FGV EMAp
ESCOLA DE MATEMÁTICA APLICADA

# Aula Prática 3

## Álgebra Linear Numérica

Gerardo Mikael Do Carmo Pereira

Professor: Antônio Carlos Saraiva Branco

RIO DE JANEIRO
2024

# Conteúdo

# 1 Introdução

O presente trabalho refere-se à aula prática 3. Este conjunto de tarefas práticas concentra-se na implementação e análise de dois algoritmos ambos funcionando com métodos iterativos. O primeiro feito em duas versões diferentes é usado para obter o autovalor dominante e o segundo obtem o autovalor mais próximo de um valor escalar "alfa"dado pelo usuário. Exploraremos diferentes cenários, buscando entender onde os algoritmos citados funcionam ou não, e tentaremos melhora-los se possível.

Ao longo dos exercícios teremos uma melhor compreensão dos métodos iterativos e suas aplicações.

# 2 Implementação

## 2.1 <Tarefa 1> Implementação do método da potência

Após a elaboração das funções de acordo com o enunciado da tarefa obtemos:

```
function [lambda, x1, k, n_error] = Metodo_potencia_1(A, x0, epsilon, M
    k = 0;

    //Usaremos x0 sobre sua maior coordenada, usaremos max para obte-la
    [max_coordenada_modulo_x0, pos_x0] = max(abs(x0));
    x0 = x0/max_coordenada_modulo_x0;
    x1 = A*x0; //Que nos retorna uma aproximação

    n_error = epsilon + 1;

    while (k <= M) & (n_error >= epsilon)
    //Atualizaremos lambda como a coordenada de maior modulo de x1
    [max_coordenada_modulo_x1, pos_x1] = max(abs(x1));
    lambda = max_coordenada_modulo_x1;

    x1 = x1/lambda;
    n_error = norm((x1 - x0), "inf");

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

Figura 1: Algoritmo iterativo para obteção do autovalor dominante usando o método da potência, Versão 1

```
function [lambda, x1, k, n_error] = Metodo_potencia_2(A, x0, epsilon, M)
    k = 0;

    //Usaremos x0 sobre sua maior coordenada, usaremos max para obte-la
    x0 = x0/norm(x0, 2);
    x1 = A*x0; //Que nos retorna uma aproximação do autovetor dominante

    n_error = epsilon + 1;

    while (k <= M) & (n_error >= epsilon)
        //Atualizaremos lambda como o escalar de x1 e x0
        lambda = (x1')*x0;

        if lambda<0 then
            x1 = -x1;
        end

        x1 = x1/norm(x1,2);
        n_error = norm(x1-x0,2);

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

Figura 2: Algoritmo iterativo para obteção do autovalor dominante usando o método da potência, Versão 2

O primeiro algoritmo implementado é o método da potência para a obtenção do autovalor dominante de uma matriz diagonalizável A. Ele itera sobre uma sequência de vetores $x_1$, $x_2$, $x_3$, ... , $x_n$ que ao longo das iterações convergem para o autovetor dominante correspondente ao autovalor dominante $\lambda$.

Na versão 1 do algoritmo, a iteração começa normalizando $x_0$ para que sua coordenada de maior módulo seja igual a 1. Isso garante que o vetor seja unitário. Então, a cada iteração, $x_1$ é calculado como $x_1 = A \times x_1$, seguido pelo cálculo do autovalor dominante $\lambda$ como a coordenada de maior módulo de $x_1$ O vetor $x_1$ é então normalizado e a diferença entre $x_1$ e $x_0$ é calculada como o critério de parada, caso o erro não seja menor que epislon o critério de parada será o número de máximo de iterações "M"dado pelo usuário.

Na versão 2, a principal diferença está na utilização do Quociente de Rayleigh para estimar $\lambda$ como $x_1^T \times A \times x_0$, onde $x_0$é normalizado pela norma-2. Se $\lambda$ é negativo, o sentido de $x_1$ é invertido. Isso é feito para garantir que a iteração converge para o autovetor dominante, independentemente do sinal do autovalor (isso é um diferencial dessa função no ). Novamente, o vetor $x_1$ é normalizado e a diferença entre $x_1$ e $x_0$ é calculada como o critério de parada, o segundo critério é novamente "M"dado pelo usuário.

Ambas as versões do algoritmo têm critérios de parada semelhantes, baseados na diferença entre os iterados consecutivos e em um limite máximo de iterações. No entanto, a maneira de calcular $\lambda$ e normalizar os vetores é diferente entre as versões.

