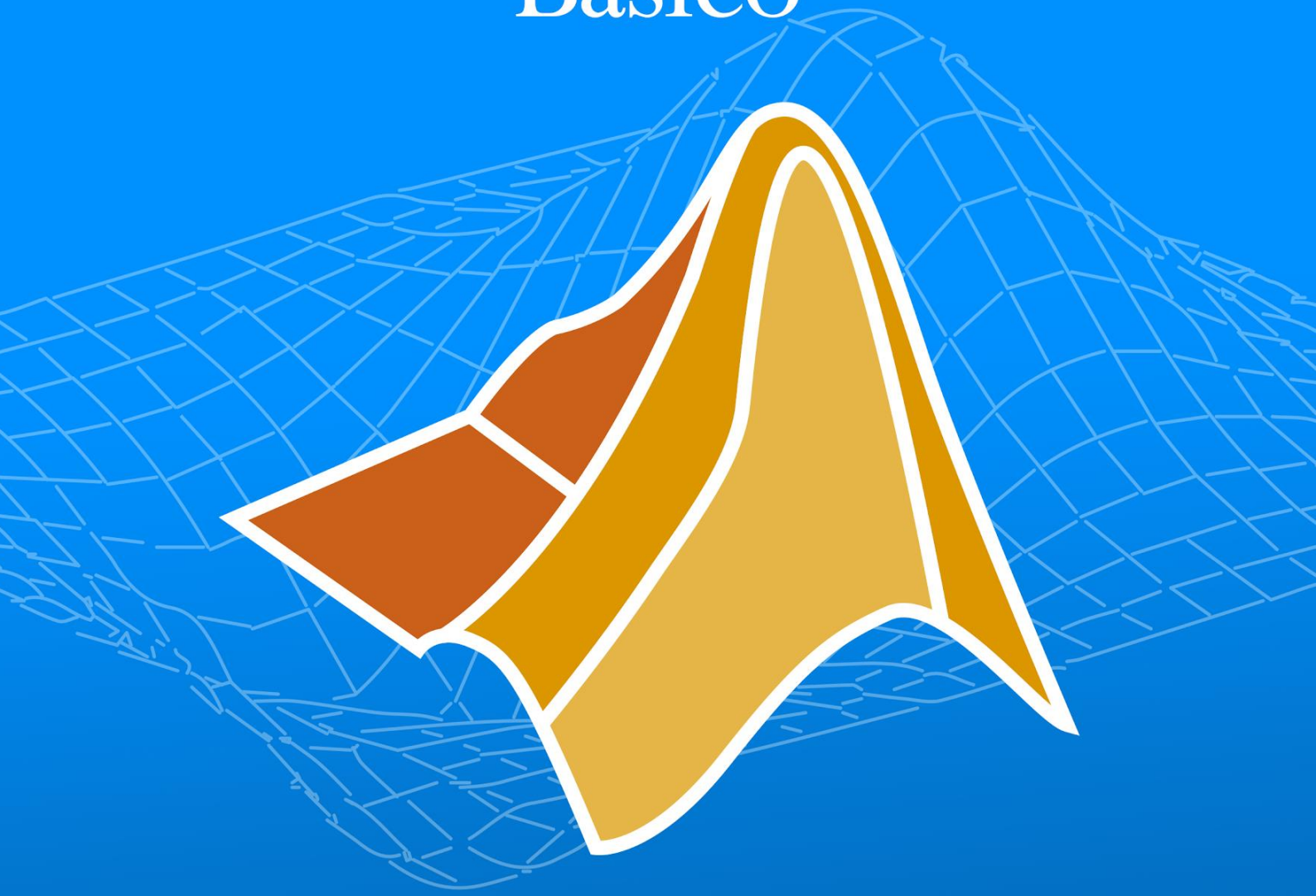
MATLAB

Básico



Apostila do minicurso

SUMÁRIO

1. O software MATLAB

O MATLAB (Matrix Laboratory) é software especializado em cálculos numéricos. Ele foi criado no fim dos anos 70 por Cleve Moler, na época presidente do departamento de ciência da computação da Universidade do Novo México. Rapidamente, o software ganhou espaço na comunidade de matemática aplicada. Em uma visita de Moler a Universidade de Stanford em 1983, um engenheiro chamado Jack Little conheceu a linguagem MATLAB. Jack Little juntou-se a Moler e Steve Bangert e em 1984 fundaram a Mathworks e, posteriormente, trabalharam no desenvolvimento de novos softwares com aplicações variadas.

O MATLAB foi criado com a proposta de ser um ambiente de fácil aprendizagem em que pudessem ser resolvidos testes e soluções matemáticas com rapidez e facilidade. A linguagem do MATLAB é bastante simples quando comparada com outras linguagens de programação, isto é, os problemas e soluções matemáticas devem ser escritos em linguagem matemática ao invés da linguagem de programação tradicional e, consequentemente, sua utilização se torna mais simples e intuitiva.

As primeiras aplicações do MATLAB foram no cenário de projeto de controle e rapidamente se expandiu para outros campos.

2. A empresa MathWorks

A MathWorks é uma empresa privada americana, fundada em 1984, especializada em software de computação matemática.

Os principais produtos desenvolvidos pela MathWorks são:

Além do MATLAB, a MathWorks possui outro software bastante conhecido, o Simulink. Este software trata-se de uma ferramenta para modelagem, simulação e análise de sistemas dinâmicos. O Simulink utiliza diagramação gráfica por blocos. Possui uma alta integração com o MATLAB. É muito utilizado para teorias de controle e processamento digital.

O Simulink é utilizado junto com o MATLAB por pesquisadores e engenheiros da área de robótica para projetar e ajustar algoritmos, modelar sistemas do mundo real e gerar automaticamente o código.

3. Aplicações do MATLAB

Devido a seu potencial em trabalhar com modelagens matemáticas de alto grau de complexidade, o MATLAB vem sendo amplamente utilizado em problemas de engenharia que exigem soluções e simulações numéricas. Os métodos numéricos que podem ser aplicados através do MATLAB vêm sofrendo grandes avanços permitindo a obtenção de soluções cada vez mais próximas da analítica ou a níveis de aproximação suficientemente confiável.

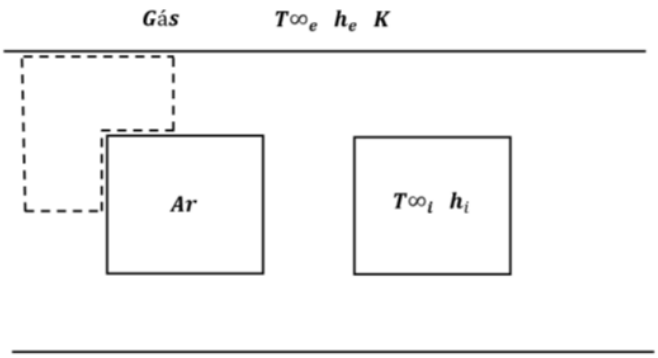
A utilização desses métodos numéricos nessa categoria de problemas tem se tornado cada vez mais importante e tem se sobressaído aos processos experimentais, visto que as simulações podem evitar falhas catastróficas, além de reduzir gastos.

Dentre as áreas de aplicação que podem ser citadas são:

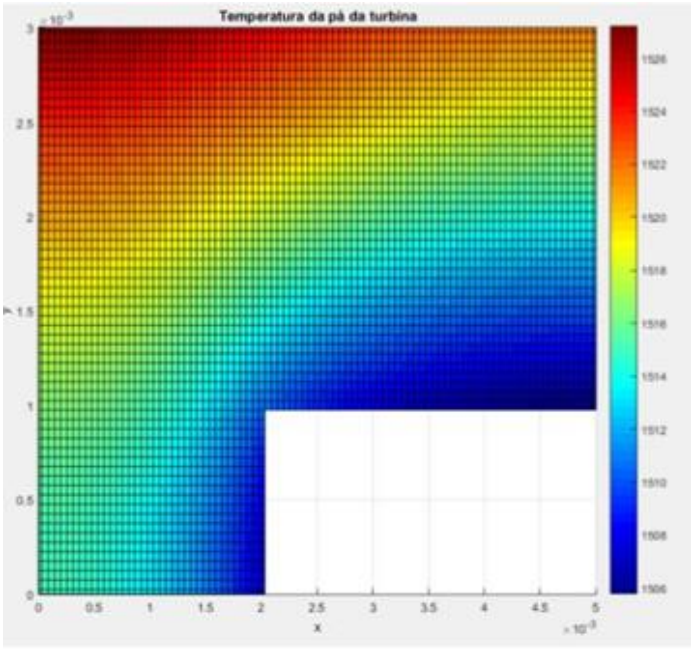
Através de códigos computacionais podem ser obtidos resoluções para os problemas, sendo que essas resoluções são armazenadas em matrizes e podem ser posteriormente representadas em gráficos facilitando a interpretação física dos resultados.

3.1. Transferência de Calor

Para estudos de transferência de calor, a situação abaixo mostra o escoamento de gases de combustão na parte externa da pá de uma turbina, que possui um canal de ar para resfriamento no seu interior.



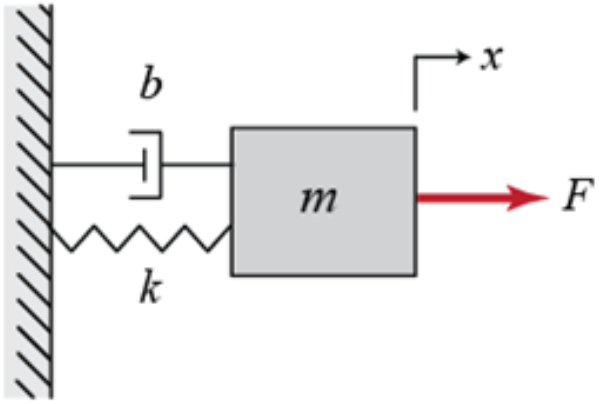
Na imagem abaixo, pode ser observada uma distribuição bidimensional de temperaturas em uma região da pá de uma turbina em regime permanente. O método matemático de solução desse problema foi o método dos volumes finitos. A região inferior direita da imagem mostra uma parte que é resfriada por um fluido refrigerante e, por isso, as temperaturas são crescentes em direção a parte superior esquerda.



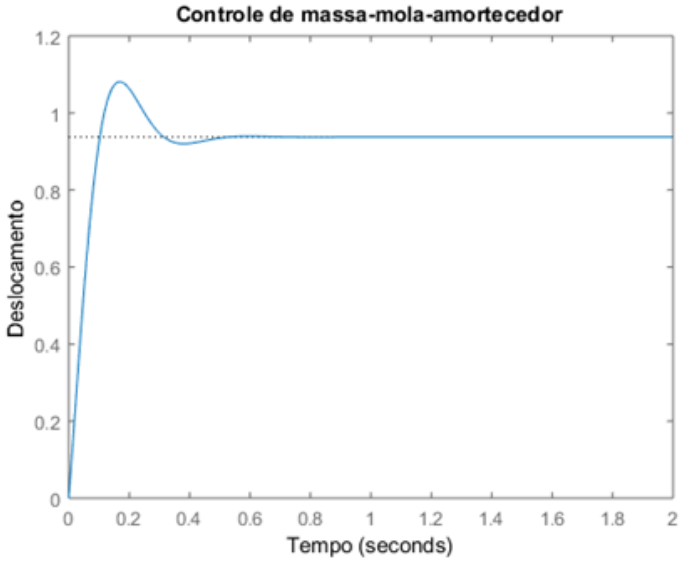
3.2. Controle de Sistemas Dinâmicos

Para controle de sistemas dinâmicos, é possível projetar controladores com os melhores parâmetros possíveis para estabilizar diversos tipos de funções de transferência.

Para o sistema massa-mola-amortecedor de um grau de liberdade que é submetido a uma força externa, mostrado na imagem abaixo:



A imagem abaixo mostra a resposta do sistema quando um controlador PID atua sobre ele. Pode-se observar que ao longo do tempo o sistema atinge a estabilidade em um valor próximo da referência, que nesse caso é um degrau unitário.

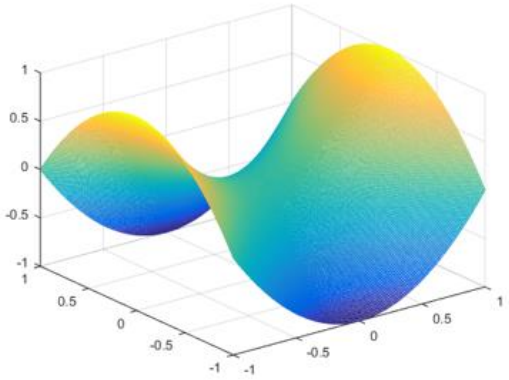


3.3. Cálculo diferencial e integral

Podemos ainda trabalhar com casos estritamente matemáticos, como resolver equações algébricas simples até problemas de valor inicial e problemas de valor de contorno envolvendo equações diferenciais.

Um exemplo a ser mostrado é a geração de um gráfico de um parabolóide hiperbólico cuja equação é dada por:

$$f(x,y) = x^2 - y^2$$



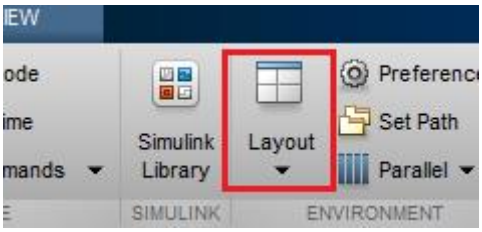
4. O funcionamento do MATLAB

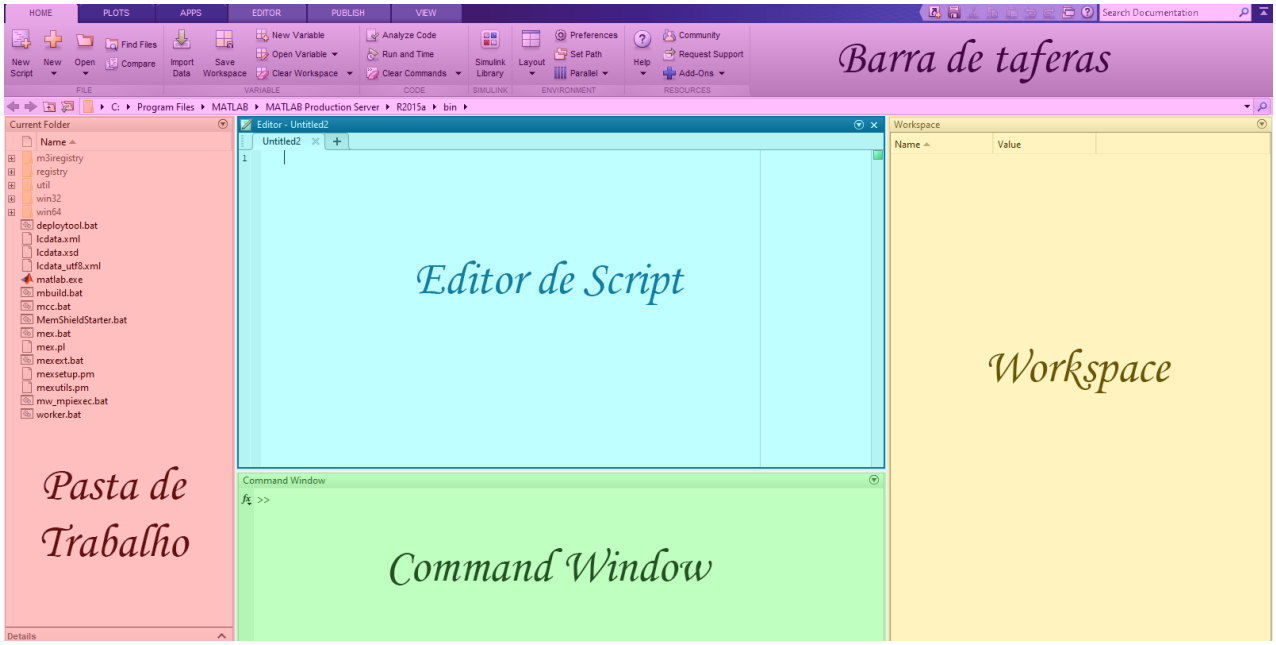
O funcionamento básico do MATLAB é baseado em matrizes. Um matriz m x n é um conjunto de m linhas e n colunas, sendo que cada posição da matriz é ocupada por um valor numérico. Dentre as aplicações utilizando matrizes, podemos citar a resolução de sistemas lineares e transformações lineares. Até mesmo um escalar é entendido pelo MATLAB como sendo matriz composta por uma linha e uma coluna.

```
>> a = 3

a =

     3

>> whos a
  Name        Size              Bytes  Class     Attributes

  a           1x1                   8  double
```

5. Interface de Trabalho

Utilizando o layout *Default* para a interface do MATLAB, ao abrir um *New Script* na parte superior esquerda da aba *Home* da barra de tarefas, pode-se observar o seguinte design para a interface do software:





- ***Command Window*:** nessa janela é possível inserir comandos interativos pelo marcador (>>) e eles serão executados de imediato. O MATLAB realiza o comando assim que a tecla *enter* é pressionada e o resultado é armazenado em uma variável (uma matriz 1x1). A variável padrão para resultados é a variável *ans.*

- ***Workspace*:** nesse local são registradas as variáveis e matrizes, que foram declaradas atualmente bem como os valores atribuídos a elas. É atualizado toda vez que o código é executado.

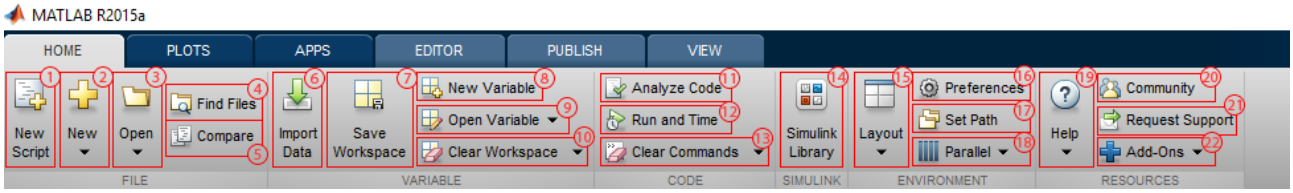
- ***Pasta de Trabalho*:** mostra os arquivos da pasta na qual o MATLAB está instalado. É possível modificar a pasta para localizar os arquivos desejados.

- ***Editor de Script*:** nesse local podem ser digitados todos os comandos desejados de um programa sem que eles sejam executados, podendo ainda serem editados em qualquer momento. A partir da execução, todo as operações do código ocorrem e as variáveis armazenadas aparecem no *Workspace*.

- ***Barra de Tarefas*:** nela se localizam os comandos como salvar e carregar um arquivo, personalização de interface, compilação e execução do programa, entre muitos outros comandos comuns do software.

5.1. Barra de Tarefas

5.1.1. Home (Início)



1 - *Abre um novo "script".*

Atalho: Ctrl + N

2 - *Cria um novo documento do MATLAB (.m).*

3 - *Abre um arquivo do MATLAB (.m) já existente.*

Atalho: Ctrl + O

4 - *Procura por um arquivo baseado no nome ou no conteúdo.*

Atalho: Ctrl + Shift + F

5 - *Compara o conteúdo de dois arquivos distintos.*

6 - *Importa data de um arquivo.*

7 - *Salva as variáveis do Workspace para um arquivo.*

8 - *Cria uma nova variável e abre o editor de variáveis.*

9 - *Abre uma variável para edição.*

10 - *Limpa todas as variáveis do Workspace.*

11 - *Analisa os códigos no MATLAB na pasta atual para codificar potenciais bugs e ineficiências.*

12 - *Executa o código e mensura o tempo de execução para melhoria de performance.*

13 - *Limpa todo o Command Window.*

14 - *Abre a janela de blocos do Simulink.*

15 - *Permite a mudança de interface do software.*

16 - *Permite ajustar as configurações.*

17 - *Modifica o caminho de busca usado pelo MATLAB na procura por arquivos.*

18 - *Modifica opções de computação paralela.*

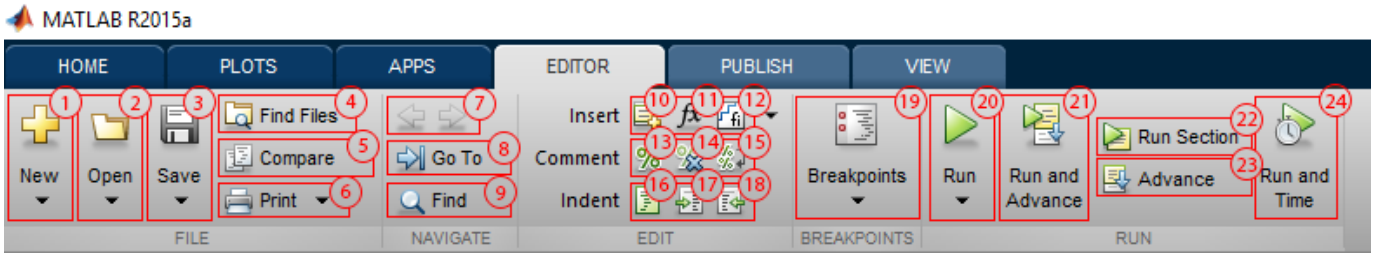
19 - *Acesso a ajuda do MATLAB.*

20 - *Acesso a comunidade online da Mathworks.*

21 - *Envia pedido de suporte técnico para a Mathworks.*

22 - *Add-ons adicionais incluindo suporte ao hardware.*

5.1.2. Editor



1 - *Cria um novo documento do MATLAB (.m).*

2 - *Abre um arquivo do MATLAB (.m) já existente.*
Atalho: Ctrl + O

3 - *Salva o arquivo MATLAB (.m) que está sendo executado.*

4 - *Procura por um arquivo baseado no nome ou no conteúdo.*
Atalho: Ctrl + Shift + F

5 - *Compara o conteúdo de dois arquivos distintos.*

6 - *Imprime o documento.*

8 - *Movimenta o cursor para uma linha, função ou seção.*

9 - *Encontra determinado texto no código e opcionalmente pode-se substituí-lo.*

10 - *Quebra e inicia nova seção.*

11 - *Abre uma função anteriormente criada.*
Atalho: Shift + F1

12 - *Constrói um objetivo numérico de ponto fixo.*

13 - *Insere um símbolo de comentário no texto.*
Atalho: Ctrl + R

14 - *Retira um símbolo de comentário do texto.*
Atalho: Ctrl + T

15 - *Permite formatar comentários organizadamente.*
Atalho: Ctrl + J

16 - *Retorno o texto para a posição inicial.*
Atalho: Ctrl + I

17 - *Executa um avanço do texto para a direita.*
Atalho: Ctrl +]

18 - *Executa um avanço do texto para a esquerda.*
Atalho: Ctrl + [

19 - *Mostra opções para pausar a execução do código.*

20 - *Salva e executa o código do script.*

21 - *Executa a seção atual do código e avança para a próxima seção.*

22 - *Executa a seção atual do código.*

23 - *Avança para a próxima seção.*

24 - *Executa o código e mensura o tempo de execução para melhoria de performance.*

6. Variáveis

6.1. Declaração de Variáveis

O MATLAB, diferentemente de linguagens de programação como C, não requer declaração prévia das variáveis a serem utilizadas e nem seus tipos. Para declarar uma variável, basta utilizar uma letra ou palavra seguida do símbolo de atribuição '=' (igual). Feito isso, em seu workspace aparecerá a variável e o respectivo valor atribuído. Durante a declaração de variáveis, deve-se atentar a:

```
NOME DA VARIÁVEL


Deve começar com uma letra maiúscula ou minúscula

Caracteres permitidos: letras, números e underline
```

Nomes de variáveis possuem distinção em letras maiúsculas e minúsculas, dessa forma, 'var' e 'Var' representam variáveis e valores diferentes. Nomes de variáveis com mais de uma palavra podem ser separados com '_' (*underline*).

