

Introdução ao MATLAB

Davi Marco Lyra Leite (davi@ieee.org)
Student Member (90700736)
Events Commission



Ramo IEEE UnB Biênio 2011/2012:



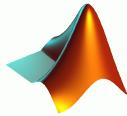
- Chair: Paula Freitas (Mecatrônica).
- Vice Chair: Lucas Pessoa (Computação).
- Tesoureiro: Arthur Amaral (Elétrica).
- Conselheiro Prof. Rafael Shayani

www.ieee.org/go/unb
sb.ieee.unb@ieee.org IEEE
Advancing Technology for Humanity

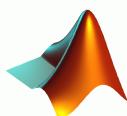
Curso de MATLAB

Conteúdo Programático

1. Introdução ao ambiente MATLAB
2. Operações com Matrizes
3. Manipulação de Vetores e Matrizes
4. Gráficos
5. Dados em MATLAB
6. Arquivos *.m
7. Controle de fluxo (laços e repetições)
8. Arquivos de funções
9. Gráficos tridimensionais



Curso de MATLAB



Conteúdo Programático

10. Funções relacionadas a estatística
11. Aplicações em processamento de sinais
12. Polinômios
13. Cálculo numérico
14. Matemática simbólica



Unidade 1

Introdução ao Ambiente Matlab



Sobre o MATLAB



- Software de análise numérica e computacional;
- Shareware (com versões disponíveis para pesquisa e estudantes);
- Tooboxes de aplicações específicas:
 - Image and signal processing,
 - Robotics
 - Aeronautics;
- Uso na indústria e academia;



Sobre o MATLAB



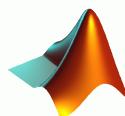
- MATLAB = Matrix Laboratory;
- Ente básico: matrizes (na verdade são tensores);
- Programação interpretativa: os códigos escritos não são diretamente compilados e sim interpretados pelo sistema;



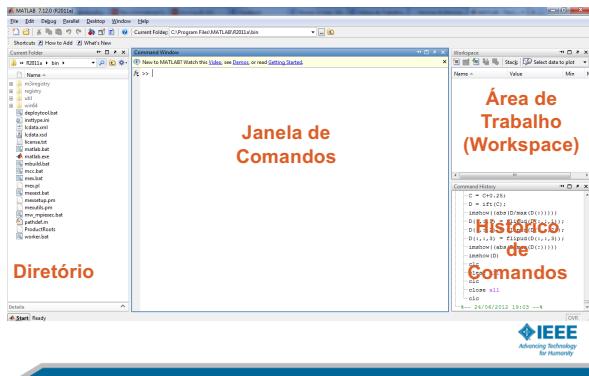
Carregando o MATLAB



- Procure o ícone na área de trabalho (algo parecido com a figura ao lado);
- Dê um duplo clique nele;
- Espere carregar;



1. Ambiente MATLAB



1. Ambiente MATLAB

- **Janela de Comandos:**
 - Onde são digitadas as operações a se realizar;
 - Apresenta sempre o sinal “>>”;
- **Histórico de Comandos:**
 - Mostra os comandos realizados;
- **Área de trabalho:**
 - Apresenta as variáveis inseridas no sistema;
- **Diretório:**
 - Indica os arquivos existentes no diretório em que se está trabalhando;

1. Ambiente MATLAB

- O seu ambiente de trabalho pode ser configurado para apresentar apenas os elementos mais úteis ao seu uso diário;
- Em qualquer visualização estará presente ao menos a janela de comandos (aquela em que aparece o sinal: “>>”)

1. Ambiente MATLAB

- Característica das entradas em MATLAB:
 - São *case sensitives* (diferem maiúsculas de minúsculas);
 - Elemento básico: matrizes (M por N)!
 - Vetores: matrizes M por 1 ou 1 por N;
 - Escalares: matrizes 1 por 1;
- Dados podem ser adicionados na janela de comandos, por comandos e funções, por arquivos *.m ou importados do ambiente externo;



1. Ambiente MATLAB

- Os dados que compõem as matrizes podem ser:
 - Números reais (parte inteira separada da parte decimal por “.”);
 - Números complexos;
 - Expressões matemáticas;
 - Strings (caracteres);



1.1 Entrando com Matrizes

- Pode-se entrar com uma matriz simples a partir da janela de comandos, para tanto usa-se uma lista explícita dos elementos
 - Elementos de uma mesma linha são separados por espaços em branco ou por vírgulas “,” e a quebra de linha é dada por “;” (ponto-e-vírgula)
 - A matriz é iniciada e finalizada por colchetes []



1.1 Entrando com Matrizes

Exemplo: `>> A = [1 2 3; 4 5 6; 7 8 9]`

Que é igual a:

`>> A = [1,2,3;4,5,6;7,8,9]`



1.1 Entrando com Matrizes

- Pressionando-se ENTER tem-se o resultado na tela;
- A matriz é salva na memória do computador e adicionada ao *Workspace*;

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
 1   2   3
 4   5   6
 7   8   9

>> A = [1, 2, 3; 4, 5, 6; 7, 8, 9]
A =
 1   2   3
 4   5   6
 7   8   9
```



1.1 Entrando com Matrizes

- Uma outra forma é entrar com a matriz linha por linha:

•Por exemplo:

`>> B = [1 2 3 4
5 6 7 8
9 10 11 12];`

```
B =
 1   2   3   4
 5   6   7   8
 9  10  11  12
```

`>> |`



1.1 Entrando com Matrizes

- Além disso, as matrizes podem ser geradas por arquivos *.m que contenham os comandos a elas relacionados
 - Para isso, basta digitar o nome do arquivo e dar ENTER
 - >> teste
- Elas também podem ser carregadas a partir de arquivos que armazenam dados ASCII usando o comando *load*
 - >> load teste.mat



1.1 Entrando com Matrizes

- Carregando o arquivo teste.mat (usando o comando *load*) devem aparecer as matrizes C e D na área de trabalho (*Workspace*);
- O comando teste (pelo arquivo teste.m) deve apresentar na tela a matriz E;



1.2 Elementos das Matrizes

- As entradas das matrizes podem ser quaisquer expressões aceitas em MATLAB e que tenham tamanho unitário;
 - Exemplo:
- ```
>> F = [-157.95 sqrt(23) (((1 + 4.7)^3)/37)*13;
 pi exp(1) 123;
 52.19 47 + 15i 222]
```



## 1.2 Elementos das Matrizes

- Um elemento individual da matriz pode ser referenciado usando-se o nome da matriz e índice do elemento entre parênteses, ()

Exemplo:  $A(1,3) = 3$ ;  $A(2,2) = 5$ ;



## 1.2 Elementos das Matrizes

- Pode-se adicionar elementos a uma matriz ou vetor indicando-se o índice desejado para o dado e o seu valor

Exemplo:  $A(2,5) = 57$ ;

```
>> A(2,5) = 57
A =
 1 2 3 0 0
 4 5 6 0 57
 7 8 9 0 0
>> |
```



## 1.2 Elementos das Matrizes

- Quando da adição de elementos cujos índices extrapolam o tamanho da matriz original, a dimensão da matriz é aumentada adicionando-se zeros às entradas não especificadas;

```
>> A(2,5) = 57
A =
 1 2 3 0 0
 4 5 6 0 57
 7 8 9 0 0
>> |
```



## 1.2 Elementos das Matrizes

- Além disso, pode-se construir uma matriz grande a partir da concatenação de matrizes menores;
- Os novos dados podem entrar como linhas, colunas ou conjunto de elementos na matriz anterior, formando uma nova matriz maior;

```
>> a = [10 11 12];
>> G = [A ; a]
```

G =

|    |    |    |
|----|----|----|
| 1  | 2  | 3  |
| 4  | 5  | 6  |
| 7  | 8  | 9  |
| 10 | 11 | 12 |

>> |



## 1.2 Elementos das Matrizes

- Note que o vetor "a" foi adicionado como linha à matriz "A" através da concatenação;
- Pode-se concatenar também várias matrizes gerando uma outra maior;
- Exemplo:

```
>> b = [1 2 3; 4 5 6; 7 8 9];
>> H = [B b]
```

H =

|   |    |    |    |   |   |   |
|---|----|----|----|---|---|---|
| 1 | 2  | 3  | 4  | 1 | 2 | 3 |
| 5 | 6  | 7  | 8  | 4 | 5 | 6 |
| 9 | 10 | 11 | 12 | 7 | 8 | 9 |

>>



## 1.2 Elementos das Matrizes

- Também é possível se retirar matrizes menores de uma matriz grande utilizando-se uma lista de índices;
- Essa lista é feita utilizando-se ":" (dois pontos) e pode ser feita de duas formas:
  - Selecionando-se elementos num intervalo específico;
  - Selecionando-se elementos por omissão;



## 1.2 Elementos das Matrizes

- $I = A(1:2,1:2)$  – pegam-se as duas primeiras linhas e as duas primeiras colunas da matriz original A;

```
>> I = A(1:2,1:2)
```

I =

|   |   |
|---|---|
| 1 | 2 |
| 4 | 5 |

>>



## 1.2 Elementos das Matrizes

- $J = H(:,1:3)$  – pegam-se as três primeiras colunas e todas as linhas (índices omitidos para as linhas) da matriz original H;

```
>> J = H(:,1:3)
```

J =

|   |    |    |
|---|----|----|
| 1 | 2  | 3  |
| 5 | 6  | 7  |
| 9 | 10 | 11 |

>> |



## 1.3 Declarações e Variáveis

- Para declarar uma variável, basta fazer:

```
>> variável = expressão
```

- Ou, simplesmente:

