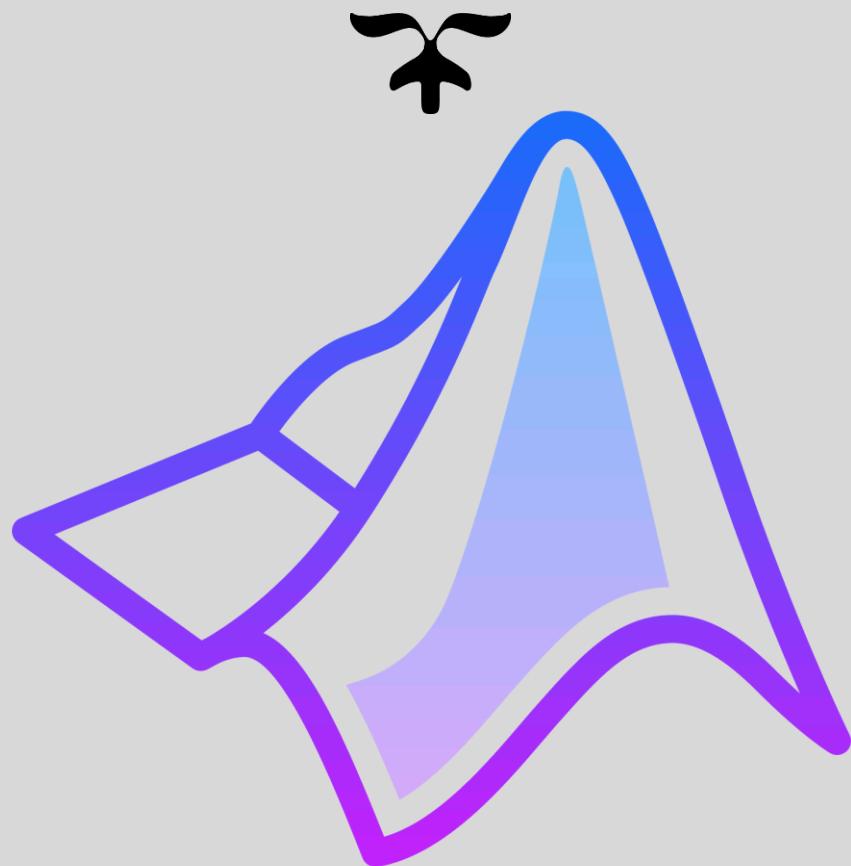




MINICURSO MATLAB

A apostila do minicurso 1.0



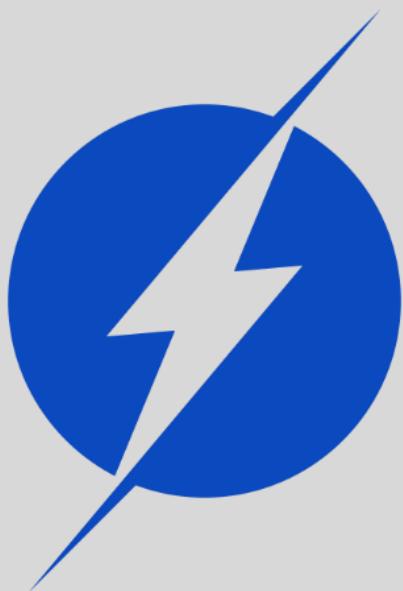
2019

PROGRAMA DE EDUCAÇÃO TUTORIAL – ENGENHARIA ELÉTRICA – UNIVERSIDADE
FEDERAL DE MINAS GERAIS

Apostila

Minicurso de MATLAB 1.0

Curso ministrado pelos alunos do Programa de
Educação Tutorial do Curso de Engenharia Elétrica
da Universidade Federal de Minas Gerais
PETEE UFMG



Belo Horizonte, Março de 2019

Prefácio

O Software

O nome MATLAB vem de laboratório de matrizes (Matrix Laboratory), e não de matemática, como alguns pensam. Mesmo com esse nome, suas funcionalidades e utilidades vão muito além disso.

MATLAB é um software interativo de alto nível, muito utilizado para desenvolvimento e implementação de algoritmos numéricos e simbólicos. Serve para construção e análise de imagens, gráficos, sistemas e simulações. Tem capacidade de atender vários tipos de usuários, do básico ao mais avançado.

É integrado ao Simulink, uma ferramenta poderosa de simulação e modelagem de sistemas, capaz de simular desde simples equações diferenciais até complexos sistemas mecânicos e elétricos.

História

A primeira versão do MATLAB foi desenvolvida no final dos anos 1970 na Universidade do Novo México com o objetivo de dar acesso a pacotes para Fortran sem requerer conhecimento dessa linguagem. Então, o programa era uma ferramenta bastante primitiva que servia apenas para fazer algumas operações numéricas com matrizes, daí o nome. Não possuía linguagem própria, toolboxes, arquivos ".m", tampouco parte gráfica. O objetivo era apenas auxiliar estudantes em aulas como Análise Numérica e Álgebra Linear. Como se sabe, naquela época, não havia editor de texto, terminais, monitores ou discos de memória. Os programas eram escritos e lidos em cartões perfurados, nos quais a lógica binária era determinada por ter furo ou não. Após processado, a saída do programa perfurava outro cartão.

Em 1983, um engenheiro percebeu o potencial comercial da ferramenta e juntou-se ao desenvolvedor para reescrever o programa. Nessa época, já estavam aparecendo no mercado os primeiros computadores pessoais (PCs). Em 1984, a equipe reescreveu o MATLAB em C utilizando um PC de 256kB de memória, sem disco rígido e a empresa Mathworks foi criada. A partir de então, o software foi gradativamente ganhando mercado e novas versões foram criadas.

No ano 2000, quando na versão 6, o MATLAB foi novamente reescrito para basear-se no LAPACK, um conjunto de bibliotecas para manipulação de matrizes.

Nota

Este material é uma reformulação da nossa apostila PETEE UFMG referente ao minicurso MATLAB 1.0 de março de 2017. A apostila foi totalmente reestruturada. O objetivo da reformulação é, sobretudo, fazer da apostila mais objetiva, clara e didática, por meio de melhores detalhamentos e explicações.

É um material de estudos criado com o intuito de auxiliar em um dos minicursos ministrados por nós do Programa de Educação Tutorial – Engenharia Elétrica da Universidade Federal de Minas Gerais: MATLAB 1.0.

O nosso objetivo é ajudar a todos que tenham interesse em aprender sobre o MATLAB, fazendo da apostila um guia básico para entender melhor o funcionamento do software e suas ferramentas.

É com muito orgulho e esmero que realizamos esta atividade.

Arthur Henrique Dias Nunes

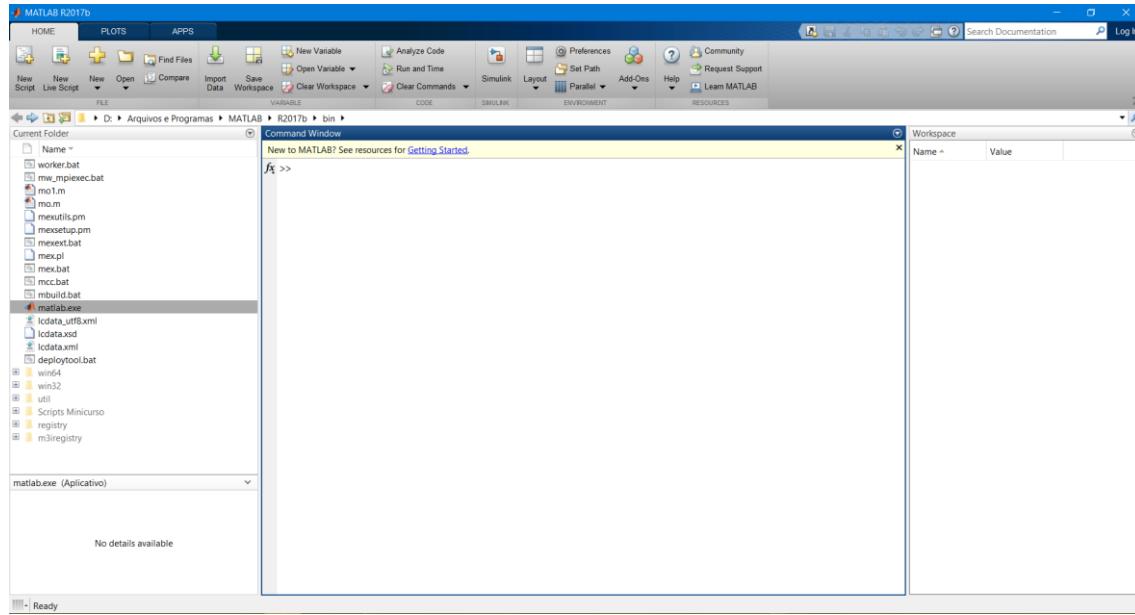
Sumário

1 O Ambiente de Trabalho	3
1.1 Command Window	4
1.2 Command History	4
1.3 Workspace.....	5
1.4 Variable Editor	6
1.5 Current Folder.....	6
1.6 Editor	7
2 Estrutura de Dados.....	8
2.1 Variáveis.....	8
2.1.1 Arranjos e Matrizes	9
2.1.2 Strings.....	11
2.1.3 Simbólicas	12
2.1.4 Complexos.....	13
2.1.5 Structs.....	13
2.1.6 Vetores de Células.....	14
2.2 Constantes.....	15
3 Comandos e Funções.....	17
3.1 Atalhos de Sintaxe	17
3.2 Comandos Básicos	17
3.3 Funções Básicas	18
4 Operadores.....	25
4.1 Aritméticos	25
4.2 Lógicos	26
4.3 Relacionais	28
4.4 Prioridades.....	28

5 Controle de Fluxo	30
5.1 Estruturas Condicionais	30
5.2 Estruturas de Repetição	31
5.3 Desvio Incondicional	32
5.3.1 Break	32
5.3.2 Continue	33
6 Scripts e Arquivos .m	35
6.1 Declaração de Funções	36
7 Gráficos	40
7.1 Bidimensionais.....	40
7.2 Tridimensionais.....	46
7.3 Comandos Auxiliares	51
7.4 Textos, Títulos e Rótulos	53
8 Polinômios	58
8.1 Raízes	58
8.2 Produto e Divisão	59
8.3 Avaliação	59
8.4 Frações Parciais	59
8.5 Ajuste Polinomial	60
9 Simulink: Introdução	64
9.1 O Ambiente.....	64
9.2 Modelagem	66
9.2.1 Exemplo.....	66
Resolução de Exercícios.....	72
Bibliografia	105

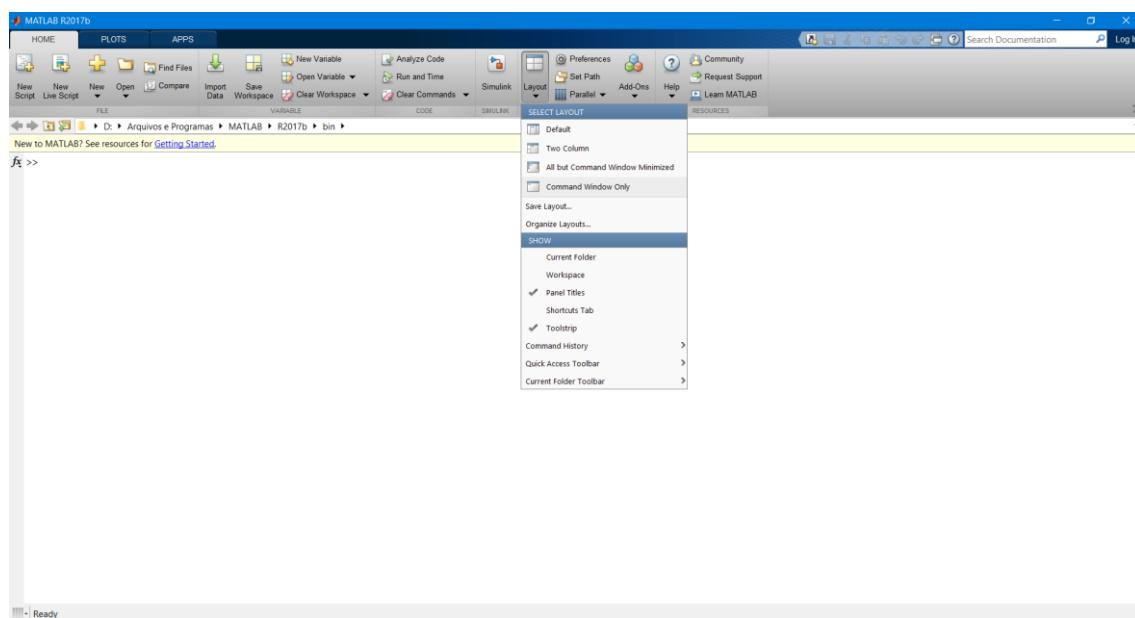
O Ambiente de Trabalho

O ambiente de trabalho MATLAB é composto, basicamente, de grandes barras de opções na parte superior e algumas janelas. Cada janela possui sua funcionalidade que explicaremos a seguir. O visual default é este:



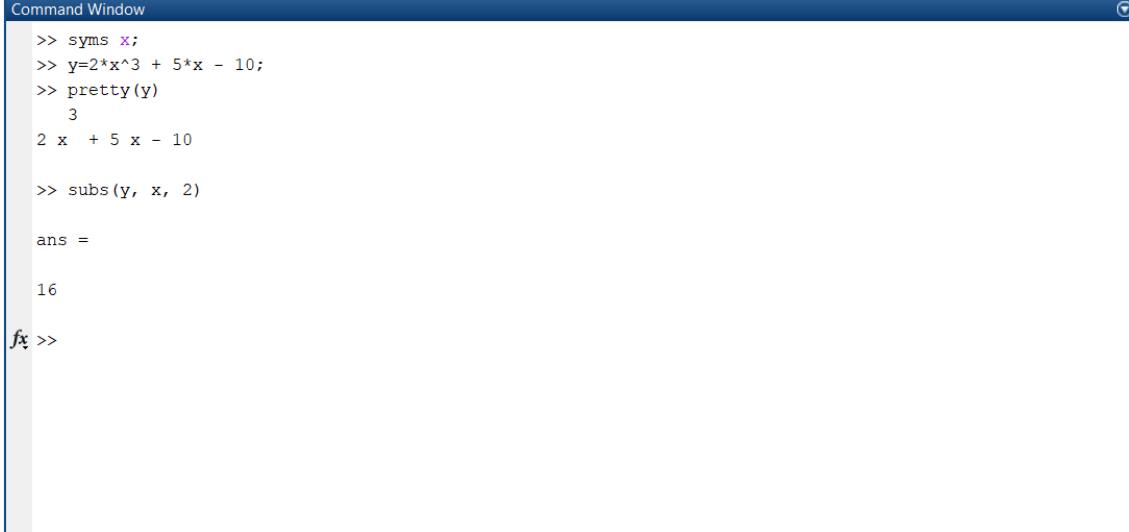
Layout default

Na primeira barra de opções “Home”, localizada na parte superior há a opção layout, na qual se é possível customizar o ambiente de trabalho da melhor forma que o usuário julgar. Também é possível customizar clicando nas janelas e as arrastando.



