Álgebra Linear - Aula Prática 3

Iara Cristina Mescua Castro

20 de maio de 2022

1. MÉTODO DA POTÊNCIA - versão 1

Escreva uma função Scilab:

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

```

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.

Critério de parada: sendo erro = x1 – x0 (diferença entre dois iterados consecutivos), parar quando $n_{erro} <$ épsilon ou $k > M$.

ALGORITMO – versão 1

$k = 0$
x0=x0/(coordenada de maior módulo de x0)
x1 = A*x0 (aproximação do autovetor dominante)
n_erro = épsilon + 1 (obriga a entrar no loop)
Enquanto $k \leq M$ e $n_erro \geq$ épsilon
    lambda = coord. de maior módulo de x1 (aproximação autovalor dominante)
    x1=x1/lambda
    n_erro = norma infinito de x1 – x0
    x0 = x1
    x1=A*x0
    k=k+1
Fim Enquanto
Mensagem e retorna

Código:

```
function [lambda, x1, k, n_erro]=Metodo_da_Potencia1_v1(A, x0, eps, M)
tic();
realtimeinit(1);
k = 0;
```

```
[n, m]=size(A)

if  n <> m then // verifica se as dimensoes de A sao iguais
  msg = gettext("%s: A nao e uma matriz quadrada. \n")
  error(msprintf(msg, "Error", 1))
end

if x0 == 'x0' then //se x0 nao foi definido, gera uma matriz de zeros
  //cujo primeiro termo e 1
  x0 = zeros(n,1)
  x0(1,1) = 1
end

if  size(x0,1) <> n then // verifica se o tamanho de x0 e compativel com A
  msg = gettext("%s: O tamanho de x0 nao e compativel com A. \n")
  error(msprintf(msg, "Error", 1))
end

if eps < 0 then // se eps for negativo, torna-o positivo para diminuir iteracoes
  eps = -eps
end

[val0,ind0]= max(abs(x0)); // maior valor de x0 em modulo de x0 e coordenada
x0 = x0/x0(ind0);
x1 = A*x0;
n_erro = eps + 1;

while k <= M & n_erro >= eps //crit de parada
  [val,ind]= max(abs(x1));
  lambda = x1(ind);
  x1 = x1/lambda;
  n_erro = norm(x1 - x0, 'inf');
  x0 = x1;
  x1 = A*x0;
  k = k + 1;
end

x1 = x1/norm(x1,'inf');
endfunction
```

ALGORITMO – versão 2

$k = 0$
x0=x0/($norma_2$ de x0)
$x1 = A * x0$ (aproximação do autovetor dominante)
n_erro = épsilon + 1 (Obriga a entrar no loop)
Enquanto $k \leq M$ e $n_erro \geq$ épsilon
    lambda = $x1T * x0$ (Quociente de Rayleigh; x0 é unitário)
    Se lambda¡0 então x1 = − x1 (Mantém x1 com mesmo sentido de x0)
    x1 = $x1/norma_2$ de x1
    n_erro = $norma_2$ de x1 – x0
    x0 = x1
    $x1 = A * x0$
    k=k+1
Fim Enquanto
Mensagem e retorna

Código:

```
function [lambda, x1, k, n_erro]=Metodo_da_Potencia1_v2(A, x0, eps, M)
tic();
realtimeinit(1);
k = 0;
[n, m]=size(A)
```

```
if  n <> m then // verifica se as dimensoes de A sao iguais
  msg = gettext("%s: A nao e uma matriz quadrada. \n")
  error(msprintf(msg, "Error", 1))
end

if x0 == 'x0' then //se x0 nao foi definido, gera uma matriz de zeros
  //cujo primeiro termo e 1
  x0 = zeros(n,1)
  x0(1,1) = 1
end

if  size(x0,1) <> n then // verifica se o tamanho de x0 e compativel com A
  msg = gettext("%s: O tamanho de x0 nao e compativel com A. \n")
  error(msprintf(msg, "Error", 1))
end

if eps < 0 then // se eps for negativo, torna-o positivo para diminuir iteracoes
  eps = -eps
end

x0 = x0/norm(x0,2);
x1 = A*x0;
n_erro = eps + 1;
while k <= M & n_erro >= eps //crit de parada
  lambda = x1' * x0;
  if lambda < 0 then
  x1 = -x1;
end
x1 = x1/norm(x1,2);
n_erro = norm(x1 - x0, 2);
x0 = x1;
x1 = A*x0;
k = k + 1;
end

x1 = x1/norm(x1,2);
endfunction
```

MÉTODO DA POTÊNCIA - EXPLICAÇÃO

O método da potência é aplicado em matrizes A_{nxn} com um autovalor dominante λ_1, sendo $|\lambda_1| > |\lambda_2| \geq ... \geq |\lambda_n|$. Esse método produz uma sequência que converge pra λ_1 e uma sequência que converge para v_1, o autovetor correspondente de λ_1.

Para os códigos das funções, comecei definindo tic() no começo e toc() no final, e com display, mostrar o tempo de execução assim que as funções forem utilizadas. E também criei algumas condições para verificar as entradas e definindo um x_0 não-nulo caso ele esteja como 'x0'. Nas iterações, teremos as sequências:

$x_1 = x_0 * A, x_2 = x_1 * A$.... Colocamos a primeira iteração por fora do while, para já começar calculando o lambda iniciar a partir dele.

Na primeira versão, usamos coordenada de maior módulo de x_1 para calcular lambda.
$\max(\text{abs}(x_k))/\max(\text{abs}(x_{k-1})) \to \lambda_1 \cdot x_{k-1}$
$x_{k+1} = A_{k_k} \approx \lambda_1 \cdot x_{k-1}$

Na segunda versão, usamos o quociente de Rayleigh para calcular lambda:
$Ax = \lambda x$
$\frac{(Ax)\cdot x}{x\cdot x} = \frac{(\lambda x)\cdot x}{x\cdot x} = \frac{\lambda(x\cdot x)}{x\cdot x} = \lambda$
No processo interativo:
$\frac{(Ax_k)\cdot x_k}{x_k\cdot x_k} = \frac{(\lambda x_{k+1})\cdot x_k}{x_k\cdot x_k} = \frac{\lambda x_{k+1}\cdot x_k}{||x_k||_2^2}$

Logo, λ_1 depende do autovetor normalizado pela norma 2, já que se a cada iteração mantivermos $||x_k||_2 = 1$, tem-se que:

$x_{k+1} \cdot x_k \to \lambda_1$

Então para obtermos esse produto interno calculamos lambda multiplicado a transposta de x_1 por x_0 a cada interação.

E repetimos essas operações em ambas versões enquanto k for menor que o número máximo de iterações M ou até a norma 'inf' ou 2 de (x1 - x0) for menor que ϵ.

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"**.

Variáveis de entrada:
A: matriz real n x 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:
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

Critério de parada: sendo erro = x1 – x0 (diferença entre dois iterados consecutivos), parar quando a n_{erro} < épsilon ou $k > M$.

ALGORITMO Método da Potência Deslocada com Iteração Inversa

k=0
x0=x0/($norma_2$ de x0)
n_erro = épsilon + 1 (Obriga a entrar no loop)
Enquanto k¡=M e n_{erro} ¿= épsilon
Resolva o sistema (A – alfa*I)*x1 = x0 para achar x1
x1 = x1/($norma_2$ de x1)
lambda = $x1^T$*A*x1 (Quociente de Rayleigh; x1 é unitário)
Se $x1^T$*x0 < 0 então x1 = − x1 (Mantém x1 com mesmo sentido de x0)
n_{erro} = $norma_2$ de x1 – x0
x0 = x1
k=k+1
Fim Enquanto
lambda1 = ...
Mensagem e retorna

Código:

```
function [lambda, x1, k, n_erro]= Potencia_deslocada_inversa(A, x0, eps, alfa, M)
k = 0;
[n, m]=size(A)

if  n <> m then // verifica se as dimensoes de A sao iguais
  msg = gettext("%s: A nao e uma matriz quadrada. \n")
  error(msprintf(msg, "Error", 1))
end

if x0 == 'x0' then //se x0 nao foi definido, gera uma matriz de zeros
  //cujo primeiro termo e 1
  x0 = zeros(n,1)
  x0(1,1) = 1
end

if  size(x0,1) <> n then // verifica se o tamanho de x0 e compativel com A
  msg = gettext("%s: O tamanho de x0 nao e compativel com A. \n")
  error(msprintf(msg, "Error", 1))
end

if eps < 0 then // se eps for negativo, torna-o positivo para diminuir int
  eps = -eps
end

I = eye(n,n);
x0 = x0/norm(x0,2);
n_erro = eps + 1;

//Resolver o sistema na primeira interacao usando a funcao Gaussian_Elimination_4
[x0, C, P] = Gaussian_Elimination_4((A - alfa * I), x0);

//AP = LU
while k <= M & n_erro >= eps
  x1 = Resolve_com_LU(C, x0, P); //resolver o restante das interacoes por dois sistemas
     triangulares
  x1 = x1/norm(x1,2);
  lambda = x1' * A * x1;
  if x1' * x0 < 0
    x1 = -x1;
  end
n_erro = norm(x1 - x0, 2);
x0 = x1;
k = k + 1;
end

endfunction
```

MÉTODO DA POTÊNCIA DESLOCADA com ITERAÇÃO INVERSA - EXPLICAÇÃO

O método da potência deslocada com iteração inversa é aplicado em matrizes A_nxn para encontrar um autovalor específico que esteja mais próximo de um alfa dado na entrada. Como sabemos:

$Ax = \lambda x$, para x diferente de 0.
Multiplicando pela inversa de A em ambos lados:

$A^{-1}Ax = A^{-1}\lambda x$
$Ix = \lambda A^{-1}x$
$x = \lambda A^{-1}x$
$A^{-1}x = \frac{1}{\lambda}x$

Com a matriz $(A - \alpha I)$

$Ax = \lambda x$
$Ax - \alpha Ix = \lambda x - \alpha Ix$
$(A - \alpha I)x = (\lambda - \alpha)x$
$\frac{x}{\lambda - \alpha} = \frac{(A-\alpha I)^{-1}(\lambda - \alpha)x}{\lambda - \alpha}$
$(A - \alpha I)^{-1}x = \frac{1}{\lambda - \alpha}x$

$v = \frac{1}{\lambda - \alpha}$, onde $|\lambda - \alpha|$ é mínimo, e correspondendo a λ_i mais próximo de α

Na nossa função, a cada iteração, o vetor x_k seria multiplicado pela inversa da matriz $(A - \alpha I)$ e normalizado. Mas em vez disso, note que não é necessário calcular a matriz inversa e iremos resolver o sistema linear a cada iteração:

$(A - \alpha I) * x_{k+1} = x_k$

Para otimizar a função, vamos resolver a primeira iteração com a função *Gaussian_Elimination_4* para além de resolver o sistema, também encontrar a matriz C com a decomposição LU de A e a matriz de permutação P.

Ao começar o while, já poderemos resolver dois sistemas triangulares com as matrizes L e U, usando a função *Resolve_com_LU*, onde C e P sempre serão os mesmos que já foram calculados, e apenas atualizando x_0 e x_1 a cada iteração.

```
function x=Resolve_com_LU(C, b, P)
b = P*b;
n = size(C,1); //linhas
L = tril(C, -1) + eye(n,n); //Matriz L (triangular inferior)

U = triu(C); //Matriz U (triagular superior)
x = zeros(n,1);

y = resolveL(L,b); //vetor y
x = resolveU(U,y); //vetor x

endfunction

//Ly = b
function y = resolveL(L,b)
n = size(L,1);
y = zeros(n,1);
y(1) = b(1);
for i=2:n
  y(i) = (b(i) - L(i,1:i-1)*y(1:i-1))/L(i,i);
end
endfunction