## 2.2 <Tarefa 2> Implementação do Método da Potência Deslocada com Iteração Inversa

Após a elaboração das funções de acordo com o enunciado da tarefa obtemos:

```
function [lambda1,x1,k,n_erro] = Potencia_deslocada_inversa (A,x0,epsilon,alfa,M)
    // Inicialização de variáveis
    k = 0;
    n = size(A, 1); // Obtendo o tamanho da matriz A para fazer uma matriz identidade do tamanho correto
    x0 = x0 / norm(x0, 2);
    n_erro = epsilon + 1;

    // Loop principal
    while k <= M & n_erro >= epsilon
        // Resolvendo o sistema (A - alfa*I)*x1 = x0 usando a função Gaussian_Elimination_4
        [x1, C_, P_] = Gaussian_Elimination_4(A - alfa*eye(n), x0);
        x1 = x1 / norm(x1, 2);

        // Cálculo do quociente de Rayleigh
        lambda = x1' * A * x1;

        // Verificando o sinal do produto interno entre x1 e x0
        if x1' * x0 < 0
            x1 = -x1;
        end

        // Cálculo da norma do erro
        n_erro = norm(x1 - x0, 2);

        // Atualização das variáveis
        x0 = x1;
        k = k + 1;
    end

    lambda1 = lambda + alfa
endfunction
```

Figura 3: Algoritmo iterativo para obteção do autovalor mais próximo a alfa

O Método da Potência Deslocada com Iteração Inversa é uma variação do Método da Potência utilizado para encontrar o autovalor mais próximo de um valor específico, denominado "alfa", em uma matriz diagonalizável A.

A ideia principal é ajustar a matriz A subtraindo $\alpha$ vezes a identidade , de modo que os autovalores de $A - \alpha I$ sejam os autovalores de A deslocados por $\alpha$. Em seguida, o Método da Potência padrão é aplicado a $A - \alpha I$ para encontrar o autovalor mais próximo de $\alpha$.

Na função, a iteração começa com um vetor $x_0$ normalizado pela norma-2. A cada iteração, um sistema de equações lineares $(A - \alpha I) \times x_1 = x_0$ é resolvido para encontrar $x_1$. O vetor é então normalizado pela norma-2 e o autovalor é calculado usando o Quociente de Rayleigh. Se o produto interno de $x_1$ e $x_0$ for negativo, o sentido de $x_1$ é invertido para manter a consistência com $x_0$ de forma semelhante a segunda versão do algoritmo do método da potência. O critério de parada é baseado na norma-2 da diferença entre os vetores $x_1$ e $x_0$, ou no número máximo de iterações "M".

## 2.3 <Tarefa 3> Testes com o método da potência

Para testar a eficiencia das duas versões do método da potência levamos em consideração 3 matrizes A diferentes além de 3 vetores x0 distintos.

Matrizes utilizadas:

$$A1 = \begin{bmatrix} 5 & 1 & 2 \\ 1 & 4 & 3 \\ 2 & 3 & 6 \end{bmatrix}, A2 = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \textbf{ e } A3 = \begin{bmatrix} -3 & 1 & 1 \\ 1 & -2 & 1 \\ 1 & 1 & -2 \end{bmatrix}$$

Uma matriz simétrica com 3 autovalores distintos, uma matriz com os autovalores não todos distintos e uma matriz com autovalor dominante negativo. como podemos ver na imagem a seguir:

```
--> spec(A1)
 ans  =

   1.7853726
   3.9148004
   9.2998270

--> spec(A2)
 ans  =

   1.0000000
   1.0000000
   4.0000000

--> spec(A3)
 ans  =

  -3.7320508
  -3.0000000
  -0.2679492
```

Figura 4: Autovalores das 3 matrizes

Além disso temos 3 vetores iniciais diferentes: $x01 = [0.1; 0.1; 0.1]$, $x02 = [1; 1; 1]$ **e** $x03 = [100; 100; 100]$