Exemplo de customização de layout

1.1 Command Window



```
Command Window
>> syms x;
>> y=2*x^3 + 5*x - 10;
>> pretty(y)
      3
    2 x  + 5 x - 10

>> subs(y, x, 2)

ans =
16

fx >>
```

Exemplo Command Window

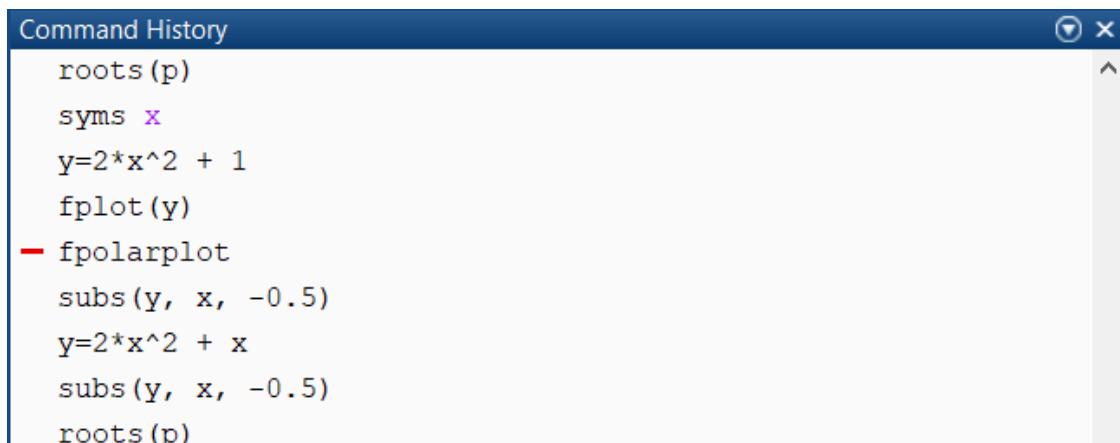
É análoga ao PowerShell ou CMD, serve para executar os comandos do MATLAB, mais recomendado para aqueles de rápida execução, uma vez que existe o editor de texto para programas e códigos maiores e mais complexos.

Dica: Existem alguns atalhos para facilitar a execução dos comandos da Command Window:

- ↑ **linha anterior;**
- ↓ **linha posterior**
- ← **move para a esquerda;**
- **move para a direita;**
- Ctrl** ← **move uma palavra para a esquerda;**
- Ctrl** → **move uma palavra para a direita;**
- Home** **move para o começo da linha;**
- End** **move para o final da linha;**
- Backspace** **apaga um caractere a esquerda;**
- Del** **apaga um caractere a direita;**
- Shift+enter** **próxima linha sem executar;**

1.2 Command History

Aparece quando se pressiona a seta para cima na Command Window, é o seu histórico de comandos, sendo salvo mesmo que feche o programa ou desligue o computador.



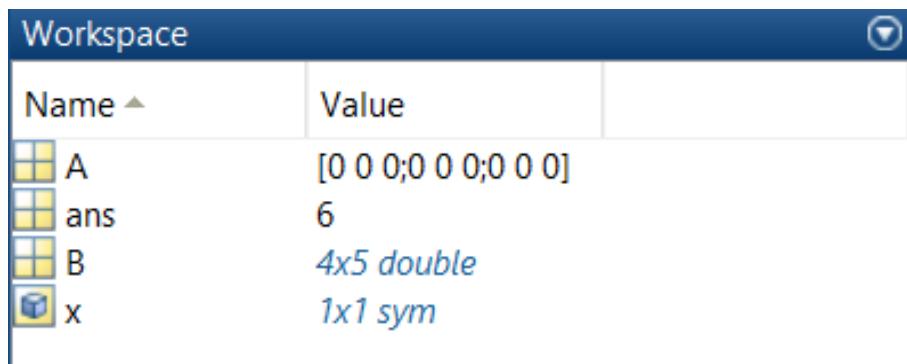
```

Command History
roots(p)
syms x
y=2*x^2 + 1
fplot(y)
fpolarplot
subs(y, x, -0.5)
y=2*x^2 + x
subs(y, x, -0.5)
roots(p)

```

Command History

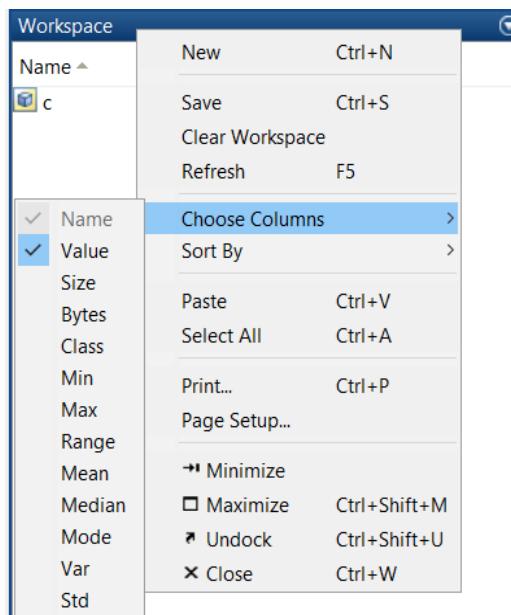
1.3 Workspace



Name	Value
A	[0 0 0; 0 0; 0 0]
ans	6
B	4x5 double
x	1x1 sym

Exemplo de Workspace

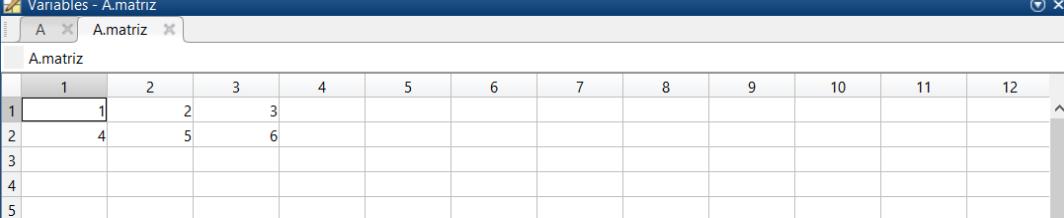
Mostra as variáveis e as estruturas existentes na memória. Clicando duas vezes em cima de uma variável é possível editá-la. Clicando com o botão direito em cima da barra do workspace, é possível configurá-lo.



Configurando o Workspace

1.4 Variable Editor

É a janela aberta ao se clicar duas vezes em alguma variável do workspace, aqui se é possível editar e visualizar melhor a variável por meio de planilhas.

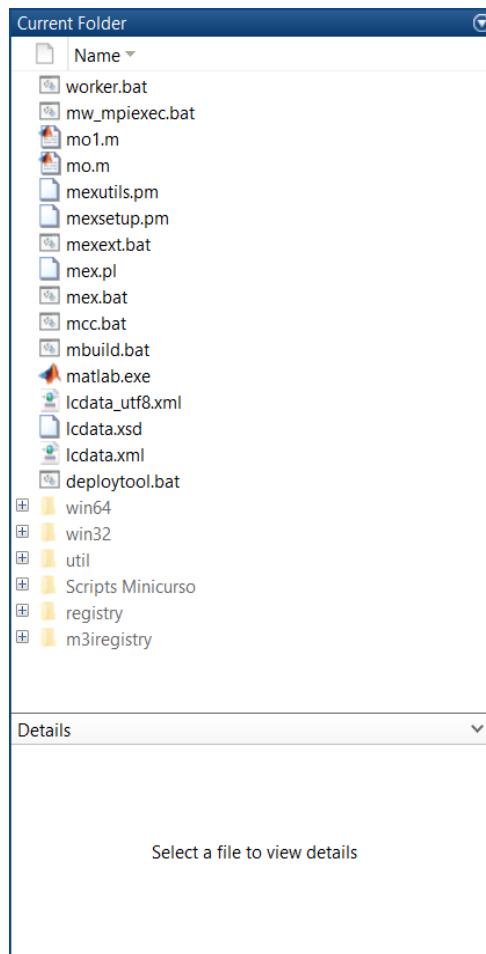


A	1	2	3	4	5	6	7	8	9	10	11	12
1	1		2		3							
2		4		5		6						
3												
4												
5												

Variable Editor

1.5 Current Folder

Janela que mostra o diretório corrente e os seus arquivos. É possível navegar nas pastas por esse janela, ou também usando, na Command Window, os comandos `cd <diretório>` para entrar, `cd ..` para voltar, `dir` para listar os arquivos e `pwd` para listar o caminho até o diretório local. Veremos a importância dos diretórios correntes quando formos criar scripts .m e declarar funções.



Current Folder

1.6 Editor



Barra de tarefas do editor

A screenshot of the MATLAB Editor window. The title bar says "Editor - D:\Arquivos de Programas\MATLAB\R2017b\distancia2pontos.m". The code in the editor is:

```
1 function resposta = distancia2pontos(x1, y1, x2, y2)
2 %distancia2pontos calcula a distância entre dois pontos
3 %Exemplo de descrição da função
4 %conteúdo e etc...
5 - resposta = sqrt( (x1-x2)^2 + (y1-y2)^2 );
6 -
7 end
```

Editor

É o editor de texto do MATLAB, possui barra de tarefas com opções auxiliares. Aberto ao se clicar em “New Script” ou ao abrir arquivos .m. Nele é possível criar e executar algoritmos, criar arquivos de funções declaradas e edições de roteiros (scripts) em geral. Será abordado também no capítulo destinado a scripts e arquivos .m.

Estrutura de Dados

Todas as variáveis e constantes são arranjos, com uma ou mais dimensões, até simples variáveis com apenas um elemento são tratadas como arranjos 1x1. Com isso, podemos fazer também strings, structs, células, containers e sets.

Sua estrutura de dados foi construída de modo a obter alta eficiência em trabalhar com operações matriciais diretamente, sendo melhor que trabalhar com laços.

2.1 Variáveis

Como toda linguagem de programação, o MATLAB faz o uso de variáveis. Porém, diferente de linguagens padrões como C e Java, no MATLAB não é necessário declarar o tipo da variável, basta atribuir seu valor em tempo real.

Exemplo:

```
>>a = 'a'; % a é uma variável do tipo char  
>>a = 2.2; % a é uma variável do tipo double
```

Os nomes das variáveis podem ter números, letras e '_', sendo obrigatoriamente primeiro uma letra e no máximo 31 caracteres.

Além disso, seus valores numéricos podem ser especificados com uma sequência numérica precedida ou não de "+" ou "-". O ponto "." indica decimais. Potências de 10 podem ser escritas usando e<expoente> ou d<expoente>, como por exemplo: 5.331e15, -2.2d-2.

Existem diferentes tipos de dados para representação e armazenamento de variáveis. Também podem ser convertidos em outros tipos de dados, usando o nome do tipo desejado e a variável como parâmetro. Exemplo:

```
>> x = 1.23 %por padrão, x será double  
x =  
1.2300  
>> a = int8(x)  
a =  
int8  
1
```

O comando class(x) retorna o tipo de dado de x em formato de string.

Alguns tipos de dados:

Tipo	Bytes	Descrição
double	8	Tipo padrão de armazenamento com precisão dupla
single	4	Armazenamento com metade da precisão double
int8	1	Inteiro com sinal, de 8 bits
int16	2	Inteiro com sinal, de 16 bits
int32	4	Inteiro com sinal, de 32 bits
int64	8	Inteiro com sinal, de 64 bits
uint8	1	Inteiro não negativo, de 8 bits
uint16	2	Inteiro não negativo, de 16 bits
uint32	4	Inteiro não negativo, de 32 bits
uint64	8	Inteiro não negativo, de 64 bits
logical	1	Tipo de dado booleano
char	2	Caractere único

O software também pode ser usado como uma calculadora científica, e portanto, possui uma variável de controle chamada ans. O nome vem de answer, e serve para armazenar a resposta de alguma operação, caso não seja atribuída a nenhuma outra variável. Pode ser usada nos cálculos e operações normalmente. Diferente de algumas calculadoras como a HP50G, ans armazena somente a última resposta.

Exemplo:

```
>>1+1
ans=
2
>>ans + 5
ans=
7
```

2.1.1 Arranjos e Matrizes

Pode se atribuir um arranjo de várias formas, utilizando colchetes e especificando elementos, utilizando dois pontos, usando a função cat, etc. Por exemplo, iremos criar um arranjo B que vai de 2 a 10 de 0.5 em 0.5, um arranjo C contendo 5 elementos igualmente espaçados entre 1 e 3 e um arranjo A contendo elementos especificados. Usaremos três sintaxes distintas. A primeira é a notação de dois pontos, onde representa intervalo, se usado somente 2:10 para B, B seria um arranjo de 2 a 10, com seus itens variando de 1 em 1, como usamos 2:0.5:10, determinamos que B será mesmo de 2 a 10, porém o passo, isto é, o intervalo entre um elemento e próximo será de 0.5. A segunda, é utilizando a função linspace, essa função cria um arranjo igualmente espaçado, recebendo três parâmetros, respectivamente, o primeiro valor, o último valor e a quantidade de elementos, muito útil na criação de arranjos para construção de gráficos:

```
>>B= 2:0.5:10;  
>>C=linspace(1, 3, 5);  
>> A=[1, 2, 3; 4, 5, 6];
```

Neste último caso, por exemplo, o MATLAB já computa o espaço de memória necessário para uma matriz 2x3 chamada A e já são atribuídos os valores. Se executarmos o mesmo comando, porém sem o ponto e vírgula, será exposito o resultado da operação, que é a atribuição da variável:

```
A=  
1 2 3  
4 5 6
```

Também é possível atualizar o tamanho, bem como os valores do arranjo. Veja os exemplos usando a mesma matriz A anterior.

