CÁLCULO NUMÉRICO

Profa. Dra. Yara de Souza Tadano *yaratadano@utfpr.edu.br*

Aula 11

Matlab – Método de Gauss-Jacobi e Gauss-Seidel

MÉTODO DE GAUSS-JACOBI

- Vamos supor, por simplicidade, um sistema de 3 equações e 3 incógnitas:

$$E_1 : a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$E_2 : a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$E_3 : a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

MÉTODO DE GAUSS-JACOBI

- Se os elementos da diagonal forem todos **não-nulos**, é possível isolar x_1 em E_1; x_2 em E_2 e x_3 em E_3.

$$x_1 = \frac{b_1 - a_{12}\,x_2 - a_{13}\,x_3}{a_{11}}$$

$$x_2 = \frac{b_2 - a_{21}\,x_1 - a_{23}\,x_3}{a_{22}}$$

$$x_3 = \frac{b_3 - a_{31}\,x_1 - a_{32}\,x_2}{a_{33}}$$

Método de Gauss-Jacobi

- De forma equivalente, podemos escrever:

$$x_1 = \frac{b_1}{a_{11}} - \frac{a_{12}}{a_{11}} x_2 - \frac{a_{13}}{a_{11}} x_3$$

$$x_2 = \frac{b_2}{a_{22}} - \frac{a_{21}}{a_{22}} x_1 - \frac{a_{23}}{a_{22}} x_3$$

$$x_3 = \frac{b_3}{a_{33}} - \frac{a_{31}}{a_{33}} x_1 - \frac{a_{32}}{a_{33}} x_2$$

Método de Gauss-Jacobi

- Para implementar, escreveremos:

$$I = \begin{bmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \frac{b_3}{a_{33}} \end{bmatrix}; \qquad D = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & -\frac{a_{13}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & -\frac{a_{23}}{a_{22}} \\ -\frac{a_{31}}{a_{33}} & -\frac{a_{32}}{a_{33}} & 0 \end{bmatrix}$$

Método de Gauss-Jacobi

- Para implementar, escreveremos:

$$\begin{bmatrix} x_1^{(j)} \\ x_2^{(j)} \\ x_3^{(j)} \end{bmatrix} = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & -\frac{a_{13}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & -\frac{a_{23}}{a_{22}} \\ -\frac{a_{31}}{a_{33}} & -\frac{a_{32}}{a_{33}} & 0 \end{bmatrix} \times \begin{bmatrix} x_1^{(j-1)} \\ x_2^{(j-1)} \\ x_3^{(j-1)} \end{bmatrix} + \begin{bmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \frac{b_3}{a_{33}} \end{bmatrix}$$

Método de Gauss-Jacobi

- E teremos:

```
>> X1 = D * X + I
```

RESUMO DE COMANDOS

break	Interrompe a execução de laços `for` e `while`
clc	Limpa a tela (command window)
disp	Exibe o conteúdo de uma variável, sem mostrar o seu nome
input	Permite ao usuário inserir variáveis, textos, valores, etc
sign	Função sinal: retorna o sinal de um argumento
if	Condiciona execução de comandos
else	Usado com o comando `if`
elseif	Usado com o comando `if`
end	Usado para terminar a execução dos comandos `if`, `for`, `while`
while	Repete comandos enquanto condição especificada for verdadeira
fprintf	Grava dados em arquivo formatado
for	Repete comandos por um número de vezes especificado

FORMATOS DE DADOS DE SAÍDA

Specifier	Description
`%c`	Single character
`%d`	Decimal notation (signed)
`%e`	Exponential notation (using a lowercase `e` as in `3.1415e+00`)
`%E`	Exponential notation (using an uppercase `E` as in `3.1415E+00`)
`%f`	Fixed-point notation
`%g`	The more compact of `%e` or `%f`, as defined in [2]. Insignificant zeros do not print.
`%G`	Same as `%g`, but using an uppercase `E`
`%i`	Decimal notation (signed)
`%o`	Octal notation (unsigned)
`%s`	String of characters
`%u`	Decimal notation (unsigned)
`%x`	Hexadecimal notation (using lowercase letters `a-f`)
`%X`	Hexadecimal notation (using uppercase letters `A-F`)

COMANDOS

Character	Description
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\\	Backslash
\'' or '' (two single quotes)	Single quotation mark
%%	Percent character

Vetores e Matrizes

- Colchetes são utilizados para armazenar vetores ou matrizes.