//Ux = y
function x = resolveU(U,y)
n = size(U,1);
x = zeros(n,1);
x(n) = y(n)/U(n,n);
for i = n-1:-1:1
  x(i) = (y(i) - U(i, i+1:n) * x(i+1:n)) / U(i, i);
end
endfunction
```

Assim como nas outras funções, repetimos essas operações enquanto k for menor que o número máximo de iterações M ou até a norma 2 de (x1 - x0) for menor que ϵ.

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.

MATRIZES COM AUTOVALORES REAIS

Matrizes simétricas sempre terão autovalores reais. Isso acontece, pois: Pelo teorema espectral, se A é uma matriz nxn simétrica, com entradas reais, então tem n autovetores ortogonais. Todas as raízes do polinômio característico de A são números reais.

Seja $\bar{v}$ o complexo conjugado de v, é verdade que $\bar{v} \cdot v \geq 0$, sendo igual apenas se $v = 0$.

$$\begin{bmatrix} a_1 - b_1 i \\ a_2 - b_2 i \\ \vdots \\ a_n - b_n i \end{bmatrix} \cdot \begin{bmatrix} a_1 + b_1 i \\ a_2 + b_2 i \\ \vdots \\ a_n + b_n i \end{bmatrix} = (a_1^2 + b_1^2) + (a_2^2 + b_2^2) + ...(a_n^2 + b_n^2)$$

, que sempre são não-negativas.

Agora, supomos que z um número complexo $z = a + bi$ e $\bar{z} = a - bi$ sua conjugada. Temos $z\bar{z} = (a + bi)(a - bi) = a^2 + b^2$, então $z\bar{z}$ é um valor não negativo e real. Se w também for um valor complexo, $\bar{w}z = \bar{w}\bar{z}$

Por contradição, vamos supor que λ é um autovalor complexo de uma matriz simétrica A. E há um autovetor não-nulo v, cujo $Av = \lambda v$.

Tomando o conjugado complexo de ambos os lados, e notando que $\bar{A} = A$ já que A tem real entradas, temos:

$Av = \lambda v \implies Av = \lambda v$. Então, usando $A^T = A$:

$\bar{v}^T Av = \bar{v}^T(Av) = \bar{v}^T(\lambda v) = \lambda(\bar{v} \cdot v)$
$\bar{v}^T Av = (A\bar{v})^T v = (\bar{\lambda}\bar{v})^T v = \lambda(\bar{v} \cdot v)$

Já que $v \neq 0$, temos que $\bar{v}v \neq 0$ e $\bar{\lambda} = \lambda$, ou seja, λ pertence aos reais.

TESTES no SCILAB

Agora iremos testar as duas primeiras funções para diversas matrizes A. Sabendo que elas terão autovalores reais, utilizarei uma função, que gera matrizes aleatórias simétricas.

```
function A = Matriz_Simetrica_Aleat(n)
A = floor(-((n^2)*rand(n,n,'uniform')) + ((n^2)*rand(n,n,'uniform')))
A = tril(A) + triu(A', 1)
endfunction

```

Que gera uma matriz aleatória entre $[-n^2, n^2]$, e forma a matriz simétrica somando a parte inferior com a parte superior da sua transposta (que é ela mesma), sem a diagonal, pois já estava incluida no tril(A).

ORDEM 3

Exemplo 1:

```
--> A = Matriz_Simetrica_Aleat(3)
 A  =

   2.   0.   2.
   0.   3.   0.
   2.   0.  -2.

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

   0.0021468
 lambda  =

   2.
 xl  =

   1.
   0.
   0.
 k  =

   101.
 n_erro  =

   1.
```

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-10), 100)

   0.0017078
 lambda  =

   2.
 xl  =

   1.
   0.
   0.
 k  =

   101.
 n_erro  =

   0.7653669

--> spec(A)
 ans  =

  -2.8284271
   2.8284271
   3.
```

OBSERVAÇÃO 1

Apesar do autovalor dominante ser 3, ambas funções retornam $\lambda = 2$. Isso ocorre pois o $x0$ inicial deve ter uma componente c_1 na direção do autovetor dominante v_1, que nesse caso é $(0, 1, 0)$.

Isso pode ser solucionado ao mudar o x_0, para $[1; 1; 1]$:

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, [1;1;1], 10^(-5), 100)

   0.0022244
 lambda  =

   3.
 xl  =

   0.0024616
   1.
   0.0024616
 k  =

   101.
 n_erro  =

   0.0027693



--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, [1;1;1], 10^(-5), 100)

   0.0010714
 lambda  =

   2.9999847
 xl  =

   0.0024616
   0.9999939
   0.0024616
 k  =

   101.
 n_erro  =

   0.0029191
```

Apesar das funções retornarem o autovalor dominante correto, note que todas as iterações foram utilizadas.

Exemplo 2:

```
--> A = Matriz_Simetrica_Aleat(3)
 A  =

   7.  -3.   2.
  -3.   6.   5.
   2.   5.   0.

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-10), 100)

   0.0026678
 lambda  =

   10.022531
 xl  =

  -0.7632176
   1.
   0.3465756
 k  =

   59.
 n_erro  =

   8.686D-11



--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-10), 100)

   0.0010403
 lambda  =

   10.022531
 xl  =

   0.5849114
  -0.7663756
  -0.2656071
 k  =

   58.
 n_erro  =

   9.260D-11

--> spec(A)
 ans  =

  -3.8082558
   6.7857249
   10.022531
```

OBSERVAÇÃO 2

À seguir, aumentando o valor de ϵ de 10^{-10} para 10^{-5}, observamos uma queda no número de iterações pela metade.

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 100)

   0.0005117
 lambda  =

   10.022558
 xl  =

  -0.7632277
   1.
   0.3465720
 k  =

   30.
 n_erro  =

   0.0000071
```

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 100)

   0.0003881
 lambda  =

   10.022531
 xl  =

   0.5849195
  -0.7663713
  -0.2656015
 k  =

   29.
 n_erro  =

   0.0000076
```

Exemplo 3:

```
--> A = Matriz_Simetrica_Aleat(3)
 A  =

   7.    3.    4.
   3.   -9.   -1.
   4.   -1.    7.