Atualizando o valor de um elemento:

```
>>A(1, 1)=0  
A =  
0 2 3  
4 5 6
```

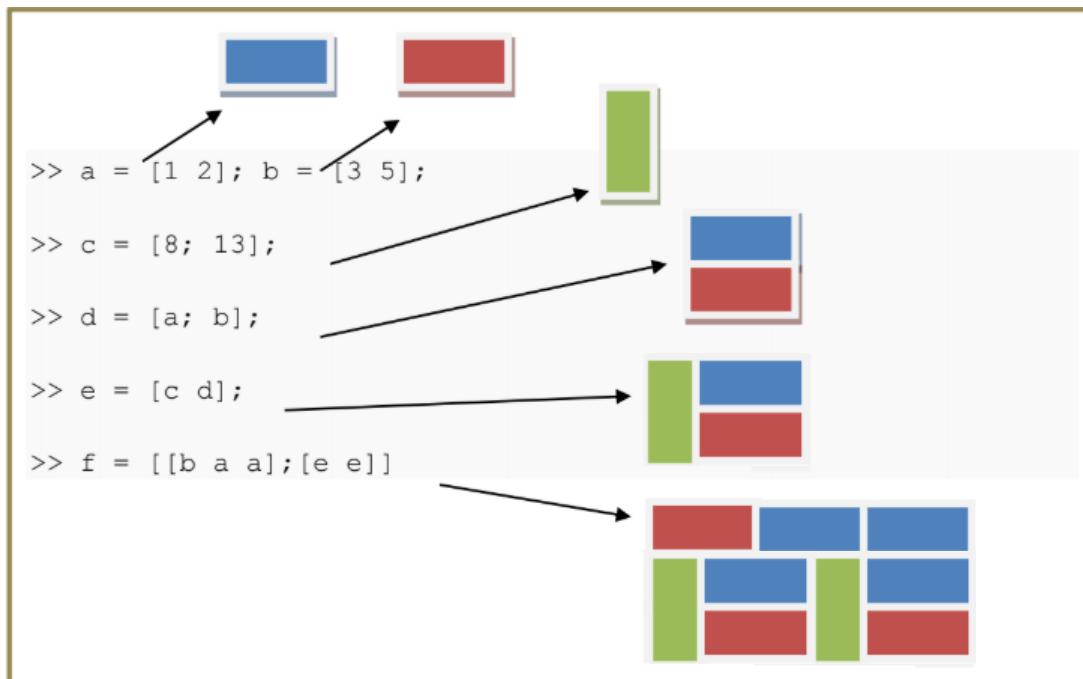
Expandindo o arranjo, note que os outros elementos serão zero:

```
>>A(3, 2)=1  
A =  
0 2 3  
4 5 6  
0 1 0
```

Recortando a matriz, “[]” indica espaço vazio e “:” indica tudo:

```
>>A(:, 2)=[ ]  
A =  
0 3  
4 6  
0 0
```

Ainda existem outras formas de se criar matrizes e arranjos. Veja mais este exemplo de criação de matrizes por concatenação direta:



Criação de matrizes por concatenação direta

2.1.2 Strings

Strings são cadeias lineares de caracteres, e portanto, também são vetores, criados utilizando aspas simples. Não são tratados como números, no entanto, o MATLAB possui várias funções e operações para as strings. Exemplos de manipulação de strings:

```

>>Nome= 'John'

Nome=
    'John'

>>Nome(1)='L'

Nome=
    'Lohn'

>>Nome(2)= 'a'+ 4

Nome=
    'Lehn'

```

Podem também ser feitas matrizes, normalmente:

```

>> Texto = ['SATOR'; 'AREPO'; 'TENET'; 'OPERA'; 'ROTAS']

Texto =
5×5 char array

    'SATOR'

```

'AREPO'

'TENET'

'OPERA'

'ROTAS'

representação decimal dos caracteres																
33	!	45	-	57	9	69	E	81	Q	93]	105	i	117	u	
34	"	46	.	58	:	70	F	82	R	94	^	106	j	118	v	
35	#	47	/	59	;	71	G	83	S	95	_	107	k	119	w	
36	\$	48	0	60	<	72	H	84	T	96	'	108	l	120	x	
37	%	49	1	61	=	73	I	85	U	97	a	109	m	121	y	
38	&	50	2	62	>	74	J	86	V	98	b	110	n	122	z	
39	,	51	3	63	?	75	K	87	W	99	c	111	o	123	{	
40	(52	4	64	@	76	L	88	X	100	d	112	p	124		
41)	53	5	65	A	77	M	89	Y	101	e	113	q	125	}	
42	*	54	6	66	B	78	N	90	Z	102	f	114	r	126	~	
43	+	55	7	67	C	79	O	91	[103	g	115	s			
44	,	56	8	68	D	80	P	92	\	104	h	116	t			

representação decimal dos caracteres																
192	À	200	È	208	Ð	216	Ø	224	à	232	è	240	ð	248	ø	
193	Á	201	É	209	Ñ	217	Ù	225	á	233	é	241	ñ	249	ù	
194	Â	202	Ê	210	Ô	218	Ú	226	â	234	ê	242	ð	250	ú	
195	Ã	203	Ë	211	Ó	219	Û	227	ã	235	ë	243	ô	251	û	
196	Ä	204	Ï	212	Ö	220	Ü	228	ä	236	ï	244	ð	252	ü	
197	Å	205	Í	213	Ӧ	221	Ӯ	229	å	237	í	245	ð	253	ӱ	
198	Æ	206	Î	214	Ӯ	222	Ӱ	230	æ	238	î	246	ö	254		
199	Ҫ	207	Ӥ	215	Ӯ	223	Ӱ	231	ܶ	239	Ӯ	247	ܰ	255	ܱ	

2.1.3 Simbólicas

Existem também, as variáveis simbólicas. Com esse tipo de dado, é possível fazer funções e expressões matemáticas, avaliar a função em um ponto, derivar, dentre outros artifícios. Exemplo:

```
>> syms x
>> y = 2*x + 4;
>> subs(y, x, -2)
ans =
0
```

Existem diversas funções para variáveis simbólicas, vamos usar como outro exemplo, simplificar a expressão $\sin^2(x) + \cos^2(x)$, que será igual a 1.

```
>> syms x
>> f=sin(x)^2 + cos(x)^2
f =
cos(x)^2 + sin(x)^2
>> simplify(f)
ans =
1
```

2.1.4 Complexos

Números complexos são o conjunto dos números reais e imaginários, podem ser representados por $a+b*i$, sendo i , por definição, a raiz quadrada de -1.

O MATLAB realiza operações com os complexos normalmente. O tipo de dado utilizado é diferente do double comum de 8 bytes, é utilizado o double(complex) de 16 bytes, como se fosse um campo double para a parte real e outro para a parte imaginária. Veja o exemplo pedindo para que o MATLAB compute raiz quadrada de -1, usando a função sqrt:

```
>> sqrt(-1)
ans =
0.0000 + 1.0000i
>> x=ans^2
x =
-1
```

Existem duas constantes pré-definidas, i e j , as quais possuem o mesmo valor de ans no exemplo anterior, $0 + 1*i$.

Portanto, podemos fazer todos os procedimentos normalmente com os complexos, criar matrizes, arranjos, operações e etc.

Lembrando que o operador aspas simples representa o complexo conjugado e também transposição de matrizes. Então, se há uma matriz de complexos e usamos o operador aspas simples, cada complexo é conjugado e a matriz transposta ao mesmo tempo. Exemplo:

```
>>A = [i, 2*i, 3+i];
>>A'
ans =
0.0000 - 1.0000i
0.0000 - 2.0000i
3.0000 - 1.0000i
```

2.1.5 Structs

Uma struct é uma estrutura que permite o armazenamento de dados potencialmente diferentes, na qual cada campo é identificada por um nome.

A construção pode ser feita por atribuição direta:

```
>> patient.name = 'John Doe';
>> patient.billing = 127.00;
>> patient.test = [79, 75, 73; 180, 178, 177.5; 220, 210,
205];
```

Como também utilizando a sintaxe:

```
>> patient = struct('name', ['John Doe'], 'billing',
[127.00], 'test', [79, 75, 73; 180, 178, 177.5; 220, 210,
205]);
```

Ao se digitar patient, todos os campos são expostos;

```
patient =
struct with fields:
    name: 'John Doe'
    billing: 127
    test: [3×3 double]
```

2.1.6 Vetores de Células

São arranjos de ponteiros, isto é, cada posição do vetor é um endereço de memória, que podem apontar para qualquer tipo de dado. Nos endereços estarão os dados úteis ao usuário. É importante perceber que, diferente de uma matriz comum, cada espaço é um ponteiro. A sintaxe utilizada são as chaves. Exemplo:

```
>> CA(1,1) = {ones(3)};
>> CA(1,2) = {'cell array'};
>> CA(2,1) = {[2]};
>> CA(2,2) = {[i]};
>> CA
CA =
2×2 cell array
{3×3 double}    {'cell array'        }
{[2]}           {[0.0000 + 1.0000i] }.
```

Para podermos acessar o conteúdo apontado, basta identificarmos o conteúdo desejado dentro das chaves.

```
>> CA{1,1}
ans =
1     1     1
1     1     1
1     1     1
```

2.2 Constantes

Também existem constantes pré definidas, usadas por funções e também pelo usuário. Podem ser alteradas, mas não é recomendado.

Constante	Valor	Tamanho	Bytes	Classe	Descrição
i	0+1i	1x1	16	double(complex)	complexo raiz(-1)
j	0+1i	1x1	16	double(complex)	complexo raiz(-1)
eps	2,22e-16	1x1	8	double	Menor valor ao somar a 1 para que se torne o próximo ponto flutuante maior que 1
realmin	2,23e-308	1x1	8	double	Menor valor de ponto flutuante
realmax	1.8e+308	1x1	8	double	Maior valor de ponto flutuante
NaN	NaN	1x1	8	double	Valor não numérico
pi	3.1416	1x1	8	double	Valor numérico de pi
inf	inf	1x1	8	double	infinito
true	1	1x1	1	logical	Booleano verdadeiro
false	0	1x1	1	logical	Booleano falso
eye(m)	mxm double	mxm	8*m^2	double	Matriz Identidade
ones(m, n)	mxn double	mxn	8*m*n	double	Matriz de 1s
zeros(m, n)	mxn double	mxn	8*m*n	double	Matriz de 0s

Obs: eye, ones e zeros são funções que retornam matrizes, mas de certa forma, podem ser pensados como constantes, uma vez que tem a mesma funcionalidade.

Exercícios:

- 1) Utilizando o MATLAB como uma simples calculadora, compute:
 - 5.17e-5 x 17
 - 13.711²
 - (resultado anterior) ÷ 5.17e-5
 - 1 ÷ -1.6022e-19
 - 50 ÷ 342,3

- 2) Use novamente o MATLAB como uma calculadora e peça para que ele calcule $0 \div 0$ e depois $10 \div 0$.
- 3) Utilizando a Command Window, atribua as seguintes variáveis:
 - a) Um escalar chamado NmrMatricula com valor de 201900
 - b) Uma matriz linha (vetor) de tamanho 1x5, chamada Notas, com todos os valores iguais a 0
 - c) Uma string chamada Ocorrencias preenchida com: O aluno não possui ocorrências
 - d) Uma arranjo 2x1 de struct, chamada Aluno, com cada struct com seus campos valendo:
 - i) Índice 1,1 campo nome = Jonatan
 - ii) Índice 1,1 campo matricula = NmrMatricula (letra a)
 - iii) Índice 1,1 campo notas = Notas (letra b)
 - iv) Índice 1,1 campo ocor = Ocorrencias (letra c)
 - v) Índice 2,1 campo nome = Patricia
 - vi) Índice 2,1 campo matricula = NmrMatricula (letra a)
 - vii) Índice 2,1 campo notas = Notas (letra b)
 - viii) Índice 2,1 campo ocor = Ocorrencias (letra c)
- 4) Utilizando o arranjo de struct da questão anterior, na aba Workspace, clique duas vezes no arranjo e observe a variável na janela aberta Variable Editor.
- 5) Ainda utilizando o arranjo de struct da questão 2, atualize os valores de seus campos para:
 - a) Campo matricula de Jonatan = NmrMatricula + 1
 - b) Campo notas de Jonatan = 10 8 9.5 7 10
 - c) Campo matricula de Patricia = NmrMatricula + 2
 - d) Campo notas de Patricia = 10 10 3 9 7.8
 - e) Campo ocor de Patricia = duas brigas em sala de aula
- 6) Repita o procedimento da questão 3) e observe o que mudou
- 7) Crie uma matriz A e B de modo que:
 - a) $A = 2+2i, 0, 7i, 13+2i$
 - b) B de modo que b_{ij} seja igual ao conjugado de a_{ij} , utilizando o operador aspas simples. Note que não é para que a matriz A seja transposta, mesmo utilizando o operador aspas simples.
- 8) Utilizando a função subs, $\text{subs}(\text{Expressão}, \text{variável}, \text{valor})$ substitui na expressão a variável indicada pelo valor indicado, crie uma expressão simbólica $y = 2x^2 - 30x + 100$ e substitua nessa expressão x por 10.
- 9) Utilizando a mesma função e a mesma expressão anterior, substitua agora, x por 5.
- 10) Utilizando ainda a mesma função e a mesma expressão anterior, substitua agora, x por 0.

Comandos e Funções

Basicamente, comandos são palavras chave que realizam ações pré-determinadas. Exemplo: o comando `clear` remove as variáveis do Workspace. Já as funções são procedimentos que retornam dados, dependendo dos parâmetros passados.

$$[\text{Variáveis retornadas}] = \text{Função} ([\text{Parâmetros}])$$

Por vezes, comandos são tratados como funções e vice-versa, suas definições e utilização são similares e se misturam. O importante é saber encontrar e entender o procedimento que precisar usar. Ambos podem ser executados tanto na Command Window quanto em um editor de texto. Um dos mais importantes é o comando `help`, porque assim, o usuário pode se informar melhor sobre qualquer comando ou função que necessitar. Por isso vamos explicá-lo agora.

O comando é muito útil e completo, com explicações, referências, links e exemplos. Basta digitar na Command Window `help <expressão>`. A expressão geralmente é uma palavra chave, ou comando/função que se deseja conhecer mais. Vale a pena usá-lo.

Há também o comando `lookfor <palavra chave>`, que procura por funções que possuem a palavra chave em sua descrição.

No tópico destinado à criação de funções, veremos que também é possível colocar descrições nas funções criadas pelos usuários, que aparecerão nos comandos `help` e `lookfor` normalmente.

3.1 Atalhos de Sintaxe

- ❖ ; no final da linha, suprime a exibição do resultado da operação;
- ❖ % Indicar comentários;
- ❖ ... continuar o comando na próxima linha;
- ❖ , separa comandos na mesma linha;

3.2 Comandos Básicos

- ❖ `help` exibe um texto de ajuda com links para mais aprofundamento;
- ❖ `help <função ou comando>` exibe um texto explicando a função ou comando com exemplos e outros tópicos relacionados;
- ❖ `clear <variável1> <variável2> <...>` apaga as variáveis especificadas;
- ❖ `clear` remove todas as variáveis;
- ❖ `clearvariables` mesmo que `clear`;
- ❖ `clear global` remove todas as variáveis globais;
- ❖ `clearfunctions` apaga as funções compiladas de M e MEX;

- ❖ `clearall` libera toda a memória apagando variáveis globais e locais e funções;
- ❖ `clc` limpa a Command Window;
- ❖ `close` fecha todas as figuras abertas;
- ❖ `who` lista todas as variáveis;
- ❖ `whos` lista todas as variáveis exibindo detalhes;
- ❖ `format` muda a formatação numérica das variáveis;
- ❖ `cd <diretório>` vai para o diretório;
- ❖ `cd ..` vai para o diretório anterior;
- ❖ `dir` lista os arquivos no diretório corrente;
- ❖ `pwd` exibe o caminho até o diretório corrente;
- ❖ `<Variável>` Exibe o valor da variável selecionada;

3.3 Funções Básicas

Aqui, serão listadas algumas funções já prontas que são úteis aos usuários, existe a opção de se criar funções, e isto será abordado no tópico destinado aos scripts e arquivos .m.