Exemplos

```
>>  var = 161.11

var =

     161.1100

>>  Var = 26

Var =

     26

>>  VaR = 333

VaR =
```

```
     333

>>  var_a = 0.1

var_a =

     0.1000
```

6.2. Variáveis reservadas

ans	Variável onde são guardados os resultados de operações não atribuídas a uma variável específica - *ans*, diminutivo de *answer*.
pi	Valor de $\pi = 3.1416$.
eps	Unidade de arredondamento da máquina, i.e., o menor valor que adicionado a 1 representa um número maior que 1
Flops	Contador do número de operações efetuadas. Estamos a falar de operações em vírgula flutuante.
Inf	Representa $+\infty$.
NaN	Not-a-Number, símbolo que representa $0/0$ ou outra indeterminação matemática.
i,j	Representa o valor da $\sqrt{-1}$.
clock	Exibe a hora atual em um vetor linha de seis elementos contendo ano, mês, dia, hora, minuto e segundo.

7. Operações Aritméticas

O MATLAB possui todas as operações básicas da matemática podendo ser utilizado como uma simples calculadora. Na tabela abaixo se encontram alguns exemplos.

Tabela 1.1 - Operações aritméticas entre dois escalares

Operação	Forma Algébrica	MATLAB
Adição	a + b	a + b
Subtração	a - b	a - b
Multiplicação	a x b	a*b
Divisão pela Direta	a ÷ b	a/b
Divisão pela Esquerda	b ÷ a	a\b
Exponenciação	a^b	a^b
Raiz Quadrada	$\sqrt{a}$	sqrt(a)
Fatorial	a!	factorial(a)

Exemplos

```
>>  x=5
x =
     5
```

```
>>  2+1
ans =
     3
```

```
>>  7-9
ans =
    -2
```

```
>>  7*2
ans =
    14
```

```
>>  11/2
ans =
    5.5000
```

```
>>  3^3
ans =
    27
```

7.1. Hierarquia das Operações Aritméticas

Sabendo que várias operações podem ser combinadas em uma simples expressão aritmética, é importante conhecer a ordem nas quais as operações serão executadas. A tabela 1.2 contém a ordem de prioridade das operações aritméticas no MATLAB.

Tabela 1.2 - Hierarquia das operações aritméticas	
Prioridade	**Operação**
1ª **2ª** **3ª** **4ª**	Parênteses Exponenciação, esquerda à direita Multiplicação e Divisão, esquerda à direita Adição e Subtração, esquerda à direita

Exemplo

```
>>  a = 5;
>>  b = 5;
>>  c = 5;
>>  x = -b+sqrt(b^2-4*4*a*c)/2*a;
ans =
     25.8388
>>  x = (-b+sqrt(b^2-4*4*a*c))/(2*a);
ans =
     0.9136
```

Na primeira operação para x a expressão escrita foi:

$$x = -b + \frac{\sqrt{b^2 - 4 \cdot a \cdot c}}{2} \cdot a$$

Na segunda operação para x a expressão escrita foi:

$$x = \frac{-b + \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

8. Rotinas ou arquivos M-Files

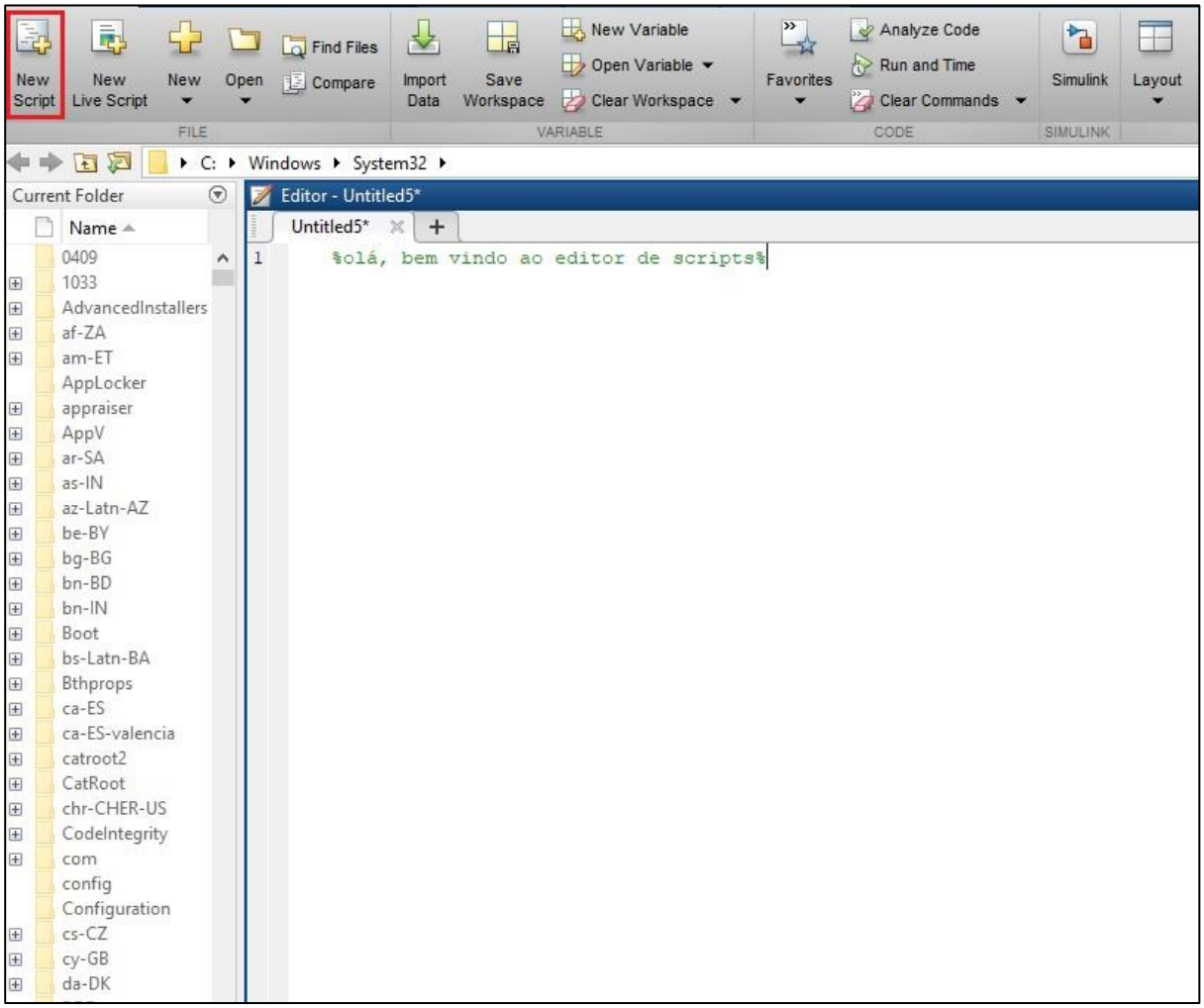
Todos os comandos do MATLAB são executados na janela *Command Window*. Embora cada comando do MATLAB possa ser executado desse modo, não é conveniente usar a janela *Command Window* para executar muitos comandos em série, especialmente se eles estiverem encadeados logicamente entre si, isto é, constituírem um programa. Muitas vezes, tentar resolver um problema no MATLAB digitando muitos comandos consecutivos pode ser difícil ou até mesmo impossível. Além disso, a janela *Command Window* não é interativa. Significa que cada vez que a tecla *Enter* é pressionada, somente o último comando digitado é executado, sendo que o restante do trabalho se torna inalterável. Uma maneira diferente de executar os comandos no MATLAB é primeiro criar um arquivo com uma lista de comandos, salvá-la, e então, rodar o arquivo. Os arquivos criados com esse propósito são denominados *rotinas, M-files* ou *script files*.

8.1. Características dos *Scripts*

- Quando um *script* é executado, o MATLAB executa os comandos na ordem em que eles foram escritos.
- *Scripts* que possuem comandos de saída (ex.: exibição de gráfico), essa saída é mostrada na *Command Window*.
- *Scripts* são convenientes pois podem ser editados e também serem executados várias vezes.
- *Scripts* devem ser salvos antes de serem executados.

8.2. Criando e salvando um *Script*

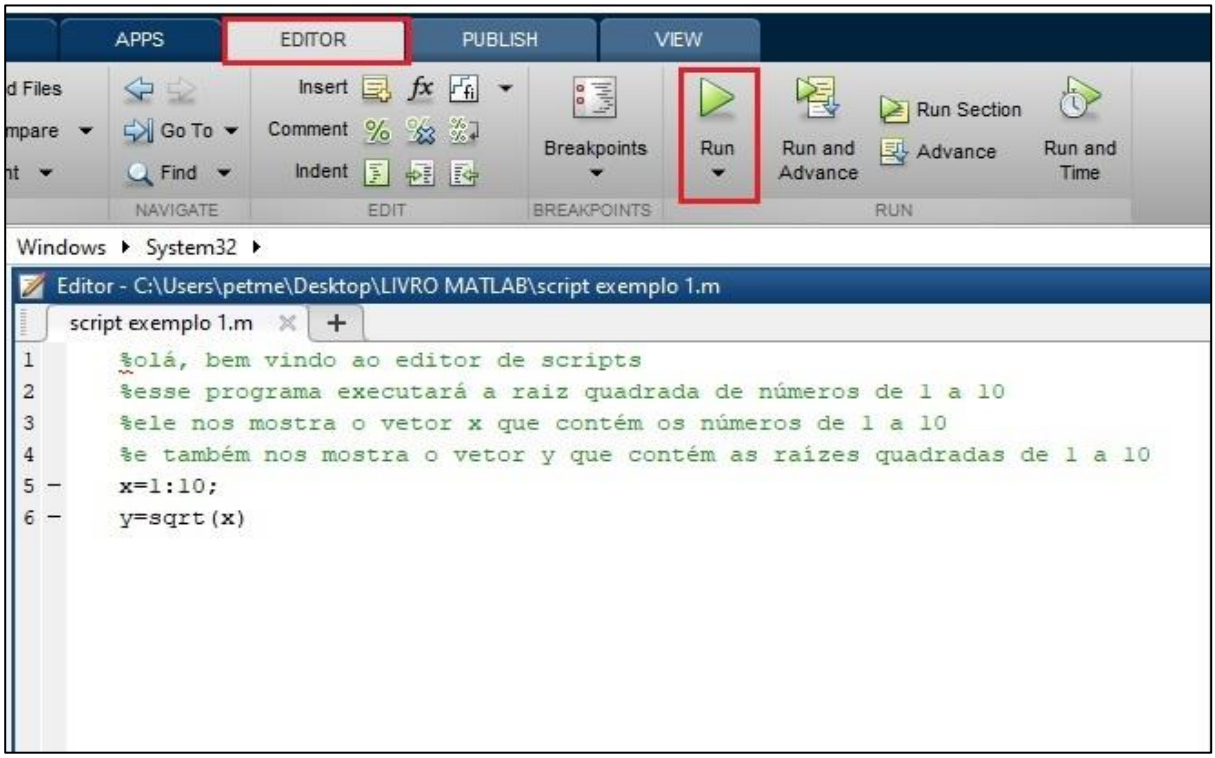
Para criar um novo *script* basta clicar no botão "*New Script*" no canto superior esquerdo do software. A janela de edição do *script* aparecerá acima da *Command Window*. Veja o exemplo a seguir:



Depois de aberto o editor de *scripts,* os comandos devem ser digitados linha por linha. Também é possível escrever as linhas de programação em outro editor de texto e colar no editor.

8.3. Executando um *script*

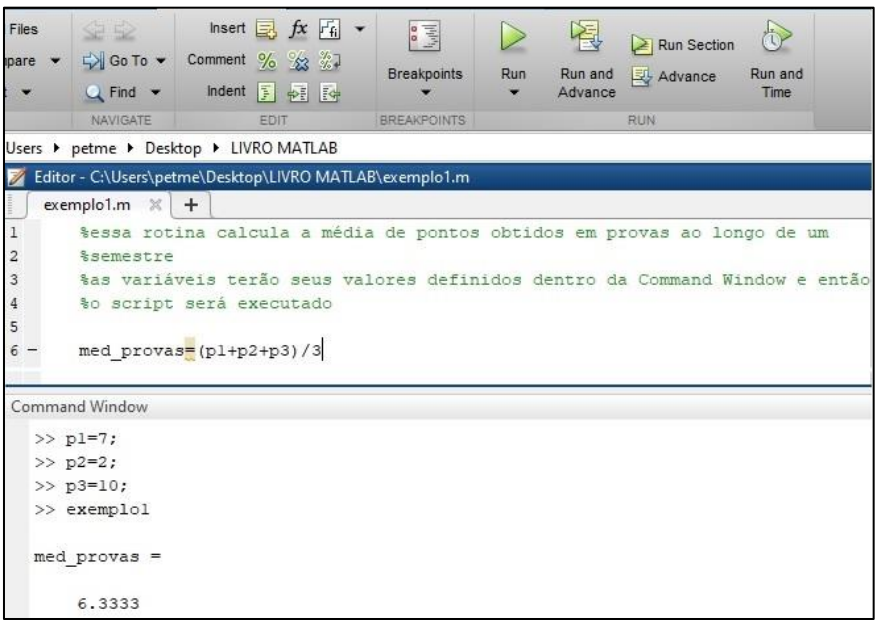
Para executarmos um *script* devemos clicar na aba "*Editor*" e em seguida basta clicar no botão "*Run*". O MATLAB requer que o *script* esteja salvo antes de executar. O exemplo mostra como executar um *script*



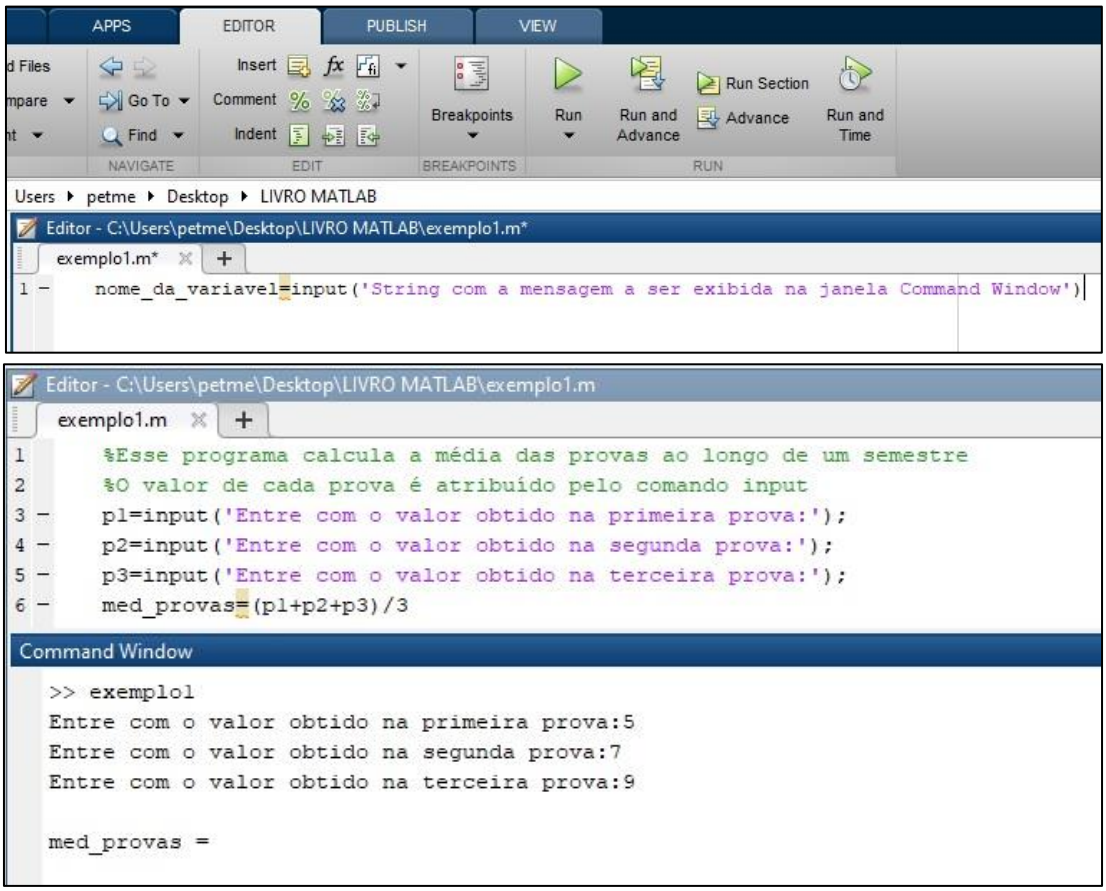
7

8.4. Definindo variáveis nos *scripts*

Os *scripts* podem ser executados a partir de variáveis que foram definidas no *Command Window* como mostra o exemplo a seguir.



As variáveis também podem ser declaradas no editor de *script* e terem seus valores atribuídos na *Command Window* como ilustra o exemplo a seguir:



Como o exemplo sugere, a função input irá retornar ao usuário uma ação, onde entre parênteses é dito entre aspas o que será exibido na tela como pedido; caso a função input não seja associada a uma variável específica o conteúdo inserido pelo usuário será armazenada na variável ***ans.***

- **Input**

Comando: variável = input('Pedido feito ao usuário escrito aqui');

Descrição: Exibe na tela o texto desejado e aguarda que o usuário entre com o valor a ser atribuído para aquela variável após apertar a tecla *Enter.*

Exemplo

```
>>  x=input('Digite o valor x:')
```

```
Digite o valor de x:10

      x =


    10
```

9. Operadores Lógicos e Relacionais e Comandos de Fluxo

9.1. Operadores Relacionais

Assim como na matemática, as linguagens de programação (incluindo a do MATLAB) possuem operadores relacionais, que servem para fazer a comparação de dois dados. No caso do MATLAB, é possível realizar a comparação de duas matrizes de mesmo número de linhas e colunas ou para comparar uma matriz e um escalar. Os operadores relacionais são:

Operador	Significado
<	menor que
<=	menor ou igual
>	maior que
>=	maior ou igual
==	Igual
~=	diferente (não igual)

Ao utilizar um operador relacional, você realiza uma "pergunta" dando como opções de resposta ***verdadeiro*** ou ***falso***. Em caso de *verdadeiro*, o resultado da operação é 1. Em caso de *falso*, o resultado é 0. Ou

seja, se a operação a < b for realizada, ela retornará 1 se a for menor que b ou retornará 0 se a for maior ou igual a b.

No MATLAB, como trabalhamos com matrizes, quando uma operação relacional for realizada ela resultará uma matriz com o resultado da comparação elemento a elemento.

Exemplo

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

A =

     1     2     3
     4     5     6

>> B = [0 6 3; 7 6 6]

B =

     0     6     3
     7     6     6

>> c = 4

c =

     4

>> A < B

ans =

     0     1     0
     1     1     0

>> A >= c

ans =

     0     0     0
     1     1     1
```

9.2. Operadores Lógicos

É possível combinar duas ou mais comparações (operações relacionais) utilizando os operadores lógicos. Os operadores lógicos são:

Operador	Significado
&	E
|	OU
~	NÃO

- **E**

Quando duas expressões forem combinadas com E (&) seu resultado só será 1 (*verdadeiro*) se o resultado das duas expressões individualmente for 1 (verdadeiro). Caso uma delas seja 0, o resultado da operação & é 0 (*falso*).

Combinação com E	Falso	Verdadeiro
Falso	0	0
Verdadeiro	0	1

- **OU**

O resultado combinação de duas expressões utilizando com OU (|) será 1 se pelo menos o resultado individual de uma das expressões for 1. OU é 0 apenas se as duas expressões combinadas forem 0.

Combinação com OU	Falso	Verdadeiro
Falso	0	1
Verdadeiro	1	1

- **NÃO**

O papel da operação NÃO (~) é inverter o resultado de toda a expressão. Ou seja, se o resultado da expressão for 1, utilizar a operação NÃO a transforma para 0, e vice-versa.

Exemplos

```
>> A = 1;
>> B = 2;
>> C = 4;

>> A > B & C > B

ans =

     logical

     0

>> A > B | C > B

ans =

     logical


     1
```

9.3. Comandos de Fluxo

Servem para desviar o fluxo natural do código, ou seja, executar ou não uma ação dependendo de uma condição ou executar uma mesma ação repetidas vezes.

9.3.1. For

O *for* é um comando de loop é utilizado quando se quer realizar uma série de comandos por um número de vezes fixo e predefinido.

A estrutura básica do for é:

for i = a:b

<comandos>

end

Isso significa que, começando uma contagem de i = a até i = b, ele executará o conjunto de comandos listados entre *for* e *end*. O comando *end* é utilizado para determinar onde termina o conjunto de comandos desse loop.

Exemplo

```
for i=1:5
    for j = 1:5
        A(i,j) = i + j;
    end
end


A =


     2     3     4     5     6
     3     4     5     6     7
     4     5     6     7     8
     5     6     7     8     9
     6     7     8     9    10


>>
```

9.3.2. If / ElseIf / Else

O operador **if-elseif-else** serve para executar um bloco de comandos se (**if**) uma condição for verdadeira. Senão se (**elseif**) uma outra condição for verdadeira executa outro bloco de comandos, senão (**else**) outro bloco de comandos será executado.

A estrutura básica do if-elseif-else é:

```
if <condição>

<comandos1>

elseif <condição>

<comandos1>

else

<comandos2>

end
```

A estrutura **elseif <condição>** é opcional, podendo inclusive ser utilizada inúmeras vezes enquanto o **if** e **else** só podem ser inseridos uma única vez onde apenas o **if** é obrigatório.

Suponhamos que a condição seja a < b. Com isso, se a for menor que b (resultado da comparação = 1), o bloco de comandos 1 é executado. Porém se a for maior ou igual a b (resultado da comparação = 0), então o bloco de comandos 2 é executado.

Exemplo

```
a = rand(1,1)
if a < 0.4
    A = linspace(0,70,8)

elseif 0.4 <= a & a < 0.7
    A = linspace(-70,7,8)

else
    A = linspace(1,50,8)
end

a =

    0.8491


A =
```

```
1.     8    15    22    29    36    43    50
```

9.3.3. While

O loop **while** é executado enquanto o resultado de condição predeterminada for verdadeiro.

A estrutura básica do while é:

```
while <condição>

<comandos>

end
```

Se a condição for a <b, enquanto a condição a < b estiver sendo satisfeita (resultado = 1), o bloco de comandos entre while e end será executado.

Exemplo

```
a = 1;
b = 10;
i = 1;
while b >= a
A(i,i) = i^2;
b = b-i;
i = i+1;
end
A =

     1     0     0     0
     0     4     0     0
     0     0     9     0
     0     0     0    16
```

Também é possível criar um loop infinito com condição de parada, se o número 1 representa uma condição verdadeira, fazer:

```
while 1
```

<comandos>

end

Criará um loop infinito; podemos então usar o comando **break** para interromper o loop quando uma condição de parada específica for atingida, uma possibilidade é fazer

while 1

<comandos>

if <condição>

break

end

Está estrutura pode ser bem útil, porém tome cuidado, pois se a condição de parada nunca for atingida você cairá em loop infinito.

Exemplo

```
c = 0;
while 1
   c
   if c == 4;
        break
   end
   c = c + 1;
end
c =

     0
c =

     1
c =

     2
c =

     3
c =
```

4

Exercícios de fixação

1 - Preencha uma matriz de dimensões 6x7 cujos termos sejam dados pelas seguintes funções:

$$A_{ij} = 2i - 3j \; se \; i \geq j$$

$$A_{ij} = \sqrt{7i^2 + j^2} \; se \; i < j$$

2 - Faça um script que calcula a média de $N > 0$ provas e receba o número de aulas e o número de faltas. O programa deve exibir se o aluno foi aprovado ou reprovado. Os critérios de aprovação são: a média das provas deve ser maior ou igual a 6 e a frequência deve ser de 70%. Exiba na tela se a pessoa foi aprovada ou reprovada, caso tenha sido reprovada, indique o motivo.

3 - Escreva um algoritmo que receba um vetor com N números inteiros e imprima qual a porcentagem de números divisíveis por 3 e 2 simultaneamente.

4 - O seu programa deve receber dois vetores tridimensionais de entrada e indicar:
• Se os vetores são ortogonais.
• Se os vetores são paralelos.
• Calcular um vetor unitário que seja ortogonal aos dois simultaneamente, se eles não forem paralelos.

10. Formatos Numéricos

No MATLAB, os números podem ser exibidos em diferentes formatos e isso torna sua utilização adequada a aplicação desejada, ao utilizar uma formatação específica todos os valores a partir deste instante serão exibidos de acordo com a nova formatação, está alteração ocorre apenas para a exibição, sendo que o conteúdo armazenado na variável permanece inalterado.

Para a exibição de valores em notação científica temos que ***e*** indica a base decimal, logo e^{-1} significará 10^{-1}.

- ***short***

  Comando: format short

  Descrição: Exibe o número com até 5 algarismos. É o formato padrão do MATLAB.

  Exemplo

```
>> format short

>> preco = 1/3


preco =


    0.3333
```

- ***long***

  Comando: format long

  Descrição: Exibe o número com até 16 algarismos.

Exemplo

```
>> format long

>> preco = 1/3


preco =

     0.333333333333333
```

- ***shorte%***

  Comando: format shorte%

  Descrição: Exibe um número em formato de notação científica com até 5 algarismos na mantissa.

  Exemplo

```
>> format shorte%

>> preco = 1/3


preco =

   3.3333e-01
```

- ***longe%***

  Comando: format longe%

  Descrição: Exibe um número em formato de notação científica com até 15 algarismos na mantissa.

Exemplo

```
>> format longe%

>> preco = 1/3


preco =

     3.333333333333333e-01
```

- ***hex***

  Comando: format hex

  Descrição: Exibe um número em formato hexadecimal.

  Exemplo

```
>> format hex

>> preco


preco =

   3fd5555555555555
```

- ***bank***

  Comando: format bank

  Descrição: Exibe um número com duas casas decimais.

Exemplo

```
>> format bank

>> preco


preco =

          0.33
```

- ***rat***

  Comando: format rat

  Descrição: Exibe um número em formato de fração.

  Exemplo

```
>> format rat

>> preco = 1/3


preco =

       1/3
```

Exercícios de fixação:

1 - Resolva as expressões básicas (OBS: adequar parênteses aos locais necessários):

a) $\frac{5+8\cdot 3-2}{4^3\cdot 3}$

b) $\sqrt{10-2\cdot 32\div 13}-\frac{1}{4}\cdot 10$

c) $2^2+\sqrt{5-4\cdot 1}$

2 - Dada a equação do segundo grau $5 \cdot x^2 + 20 \cdot x + 10 = 0$, obtenha as soluções da equação utilizando a fórmula de Bhaskara.

3 - Dada a expressão $1 + 11{,}5 \cdot \frac{1}{17}$, declare seu resultado nos seguintes formatos:

- format long
- format shot
- format hex

4 - A expressão que modela o crescimento da população brasileira pode ser escrita como $P(t) = \frac{157273000}{(1+e^{-0.0313 \cdot (t-1913.25)})}$, onde t é dado em anos. Determine o valor da população para o ano atual (adote e=2.718).

11. Funções

11.1. Comandos básicos

- ***Limpar Command Window***

  Comando: clc

  Descrição: Executa a limpeza de todos os comandos já executados na *Command Window.*

- ***Limpar Variável***

  Comando: clear nome_variável

  Descrição: Apaga a variável e o valor que foi anteriormente atribuído a ela.

- ***Limpar todas as variáveis***

  Comando: clear all

  Descrição: Apaga todas as variáveis armazenadas no *Workspace*.

- ***Ajuda***

  Comando: help nome_comando

Descrição: Exibe na *Command Window* uma explicação relativa ao funcionamento do comando desejado. Caso seja digitado apenas *help,* o MATLAB exibirá tópicos de ajuda na janela de comandos.

Exemplo

```
>> help sin

 sin - Sine of argument in radians

      This MATLAB function returns the sine of the elements of X.

      Y=sin(X)
```

```
    Reference page for sin

    See also asin, asind, sind, sinh.

    Other uses of sin

        fixedpoint/sin, symbolic/sin
```

- ***Procurar por***

  Comando: lookfor comando

  Descrição: Procura por funções relacionadas diretamente com a palavra que foi digitada.

  Exemplo

```
>> lookfor inverse

ifft        - Inverse discrete Fourier transform.

ifft2       - Two-dimensional inverse discrete Fourier transform.

ifftn       - N-dimensional inverse discrete Fourier transform.

ifftshift   - Inverse FFT shift.

acos        - Inverse cosine, result in radians.

acosd       - Inverse cosine, result in degrees.

acosh       - Inverse hyperbolic cosine.

acot        - Inverse cotangent, result in radian.

acotd       - Inverse cotangent, result in degrees.

acoth       - Inverse hyperbolic cotangent.

acsc        - Inverse cosecant, result in radian.

acscd       - Inverse cosecant, result in degrees.
```

```
acsch      - Inverse hyperbolic cosecant.

asec       - Inverse secant, result in radians.

asecd      - Inverse secant, result in degrees.

asech      - Inverse hyperbolic secant.

asin       - Inverse sine, result in radians.

asind      - Inverse sine, result in degrees.

asinh      - Inverse hyperbolic sine.

atan       - Inverse tangent, result in radians.

atan2      - Four quadrant inverse tangent.

atan2d     - Four quadrant inverse tangent, result in degrees.

atand      - Inverse tangent, result in degrees.

atanh      - Inverse hyperbolic tangent.

invhilb    - Inverse Hilbert matrix.

ipermute   - Inverse permute array dimensions.

inv        - Matrix inverse.
```

- ***Variáveis do Workspace***

  Comando: who

  Descrição: Exibe quem são as variáveis atualmente armazenadas no *Workspace*.

  Exemplo

```
>> who

 your variables are:
```

```
a b
```

- ***Variáveis detalhadas***

  Comando: whos

  Descrição: Exibe quem são as variáveis atualmente armazenadas no *Workspace*, bem como sua dimensão, número de bytes e classe da variável.

  Exemplo

```
>> whos
 Name        Size  Bytes      Class
     a       1x1    8         double
     b       2x3    48        double
```

- ***Salvar variáveis do Workspace***

  Comando: save nome_ficheiro

  Descrição: Permite salvar as variáveis contidas atualmente no *Workspace* em formato binário ou formato ascii. Caso não seja especificado o nome do arquivo, as variáveis serão salvas em um arquivo matlab.mat. Também é possível realizar esse procedimento clicando no ícone *Save Workspace* da barra *Home*.

- ***Carregar variáveis do Workspace***

  Comando: load nome_ficheiro

  Descrição: Permite abrir no *Workspace* variáveis que foram anteriormente salvas em um arquivo. Caso não seja especificado o nome do arquivo, o arquivo aberto será o matlab.mat. É possível carregar as variáveis utilizando o botão *Import Data* da barra *Home*.

SALVAR E CARREGAR VARIÁVEIS

Para utilizar os comandos save e load é necessário executar o MATLAB como administrador. Caso contrário, haverá um erro de permissão.

```
Error using save

Unable to write file teste.mat: permission denied.
```

11.2. Funções elementares

11.2.1. Funções trigonométricas

- **Seno, cosseno e tangente**

Comando: sin(x), cos(x) e tan(x)

Descrição: Calcula o valor do seno/cosseno/tangente de um número x, sendo que o valor de x deve ser dado em radianos.

Exemplo

```
>>  x = pi/3;

>>  sin(x)

ans =

    0.8660
```

Também é possível calcular o valor do seno/cosseno/tangente de um número x em graus.

Comando: sind(x), cosd(x) e tand(x)

Exemplo

```
>>  x = 60;

>>  sind(x)

ans =

      0.8660
```

- **Secante, cossecante e cotangente**

Comando: sec(x), csc(x) e cot(x)

Descrição: Calcula o valor da secante/cossecante/cotangente de um número x, sendo que o valor de x deve ser dado em radianos.

Exemplo

```
>>  x = pi/6;

>>  sec(x)

ans =

      1.1547
```

Novamente é possível calcular o valor da secante/cossecante/cotangente de um número x em graus.

Comando: secd(x), cscd(x) e cotd(x)

Exemplo

```
>>  cotd(90)

ans =

      0
```

- **Seno, cosseno e tangente hiperbólicos**

Comando: sinh(x), cosh(x) e tanh(x)

Descrição: Calcula o valor do seno/cosseno/tangente hiperbólico de um número x, sendo que o valor de x deve ser dado em radianos.

Exemplo

```
>>  x = pi/3;

>>  sinh(x)

ans =

     1.2494
```

- **Funções de arco**

Comando: asin(x), acos(x), atan(x), asec(x), acsc(x), acot(x); asinh(x), acosh(x), atanh(x), asech(x), acsch(x) e acoth(x).

Descrição: Retorna o valor do ângulo (em radianos), cuja função trigonométrica vale x.

Exemplo

```
>>  x = 0.8860;

>>  asin(x)

ans =

     1.0471
```

- **Converter radiano para grau**

Comando: radtodeg(x)

Descrição: Executa a conversão do valor do ângulo x dado em radianos para um valor em graus.

Exemplo

```
>>  radtodeg(pi/3)

ans =

      60.000
```

- **Converter grau para radiano**

Comando: degtorad(x)

Descrição: Executa a conversão do valor do ângulo x dado em graus para um valor em radianos.

Exemplo

```
>>  degtorad(180)

ans =

      3.1416
```

11.2.2. Funções exponenciais

- **Função exponencial de base euleriana**

Comando: exp(x)

Descrição: Executa uma operação de potenciação, cuja base é o número de Euler e o expoente é x (e^x).

Exemplo

```
>>  exp(2) %e^2%

ans =

      7.3891
```

- **Função logarítmica de base euleriana**

  Comando: log(x)

  Descrição: Calcula o logaritmo de x na base do número de Euler (ln x).

  Exemplo

```
>>  log(exp(2)) %ln(e²)%

ans =

     2
```

- **Função logarítmica de base comum/decimal**

  Comando: log10(x)

  Descrição: Calcula o logaritmo de x na base 10.

  Exemplo

```
>>  log10(1000)
ans =
     3
```

- **Função logarítmica de base qualquer**

  Comando: logβ(x)

  Descrição: Calcula o logaritmo de x na base β que você escolherá de acordo com suas necessidades.

  Exemplo

```
>>  log2(8)
```

```
ans =
      3
```

- **Raiz quadrada**

  Comando: sqrt(x)

  Descrição: Calcula a raiz quadrada de x.

  Exemplo

```
>>  sqrt(64)
ans =
      8
```

11.2.3. Funções numéricas

- **Arredondamento para o mais próximo**

  Comando: round(x)

  Descrição: Arredonda um número decimal para o inteiro mais próximo.

  Exemplo

```
>>  round(1.43)
ans =
      1
```

- **Arredondamento para cima**

  Comando: ceil(x)

  Descrição: Arredonda um número decimal para o inteiro acima.

  Exemplo

```
>>  ceil(1.5)

ans =

     2
```

- **Arredondamento para baixo**

  Comando: floor(x)

  Descrição: Arredonda um número decimal para o inteiro abaixo.

  Exemplo

```
>>  floor(1.5)

ans =

     1
```

- **Resto da divisão**

  Comando: rem(x, y)

  Descrição: Obtém-se o valor do resto da divisão de x por y.

  Exemplo

```
>>  rem(8,5)

ans =

     3
```

- **Fatorial**

Comando: factorial(x)

Descrição: Retorna o valor do fatorial de x.

Exemplo

```
>>  factorial(5)

ans =

      120
```

Exercícios de fixação:

1 - Escreva um código para obtenção da área de um triângulo retângulo com a hipotenusa valendo 15 e um dos ângulos agudos 30°.

2 - Após realizar o arredondamento dos números decimais e das frações, resolva a expressão:

$$11.3 + \frac{35}{3} - 12.5 + \frac{52}{3} - 16.2 + 17$$

3 – A aproximação de uma função seno pela série de Maclaurin pode ser dada pela equação:

$$sen(x) \cong \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

Obtenha uma aproximação do seno de 45° pela série de Maclaurin desprezando os termos onde n>4.

OBS: se atentar a utilização aos locais em que o ângulo deve ser utilizado em radianos.

4 - Resolva (para t = 5 e 10):

- $\ln(t^2 + t + 2)$
- $e^{t\cdot(1+\cos(3\cdot t))}$

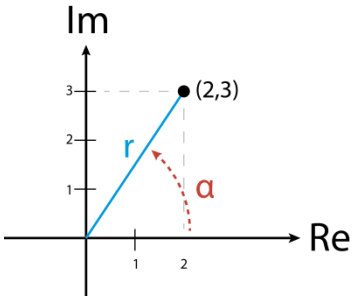
- $sec^2(t) + \cot(t) - 1$

12. Números complexos

A definição de números complexos pode ser feita de diferentes maneiras no MATLAB. Serão abordadas algumas delas nessa apostila:

12.1. Coordenadas cartesianas e polares

Sabe-se que números complexos podem ser representados em um plano cartesiano, onde eixo x irá conter a parte real desse número e o eixo y irá conter sua parte imaginária.



Essa representação pode ser feita tanto em coordenadas cartesianas, onde são dados os valores de x e y, ou coordenadas polares, onde a coordenada será dada por um ângulo (α) e um raio (r) em relação à origem.

Coordenadas retangulares para polares:

$$r = \sqrt{x^2 + y^2} \qquad \alpha = tan^{-1}\left(\frac{y}{x}\right)$$

Coordenadas polares para retangulares:

$$x = r \cdot \cos(\alpha) \qquad y = r \cdot sen(\alpha)$$

Um número complexo pode ser definido no MATLAB utilizando tanto a letra i como a letra j ou ainda a raiz de -1.

Exemplo

```
>> x = 3+4i

x =

   3.0000 + 4.0000i

>> y = 5+3j

y =

   5.0000 + 3.0000i

>> z = 3+sqrt(-1)

z =

   3.0000 + 1.0000i
```

As operações aritméticas básicas envolvendo números complexos devem ser escritas da mesma forma que as operações envolvendo números reais.

Exemplo

```
>> c1 = 3+2i

c1 =

   3.0000 + 2.0000i

>> c2 = 5-i

c2 =
```

```
   5.0000 - 1.0000i

>> c1+c2

ans =

   8.0000 + 1.0000i

>> c3 = 4

c3 =

     4

>> c1-c2/c3

ans =

   1.7500 + 2.2500i

>> i^2

ans =

    -1
```

Abaixo serão mostradas algumas funções utilizadas para estabelecer relações entre a representação algébrica (cartesiana) e a representação polar:

- **Valor absoluto (módulo)**

  Comando: abs(x)

  Descrição: Retorna o valor absoluto de um número complexo ou de um vetor de números complexos.

Exemplo

```
>>  x = [1+2i 2+3i]

x =

     1.0000 + 2.0000i      2.0000+3.0000i


>>  abs(x)

ans =

     2.2361      3.6056
```

- **Ângulo de fase**

  Comando: angle(x)

  Descrição: Retorna o valor do ângulo de fase (em radianos) de um número complexo.

Exemplo

```
>>  x = [3 + 2i]

x =

     3.0000 + 2.0000i

>>  angle(x)

ans =

     0.5880
```

- **Conjugado**

  Comando: conj(x)

  Descrição: Retorna o valor do complexo conjugado de um dado número complexo x.

Exemplo

```
>>  conj(1+4i)

ans =

     1.0000 - 4.0000i
```

- **Valor imaginário**

  Comando: imag(x)

  Descrição: Retorna o valor da parte imaginária de um dado número complexo x.

Exemplo

```
>>  imag(4+5i)

ans =

     5
```

- **Valor real**

  Comando: real(x)

  Descrição: Retorna o valor da parte real de um dado número complexo x.

Exemplo

```
>>  real(4+5i)

ans =
```

```
    4
```

Exercícios de fixação:

1 - Escreva os seguintes números complexos na forma polar:

- $\sqrt{3} + i$
- $4 - 3i$
- $5i$

2 - Escreva os seguintes números complexos na forma cartesiana:

- $\rho = 10\ e\ \theta = 135°$
- $\rho = 5\ e\ \theta = 240°$
- $\rho = 1\ e\ \theta = 200°$

3 – Resolva a equação $x^4 + 10 \cdot x^2 + 20 = 0$ e caso a mesma possua raízes complexas represente-as nas formas cartesiana e polar.

13. Matrizes

13.1. Vetores e Matrizes

No MATLAB, a declaração de variáveis do tipo vetor e matriz é feita utilizando-se uma letra ou palavra seguida do símbolo de atribuição 'igual' (=). Contudo, a identificação de um vetor ou matriz é dada pela utilização de colchetes que delimitam seus elementos.

Para esses tipos de variáveis, os elementos de uma mesma linha são separados por espaço simples ou por vírgula (,). Enquanto elementos de linhas diferentes são separados por ponto e vírgula (**;**). É possível **no *script*** a utilização do *Enter* após cada ponto e vírgula para melhorar a visualização dos elementos da matriz, de forma mais semelhante a maneira como que escrevemos.

Exemplos

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

M =

      1    2    3

      4    5    6

      7    8    9


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

N =

      1    2    3

      4    5    6

      7    8    9
```

13.2. Matrizes geradas por comandos

Alguns tipos de vetores e matrizes podem ser gerados a partir de comandos, conforme abaixo:

- **Geração de vetor por incremento**

Comando: x = valor inicial:espaçamento:valor final

Descrição: Para criar um vetor por incremento, basta declarar a variável e determinar o primeiro elemento, qual é o incremento até chegar ao último elemento e o último elemento, separando estes parâmetros entre si utilizando dois pontos (:), conforme o exemplo a seguir:

Caso o seu espaçamento seja unitário, o termo intermediário pode ser omitido, fazendo:

x = valor inicial:valor final

Exemplo

```
>>  y = 1:0.1:2

y =

Colums 1 through 4

     1.0000   1.1000   1.2000   1.3000

Colums 5 through 8

     1.4000   1.5000   1.6000   1.7000

Colums 9 through 11

     1.8000   1.9000   2.0000


>>  z = 1:5

z =

     1   2   3   4   5
```

O MATLAB também nos permite criar vetores a partir de outros vetores. No exemplo a seguir, o vetor "y" será criado a partir do vetor x.

Exemplo

```
>> x = [pi*0.1 pi*0.2 pi*0.3 pi*0.4 pi*0.5 pi*0.6 pi*0.7 pi*0.8
pi*0.9]

x =

    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
2.1991    2.5133    2.8274

>> y=sin(x)

y =
```

```
    0.3090    0.5878    0.8090    0.9511    1.0000    0.9511
0.8090    0.5878    0.3090
```

- ***linspace***

Comando: linspace(primeiro_elem, último_elem, número_elem)

Descrição: O comando *linspace* cria um vetor utilizando como parâmetros o primeiro elemento, o último elemento e a quantidade de elementos no vetor, onde estes parâmetros ficam separados entre si por virgulas, conforme o exemplo:

Exemplo

```
>>  x = linspace(1,100,8)

x =

Colums 1 through 5

     1.0000   15.1529   29.2557   43.4286   57.5714

Colums 6 through 8

     71.7143   85.8571   100.0000
```

- ***rand***

Comando: rand(linhas, colunas)

Descrição: Cria um vetor, ou matriz, com elementos randômicos entre 0 e 1.

Exemplo

```
>>  a = rand(3,4)

a =

     0.0350   0.1529   0.2557    0.5559
```

```
    0.7543    0.4571    0.0430    0.8557

    0.6162    0.8571    0.1760    0.9311
```

- ***magic***

  Comando: magic(n)

  Descrição: Cria uma matriz *(n* x *n)* com números inteiros de 1 a n^2 em ordem aleatória, onde a soma entre elementos da diagonal principal, secundária, da 1ª linha, 1ª coluna, n-ésima linha e n-ésima colunas são todas iguais entre si.

Exemplo

```
>>  magic(3)

ans =

     8    1    6

     3    5    7

     4    9    2
```

- ***zeros***

  Comando: zeros(linhas,colunas)

  Descrição: Cria uma matriz ou um vetor onde todos os elementos valem 0.

Exemplo

```
>>  zeros(3,4)

ans =

     0    0    0    0
```

```
     0     0     0     0

     0     0     0     0
```

- ***ones***

  Comando: ones(linhas,colunas)

  Descrição: Cria uma matriz ou um vetor onde todos os elementos valem 1.