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 100)

   0.0020089
 lambda  =

   11.104636
 xl  =

   1.
   0.1019869
   0.9496677
 k  =

   86.
 n_erro  =

   0.0000091
```

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 100)

   0.0013625
 lambda  =

   11.104616
 xl  =

   0.7231463
   0.0737448
   0.6867467
 k  =

   83.
 n_erro  =

   0.0000098

--> spec(A)
 ans  =

  -9.7251760
   3.6205599
   11.104616
```

OBSERVAÇÃO 3

Mesmo as funções operando normalmente, o número de iterações necessário em ambas versões parece alto para matrizes de ordem 3. O tempo de execução da versão 2 se saiu melhor, cerca de duas vezes mais rápido, em todos os casos.

ORDEM 5

Exemplo 1:

```
--> A = Matriz_Simetrica_Aleat(5)
 A  =

  -4.   -8.   -19.   14.  -6.
  -8.   -6.    0.    2.   -23.
  -19.   0.    7.   -2.    4.
   14.   2.   -2.   -3.    5.
  -6.   -23.   4.    5.    7.

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 100)

   0.010076
 lambda  =

  -32.817233
 xl  =

  -0.9826670
  -1.
  -0.3598414
   0.6333007
  -0.7690983
 k  =

   62.
 n_erro  =

   0.0000091
```

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 100)

   0.0007278
 lambda  =

  -32.817128
 xl  =

  -0.5592308
  -0.5690932
  -0.2047791
   0.3604049
  -0.4376844
 k  =

   61.
 n_erro  =

   0.0000090

--> spec(A)
 ans  =

  -32.817128
  -15.673790
  -1.5116588
   24.187534
   26.815042
```

Exemplo 2:

```
--> A = Matriz_Simetrica_Aleat(5)
 A  =

   9.    14.  -17.   7.   -3.
   14.   18.  -5.   -2.   -2.
  -17.  -5.   -16.  -2.   -1.
   7.   -2.   -2.   -13.  -11.
  -3.   -2.   -1.   -11.  -15.

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 100)

   0.0024558
 lambda  =

   33.215394
 xl  =

   0.9369230
   1.
  -0.4286615
   0.1467900
  -0.1243730
 k  =

   45.
 n_erro  =

   0.0000078
```

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 100)

   0.0008221
 lambda  =

   33.215398
 xl  =

   0.6467534
   0.6902971
  -0.2959079
   0.1013268
  -0.0858571
 k  =

   44.
 n_erro  =

   0.0000094

--> spec(A)
 ans  =

  -25.921430
  -24.820367
  -5.5517638
   6.0781630
   33.215398
```

Exemplo 3:

```
--> A = Matriz_Simetrica_Aleat(5)
 A  =

   15.  -15.  -9.    12.  -5.
  -15.  -20.   11.   4.    4.
  -9.    11.   0.   -4.   -3.
   12.   4.   -4.   -4.   -24.
  -5.    4.   -3.   -24.  -10.

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 100)

   0.002227
 lambda  =

  -37.610542
 xl  =

   0.3509048
   0.9974532
  -0.3934177
  -1.
  -0.9931453
 k  =

   101.
 n_erro  =

   0.0001898
```

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 100)

   0.0012045
 lambda  =

  -37.606928
 xl  =

  -0.1943732
  -0.5525092
   0.2179219
   0.5539200
   0.5501230
 k  =

   101.
 n_erro  =

   0.0001187

--> spec(A)
 ans  =

  -37.606929
  -21.896340
  -5.1686099
   11.982496
   33.689382
```

Mesmo com a mesma ordem das demais, note que esse exemplo utilizou todas as iterações possíveis, podemos ajustar M para 200. É um número que parece relativamente grande para uma matriz de ordem 5. Mesmo aqui, o tempo de execução da versão 2 além de ser mais rápido, exige menos iterações que a versão 1.

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 200)

   0.0019444
 lambda  =

  -37.606743
 xl  =

   0.3508203
   0.9974350
  -0.3933721
  -1.
  -0.9930712
 k  =

   128.
 n_erro  =

   0.0000097
```

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 200)

  0.0024673
lambda  =

 -37.606929
xl  =

 -0.1943336
 -0.5525232
  0.2179055
  0.5539443
  0.5501050
k  =

  124.
n_erro  =

  0.0000095
```

OBSERVAÇÃO 4

Ao fazer $spec(A)$ para verificar os autovalores de A, observamos que nos exemplos 1 e 3 o autovalor dominante é negativo. Mesmo assim, ambas funções continuam operando normalmente.

OBSERVAÇÃO 5

Era esperado um aumento drástico no número de iterações pelo tamanho das matrizes A ser maior em relação às de ordem 3. Mas houve o oposto, e o número de iterações manteve-se próximo, e até diminuiu em alguns exemplos.

ORDEM 10

Exemplo 1:

```
--> A = Matriz_Simetrica_Aleat(10);

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 200)

  0.0023719
lambda  =

  225.55564
xl  =

  0.3596581
 -0.1026468
 -0.5251411
  0.1795587
 -0.4477778
 -0.1211318
 -0.2669763
  1.
 -0.2888288
  0.6900846
k  =

  101.
n_erro  =

  0.0000089
```

A partir de agora gerei a matriz aleatória simétrica com ; no final para não ocupar muito espaço.

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 200)

  0.0018331
lambda  =

  225.55498
xl  =

 -0.2374618
  0.0677714
  0.3467210
 -0.1185521
  0.2956416
  0.0799763
  0.1762692
 -0.6602423
  0.1906972
 -0.4556232
k  =

  99.
n_erro  =

  0.0000097
```

OBSERVAÇÃO 6

Ajustei M para 200, esperando um aumento no número de iterações, mas novamente, ele está próximo do que foi utilizado em ordens menores. O tempo de execução em ambas funções continuou o mesmo.

Sabendo que há 10 autovalores agora, há uma forma de pegar diretamente o autovalor dominante e não precisar analisar todos os valores, com os seguintes comandos:

```
--> eg = spec(A);

--> [val,ind] = max(abs(eg));

--> lambda = eg(ind)
 lambda  =

   225.55498
```

Pegamos o valor máximo do módulo de $spec(A)$ e seu índice, para então, usar esse índice e pegar seu valor sem módulo. Lambda continua compatível com o que foi obtido das duas funções.