Os valores responsáveis pelos critérios de parada estão fixados, com epsilon = 1e-6, e M = 100 como número máximo de iterações. Após adicionar na função uma chamada de tic() toc() para fazer a medição do tempo gasto no processo podemos começar os testes:

```
--> // Matrizes A1, A2 e A3 fornecidas

--> A1 = [5 1 2; 1 4 3; 2 3 6];

--> A2 = [2 1 1; 1 2 1; 1 1 2];

--> A3 = [-3 1 1; 1 -2 1; 1 1 -2];

-->

--> // Vetores x01, x02 e x03

--> x01 = [0.1; 0.1; 0.1];

--> x02 = [1; 1; 1];

--> x03 = [100; 100; 100];

-->

--> // Parâmetros de teste

--> epsilon = 1e-6; // Precisão

--> M = 100; // Número máximo de iterações
```

```
disp('Testando Matriz A1, Vetor x01:');
[lambda1, x11, k1, n_error1, clock1] = Metodo_potencia_1(A1, x01, epsilon, M);
disp('Testando Matriz A1, Vetor x02:');
[lambda2, x12, k2, n_error2, clock2] = Metodo_potencia_1(A1, x02, epsilon, M);
disp('Testando Matriz A1, Vetor x03:');
[lambda3, x13, k3, n_error3, clock3] = Metodo_potencia_1(A1, x03, epsilon, M);
disp('Testando Matriz A2, Vetor x01:');
[lambda4, x14, k4, n_error4, clock4] = Metodo_potencia_1(A2, x01, epsilon, M);
disp('Testando Matriz A2, Vetor x02:');
[lambda5, x15, k5, n_error5, clock5] = Metodo_potencia_1(A2, x02, epsilon, M);
disp('Testando Matriz A2, Vetor x03:');
[lambda6, x16, k6, n_error6, clock6] = Metodo_potencia_1(A2, x03, epsilon, M);
disp('Testando Matriz A3, Vetor x01:');
[lambda7, x17, k7, n_error7, clock7] = Metodo_potencia_1(A3, x01, epsilon, M);
disp('Testando Matriz A3, Vetor x02:');
[lambda8, x18, k8, n_error8, clock8] = Metodo_potencia_1(A3, x02, epsilon, M);
disp('Testando Matriz A3, Vetor x03:');
[lambda9, x19, k9, n_error9, clock9] = Metodo_potencia_1(A3, x03, epsilon, M);
```

```
disp('Testando Matriz A1, Vetor x01:');
[lambda1, x11, k1, n_error1, clock1] = Metodo_potencia_2(A1, x01, epsilon, M);
disp('Testando Matriz A1, Vetor x02:');
[lambda2, x12, k2, n_error2, clock2] = Metodo_potencia_2(A1, x02, epsilon, M);
disp('Testando Matriz A1, Vetor x03:');
[lambda3, x13, k3, n_error3, clock3] = Metodo_potencia_2(A1, x03, epsilon, M);
disp('Testando Matriz A2, Vetor x01:');
[lambda4, x14, k4, n_error4, clock4] = Metodo_potencia_2(A2, x01, epsilon, M);
disp('Testando Matriz A2, Vetor x02:');
[lambda5, x15, k5, n_error5, clock5] = Metodo_potencia_2(A2, x02, epsilon, M);
disp('Testando Matriz A2, Vetor x03:');
[lambda6, x16, k6, n_error6, clock6] = Metodo_potencia_2(A2, x03, epsilon, M);
disp('Testando Matriz A3, Vetor x01:');
[lambda7, x17, k7, n_error7, clock7] = Metodo_potencia_2(A3, x01, epsilon, M);
disp('Testando Matriz A3, Vetor x02:');
[lambda8, x18, k8, n_error8, clock8] = Metodo_potencia_2(A3, x02, epsilon, M);
disp('Testando Matriz A3, Vetor x03:');
[lambda9, x19, k9, n_error9, clock9] = Metodo_potencia_2(A3, x03, epsilon, M);
```

Figura 5: Definindo os parâmetros e fazendo os testes

| Teste | Autovalor Dominante | N de Iterações | Erro | Tempo Gasto (s) |
|---|---|---|---|---|
| Matriz A1, Vetor x01 | 9.2998284 | 15 | $6 \times 10^{-7}$ | 0.0004658 |
| Matriz A1, Vetor x02 | 9.2998284 | 15 | $6 \times 10^{-7}$ | 0.0003964 |
| Matriz A1, Vetor x03 | 9.2998284 | 15 | $6 \times 10^{-7}$ | 0.0002748 |
| Matriz A2, Vetor x01 | 4 | 1 | 0 | 0.0001015 |
| Matriz A2, Vetor x02 | 4 | 1 | 0 | 0.000127 |
| Matriz A2, Vetor x03 | 4 | 1 | 0 | 0.0001018 |
| Matriz A3, Vetor x01 | 3.7320508 | 101 | 2 | 0.001295 |
| Matriz A3, Vetor x02 | 3.7320508 | 101 | 2 | 0.0015631 |
| Matriz A3, Vetor x03 | 3.7320508 | 101 | 2 | 0.0014249 |