Exemplo

```
>>  ones(4,3)

ans =

     1     1     1

     1     1     1

     1     1     1

     1     1     1
```

- ***eye***

  Comando: eye(n)

  Descrição: Cria uma matriz identidade com a dimensão *(n x n).*

Exemplo

```
>>  eye(4)

ans =

     1     0     0     0

     0     1     0     0
```

```
     0    0    1    0

     0    0    0    1
```

- ***pascal***

  Comando: pascal(n)

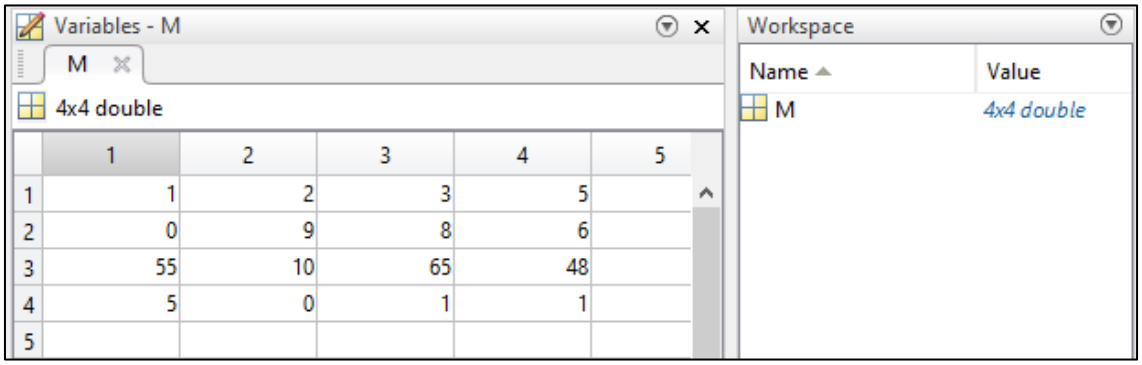
  Descrição: Cria uma matriz composta pelo triângulo de pascal com dimensão *(n* x *n).*

Exemplo

```
>>  pascal(5)

ans =

     1    1     1     1     1

     1    2     3     4     5

     1    3     6    10    15

     1    4    10    20    35

     1    5    15    35    70
```

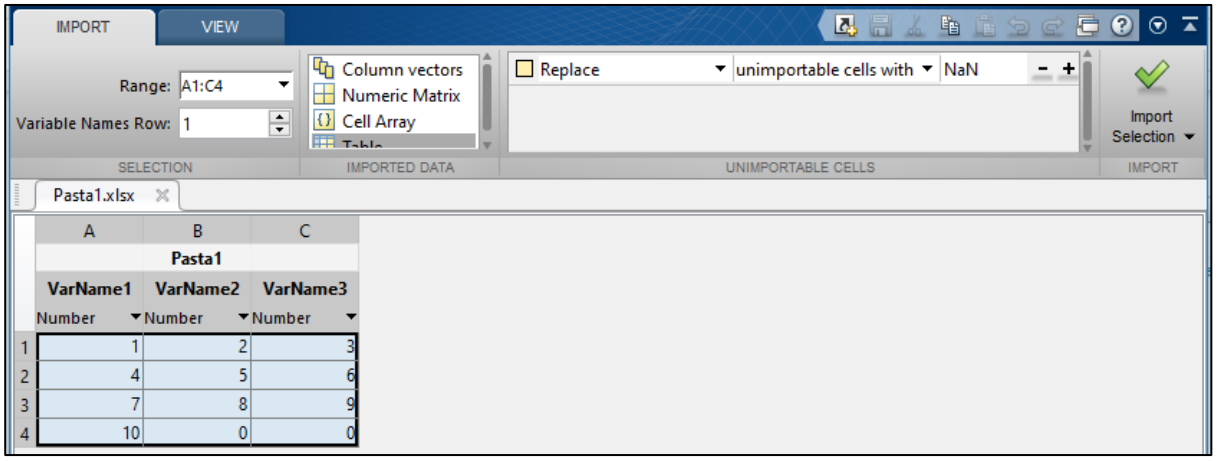
13.3. Recursos para criação de variáveis

É possível também utilizar o recurso "*New Variable*", que se encontra na barra de tarefas do MATLAB, o qual abre uma planilha dentro do MATLAB, semelhante ao ambiente do Excel, na qual você pode digitar os valores dos elementos de uma matriz.



	1	2	3	4	5
1	1	2	3	5	
2	0	9	8	6	
3	55	10	65	48	
4	5	0	1	1	
5					

Dados também podem ser importados utilizando o recurso "*Import Data*". É possível escolher o tipo de saída do dado importado, sendo ele um vetor com as colunas dos dados importados, uma matriz com todos os dados, tabela ou *cell array*.



	A	B	C
		Pasta1	
	VarName1	VarName2	VarName3
	Number	Number	Number
1	1	2	3
2	4	5	6
3	7	8	9
4	10	0	0

13.4. Elementos da matriz

É possível alterar e adicionar elementos específicos de uma matriz usando a referência da localização do elemento entre parênteses.

Exemplo:

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

A =

     5     3     4
```

```
     6     4     8
     9     9     9

>> A(2,2) = 25

A =

     5     3     4
     6    25     8
     9     9     9
```

O comando A(2,2) = 25 tem por função alterar o elemento da 2ª linha e 2ª coluna de A para 25.

Semelhantemente, é possível adicionar um elemento à matriz, alterando seu tamanho. Dessa forma, os elementos da nova linha ou coluna que não forem preenchidos serão 0.

Exemplo

```
>> A

A =

     5     3     4
     6     4     8
     9     9     9

>> A(4,3)=6

A =

     5     3     4
     6    25     8
     9     9     9
     0     0     6

>> A(3,4)=5

A =

     5     3     4     0
```

```
     6    25     8     0
     9     9     9     5
     0     0     6     0
```

Se o elemento adicionado estiver na diagonal principal, será criada uma linha e uma coluna com zeros.

Exemplo

```
>> A(5,5)=1

A =

     5     3     4     0     0
     6    25     8     0     0
     9     9     9     5     0
     0     0     6     0     0
     0     0     0     0     1
```

O comando *B(:,n)* referência os elementos da matriz *B* em todas as linhas da coluna *n*, pois ao inserirmos dois pontos (:) no parâmetro que indica, estaremos nos referenciando a todo o conteúdo daquele parâmetro.

Exemplo

```
B =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1

>> B(:,4)=3

B =

     1     0     0     3     0
```

```
     0     1     0     3     0
     0     0     1     3     0
     0     0     0     3     0
     0     0     0     3     1
```

O comando *B(n,:)* referência os elementos da matriz *B* em todas as colunas da linha *n*.

Exemplo

```
>> B(5,:)=3

B =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     3     3     3     3     3
```

Também podemos realizar fatiamentos em matrizes usando dois pontos e dois valores, um antes e outro após os dois pontos, **a:b** indica um fatiamento de **a** até **b.**

O comando *B(:,m:n)* referência os elementos da matriz *B* em todas as linhas **entre** as colunas *m* e *n*.

Exemplo

```
>> B(:,2:4)=7

B =

     1     7     7     7     0
     0     7     7     7     0
     0     7     7     7     0
     0     7     7     7     0
     0     7     7     7     1
```

O comando *B(m:n,:)* referência os elementos da matriz *B* entre as linhas *m* e *n*, em todas as colunas.

Exemplo

```
>> B(1:3,:)=12

B =

    12    12    12    12    12
    12    12    12    12    12
    12    12    12    12    12
     0     0     0     1     0
     0     0     0     0     1
```

O comando *B(m:n,p:q)* referência os elementos da matriz *B* em todas as colunas entre as linhas *m* e *n*, e as colunas entre as linhas *p* e *q*.

Exemplo

```
B =
     1     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0
     0     0     1     0     0     0     0     0
     0     0     0     1     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     1

>> B(1:3,5:8)=6

B =
     1     0     0     0     6     6     6     6
     0     1     0     0     6     6     6     6
```

```
     0     0     1     0     6     6     6     6
     0     0     0     1     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     1
```

13.5. Funções com matrizes

No MATLAB é possível realizar algumas funções com as matrizes. Dentre as inúmeras existentes, é possível destacar as seguintes:

- **Transposição**

  Comando: ctranspose(M)

  Descrição: O comando *ctranspose* realizará a transposição da matriz desejada

Exemplo

```
>> A = [2 1 12; 6 9 4; 1 7 6]

A =

     2     1    12
     6     9     4
     1     7     6

>> ctranspose(A)

ans =

     2     6     1
     1     9     7
    12     4     6
```

- **'**

Comando: A'

Descrição: O uso do apóstrofo ao lado da matriz realizará a transposição da matriz desejada, assim como o *ctranspose*.

Exemplo

```
>> A'

ans =

     2     6     1
```

```
     1     9     7
    12     4     6
```

- **Determinante**

Comando: det(A)

Descrição: Essa função calcula o determinante da matriz desejada, lembre que é apenas possível calcular determinante de matrizes quadradas.

Exemplo

```
>> det(A)

ans =

   416
```

- **Inversa**

  Comando: inv(A)

  Descrição: Essa função nos retorna a matriz inversa da matriz desejada, caso seja possível.

Exemplo

```
>> inv(A)

ans =

    0.0625    0.1875   -0.2500
   -0.0769    0.0000    0.1538
    0.0793   -0.0313    0.0288
```

- **Autovalores e Autovetores**

Comando: eig(A)

Descrição: A função eig retorna os **autovalores** e **autovetores** da matriz. Se utilizada apenas *eig(M)*, a fução retorna um vetor com os **autovalores**. Se utilizada na forma *[P, D] = eig(M)*, **D** será a matriz diagonal com os autovalores e a matriz **P** será a matriz dos autovetores associados ao respectivos autovalores de **D**.

Exemplo

```
>> A = [3 0 0; 0 3 2; 0 -1 0]

A =

     3     0     0
     0     3     2
     0    -1     0

>> eig(A)

ans =

     1
     2
     3

>> [P,D]=eig(A)

P =

         0         0    1.0000
    0.7071   -0.8944         0
   -0.7071    0.4472         0


D =

     1     0     0
     0     2     0
     0     0     3
```

- **Polinômio Característico**

Comando: poly(A)

Descrição: A função *poly*, quando utilizada com matrizes quadradas, retorna um vetor linha com os coeficientes do polinômio característico, proveniente da equação $det\ (\lambda I\ -\ A) = 0$.

Exemplo

```
>> A = [3 0 0; 0 3 2; 0 -1 0];

>> poly(A)

ans =

     1    -6    11    -6
```

14. Operações com matrizes

14.1. Adição e Subtração

Considerando duas matrizes A e B com as mesmas dimensões, a adição no MATLAB é feita utilizando o comando A+B. Assim, cada elemento da matriz A é somado ao elemento de posição correspondente na matriz B.

Exemplo

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

A =

      1    2    3

      4    5    6


>>  B=[6 5 4;3 2 1]

B =

      6    5    4

      3    2    1


>>  A+B

ans =

      7    7    7

      7    7    7
```

Semelhantemente, a subtração é realizada com o comando A-B.

Exemplo

```
>>  A-B

ans =

     -5    -3    -1

      1     3     5
```

14.2. Multiplicação entre matrizes

Na multiplicação matricial, os elementos da matriz resultante do produto de duas matrizes A e B é igual ao somatório do produto dos elementos da i-ésima linha de A com os termos da j-ésima coluna de B. Para que a operação A*B possa ser efetuada, o número de colunas da matriz A deve ser igual ao número de linhas da matriz B. Vale lembrar que essa operação não é comutativa, ou seja A*B≠B*A.

Exemplo

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

A =

     1    2    3

     4    5    6

>>  B = [1 2; 3 4; 5 6]

B =

     1     2
     3     4
     5     6

>> A*B
ans =

    22    28
    49    64
```

14.3. Divisão

Na sintaxe do MATLAB, o resultado da linha de comando A/B é igual ao produto de A pela inversa de B à direita de A, utilizando a barra "\" indicaria para nós que B seria multiplicado pela inversa de A à esquerda de B. Isso nos leva a uma possibilidade para resolver sistemas lineares conhecendo a matriz dos coeficientes do sistema e o vetor de termos independentes.

Considerando um sistema A*x = b, a divisão à esquerda (x = A\b) é equivalente a achar o vetor solução do sistema.

Dica de memorização: pense na barra da divisão e na contra barra da seguinte maneira, se a for barra "/", está tombada para a direita logo você estará multiplicando pela inversa da matriz a direita, se for a contra barra "\" ela está tombada para a esquerda, logo será realizada a multiplicação pela matriz inversa à esquerda).

Exemplo

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

A =

     2     3     5
     8     5     4
     1     5     1

>> b = [1 ; 9 ; 12]

b =

     1
     9
    12

>> x = A\b

x =

    0.2030
    2.6541
```

```
   -1.4737
```

Antes de realizar as referidas operações de divisão, deve-se verificar se A e B cumprem requisitos tais como terem número de linhas e coluna coerentes com as operações de multiplicação que estão implícitas, A ou B ser invertível.

A divisão à direita (x = A/b) equivale a encontrar o vetor solução de um sistema x*A = b. Dessa forma, A e b devem conter o mesmo número de colunas.

Exemplo

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

A =

     1     2     3
     3     4     5
     6     7     8

>> b = [3 3 6]

b =

     3     3     6

>> x = A/b

x =

    0.5000
    0.9444
    1.6111
```

14.4. Transposição de Matrizes

A operação de transpor a matriz serve para trocar linhas por colunas, ou seja, será uma matriz tal que $Bij = Aji$. Seja **A** uma matriz, fazendo **B** = **A'**, teremos que **B** será **A** transposta.

Exemplo

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

A =

     1     2     3
     4     5     6
     7     8     9


>> B = A'

B =

     1     4     7
     2     5     8
     3     6     9
```

14.5. Elemento a elemento

Operações com arrays **significa realizar operações elemento a elemento com as matrizes**, ou seja, cada elemento a_{ij} de uma matriz A se relacionará com um elemento b_{ij} da matriz B correspondente. Dessa forma, as operações devem ser realizadas com matrizes de mesmas dimensões.

As adições e subtrações, como já são realizadas elementos a elemento, são realizadas normalmente: A+B e A-B.

A multiplicação elemento a elemento é realizada utilizando a indicação de **ponto** antes do símbolo de multiplicação na forma A.*B.

Exemplo

```
>> A = rand(3,4)

A =

    0.7094    0.6797    0.1190    0.3404
    0.7547    0.6551    0.4984    0.5853
    0.2760    0.1626    0.9597    0.2238
```

```
>> B = rand(3,4)

B =

    0.7513    0.6991    0.5472    0.2575
    0.2551    0.8909    0.1386    0.8407
    0.5060    0.9593    0.1493    0.2543

>> A.*B

ans =

    0.5329    0.4752    0.0651    0.0877
    0.1925    0.5836    0.0691    0.4920
    0.1397    0.1560    0.1433    0.0569
```

Da mesma forma, é possível realizar divisão e potenciação utilizando A./B e A.^B

Exemplo

```
>> A

A =

    0.7094    0.6797    0.1190    0.3404
    0.7547    0.6551    0.4984    0.5853
    0.2760    0.1626    0.9597    0.2238

>> B

B =

    0.7513    0.6991    0.5472    0.2575
    0.2551    0.8909    0.1386    0.8407
    0.5060    0.9593    0.1493    0.2543
```

```
>> A./B

ans =

    0.9442    0.9723    0.2175    1.3218
    2.9585    0.7353    3.5951    0.6962
    0.5456    0.1695    6.4285    0.8802

>> A.^B

ans =

    0.7726    0.7634    0.3120    0.7577
    0.9307    0.6860    0.9080    0.6374
    0.5214    0.1751    0.9939    0.6834
```

14.6. Operações com Vetores

Podemos interpretar os vetores como matrizes do tipo linha ou coluna. Dessa forma é possível realizar as mesmas operações as quais eram realizadas com matrizes e outras operações básicas realizadas com vetores em $\mathbb{R}^2$ e $\mathbb{R}^3$.

- **Adição**

  Comando: $v + k$

Exemplo

```
>> v

v =

     1     2     3     4     5
>> k

k =

     2     4     6     8    10

>> v+k

ans =     3     6     9    12    15
```

- **Subtração**

  Comando: k - v

Exemplo

```
>> v

v =

     1     2     3     4     5
>> k

k =

     2     4     6     8    10

>> k-v

ans =

     1     2     3     4     5
```

- **Multiplicação (elemento a elemento)**

*Comando: v.*k*

Descrição: Descrição: Sendo *v* e *k* dois vetores de mesma dimensão, esse comando nos retornará a multiplicação do primeiro elemento de *v* pelo primeiro elemento de *k* e assim por diante até o n-ésimo termo de *v* multiplicado pelo n-ésimo termo de *k*.

Exemplo

```
>> v

v =

     1     2     3     4     5
>> k

k =

     2     4     6     8    10

>> v.*k

ans =

     2     8    18    32    50
```

- **Divisão (elemento a elemento)**

Comando: v./k

Descrição: Sendo *v* e *k* dois vetores de mesma dimensão, esse comando nos retornará a divisão do primeiro elemento de *v* pelo primeiro elemento de *k* e assim por diante até o n-ésimo termo de *v* dividido pelo n-ésimo termo de *k*.

Exemplo

```
>> v

v =

     1     2     3     4     5
>> k

k =

     2     4     6     8    10

>> v./k
ans =

    0.5000    0.5000    0.5000    0.5000    0.5000
```

- **Potenciação (elemento a elemento)**

  Comando: v.^2

  Descrição: Sendo *v* e *k* dois vetores de mesma dimensão, esse comando nos retornará a potenciação do primeiro elemento de *v* pelo primeiro elemento de *k* e assim por diante até o n-ésimo termo de *v* elevado pelo n-ésimo termo de *k*.

Exemplo

```
>> v

v =

     1     2     3     4     5
>> k

k =

     1     2     3     4     5

>> v.^v

ans =
```

```
     1     4     27     256     3125
```

- **Módulo**

Comando: norm(v)

Descrição: Esse comando nos retorna a norma (módulo) de um vetor

Exemplo

```
>> v

v =

     1     2     3     4     5

>> norm(v)

ans =

    7.4162
```

- **Produto Vetorial**

  Comando: cross(r ,f)

  Descrição: Esse comando nos retorna o produto vetorial de dois vetores. Observe que para que esse comando seja realizado, os vetores devem ter **3 dimensões obrigatoriamente**, vale lembrar também que o produto vetorial não é comutativo portanto cross(r, f) ≠ cross(f, r).

Exemplo

```
>> r
```

```
>> v

v =

     1     2     3     4     5

>> r

r =

     1     2     3

>> f

f =

     4     5     6

>> cross(r,f)

ans =

    -3     6    -3

>> cross(v,f)

Error using cross (line 25)
A and B must be of length 3 in the dimension in which the cross
product is taken.
```

- **Produto Escalar ou Produto Interno**

Comando: dot(r,f)

Descrição: Tal comando nos retorna o produto escalar (produto interno) dos vetores desejados. Observe que para o produto escalar não temos a restrição de dimensões como o *cross,* apenas a exigência que os vetores a serem multiplicados sejam de mesma dimensão.

Exemplo

```
>> r

r =

     1     2     3

>> f

f =

     4     5     6


>> dot(r,f)

ans =

    32
```

- **Soma interna**

Comando: sum(v)

Descrição: O comando *sum(v)* retorna a soma dos elementos de *v*

Exemplo

```
>> v

v =

     1     2     3     4     5

>> sum(v)

ans =

    15
```

- **Maior valor de um vetor**

Comando: max(v)

Descrição: O comando *max(v)* retorna o elemento de maior valor de *v*

Exemplo

```
>> v

v =

     1     2     3     4     5

>> max(v)

ans =

     5
```

- **Mínimo valor de um vetor**

Comando: min(v)

Descrição: Analogamente, o comando *min(v)* retorna o elemento de menor valor do vetor *v*

Exemplo

```
>> v

v =

     1     2     3     4     5

>> min(v)

ans =

     1
```

- **Raiz quadrada**

  Comando: sqrt(v)

  Descrição: Esse comando nos retorna a raíz quadrada de cada componente do vetor desejado

Exemplo

```
>> v

v =

     1     2     3     4     5

>> sqrt(v)

ans =

    1.0   1.4142    1.7321    2.0000    2.2361
```

- **Ordem crescente**

  Comando: sort(j)

  Descrição: Esse comando nos retorna a ordem crescente do vetor desejado

Exemplo

```
>> j = [9 1 3 0 12 15]

j =

     9     1     3     0    12    15

>> sort(j)

ans =

     0     1     3     9    12    15
```

- **Ordem decrescente**

Comando: -sort(-j)

*Descrição: P*ara ordenarmos os valores de um vetor *j* em ordem decrescente deve-se usar um macete no comando *sort(j),* fazendo *-sort(-j)*. Fazer -*sort(-j)* é como se fizéssemos *sort(-j)* o que ordenaria de ordem crescente todos os elementos de j multiplicados por - 1, e multiplicaríamos tudo depois por -1 para voltar os sinais ao seu valor original, obtendo assim a ordem decrescente dos elementos de j .

Exemplo

```
>> j = [9 1 3 0 12 15]

j =

     9     1     3     0    12    15

>> -sort(-j)

ans =

    15    12     9     3     1     0
```

- **Transposição de vetores**

Comando: ctranspose(v) ou v'

Descrição: A transposição de um vetor pode ser realizada de maneira semelhante como realizada para uma matriz

Exemplo

```
>> v

v =

     1     2     3     4     5

>> v'

ans =

     1
     2
     3
     4
     5
```

- **Dimensões de um vetor**

Comando: size(v)

Descrição: O comando *size(v)* nos retorna as dimensões do vetor *v* (linhas, colunas). Como todo vetor é uma matriz, esse comando também se aplica a matrizes

Exemplo

```
>> A

A =

     2     4
     5     8

>> size(A)

ans =

     2     2

>> v
```

```
v =

     1     2     3     4     5


>> size(v)

ans =

     1     5
```

- **Comprimento de um vetor**

  Comando: length(v)

  Descrição: O comando *length(v)* nos retorna a quantidade de elementos do vetor *v*. Se esse comando for usado para matrizes ele nos retornará a quantidade de colunas dessa matriz

Exemplo

```
> length(v)

ans =

     5
```

Exercícios de fixação

1 - Crie um vetor coluna, onde o primeiro elemento é 15, o último -25 e o decremento vale -5 (o vetor coluna pode ser criado como o vetor transposto do vetor linha).

2 - Crie um vetor coluna contendo 12 elementos espaçados igualmente, onde o primeiro elemento é -1 e o último é -15.

3 - Construa a matriz indicada abaixo usando notação vetorial para declarar vetores cujos elementos sejam espaçados igualmente e/ou o comando *linspace* para entrar com as linhas.

$$B = \begin{bmatrix} 1 & 4 & 7 & 10 & 13 & 16 & 19 & 22 & 25 \\ 72 & 66 & 60 & 54 & 48 & 42 & 36 & 30 & 24 \\ 0 & 0.125 & 0.250 & 0.375 & 0.500 & 0.625 & 0.750 & 0.875 & 1.000 \end{bmatrix}$$

4 - Declare a seguinte matriz *A*:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 \\ 21 & 18 & 15 & 12 & 9 & 6 & 3 \\ 5 & 10 & 15 & 20 & 25 & 30 & 35 \end{bmatrix}$$

a)Crie uma matriz *B* (*3x4*) a partir dos seguintes elementos de *A*: primeira, terceira e quarta linhas e primeira, terceira, quinta e sétima colunas.

b)Crie um vetor linha *u* de 15 elementos a partir dos seguintes elementos de A: terceira linha, quinta coluna e sétima coluna.