Exemplo 2:

```
--> A = Matriz_Simetrica_Aleat(10);

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 200)

   0.001498
 lambda  =

   222.09866
 xl  =

   0.4822397
   0.2374830
  -0.7007821
  -0.5441397
   0.1546828
   1.
  -0.5765910
   0.3775411
   0.0386160
   0.7591598
 k  =

   55.
 n_erro  =

   0.0000085



--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 200)

   0.0006972
 lambda  =

   222.09744
 xl  =

   0.2715879
   0.1337452
  -0.3946660
  -0.3064485
   0.0871145
   0.5631788
  -0.3247246
   0.2126228
   0.0217479
   0.4275437
 k  =

   53.
 n_erro  =

   0.0000100
```

```
--> eg = spec(A);

--> [val,ind] = max(abs(eg));

--> lambda = eg(ind)
 lambda  =

   222.09744
```

ORDEM 100

Exemplo 1:

Sabendo que agora temos 100 autovalores e um autovetor dominante 100x1, vamos colocar ; no final dos comandos para não ocupar espaço com ele.

```
--> A = Matriz_Simetrica_Aleat(100);

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 200);

   0.0062189

--> lambda
 lambda  =

  -80961.996

--> k
 k  =

   201.

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 200);

   0.0075556

--> lambda
 lambda  =

  -80196.453

--> k
 k  =

   201.
```

```
--> eg = spec(A);

--> [val,ind] = max(abs(eg));

--> lambda = eg(ind)
 lambda  =

  -80202.763
```

OBSERVAÇÃO 7

Mantivemos M em 200 e dessa vez houve sim, um aumento no número necessário de iterações, gastando todas as 200. Mesmo assim parece estar convergindo para o autovalor dominante correto, visto que está próximo de lambda obtido por spec(). O tempo de execução aumentou também, mas não drasticamente. Assim como nas outras ordens, a versão 2 é sempre mais eficaz em número de iterações.

Exemplo 2:

```
--> A = Matriz_Simetrica_Aleat(100);

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 500);

   0.0104132

--> A = Matriz_Simetrica_Aleat(100);

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 500);

   0.0159816

--> lambda
 lambda  =

  -79147.556

--> k
 k  =

   501.

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 500);

   0.0154485

--> k
 k  =

   501.
```

```
--> eg = spec(A);

--> [val,ind] = max(abs(eg));

--> lambda = eg(ind)
 lambda  =

   76922.769
```

OBSERVAÇÃO 8

Houve algo incomum nesse exemplo, pois além de gastar todas as iterações, o autovalor dominante obtido não parece estar próximo do verdadeiro, visto que é negativo e o outro positivo. Podemos supor que há dois maiores autovalores que em módulo são bem próximos, então vamos aumentar o número de iterações máximo M para 100000.

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 100000);

   0.4949434

--> lambda
 lambda  =

   76922.569

--> k
 k  =

   18488.

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 100000);

   0.3661261

--> lambda
 lambda  =

   76922.769

--> k
 k  =

   18375.
```

OBSERVAÇÃO 9 Agora obtivemos o lambda correto em ambas versões. Além disso, o número de iterações foi extremamente grande, chegando a 18000. Mesmo assim, os tempos de execução não passaram de meio segundo. A observação 6 provavelmente está incorreta, visto que mesmo próximo de lambda as 200 iterações estava longe de ser suficiente para uma boa precisão.

```
--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v1(A, 'x0', 10^(-20), 100000);

   2.6805189

--> lambda
 lambda  =

   76922.769

--> [lambda, xl, k, n_erro] = Metodo_da_Potencial_v2(A, 'x0', 10^(-20), 100000);

   2.3645308

--> lambda
 lambda  =

   76922.769

--> k
 k  =

   100001.
```

OBSERVAÇÃO 10 Mesmo a precisão já sendo boa, por curiosidade, diminuí ϵ de 10^{-5} para 10^{-20}. Observe que todas as 100000 iterações foram gastas.

CONCLUSÕES

· Na prática, ambos métodos tiveram desempenhos similares, mas na maioria dos casos, a versão 2 se mostrou mais eficaz, mesmo com poucas iterações de diferença.

· Ao definir um ϵ muito pequeno, o desempenho de ambas funções caiu drasticamente. Os resultados são praticamente os mesmos, e $\epsilon = 10^{-5}$ funcionou bem para todos os casos.

· É preciso ter cuidado ao definir x_0, já que além de não poder ser nulo, precisa atender à condição citada na observação 1, e dependendo de seus valores pode acabar aumentando desnecessariamente o

número de iterações. Somado à isso, se $x0$ for exatamente igual a um autovetor de algum lambda, a função retorna esse lambda que talvez não necessariamente seja o dominante, mas por sorte, isso não aconteceu em nenhum de nossos exemplos.

---

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.

   O Teorema (Gershgorin) diz que seja $A = (aij)$ uma matriz quadrada complexa. Então todo autovalor de A está em um dos discos de Gershgorin. Logo, a teoria dos discos de Gershgorin é um resultado elementar que permite fazer deduções rápidas sobre as localizações dos autovalores.

   Para essa questão, além de utilizar a função de gerar matrizes simétricas aleatórias, também utilizei uma função que obtém os centros e raios dos discos de Gerschgorin nessas matrizes, devolvendo eles como vetores um ao lado do outro.

Código:

```
function [x]=Discos_de_Gershgorin(A)
//c: vetor de centros,
//r: vetor de raios
n = size(A,1);
c = diag(A);
r = sum(abs(A),2) - abs(c);
x = [c r]
endfunction

```

Agora é possível usar o Método da Potência Inversa para calcular um autovalor específico partindo de um alfa que já esteja próximo a ele.

ORDEM 3

Exemplo 1:

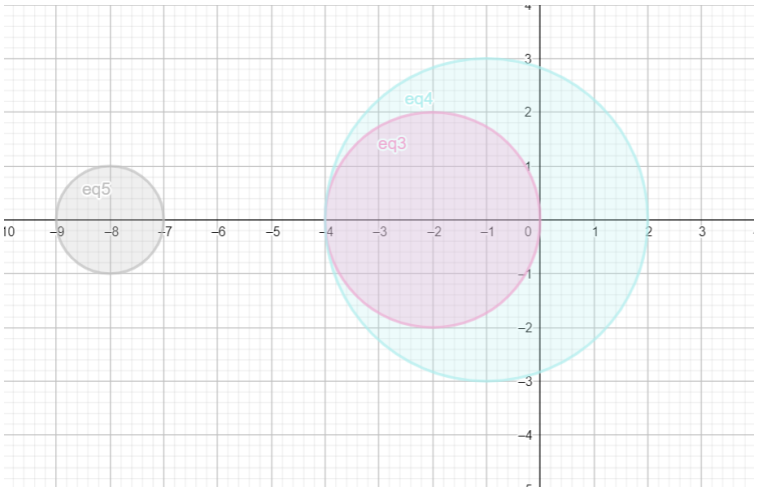
Vamos tentar calcular o autovalor dominante:

```
--> A = Matriz_Simetrica_Aleat(3)
 A  =

  -2.  -2.   0.
  -2.  -1.  -1.
   0.  -1.  -8.

--> [x]=Discos_de_Gershgorin(A)
 x  =

  -2.   2.
  -1.   3.
  -8.   1.
```



Observamos os centros: -2, -1, -8. O maior em módulo vem do -8, que está em um disco completamente afastado dos demais então não há riscos. Testando com $\alpha = -10$, obtemos:

```
--> [lambda, xl, k, n_erro]=Metodo_da_Potencia2(A, 'x0', 10^(-5), -10, 50)
 lambda  =

 -8.1537575
 xl  =

   0.0493323
   0.1517873
   0.9871813
 k  =

   12.
 n_erro  =

   0.0000031

--> spec(A)
 ans  =

 -8.1537575
 -3.4805092
  0.6342667
```

O teste deu certo, e obtivemos o autovalor dominante que está mais próximo de alfa.

Exemplo 2:

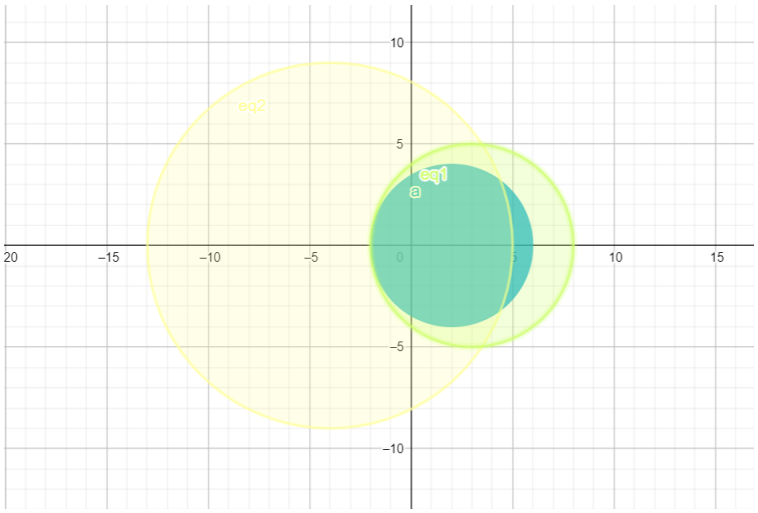
Agora vamos tentar calcular o **MENOR** autovalor em módulo:

```
--> A = Matriz_Simetrica_Aleat(3)
 A  =

   2.  -4.   0.
  -4.  -4.   5.
   0.   5.   3.

--> [x]=Discos_de_Gershgorin(A)
 x  =

   2.   4.
  -4.   9.
   3.   5.
```



Observamos os centros: 2, -4, 3. O menor em módulo vem do 2, há riscos já que todos os discos se intersetam no meio. Então testando com $\alpha = 2$, obtemos:

```
--> [lambda, xl, k, n_erro]=Metodo_da_Potencia2(A, 'x0', 10^(-5), 2, 50)
 lambda  =

   2.3547647
 xl  =

   0.8219348
  -0.0728984
   0.5648973
 k  =

   5.
 n_erro  =

   0.0000015

--> spec(A)
 ans  =

  -7.9070915
   2.3547647
   6.5523268
```

Novamente, tivemos sucesso e obtivemos o menor autovalor em módulo.

ORDEM 5

Exemplo 1:

Vamos tentar calcular o autovalor dominante:

```
-->  A = Matriz_Simetrica_Aleat(5)
 A  =

  -1.   -22.   14.  -2.   -2.
  -22.  -9.    8.   -15.  -11.
   14.   8.   -19.   14.  -6.
  -2.   -15.   14.   2.   -23.
  -2.   -11.  -6.   -23.   4.

--> [x]=Discos_de_Gershgorin(A)
 x  =

  -1.    40.
  -9.    56.
  -19.   42.
   2.    54.
   4.    42.
```

Observamos os centros na primeira coluna, e podemos deduzir que o autovalor dominante venha do -19, pois é o maior centro em módulo. Entretanto, ao visualizar os discos é evidente que todos se intersetam, pelo fato de seus raios estarem cada vez maiores, então a precisão diminui.

```
--> [lambda, x1, k, n_erro]=Metodo_da_Potencia2(A, 'x0', 10^(-5), -19, 100)
 lambda  =