Tabela 1: Resultados dos testes do Método da Potência - Versão 1

| Teste | Autovalor Dominante | N de Iterações | Erro | Tempo Gasto (s) |
|---|---|---|---|---|
| Matriz A1, Vetor x01 | 9.2998270 | 14 | $9 \times 10^{-7}$ | 0.0002431 |
| Matriz A1, Vetor x02 | 9.2998270 | 14 | $9 \times 10^{-7}$ | 0.0002998 |
| Matriz A1, Vetor x03 | 9.2998270 | 14 | $9 \times 10^{-7}$ | 0.0002018 |
| Matriz A2, Vetor x01 | 4.0000000 | 1 | 0 | 0.0001050 |
| Matriz A2, Vetor x02 | 4.0000000 | 1 | 0 | 0.0001261 |
| Matriz A2, Vetor x03 | 4.0000000 | 1 | $1.923 \times 10^{-16}$ | 0.0000789 |
| Matriz A3, Vetor x01 | -3.7320508 | 7 | $9 \times 10^{-7}$ | 0.0001392 |
| Matriz A3, Vetor x02 | -3.7320508 | 7 | $9 \times 10^{-7}$ | 0.0002797 |
| Matriz A3, Vetor x03 | -3.7320508 | 7 | $9 \times 10^{-7}$ | 0.0001473 |

Tabela 2: Resultados dos testes do Método da Potência - Versão 2

Duas coisas se destacam na tabela de resultado:

- A matriz com autovalores não distintos convergiu em 1 iteração

Após testar em diferentes matrizes com características semelhantes (autovalores não distintos e matriz simétrica) e não obter o mesmo resultado de uma iteração algumas hipóteses foram levantadas sobre a causa dessa convergencia ultra rápida. Vetor inicial próximo do autovetor associado ao autovalor diferente, suspeitei que se o autovalor chute estiver próxima do autovetor associado ao autovalor diferente, o método da potência pode convergir rapidamente para esse autovalor. Isso ocorre porque a iteração do método da potência amplificará rapidamente a contribuição desse autovetor dominante. Porém com vetores x0 com valores muito grandes e muito pequenos o resultado foi o mesmo.

- A matriz com autovalores negativos não teve o autovalor correto encontrado pela versão1 da função o que estourou o limite do número de iteraçoes

A função potência, embora seja um método eficaz para encontrar o autovalor dominante de uma matriz, falha em encontrar autovalores negativos devido à sua natureza de maximizar o módulo da componente dominante do vetor resultante em cada iteração.

Quando uma matriz tem autovalores negativos, a iteração do método da potência

pode fazer com que o vetor converja para um múltiplo positivo do autovetor correspondente ao autovalor dominante em módulo, em vez de convergir para o autovetor correspondente ao autovalor negativo.

Para resolver esse problema modifiquei a função de forma que caso o maior autovalor em módulo seja negativo usaremos o negativo do valor encontrado como máximo, pois este se trata de um módulo.

A versão final do método da potência 1 que será anexada é a seguinte que conta como modificações o tic() toc() e a condição que garante seu funcionamento para matrizes com autovalores negativos:

```
function [lambda, x1, k, n_error, clock1] = Metodo_potencia_1(A, x0, epsilon, M)


    tic();
    k = 0;

    //Usaremos x0 sobre sua maior coordenada, usaremos max para obte-la
    [max_coordenada_modulo_x0, pos_x0] = max(abs(x0))

    if x0(pos_x0) < 0 then
        max_coordenada_modulo_x0 = - max_coordenada_modulo_x0;
    else
    end

    x0 = x0/max_coordenada_modulo_x0;
    x1 = A*x0; //Que nos retorna uma aproximação

    n_error = epsilon + 1;

    while (k <= M) & (n_error >= epsilon)
    //Atualizaremos lambda como a coordenada de maior modulo de x1
    [max_coordenada_modulo_x1, pos_x1] = max(abs(x1));
    lambda = max_coordenada_modulo_x1;

    x1 = x1/lambda;
    n_error = norm((x1 - x0), "inf");

    x0 = x1;
    x1 = A*x0;
    k = k+1;
    end
    clock1 = toc();

    disp("autovalor dominante");
    disp(lambda);
    disp("Número de iterações");
    disp(k);
    disp("erro");
    disp(n_error);
    disp("tempo gasto (em segundos):");
    disp(clock1);
endfunction
```