Estrutura de dados:

- ❖ `X = menu('Título', 'Opção1', 'Opção2', ...)` é aberta uma caixa de diálogo para o usuário escolher entre as opções. Se a opção n é escolhida, X recebe o valor n. Se nenhuma opção é escolhida, X será 0.
- ❖ `class(X)` retorna a classe de X em formato string;
- ❖ `double(X)` converte X para o tipo double;
- ❖ `single(X)` converte X para o tipo single;
- ❖ `int8(X)` converte X para o tipo int8;
- ❖ `int16(X)` converte X para o tipo int16;
- ❖ `int32(X)` converte X para o tipo int32;
- ❖ `int64(X)` converte X para o tipo int64;
- ❖ `uint8(X)` converte X para o tipo uint8;
- ❖ `uint16(X)` converte X para o tipo uint16;
- ❖ `uint32(X)` converte X para o tipo uint32;
- ❖ `uint64(X)` converte X para o tipo uint64;
- ❖ `logical(X)` converte X para o tipo logical;
- ❖ `char(X)` converte X para o tipo char;
- ❖ `save('Nome.mat', 'Variável1', 'Variável2', ...)` Salve as variáveis atuais em um arquivo com o nome especificado, com a extensão .mat. Obs: todos os argumentos devem ser em strings.
- ❖ `load('Nome.mat')` Carrega o workspace salvo com o nome "nome.mat"). Caso salvo em formato padrão (.mat) pode ser atribuído a uma variável. Esta será uma struct contendo as variáveis do workspace salvo como campos. Exemplo

```
>> a=2; b=3;
>> save('var1', 'a', 'b')
>> clear, clc
>> x = load('var1.mat')
x =
  struct with fields:
```

```
a: 2
b: 3
```

Criação e manipulação de arranjos:

- ❖ `linspace(Primeiro valor, Último valor, Quantidade de elementos)` cria um arranjo com os elementos igualmente espaçados do primeiro ao último valor. Útil para criar arranjos usados na plotagem de gráficos;
- ❖ `ones(Linhas, Colunas)` cria uma matriz do tamanho especificado onde todos os elementos são preenchidos com 1;
- ❖ `zeros(Linhas, Colunas)` cria uma matriz do tamanho especificado onde todos os elementos são preenchidos com 0;
- ❖ `eye(Ordem)` cria uma matriz identidade quadrada da ordem especificada;
- ❖ `cat(Dimensão, Arranjo1, Arranjo2, ...)` Concatena arranjos, útil para criar arranjos de mais de duas dimensões;
- ❖ `size(Arranjo)` retorna um arranjo com as dimensões do arranjo;
- ❖ `mean(Arranjo)` arranjo n-1 com a média aritmética de cada vetor;
- ❖ `std(Arranjo)` arranjo n-1 com o desvio padrão de cada vetor ;
- ❖ `prod(Arranjo)` arranjo n-1 com o produto dos elementos de cada vetor;
- ❖ `max(Arranjo)` arranjo n-1 com o maior elemento de cada vetor;
- ❖ `min(Arranjo)` arranjo n-1 com o menor elemento de cada vetor ;
- ❖ `sort(Arranjo)` arranjo n-1 com cada vetor ordenado;
- ❖ `det(Matriz)` retorna o determinante da matriz;
- ❖ `trace(Matriz)` retorna o traço da matriz;
- ❖ `inv(Matriz)` retorna a inversa da matriz, caso exista;

Impressão e entrada:

- ❖ `input('mensagem')` exibe a mensagem e recebe um valor do teclado;
 - ❖ `inputdlg('mensagem')` exibe a mensagem e recebe um valor do teclado usando uma caixa de diálogo;
 - >>`w= input('insira o valor de w');`
 - ❖ `disp('mensagem única de saída')` imprime uma mensagem, recebe apenas um argumento, então se quiser partir a mensagem para, por exemplo, imprimir um valor, é necessário fazer da entrada em arranjo.
 - ❖ `fprintf('formato', variável)` Imprime a variável no formato especificado.
- Exemplos:**

```
>> w= input('insira o valor de w\n');
insira o valor de w
5
w=
5
>>disp (w)
5
>>disp(['w tem o valor de ', num2str(w)])
w tem o valor de 5
>>fprintf('w = %i\n', w)
```

```
w = 5
```

Dica: Existem sequências especiais de caracteres da função `fprintf`:

<code>%s</code>	<i>formato string da variável</i>
<code>%d</code>	<i>formato inteiro da variável</i>
<code>%i</code>	<i>formato inteiro da variável</i>
<code>%f</code>	<i>formato ponto flutuante da variável</i>
<code>%e</code>	<i>formato ponto flutuante em notação científica da variável</i>
<code>%g</code>	<i>formato mais compacto de %f ou %e</i>
<code>\n</code>	<i>quebra de linha</i>
<code>\t</code>	<i>tab</i>

Funções simbólicas:

- ❖ `diff(Expressão)` retorna a derivada;
- ❖ `int(Expressão)` retorna a integral;
- ❖ `compose(F, G)` compõe $F(G(x))$;
- ❖ `expand(Expressão)` Expande a expressão;
- ❖ `finverse(Expressão)` retorna a inversa da função;
- ❖ `pretty(Expressão)` expõe a função de forma mais bonita e organizada;
- ❖ `simplify(Expressão)` simplifica a expressão;
- ❖ `solve(Expressão)` encontra as raízes da expressão;
- ❖ `limit(Expressão, variável, valor)` Calcula o limite da expressão quando a variável tende ao valor especificado. Pode ser incluído um quarto argumento ('right' ou 'left' impondo que o limite é pela esquerda ou direita);
- ❖ `subs(Expressão, variável, valor)` substitui na expressão a variável indicada pelo valor indicado, Exemplo:

```
>>syms x y
>>z=2*x + 2*y
z=
2*x + 2*y
>>f=subs(z, x, 2)
f=
2*y + 4
>>subs(f, y, -2)
ans=
0
```

Complexos:

- ❖ `real(Complexo)` retorna a parte real do número complexo;
- ❖ `imag(Complexo)` retorna a parte imaginária do número complexo;
- ❖ `abs(Complexo)` retorna o valor absoluto/módulo de um número complexo;
- ❖ `angle(Complexo)` calcula o ângulo de um número complexo;

- ❖ `conj(Complexo)` retorna o complexo conjugado, mesmo que utilizar o comando aspas simples `complexo'`;

Randômicas:

- ❖ `rand` retorna número pseudo-aleatórios de distribuição uniforme entre 0 e 1;
- ❖ `randn` retorna número pseudo-aleatório com distribuição normal padrão(média 0 e variância 1);
- ❖ `randi(A, m, ...)` A pode ser um valor que será referente ao máximo valor que poderá retornar, ou um arranjo de dois valores, onde o primeiro será o menor e o segundo o maior, limitando os valores possíveis da distribuição. m e os seguintes serão as dimensões do arranjo de saída. Caso seja somente m, será uma matriz $m \times m$, caso tenha mais, quantos quiser, serão as dimensões do arranjo: Exemplo

```
>>randi([0, 10], 2, 2, 2)
```

retornará um arranjo $2 \times 2 \times 2$ com valores aleatórios de distribuição uniforme entre 0 e 10:

```
ans(:,:,1) =
```

```
8     1  
9    10
```

```
ans(:,:,2) =
```

```
6     3  
1     6
```

- ❖ `randperm(Quantidade)` realiza uma permutação aleatória de um vetor de valores inteiros de 1 até Quantidade, com diferença de 1 entre dois valores adjacentes. Ou seja, permuta os naturais a partir do 1 até Quantidade, como se tivesse criado um vetor= `linspace(1, Quantidade, Quantidade)`. Exemplo:

```
>>A=linspace(1, 10, 10)
```

```
A =
```

```
1     2     3     4     5     6     7     8     9  
10
```

```
>> randperm(10)
```

```
ans =
```

```
8     3     9     6     7     10    5     1     2  
4
```

Trigonométricas:

- ❖ `deg2rad(X)` converte X de graus para radianos;
- ❖ `rad2deg(X)` converte X de radianos para graus;

- ❖ `acos(X)` retorna o arco cosseno de X;
- ❖ `acosh(X)` retorna o arco cosseno hiperbólico de X;
- ❖ `acot(X)` retorna o arco cotangente de X;
- ❖ `acoth(X)` retorna o arco cotangente hiperbólico de X;
- ❖ `acsc(X)` retorna o arco cossecante de X;
- ❖ `acsch(X)` retorna o arco cossecante hiperbólico de X;
- ❖ `asec(X)` retorna o arco secante de X;
- ❖ `asech(X)` retorna o arco secante hiperbólico de X;
- ❖ `asin(X)` retorna o arco seno de X;
- ❖ `asinh(X)` retorna o arco seno hiperbólico de X;
- ❖ `atan(X)` retorna o arco tangente de X;
- ❖ `atanh(X)` retorna o arco tangente hiperbólico de X;
- ❖ `atan2(X)` retorna o arco tangente de quatro quadrantes de X;
- ❖ `cos(X)` retorna o cosseno de X;
- ❖ `cosh(X)` retorna o cosseno hiperbólico de X;
- ❖ `cot(X)` retorna a cotangente de X;
- ❖ `coth(X)` retorna a cotangente hiperbólico de X;
- ❖ `csc(X)` retorna a cossecante de X;
- ❖ `csch(X)` retorna a cossecante hiperbólico de X;
- ❖ `sec(X)` retorna a secante de X;
- ❖ `sech(X)` retorna a secante hiperbólica de X;
- ❖ `sin(X)` retorna o seno de X;
- ❖ `sinh(X)` retorna o seno hiperbólico de X;
- ❖ `tan(X)` retorna a tangente de X;
- ❖ `tanh(X)` retorna a tangente hiperbólica de X;

Aritméticas:

- ❖ `exp(X)` retorna a exponencial e X;
- ❖ `expm1(X)` mesmo que `exp(X)-1`;
- ❖ `log(X)` logaritmo natural de X;
- ❖ `log2(X)` logaritmo de X na base 2;
- ❖ `log10(X)` logaritmo de X na base 10
- ❖ `sqrt(X)` raiz quadrada de X;

Numéricas

- ❖ `ceil(X)` retorna o arredondamento de X em direção a infinito;
- ❖ `fix(X)` retorna o arredondamento X em direção a zero;
- ❖ `floor(X)` retorna o arredondamento X em direção a infinito negativo;
- ❖ `round(X)` retorna o arredondamento X para o inteiro mais próximo;
- ❖ `gcd(X, Y)` retorna o máximo divisor comum de X e Y;
- ❖ `lcm(X, Y)` retorna o mínimo múltiplo comum de X e Y;
- ❖ `rem(X, Y)` retorna o resto da divisão de X por Y, resto de X/Y;
- ❖ `sign(X)` retorna 0 caso X seja 0, retorna 1 caso X seja maior que 0 ou retorna -1 caso X seja menor que 0.

Gráficos:

- ❖ `plot(X, Y, TipoLinha, X2, Y2, TipoLinha2...)` plota em uma mesma imagem um gráfico de X versus Y com o tipo de linha especificado e mais quantos gráficos forem passados por parâmetro. Também pode ser

usada como `plot(Y)`, na qual será plotado os índices de Y versus seus valores;

- ❖ `fplot('funcao', [máximo, mínimo], TipoLinha...)` plota em uma mesma imagem um gráfico da função especificada pela string do valor máximo até o mínimo com o tipo de linha especificado e mais quantos gráficos forem passados por parâmetro;
- ❖ `title('Título')` Insere ou modifica o título do gráfico;
- ❖ `xlabel('x')` Insere ou modifica o escrito no eixo x do gráfico;
- ❖ `ylabel('y')` Insere ou modifica o escrito no eixo y do gráfico;
- ❖ `zlabel('z')` Insere ou modifica o escrito no eixo z do gráfico;
- ❖ `grid` Mostra linhas de grade no gráfico caso não tenha, ou as remove, caso tenha. Também pode ser usado como `grid on` ou `grid off`;
- ❖ `close(a)` fecha a janela(gráfico) de índice a;
- ❖ `close all` fecha todas as janelas(gráficos);

As funções e utilização dos gráficos estão melhores explicadas e descritas no tópico destinado aos gráficos

Polinômios:

- ❖ `roots(Polinômio)` retorna as raízes do polinômio;
- ❖ `poly(Raízes)` retorna os coeficientes do polinômio em que as raízes são os números do arranjo de parâmetro;
- ❖ `conv(Polinômio1, Polinômio2)` retorna os coeficientes da multiplicação dos polinômios;
- ❖ `deconv(Polinômio1, Polinômio2)` retorna dois vetores, o primeiro é o quociente da divisão entre os polinômios, e o segundo o resto, normalmente usa-se `[q, r] = deconv(y1, y2)`;
- ❖ `polyval(Polinômio, Valor(es))` retorna o resultado das avaliações do polinômio nos valores fornecidos;
- ❖ `polyfit(x, y, n)` retorna os coeficientes do polinômio ajustado de grau n nos valores de x e y;
- ❖ `[r, p, k] = residue(n, d)` sendo n e d, o numerador e denominador, respectivamente, a função retorna r=resíduo, p=polo, k=termo direto;
- ❖ `[n, d] = residue(r, p, k)` sendo r=resíduo, p=polo, k=termo direto, a função retorna n e d, o numerador e denominador, respectivamente;

As funções e utilização dos polinômios estão melhores explicadas e descritas no tópico destinado aos polinômios.

Exercícios:

- 1) Existe um procedimento chamado decomposição de Cholesky. Neste procedimento, a matriz, caso seja definida positiva, é decomposta em um produto entre uma matriz triangular inferior e sua adjunta, facilitando e otimizando os processos computacionais. O MATLAB possui uma função para calcular a decomposição de Cholesky. Utilizando os comandos `help` e `lookfor`, descubra qual é essa função e calcule a decomposição de Cholesky para $A=[9 \ 6 \ -3 \ 3; \ 6 \ 20 \ 2 \ 22; \ -3 \ 2 \ 6 \ 2; \ 3 \ 22 \ 2 \ 28]$;
- 2) Tomando por base algumas das funções vistas anteriormente:
 - a) Calcule a raiz quadrada de 187.9915

- b) Arredonde o resultado em direção a:
- O inteiro mais próximo
 - Zero
- c) Calcule a exponencial de 3
- d) Arredonde o resultado em direção a:
- O inteiro mais próximo
 - Infinito
- e) Calcule o logaritmo natural de 3
- f) Arredonde o resultado em direção a:
- O inteiro mais próximo
 - Infinito
- 3) Utilizando funções de criação e manipulação de arranjos faça:
- Um arranjo A 1x3 preenchido com 1, 2 ,3
 - Uma matriz B 3x3 em que cada linha é igual a A
 - Um arranjo P 1x2 preenchido com 2, 2
 - Uma matriz Q 2x2 em que cada linha seja igual a P
 - Um arranjo R 2x2x2 em que cada matriz 2x2 seja igual a Q
- 4) Utilizando os arranjos criados na questão anterior e as funções de manipulação de arranjos, calcule:
- As dimensões de R
 - O traço de B
 - O determinante de B
 - O traço de Q
 - O determinante de Q
- 5) Usando as funções de impressão e entrada:
- Imprima a mensagem Hello World
 - Peça que o usuário insira um valor e o atribua a variável x
 - Imprima a mensagem Olá, programador, x = <valor de x>
- 6) Crie:
- Um arranjo a1, com 5 elementos igualmente espaçados entre 0 e 2π
 - Um arranjo b1 = seno (a1)
 - Um arranjo a2, com 10 elementos igualmente espaçados entre 0 e 2π
 - Um arranjo b2 = seno (a2)
 - Um arranjo a3, com 100 elementos igualmente espaçados entre 0 e 2π
 - Um arranjo b3 = seno (a3)
- 7) Utilizando os arranjos anteriores, use a função plot para plotar, simultaneamente, três gráficos de $a_i \times b_i$, $i=1,2,3$.
- 8) Compare os desenhos dos gráficos anteriores e diga qual é o 1, 2 e 3. Justifique sua resposta.
- 9) Calcule o ângulo e o módulo do complexo $13.5 + 13.5i$, em seguida, converta o resultado do ângulo de radianos para graus.
- 10) Repita o procedimento anterior para o complexo $20i$.