```
>> expressão
```

o que resultará na atribuição do resultado à variável padrão "ans"



## 1.3 Declarações e Variáveis

- As expressões podem ser compostas por valores numéricos e operações matemáticas;
- O MATLAB efetua o cálculo da expressão e salva o resultado nas matrizes que serão mostradas na tela;
- Caso seja posicionado o sinal ";" no fim da expressão, o resultado não é impresso na tela, mas mesmo assim é salvo na memória;



## 1.3 Declarações e Variáveis

- Expressões muito grandes podem ser quebradas em várias linhas colocando-se " ..." (um espaço em branco seguido por três pontos) após o último valor da linha desejada;
- Exemplo:

```
>> s = 1 + 2 + 3 + 4 + 5 + 6 + 7 ...
+ 8 + 9 + 10 + 11 + 12 + 13
s =
91
>>
```



## 1.3 Declarações e Variáveis

- Os espaços entre os números e os sinais das operações matemáticas são opcionais, todavia o espaço em branco antes dos três pontos é obrigatório;
- As variáveis e funções são descritas por conjuntos de letras ou conjuntos de letras e números – em que os 63 primeiros elementos são identificados.



## 1.4 Números e Operações Aritméticas

- Notação decimal é feita usando-se o ponto “.” – visto que o programa segue a notação para o sistema decimal adotada nos EUA;
- O sinal de menos “-” é utilizado para expressar valores negativos (pode estar separado ou não do número a que se refere com um espaço em branco);

```
>> 15.37
ans =
15.3700
>> - 12
ans =
-12
>> -13
ans =
-13
>>
```



## 1.4 Números e Operações Aritméticas

- Potências de 10 podem ser incluídas usando-se os sufixos en ou En ( $10^n = en = En$ ), como na calculadora HP 50G.

```
>> 15*10^2
ans =
1500
>> 15e2
ans =
1500
>> 1SE2
ans =
1500
```



## 1.4 Números e Operações Aritméticas

- Operações aritméticas usuais:

|      |          |               |       |
|------|----------|---------------|-------|
| i.   | <b>+</b> | Adição        | 3     |
| ii.  | <b>-</b> | Subtração     | ans = |
| iii. | <b>*</b> | Multiplicação | -1    |
| iv.  | <b>^</b> | Exponenciação | ans = |

```
>> 1 + 2, 3 - 4, 2*3, 5^2
ans =
3
ans =
-1
ans =
6
ans =
25
```



## 1.4 Números e Operações Aritméticas

- Operações aritméticas usuais:

v. /      Divisão à direita  
 vi. \      Divisão à esquerda (menos usual)

```
>> 15/3
```

```
ans =
```

```
5
```

```
>> 15\3
```

```
ans =
```

```
0.2000
```



## 1.4 Números e Operações Aritméticas

•Números complexos podem ser utilizados em qualquer operação envolvendo o MATLAB;

•Eles podem ser descritos por uma soma de parte real com imaginária (em que “i” ou “j” podem ser usados para representar a unidade imaginária):

```
>> z = a + b*i
```

Ou:

```
>> z = a + b*j
```



## 1.4 Números e Operações Aritméticas

•Um número complexo pode ser ainda declarado usando-se a função *complex* da seguinte forma:

```
>> z = complex(a,b)
```

em que “a” será a parte real e “b” a parte imaginária;



## 1.4 Números e Operações Aritméticas

• Além disso, é possível se utilizar a notação de Euler e declarar o número complexo pelo seu módulo e sua fase.

• Exemplo:

```
>> rho = 5; theta = pi/4;
>> w = rho*exp(i*theta)
```



## 1.4 Números e Operações Aritméticas

• Por fim, uma matriz complexa pode ser declarada de várias formas, sendo duas delas apresentadas a seguir:

e  

$$\begin{aligned} &>> K = [1 \ 2; 3 \ 4] + i*[5 \ 6; 7 \ 8]; \\ &>> K = [1 + 5*i \ 2 + 6*i; 3 + 7*i \ 4 + 8*i]; \end{aligned}$$

• Nesse segundo caso é possível se omitir o sinal “\*” antes de “i”.



## 1.5 Exercícios

1. Escreva um vetor ( $V$ ) que contenha os números de 12 a 18;
2. Escreva uma matriz  $4 \times 5$  que contenha os primeiros 20 múltiplos de 2 em sequência ( $L(1,1) = 2$ ,  $L(1,2) = 4$ , ...,  $L(4,5) = 40$ );
3. Realize os cálculos:
  - $z = (12 + 47 + \pi)/3.96$
  - $t = \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64}$
4. Faça  $W = V + 2$ . O que acontece?
5. Escreva  $M = \text{ones}(4,5)$ . Como é essa matriz?
  - Faça  $M = M^5$ . O que acontece?
6. Com o resultado de (5), proponha outra solução para (2).



# Unidade 2

## Operações com Matrizes



### 2. Operações com Matrizes

•São várias as operações que podem ser realizadas com Matrizes em MATLAB, entre elas:

- i. Transposta;
- ii. Adição e Subtração;
- iii. Multiplicação (por escalar e matricial);
- iv. Divisão à direita;
- v. Divisão à esquerda;
- vi. Exponenciação;
- vii. Operações com conjuntos.



#### 2.1 Transposta

•A transposta de uma matriz significa trocar as linhas pelas colunas, ou seja, o que era linha 1 vira coluna 1, o que era linha 2 vira coluna 2 e assim por diante;

•Para um vetor, a transposta consiste em transformar um vetor linha em um vetor coluna;

•Para um escalar a transposição nada significa.



## 2.1 Transposta

- A transposta é feita colocando-se um apóstrofo “ ‘ ” à direita da matriz/vetor.

**Exemplo:**

```
>> B, B'
```

B =

|   |    |    |    |
|---|----|----|----|
| 1 | 2  | 3  | 4  |
| 5 | 6  | 7  | 8  |
| 9 | 10 | 11 | 12 |

ans =

|   |   |    |
|---|---|----|
| 1 | 5 | 9  |
| 2 | 6 | 10 |
| 3 | 7 | 11 |
| 4 | 8 | 12 |



## 2.1 Transposta

- Para matrizes complexas o sinal “ ‘ ” dará o conjugado complexo transposto;

- Para se obter apenas a transposta deve-se proceder colocando-se um ponto antes do apóstrofo: K1 = K.’

**Exemplo:**

```
>> K, K.'
```

K =

|                  |                  |
|------------------|------------------|
| 1.0000 + 5.0000i | 2.0000 + 4.0000i |
| 3.0000 + 7.0000i | 4.0000 + 8.0000i |

ans =

|                  |                  |
|------------------|------------------|
| 1.0000 + 5.0000i | 3.0000 + 7.0000i |
| 2.0000 + 6.0000i | 4.0000 + 8.0000i |

>> K'

ans =

|                  |                  |
|------------------|------------------|
| 1.0000 + 5.0000i | 3.0000 - 7.0000i |
| 2.0000 - 6.0000i | 4.0000 - 8.0000i |

## 2.2 Adição e Subtração

- Existem dois tipos de adição e subtração para matrizes em MATLAB. O primeiro está apresentado no exercício (4), em que se realiza a adição de um escalar à matriz e o resultado é a matriz com todos os valores incrementados pelo valor adicionado.

```
>> A1 = A + 5
```

A1 =

|    |    |    |
|----|----|----|
| 6  | 7  | 8  |
| 9  | 10 | 11 |
| 12 | 13 | 14 |



## 2.2 Adição e Subtração

- O segundo caso é a adição matricial propriamente dita. Todavia, deve-se respeitar a dimensão das parcelas – uma matriz  $M \times N$  só pode ser somada com uma matriz  $M \times N$ .

```
>> L = B + C
```

L =

|    |    |    |    |
|----|----|----|----|
| 2  | 4  | 6  | 8  |
| 10 | 12 | 14 | 16 |
| 18 | 20 | 22 | 24 |



## 2.2 Adição e Subtração

- Caso contrário aparecerá a seguinte mensagem:

*??? Error using ==> plus  
Matrix dimensions must agree.*



## 2.3 Multiplicação

- Para o caso da multiplicação existem 4 formas de se realizar essa operação:

- multiplicação por escalar;
- multiplicação matricial;
- produto escalar;
- produto vetorial.



## 2.3 Multiplicação

(a) A multiplicação por escalar segue o mesmo princípio da adição por escalar: todos os valores da matriz são multiplicados pelo escalar (como no exercício (5)).

•Exemplo:

```
>> A2 = 2*A
```

```
A2 =
```

|    |    |    |
|----|----|----|
| 2  | 4  | 6  |
| 8  | 10 | 12 |
| 14 | 16 | 18 |



## 2.3 Multiplicação

(b) A multiplicação matricial tem uma regra a mais a ser seguida: a segunda dimensão da primeira matriz deve ser igual à primeira dimensão da segunda matriz – se A é  $M \times N$  B deve ser  $N \times T$  para se multiplicar por A, gerando C que será  $M \times T$ .

•Vale-se ressaltar que essa multiplicação não é comutativa!!



## 2.3 Multiplicação

•Exemplo:

```
>> M = B*ones(4,5)
```

```
M =
```

|    |    |    |    |    |
|----|----|----|----|----|
| 10 | 10 | 10 | 10 | 10 |
| 26 | 26 | 26 | 26 | 26 |
| 42 | 42 | 42 | 42 | 42 |

•Caso isso não seja seguido, aparecerá o seguinte erro:

*??? Error using ==> mtimes*

*Inner matrix dimensions must agree.*



## 2.3 Multiplicação

(c) O produto interno entre dois vetores, indicado em notação por  $\langle x, y \rangle$  ou por  $x \cdot y$  e equivalente ao somatório dos produtos dos componentes de mesmo índice dos vetores, é dado por  $z = \text{dot}(x, y)$ .

• Exemplo:

```
>> x = [1 2 3];
>> y = [4 5 6];
>> z = dot(x, y)
```

$z =$

32



## 2.3 Multiplicação

(d) O produto vetorial entre dois vetores, indicado em notação por  $x \times y$ , é dado em MATLAB por  $t = \text{cross}(x, y)$ .

• Exemplo:

```
>> t = cross(x, y)
```

$t =$

-3      6      -3



## 2.4 Divisão

- A divisão de matrizes corresponde à multiplicação de um lado ou de outro pela matriz inversa daquela que é considerada o divisor.
- Existe, portanto, a diferença entre divisão à esquerda e divisão à direita.



## 2.4 Divisão

- A divisão à direita ( $B/A$ ) corresponde à multiplicação à direita pela inversa ( $B * \text{inv}(A)$ );
- Analogamente, a divisão à esquerda corresponde à multiplicação à esquerda pela inversa ( $\text{inv}(A) * B$ );



## 2.4 Divisão

- Exemplo de divisão à direita:

```
>> N, O
N =
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
O =
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
>> P = N\O
Warning: Matrix is close to singular or badly scaled.
 Results may be inaccurate. RCOND = 1.093816e-018.
P =
1.5833 -2.0000 -1.2500 1.6667
1.5833 -2.0000 -1.2500 1.6667
1.5833 -2.0000 -1.2500 1.6667
1.5833 -2.0000 -1.2500 1.6667
```



## 2.4 Divisão

- Exemplo de divisão à esquerda:

```
>> P = O\N
Warning: Matrix is close to singular or badly scaled.
 Results may be inaccurate. RCOND = 1.387779e-018.
P =
-1.2500 -1.2500 -1.2500 -1.2500
 0.5000 0.5000 0.5000 0.5000
 0.7500 0.7500 0.7500 0.7500
 0 0 0 0
```



## 2.5 Exponenciação

- A exponenciação para matrizes é válida apenas para matrizes quadradas – devido à questão do dimensionamento das matrizes. Assim, sendo A uma matriz quadrada,  $A^p$  indica A elevado à pésima potência (ou seja, são feitas repetidas multiplicações de A por A).



## 2.5 Exponenciação

• Exemplo:

```
>> N
N =
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
>> N^5
ans =
8192 8192 8192 8192
8192 8192 8192 8192
8192 8192 8192 8192
8192 8192 8192 8192
```



## 2.6 Operações com Conjuntos

- Esse termo se refere às operações realizadas elemento a elemento (para valores que ocupam a mesma posição em matrizes distintas);
- Utiliza-se a mesma notação das operações normais, todavia precedida por um ponto: “.”\*, “./”, “.\” e “.^”.



## 2.6 Operações com Conjuntos

**•Exemplos:**

```
>> B, C, B.*C
ans =
B =
1 2 3 4
5 6 7 8
9 10 11 12
>> D./B
ans =
13.0000 7.0000 5.0000 4.0000
3.4000 3.0000 2.7143 2.5000
2.3333 2.2000 2.0909 2.0000
>> D.\B
ans =
0.0769 0.1429 0.2000 0.2500
0.2941 0.3333 0.3684 0.4000
0.4286 0.4545 0.4783 0.5000
>> E.^2
ans =
1 4 9 16
25 36 49 64
81 100 121 144
```



## 2.7 Exercícios

1. Faça  $A + 5 - 2 - 2*B(1:3,1:3) - 7$ .
2. Calcule  $A*B$ .
3. É possível fazer alguma multiplicação por A tendo a matriz B à esquerda?
4. Calcule  $B.*E$  e  $x*B$ .
5. Calcule  $x^y$ ,  $y^x$ . Qual a semelhança desse resultado? O que essa operação significa?
6. Calcule  $x^y$ . Explique o resultado.
7. Escreva um vetor  $(3,1)$  e calcule o seu produto vetorial por x e por y. O que acontece? Explique.



## 2.7 Exercícios

8. É possível se fazer  $B/D$ ? E  $B\backslash D$ ? Qual a diferença entre essas operações?
9. Calcule  $A^3$  e  $A.^3$ . Explique os diferentes resultados.
10. Por que não é possível fazer  $B^2$ ?
11. Faça  $x_1 = 2*x$  e  $y_1 = 3*y$ . O que acontece com o produto escalar de  $x_1$  e  $y_1$  se comparado ao produto escalar de  $x$  e  $y$ ?
12. Faça  $x^G'$  e  $G^x'$ . Qual a diferença entre os resultados?



# Unidade 3

## Manipulação de Vetores e Matrizes



### 3. Manipulação de Vetores e Matrizes

- O MATLAB permite a manipulação de linhas, colunas, elementos individuais e mesmo parte de matrizes;
- Para isso, basta selecionar o intervalo de índices que se deseja e alterar os seus valores.



#### 3.1 Gerando Vetores

- Vetores podem ser gerados através de listas explícitas de valores (como no caso de matrizes);
- Mas também podem ser formados por uma lista de elementos igualmente espaçados;



### 3.1 Gerando Vetores

- Caso o desejo seja produzir um conjunto de números em que cada elemento seja uma unidade basta utilizar os dois pontos entre os limites desejados;

• Exemplo:

>> t1 = 4:9

t1 =

4 5 6 7 8 9



### 3.1 Gerando Vetores

- Pode-se trabalhar com outros incrementos, declarando-os da seguinte forma:

>> vetor = limite inferior : incremento : limite superior

• Exemplo:

>> t2 = 4 : 0.5 : 9

t2 =

4 4.5 5 5.5 6 6.5 7 7.5 8  
8.5 9



### 3.1 Gerando Vetores

- Outra forma de se trabalhar é definindo o tipo de espaçamento entre valores e o número de valores no intervalo;

- O espaçamento pode ser linear (comando *linspace*) ou logarítmico (comando *logspace*) – no segundo caso definem-se as potências de 10 inferior e superior.



### 3.1 Gerando Vetores

- Exemplo:

```
>> t3 = linspace(0,2,6) = [0 0.4 0.8 1.2 1.6 2]
```

- Exemplo:

```
>> t4 = logspace(0,6,7) = [1 10 100 1000 10000
100000 1000000]
```



### 3.1 Gerando Vetores

- Um vetor também pode ser obtido extraindo-se uma linha ou coluna de uma matriz, assim, basta fazer:

```
>> vetor = matriz(linha desejada, :)
Ou
>> vetor = matriz(: , coluna desejada)
```



### 3.2 Manipulação de Matrizes

- Elementos de matrizes podem ser alterados atribuindo-lhes um novo valor. Basta indicar qual elemento (ou lista de elementos) se deseja modificar (através dos índices) e indicar o novo valor que ele deve receber.

```
>> Matriz(indice da linha, indice da coluna) = novo
valor
```



## 3.2 Manipulação de Matrizes

- Exemplo:

```
>> z
z =
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1

>> z(1:2:end,:) = 2
z =
2 2 2 2 2
1 1 1 1 1
2 2 2 2 2
1 1 1 1 1

>> z(1:2:end,1:3:end) = 5
z =
5 2 2 5 2
1 1 1 1 1
5 2 2 5 2
1 1 1 1 1
```



## 3.2 Manipulação de Matrizes

- Com isso é possível se modificar a matriz de forma a comportar novos dados conforme operações posteriores sejam realizadas – ao invés de criar novas matrizes o tempo todo.
- Essa implementação é mais rápida do que o uso de laços para modificação de uma série de valores.



## 3.2 Manipulação de Matrizes

- Para selecionar linhas ou colunas que não estejam em sequência, basta listá-las como se fossem elementos de um vetor, da seguinte forma:

`Matriz([linha1 linha2 linha3 ...],[coluna1 coluna2 coluna3 ...])`



# Unidade 4

## Gráficos



### 4. Gráficos

- Com o MATLAB é possível plotar gráficos a partir de dados salvos em vetores ou matrizes em diferentes escalas.
- Alguns dos comandos mais comuns são:
  - **plot** – escala linear
  - **loglog** – escala loglog
  - **semilogx** – escala logarítmica em Ox;
  - **semilogy** – escala logarítmica em Oy;
  - **bar** – gráfico de barras;
  - **stem** – sequência discreta de dados;
  - **compass** – setas que emanam da origem (formato de bússola)



### 4.1 Gráficos Bidimensionais

- Sendo V um vetor, o comando **plot(V)** nos dará um gráfico com eixos lineares dos valores de V versus seus índices;
- Se x e y são vetores, **plot(x,y)** produzirá o gráfico de y em função de x.



## 4.1 Gráficos Bidimensionais

- Exemplo:

```
>> x1 = -5:0.5:5;
>> y1 = -x1.^2 +12;
>> plot(x1,y1)
```



## 4.1 Gráficos Bidimensionais

- Outra ferramenta interessante é plotar diversas linhas em um mesmo gráfico, visando comparar o comportamento de conjuntos diferentes de dados.
- Isso pode ser feito tomando-se V, X ou Y como matrizes, ou colocando-se diferentes entradas na função plot;



## 4.1 Gráficos Bidimensionais

- Exemplos:

```
>> plot(x,G');
>> plot(G,y);
>> plot(x1,y1,x1,cos(x1),x1,sinh(x1));
```



## 4.1.1 Vários Gráficos em uma Figura

- Um comando muito interessante e que permite colocar vários gráficos em uma mesma figura é o *subplot*;
- Nele indica-se o número de linhas com gráficos desejadas, o número de colunas a se utilizar e qual a posição de um determinado gráfico na ordem.



## 4.1.1 Vários Gráficos em uma Figura

- Exemplo:

```
>> x2 = -3*pi:pi/8:3*pi;
>> figure;
>> subplot(3,2,1); plot(x2,cos(x2));
>> subplot(3,2,2); plot(x2,cos(x2 + pi/6));
>> subplot(3,2,3); plot(x2,sin(x2 + pi/9));
>> subplot(3,2,4); plot(x2,cos(5*x2 + pi/32));
>> subplot(3,2,5); plot(x2,cos(x2./4 + pi/9));
>> subplot(3,2,6); plot(x2,sin(x2));
```



## 4.2 Números Complexos

- Se os dados a se plotar são números complexos, usualmente a parte imaginária é descartada e plota-se apenas a parte real versus o índice.
- Todavia, ao se entrar com um argumento complexo na função *plot*, será dada a representação da parte real versus a parte imaginária;



## 4.2 Números Complexos

- Exemplo:  
`>> t5 = 0 : pi/12 : 2*pi;  
>> plot(exp(i*t5),'o');  
>> axis equal`
- Desenha um dodecágono centrado na origem do plano complexo e com afixos de módulo 1 e argumentos diferindo em  $\pi/12$  radianos;
- O comando *axis equal* obriga os eixos a terem o mesmo comprimento.



## 4.2 Números Complexos

- Outra possibilidade é usar a função *compass*, que apresentará os vetores complexos com seus módulos e argumentos;
- Exemplo:  
`>> t5 = 0 : pi/12 : 2*pi;  
>> compass(exp(i*t5),'o');`



## 4.3 Coordenadas polares

- Ainda existe outro comando chamado *polar* usando para plotar dados em coordenadas polares;
- Uso: *polar(theta,rho)*;
- Exemplo:  
`>> t6 = 0 : pi/12 : 2*pi;  
>> polar(t6, sin(2*t6).*cos(2*t6), '-r');`



## 4.4 Anotações nos Gráficos

- Existem duas possibilidades simples de se fazer anotações no gráfico:
  - A primeira é editando a figura gerada, indo em Edit -> Figure properties;
    - Pode-se escolher o tamanho e a cor de linha;
    - Pode-se alterar o título e a escala dos eixos;
    - Pode-se adicionar um título ao gráfico, entre outras coisas.




---



---



---



---



---



---



---



---

## 4.4 Anotações nos Gráficos

- A segunda possibilidade é utilizar os comandos específicos para essas edições:
  - >> title('texto') – adiciona um título ao gráfico
  - >> xlabel('texto') – título ao eixo Ox do gráfico
  - >> ylabel('texto') – título ao eixo Oy do gráfico
  - >> text('texto') – adiciona um texto ao gráfico
  - >> grid – adiciona linhas de grade no gráfico
  - >> legend('texto') – adiciona uma legenda




---



---



---



---



---



---



---



---

<http://minicursoieb2012.blogspot.com.br/>

**OPORTUNIDADE:  
MINI-CURSO ENGENHARIA  
BIOMÉDICA NA PRÁTICA DA  
UFSC**




---



---



---



---



---



---



---



---

<http://goo.gl/B7NyR>  
<http://goo.gl/CjZ1L>

**1º LINK: APOSTILA DE CMEX  
2º LINK: FAST CODING IN MATLAB**



# Unidade 5

## Dados



### 5. Dados

- Além de valores numéricos, o MATLAB aceita como entrada outros três tipos básicos de dados:
  - Strings
  - Cell arrays
  - Structs



## 5.1 Strings

- São formadas por concatenação de caracteres;
  - Variáveis tipo texto;
  - Notação: 'texto'
  - Exemplo:
- ```
>> texto = 'hello world'
```



5.2 Cell Array

- Tradução literal: conjunto de células;
- Corresponde a um “aglomerado” com campos (chamados de células) que podem conter dados de tipos diferentes e tamanhos diferentes;
- Elementos são declarados entre chaves;

```
>> tabela(1,1) = {'curso de matlab'};
>> tabela(1,2) = {10};
>> tabela(2,1) = {'unidades por dia'};
>> tabela(2,2) = {[13 45 78 9 10]};
>> tabela

tabela =
{'curso de matlab'    [      10]
'unidades por dia'   [1x5 double]}
```



5.3 Struct

- Também corresponde a um conjunto de elementos diversos;
- Tradução literal: estrutura;
- Cada campo pode corresponder a um tipo de dados diferente;
- Não corresponde a uma “tabela” e sim a um banco de dados com diversas entradas;



5.3 Struct

- Exemplo:

```
>> s = struct('type',{'big','little'},'color','red','x',{3 4})
```
- Para se referenciar a cada elemento da estrutura usa-se “nome da estrutura”.“elemento desejado”, por exemplo:

```
>> s.x
ans = 3
ans = 4
```



5.4 Comandos úteis

- Para verificar como estão os elementos na sua área de trabalho do MATLAB existem alguns comandos úteis:
 - ✓ who – indica as variáveis no Workspace
 - ✓ whos – indica informações sobre as variáveis
 - ✓ size – dá o tamanho das matrizes e vetores
 - ✓ help função – dá informação sobre a função desejada
 - ✓ doc – análogo ao help, abre a janela de documentação sobre o elemento
 - ✓ lookfor – procura nos arquivos fonte do MATLAB citações sobre aquela função



5.4 Comandos úteis

- Para lidar com os processos em execução ou em desenvolvimento:
 - ✓ Ctrl + C – interrompe o processo em execução
 - ✓ Ctrl + D – abre o código fonte de uma determinada função
 - ✓ F9 – executa um passo selecionado em uma lista de comandos
 - ✓ F5 – executa o arquivo *.m que está sendo editado
 - ✓ clc – limpa a janela de comandos
 - ✓ clear variable – limpa a variável desejada
 - ✓ figure – gera uma nova figura em branco



5.4 Comandos úteis

- Para alterar o formato de exibição de dados na tela usa-se o comando *format* seguido de:

✓ short
✓ shorte
✓ long
✓ longe
✓ hex
✓ rat
✓ bank
✓ ++

Para descobrir o que cada um faz, digite: *help format*



Unidade 6

Arquivos *.m



6. Arquivos *.m

- São arquivos utilizados para resolver dois problemas encontrados na programação em MATLAB:
 - ✓ Acrescentar um grande número de comandos
 - ✓ Realizar repetições de rotinas implementadas anteriormente
- Com isso, é possível se evitar ficar introduzindo comandos no prompt o tempo todo.



6. Arquivos *.m

- Eles são arquivos de texto comuns, editáveis inclusive no bloco de notas;
- Os comandos escritos são então executados como se fossem introduzidos manualmente no prompt;
- Permite realizar a programação em MATLAB.



6. Arquivos *.m

- Características:
 - ✓ Não é necessário colocar cabeçalho;
 - ✓ Não é necessário identificar previamente o tipo de comando que será realizado;
 - ✓ É possível se comentar as linhas de comando a fim de organizar o desenvolvimento (para isso usa-se o sinal %);
 - ✓ Podem ser usados para criar funções em MATLAB;



6. Arquivos *.m

- Exemplo de arquivo:
 - teste.m
 - fft.m
 - sin.m
- Para abrir um novo arquivo, digite: “Ctrl + N”



6. Arquivos *.m

- Atividade:
 - ✓ Abrir um novo script e digitar as seguintes matrizes:
 - $A = [1:3; 4:6; 7:9];$
 - $B = [1:4; 5:8; 9:12];$
 - $C = B;$
 - $D = [13:16; 17:20; 21:24];$
 - ✓ Salvar o arquivo com o nome aula2.m;
 - ✓ Digitar na janela de comandos aula2, dar ENTER e ver o que acontece;



6. Arquivos *.m

- Atividade:
 - ✓ Abrir um novo script e os seguintes comandos:
 - figure;
 - $f1=1;$
 - $f2=3;$
 - $f3=5;$
 - $f4=10;$
 - $f5=20;$
 - $fsamp = 10*f5$
 - $dt = 1/fsamp$
 - $nsamp = 2*pi/dt$



6. Arquivos *.m

- Atividade:
 - ✓ Digitar ainda:
 - $sub=1$
 - $dtsub=sub*dt$
 - $tmax=2*pi$
 - $t=0:dtsub:tmax;$
 - $y1 = sin(2*pi*f1*t);$
 - $y2 = sin(2*pi*f2*t)+2;$
 - $y3 = sin(2*pi*f3*t)+4;$
 - $y4 = sin(2*pi*f4*t)+6;$
 - $y5 = sin(2*pi*f5*t)+8;$
 - $ytudo = (y1 + y2 + y3 + y4 + y5 -20)/5+10;$



6. Arquivos *.m

- Atividade:
 - ✓ Digitar por fim:
 - plot(t,y1,'y',t,y2,'g',t,y3,'c',t,y4,'b',t,y5,'r',t,
y tudo,'k')
 - title(['dt=',num2str(dtsub)])
 - axis tight
 - pause
 - ✓ Salvar o arquivo com o nome aula2a.m;
 - ✓ Digitar na janela de comandos aula2a, dar ENTER e ver o que acontece;



Unidade 7

Controle de Fluxo



7. Controle de Fluxo

- O controle de fluxo se divide em três famílias de elementos:
 - ✓ Laços de repetição (for e while)
 - ✓ Laços condicionantes (if, else, elseif)
 - ✓ Seleção de casos (switch, case)



7.1 Laços de Repetição

- Como o próprio nome já diz, consiste em implementar um laço que contém procedimentos que se repetirão até que alguma condição seja satisfeita;
- Dividido em duas classes:
 - ✓ Laço For
 - ✓ Laço While



7.1.1 Laço FOR

- Obriga o programa a permanecer no loop enquanto o valor inicial não alcançar o valor final, sendo ele acrescido de um incremento a cada *loop*;



7.1.1 Laço FOR

- Sintaxe:
 - `for valor = início:incremento:final`
 - comandos
 - expressões
 - outros laços
 - seleções
 - etc
- `end`



7.1.1 Laço FOR

- Características:
 - ✓ Ele é realizado em três etapas distintas:
 - A primeira a aplicação da condição inicial;
 - A segunda a verificação se a condição (valor menor que valor final) é respeitada e a realização dos comandos internos caso ela seja verdadeira;
 - A terceira é a saída do laço quando a condição não é mais verdadeira;
 - ✓ Comando end é o limite inferior;



7.1.1 Laço FOR

- Exemplo:


```
for i = 1:2:12
        AA(i,1) = i^2;
        AA(i,2) = i^3;
      end
      AA
```
- Assim ele exibe a matriz AA ao terminar o programa



7.1.1 Laço FOR

- Exemplo:


```
A1 = zeros(10,5); % não é obrigatório
B1 = zeros(10,5); % não é obrigatório
```

```
for i = 1:10
  for j = 1:5
    A1(i,j) = i + j;
    B1(i,j) = i - j;
  end
end
A1, B1
C1 = A1 + B1
```



7.1.2 Laço WHILE

- Permanece no *loop* (entre as linhas while e end) enquanto a expressão booleana de entrada for verdadeira;



7.1.2 Laço WHILE

- Sintaxe:
 - while expressão_booleana
 - comandos
 - expressões
 - outros laços
 - seleções
 - etc
 - end



7.1.2 Laço WHILE

- Características:
 - ✓ Ele é realizado em duas etapas distintas:
 - Testa a condição da expressão booleana;
 - Se verdadeira, executa os comandos, se falsa, encerra o laço;
 - ✓ Comando end é o limite inferior;
 - ✓ Terminado o laço, ele continua a executar o corpo do programa;



7.1.2 Laço WHILE

- Exemplo:


```
a = 1; b = 23;
while a < b
    clc
    a = a + 2
    b = b - 1
    pause(.1) % pausa por 0,1 segundo
end
disp('Fim do Laço')
```



7.1.2 Laço WHILE

- Exemplo:


```
a = 1; b = 23; i = a;
while a < b
    clc
    a = a + 2
    b = b - 1
    i = i + 1
    x(i) = (a + b)^2;
    pause(.1) % pausa por 0,1 segundo
end
disp('Fim do Laço')
```



7.2 Laços Condicionantes

- Ao invés de realizar uma repetição de procedimentos no caso de uma expressão booleana, realizam um comando específico;
- Fazem a seleção de acordo com a resposta dos testes, comparações, verificações.



7.2 Laços Condicionantes

- Comparações

| Símbolo | Significado |
|----------|------------------|
| $= =$ | Igual |
| $\sim =$ | Diferente |
| $<$ | Menor que |
| $>$ | Maior que |
| $< =$ | Menor ou igual a |
| $> =$ | Maior ou igual a |
| \sim | Negação |
| $\&\&$ | E |
| $ $ | ou |



7.2 Laços Condicionantes

- Estabelece-se uma condição seguindo a tabela anterior e caso verdadeira, um comando é realizado;
- Se a condição inicial (if condição) não for verdadeira, pode-se estabelecer outras condições (elseif) ou apenas uma solução para o caso contrário (else);
- A sintaxe é similar a outras linguagens de programação.



7.2 Laços Condicionantes

- Sintaxe:
 - if A (sinal da tabela) B
 ação 1
 - elseif
 ação 2
 - else
 ação 3
 - end



7.2 Laços Condicionantes

- Ou seja:
 - ✓ Se a primeira condição for verdadeira (*if*) execute a ação 1;
 - ✓ Se ela for falsa, mas a segunda condição for verdadeira (*elseif*), execute a ação 2;
 - ✓ Se ambas forem falsas, execute a ação 3.



7.2 Laços Condicionantes

- Exemplo:

```
A = zeros(5,5);
for i = 1:5
    for j = 1:5
        if i == j
            A(i,j) = 2;
        elseif abs(i-j) == 1
            A(i,j) = -1;
        else
            A(i,j) = 37;
        end
    end
end
A
```



7.3 Comando Break

- Às vezes se faz interessante sair de um loop caso alguma condição seja satisfeita, sem necessitar realizar todas as repetições e testes;
- O comando *break* permite essa saída antecipada de um laço *for* ou *while*, fazendo que um laço mais interno seja terminado imediatamente.



7.3 Comando Break

- Exemplo:

% Rotina que modifica a matriz A
clc, %clear all, close all

```
x = 's'  
for i = 1:5  
    if x == 'q'  
        break  
    end  
  
    j = 1  
    while j<= 5  
        [A('num2str(i)', 'num2str(j)') = 'num2str(A(i,j))']
```



7.3 Comando Break

- Exemplo:

x = input('Modificar? (s-sim, n-não, p-próxima linha, q-sair) => 's'); % input que recebe string

```
if x == 's'  
    A(i,j) = input('Entre com o novo valor => ');  
    j = j+1; clc  
end  
  
if x == 'n'  
    j = j+1; clc  
end  
  
if x == 'p'  
    i = i+1; clc  
end
```



7.3 Comando Break

- Exemplo:

```
if x == 'q'  
    break  
end
```

```
if x == 'p'  
    i = i+1; clc  
end  
end  
clc  
end
```



7.4 Seleção de Casos

- Existe também uma estrutura que realiza a comparação dos valores dos dados com outros que foram especificados pelo usuário;
- Caso o valor seja igual a um desses, uma ação é executada;
- Essa estrutura se chama SWITCH.



7.4 Seleção de Casos

- Sintaxe:
 - switch valor_de_teste
 - case A
 - ação 1
 - case B
 - ação 2
 - case C
 - ação 3
 - otherwise
 - ação 4
 - end



7.4 Seleção de Casos

- A comparação é feita entre o valor de teste e os valores de cada um dos “cases”;
- Se nenhuma igualdade for observada, então a instrução após o comando otherwise é realizada;
- Apenas um dos casos é executado e após esse procedimento, o comando é encerrado.



7.4 Seleção de Casos

- A cada caso, podem ser atribuídos mais de um valor de teste, basta separá-los por vírgulas (A, B, C) ou colocá-los como componentes de uma *Cell Array* ({A, B, C})
- A instrução *otherwise* apesar de ser habitual não é necessária;



7.4 Seleção de Casos

- Exemplo:
- ```

x = input('Entre com um valor de 1 a 5 => ')
switch x
 case 1
 disp('Opção A')
 case 2
 disp('Opção B')
 case {3, 4, 5}
 disp('Opção C')
 otherwise
 disp('Nenhuma opção selecionada')
end

```



## 7.5 Exercícios

```

% Aproximação de uma função quadrada por um
% somatório de senóides
t = [0:256]'/256;
w0 = 2*pi; x = square(w0*t);
disp('Série de Fourier de uma onda quadrada');
disp(' ')
cont = 1;

```



## 7.5 Exercícios

```

while cont==1
 nin = input('Quantos harmônicos para aproximação? ');
 xhat = zeros(size(x));
 for k = 1:2:nin
 xhat = xhat + sin(k*w0*t)*4/(k*pi);
 end
 figure; plot(t,x,t,xhat); xlabel('Tempo (s)');
 title(['Sinal composto de ',num2str(nin),' harmônicos']);
 stp = input('1 -> repetir; 0 -> sair (e tecle Enter)');
 if stp == 1
 cont = 1;
 elseif stp ==0
 break
 end
end

```



# Unidade 8

## Arquivos de Funções



## 8. Funções em MATLAB

- Os arquivos \*.m em MATLAB podem ser usados para definir funções próprias, conforme o desejo do usuário. Para isso, eles devem apresentar:
  - ✓ Argumentos de entrada
  - ✓ Argumentos de saída
  - ✓ Corpo da função



## 8. Funções em MATLAB

- Sintaxe das funções:

```
function [outputs] =
 nome_da_funcao([inputs])
 % comentário contendo a explicação do que a
 função faz

 Corpo da função

end
```



## 8. Funções em MATLAB

- Exemplo:

```
function y = polinomio2grau(x,a,b,c)
 % define um polinômio de segundo grau
 y = a*x.^2 + b*x + c;
 plot(y)
```



### 8.1 Exercício

- Escrever uma função que tome como entrada um vetor  $x$  e os coeficientes de um polinômio do 3º grau e plote o gráfico desse polinômio – indicando a nomenclatura dos eixos e sendo a curva tracejada e de cor vermelha.
- Escrever uma função que tome como entrada um vetor  $x$  e dê como saída o produto do cosseno pelo seno de  $x$  acrescido de uma unidade.
- Escrever uma função que calcule a aproximação para onda quadrada realizada no exercício da unidade 7.



# Unidade 9

## Gráficos Tridimensionais



### 9. Gráficos 3D

- Os gráficos tridimensionais em MATLAB podem ser plotados usando diversas funções, entre elas:
  - ✓ `plot3` – plotar em espaço 3D
  - ✓ `fill3` – preencher superfície 3D com cor especificada
  - ✓ `contour` – plotar contorno 2D
  - ✓ `contour3` – plotar contorno 3D
  - ✓ `clabel` – plotar curva de nível
  - ✓ `quiver` – plotar gradiente em pontos da imagem
  - ✓ `mesh` – plotar malha 3D
  - ✓ `meshc` – combinação `mesh-contour`



### 9. Gráficos 3D

- Além delas ainda existem as seguintes funções:
  - ✓ `surf` – plotar superfície 3D
  - ✓ `surfc` – combinação `surf-contour`
  - ✓ `surfl` – plotar superfície com iluminação
  - ✓ `slice` – plot visualização volumétrica
  - ✓ `cylinder` – gera um cilindro
  - ✓ `sphere` – gera uma esfera



## 9. Gráficos 3D

- Para informações mais detalhadas acerca de cada tipo de gráfico tridimensional, consulte o help da função desejada.
- Serão apresentados alguns exemplos de gráficos 3D mais comuns, para melhor compreensão de seu funcionamento.



### 9.1 Comando PLOT3

- É o análogo tridimensional do comando plot;
- Dá uma linha no espaço em para x, y e z.
- Exemplo:  
 $t = 0:pi/50:10*pi;$   
 $plot3(sin(t),cos(t),t);$



### 9.1 Comando PLOT3

- Para alterar a cor e o formato da linha, e apresentar a indicação de pontos usa-se o mesmo padrão de string da função plot;
- Para dar nome ao gráfico e aos eixos usam-se os comandos:  
✓ title('texto');  
✓ xlabel('texto');  
✓ ylabel('texto');  
✓ zlabel('texto');



## 9.2 Comando MESH

- Como falado, o comando mesh é utilizado para criar uma malha 3D. Para tanto, os elementos da matriz Z são tidos como função das matrizes X e Y – quando usa-se mesh(X,Y,Z).
- Exemplo:  
 $[X,Y] = meshgrid(-5:0.2:5,-5:0.2:5);$   
 $Z = X.*\exp(-X.^2 - Y.^2);$   
 $mesh(X,Y,Z)$



## 9.3 Comando SURF

- O comando surf é usado para plotar uma superfície 3D em MATLAB;
- Sendo surf(X,Y,Z,C) o gráfico é plotado usando as cores definidas em C;
- Para surf(X,Y,Z) as cores são definidas segundo a altura no eixo Oz
- Exemplo:  
 $[X,Y,Z] = peaks(30);$   
 $surf(X,Y,Z)$   
 $\% axis([-3 3 -3 3 -10 5])$



## 9.4 Comando SURFC

- O comando surfc é usado para plotar uma superfície 3D em MATLAB e apresentar as linhas de contorno no plano inferior (indicando curvas de nível para o gráfico);
- Para surf(X,Y,Z) as cores são definidas segundo a altura no eixo Oz;
- Exemplo:  
 $[X,Y,Z] = peaks(30);$   
 $surfc(X,Y,Z)$   
 $\% axis([-3 3 -3 3 -10 5])$



## 9.5 Comando CONTOUR3

- O comando contour3 é usado para plotar linhas de contorno em 3D;

- Exemplo:

```
[X,Y] = meshgrid([-2:.25:2]);
Z = X.*exp(-X.^2-Y.^2);
contour3(X,Y,Z,30)
surface(X,Y,Z,'EdgeColor',[.8 .8 .8], 'FaceColor',
 'none')
grid off
view(-15,25)
colormap cool
```



## 9.6 Comando CLABEL

- O comando clabel é usado para plotar informações nas curvas de nível geradas com comando contour;

- Exemplo:

```
[x,y] = meshgrid(-2:.2:2);
z = x.^exp(-x.^2-y.^2);
[C,h] = contour(x,y,z);
clabel(C,h);
```



## 9.6 Comando CLABEL

- Nesse segundo exemplo todos os números estão na mesma direção

- Exemplo:

```
[x,y,z] = peaks;
[C,h] = contour(x,y,z);
clabel(C,h,'FontSize',15,'Color','r','Rotation',0)
```



## 9.7 Exercício

1. Escrever e rodar o código a seguir e explicar o que acontece.



## 9.7 Exercícios

```
n = 10; m = 10;
x = linspace(0,n,20); y = linspace(0,m,20);
[x, y] = meshgrid(x,y);
a = zeros(m,n);

for i = 1:m
 for j = 1:n
 a(i,j) = sqrt(i + j);
 end
end

z = [a+0.5 a'-0.5 ; (a.^2)/5 ((a'-0.1).^2)/2];
figure(39)
surf(x,y,z);
shading flat
```



## 9.7 Exercício

2. Escrever uma função que realize os mesmos procedimentos apresentados no código anterior, na qual o usuário entre apenas com os valores de m e n;



# Unidade 10

## Estatística



### 10. Funções de Estatística

- Existem algumas funções de estatística importantes em MATLAB. Considerando x um vetor, algumas delas são:
  - ✓ `mean(x)` – calcula o valor médio de x
  - ✓ `max(x)` – informa o valor máximo de x
  - ✓ `min(x)` – informa o valor mínimo de x
  - ✓ `var(x)` – indica a variância de x
  - ✓ `std(x)` – indica o desvio padrão de x
  - ✓ `median(x)` – calcula a mediana de x
  - ✓ `hist(x,intervalos)` – plota o histograma dos dados dentro de intervalos definidos pelo usuário



### 10. Funções de Estatística

- Exemplo 1:**  
`x = rand(1,500); % gera um vetor de números aleatórios`  
`x_med = mean(x)`  
`x_max = max(x)`  
`x_min = min(x)`  
`x_var = var(x)`  
`x_std = std(x)`  
`x_median = median(x)`



## 10. Funções de Estatística

- Exemplo 2:

```
dados1 = 2*rand(1,500) +2;
dados2 = randn(1,500) + 3;
```

```
subplot(2,1,1), plot(dados1)
axis([0 500 0 6])
title('Números Aleatórios 1')
```

```
subplot(2,1,2), plot(dados2)
title('Números Aleatórios 1')
xlabel('índice')
```



## 10. Funções de Estatística

- Continuação do exemplo 2:

```
min1 = min(dados1)
max1 = max(dados1)
mean1 = mean(dados1)
```

```
min2 = min(dados2)
max2 = max(dados2)
mean2 = mean(dados2)
```



## 10. Funções de Estatística

- Continuação do exemplo 2:

```
figure(301)
bins1 = 10; % número de intervalos para o cj de
 dados 1
bins2 = 0:0.5:6; % valores limites dos intervalos
 para os dados 2
```

```
subplot(2,1,1); hist(dados1,bins1)
xlabel('Intervalos'); ylabel('Contagens');
title('Histograma dados 1')
```

```
subplot(2,1,2); hist(dados2,bins2)
xlabel('Intervalos'); ylabel('Contagens');
title('Histograma dados 2')
```



## 10.1 Aplicação

- Estudo de dados ruidosos

```
% Sinal Ruidoso
figure(302);

% Dados do sinal e do ruído
t = linspace(0,10,512); % Intervalo de tempo de amostragem
s = sin(2*pi/5*t); % Sinal
n = 0.1*randn(size(t)); % Ruído aditivo gaussiano, desvio padrão 0.1

% Plotando os dados
subplot(2,1,1), plot(t,s)
axis([t(1) t(end)-1 1])
title('Sinal');
xLabel('Tempo (s)')

subplot(2,1,2), plot(t,n)
axis([t(1) t(end)-1 1])
title('Ruído');
xLabel('Tempo (s)')

pause;
```



## 10.1 Aplicação

- Estudo de dados ruidosos

```
% Adicionando ruído
x = s + n;
disp('Relação Sinal-Ruído (SNR), dB')
SNR = 20*log10(std(s)/std(n)); % SNR em dB
SNR2 = 10*log10(var(s)/var(n)); % SNR em dB (forma alternativa usando a variância)

figure(303);
plot(t,x)
xLabel('Tempo (s)')
yLabel('Amplitude do sinal');
title('Sinal Ruidoso')
pause
```



## 10.1 Aplicação

- Estudo de dados ruidosos

```
y = zeros(size(t)); % Alocando a memória para o sinal filtrado
% Filtragem do sinal
Y(1) = x(1);
Y(2) = (x(2) + x(1))/2;
Y(3) = (sum(x(1:3)))/3;

% Média para 4 amostras
for k = 4:length(t)
 y(k) = (sum(x(k-3:k)))/4;
end

disp('Relação Sinal-Ruído (SNR) do sinal de SAÍDA, dB')
SNRsaída = 20*log10(std(s)/std(y-s))
```



## 10.1 Aplicação

- Estudo de dados ruidosos

```
figure(304);
subplot(2,1,1), plot(t,x)
xLabel('Tempo (s)')
yLabel('Amplitude do sinal')
title('Sinal de Entrada')

subplot(2,1,2), plot(t,y)
xLabel('Tempo (s)')
yLabel('Amplitude do Sinal')
title('Sinal de Saída (Filtrado)')
```



## 10.1 Aplicação

- Estudo de dados ruidosos

```
% Filtragem usando a função específica do MATLAB
taps = 100;
y = filter(ones(1,taps),taps,x);]

disp('Relação Sina-Ruido (SNR) do sinal de SAÍDA, dB')
SNRsaida = 20*log10(std(s)/std(y-s))

figure(305);
subplot(2,1,1), plot(t,x)
xLabel('Tempo (s)')
yLabel('Amplitude do sinal')
title('Sinal de Entrada')

subplot(2,1,2), plot(t,y)
xLabel('Tempo (s)')
yLabel('Amplitude do Sinal')
title('Sinal de Saída (Filtrado)')
```



# Unidade 11

## Aplicações em Processamento de Sinais



## 11. Aplicações em Sinais

- Os sinais em MATLAB se apresentarão de três formas básicas:
  - ✓ Vetores – para sinais unidimensionais no tempo (um vetor para a amplitude do sinal e outro para o intervalo de tempo demandado na amostragem);
  - ✓ Matrizes – para imagens em tons de cinza (sinais bidimensionais em que cada pixel apresenta um valor de intensidade);
  - ✓ Tensores – para imagens coloridas ou dados de imagem com variações temporais ou espaciais.

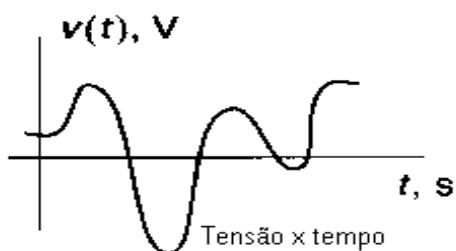


### 11.1 Sinais

- Os sinais são normalmente grandezas associadas à dimensão tempo ou espaço;
- Eles normalmente são compostos de uma sequência de dados onde seus elementos não estão relacionados, uns com os outros, pela variável temporal.



### 11.1 Sinais



## 11.2 Processamento de Sinais

- O processamento de sinais basicamente consiste em trabalhar com um conjunto de dados (proveniente de um sinal) para extrair alguma informação, para reconstruí-lo ou melhorá-lo;
- Ele está presente no nosso dia-a-dia, por exemplo nas telecomunicações em que se tem a modulação dos dados, a compressão das imagens.
- Além disso, apresenta várias aplicações em medicina, robótica, sistemas elétricos, geologia.



## 11.2 Processamento de Sinais

- O processamento de sinais pode, então, ser dividido em três classificações:
  - ✓ Processamento de baixo nível: entrada e saída são da mesma natureza (ondas, imagens) – ex: remoção de ruído;
  - ✓ Processamento de médio nível: entrada é um sinal e saída é um atributo desse sinal (ex: segmentação);
  - ✓ Processamento de alto nível: processos que envolvem dar sentido a algo a partir dos dados de entrada, pode envolver algoritmos inteligentes;

Agradecimento: Alexandre Zaghetto, DSc.



## 11.3 Aplicação em Processamento de Áudio

- Os sinais de áudio são formados por vetores de amplitude que variam com o tempo;
- Além disso, tem-se o vetor relacionado às amostras temporais.
- Exemplo:
 