  -18.791152
 x1  =

   0.1800806
  -0.2052630
  -0.6829061
  -0.3272411
  -0.5932879
 k  =

   4.
 n_erro  =

   0.0000002

--> spec(A)
 ans  =

  -49.382686
  -18.791152
  -5.6450111
   19.660548
   31.158301
```

Ao escolher $\alpha = -19$ não obtivemos sucesso, mesmo provavelmente ele sendo desse disco, seu autovalor correspondente estava muito distante, e a função acabou convergindo para o segundo maior autovalor, de outro disco.

Exemplo 2:

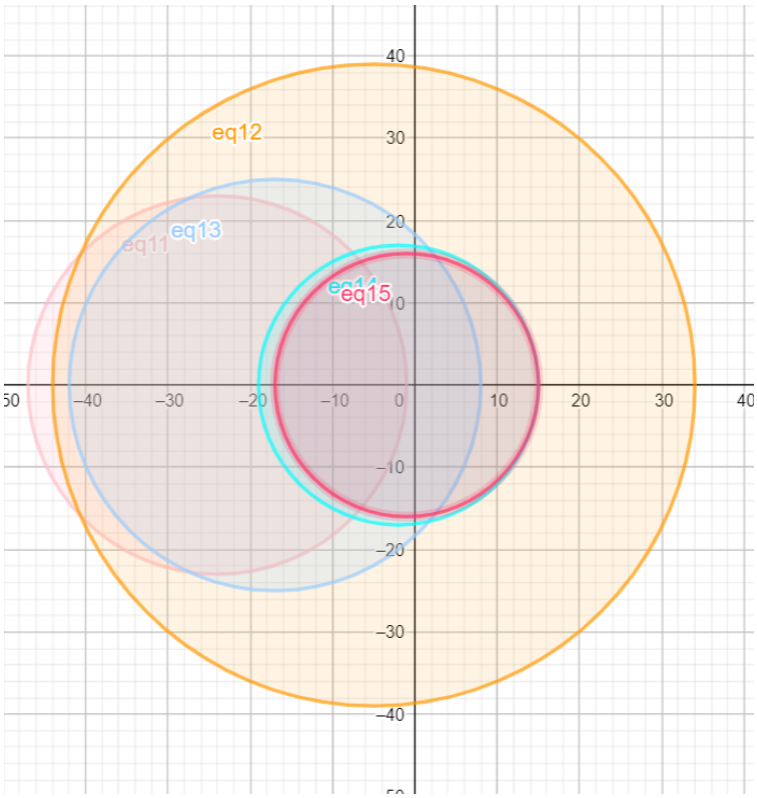
Vamos tentar calcular o **MENOR** autovalor em módulo:

```
--> A = Matriz_Simetrica_Aleat(5)
 A  =

  -24.  -9.   -1.    4.    9.
  -9.   -5.   -14.  -14.   2.
  -1.   -14.  -17.   7.   -3.
   4.   -14.   7.   -2.   -2.
   9.    2.   -3.   -2.   -1.

--> [x]=Discos_de_Gershgorin(A)
 x  =

  -24.   23.
  -5.    39.
  -17.   25.
  -2.    27.
  -1.    16.
```




```
--> [lambda, xl, k, n_erro]=Metodo_da_Potencia2(A, 'x0', 10^(-5), -1, 100)
 lambda  =

   2.4458866
 xl  =

   0.3535033
  -0.0860599
  -0.0859324
   0.0368462
   0.9267616
 k  =

   8.
 n_erro  =

   0.0000046

--> spec(A)
 ans  =

  -32.728290
  -22.989146
  -14.511339
   2.4458866
   18.782888
```

Ao usar $\alpha = -1$, obtivemos sucesso em obter o menor autovalor em módulo.

ORDEM 15

```
--> A = Matriz_Simetrica_Aleat(15);

--> [x]=Discos_de_Gershgorin(A)
 x  =

  -16.    810.
   55.    1154.
   147.   1330.
  -107.   745.
   64.    1410.
  -11.    1179.
  -92.    871.
  -33.    894.
  -123.   843.
  -71.    1032.
  -22.    1133.
   6.     1033.
   15.    864.
   132.   1241.
   50.    749.
```

```
--> [lambda, xl, k, n_erro]=Metodo_da_Potencia2(A, 'x0', 10^(-5), 147, 200)
 lambda  =

   135.24791
 xl  =

  -0.2963059
  -0.0524896
  -0.1958982
   0.3562914
  -0.0941386
   0.1010538
   0.4947000
  -0.2960447
  -0.0413000
  -0.0372313
   0.2667683
   0.1379603
   0.1976781
   0.2339119
   0.4534334
 k  =

   25.
 n_erro  =

   0.0000075
```

```
--> eg = spec(A);

--> [val,ind] = max(abs(eg));

--> eg(ind)
 ans  =

   718.36801
```

Houve uma perda drástica de precisão ao aumentar a ordem para 15. Não apenas não obtivemos o autovalor dominante, mas ele também não estava entre os 5 maiores.

CONCLUSÕES

Todos os discos dos exemplos se encontram no eixo x, por serem de matrizes A reais e não terem uma parte imaginária, logo, os autovalores não podem estar fora do eixo x também. Para matrizes

completamente aleatórias, discos com ordem maior que 10 já irão perder muita precisão, pois a chance de se intersetarem é muito grande. Entretanto funciona perfeitamente se:

1) Os centros estiverem extremamente afastados um do outro, em outras palavras, se a diagonal de A tiver valores muito distintos.

2) Se os raios forem extremamente pequenos em comparação com o centro, em outras palavras, se a soma dos módulos das linhas (tirando a diagonal) for bem pequena. Pois estão, os autovalores estarão restritos àqueles espaços e haverá precisão.

---

5. Faça outros testes que achar convenientes ou interessantes!!!

TESTES - MÉTODO DA POTÊNCIA para AUTOVALORES INTEIROS

Concluímos anteriormente que a versão 2 exerce um melhor desempenho que a 1, e também, a mudança de iterações dependendo da ordem das matrizes reais. Mas como será o desempenho para matrizes cujo todos seus autovalores são inteiros? Será mais rápida?

Para isso, vou utilizar como base o teorema de matrizes $A = QDQ^T$ serem diagonalizáveis e terem seus autovalores iguais à diagonal de D. Criei uma função para gerar matrizes de ordem n aleatórias com essa decomposição:

Código:

```
function [A, x] = Matriz_Aleat(n)
//Gera Matriz A aleatoria com autovalores x reais e inteiros.
//Esses autovalores sao o proprio D
Q = Matriz_Ortogonal_Aleat(n)
D = Matriz_Diagonal_Aleat(n)

A = Q*D*Q'
x = real(spec(A)) //ignora a parte imaginaria que aparece por erros de casa decimal

endfunction
```

```
function [A] = Matriz_Diagonal_Aleat(n)
A = zeros(n)
for i = 1:n
A(i,i) = floor(-((n*n-1)*rand(1,1,'uniform')) + ((n*n-1)*rand(1,1,'uniform')))
end
endfunction
```

```
function [A] = Matriz_Ortogonal_Aleat(n)