- Vetor linha:

```
>> a = [1 2 3 4]
```

- Vetor coluna:

```
>> a = [1; 2; 3; 4]
```
ou

```
>> a=[1 2 3 4]'
```
ou ainda:

```
>> a = [1
        2
        3
        4]
```

MATRIZ

- Uma matriz pode ser armazenada por:

```
>> A=[a11 a12 a13; a21 a22 a23; a31 a32 a33];
```

- Ou por:

```
>> A=[a11 a12 a13
      a21 a22 a23
      a31 a32 a33];
```

- Ou ainda por:

```
>> A=[[a11 a12 a13]'[a21 a22 a23]'[a31 a32 a33]'];
```

MATRIZ

- Podemos acessar cada valor de uma matriz utilizando a estrutura:

```
>> A(i,j)
```

- Por exemplo:

```
>> A(1,1) = a11
```

Símbolo :

- O : é muito utilizado para manipular matrizes. Se um : for colocado entre dois números, o MATLAB irá gerar números entre eles adicionando um.

```
>> i = 1:5
```

i será igual a 1, depois 2, depois 3, depois 4 e por último 5.

Variações interessantes seriam:

```
>> i = 1:0.5:3
```

Isto resulta em: i = 1.0 1.5 2.0 2.5 3.0

```
>> i = 5:-1:1
```

Isto resulta em: i = 5 4 3 2 1

Algoritmo do método de Gauss-Jacobi

ENTRADA: A (matriz $n \times n$ com $a_{jj} \neq 0$, $j = 1, ..., n$), $\mathbf{b}$, aproximação inicial $\boldsymbol{x}^{(0)}$, precisão *tol*, número máximo de iterações *max*.

SAÍDA: solução aproximada $\boldsymbol{x}^{(m)} = [x_j^{(m)}]$ ou mensagem de falha.

Passo 1: Para $i = 1 : n$ (contador das linhas da matriz)

Passo 2: Para $j = 1 : n$ (contador das colunas da matriz)

Passo 3: Se i = j

D(i, j) = 0

I (i, 1) = B(i, 1) / A(i, i);

senão

D(i, j) = -A(i, j) / A(i, i);

Algoritmo do método de Gauss-Jacobi

Fim dos Passos 1, 2 e 3

Passo 4: Enquanto $k < nmax$

X1 = D * X + I;

Passo 5: Para i = 1: n

ERx(*i*, *1*) = abs((X1(*i*, *1*) – X(*i*, *1*))/ X1(*i*, *1*))*100;

mtol(i,1) = tol;

Fim Passo 5

Passo 6: Se ERx < mtol

SAÍDA (O vetor solução é: X1)

Método de Gauss-Jacobi

- **VARIÁVEIS**

- Ordem da matriz: `ord`
- Matriz dos coeficientes: A
- Vetor X inicial: X
- Vetor dos termos independentes: B
- Precisão: `tol`
- Número máximo de iterações: `max`

Implementação

```
clear, clc
% Recebe a ordem do sistema
      ord = input('Ordem da matriz:');
% Recebe a matriz dos coeficientes
      A = input('matriz dos coeficientes:');
% Recebe o vetor X inicial
      X = input('Entre com o vetor X inicial:');
% Recebe o vetor dos termos independentes b
      B = input('Entre com o vetor B:');
% Recebe o erro pré-estabelecido
      tol = input('Entre com a tolerância:');
% Recebe o vetor dos termos independentes b
      max = input('Entre com o número máximo de
                  iterações');
```

Implementação

```
% Processamento
k = 1;
for i = 1:ord
    for j = 1:ord
        if i == j
            % Matriz que armazena os termos que
                multiplicam por xi
            D(i,j) = 0;
            % Matriz que armazena os termos somam
              com D(i,j)
            I(i,1) = B(i,1)/A(i,i);
        else
            D(i,j) =-A(i,j)/A(i,i);
        end
    end
end
```

Implementação

```
While (k < max)
    X1 = D*X + I;
  for i = 1:ord


    ERX(i,1)= abs((X1(i,1)-X(i,1))/X1(i,1))*100;
    mtol(i,1)=tol;
  end
    if ERX < mtol
        fprintf('%1.5f\n',X1);
        break
    end
    X = X1;
    k = k+1;
end
```