```
load handel
for k = 1:0.2:2
 sound(y,k*Fs)
end
```



## 11.3 Aplicação em Processamento de Áudio

- Exemplo:
- ```
load gong.mat
gong = audioplayer(y,Fs);
play(gong);

load chirp.mat
for k = 1:0.2:2
    sound(y,k*Fs)
end
```



11.3 Aplicação em Processamento de Áudio

- Exemplo:
- ```
fs = 44100; T = 1/fs;
t = 0:T:2.25;
f1 = 50;
omega1 = 2*pi*f1;
phi = 2*pi*0.75;
x1 = cos(omega1*t + phi);
subplot(2,1,1); plot(t,x1);
sound(0.9*x1,fs);
```



## 11.3 Aplicação em Processamento de Áudio

- Exemplo:
- ```
phi = 2*pi*0.25;
x1 = cos(omega1*t + phi);
x2 = cos(2*pi*150*t + phi)/3;
x3 = cos(2*pi*250*t + phi)/5;
x4 = cos(2*pi*350*t + phi)/7;
x5 = cos(2*pi*450*t + phi)/9;
xcomplex = x1 + x2 + x3 + x4 + x5;

subplot(2,1,2); plot(t,xcomplex);
sound(0.9*xcomplex,fs);
```



11.4 Aplicação em Processamento de Imagens

- Em processamento de imagens, ao invés do sinal ser variante no tempo, ele é variante no espaço;
- As imagens, quando formadas por uma única matriz, são em escala de cinza – em que há a variação apenas de intensidade de luz;
- Para imagens coloridas, tem-se tensores de tamanho M_{pixels} por N_{pixels} por 3, em que cada matriz corresponderá a uma cor: R, G, B.



11.4 Aplicação em Processamento de Imagens

- Existem duas formas de importar os arquivos de imagem:
 - ✓ A primeira é usando o comando `Matriz = imread('Nome da Imagem.formato');`
 - ✓ A segunda é selecionar o arquivo no diretório e arrastá-lo para a área de trabalho – o que pode gerar problemas para arquivos indexados ao dividi-los em matrizes distintas;



11.4 Aplicação em Processamento de Imagens

- A imagem colorida importada então se mostrará da seguinte forma na área de trabalho:

Matriz <Mpixels x Npixels x 3 uint8>

- Sendo que a notação `uint8` indica o uso de 8 bits por pixel;



11.4 Aplicação em Processamento de Imagens

- Desse modo, temos que:
 - ✓ `Matriz(:,:,1)` = indica a matriz que contém as intensidades para o vermelho;
 - ✓ `Matriz(:,:,2)` = indica a matriz que contém as intensidades para o verde;
 - ✓ `Matriz(:,:,3)` = indica a matriz que contém as intensidades para o azul;
- E por se tratarem de matrizes de intensidade, elas serão representadas em escala de cinza;



11.4 Aplicação em Processamento de Imagens

- Existem três comandos que permitem a visualização dos dados da imagem, são eles:
 - ✓ `imshow(Matriz)` – mostra a imagem RGB dada, ou cada um de seus componentes, sem a presença de eixos nas laterais;
 - ✓ `image(Matriz)` – apresenta a imagem mostrando eixos que indicam a posição dos elementos de acordo com o índice dos pixels;
 - ✓ `imagesc(Matriz)` – análogo ao comando `image`, mas a imagem é escalonada para usar todo o colormap;



11.4 Aplicação em Processamento de Imagens

- Além deles, o comando `plot` pode ser usado para apresentar um gráfico que contenha as intensidades dos valores em um dado conjunto de linhas da imagem.



11.4 Aplicação em Processamento de Imagens

- Fazer download do arquivo presente em: <https://dl.dropbox.com/u/15850261/Imagens.zip>
- Descomprimi-lo, salvando as pastas das imagens no diretório em que o MATLAB está sendo executado.
- Esperar as próximas instruções.



11.4 Aplicação em Processamento de Imagens

- No diretório, abrir a pasta imagens para teste e digitar e dar ENTER:
`Lena = imread('lena_color_512.tif');`
- Verificar o surgimento do tensor "Lena" no Workspace do MATLAB;
- Executar:
`figure,imshow(Lena(:,:,1))`
`figure,imshow(Lena(:,:,2))`
`figure,imshow(Lena(:,:,3))`
- Observar e explicar as diferenças entre as imagens.



11.4 Aplicação em Processamento de Imagens

- Fazer:
 - `Lena1(:,:,1) = abs(Lena(:,:,1) - 35);`
 - `Lena1(:,:,2) = Lena(:,:,2);`
 - `Lena1(:,:,3) = Lena(:,:,3);`
 - `figure,imshow(Lena1);`
- O que aconteceu com a imagem?



11.4 Aplicação em Processamento de Imagens

- Percebe-se uma redução dos níveis de vermelho e com isso uma alteração na intensidade luminosa da imagem (visto que parte de seus pixels apresenta agora valor mais baixo).
- Testem outros valores para mudanças dos dados dos pixels da imagem e analisem o comportamento final.



11.4 Aplicação em Processamento de Imagens

- Agora abram a pasta “Imagens Para Realce”;
- Façam a leitura da imagem “Fig3.04(a).jpg”, da seguinte forma:
 $A1 = imread('Fig3.04(a).jpg');$
- Essa imagem está em escala de cinza, logo a sua representação no MATLAB é por apenas uma matriz;



11.4 Aplicação em Processamento de Imagens

- Para essa imagem, testaremos realces no domínio da imagem;
- Inicialmente testaremos a análise em negativo, para tanto, façam:
 $A1a = max(A1(:)) - A1;$
- Com isso, os pixels que estavam em preto, agora se mostrarão em tons claros e vice-versa.



11.5 Exercício

1. Ler as outras imagens presentes na pasta imagens para realce e cujo nome comece com Fig3 e salvá-las no MATLAB com os nomes A2, A3, A4, A5, A6 e A7.
2. Para imagem A2, fazer:
 - i. $A2a = \text{uint8}(3 * \log(\text{double}(1 + A2)))$;
 - ii. O que acontece?
3. Para imagem A3, fazer:
 - i. $A3a = \text{uint8}(4 * (\text{double}(A3) .^ 0.6))$;
 - ii. Variar o expoente e explicar os resultados.
4. Para imagem A4:
 - i. $A4a = 4 * (\text{double}(A4.^5))$;



11.5 Exercício

5. Para as imagens A5, analisar o histograma, fazendo:
 - i. `figure, imshow(A5);`
 - ii. Usar o comando `A5a = histeq(A5);`
 - iii. Fazer `figure, imshow(A5a);`
6. Para as imagens A6 e A7, combinar as operações realizadas e verificar os resultados.



Unidade 12

Polinômios



12. Polinômios

- O MATLAB apresenta funções para análise e manipulação de polinômios que permitem:
 - ✓ Diferenciá-los;
 - ✓ Integrá-los;
 - ✓ Calcular suas raízes;



12. Polinômios

- Um polinômio em MATLAB é representado por um vetor que contém os seus coeficientes em ordem decrescente;
- Por exemplo, o polinômio
 - $p(x) = x.^4 - 12*x.^3 + 0*x.^2 - 12*x + 1$
- É escrito como:
 - $p = [1 -12 0 -12 1]$
- Sendo que os termos com coeficiente nulo devem ser incluídos.



12. Polinômios

- Para calcular as raízes de p , usa-se a função `roots`, desse modo:
 $r = \text{roots}(p)$
- Devolverá os valores em que $p(x)$ é nulo;
- Esse vetor será, por convenção, um vetor coluna.



12. Polinômios

- A partir de um conjunto de raízes, é possível se obter o polinômio usando a função poly, desse modo:
 $pp = \text{poly}(r)$
- Devolverá os valores dos coeficientes correspondentes ao polinômio $p(x)$;



12. Polinômios

- Para a multiplicação de dois polinômios existe a função conv, assim, sendo a e b vetores que contém os coeficientes de polinômios
 $c = \text{conv}(a,b)$
- Conterá os coeficientes do polinômio correspondente ao produto entre eles.



12. Polinômios

- Para a divisão de dois polinômios existe a função deconv, que retorna o quociente e o resto da divisão entre polinômios a e b, então:
 $[q,r] = \text{deconv}(a,b)$
- “q” conterá os coeficientes do polinômio quociente, enquanto “r” corresponde aos coeficientes do resto.



12. Polinômios

- Para a adição de dois polinômios não existe uma função específica, então, deve-se analisar a ordem desses polinômios e completar os coeficientes do menor para os vetores serem do mesmo tamanho:
 $a = 1:3;$ $b = 5:-1:1;$
- Então será necessário completar o valor de a com zeros, para ele ficar do mesmo grau que b e possibilitar a soma:
 $a1 = [0 \ 0 \ a];$



12. Polinômios

- Assim como para o cálculo das raízes, existe uma função específica para encontrar a derivada de um polinômio, ela é a polyder, e é usada da seguinte forma:
 $a_der = polyder(a);$



12. Polinômios

- Para calcular a derivada da multiplicação, usa-se:
 $polyder(a,b)$
- Para a derivada o quociente:
 $[n,d] = polyder(a,b)$



12. Polinômios

- Analogamente à derivada, existe uma função a ser usada para calcular a integral de um polinômio, chamada polyint:
`a_int = polyint(a);`
- Assim, ela realizará a integral indefinida de "a", gerando um polinômio com coeficiente independente nulo.



12. Polinômios

- Caso se deseje especificar o termo independente, isso deve ser feito da seguinte forma:
`polyint(polinomio,termo_independente).`
- Exemplo:
`a_int = polyint(a,55);`



12. Polinômios

- Existe ainda uma função que permite avaliar os valores do polinômio, chamada polyval;
- Ela calcula o polinômio para o valor desejado de x;
- Exemplo:
`x = -15:15;`
`valores = polyval(a,x);`
`plot(x,valores)`



12.1 Polinômios: Ajuste de Curvas

- Pode-se usar ainda polinômios para fazer o ajuste de curvas, usando o método dos mínimos quadrados.
- Para isso, usa-se a função polyfit.
- Nela, são fornecidos os dados das abscissas, ordenadas e o grau do polinômio que se deseja usar no ajuste.



12.1 Polinômios: Ajuste de Curvas

- Da seguinte forma:
 $p_{ajuste} = \text{polyfit}(\text{abscissas}, \text{ordenadas}, \text{grau})$
- Ou, de forma mais usual:
 $p_{ajuste} = \text{polyfit}(x, y, n)$



12.1 Polinômios: Ajuste de Curvas

- Lembrando que para definir corretamente um polinômio de grau n são necessários $n+1$ pontos;
- Todavia, deve-se tomar cuidado de não exagerar o grau do polinômio a ser utilizado na aproximação, visto que isso pode resultar numa curva pouco suave.



12.1 Polinômios: Ajuste de Curvas

- O MATLAB ainda apresenta uma família de funções própria para realizar o ajuste de curvas;
- A função para dados unidimensionais é a `interp1`, em que são fornecidos como parâmetros de entrada os vetores `x` e `y`, além de um vetor que contém as abscissas dos pontos da curva interpolada;



12.1 Polinômios: Ajuste de Curvas

- Assim:
 $p_{interpolado} = interp1(x, y, x_{inter}, \text{método});$
- E o método de interpolação é opcional, sendo as possibilidades: 'nearest', 'linear', 'spline', 'pchip', 'cubic';
- Para mais informações sobre cada um dos métodos, acesse: `doc interp1`



12.1 Polinômios: Ajuste de Curvas

- Com esse método de interpolação, ainda é possível procurar algum valor que extrapole aqueles presentes no vetor `x_inter`, definindo
- Assim:
 $p_{interpolado} = interp1(x, y, x_{inter}, \text{método}, \text{valor_extrapolado});$



12.2 Interpolação Bidimensional

- Análoga a interp1, existe a função interp2 que realiza a interpolação bidimensional.
- A diferença é que nela se faz necessário definir não só o x_inter, mas o y_inter, ficando:

$$z_{interpolado} = \text{interp2}(x, y, x_{inter}, y_{inter}, \text{método});$$



Unidade 13

Integração e Diferenciação Numérica



13. Cálculo Numérico

- O MATLAB inclui funções que possibilitam aproximar numericamente a integral e a inclinação de uma curva;
- Isso também é possível para funções tabuladas em pontos igualmente espaçados sobre a região de interesse.



13.1 Integração

- Pode-se usar, por exemplo, uma aproximação trapezoidal para calcular a integral. Para tanto, usa-se a função trapz(x,y).
 - $\text{area} = \text{trapz}(x,y)$
- Todavia, essa aproximação é melhor quando as distâncias entre os x_i 's são pequenas – soma de Riemann.



13.1 Integração

- Desse modo, seria necessário testar valores para os intervalos de x a fim de ter um resultado “ótimo”.
- Esse processo de chutar valores para o intervalo não é interessante.
- Para tanto, o MATLAB desenvolveu as funções quad e quadl que realizam esse cálculo usando intervalos de comprimentos precisos.



13.1 Integração

- Sintaxe:
 - $\text{integral} = \text{quadl}(\text{funcao}, \text{x_min}, \text{x_max})$
- Ou
 - $\text{integral} = \text{quad}(\text{funcao}, \text{x_min}, \text{x_max})$



13.1 Integração

- Exemplo:
- Definindo o polinômio (em um arquivo .m separado):