Figura 6: Versão final do algoritmo

## 2.4 <Tarefa 4> Discos de Gershgorin e potencia deslocada inversa

Para a execução da tarefa 4 foi criado o algoritmo CalcularAutovaloresGerschgorin-DesocadaInversa projetado para estimar os autovalores de uma matriz simétrica aleatória usando o método dos Discos de Gerschgorin e, em seguida, refiná-los usando o método da potência deslocada inversa.

O algoritmo começa gerando uma matriz aleatória simétrica A de tamanho n×n. A simetria garante que A possua apenas autovalores reais, simplificando o problema.Em

seguida, ele utiliza o método dos Discos de Gerschgorin para obter uma estimativa inicial dos autovalores da matriz A. Essas estimativas são armazenadas no vetor v0.

Para cada estimativa de autovalor em v0, o algoritmo aplica o método da potência deslocada inversa. Este método é uma técnica iterativa para encontrar um autovalor de uma matriz que está próximo de um valor desejado. Ele usa uma matriz deslocada $A - \lambda I$ onde lambda é a estimativa do autovalor, e então itera sobre uma sequência de vetores até convergir para o autovetor associado ao autovalor dominante.

Depois de calcular as suposições de autovalores com o método da potência deslocada inversa, o algoritmo compara essas suposições com os autovalores reais da matriz A. Isso é feito utilizando a função spec(A) que retorna os autovalores reais da matriz. As estimativas e os autovalores reais são então exibidos para comparação.

```
function [A, v0, aval_supos, aval_real] = Calcular_autovalores_gerschgorin_desocada_inversa(n)
    //Gerando uma matriz aleatória A simétrica
    A = MatrizSimetricaAleatoria(n);

    //Geramos estimativas dos autovalores usando os discos de Gerschgorin
    v0 = estimar_autovalores_gerschgorin(A);

    //Para cada estimativa, vamos usar a potência deslocada e descobrir o autovalor mais próximo
    x0 = rand(n, 1); //Vetor inicial aleatório
    epsilon = 1e-6; //Precisão desejada
    M = 100; //Número máximo de iterações

    aval_supos = zeros(n, 1);

    for i = 1:n
        [lambdal,x1,k,n_erro] = Potencia_deslocada_inversa(A,x0,epsilon,v0(i),M);
        aval_supos(i) = lambdal;
    end

    //Ao fim da iteração temos todas as suposições e compararemos então com os autovalores reais
    aval_real = spec(A);
    disp("Suposições de Autovalores:");
    disp(aval_supos);
    disp("Autovalores Reais:");
    disp(aval_real);
endfunction
```

Figura 7: Função criada para a tarefa 4

Ao realizar diversos testes verificou-se que os valores supostos são geralmente maiores que os reais, mas sempre se aproximam do maior autovalor.

```
--> [A, v0, a1, a2] = Calcular_autovalores_gerschgorin_desocada_inversa(6)

 "Suposições de Autovalores:"

  5.7474747
  6.5410547
  6.5275268
  6.2038414
  5.2274731
  5.7767837

 "Autovalores Reais:"

 -1.4841803
 -0.8592737
 -0.4037794
  0.0057431
  0.5142776
  6.0244760
```

Figura 8: Valores calculados e reais obtidos pela função

Uma possível explicação para isso seria que. Os discos de Gerschgorin fornecem estimativas conservadoras dos autovalores de uma matriz. Cada disco é centrado em

um elemento da diagonal principal da matriz e tem um raio igual à soma dos valores absolutos dos elementos fora da diagonal da linha ou coluna correspondente. Essas estimativas são conservadoras no sentido de que os autovalores reais estão contidos nos discos, mas os discos podem ser maiores do que o necessário. Portanto, os autovalores estimados usando os discos de Gerschgorin podem ser superestimados. E visto que usamos o método da potencia deslocada que é uma técnica iterativa para encontrar um autovalor próximo de um valor específico. Ele converge para o autovalor dominante mais próximo do deslocamento dado. No entanto, a convergência não é garantida para o autovalor correto em uma iteração finita. Portanto, mesmo que o método da potência deslocada inversa seja capaz de refinar as estimativas, ainda pode haver um desvio em relação aos autovalores reais.