5 - Utilizando o comando *eye*, crie a matriz *A* abaixo (à esquerda). Então, usando o operador dois pontos para referenciar elementos no arranjo, modifique a matriz *A* de forma que seus elementos assumam os valores mostrados na matriz *A* (à direita)

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad A = \begin{bmatrix} 2 & 2 & 2 & 0 & 5 & 5 & 5 \\ 2 & 2 & 2 & 0 & 5 & 5 & 5 \\ 3 & 3 & 3 & 0 & 5 & 5 & 5 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 4 & 4 & 7 & 0 & 9 & 9 & 9 \\ 4 & 4 & 7 & 0 & 9 & 9 & 9 \\ 4 & 4 & 7 & 0 & 9 & 9 & 9 \end{bmatrix}$$

6 - Crie uma matriz *M*(*3x3*) onde todos os elementos sejam 1 e uma matriz *B(2x2)* onde todos os elementos sejam 5. Em seguida, adicione elementos a matriz *A* por anexação da matriz *B*, tal que a matriz *A* transforme-se em:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 5 & 5 \end{bmatrix}$$

7 - O coeficiente de atrito (μ) pode ser determinado experimentalmente medindo o módulo da força necessária para mover uma massa m sobre uma superfície horizontal plana com atrito. Quando F é medida e sendo conhecidos os valores de m, o coeficiente de atrito cinético pode ser determinado por (g=9.81 m/s^2):

$$\mu = \frac{F}{m \cdot g}$$

Dada a tabela abaixo, determine o coeficiente de atrito para cada medida e armazene o resultado em um vetor.

Medida	1	2	3	4	5	6
Massa m (kg)	2	4	5	10	20	50
Força F (N)	12.5	23.5	30	61	117	294

15. Polinômios

No Matlab, um polinômio é representado por um vetor-linha contendo os coeficientes do polinômio em ordem decrescente. Por exemplo, o polinômio de $-x^5 - 5x^4 + 8x^2 + 30$ é representado pelo vetor p = [-1 -5 0 8 0 30].

15.1. Raízes de um polinômio

Comando: roots(p)

Descrição: Esse comando nos retorna as raízes do polinômio desejado, uma vez que seja declarado um vetor com os coeficientes desse polinômio.

Exemplo

```
>> p = [-1 -5 0 8 0 30]; %esse vetor representa o polinô-
mio −x^5−5x^4 + 0x^3 + 8x^2 + 0x − 30%
>> r = roots(p)

r =

  -4.5416 + 0.0000i
  -2.2503 + 0.0000i
   1.6741 + 0.0000i
   0.0589 + 1.3229i
   0.0589 - 1.3229i
```

- **Determinando um polinômio por suas raízes**

  Comando: poly(r)

  Descrição: É possível também fazer o caminho inverso. Ou seja, a partir das raízes, encontrar o polinômio. Quando esta função é utilizada com vetores, seja o vetor do tipo linha ou coluna, e que contenha as raízes do polinômio, a função *poly* retorna um vetor linha com os coeficientes do polinômio que possui essas raízes.

Exemplo

```
>> poly(r)

ans =

    1.0000    5.0000    0.0000   -8.0000    0.0000  -30.0000
```

*OBS: Caso apareça um sinal de negativo (-), na frente do número zero, por exemplo -0.0000 basta ignorar este sinal.

15.2. Produto e Divisão de Polinômios

Comando: conv(p1,p2)

Descrição: Se tivermos dois polinômios p1 e p2, o produto desses polinômios pode ser calculado com essa função.

Exemplo

```
>> p1 = [1 2 -3]

p1 =

     1     2    -3

>> p2 = [-1 5]

p2 =

    -1     5

>> conv(p1,p2)

ans =
```

```
    -1     3    13   -15
```

Nesse caso, os polinômios $p_1(x) = x^2 + 2x - 3$ e $p_2(x) = -x + 5$, quando multiplicados, resultam no polinômio $p(x) = -x^3 + 3x^2 + 13x - 15$, conforme o código acima demonstra.

Comando: deconv(p1,p2)

Descrição: É possível também realizar a divisão de polinômios utilizando essa função onde *p1* é polinômio que será dividido *p2* seu respectivo polinômio divisor.

Exemplo

```
>> p1 = [1 2 -3];

>> p2 = [-1 5];

>> [q, r] = deconv(p1,p2)

q =

    -1    -7


r =

     0     0    32
```

A função *deconv* retorna dois polinômios: *q* é o quociente e *r* o resto. Neste caso, o quociente da divisão é $q(x) = -x - 7$ e o resto é $r(x) = 32$.

15.3. Avaliação de Polinômios

Consideremos a função polinomial $f(x) = 2x^4 - 5x^3 + 8x^2 - 10x + 40$. Para avaliar numericamente uma função polinomial pode-se fazê-lo de duas formas: se o valor da variável independente x for um escalar

basta apenas escrever a equação na forma algébrica que o Matlab avalia a expressão e retorna o valor do polinômio:

Exemplo

```
>> x = 2;

>> f = 2*x^4 - 5*x^3 + 8*x^2 - 10*x + 40

f =

44
```

Se, no entanto, a variável x for um vetor contendo um intervalo de valores, então será necessário utilizar o operador ponto-escalar, como no exemplo:

Exemplo

```
>> x = 0:0.5:2;

>> f = 2*x.^4 - 5*x.^3 + 8*x.^2 - 10*x + 40

f =

40.0000 36.5000 35.0000 36.2500 44.0000
```

Comando: polyval(p,x)

Descrição: O segundo método consiste na utilização do vetor-linha *p* contendo os coeficientes do polinômio. Para avaliar numericamente o polinômio para um dado valor ou conjunto de valores *x*.

Exemplo

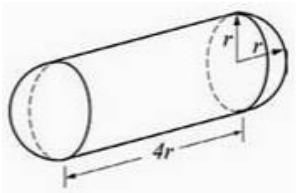
```
>> x = 0:0.5:2;

>> p = [2 -5 8 -10 40];

>> y = polyval(p,x)

y =

40.0000 36.5000 35.0000 36.2500 44.0000
```

Exercícios de fixação

1 - Avalie o polinômio $y = 1.5x^3 - 5x^2 + x + 2$ no domínio $-2 \leq x \leq 2$ com incremento de 0.1. Crie primeiramente um vetor x.

2 - Divida o polinômio $4x^4 + 6x^3 - 2x^2 - 5x + 3$ pelo polinômio $x^2 + 4x + 2$

3 - Um tanque de gás possui o formato de um cilindro com as extremidades hemisféricas. O raio do cilindro é r e o comprimento é $4r$. Determine r sabendo que $V = 0{,}85m^3$



16. Gráficos

16.1. Gráficos Bidimensionais

O MatLab se apresenta como uma ferramenta potente e eficaz quanto a geração de informações gráficas de até quatro variáveis.

A versatilidade está presente na formatação dos gráficos, onde dispomos de diversas ferramentas para alterar cor, forma, inserir texto, legendas, títulos, criar animações, vídeos, etc.

Vamos observar alguns comandos básicos para criação de gráficos:

- **Criar um gráfico**

Comando: figure

Esse comando não é necessário para criar um primeiro gráfico, mas importante para os demais, ele abre uma nova janela para plotar gráficos, isso significa que quando você for plotar o segundo gráfico em diante, pode-se fazer utilizando ou não o comando ***figure,*** se o figure não for utilizado, o próximo gráfico substituirá o anterior, utilizando o ***figure*** o gráfico será plotado em uma nova janela, mantendo os todos gráficos, daí a importância do figure.

- **Plotar o gráfico**

Comando: plot(x,y)

Descrição: Esse é o comando mais básico para gerar um gráfico que relacione os vetores x e y. Esses vetores devem possuir as mesmas dimensões, em plot(**x**, **y**) o argumento x é o vetor que contém os valores do domínio e y suas respectivas imagens.

OBS: Outros parâmetros podem ser incluídos em plots.

- **Identificação dos eixos**

Comando: xlabel('texto_X') e ylabel('texto_Y')

Descrição: Com esses comandos, você pode nomear os eixos x e y conforme o nome das variáveis que você estiver utilizando.

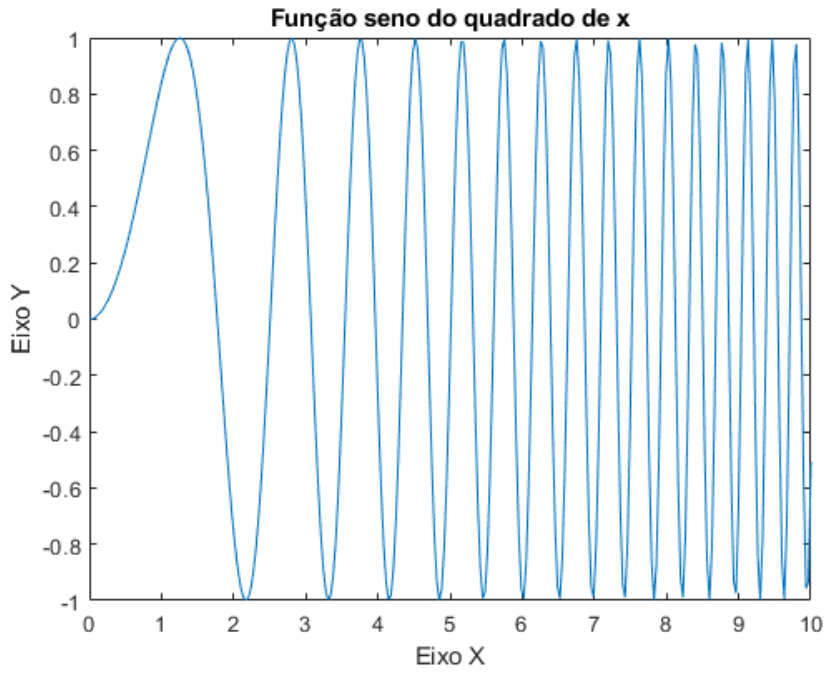
- **Título para o gráfico**

Comando: title('Texto do título')

Descrição: É utilizado para definir o título do gráfico.

Exemplo

```
>> x=linspace(0,10,300);
>> y=sin(x.^2);
>> figure
>> plot(x,y) %comando para plotar x e y
>> xlabel('Eixo X'); ylabel('Eixo Y'); %nomeando os eixos.
>> title('Função seno do quadrado de x'); %adicionando o título
```



- **Malha quadriculada**

Comando: grid , grid off

Descrição: Com o grid, você pode escolher exibir a malha quadriculada no fundo do gráfico, para ajudar a visualizar as retas paralelas aos eixos. Para esconde-la, utilize grid off;

***OBS:** O grid deve ser utilizado no código após o plot.

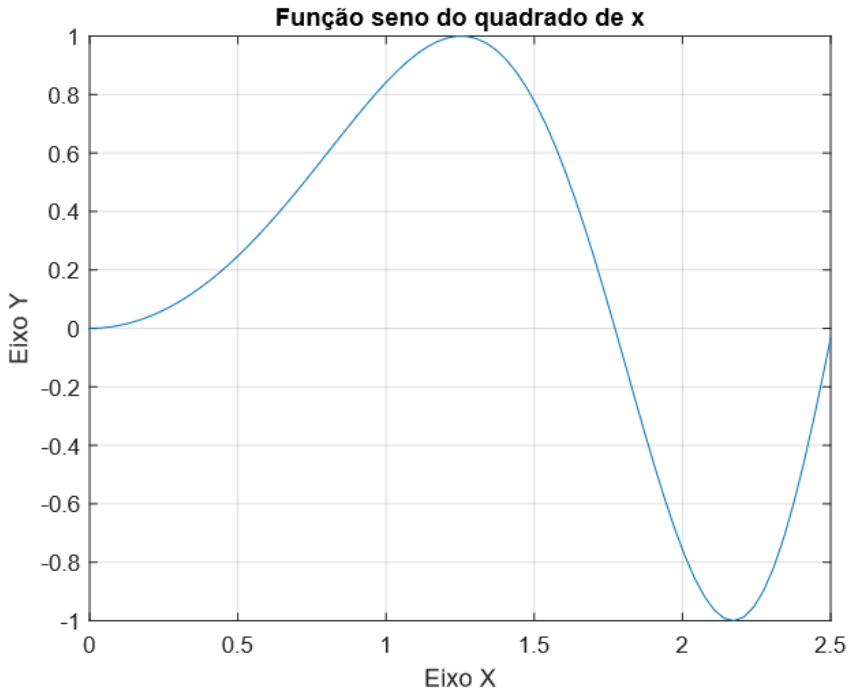
- **Definir limites dos eixos**

Comando: xlim([lim_inferior,lim_superior]); ylim([lim_inferior,lim_superior])

Descrição: Esse comando permite que você defina os limites dos eixos exibidos na tela do gráfico, vejamos como o exemplo acima fica utilizando como limite $0 \leq x \leq 2.5$ e $-1 \leq y \leq 1$.

Exemplo

```
>> x=linspace(0,10,300);
>> y=sin(x.^2);
>> figure
>> plot(x,y) %comando para plotar x e y
>> xlabel('Eixo X'); ylabel('Eixo Y'); %nomeando os eixos.
>> title('Função seno do quadrado de x'); %adicionando o título
>> xlim([0,2.5]);
>> ylim([-1,1]);
>> grid
```



- **Plotar vários gráficos na mesma janela**

Comando: hold on

Descrição: O hold on, mantém as linhas do gráfico que foram definidas em um comando plot, e faz com que os demais plots sejam então feitos na mesma janela.

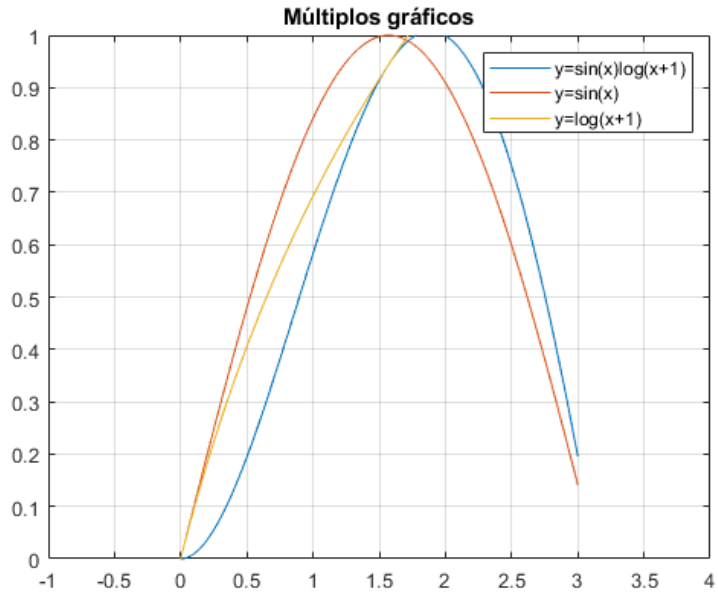
- **Gerar uma legenda para os múltiplos gráficos**

Comando: legend('legenda 1', 'legenda 2', 'legenda 3')

Descrição: Permite gerar uma legenda para os gráficos na janela de plotagem, onde a primeira legenda será atribuída ao primeiro gráfico plotado, a segunda legenda ao segundo gráfico e assim por diante..

Exemplo

```
>> x=0:0.01:3;

>> y=sin(x).*log(x+1);

>> plot(x,y)

>> grid;

>> xlim([-1 4]);

>> ylim([0 1]);

>> hold on %a partir daqui os demais plots serão feitos na mesma
janela

>> y2=sin(x);

>> y3=log(x+1);

>> plot(x,y2) %segundo plot

>> plot(x,y3) %terceiro plot

>> title('Múltiplos gráficos')

>> legend('y=sin(x)log(x+1)','y=sin(x)','y=log(x+1)') %legenda dos
3 gráficos
```



OBS: Se no exemplo anterior após printar na mesma janela os 3 gráficos, **y=sin(x)log(x+1), y=sin(x), y=log(x+1)** você precisasse de um 4º gráfico, mas deseja-se que ele ficasse em uma janela distinta dos 3 anteriores, basta escrever em seu código o comando **figure** antes de plotar o 4º gráfico.

- **Cor da linha**

  Comando: 'Color',parâmetro

Há três formas de se definir a cor de uma linha, ou de um elemento gráfico, pelo nome, pelo atalho, ou pelo array de fração de RGB.

RGB	Atalho	Nome
[1 1 0]	y	yellow
[1 0 1]	m	magenta
[1 0 1]	r	red
[0 1 0]	g	green
[0 0 1]	b	blue

Outros atalhos: cyan sem, 99mplí(w), black(k).

Funcionamento da fração RGB:

[.3 0 1] = 30% de vermelho, 0% de verde, 100% de azul

[.2 .5 1] = 20% de vermelho, 50% de verde, 100% de azul

[.9 .42 .2] = 90% de vermelho, 42% de verde, 20% de azul

Verifique o **Apêndice A** para obtenção da porcentagem da cor desejada.

Exemplo de código

```
%as três formas abaixo são equivalentes

>> plot(x,y,'color','green')

>> plot(x,y,'color','g')

>> plot(x,y,'color',[0 1 0])
```

- **Estilo da linha**

Comando: '–'

Descrição: Dentro do comando plot, você pode atribuir à linha de gráfico um estilo de linha ou marcador para representar os seus pontos, para alterar o estilo, os marcadores disponíveis estão listados:

'-'	linha cheia
'–'	linha tracejada
':'	linha pontilhada
'-.'	Linha traço-ponto

‘+’	marcador em cruz
‘o’	marcador circular
‘*’	marcador em asterisco
‘x’	marcador em x
‘square’ ou ‘s’	marcador quadrado
‘diamond’ ou ‘d’	marcador em diamante
‘^’	marcador triangular para cima
‘v’	marcador triangular para baixo
‘>’	marcador triangular para direita
‘<’	marcador triangular para esquerda
‘pentagram’ ou ‘p’	marcador estrela 5 pontas
‘hexagram’ ou ‘h’	marcador estrela de 6 pontas

Exemplo de código

```
%o plot abaixo terá linha da cor verde e linha tracejada

>> plot(x,y,'-','color','green')
```

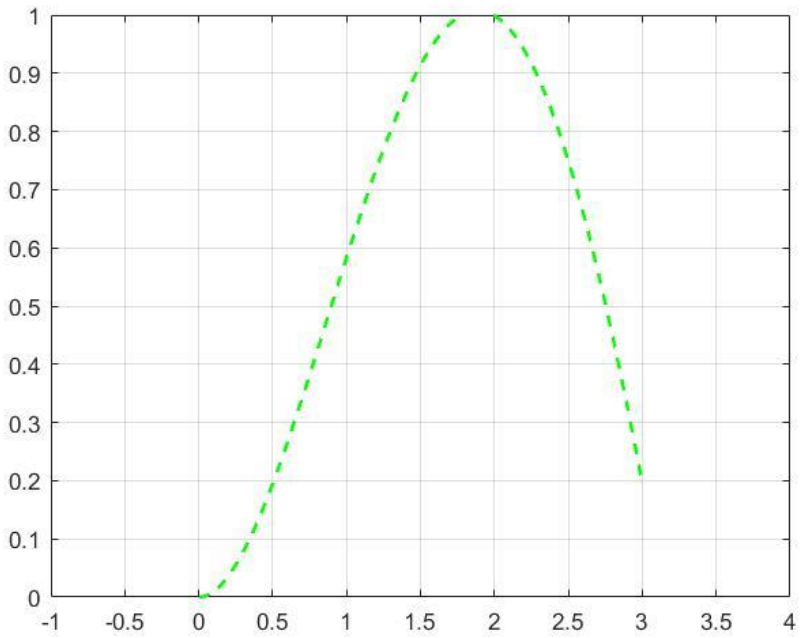
- **Peso da linha**

Comando: 'Linewidth', 3

Descrição: Altera a espessura da linha no gráfico, o número que sucede 'Linewidth' é o peso associado.

Exemplo

```
%o plot abaixo terá linha da cor verde e linha tracejada, e espes-
sura 50% maior que o padrão.

>> x=0:0.01:3;
>> y=sin(x).*log(x+1);
>> plot(x,y,'-','linewidth',1.5,'color','green')
```



- **Plotar várias janelas de gráfico na mesma figura**

Comando: subplot(m,n,p)

Descrição: Com esse comando, podemos plotar vários gráficos na mesma janela de figura. Os valores de *m* e *n* devem ser fixados, e designam a quantidade de gráficos que será mostrada na tela em que m é o número de gráficos por linha e n é o número de gráficos por coluna. O valor de p varia de acordo com a posição que é desejada mostrar da seguinte forma, **subplot(2,2, p)** onde $1 \leq \mathbf{p} \leq 4$ sendo que a contagem para posição p é feita variando as posições linha a linha como explicado no exemplo abaixo.

Exemplo

```
%os subplots abaixo são distribuídos em 2 linhas e 2 colunas

>> sub1 = subplot(2,2,1)
%posição linha 1 e coluna 1
>> sub1 = subplot(2,2,2)
%posição linha 1 e coluna 2
>> sub1 = subplot(2,2,3)
%posição linha 2 e coluna 1
>> sub1 = subplot(2,2,4)
%posição linha 2 e coluna 2
```

- **Mudar cor do background**

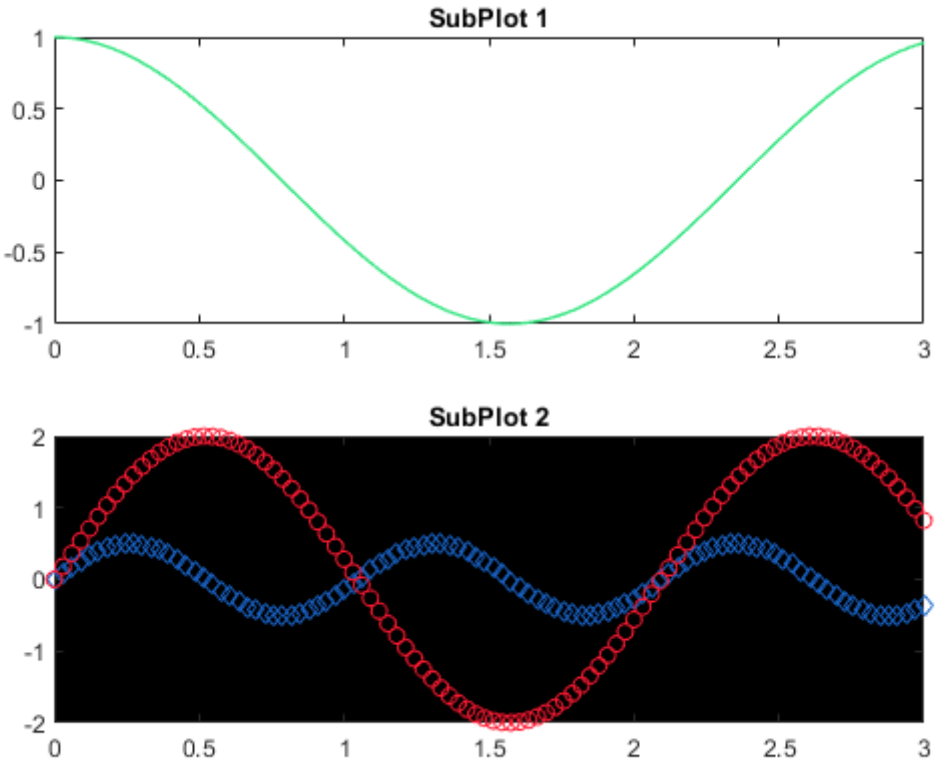
Comando: set(gca, 'Color', parâmetro)

Descrição: Para mudar a cor de fundo, essa linha de comando é a solução!