Operadores

Eventualmente será necessário realizar operações, tanto aritméticas quanto lógicas. Para isso, como toda linguagem de programação, o MATLAB usa operadores.

4.1 Aritméticos

São os operadores utilizados para se fazer as contas matemáticas mais comuns. Os operadores podem funcionar de forma diferente dependendo dos itens operados. Para facilitar a explicação e o resultado, vamos supor:

a, b vetores

A, B matrizes

c, d escalares

z complexo

Z matriz de complexos

Operação	Expressão	Resultado
Adição Escalar	c+d	c+d
Multiplicação Escalar	c*d	c*d
Divisão Escalar	c/d	c/d
Divisão Escalar	c\ d	d/c
Adição Escalar	c.+d	c+d
Multiplicação Escalar	c.*d	c*d
Divisão Escalar	c./d	c/d
Divisão Escalar	c.\ d	d/c
Potenciação	c^d	c^d
Potenciação	c.^d	c^d

Operação	Expressão	Resultado
Adição Escalar	a+c	[a1+c, a2+c ... an+c]
Adição Vetorial	a+b	[a1+b1, a2+b2...an+bn]
Multiplicação Escalar	a*c	[a1*c, a2*c ... an*c]
Multiplicação Vetorial	a.*b	[a1*b1, a2*b2...an*bn]
Divisão Vetorial	a./b	[a1/b1, a2/b2...an/bn]
Divisão Vetorial	a.\b	[b1/a1, b2/a2...bn/an]
Potenciação	a.^c	[a1^c, a2^c ... an^c]
Potenciação	a.^b	[a1^b1, a2^b2 ... an^bn]

Operação	Expressão	Resultado
Adição Escalar	A+c	aij+c
Adição Matricial	A+B	aij+bij
Multiplicação Escalar	A*c	aij*c
Multiplicação Ponto a Ponto	A.*B	aij*bij
Multiplicação Matricial	A*B	AB
Divisão Ponto a Ponto	A./B	aij/bij
Potenciação	A.\B	bij/aij
Potenciação	A^c	A^c
Potenciação	A.^c	aij^c
Potenciação	A.^B	aij^bij
Potenciação	c.^A	c^aij
Potenciação	c^A	c^A

Operação	Expressão	Resultado
Conjugação Complexa	z'	conjugado de z
Transposição	a'	a transposto
Transposição	A'	A transposto
Transposição e Conjugação	Z'	Z transposto e zij conjugado

O operador “.” antes do operador aritmético indica que a operação é ponto a ponto. Por exemplo, uma multiplicação matricial padrão é expressa por A*B. No entanto, se for feito A.*B, significa que a multiplicação é ponto a ponto, ou seja, cada elemento aij será multiplicado pelo bij ($[a_{11}*b_{11}, a_{12}*b_{12} \dots a_{1n}*b_{1n}; \dots \dots a_{nn}*b_{nn}]$).

Observe que, diferente de outras linguagens, o MATLAB não possui um operador para se computar o resto de uma divisão. Para isso, deve se usar a função rem(X, Y), que retornará o resto de $X \div Y$.

4.2 Lógicos

Os operadores lógicos são usados para se avaliar mais de uma expressão, muito utilizados em estruturas condicionais e de repetição, que serão explicados a frente. Como ainda não vimos tais estruturas, vamos usar a própria linha de comando para cálculos lógicos. A variável ans nos dirá se o resultado é verdadeiro ou falso, sendo do tipo de dado booleano (logical).

Para o MATLAB, zero é falso e qualquer valor diferente de zero é verdadeiro, sendo um o valor padrão.

Utilizaremos a seguinte sintaxe:

<expressão> <operador> <expressão>

Por exemplo se pedirmos para o MATLAB avaliar “verdadeiro e verdadeiro”, pela conjunção “e”, a resposta final será verdadeiro, exemplo (operador “e” é expresso por &):

```
>>2 & 5
```

```
ans =
```

logical

```
1
```

Operadores:

- ❖ & conjunção e, se alguma das expressões for falsa, retornará falso;
- ❖ | disjunção ou, retorna verdadeiro se pelo menos uma expressão for verdadeira;
- ❖ ~ negação não, retorna verdadeiro se for falso ou falso se for verdadeiro;
- ❖ && e curto circuito;
- ❖ || ou curto circuito.

Os operadores de curto circuito tem as mesmas respostas dos padrões. Sua diferença é na avaliação dos valores. No caso do “e curto circuito”, caso a primeira expressão seja falsa, o MATLAB não avaliará a segunda, e retornará 0, enquanto que no “e padrão”, sempre ambas as expressões são avaliadas. Já no “ou curto circuito”, a segunda expressão não é avaliada caso a primeira seja verdadeira, e 1 já é retornado. Exemplo:

```
>> x = [2 2]; y = [1 2 3];
>> a = 1; x = [2 2]; y = [1 2 3];
>> a == 0 && x+y == 1      % e curto circuito
ans =
```

logical

```
0
```

```
>> a == 0 & x+y == 1      % e
Matrix dimensions must agree.
```

Note que x+y está incorreto porque os vetores não são do mesmo tamanho, no entanto, no caso curto circuito, o MATLAB não avaliou a expressão, já que só com a primeira expressão, neste caso, já é possível chegar a resposta.

Dica: o operador OU-EXCLUSIVO no MATLAB é representado pela função xor(A, B):

```
>>A = [1 0]; B = [1; 0];
```

```
>>xor(A, B)
```

```
ans =
```

2×2 logical array

```
0 1
```

```
1 0
```

```
>>xor(1, 0)
```

```
ans =
```

logical

```
1
```

4.3 Relacionais

São necessários para fazer avaliações comparativas, como: a variável X é menor do que 10? Para analisarmos, iremos fazer como no tópico anterior, usar a própria linha de comando para calcular, e irá retornar 0 caso seja falso e 1 caso seja verdadeiro. Esses operadores também são muito utilizados em estruturas condicionais e laços.

Operadores:

- ❖ > maior que;
- ❖ < menor que;
- ❖ >= maior ou igual que;
- ❖ <= menor ou igual que;
- ❖ == igual a;
- ❖ ~= diferente de (não igual);

Exemplo:

```
>>x=4;
>> x < 10
ans =
```

logical

1

4.4 Prioridades

Geralmente, o MATLAB realiza as operações da esquerda para direita. No entanto, alguns operadores tem prioridades sobre outros, e por isso, são executados primeiro. Por exemplo, $a + b*c$ será executado como $a + (b*c)$ – multiplicação primeiro, como na linguagem matemática. Use os parênteses para especificar a ordem desejada.

Lista de prioridades:

1. ()
2. . ' . ^ ' ^
3. . ^- . ^+ . ^~ ^- ^+
4. + (unário) - (unário) ~
5. . * . / . \ * / \
6. + -
7. :
8. < <= > >= == ~=
9. &
10. |
11. &&
12. ||

Exercícios

- 1) Utilizando o MATLAB como uma simples calculadora, compute:
 - a) 13.711^2
 - b) 13.711^3
 - c) $[4, 1, 2]$ transposto
- 2) Veja a diferença entre os operadores \ e / ao se calcular $10/5$ e $10\backslash 5$.
- 3) É possível inserir algum valor de x para que a expressão lógica $10 > x > 2$ retorne 1 (verdadeiro)? Justifique.
- 4) Para quais valores de x irá retornar 1 a expressão $(2 < 10 == 1) + 10 > x$ irá retornar 1 (verdadeiro)? Justifique.
- 5) Determine pelo menos um valor para cada variável a , b , c e d para que a expressão seguinte retorne 1 (verdadeiro): $((a > 10) == 0) + b > c == \sim d$
- 6) Determine pelo menos um valor para cada variável a , b , c e d para que a expressão seguinte retorne 0 (falso): $((a > 10) == 0) + b > c == \sim d$
- 7) Faça:
 - a) $A = \begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{matrix}$
 - b) $B = \begin{matrix} 1 & 6 & 1 \\ 2 & 9 & 0 \\ 1 & 1 & 1 \end{matrix}$
 - c) $C = \begin{matrix} 2 & 4 & 5 \end{matrix}$
 - d) $D = 7$
- 8) Utilizando os arranjos anteriores, calcule:
 - a) AB
 - b) $A_{ij} \times B_{ij}$, para todo i e j
 - c) DA
 - d) $C_{ij}D$
 - e) D^C
- 9) Repita os cálculos de 8)a) e 8)b) e compare os resultados obtidos pelo uso do `.`.
- 10) Crie um vetor A com seis elementos valendo “`rand > 0.5`”. Sabendo que a função `rand` retorna números de distribuição uniforme entre 0 e 1, quantos 0s e quantos 1s você acha que A vai possuir? Repita o procedimento algumas vezes e observe.

Controle de Fluxo

Na programação, por vezes, são necessárias estruturas mais complexas, que permitam realizar repetições controladas e comandos condicionais. Em todas do MATLAB é necessário o end para indicar o fim. Para isso, utilizaremos as seguintes estruturas:

5.1 Estruturas Condicionais

Bem como em muitas outras linguagens de programação, no MATLAB, há a estrutura condicional if-else. O else é usado no caso contrário, ou seja, se a condição do if for falsa, será executado o que está no else. A sintaxe é a seguinte:

```
if <condição>
    <comandos>
else
    <comandos>
end
if x>10
    disp('x maior que 10')
else
    disp('x não é maior que 10')
end
```

É possível também criar aninhamentos if e else para várias avaliações. Exemplo:

```
if x==1
    disp('x vale 1')
elseif x==2
    disp('x vale 2')
elseif x==3
    disp('x vale 3')
else
    disp('x não vale 1, 2 ou 3')
end
```

Há também outra estrutura condicional, o switch. Uma variável é avaliada e separadas em casos, vejamos a sintaxe e o exemplo:

```
switch <variável>
    case <valor1>
        <comandos1>
    case <valor2>
        <comandos2>
    otherwise
        <comandos3>
end
X=input('insira o valor de X')
switch X
    case 1
        disp('X vale 1')
    case 2
        disp('X vale 2')
    case 7
        disp('X vale 7')
    otherwise
        disp('X vale qualquer outra coisa')
end
```

Usar switch ou aninhamentos if-else é, basicamente, a mesma coisa. A diferença será na praticidade e compreensão do código, dependendo das operações e procedimentos.

5.2 Estruturas de Repetição

As estruturas de repetição também são conhecidas como laços ou loops. O primeiro deles é o while. Funciona da seguinte forma

```
while <condição>
    <comandos>
end
```

A cada iteração, a condição é avaliada, se for verdadeira, os comandos são realizados, em seguida a condição é avaliada novamente, se verdadeira, os comandos realizados novamente, até que a condição seja falsa. Exemplo:

Vamos criar um vetor A = [64, 32, 16, 8, 4, 2, 1]:

```
>>valor=128, contador=0;
>>while valor>1
    valor = valor/2
    contador = contador + 1
    A(contador) = valor;
end
>>A
A=
64 32 16 8 4 2 1
```

Também há a estrutura for, mais utilizada quando se há uma variável de controle contadora. A sintaxe é a seguinte:

```
for <variável> = <arranjo>
    <comandos>
end
```

Assim, a variável de controle assumirá, em cada iteração, um valor do arranjo, em sequência, do primeiro ao último. Quando chegar ao último, o laço é encerrado. Exemplo, vamos criar um arranjo de 10 elementos, no qual cada posição vale o quadrado do seu índice: A=[1, 4, 9, 16, 25, 36, 49, 64, 91, 100]

```
>>for i = 1:10
    A(i)=i^2;
end
>>A =
1          4          9         16         25         36         49         64         81
100
```

Bem como switch e if-else, while e for são muito parecidos e tem o mesmo efeito prático. É necessário atenção para traduzir a sintaxe de uma estrutura para a outra e cuidado para não criar, accidentalmente, loops infinitos (condição sempre verdadeira) que irão atrapalhar na execução do seu algoritmo ou programa.

5.3 Desvio Incondicional

Os comandos de desvio incondicional são os comandos break e continue, com as mesmas funcionalidades de outras linguagens de programação.