```
function y = polinomio2grauA(x)
y = 1*x.^2 + 2*x + 3;
```
- Calculando a integral (na janela de comandos)


```
quadl(@polinomio2grauA,0,3)
quadl('polinomio2grauA',0,3)
```



13.1 Integração

- Exemplo:
- Outra forma, é declarar a função da seguinte maneira:


```
F = @(x) 1./(x.^3-2*x-5);
```
- E, em seguida, calcular a integral.


```
quadl(F,0,3)
```



13.1 Integração

- O MATLAB ainda apresenta uma função para cálculo da integral em duas dimensões a dblquad;
- Para tanto, é necessário definir uma função $f(x,y)$



13.1 Integração

- Exemplo:
 - $F = @(x,y)y*\sin(x)+x*\cos(y);$
 - $Q = dblquad(F,pi,2*pi,0,pi)$
- Em que F será uma função de x e y, e Q retornará o valor da integral;



13.1 Integração

- Exemplo:


```
F1 = @(x,y) sin(x).*cos(y)+1;
x = linspace(0,pi,20);
y = linspace(-pi,pi,20);
[xx,yy] = meshgrid(x,y);
zz = F1(xx,yy);
figure(6);
surf(xx,yy,zz)
xlabel('x'), ylabel('y')
title('Gráfico de F1')
shading interp
```



13.1 Integração

- Exemplo:


```
x1 = linspace(-pi,pi,20);
[xx1,yy1] = meshgrid(x1,x1);
zz1 = F1(xx1,yy1);
figure(7);
surf(xx1,yy1,zz1)
xlabel('x'), ylabel('y')
title('Gráfico 2 F1')
shading interp
```

volume = dblquad(F1,0,pi,-pi,pi)



13.1 Integração

- Para três dimensões existe a função triplequad, análoga à dblquad e à quadl.



13.2 Diferenciação

- Bem mais difícil que a integração e sensível a pequenas variações nas curvas, a diferenciação numérica é evitada sempre que possível, principalmente para dados obtidos em experimentos!
- O melhor sempre é fazer um ajuste do polinômio antes de aplicar a diferenciação numérica.



13.2 Diferenciação

- O MATLAB, entretanto, possui uma função que calcula a derivada aproximada a partir de dados tabulados chamada diff;
- Ela calcula a derivada tomando como base o quociente de Newton, da seguinte forma:

$$f'(x_k) = (f(x_{k+1}) - f(x_k)) / (x_{k+1} - x_k)$$



13.2 Diferenciação

- Exemplo:

```
dx = .001; x = 0:dx:10;
y = 8 + 2*cos(x).*sin(3*x);
diffy = diff(y); dxdy = diffy/dx;
plot(x,y,'b','LineWidth',2.5);
hold on
plot(x(2:end),dxdy,'r','LineWidth',1.5);
hold off; grid
legend('y','dy/dx');
```



13.2 Diferenciação

- Ainda é possível calcular o gradiente e o laplaciano para conjuntos de dados bidimensionais;
- Para o gradiente usa-se a função gradient;
- Para o laplaciano a função del2;



13.2 Diferenciação

- Exemplo (Gradiente):

```
[x,y,z] = peaks(20);
dx = x(1,2) - x(1,1) % espaçamento na direção x
dy = y(2,1) - y(1,1) % espaçamento na direção y
[dzdx,dzdy] = gradient(z,dx,dy)
contour(x,y,z)
hold on
quiver(x,y,dzdx,dzdy)
hold off
title('Gráfico de setas de gradientes')
```



13.2 Diferenciação

- Exemplo (Laplaciano):


```
[x,y,z] = peaks;
dx = x(1,2) - x(1,1); % espaçoamento na direção x
dy = y(2,1) - y(1,1); % espaçoamento na direção y
L = del2(z,dx,dy);

surf(x,y,z,abs(L));
shading interp

title('Laplaciano discreto colorido')
```



Unidade 14

Matemática Simbólica



14. Matemática Simbólica

- A matemática simbólica em MATLAB consiste usar expressões com letras e algarismos para serem resolvidas ao invés de trabalhar sempre com dados numéricos;
- Para tanto usa-se a função `syms` para atribuir valor simbólico aos elementos;



14. Matemática Simbólica

- Assim, `syms a` torna “a” um elemento simbólico ao invés de uma matriz;
- A função `subs(elemento)` atribui ao elemento um valor numérico;
- Para determinar funções compostas usa-se `compose()` e para inversas `finverse()`;



14. Matemática Simbólica

- Derivar, integrar e limite: `diff()`, `int()`, `limit()`, respectivamente.
- Formatar e simplificar as expressões: `pretty()`, `simplify()`, `collect()`, `expand()`, `simple()`
- Converter entre polinômios e simbólicos: `poly2sym()`; `sym2poly()`;
- Traçar gráficos para simbólicos: `ezplot()`



Unidade 15

Outros Comandos Úteis



15. Comandos Úteis

- Existem vários outros comandos úteis em MATLAB, entre eles, temos:
 - ✓ `Cart2pol` – muda as coordenadas de cartesianas para polares
 - ✓ `Cart2sph` – muda as coordenadas de cartesianas para esféricas
 - ✓ `flipud` – inverte colunas da matriz;
 - ✓ `fliplr` – inverte linhas da matriz;

Para outros, consultem:

<http://www.mec.ita.br/~adade/Matlab/Web/anexo.htm>