Exemplo

```
>> x=linspace(0,3,100)

>> y=sin(3*x)

>> z=cos(2*x)

>> w=1/2*sin(6*x)
%subplots ----------------------------------------------------------
>> sub1=subplot(2,1,1)
```

```
>> sub2=subplot(2,1,2)
%-----------------------------------------------------------------------
%plotando no subplot 1; posição 1ª linha, 1ª coluna
>> plot(sub1,x,z,'Linewidth',1,'Color',[.2 .9 .5])

>> title(sub1,'SubPlot 1')

%plotando no subplot 2; posição 2ª linha, 1ª coluna
>> plot(sub2,x,w,'d','Linewidth',0.5,'Color',[.1 .4 .8])

>> hold on;

>> plot(sub2,x,2*y,'o','Linewidth',0.5,'Color',[1 .1 .2])

>> set(gca,'Color','k') %mudando a cor de fundo no subplot 2

>> title(sub2,'SubPlot 2')
```



- **Curvas Polares**

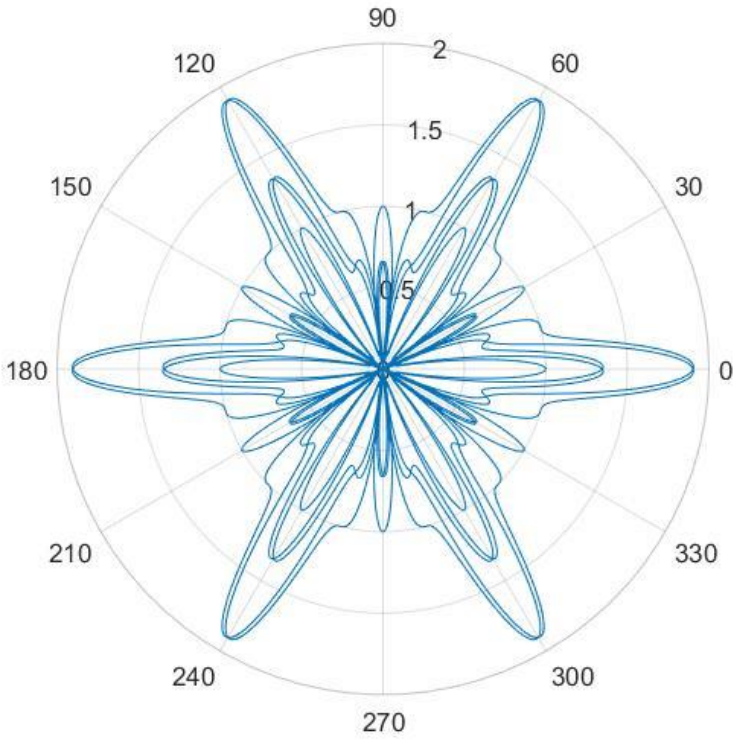
Curvas polares são traçadas em um plano em função de dois argumentos, ρ e θ e para plotar uma curva em coordenadas polares basta usar o seguinte comando:

Comando: polarplot(ρ,θ)

Exemplo:

Neste exemplo temos que $\boldsymbol{\rho} = sen^2(1{,}2\theta) + cos^3(6\theta)$ e $0 \leq \boldsymbol{\theta} \leq 16\pi$

```
>> theta = 0:0.01:16*pi;

>> rho = (sin(1.2*theta)).^2 + (cos(6*theta)).^3;

>> polarplot(theta,rho);
```



16.2. Gráficos Tridimensionais

- **Meshgrid**

Comando: [X,Y]= meshgrid(x,y)

Descrição: Uma vez definido os vetores x e y, deseja-se criar uma matriz de valores para gerar um conjunto de pontos em um domínio retangular para associar a uma imagem Z. Para isso é utilizado o comando **meshgrid** que vai gerar matrizes X e Y que correspondem ao conjunto de pontos neste domínio retangular.

Exemplo

```
>> x=linspace(-3,3,100);
>> y=linspace(-3,3,100);
>> [X,Y]=meshgrid(x,y) %domínio retangular X,Y em função dos veto-
res x e y

X =

   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.6667    2.3333    3.0000
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.6667    2.3333    3.0000
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.6667    2.3333    3.0000
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.6667    2.3333    3.0000
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.6667    2.3333    3.0000
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.6667    2.3333    3.0000
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.6667    2.3333    3.0000
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.6667    2.3333    3.0000
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.6667    2.3333    3.0000
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.6667    2.3333    3.0000


Y =

   -3.0000   -3.0000   -3.0000   -3.0000   -3.0000   -3.0000   -3.0000   -3.0000   -3.0000   -3.0000
   -2.3333   -2.3333   -2.3333   -2.3333   -2.3333   -2.3333   -2.3333   -2.3333   -2.3333   -2.3333
   -1.6667   -1.6667   -1.6667   -1.6667   -1.6667   -1.6667   -1.6667   -1.6667   -1.6667   -1.6667
   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000
   -0.3333   -0.3333   -0.3333   -0.3333   -0.3333   -0.3333   -0.3333   -0.3333   -0.3333   -0.3333
    0.3333    0.3333    0.3333    0.3333    0.3333    0.3333    0.3333    0.3333    0.3333    0.3333
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    1.6667    1.6667    1.6667    1.6667    1.6667    1.6667    1.6667    1.6667    1.6667    1.6667
    2.3333    2.3333    2.3333    2.3333    2.3333    2.3333    2.3333    2.3333    2.3333    2.3333
    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000
```

- **Plotar curva tridimensional**

Para plotar uma curva tridimensional utilizando o domínio retangular definido entre com os valores dos vetores discretizados x, y. deve-se associar uma função Z que seja imagem deste domínio retangular.

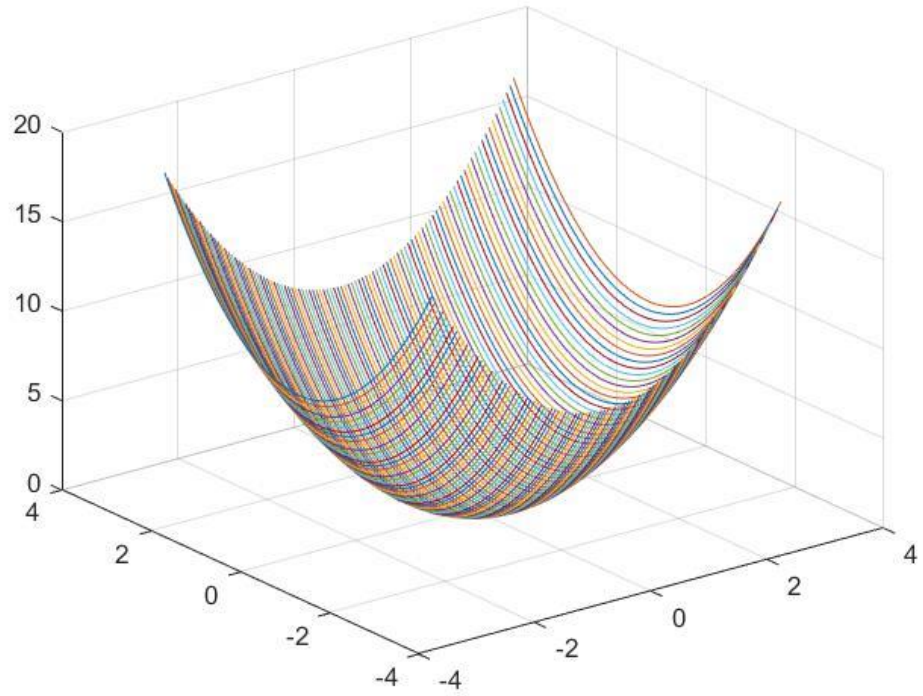
- **Plotar uma curva tridimensional através de um conjunto de curvas tridimensionais.**

  Comando: plot3(X, Y, Z)

No exemplo abaixo, há um conjunto de curvas tridimensionais sendo plotadas no mesmo espaço. Para usar esse comando, é exigido que você utilize matrizes X, Y e Z com o mesmo tamanho.

Exemplo

```
>> x=linspace(-3,3,100);

>> y=linspace(-3,3,100);

>> [X,Y]=meshgrid(x,y);

>> Z=X.^2+Y.^2

>> plot3(X,Y,Z)
```



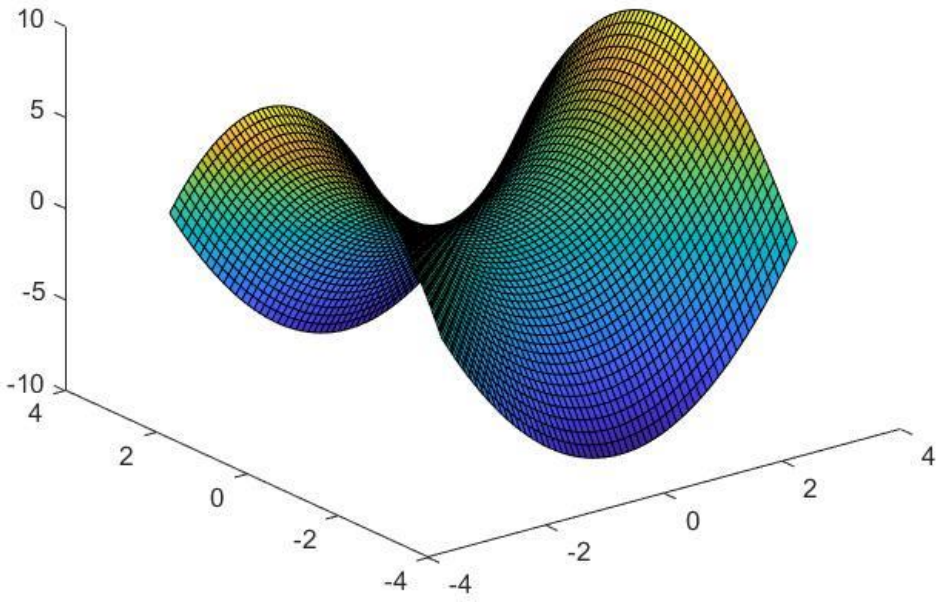
- **Plotar superfície**

Comando: surf(X,Y,Z)

Descrição: Para usar esse comando, é exigido que você utilize matrizes X, Y e Z com o mesmo tamanho.

Exemplo

```
>> x=linspace(-3,3,60);


>> y=linspace(-3,3,60);
>> [X,Y]=meshgrid(x,y);
>> Z=X.^2-Y.^2;
>> surf(X,Y,Z)
```



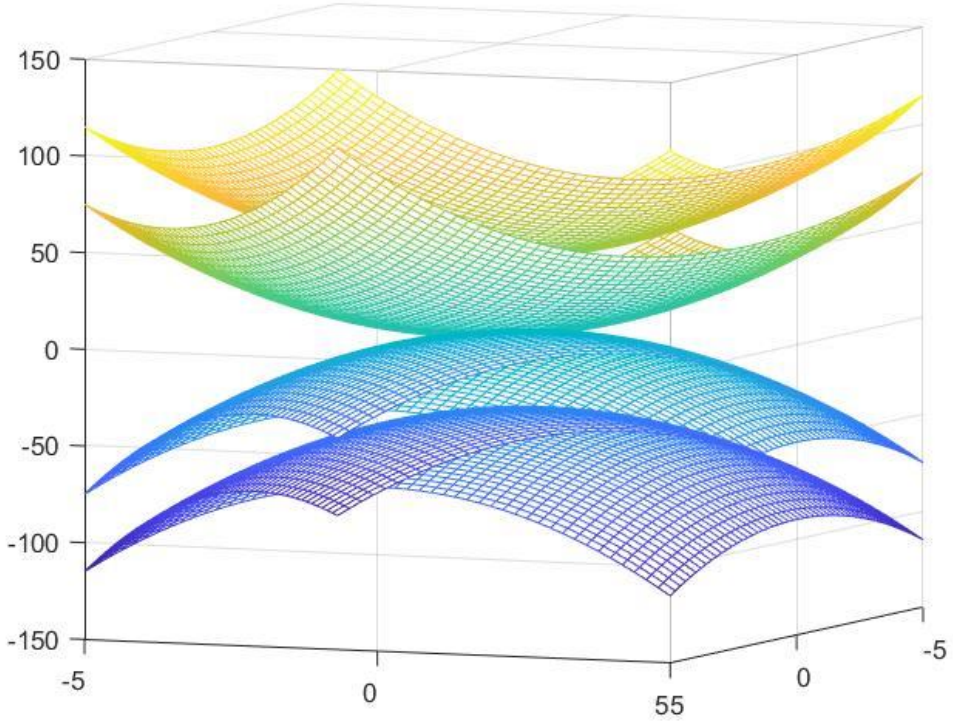
- **Plotar malha**

Comando: mesh(X,Y,Z)

Descrição: Enquanto o surf gera uma superfície entre os pontos, o comando mesh mostrará os pontos e elementos da função Z(X,Y).

Exemplo

```
>> x = linspace(-5,5,50);

>> y = linspace(-5,5,50);

>> [X,Y]=meshgrid(x,y);

>> Z=Y.^2 + 2*X.^2;

>> grid

>> hold on

>> mesh(X,Y,Z);

>> mesh(X,Y,-Z);

>> Z=40+Y.^2 + 2*X.^2;

>> mesh(X,Y,Z);

>> mesh(X,Y,-Z);
```



Estas funções podem ser complementadas pelos mesmos ajustes que foram realizados nas funções bi-dimensionais de gráfico, como a mudança de cor, aumento ou diminuição do peso da linha, mudança de padrão e estilo de linha, etc.

- **Curvas paramétricas**

Uma curva espacial pode ser dada através de equações paramétricas em função de um único parâmetro real.

Exemplo:

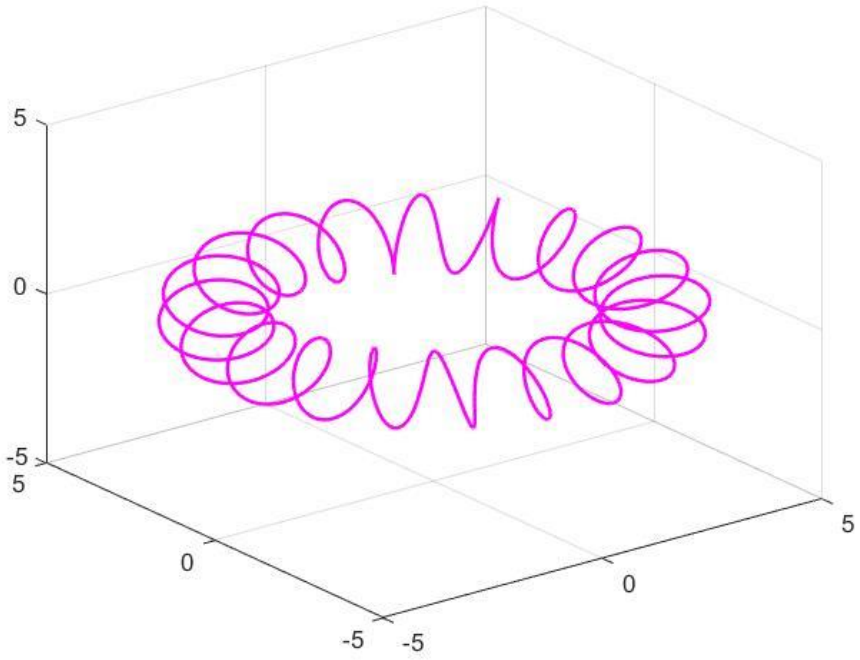
Seja a **espiral toroidal** dada pelas equações paramétricas:

$$x = (2 + \text{sen}\,1{,}5t)\cos t \qquad y = (2 + \text{sen}\,1{,}5t)\,\text{sen}\,t \qquad z = \cos 20t$$

Onde o parâmetro **t** varia entre $0 \leq t \leq 2\pi$

```
>> t = linspace(0,2*pi,10000);
```

```
>> x = (4 + sin(t.*20)).*cos(t);

>> y = (4 + sin(t.*20)).*sin(t);

>> z = cos(t.*20);

>> plot3(x,y,z,'linewidth',1.5,'color','m')

>> xlim([-5,5])

>> ylim([-5,5])

>> zlim([-5,5])

>> grid
```



- **Superfícies paramétricas**

Uma superfície espacial pode ser dada através de equações paramétricas em função de dois parâmetros reais.

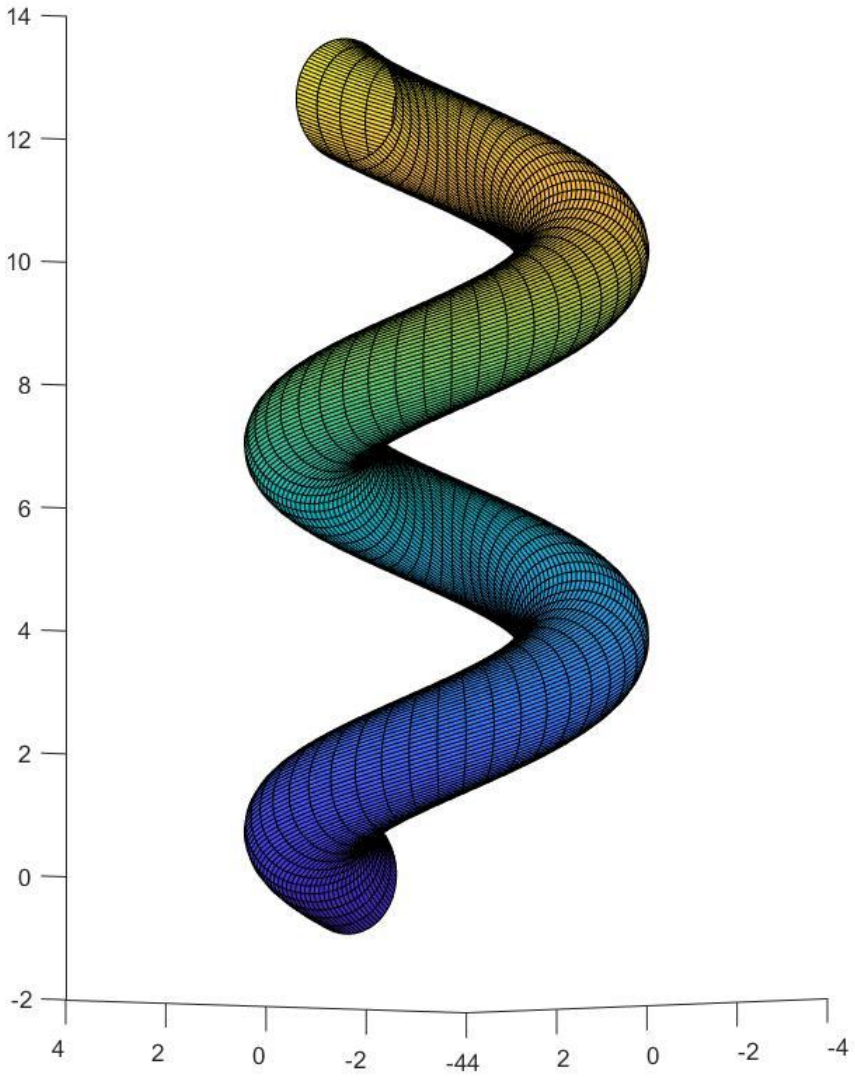
Exemplo:

Seja a superfície dada pelas equações paramétricas:

$$x = (2 + \operatorname{sen} v)\cos u \qquad y = (2 + \operatorname{sen} v)\operatorname{sen} u \qquad z = u + \cos v$$

Onde o parâmetro **u** varia entre $0 \leq u \leq 4\pi$ e **v** varia entre $0 \leq v \leq 4\pi$

```
>> u1 = linspace(0,4*pi,90);

>> v1 = linspace(0,4*pi,90);

>> [u,v]=meshgrid(u1,v1);

>> X = (2 + sin(v)).*cos(u);

>> Y = (2 + sin(v)).*sin(u);

>> Z = u + cos(v);

>> surf(X,Y,Z)
```



- **Ponto de vista**

Comando: view(az,el)

Descrição: Especifique os ângulos associados à vista do gráfico com precisão utilizando os parâmetros *az*(ângulo azimutal) e *el*(ângulo de elevação). Dados em grau.

Exemplo

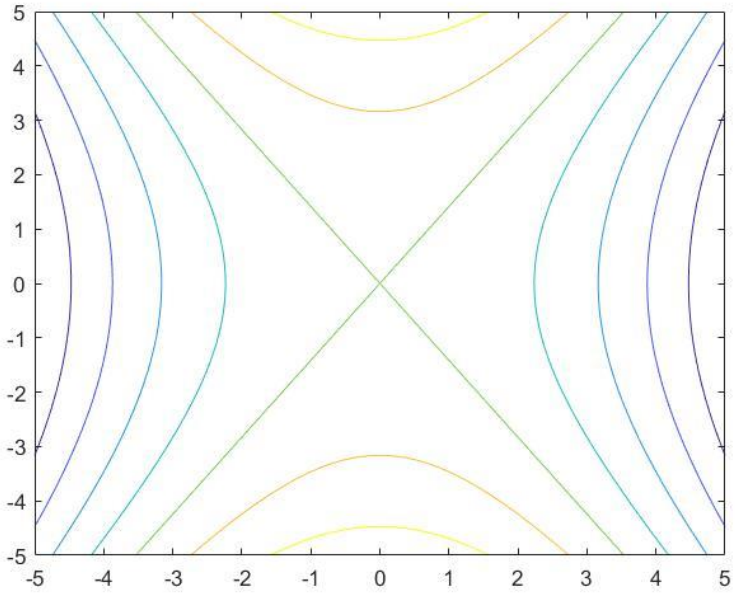
```
>> surf(X,Y,Z)

>> view(60,30)
```

- **Curvas de nível**

  Comando: contour(X,Y,Z)
  Descrição: Crie um mapa de curvas de níveis bidimensional utilizando.

Exemplo

```
>> x = linspace(-5,5,1000);

>> y = linspace(-5,5,1000);

>> [X,Y]=meshgrid(x,y);

>> Z=Y.^2 - 2*X.^2;

>> grid

>>contour(X,Y,Z)
```



- **Plotando Superfícies dadas implicitamente por equações.**

Algumas superfícies; a exemplo de algumas quadricas não podem ser explicitadas como uma função assim como ocorre com o hiperboloide de uma folha $x^2 + y^2 - z^2 = 1$

Comando: variável = @(x, y, z) função 115mplícita
fimplict3(*variável*)

Devemos utilizar estes comandos da seguinte forma, primeiro tomar a função implícita e iguala-la a zero passando todos os demais termos para apenas um membro da equação, por exemplo:

$$x^2 + y^2 - z^2 = 1 \rightarrow x^2 + y^2 - z^2 - 1 = 0$$

Em seguida tome o membro não nulo da equação e insira-o no comando @(x,y,z), podemos chamar de **f** a variável onde atribuiremos a nossa função, desta forma teríamos algo similar a:

$$\mathbf{f} = @(x,y,z)x^2 + y^2 - z^2 - 1$$

Então use *fimplict3*(***f***) para printar a superfície.