5.3.1 Break

O “break” é geralmente é utilizado dentro de uma estrutura condicional, que por sua vez, está dentro de uma estrutura de repetição. Serve para forçar a saída do loop onde está. Atente-se ao exemplo para melhor compreensão:

Vamos fazer um laço while infinito, ou seja, a condição será sempre verdadeira. Dentro desse laço vamos somar sempre um a uma variável qualquer, e iremos verificar se ela é maior que um valor, se for, utilizaremos o comando de parada imediata:

```
x=5;  
while 1  
    x=x+1;  
    if x>40  
        break  
    end  
end
```

5.3.1 Continue

O “continue” também é geralmente usado dentro de uma estrutura condicional, dentro de uma estrutura de repetição. Serve para forçar o laço a ir para a próxima iteração, ou sair do laço, caso seja a última iteração, ignorando os comandos ainda não executados na iteração presente. Exemplo:

Vamos realizar a impressão dos valores que a variável i assume ao passar pelo laço for, variando de 1 a 5. No entanto, caso i seja 4, não queremos que seja impresso:

```
>> for i = 1:5  
    if i == 4  
        continue  
    else  
        fprintf('%d ', i)  
    end  
end  
1 2 3 5
```

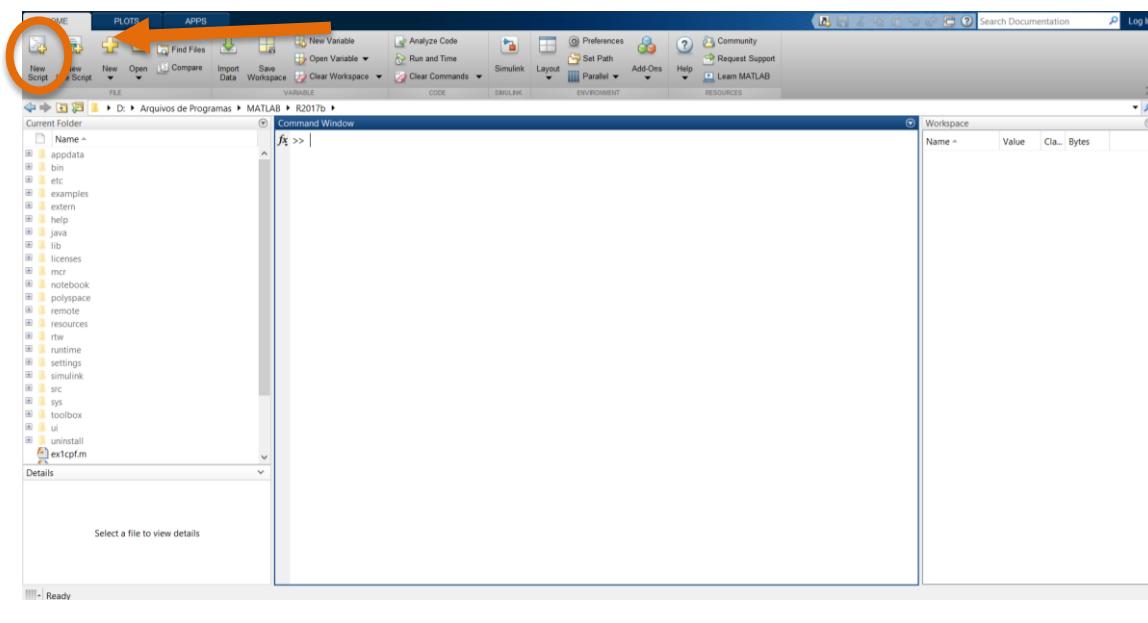
Exercícios

- 1) Crie a e b, dois escalares com valores a sua escolha. Receba do programador, um número para uma variável de controle e diga que, caso o número seja igual a 1, a e b serão somados. Faça uma estrutura condicional usando if-else que caso o número seja igual a 1, a e b serão somados.
- 2) Adicione a sequência de comandos anteriores, a opção de número 2, que indicará a subtração a – b.
- 3) Adicione a sequência de comandos anteriores, a opção de número 3, que indicará a multiplicação a x b.
- 4) Adicione a sequência de comandos anteriores, a opção de número 4, que indicará a divisão a ÷ b.
- 5) Refaça a sequência de comandos utilizando a estrutura condicional switch.
- 6) Reescreva a sequência de comandos a seguir utilizando a sintaxe if-else:
Nota=input('Digite a nota do aluno(0 a 5)')
Nota=round(nota)
switch Nota
case 0
display('Procure o professor para ajuda')
case 1
display('Precisa se dedicar mais')
case 2
display('Ainda um pouco mais')
case 3
display('Bom, 60 é 100')
case 4
display('Parabéns')
case 5('Sensacional!!!')
end
- 7) Utilizando uma estrutura de repetição, crie um vetor G de tamanho 1x10, com seus valores iguais a $10^2, 9^2, 8^2 \dots 1^2$.
- 8) Caso tenha usado for, repita a questão anterior usando while. Caso tenha usado while, repita usando for.
- 9) Utilizando a estrutura de repetição for, crie um vetor H de tamanho 1x5, com valores iguais a $3^3, 3^3, 1^3, 2^3, 3^3$.
- 10) Faça o mesmo vetor H da questão anterior, agora utilizando while.

Scripts e Arquivos .m

Para tarefas mais longas e complexas, ao invés de digitar comando por comando na Command Window ou usar shift+enter várias vezes, é possível criar linhas de código para serem executadas em sequência posteriormente. São os chamados scripts, ou seja, roteiros. Você pode criá-los usando algum editor de texto como o próprio bloco de notas do Windows ou Notepad++ e salvá-los com a extensão .m.

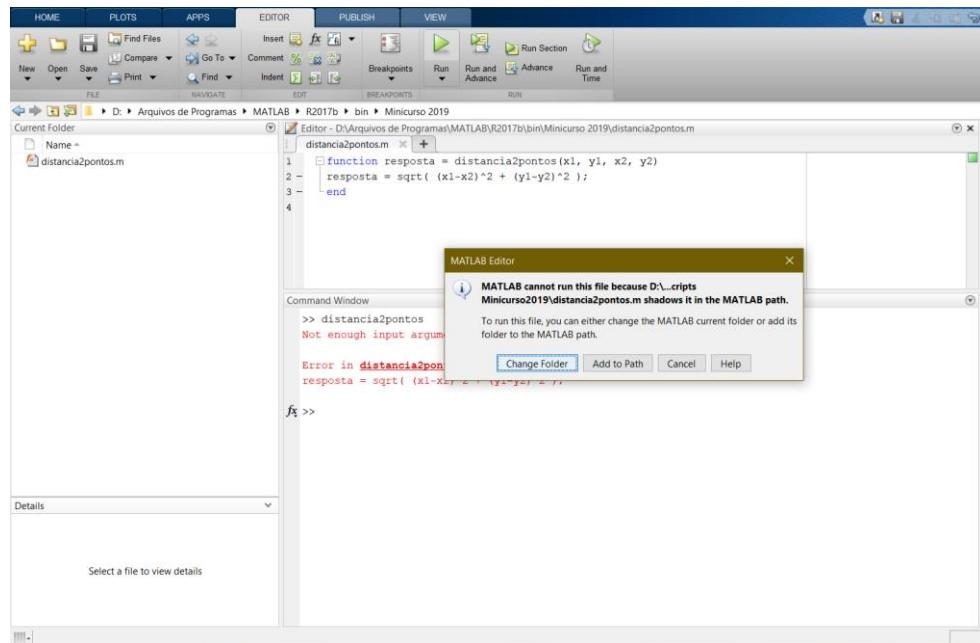
Porém, é mais prático usar as próprias ferramentas do MATLAB para se criar os scripts. Basta clicar em New Script no canto superior esquerdo ou usar o atalho Ctrl+N:



New Script

Uma nova janela será aberta juntamente com uma nova barra de tarefas, o editor do MATLAB.

Após escrever seu código clique em salvar. Note que, para poder ser executado, ele precisa estar em um certo diretório e aberto na janela Current Folder. Pode navegar até o diretório onde está o script ou usar a opção Change Folder e/ou Add to Path, que aparecerá quando tentar executar o script. Depois de adicionado, o script rodará normalmente ao clicar na opção run ou ao digitar o nome do arquivo na Command Window.



Executando script

6.1 Declaração de Funções

É possível usar os scripts para se declarar funções, neste caso, o nome do arquivo .m deve ser exatamente o mesmo nome da função. A sintaxe é a seguinte:

```
function <variáveis retornadas> = <nome da função avaliada nos parâmetros>
```

```
<procedimentos>
```

```
end
```

Por exemplo; vamos criar uma função que recebe quatro parâmetros, coordenadas x e y de dois pontos, e retorna a distância entre esses pontos.

`distancia2pontos.m`

```
function resposta = distancia2pontos(x1, y1, x2, y2)
resposta = sqrt( (x1-x2)^2 + (y1-y2)^2 );
end
```

Command Window:

```
>> distancia2pontos(0,0,3,4)
```

```
ans =
```

5

The screenshot shows the MATLAB IDE interface. In the Editor window, a file named 'distancia2pontos.m' is open, containing the following code:

```

function resposta = distancia2pontos(x1, y1, x2, y2)
    resposta = sqrt( (x1-x2)^2 + (y1-y2)^2 );
end

```

In the Command Window, the following command is entered and executed:

```

>> distancia2pontos(0,0,3,4)
ans =
5

```

Exemplo distancia2pontos

Para o caso de retornar um vetor ou mais de uma variável, deve se proceder da seguinte forma:

Vamos supor que, além de calcular a distância, nossa função `distancia2pontos` também retorne a média de `x1` e `x2`.

`distancia2pontos.m`

```

function [resposta, mediax] = distancia2pontos(x1, y1, x2,
y2)

resposta = sqrt( (x1-x2)^2 + (y1-y2)^2 );
mediax= (x1+x2)/2;

end

```

Command Window:

```

>> distancia2pontos(0,0,3,4)
ans =
5
>> [x, y] = distancia2pontos(0,0,3,4)
x =
5
y =
1.5000

```

As funções padrão possuem descrições que podem ser usadas para auxílio quando executados os comandos `help` e `lookfor`. Essas descrições podem ser também adicionadas às funções criadas pelos usuários. Basta adicionar comentários em sua descrição. Exemplo:

The screenshot shows the MATLAB interface. The Editor window displays the code for 'distancia2pontos.m'. The Command Window below it shows the results of running 'lookfor distancia' and 'help distancia2pontos'.

```

Editor - D:\Arquivos de Programas\MATLAB\R2017b\distancia2pontos.m
1 function resposta = distancia2pontos(x1, y1, x2, y2)
2 %distancia2pontos calcula a distância entre dois pontos
3 %Exemplo de descrição da função
4 %conteúdo e etc...
5 -    resposta = sqrt( (x1-x2)^2 + (y1-y2)^2 );
6 -
7

Command Window
>> lookfor distância
distancia2pontos           - calcula a distância entre dois pontos
>> help distancia2pontos
distancia2pontos calcula a distância entre dois pontos
Exemplo de descrição da função
conteúdo e etc...

fx >>

function resposta = distancia2pontos(x1, y1, x2, y2)
%distancia2pontos calcula a distância entre dois pontos
%Exemplo de descrição da função
%conteúdo e etc...
resposta = sqrt( (x1-x2)^2 + (y1-y2)^2 );
end

>> lookfor distância
distancia2pontos           - calcula a distância entre
dois pontos

>> help distancia2pontos
distancia2pontos calcula a distância entre dois pontos
Exemplo de descrição da função
conteúdo e etc...

```

Dica: o comando “return” é usado para forçar o retorno, isto é, a finalização da função. Geralmente utilizado para terminar a função antecipadamente, caso tenha alguma condição excepcional.

Exercícios

- 1) Refaça o exercício 6) do capítulo Controle de Fluxo, em um script. Inicie o script limpando as variáveis e a Command Window.
- 2) Refaça o exercício 7) do capítulo Controle de Fluxo, em um script. Inicie o script limpando as variáveis e a Command Window.
- 3) Refaça o exercício 9) do capítulo Controle de Fluxo, em um script. Inicie o script limpando as variáveis e a Command Window.
- 4) Crie um arquivo .m chamado ex1log.m que define a função ex1log. A função receberá dois escalares x e y e retornará o log_yx. Lembre-se que: log_ba = log_a/log_b.
- 5) Modifique a função anterior para que x e y possam ser arranjos de mesmo tamanho, fazendo com que a função retorne log_yx.

- 6) Crie um arquivo .m chamado ex1ord que define a função ex1ord. A função deverá receber um arranjo linear e retornar outro arranjo, com os mesmos valores do primeiro ordenados em ordem crescente. Use o método de ordenação que lhe deixar mais confortável.
- 7) Os dois últimos dígitos de um cpf são calculados da seguinte forma:
O número é composto por nove dígitos e mais dois verificadores, logo ABCDEFGHI-JK. Para se calcular J é feito:
$$(10A + 9B + 8C + 7D + 6E + 5F + 4G + 3H + 2I) \div 11$$

Analisamos o resto. Se o resto for 0 ou 1 J será 0. Se não, $J = 11 - \text{resto}$.
Para K, fazemos:
$$(11A + 10B + 9C + 8D + 7E + 6F + 5G + 4H + 3I + 2J) \div 11$$

Analisamos o resto. Se o resto for 0 ou 1 K será 0. Se não, $K = 11 - \text{resto}$.
Crie um arquivo .m chamado de ex1cpf que defina a função ex1cpf. A função deverá receber um número de cpf completo em forma de escalar e conferir se os dois últimos dígitos estão corretos, retornando 1 caso estejam e 0 caso estejam incorretos.
- 8) Faça um script no MATLAB que pede do usuário um número de cpf. Logo após, execute o mesmo procedimento da questão anterior e imprima a mensagem “CPF correto”, caso esteja correto ou “CPF incorreto”, caso esteja incorreto.
- 9) Crie um script no MATLAB para calcular a média de notas de uma determinada quantidade de alunos. O procedimento deverá ser de seguinte forma. O programa deve pedir ao usuário um número N referente a quantidade de alunos. Em seguida, deverá coletar as N notas, também do usuário. No final, deverá imprimir o valor da nota média entre os alunos.
- 10) Crie um script no MATLAB que peça ao usuário dois vetores lineares. Em seguida, peça que digite 1 ou 2. Caso o usuário digite 1, imprima o escalar resultante do produto interno entre os vetores. Caso o usuário digite 2, imprima a matriz quadrada resultante do produto externo desses vetores.
Note que, a diferença entre o cálculo no caso 1 ou no caso 2, pode ser uma simples transposição. Como os dois vetores são da forma $A = [a_1 \ a_2 \ \dots \ a_n]$, isto é, lineares, para retornar um escalar, basta transpor o segundo vetor e realizar uma multiplicação matricial padrão. Enquanto que para retornar uma matriz, basta transpor o primeiro, ao invés do segundo, e realizar uma multiplicação matricial padrão.

Gráficos

O MATLAB também é uma importante ferramenta de construção de gráficos em duas ou três dimensões, com várias opções e utilidades.

