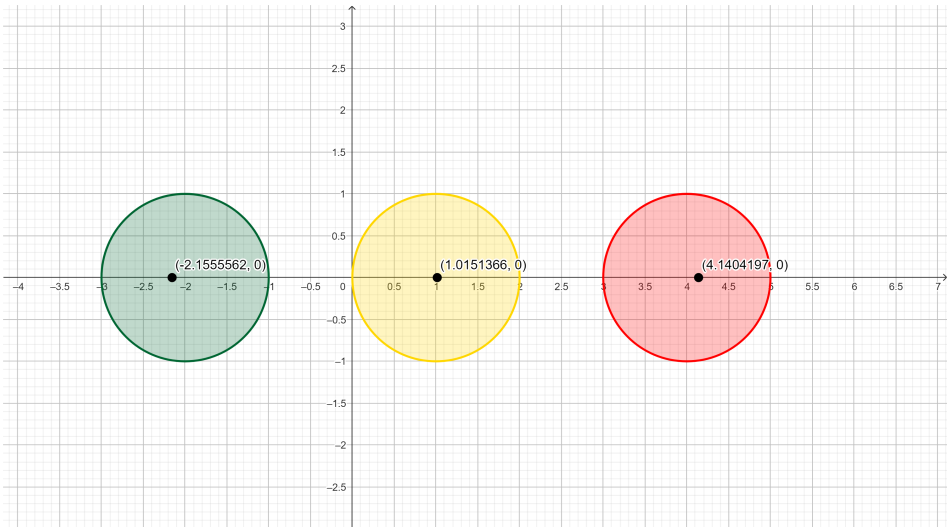

Figura 1: Discos de Gerschgorin (Sem Intersecção)

Entretanto, existe um problema, se os discos se intersectarem o método para encontrar os autovalores pode falhar, pois um autovalor pode, simutaneamente, ser o mais próximo de dois centros, como segue no exemplo abaixo

```
--> [C]=Gerschgorin_Real(A)
 C  =

   1.    2.
   2.    4.
   4.    4.

--> Find_eigenvalues(A,x0,epsilon,C(:,1),M)
 ans  =

   0.7024336
   0.7024336
   6.5160459
```

Analisando o espectro da matriz, vemos que um dos autovalores foi excluído e substituído por uma repetição do outro

```
--> spec(A)
 ans  =

  -0.2184795
   0.7024336
   6.5160459
```

Analisando graficamente, vemos que os discos de Gerschgorin de centros 1 e 2 possuem ambos o autovalor 0.7024336, que é o mais próximo do disco de centro 2, mas acaba sendo mais próximo

do disco de centro 1 do que o -0.2184795, que também está no disco de centro 1, como segue na figura abaixo

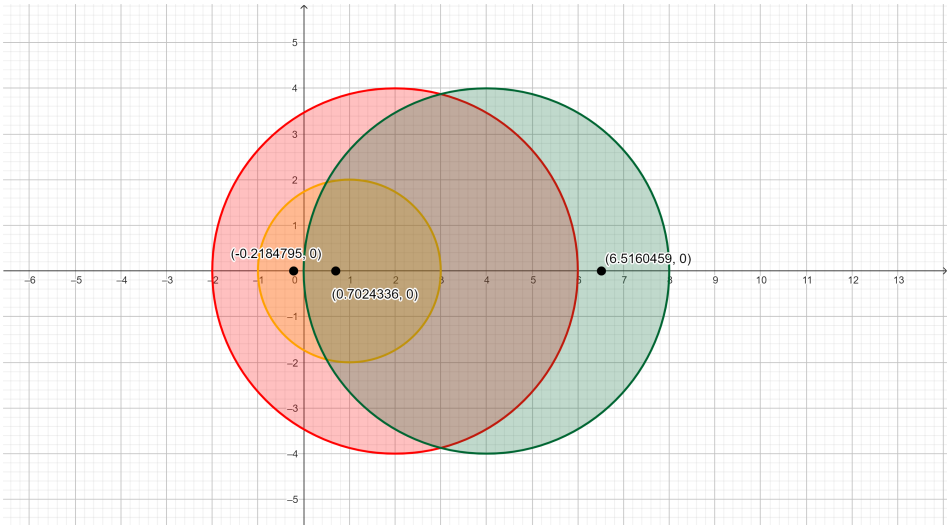

Figura 2: Discos de Gerschgorin (Com Interseccção)

5) Faça outros testes que achar convenientes ou interessantes!!! :)

Vamos agora efetuar alguns testes para alguns casos específicos que podem ocorrer.

Teste A - Autovalores Dominantes Repetidos

Teste A1 - $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, x_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

```
--> [lambda,x1,k,n_erro,time]=Metodo_potencia_1_corrigido(A,x0,epsilon,M)
 lambda  =

   1.
 x1  =

   1.
   1.
   1.
 k  =

   1.
 n_erro  =

   0.
 time  =

   0.0002323

--> [lambda,x1,k,n_erro,time] = Metodo_potencia_2(A,x0,epsilon,M)
 lambda  =

   1.0000000
 x1  =

   0.5773503
   0.5773503
   0.5773503
 k  =

   1.
 n_erro  =

   0.
 time  =

   0.0000706
```

Para a identidade é simples e, de fato, funciona. Agora, vamos ver outro caso

Teste A2 - $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 1 \\ 0 & 0 & 5 \end{bmatrix}, x_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

```
--> [lambda,x1,k,n_erro,time]=Metodo_potencia_1_corrigido(A,x0,epsilon,10^6)
 lambda  =

   5.0000500
 x1  =

   0.
   5.0000500
   0.0002500
 k  =

   100001.
 n_erro  =

   4.999D-10
 time  =

   1.5587959

--> [lambda,x1,k,n_erro,time] = Metodo_potencia_2(A,x0,epsilon,10^6)
 lambda  =

   5.0000500
 x1  =

   0.
   5.0000500
   0.0002500
 k  =

   100001.
 n_erro  =

   4.999D-10
 time  =

   1.0349869
```