Exemplo 1

- Considere o sistema:

$$\begin{cases} 3x_1 - 0{,}1x_2 - 0{,}2x_3 = 7{,}85 \\ 0{,}1x_1 + 7x_2 - 0{,}3x_3 = -19{,}3 \\ 0{,}3x_1 - 0{,}2x_2 + 10x_3 = 71{,}4 \end{cases}$$

- A solução verdadeira é $x_1 = 3,\ x_2 = -2{,}5,\ x_3 = 7$

- Use o **Método de Gauss-Jacobi** para obter a solução aproximada com $e_s = 0{,}01\%$

Exemplo 1

- A tabela abaixo apresenta os valores de x_1, x_2 e x_3 a cada iteração.

Iteração (j)	$x_1^{(j)}$	$x_2^{(j)}$	$x_3^{(j)}$
1	2,61667	-2,75714	7,14000
2	3,00076	-2,48852	7,00636
3	3,00081	-2,49974	7,00021
4	3,00002	-2,50000	6,99998
5	3,00000	-2,50000	7,00000

Algoritmo do método de Gauss-Seidel

ENTRADA: A (matriz $n \times n$ com $a_{jj} \neq 0$, $j = 1, ..., n$), **b**, aproximação inicial $\boldsymbol{x}^{(0)}$, precisão *tol*, número máximo de iterações *max*.

SAÍDA: solução aproximada $\boldsymbol{x}^{(m)} = [x_j^{(m)}]$ ou mensagem de falha.

Passo 1: Para $i = 1 : n$ (contador das linhas da matriz)

Passo 2: Para $j = 1 : n$ (contador das colunas da matriz)

Passo 3: Se i = j

D(i, j) = 0

I (i, 1) = B(i, 1) / A(i, i);

senão

D(i, j) = -A(i, j) / A(i, i);

Algoritmo do método de Gauss-Seidel

Fim dos Passos 1, 2 e 3

Passo 4: Enquanto $k < nmax$

X1 = X

Para i = 1:n

X1 = D * X1 + I;

Passo 5: Para i = 1: n

ERx(*i*, *1*) = abs((X1(*i*, *1*) – X(*i*, *1*))/ X1(*i*, *1*))*100;

mtol(i,1) = tol;

Fim Passo 5

Passo 6: Se ERx < mtol

SAÍDA (O vetor solução é: X1)

Método de Gauss-Seidel

- **VARIÁVEIS**

- Ordem da matriz: `ord`
- Matriz dos coeficientes: A
- Vetor X inicial: X
- Vetor dos termos independentes: B
- Precisão: `tol`
- Número máximo de iterações: `max`

Implementação

```
clear, clc
% Recebe a ordem do sistema
      ord = input('Ordem da matriz:');
% Recebe a matriz dos coeficientes
      A = input('matriz dos coeficientes:');
% Recebe o vetor X inicial
      X = input('Entre com o vetor X inicial:');
% Recebe o vetor dos termos independentes b
      B = input('Entre com o vetor B:');
% Recebe o erro pré-estabelecido
      tol = input('Entre com a tolerância:');
% Recebe o vetor dos termos independentes b
      max = input('Entre com o número máximo de
                  iterações');
```

Implementação

```
% Processamento
k = 1;
for i = 1:ord
    for j = 1:ord
        if i == j
            % Matriz que armazena os termos que
                multiplicam por xi
            D(i,j) = 0;
            % Matriz que armazena os termos somam
              com D(i,j)
            I(i,1) = B(i,1)/A(i,i);
        else
            D(i,j) =-A(i,j)/A(i,i);
        end
    end
end
```

Implementação

```
While (k < max)
    X1 = X;
  for i = 1:ord


    X1(i,1) = D(i,:)*X1 + I(i,1);
      (Obs. A expressão D(i,:)*X1 realiza uma
       multiplicação entre dois vetores
       (uma linha da matriz e o vetor X))
  end
```

Implementação

```
    for i = 1:ord

        ERX(i,1)= abs((X1(i,1)-X(i,1))/X1(i,1))*100;
        mtol(i,1)=tol;
    end
        if ERX < mtol
            fprintf('%1.5f\n',X1);
            break
        end
        X = X1;
        k = k+1;

end
```