7.1 Bidimensionais

A função básica para plotar um gráfico bidimensional é a função `plot`:

`plot(X, Y, TipoLinha, X2, Y2, TipoLinha2...)` plota em uma mesma imagem um gráfico de X versus Y com o tipo de linha especificado e mais quantos gráficos forem passados por parâmetro. Também pode ser usada como `plot(Y)`, na qual será plotado os índices de Y versus seus valores;

Exemplo:

X será um vetor igualmente espaçado de zero até 2π , com 100 valores, e Y o seno de X.

```
>> X=linspace(0, 2*pi, 100); Y=sin(X);
>> plot(X,Y)
```

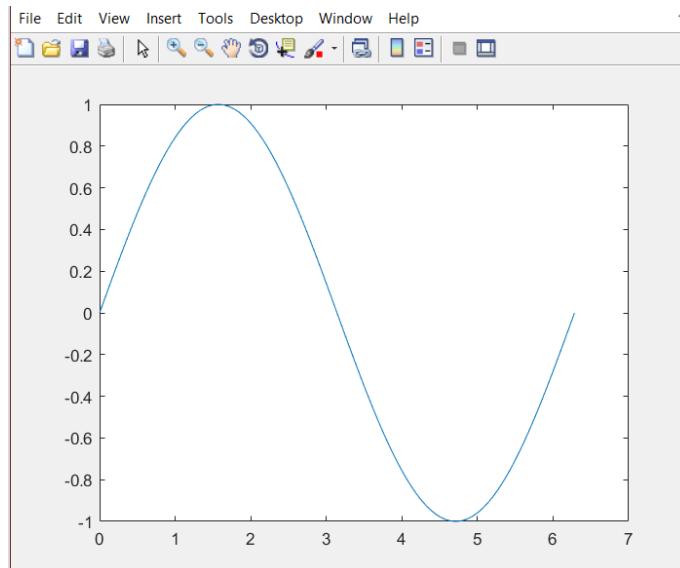


Gráfico plotado

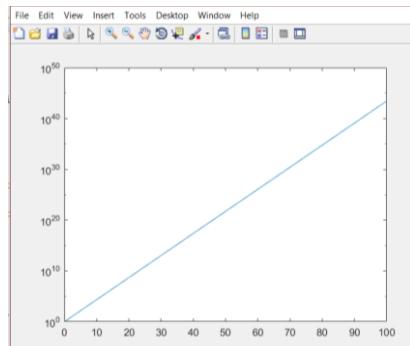
Existem também variantes da função `plot` para dar ao gráfico uma cara diferente, veja alguns exemplos, que recebem os mesmos parâmetros da função `plot`:

Escala logarítmica:

`loglog` plota em escala logarítmica em x e em y, enquanto que `semilogx` ou `semilogy` plota em escala logarítmica em x ou y, respectivamente.

```
>> X=linspace(0, 100, 100); Y=exp(X);
```

```
>> semilogy(X, Y)
```

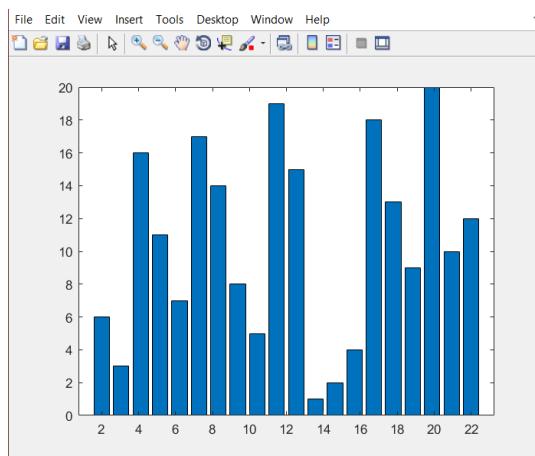


Estilos especiais:

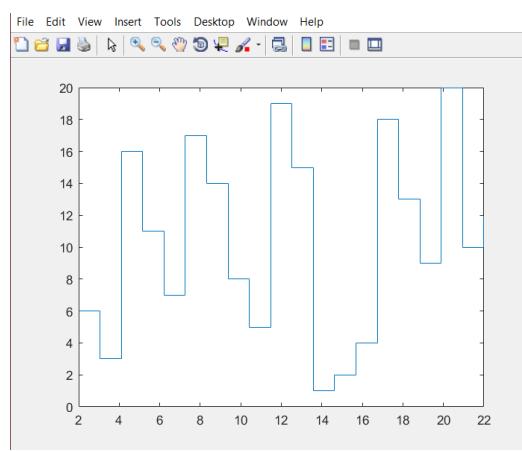
`bar` plota em gráfico de barras; `stairs` em degraus; `errorbar` em barras de erro; `hist` em histograma, a sequência precisa ser monótona e não decrescente; `rose` em histograma em ângulo; `compass` em forma de bússola; `feather` em forma de pena; `comet` mostra a trajetória do gráfico; `stem` em sequência discreta.

```
>>X=linspace(2, 22, 20); Y=randperm(20); F=X.^2; %F servirá para o histograma
```

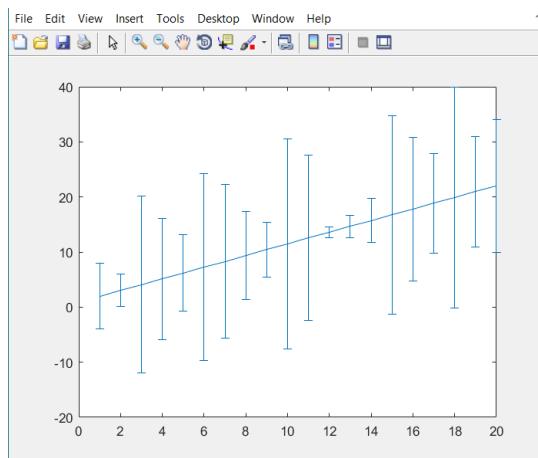
```
>> bar(X, Y)
```



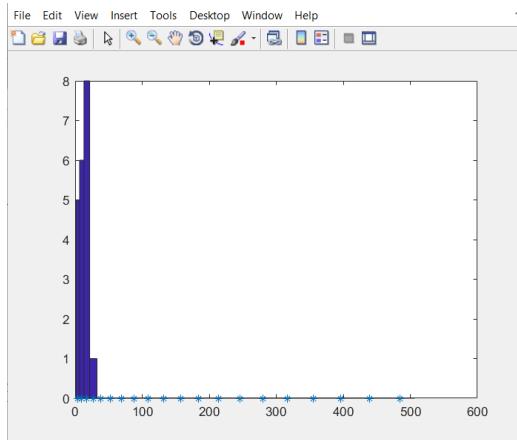
```
>>stairs(X, Y)
```



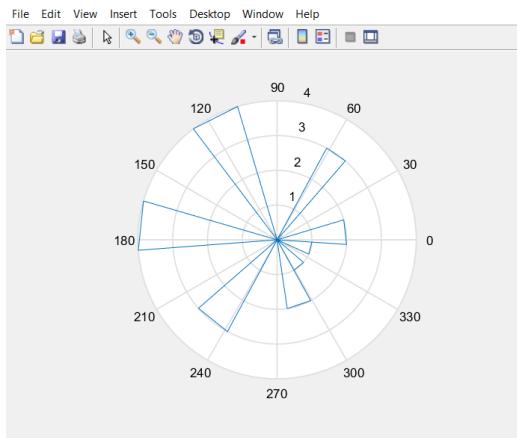
```
>>errorbar(X, Y)
```



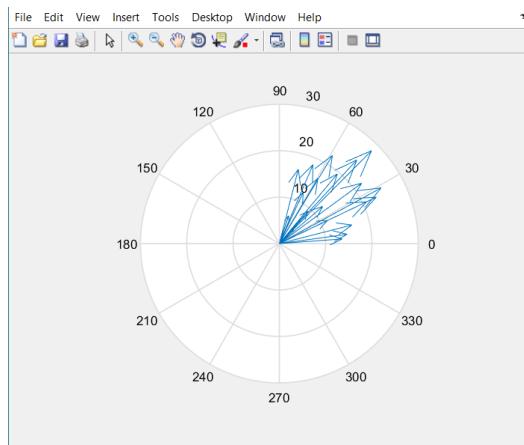
```
>>hist(X, F) %A sequência precisa ser monótona e não decrescente
```



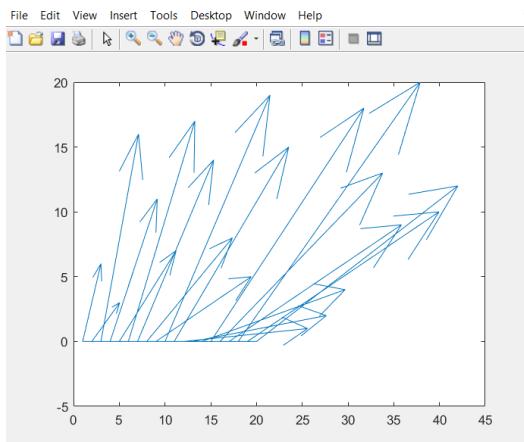
```
>>rose(X, Y)
```



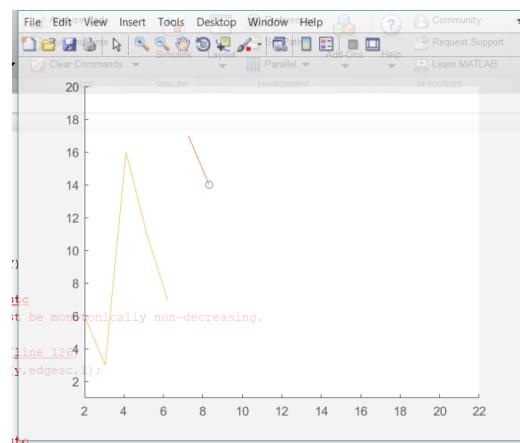
```
>>compass(X, Y)
```



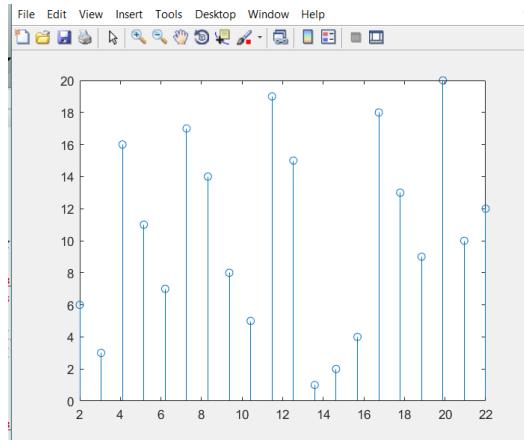
```
>>feather(X, Y)
```



```
>>comet(X, Y) % a trajetória foi percorrida rapidamente
```



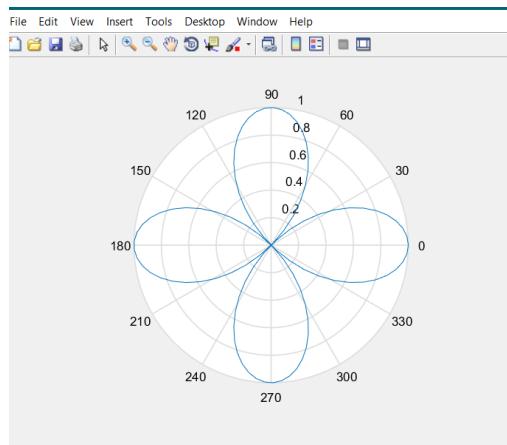
```
>>stem(X, Y)
```



Coordenadas polares:

Para plotar gráficos em coordenadas polares usa-se a função polar ou polarplot, que fazem o mesmo procedimento.

```
>> theta = linspace(0, 2*pi, 100); R=cos(2*theta);
>> polar(theta, R)
```

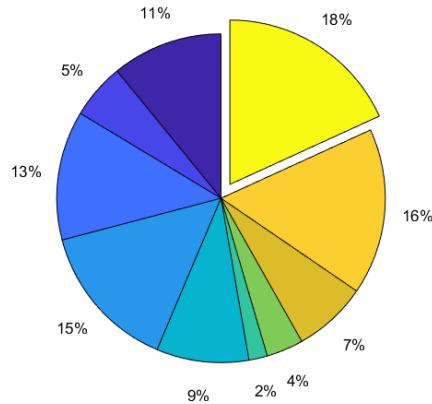


Gráficos em pizza:

Para gráficos em pizza, usa-se a função pie(Y, explode), a qual plota Y em gráfico de pizza. O vetor explode é para se destacar algumas fatias. É um arranjo do mesmo tamanho de Y, preenchido com 0 caso não se deseje destacar e 1 caso deseje. Exemplo:

```
>> x = randperm(10)
x =
       6         3         7         8         5         1         2         4         9
10
>> explode = (x == 10)
explode =
1×10 logical array
     0     0     0     0     0     0     0     0     0     1
```

```
>> pie(x, explode)
```



Como vimos nos argumentos de plot, bem como nos das outras funções, há strings utilizadas para definir o estilo do gráfico. As strings são formadas por caracteres que irão definir a cor, o tipo de linha(ou ponto) ou ambos.

Tipos de gráficos:

Caractere	Cor	Caractere	Ponto	Caractere	Linha
y	amarelo	-	(Contínua)
m	lilás	o	oooooo	--	-----
c	ciano	*	*****	-.	-.-.-.-.-.-.
r	vermelho	+	++++++	:
g	verde	x	xxxxxx		
b	azul	s	(Quadrados)		
w	branco	v	(Triângulos)		
k	preto	^	(Triângulos)		
		P	(Estrelas)		

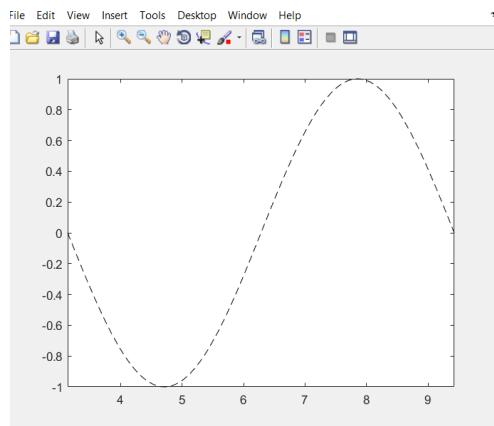
Há a função fplot para plotar gráficos de funções prontas.

fplot('funcao', [máximo, mínimo], TipoLinha...) plota em uma mesma imagem um gráfico da função especificada pela string do valor máximo até o mínimo com o tipo de linha especificado e mais quantos gráficos forem passados por parâmetro;

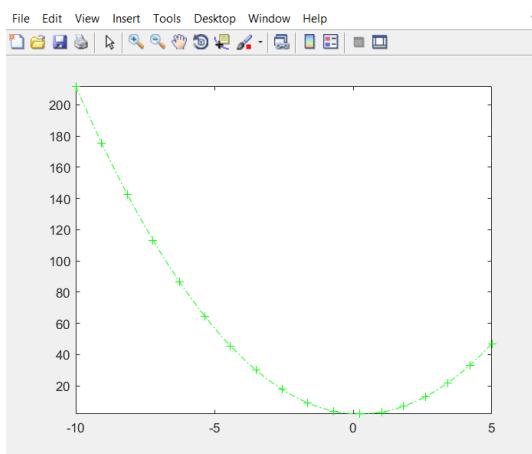
Seu primeiro argumento também pode ser uma expressão de uma variável simbólica.