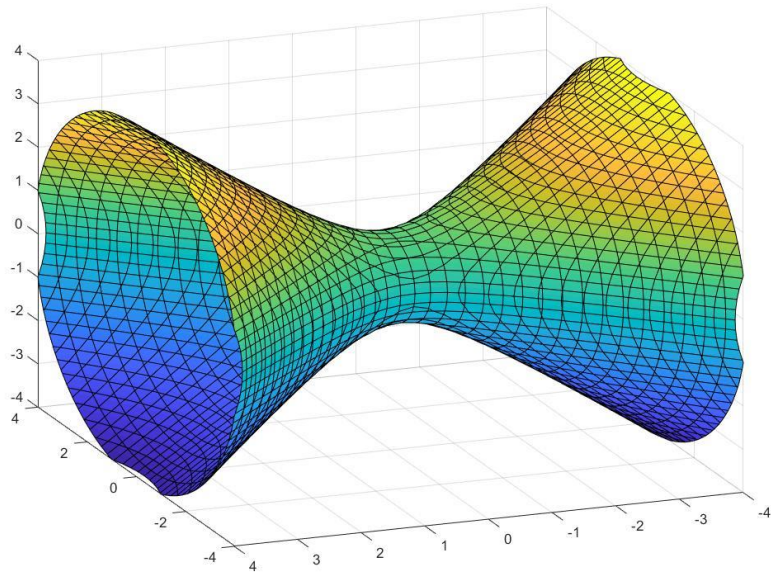
Para determinar o intervalo de exibição use:

*Comando: inteval = [**X**min , **X**max , **Y**min, **Y**max, **Z**min, **Z**max]*

O comando interval após declarado deve ser passado como parâmetro ao comando fimplict3() como visto no exemplo abaixo.

Exemplo

```
>> f = @(x,y,z) +x.^2 - y.^2 + z.^2 -1; %declarando a função
115mplícita

>> interval = [-4 4 -4 4 -4 4]; %intervalo de exibição

>> fimplicit3(f,interval) %print
```



Exercícios de fixação:

1 - Descubra quais são as raízes da função $\boldsymbol{f(x) = 3 \cdot x^3 - 26 \cdot x + 10}$ utilizando gráficos.

2 - Trace os gráficos da função seno e cosseno de um número no interval de 0 a 5π. Coloque a curva do seno pontilhada e azul. Coloque a curva do cosseno tracejada e vermelha. Nomeie também os eixos x e y.

3 - A expressão que modela o crescimento da população brasileira pode ser escrita como $P(t) = \frac{157273000}{(1+e^{-0.0313\cdot(t-1913.25)})}$, onde t é dado em anos. Mostre num gráfico como varia o crescimento populacional entre os anos de 1900 e 2100.

4 - Sendo as equações parametrizadas:

$$x = sen(t)$$
$$y = \cos(t)$$
$$z = t$$

Obtenha o gráfico tridimensional para t variando de 0 a 6π.

17. Interpolação e ajuste de curvas

Quando se trabalha com um conjunto de dados coletados de maneira discreta, por vezes, é vantajoso pensar em uma função analítica contínua que representa aquele conjunto de dados da melhor maneira possível. Esses dados podem ser ou não pertencentes a essa curva. E a diferença entre fazer com que os dados pertençam à função e fazer com que a função se adeque da melhor maneira aos dados é a exata diferença entre interpolar uma curva e ajustar uma curva.

17.1. Interpolação Polinomial

Na interpolação, normalmente fazemos uma conexão entre alguns pontos, por um tipo específico de função. Para interpolar uma reta a um conjunto de dados, precisamos que o conjunto contenha apenas dois pontos. Com isso, garante-se que uma reta seja gerada pelo conjunto de dados e que ela passe por todos os pontos.

Para um polinômio de segundo grau, o número de pontos exigidos para gerar uma parábola é de três pontos. E para um polinômio de grau N é de N+1 pontos.

Para interpolar um polinômio de grau N a um conjunto de N+1 pontos, resolve-se a equação linear:

$$a_0 + a_1 x_i + a_2 {x_i}^2 + \cdots + a_{n+1} {x_i}^n = y_i$$

Que é resolvida em um sistema de N+1 equações, uma equação para cada ponto i={1,...N+1}.

17.2. Interpolação Polinomial Linear

Suponha que tenha sido importada para o MATLAB uma tabela contendo os seguintes valores para um dado experimento que relaciona a profundidade de um túnel (P), em metro, com a temperatura(T), em Kelvin. Para realizar uma interpolação de primeiro grau no nosso conjunto de dados basta utilizar interp1(P,T,p) em que p é um valor de P não contido no conjunto fornecido mas que esteja entre dois valores consecutivos conhecidos.

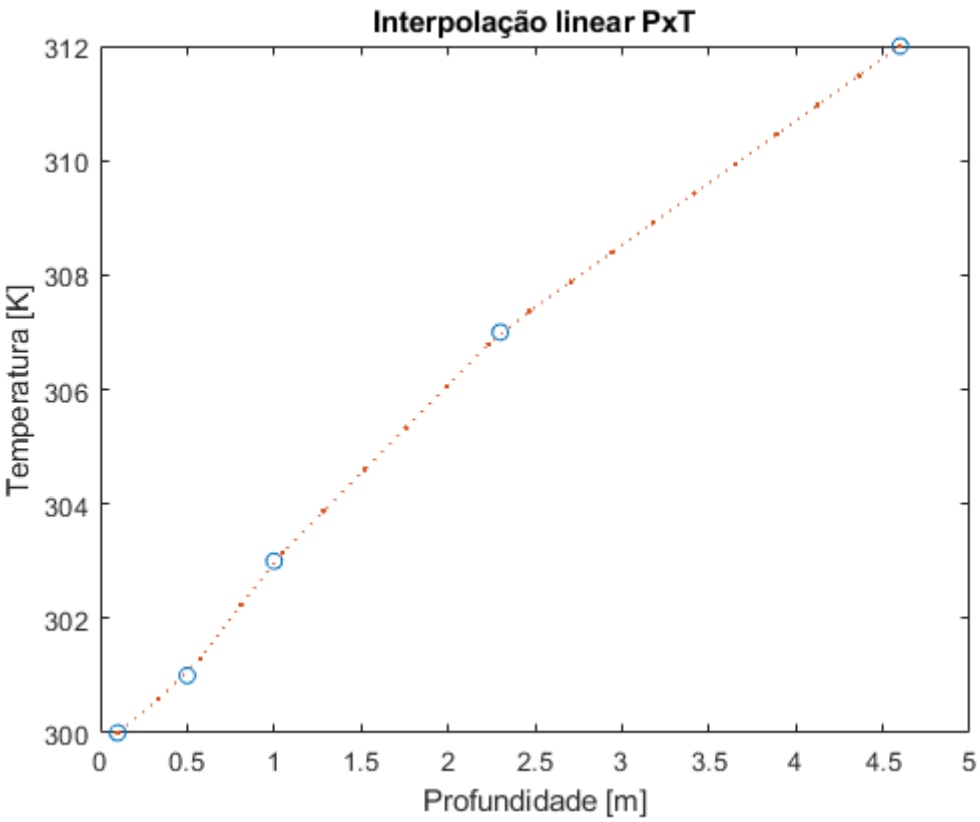
Exemplo

```
>> P=[0.1;0.5;1;2.3;4.6]

P =

    0.1000
    0.5000
    1.0000
    2.3000
    4.6000
```

```
>> T=[300;301;303;307;312]

T =

   300
   301
   303
   307
   312

>> interp1(P,T,3.5)

ans =

  309.6087
```

Por esse método, é impossível predizer um valor antes de 0.1000 ou depois de 4.6000 para p ∈ P. Visto que não se sabe como estimar uma inclinação razoável para o conjunto de dados antes e depois disso.

Abaixo um gráfico da linha poligonal que contém os valores que podem ser interpolados no nosso conjunto de dados (circulados em azul).



Para interpolar um segmento de reta, são necessários dois pontos. Para interpolar uma parábola são necessários 3 pontos.

$$y_1 = ax_1^2 + bx_1 + c$$
$$y_2 = ax_2^2 + bx_2 + c$$
$$y_3 = ax_3^2 + bx_3 + c$$

E para polinômios de grau n, são necessários n+1 pontos.

17.3. Ajuste de curvas pelo comando *polyfit*

Comando: p=polyfit(x,y,n)

Descrição: Gera um vetor que contém os coeficientes de ajuste a partir de pontos conhecidos

```
>> p=polyfit(x,y,n)
```

Sendo:

p = o vetor que receberá os **coeficientes** do polinômio de ajuste
x = o vetor que contém os **valores das abcissas** dos pontos que serão ajustados
y = o vetor que contém os **valores das ordenadas** dos pontos que serão ajustados
n = indica o **grau** do polinômio que se deseja o ajuste (n=1, 2, 3...)

17.4. Ajuste de curvas não polinomiais pelo comando *polyfit*

Quando temos funções que não são polinomiais mas precisam ser ajustadas, também podemos usar o comando *polyfit* com certas adaptações aos vetores que contenham suas abcissas, ordenadas como mostrado na tabela modelo a seguir:

- **Potência:** $y = bx^m$

```
>> p=polyfit(log(x),log(y),1)
```

- **Exponencial:** $y = be^{mx}$

```
>> p=polyfit(x,log(y),1)
```

$y = b10^{mx}$

```
>> p=polyfit(x,log10(y),1)
```

- **Logarítimica:** $y = m \ln x + b$

```
>> p=polyfit(log(x),y,1)
```

$y = m \log x + b$

```
>> p=polyfit(log10(x),y,1)
```

- **Hiperbólica:** $y = \frac{1}{mx+b}$

```
>> p=polyfit(x,1./y,1)
```

Exercícios de fixação

1 - A temperatura em determinado ponto do Oceano Atlântico está sendo avaliada em função da profundidade de maneira experimental. A cada 50 centímetros de profundidade, a partir da superfície, um novo registro é realizado. Os valores obtidos foram:

T = [27.0228 25.4152 23.7288 22.3201 21.4839 21.3796]

(a) Faça uma interpolação linear e estime a temperatura a 1,75m.
(b) Faça uma interpolação polinomial de 5o grau e estime a temperatura a 1,75m.
(c) Compare os dois valores encontrados para a temperatura a 1,75m.
(c) Utilize o mesmo polinômio interpolador para predizer a próxima temperatura a ser registrada.

2 - Crie um script para ajustar curvas polinomiais ao conjunto de pontos da workspace no arquivo ajuste.mat de diferentes graus, plotando gráficos das curvas para verificar a qualidade da interpolação.

3 - A equação de estado do gás ideal relaciona volume, pressão, temperatura e a quantidade de um gás. $V = \frac{nRT}{P}$. Um experimento foi realizado para determinar a constante universal dos gases. No experimento, 0.05 mol de gás é submetido a várias pressões, enquanto o gás vai sendo comprimido em um recipiente apropriado. A cada novo volume ocupado pelo gás são medidas sua pressão e temperatura. De acordo com os dados fornecidos, construa um gráfico $V\ x\ T/P$ e realize um ajuste linear no conjunto de pontos para se determinar R.

V(L)	0,75	0,65	0,55	0,45	0,35
T(°C)	25	37	45	56	65
P(atm)	1,63	1,93	2,37	3,00	3,96

4 - A população da China no período de 1940 a 2000 é mostrada na tabela abaixo:

Ano	1940	1950	1960	1970	1980	1990	2000
População (milhões)	537	557	682	826	981	1.135	1.262

a) Determine a função exponencial que melhor se ajusta ao conjunto de pontos. Use essa mesma função para estimar a população no ano de 1955.

b) Ajuste o conjunto de pontos utilizando uma equação quadrática e utilize essa função para estimar a população no ano de 1955.

c) Interpole *spline* e linearmente esse conjunto de pontos. Utilize essas funções para estimar a população no ano de 1955.

5 - Escreva uma função que ajuste certo conjunto de pontos a partir de uma potência da forma $y = bx^m$. Salve a função como `[b,m]=polyfit(x,y,n)`, onde os argumentos x e y são vetores contendo as coordenadas dos pontos e os argumentos b e m são constantes da equação de ajuste. Teste a função no seguinte conjunto de pontos e construa um gráfico que mostre simultaneamente os dados e a função. (Não se esqueça de adaptar as abcissas e ordenadas do comando *polyfit* como visto acima)

x	0,5	1,9	3,3	4,7	6,1	7,5
y	0,8	10,1	25,7	59,2	105	122

18. Mínimo e Máximo de Uma Função

Comando: x = fminbnd('função', x1,x2)
Descrição: Fornece o ponto onde a função desejada tem seu mínimo valor para o intervalo *x1* até *x2*

Comando 2: [x valor] = fminbnd('função',x1,x2)
Descrição 2: No comando acima, *x* será o ponto onde a função apresenta seu menor valor, e *valor* será a imagem dessa função nesse ponto de mínimo.

Exemplo

```
>> [x valor]=fminbnd('x^3 -12*x^2 +40.25*x -36.5',0,8)
%valor onde a função apresenta o mínimo valor%
x =
    5.6073

%o mínimo valor da função no intervalo especificado%
valor =

  -11.8043
```

Quando se deseja encontrar o **máximo** valor da função em um dado intervalo, basta multiplicar toda a função desejada por **"-1**" e manter a mesma estrutura de comando, e se obterá o valor mínimo, que se multiplicado por **"-1"** é o **valor máximo da função original**.

Exemplo

```
>> [x valor]=fminbnd('-x^3 +12*x^2 -40.25*x +36.5',0,8)
%valor onde a função apresenta o mínimo valor%
x =

    2.3927

%o mínimo valor da função no intervalo especificado%
valor =

   -4.8043

>> max = valor*(-1)
max =

   4.8043
```

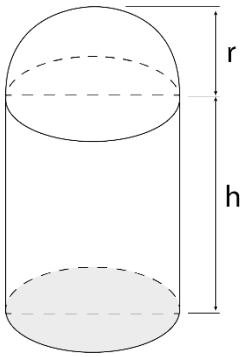
Exercícios

1 - Seja $f(x) = x^3 - 3x^2 + 3$, determine os valores máximo e mínimo de f em $[-2, 3]$. Em que pontos estes valores são atingidos?

2 - Determine as dimensões do retângulo de área máxima e cujo perímetro é 20 cm.

3 - Duas partículas P e Q movem-se, respectivamente, sobre os eixos $0x$ e $0y$. A função de posição P é $x = \sqrt{t}$ e a de Q, $y = t^2 - \frac{3}{4}$, $t \geq 0$. Determine o instante em que a distância entre P e Q seja a menor possível.

4 - Um sólido será construído acoplando-se a um cilindro circular reto, de altura h e raio r, uma semi-esfera de raio r. Deseja-se que a área da superfície do sólido seja 5π. Determine r e h para que o volume seja máximo.



19. Diferenciação

A derivada de uma função em um dado ponto é definida como a inclinação da reta tangente a esse ponto. De forma aproximada, se pegarmos dois pontos (x_1, y_1) e (x_2, y_2) muito próximos, podemos aproximar a derivada a partir da inclinação da reta que passa por esses dois pontos: $\frac{dy}{dx} \approx \frac{y_2 - y_1}{x_2 - x_1}$

No MATLAB é possível fazer esse cálculo das derivadas de uma função de forma aproximada utilizando dois pontos suficientemente próximos e utilizando o comando diff.

A função diff calcula a diferença entre dois pontos adjacentes em um vetor, gerando um novo vetor com a diferença. Dessa forma, seja x = [0, 1, 2, 3, 4, 5] e y = [2 3 5 1 9 0], o vetor gerado por diff(x) será [1, 1, 1, 1, 1] enquanto que diff(y) resultará [1, 2, -4, 8, -9]. Sabendo essas diferenças é possível avaliar, de forma aproximada, a derivada de uma função em um intervalo.

Se diff for utilizado com uma matriz, a função tratará cada coluna da matriz como um vetor e calculará as diferenças para as colunas.

No MATLAB, a derivada em um intervalo da função, será utilizado diff(y) ./ diff(x).

Por exemplo, se for desejado avaliar a derivada da função $f(x) = x^5-4x^4+2x^2-8x+5$ no intervalo [-1,5], basta:

- criar um vetor x com intervalos pequenos de -1 a 5;
- avaliar a função nesses intervalos e adicionar estes valores em um vetor y;
- calcular dy = diff(y);
- calcular dx = diff(x);
- avaliar a derivada df como sendo dy/dx;

Exercícios

1 - Derive as seguintes equações utilizando variáveis simbólicas:

a) $y = sen\, x \cdot (2x - 1) + \cos x \cdot (x^2 + 1)$

b) $y = 8t \cdot (t^2 + 3)^3$

c) $y = \frac{2t+3}{t^2+3t+9}$

2 - Encontre a inclinação da reta tangente a curva $y = \ln x + 1 - x^2$ no ponto $x = 5$.

3 - Calcule $\lim_{x \to 0} \frac{sen(a+2x) - 2 \cdot sen(a+x) + sen\, a}{x^2}$

4 - Se uma bola for atirada verticalmente para cima com uma velocidade de 24,5 m/s, então sua altura depois de t segundos é $s = 24{,}5t - 4{,}9t^2$.

a) Qual a altura máxima atingida pela bola?
b) Qual a velocidade da bola quando estiver 29,4m acima do solo da subida? E na descida?

5 - A quantidade de carga Q, em coloumbs (C), que passa através de um ponto em um fio até o instante t em segundos é dada por $Q(t) = t^3 - 2t^2 + 6t + 2$. Encontre a corrente em:

a) t = 0,5 s
b) t = 1s.

20. Integração Numérica

O MATLAB nos fornece alguns métodos de integração por métodos iterativos (numéricos), neste capítulo serão abordados dois comandos.

20.1. Quadratura de Simpson

Comando: quad(@(x)(fun),a,b) ***ou*** *('fun',a,b)*

Descsrição: A primeira sintaxe do comando parece mais trabalhosa a primeira vista, e também para integrações em apenas uma variável, mas é um excelente recurso quando se necessita a resolução de integrais múltiplas. *Fun* se trata da função a ser integrada e deve ser inserida como uma *string* quando não se deseja declarar as variáveis. Os valores *a* e *b* são respectivamente o limite inferior e superior da integração desejada. Perceba que integrações numéricas apenas são úteis quando se deseja uma resposta expressa em números, quando se deseja realizar uma integração a fim de se obter a anti-derivada de uma função deve-se recorrer ao Toolbox de Matemática Simbólica. **Não se esqueça** que por se tratar de um método iterativo, as operações matemáticas devem ser expressas **elemento a elemento.**

Exemplo

```
>> i=quad('x.^2',0,2) %integral de x^2 de 0 a 2
i =
    2.6667

>> i=quad(@(x)(x.^2),0,2) %integral de x^2 de 0 a 2
i =
    2.6667
x
```

20.2. Quadratura de Simpson para Integrais Duplas

Comando: dblquad(@(x,y)(fun),a,b,ay,by) ***ou*** *('fun',a,b,ay,by)*

Descrição: Como dito acima, a sintaxe usando o arroba (@) é mais apropriada para integrais múltiplas, pois é através dela que determinamos a ordem de integração da função desejada (a ordem de declaração das variáveis determina a ordem de integração), caso deseja-se escrever a função a ser integrada como uma *string*, ou seja, dentro das aspas (") a ordem de integração padrão do MATLAB é a ordem alfabética.

Exemplo

```
>> i=dblquad(@(x,y)(x.^2+y),0,2,0,3)
i =
    17
>> i=dblquad('x.^2+y',0,2,0,3)
i =
    17
```

20.3. Quadratura de Simpson para Integrais Triplas

Comando: triplequad(@(x,y,z)(fun),a,b,ay,by,az,bz) ***ou*** *('fun',a,b,ay,by,az,bz)*

Descrição: O raciocínio e sintaxe de comandos seguem as integrais duplas, a única diferença é o início do comando.

Exemplo

```
>> i=triplequad(@(x,y,z)(x.^2+y+z),0,2,0,3,0,1)
i =
    20
>> i=triplequad('x.^2+y +z',0,2,0,3,0,1)
i =
    20
```

20.4. Comando Integral

Comando: fun=@(x) x.^2 | integral(fun,a,b)

Descrição: O comando integral não difere tanto da Quadratura de Simpson, pois também é um método numérico para a aproximação de integrais definidas. A grande diferença é que a função será integrada pelo método numérico mais adequado, definido pelo MATLAB. A função a ser integrada é determinada antes de se executar o comando, o que pode resultar em uma menor poluição visual, porém mais linhas de código. Por se tratar de um método iterativo, também se faz necessário que a função seja escrita considerando as **operações elemento a elemento**.

Exemplo

```
>> fun = @(x) x.^2; %função definida por x^2
>> i=integral(fun,0,2) %integral de x^2 de 0 a 2
i =
    2.6667
```

20.5. Comando Integral

Comando: fun=@(x,y) x.^2+y | integral2(fun,a,b,ay,by)

Descrição: Segue a mesma lógica de integração para uma variável, mas atente que o comando que era *integral* passa a ser *integral2*. Não se esqueça que a ordem de declaração das variáveis determina a ordem de integração da sua função.

Exemplo

```
>> fun = @(x,y) x.^2+y;
>> i=integral2(fun,0,2,0,3)
i =
 17.0000
```

20.6. Integral Numérica Tripla

Comando: fun = @(x, y, z) x.^2+y+z | integral3(fun, a, b, ay, by, az, bz)

Descrição: Sua única diferença para integrais duplas e singulares é que o comando passa a ser *integral3*. Não há diferenças de sintaxe.

Exemplo

```
>> fun = @(x,y,z) x.^2+y+z;
>> i=integral3(fun,0,2,0,3,0,1)
i =
   20.0000
```

21. Matemática Simbólica

Muitos usuários tem uma grande dificuldade a se acostumar com a lógica e funcionamento de softwares de programação, uma excelente alternativa para isso pode ser a *Toolbox* de Matemática Simbólica disponibilizada pelo MATLAB por possuir sintaxes muito simples e intuitivas.

21.1. Declarando Variáveis Simbólicas

Comando: syms var1 var2 varn

Descrição: Para se declarar variáveis simbólicas usamos o comando *syms* e então declaramos as variáveis desejadas; variáveis simbólicas são interessantes por nós permitirem realizar as operações matemáticas sem atribuir valores a estas variáveis, o que se assemelha muito a forma como fazemos os cálculos utilizando papel e caneta, isto se tornará mais claro durante os exemplos do capitulo.