//para gerar um num. aleat. entre [a,b]
//  r = a + (b-a)*rand();

//matriz A com num. aleat. entre [-n^2 e n^2]
A = floor(-((n*n-1)*rand(n,n,'uniform')) + ((n*n-1)*rand(n,n,'uniform')))
A = orth(A)
endfunction
```

É gerada uma matriz ortogonal aleatória de ordem n, uma matriz diagonal aleatória de ordem n, e multiplicadas $A = QDQ^T$, para obter A.

Testando com o **Método da Potência**:

ORDEM 5

```
--> [A, x] = Matriz_Aleat(5)
 A  =

  -3.9395101  -2.2725729   2.4602586   7.096047   -2.1436141
  -2.2725729   4.3055999  -1.8330169  -5.8232084   2.1621087
   2.4602586  -1.8330169   1.3130269  -2.4107973   2.7205389
   7.096047   -5.8232084  -2.4107973   11.54926    4.3678664
  -2.1436141   2.1621087   2.7205389   4.3678664   10.771624
 x  =

  -9.0000000
   18.000000
   12.000000
  -1.0000000
   4.0000000
```

```
--> [lambda, xl, k, n_erro]=Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 100)

   0.0007946
 lambda  =

   17.999984
 xl  =

   0.3309228
  -0.4211184
   0.0133758
   1.
   0.3851938
 k  =

   27.
 n_erro  =

   0.0000072
```

```
--> [lambda, xl, k, n_erro]=Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 100)

   0.0005537
 lambda  =

   18.000000
 xl  =

  -0.2762110
   0.3514950
  -0.0111637
  -0.8346670
  -0.3215045
 k  =

   26.
 n_erro  =

   0.0000097
```

Exemplo 2:

```
--> [A, x] = Matriz_Aleat(5)
 A  =

   0.7439529   0.2518848  -0.0917114   0.1260699   2.0371005
   0.2518848   1.4666546  -0.2943487   1.1734218   0.3325278
  -0.0917114  -0.2943487   1.8811297   1.3190721   1.3136739
   0.1260699   1.1734218   1.3190721   2.1593999   1.9978546
   2.0371005   0.3325278   1.3136739   1.9978546   2.7488628
 x  =

   6.0000000
   2.0000000
  -1.0000000
   1.366D-16
   2.0000000
```

```
--> [lambda, xl, k, n_erro]=Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 100)

   0.0003913
 lambda  =

   5.9999700
 xl  =

   0.4101693
   0.2686591
   0.5480831
   0.8039818
   1.
 k  =

   12.
 n_erro  =

   0.0000094
```

```
--> [lambda, xl, k, n_erro]=Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 100)

   0.0002924
 lambda  =

   6.0000000
 xl  =

   0.2773442
   0.1816592
   0.3705973
   0.5436284
   0.6761700
 k  =

   12.
 n_erro  =

   0.0000084
```

ORDEM 10

```
--> [A, x] = Matriz_Aleat(10);

--> [lambda, xl, k, n_erro]=Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 1000);

   0.0036104

--> k
 k  =

   164.

--> [lambda, xl, k, n_erro]=Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 1000);

   0.0017655

--> k
 k  =

   160.



--> [A, x] = Matriz_Aleat(10);

--> [lambda, xl, k, n_erro]=Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 1000);

   0.0010643

--> k
 k  =

   37.

--> [lambda, xl, k, n_erro]=Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 1000);

   0.0006733

--> k
 k  =

   37.
```

ORDEM 100

```
--> [A, x] = Matriz_Aleat(100);

--> [lambda, xl, k, n_erro]=Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 500);

   0.0136692

--> k
 k  =

   303.

--> [lambda, xl, k, n_erro]=Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 500);

   0.00674

--> k
 k  =

   310.
```

Podemos concluir que o tempo de execução parece um pouco menor em matrizes menores, e houve uma melhora de desempenho na primeira versão.

TESTES - MÉTODO DA POTÊNCIA para AUTOVALORES IMAGINÁRIOS

Testando com matrizes completamente aleatórias:

ORDEM 5

```
--> A = floor(-((n^2)*rand(n,n,'uniform')) + ((n^2)*rand(n,n,'uniform')))
 A  =

  -15.    1.     0.   -17.   -9.
   13.   -6.     3.   -10.   -15.
   0.     22.   -9.   -4.     1.
  -19.    6.    -10.  -1.    -2.
   3.    -2.     14.   9.    -1.

--> [lambda, xl, k, n_erro]=Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 1000);

   0.0010129

--> lambda
 lambda  =

  -27.850468

--> k
 k  =

   42.
```

```
--> [lambda, x1, k, n_erro]=Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 1000);

   0.0008387

--> lambda
 lambda  =

  -27.850495

--> k
 k  =

   40.

--> spec(A)
 ans  =

  -27.850154 + 0.i
  -21.860415 + 0.i
   14.147628 + 0.i
   1.78147   + 13.30435i
   1.78147   - 13.30435i



--> A = floor(-((n^2)*rand(n,n,'uniform')) + ((n^2)*rand(n,n,'uniform')))
 A  =

  -4.   -5.    11.   4.   -7.
  -1.    2.    2.   -3.    7.
  -11.  -11.  -21.  -2.    7.
   6.    14.  -7.   -16.  -19.
   4.   -5.   -6.   -8.   -5.

--> [lambda, x1, k, n_erro]=Metodo_da_Potencial_v1(A, 'x0', 10^(-5), 1000);

   0.0008904

--> lambda
 lambda  =

  -22.082355

--> k
 k  =

   35.
```

```
--> [lambda, xl, k, n_erro]=Metodo_da_Potencial_v2(A, 'x0', 10^(-5), 1000);

   0.0004744

--> lambda
 lambda  =

  -22.082494

--> k
 k  =

   36.

--> spec(A)
 ans  =

  -22.082569 + 0.i
  -13.434452 + 8.0690519i
  -13.434452 - 8.0690519i
   2.4757369 + 9.4511434i
   2.4757369 - 9.4511434i
```

Podemos concluir que o lambda obtido em ambos testes corresponde ao autovalor que tem a maior parte real em módulo.

Desculpe pelo relatório enorme. ¨◡