De fato, para autovalores repetidos, o método ainda converge , mas a depender, pode demorar. Sem perda de generalidade, considere que há somente dois autovalores dominantes repetidos

$\lambda_1 = \lambda_2 = \lambda$. Então, teríamos

$$\begin{aligned} x_k = A^k x_0 &= \lambda^k c_1 v_1 + \lambda^k c_2 v_2 + \lambda_3^k c_3 v_3 + ... + \lambda_n c_n v_n \\ &= \lambda^k \left(c_1 v_1 + c_2 v_2 + \left(\frac{\lambda_3}{\lambda}\right)^k c_3 v_3 + ... + \left(\frac{\lambda_n}{\lambda}\right)^k c_n v_n \right) \\ &\to \lambda^k c_1 v_1 + \lambda^k c_2 v_2 \end{aligned}$$

isto é, o autovetor x relativo a λ se aproxima de uma combinação linear dos vetores v_1 e v_2 (claro, que sempre estaremos normalizando os vetores nas iterações, para que a apoximação não tenda a ∞ devido ao λ^k). Como se observa, o número de iterações é bem maior, sendo necessária mais de 10^6 iterações para o autovetor convergir. O caso de A não ser diagonalizável também se encaixa neste patamar. Na prática, o problema na convergência, em geral, irá advir da escolha do x_0, como veremos adiante.

Teste B - Autovalores complexos

Para este caso, enfretamos um problema que é o fato de que estamos analisando se $\lambda_k < 0$ no decorrer de cada iteração, sendo que neste contexto $\lambda_k \in \mathbb{C}$. O método, portanto, só irá funcionar se ele não retornar erro, isto é, se não ocorrer de existir algum $k \in \mathbb{N}$ tal que $\lambda_k \in \mathbb{C} \backslash \mathbb{R}$.

Teste C - Escolhendo o x_0

Devemos ser cautelosos na escolha do x_0, pois o método só irá funcionar se o vetor inicial x_0 tiver uma componente não-nula na direção do vetor dominante v_1. Caso contrário, teríamos $x_k \to \lambda^k \cdot 0 \cdot v_1 = 0$ **teoricamente**. Pode ocorrer de no decorrer das iterações, mesmo neste caso, um erro de arredondamento ocorrer e fazer com que algum x_k tem a coordenada na direção de v_1 diferente de 0 e, neste caso, os erros de arrendondamento de fato ajudam! Claro que o contrário poderia ocorrer e haver um arredondamento que anulasse essa coordenada, mas isto é mais difícil de acontecer. Segue abaixo um exemplo onde o x_0 escolhido falha para aproximar o resultado.

Teste C - $A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -1 \\ 0 & 0 & 0.1 \end{bmatrix}, x_0 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$

Veja que o autovalor dominante neste caso é 2. Mas ao aplicar no algoritmo, ele retorna 1.

```
--> [lambda,x1,k,n_erro,time]=Metodo_potencia_1_corrigido(A,x0,epsilon,M)
 lambda  =

   1.
 x1  =

   1.
   0.
   0.
 k  =

   1.
 n_erro  =

   0.
 time  =

   0.0001842

--> [lambda,x1,k,n_erro,time] = Metodo_potencia_2(A,x0,epsilon,M)
 lambda  =

   1.
 x1  =

   1.
   0.
   0.
 k  =

   1.
 n_erro  =

   0.
 time  =

   0.0001657
```

Teste D - Testando Dimensão Alta

Vamos gerar uma matriz aleatória (simétrica) $1000x1000$ com a função *Generate_symetric_matrix* anexada no arquivo *Generate_symetric_matrix.sci* e o vetor $x_0 = (1, 1, ..., 1)$ e analisar como os métodos vão se sair para encontrar o autovalor dominante e o número de iterações necessárias.

```
--> [lambda,x1,k,n_erro,time]=Metodo_potencia_1_corrigido(A,x0,epsilon,M)
 lambda  =

   499.90863
 x1  =

   481.33207
   475.96281
   465.38877
   ...
   474.16549
   471.58954
 k  =

   7.
 n_erro  =

   3.221D-11
 time  =

   0.0173184

--> [lambda,x1,k,n_erro,time] = Metodo_potencia_2(A,x0,epsilon,M)
 lambda  =

   499.90863
 x1  =

   16.063271
   15.884085
   15.531202
   ...
   15.824104
   15.738138
 k  =

   7.
 n_erro  =

   8.451D-12
 time  =

   0.0030689
```

Como se observa, ambos os métodos funcionaram, convergindo para o mesmo autovalor e os tempos de execução são similares, mas o **quociente de Raylleigh** se mostra um pouco mais rápido que a **mudança de escala**, sugerindo que para dimensões maiores, este método seja de melhor uso que o outro.

Teste E - Não utilizar um autovalor como parâmetro α para a Potência Inversa Deslocada

Utilizar λ como o parâmetro α irá prejudicar o algoritmo, pois é necessário que a matriz $A - \alpha I$ seja inversível, que não será o caso, pois nesse caso $\lambda - \alpha = \lambda - \lambda = 0$ será autovalor. De fato, temos

Teste E - $A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 3 \\ 2 & 2 & 2 \end{bmatrix}, x_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \alpha = 1$

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

   Nan
 x1  =

   Nan
   Nan
   Nan
 k  =

   1.
 n_erro  =

   Nan
 time  =

   0.0006376
```