Exemplo

```
>> syms x y z;
>> x

x =

x %observe que o valor da variável x é o próprio x; pois x foi declarado como variável simbólica; o mesmo fato ocorre para as variáveis y e z declaradas simbolicamente%
```

21.2. Convertendo resultados simbólicos em numéricos

Comando: double(x)

Descrição: Sendo *x* uma variável que tenha recebido um valor de configuração simbólica, podemos expressá-lo numericamente convertendo-a em uma nova variável do tipo *double*

Exemplo

```
>> syms x
>> s=int(x,0,pi) %integral de x de 0 a pi; a integração simbólica será discutida neste capítulo mais adiante%
```

```
s =

pi^2/2

>> k=double(s)

k =

    4.9348
```

21.3. Formatação de funções

Comando: pretty(função)

Descrição: O comando *pretty* nos retorna a expressão matemática da maneira que estamos acostumados a escrever manualmente, pode ser útil para expressões grandes e com muitas frações.

Exemplo

```
>> syms x y z
>> S=x^2 +2*y +1/z

S =

2*y + x^2 + 1/z

>> pretty(S)
       2   1
2 y + x  + -
           z
```

21.4. Simplificação de funções

Comando: simplify(S)

Descrição: Comando usado para simplificar funções simbólicas

Exemplo

```
>> syms x

>> simplify(log(x)+cos(x)^2+sin(x)^2)

ans =

log(x) + 1
```

21.5. Expandindo funções

Comando: expand(S)

Descrição: Realiza a expansão de uma expressão matemática determinada simbolicamente

Exemplo

```
>> syms x

>> S =(x+1)^3/(x-1)^2;
>> expand(S)

ans =

(3*x)/(x^2 - 2*x + 1) + 1/(x^2 - 2*x + 1) + (3*x^2)/(x^2 - 2*x + 1)
+ x^3/(x^2 - 2*x + 1)
```

21.6. Avaliação de funções simbólicas com valores numéricos

Comando: subs(S,var,número)

*Descrição: Sendo **S** é a expressão simbólica*, *var* é a variável a receber o valor numérico, e *número* é o valor numérico que se deseja avaliar na expressão.

Exemplo

```
>> syms x
```

```
>> S = x^2 +2*x +1;

>> T=subs(S,x,1)

 T =

 4
```

Caso desejarmos substituir para vários valores podemos elaborar um vetor e então usá-lo para substituir os valores

Exemplo

```
>> syms x

>> S = x^2 +2*x +1;

>> T=subs(S,x,[1:4])

T =

[4, 9, 16, 25]
```

Em casos que a expressão desejada possua mais de uma variável a estrutura do comando fica como

Comando: R = subs(S,{var1,var2,...,varn},{num1,num2,...,numn})

Exemplo

```
>> syms x t

>> S = x^2 + x +2*t;

>> R=subs(S,{x,t},{1,2})

 R =

 6
```

21.7. Resolvendo Equações Algébricas

Comando: solve(S) ou solve(S,var)

Descrição: Sendo S é a equação a ser determinada as raízes, e *var* é a variável para que se deseja resolver a equação. Uma observação importante é que quando não se determina o lado direito da igualdade, o MATLAB reconhece automaticamente como 0. Para se adicionar a igualdade de uma equação devemos usar o símbolo de igualdade duas vezes ==

Exemplo

```
>> syms x

>> S=x^2 -x -6;

>> k=solve(S)

 k =

  -2

  3

%as raízes da expressão S são atribuídas ao vetor k%

%caso S fosse determinado como (x^2-x-6==0) o resultado seria o
mesmo%
```

21.8. Resolvendo um Sistema de Equações

Comando: solve(eq1,eq2,...,eqn,var1,var2,...,varn)

Descrição: Para se obter a solução de um sistema de equações, devemos adicionar as equações , que devem estas serem adicionadas com o uso da vírgula.

1 - Se o número **n** de equações é igual ao número de variáveis nas equações, o MATLAB apresenta uma solução numérica para todas as variáveis.

2 - Se o número de variáveis for **maior** que a de equações, o MATLAB apresenta uma solução para **n** variáveis em termos do restante das variáveis

3 - Quando o número de variáveis supera o número de equações você pode escolher para que variáveis o sistema será resolvido

Exemplo

```
>> syms x y t

>> S

 S =

 16*t + 10*x + 12*y

 >> [xt yt]=solve(S,'5*x -y=13*t')

 xt =

 2*t

 yt =

 -3*t

%nesse exemplo criamos um vetor e atribuímos seus valores as solu-
ções do sistema linear%

%a solução foi determinada em função de t pois como não foi defi-
nida manualmente, o MATLAB adota a ordem de aparição das variá-
veis%
```

21.9. Plotando funções simbólicas

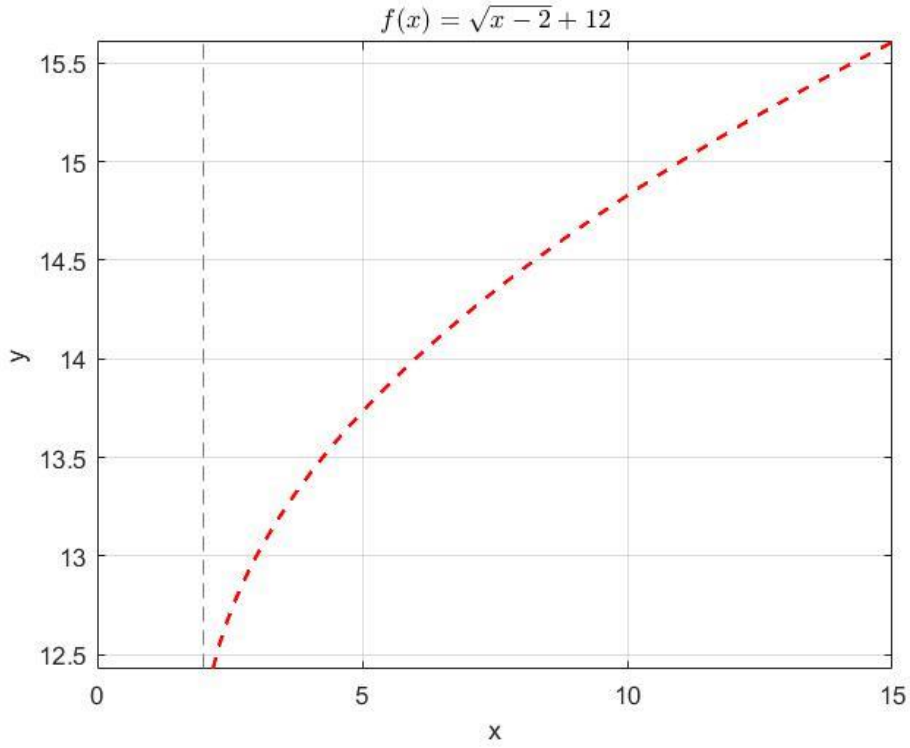
Comando: fplot(S) ***ou*** *fplot(S,[xmin,xmáx])* ***ou*** *ezplot(S,[xmin,xmáx], n)*

Desrição: O comando de representação gráfica de funções simbólicas é bem simples, e podemos delimitar seus limites em *x,* n representa todos os parâmetros aprendidos para gráficos, como cor da linha, tipo da linha, espessura da linha e etc.

Exemplo

```
>> syms x ;

>> S= sqrt(x-2)+12;

>> fplot(S,[0 15],'--','linewidth',1.5,'color','r')

>> grid

>> xlabel('x')

>> ylabel('y')

>> title('$f(x)=\sqrt{x-2}+12$','interpreter','latex')
```

A parte marcada é o comando para adicionar um título ao gráfico, neste título adicionamos a lei de formação que define esta função; sua exibição só acontece pois utilizamos um interpretador LaTeX nativo do matlab, caso não fizéssemos isto seria printado no titulo sqrt ao invés do símbolo de raiz, caso você saiba utilizar comandos em LaTeX você pode usar seus comandos junto dos parâmetros **'interpreter'** e **'latex',** observe este fato no gráfico a seguir.



21.10. Limites em funções simbólicas

Comando: limit(S,var,a,'sentido')

Descrição: Sendo *S* uma variável que contenha a expressão simbólica a ser avaliada no limite, *var* será a variável que desejamos investigar no limite, por exemplo, x, *a* será o ponto que se deseja investigar o limite, por exemplo, Inf (infinito) e o *'sentido'* indica o sentido que o limite deve ser avaliado, as opções possíveis são: esquerda ('left') e direita ('right'), lembre que é necessário o uso de apóstrofe para o sentido.

Exemplo

```
>> syms x
>> S=1/x;
>> limit(S,x,Inf) %quando x tende ao infinito a função tende a
zero%
 ans =
 0

>> limit(S,x,0,'right') %quando x tende a 0 pela direita a função
tende ao infinito%
 ans =
Inf

>> limit(S,x,0,'left') %quando x tende a 0 pela esquerda a função
tende ao infinito mas na parte negativa do gráfico%
 ans =
 -Inf
```

Limites de funções com **mais de uma variável** também podem ser calculados, entretanto não há um comando específico para esse fim, entretanto isso não é um problema, veja no exemplo a seguir como proceder.

Exemplo

```
>> syms x t
>> S=1/(x*cos(2*t));
>> limit(limit(S,x,1),t,0) %nessa linha de comando fizemos nada
```

```
mais nada menos que um limite de um limite, primeiramente realiza-
mos o limite da função S quando x tende a 1, em seguida, o resul-
tado desse limite será usado para calcular o limite quando t tende
a 0%

ans =

1
```

Caso seja difícil visualizar o que está acontecendo, também podemos realizar etapa por etapa, salvando seus respectivos resultados em outras variáveis auxiliares, veja no exemplo a seguir:

Exemplo

```
>> syms x t
>> S=1/(x*cos(2*t));
>> S1=limit(S,x,1)
S1 =
1/cos(2*t)
>> S2=limit(S1,t,0)
S2 =
1
```

22. Derivação Simbólica

Comando: diff(S)

Descrição: Quando desejarmos diferenciar de maneira simbólica, devemos usar o comando *diff(S)* sendo *S* uma variável simbólica que contenha a função a ser derivada. Expressões também podem ser derivadas diretamente sem a necessidade de uma variável para se armazenar a função

Exemplo

```
>> syms x

>> S = 3*x + 4*x^3;

>> diff(S)

ans =

12*x^2 + 3
```

Também é possível fazer derivação sem declarar vetores, inserindo diretamente a expressão desejada.

Exemplo

```
>> diff(2*x +x^2)

ans =

2*x + 2
```

22.1. Derivação Parcial Simbólica

Comando: diff(S,var)

Descrição: A adição de uma vírgula nos oferece a opção de uma **derivação parcial**. Após a vírgula devemos adicionar a variável que desejamos derivar a função. Quando se deseja derivar uma função

várias vezes em relação a diferentes variáveis a **sugestão** é que atribua o resultado da derivação a uma célula (variável) e então se execute as operações quantas vezes forem necessárias.

Exemplo

```
>> syms x y
>> diff(x^2 +2*x +y,y) %derivando a função em relação a y%
ans =
1

>> diff(x^2 +2*x +y,x) %derivando a mesma função em relação a x%
ans =
2*x + 2

>> syms x y
>> S=x^2+2*x+2*y;
>> Sx=diff(S,x)        %a primeira derivada parcial em relação a x%
Sx =
2*x + 2

>> Sy=diff(Sx,y)      %a derivada segunda, mas em relação a y%
Sy =
0
```

23. Integração Simbólica

Comando: int(S,var)

Descrição: A integração executada de maneira simbólica será feita de maneira semelhante à derivação. Quando não se escolhe um intervalo de integração, o resultado obtido será sua anti-derivada. Note que a constante de integração é automaticamente adotada como 0.

Exemplo

```
>> syms x

>> S=x^2 +1;

>> int(S)

 ans =

(x*(x^2 + 3))/3
```

Como na derivação, a **adição da vírgula** após o vetor *S* ou a expressão a ser integrada, nos dará a variável que deve ser respeitada na integração.

Exemplo

```
>> int(S,z)

 ans =

 z*(x^2 + 1)
```

23.1. Integração Definida Simbólica

Comando: int(S,var,a,b)

Descrição: Não há grandes novidades aqui. Como visto na integração numérica, *a* e *b* serão os limites de integração.

Exemplo

```
>> B=x^2 + 2 + z;

>> int(B,z,0,1)

 ans =

 x^2 + 5/2  %a integral foi calculada para z em uma função de duas
variáveis, onde os limites de integração para z foram atribuidos%
```

23.2. Integração Múltipla Definida Simbólica

Comando: int(int(S,var1,a,b),var2,a2b2) ***ou*** *int(int(int(S,var1,a,b),var2,a2b2),var3,a3,b3)*

Descrição: A grande diferença é que para integrais múltiplas não há um comando específico no universo dos simbólicos. Uma solução alternativa é realizar o comando *int* mais de uma vez. Caso o usuário sinta-se muito confuso para escrever em apenas uma linha, é sugerido que para cada etapa de integração o resultado seja salvo em uma célula (variável) e se execute a integração quantas vezes for necessário

Exemplo

```
>> S=x^2+y+z;

>> ix=int(S,x,0,2) %integramos a função salva na variável S em re-
lação a x e salvamos o resultado na variável ix%

ix =

2*y + 2*z + 8/3

>> ixy=int(ix,y,0,3) %integramos o resultado ix em relação a y e o
salvamos na variável ixy%

ixy =
```

```
6*z + 17

>> ixyz=int(ixy,z,0,1) %integramos o resultado ixy em relação a z
e o salvamos na variável ixyz%

ixyz =

20
```

Exercícios de Fixação

1 - Derive a seguinte expressão e calcule sua integral definida:

$$\int_0^5 \frac{1dx}{0.8x^2+0.5x+2}$$

2 - Defina x como uma variável simbólica e crie as duas expressões simbólicas a seguir $\boldsymbol{S} = x^3 - 9x^2 + 27x - 27$ e $\boldsymbol{S1} = (x+3)^3 - x^2 - 5x - 12$ e execute as seguintes operações (use o comando *subs* para avaliar o valor numérico de todas as operações tomando x=10).

a) S.S1

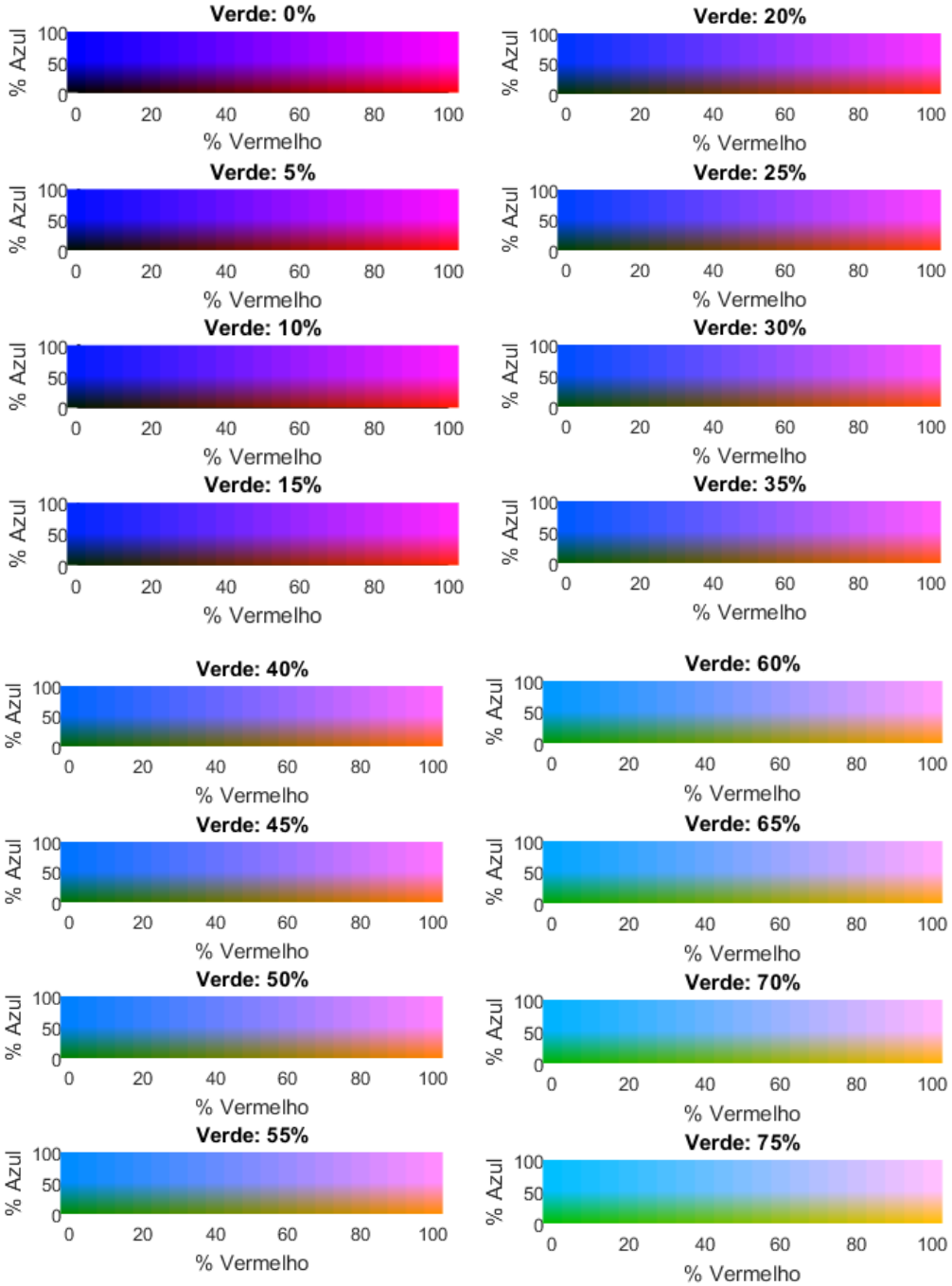
b) $\frac{S}{S1}$

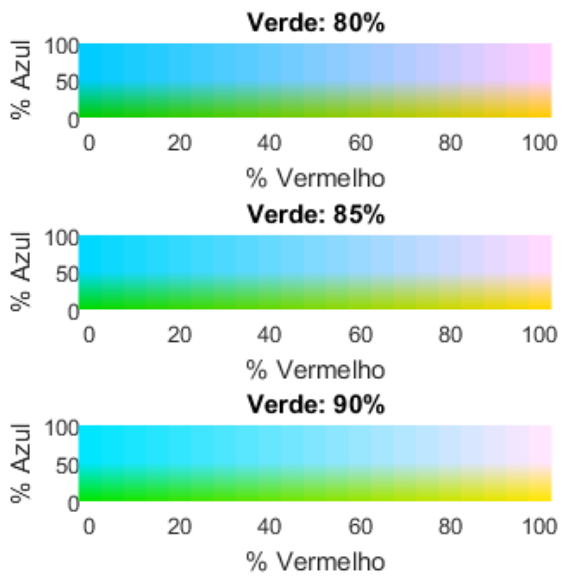
c) S +S1

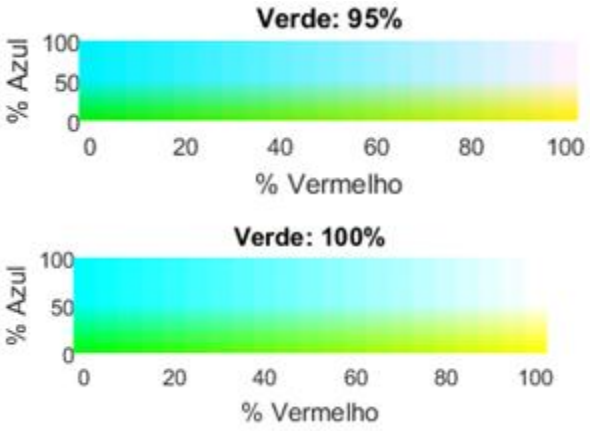
3 - Com o uso de operações simbólicas, calcule a integral indefinida $\boldsymbol{I} = \int e^{2x}\sqrt{2-e^{2x}}$ e a derivada de $e^{2x}\sqrt{2-e^{x}}$

Apêndice A - Cores RGB

Neste apêndice, são encontradas mapas de cores de acordo com sua porcentagem RGB. Um laranja claro, por exemplo pode ser tomado em Vermelho 100%, Verde 55%, Azul 0%, correspondendo ao vetor [1 .55 0]







Para construir escalas de cinza utilize as mesmas porcentagens em todas as coordenadas do vetor de cor. O branco é [1 1 1]. O preto é [0 0 0], então um cinza escuro pode ser escrito por [.15 .15 .15] e um cinza claro por [.8 . 8 .8]

Apêndice B - Algoritmos para gerar animações

1. Somente visualizando sequência de frames

Exemplo

```
x=linspace(0,10,100);
t=linspace(0,4,20);

for (i=1:20)
    y=t(i)*(x.^2)/100;
    plot(x,y);
    grid;
    xlim([-1 4]);
    ylim([0 1]);
    pause(4/20);
end
```

No algoritmo, definimos uma variável x e uma variável t. antes de iniciar o laço que vai apresentar o gráfico de maneira animada. Dentro do laço, y se mostra como função de x e t. Uma forma de representar as três coordenadas do problema, seria mostrar um gráfico tridimensional que revele todas as configurações possíveis dentro de um domínio para os valores de y, x e t.

Estamos, no entanto, particularmente interessados em visualizar o comportamento de uma coordenada y em função da coordenada y para cada um dos valores de t, de modo sequencial.

Devemos estabelecer que os limites dos eixos permaneçam fixos no decorrer do laço e utilizar a função pause(tempo em segundos) para determinar um tempo de pausa entre uma rodada do loop e outra.

2. Salvando um arquivo

Exemplo

```
clear
x=linspace(0,10,100);
t=linspace(0,4,20);
J=1;
for (i=1:20)
    y=t(i)*(x.^2)/100;
    plot(x,y);
```

```
    grid;
    xlim([-1 4]);
    ylim([0 1]);
    F(J)=getframe(gcf);
    J=J+1;

end

video = VideoWriter ('NomeDoArquivo','PARÂMETRO');
video.FrameRate=20;
open(video);
writeVideo(video,F);
close(video);
```

Substitua 'NomeDoArquivo' pelo nome de arquivo que desejar,

Substitua 'PARÂMETRO' para um dos parâmetros abaixo, de acordo com a necessidade:

PARÂMETRO	Função
'MPEG-4'	Gera arquivo em .mp4
'Motion JPEG AVI'	Gera arquivo em .avi a partir de imagens JPEG
'Uncompressed AVI'	Gera um arquivo em .avi não comprimido.

Com algoritmo semelhante, inicializamos uma variável J que atua na construção de um vetor que armazena os dados referentes a captura de frames a cada ciclo do laço, variável F.

No fim do comando *for,* a primeira linha de comando atribui à *video* a função videowriter que como entrada tem o nome do arquivo e o tipo de arquivo que será gerado.

Video.FrameRate é uma função que estabelece a quantidade de frames por segundo que são mostrados no vídeo. Esse parâmetro deve ser ajustado de acordo com a quantidade de frames que foram gerados e a velocidade esperada no vídeo resultante.

Os demais comandos serão os responsáveis por armazenar os frames na variável *video*.