Exemplos:

```
>>fplot('sin', [pi, 3*pi], 'k--')
```



```
>>syms x
>>f=2*x^2 - x + 2;
>>fplot(f, [-10, 5], 'g-.+')
```

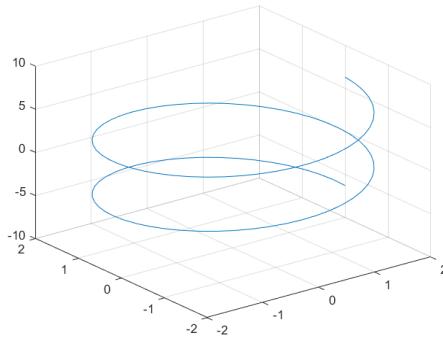


7.2 Tridimensionais

Podemos plotar gráficos tridimensionais muito similarmente aos bidimensionais, usando a função `plot3`, tendo x, y, e z como três vetores. Exemplo:

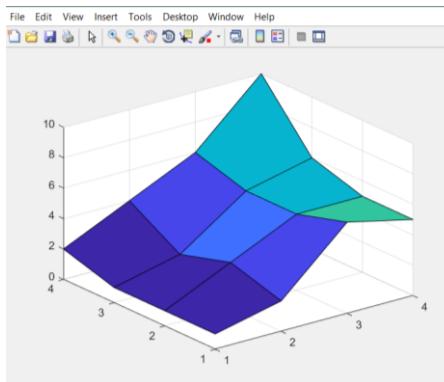
Hélice cilíndrica de raio 2:

```
>>t = linspace(-2*pi, 2*pi);
>>x = 2*cos(t); y = 2*sin(t); z = t;
>>plot3(x, y, z);
>>grid on
```



Para plotar os demais gráficos tridimensionais, as superfícies, precisamos de matrizes. A função padrão para se plotar gráficos tridimensionais é a função `surf(X, Y, Z)`. É análoga a função `plot` para gráficos bidimensionais. No exemplo a seguir, primeiro criamos os espaços das matrizes X e Y, em seguida, foram definidos os valores da matriz Z:

```
>>a=[1, 2, 3, 4];
>>[X, Y]= meshgrid(a);
>>Z=[ 1, 2, 6, 5; 1, 3, 5, 5; 1, 2, 5, 6; 2, 4, 6, 10; ];
>>surf(X, Y, Z) %ou simplesmente surf(Z)
```



X e Y são desnecessários nesse caso, porque são espaços variando de acordo com os índices dos valores de Z. No entanto, para alguns gráficos, são fundamentais.

Para a definição de X e Y como matrizes ideais para a plotagem, foi usada a função `meshgrid`, que retornou as duas matrizes. Sua sintaxe é daquela forma, primeiro criamos o vetor que definirá o tamanho e os intervalos e utilizamos como parâmetro da `meshgrid`.

```
>>a = [1, 2, 3, 4];
>>[X, Y] = meshgrid(a);
```

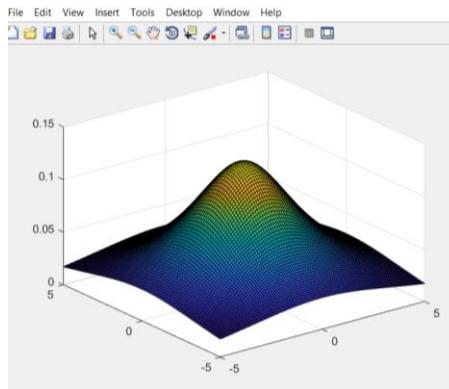
Existem também outras funções para se plotar gráficos tridimensionais em estilos diferentes.

A função `contour` plota o gráfico bidimensional de X, Y em curvas de nível; `contour3` plota as curvas de nível em três dimensões; `mesh` plota o gráfico em malha; `meshc` junção de `mesh` e `contour`; `surfc` junção de `surf` e `contour`; `surf`

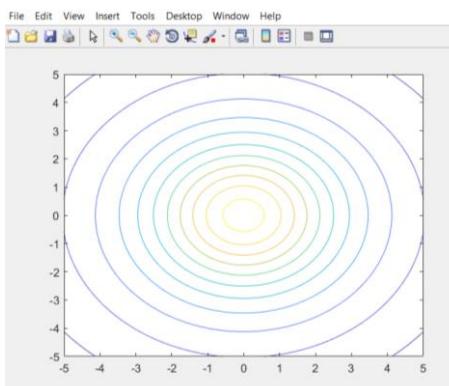
surf com iluminação; plot3 plota o gráfico em três dimensões em um conjunto de linhas; comet3 a trajetória.

Exemplos:

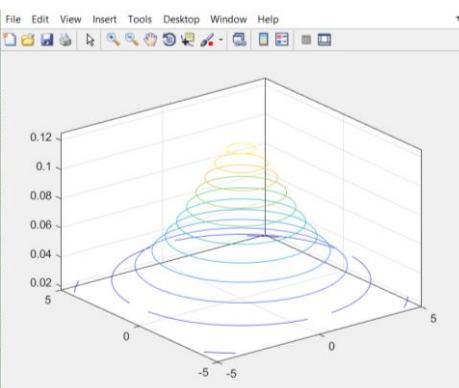
```
>> a = linspace(-5,5,100); [X, Y] = meshgrid(a);  
>> Z = 1./((X.^2-1)+(Y.^2-1)+10);  
>> surf(X, Y, Z)
```



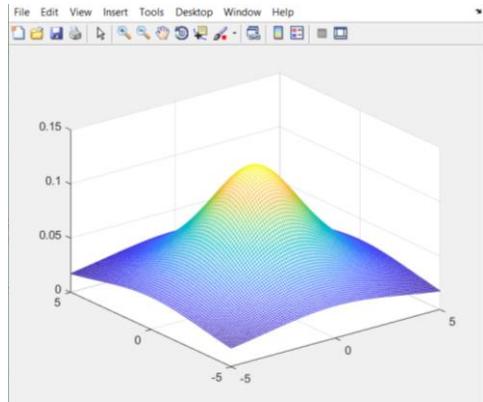
```
>>contour(X, Y, Z)
```



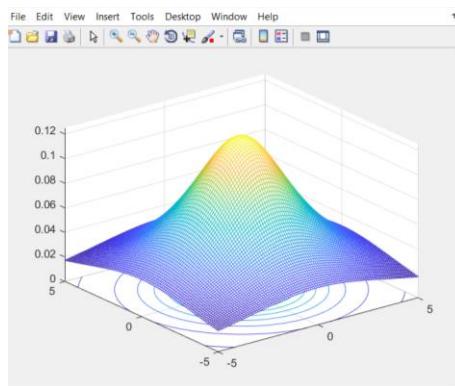
```
>>contour3(X, Y, Z)
```



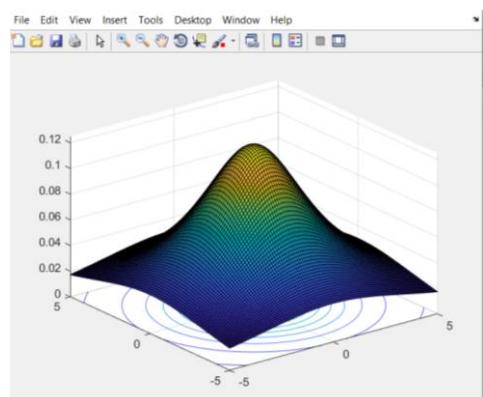
```
>>mesh(X, Y, Z)
```



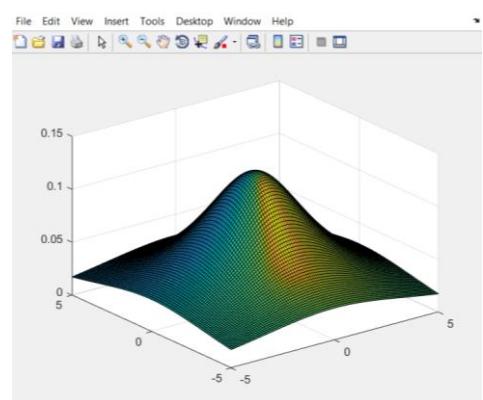
```
>>meshc(X, Y, Z)
```



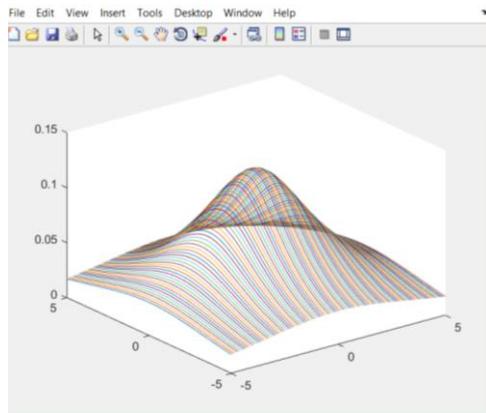
```
>>surfc(X, Y, Z)
```



```
>>surf1(X, Y, Z)
```

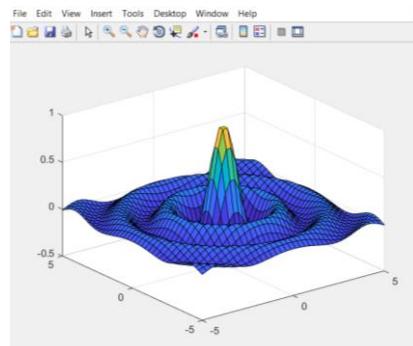


```
>>plot3(X, Y, Z)
```

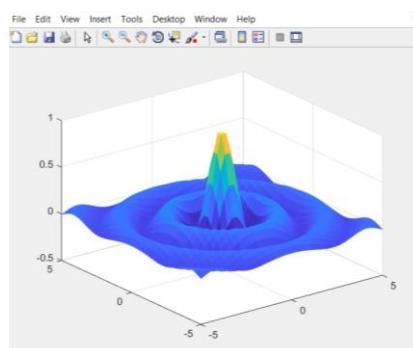


A aparência e cor do gráfico pode ser alterada com as funções `shading` e `colormap`. `colormap` recebe como parâmetro uma string, ou um vetor de 3 elementos. Caso a string seja “`default`”, será da cor padrão. Caso seja um vetor, a cor será em `rgb`. Sendo cada valor corresponde a uma cor, variando de 0 a 1.

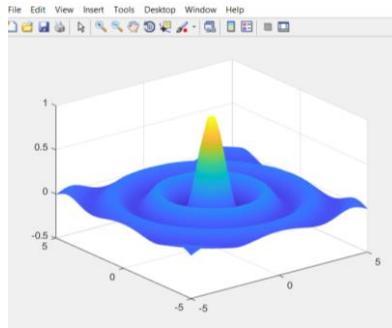
```
>>a = linspace(-5, 5, 40); [X, Y]=meshgrid(a);
>>Z = sinc(sqrt(X.^2+Y.^2));
>>surf(X, Y, Z) %shading faceted
```



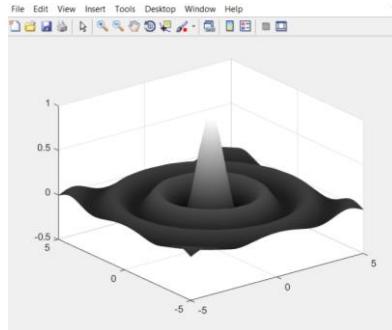
```
>>shading flat
```



```
>>shading interp
```



```
>>colormap (gray)
```



7.3 Comandos Auxiliares

Alguns comandos modificam as figuras e o gráfico, auxiliando na interpretação e montagem de acordo com a intenção do usuário. O comando `figure` permite abrir outra figura, enquanto que `figure(N)` volta na figura N, assim, criamos vários gráficos em figuras separadas:

Dica: Para poder plotar gráficos em diferentes figuras, simultaneamente, use o comando `figure`. Exemplo:

```
figure  
fplot('sin')  
  
figure  
fplot('cos')
```

Para visualizar ou editar a figura N: `figure(N)`.

Para poder plotar diferentes gráficos em uma mesma figura pode-se proceder de duas formas. A primeira é plotar também em um mesmo gráfico, para isso, utiliza-se o comando `hold on`. Esse comando, congela o gráfico e permite que plote outros em cima dele.

Dica: O comando “hold on” permite que a imagem permaneça. Assim, se usar comandos de plotagem separadamente, irão ser sob a mesma figura

`plot(...)`

`hold on`

`plot(...)`

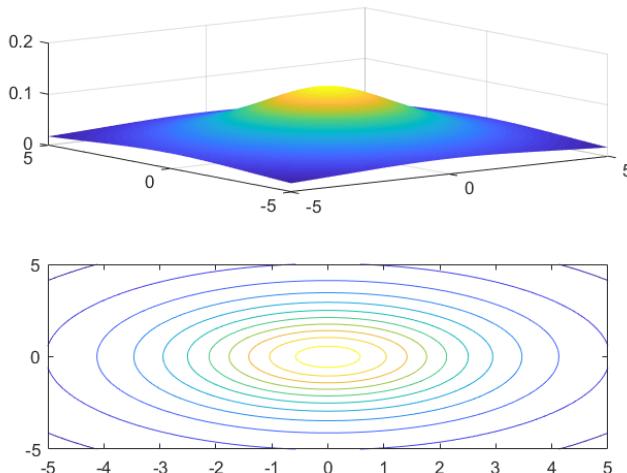
`plot(...)`

É útil para plotar parte a parte de gráficos tridimensionais.

A segunda maneira é subdividir a figura em vários gráficos, para isso usa-se o comando `subplot(m,n, X)`. O comando divide a figura em uma matriz $m \times n$, onde cada posição será um gráfico, e já seleciona a posição X.

Exemplo:

```
>> a = linspace(-5,5,100); [X, Y] = meshgrid(a); Z =
1./((X.^2-1)+(Y.^2-1)+10);
>> subplot(2, 1, 1)
>> surf(X, Y, Z)
>> shading interp
>> subplot(2, 1, 2)
>> contour(X, Y, Z)
```



O comando `grid` Mostra linhas de grade no gráfico caso não tenha, ou as remove, caso tenha. Também pode ser usado como `grid on` ou `grid off`;

O comando `axis` edita os eixos na escala e aparência.

`axis([XMIN XMAX YMIN YMAX])` define os limites do eixo x e y;

`axis([XMIN XMAX YMIN YMAX ZMIN ZMAX])` define os limites do eixo x, y e z (para gráficos tridimensionais);

`axis([XMIN XMAX YMIN YMAX ZMIN ZMAX CMIN CMAX])` define os limites do eixo x, y e z e os limites da escala de cores;

`axis on / axis off` liga ou desliga os eixos da figura;

`axis square` iguala os tamanhos (não os intervalos) dos eixos;
`axis equal` iguala os intervalos dos eixos;
`axis auto` retorna os eixos para a configuração default;
`axis normal` retira as edições feitas.

7.4 Textos, Títulos e Rótulos

Existem também funções para adicionar textos, títulos e rótulos aos gráficos.

`title('Título')` Insere ou modifica o título do gráfico;
`xlabel('x')` Insere ou modifica o escrito no eixo x do gráfico;
`ylabel('y')` Insere ou modifica o escrito no eixo y do gráfico;
`zlabel('z')` Insere ou modifica o escrito no eixo z do gráfico;
Há também o `gtext('texto')`, comando que adiciona texto na posição indicada pelo mouse.

O MATLAB suporta um subconjunto de marcadores TeX. Portanto, é possível adicionar aos gráficos caracteres especiais e letras gregas, bem como sobrescritos, subscritos e modificador o tipo de texto utilizando a marcação TeX.

Modificadores:

Modificador	Ação	Exemplo
<code>^{}</code>	Sobrescrito	'texto^{sobrescrito}'
<code>_{}{}</code>	Subscrito	'texto_{subscrito}'
<code>\bf</code>	Negrito	'\bf texto'
<code>\it</code>	Itálico	'\it texto'
<code>\sl</code>	Oblíqua(geralmente itálico)	'\sl texto'
<code>\rm</code>	Fonte padrão	'\rm texto'
<code>\fontname{fonte}</code>	Alterar a fonte	'\fontname{Calibri} texto'
<code>\fontsize{tamanho}</code>	Alterar o tamanho	'\fontsize{11} texto'
<code>\color{nome}</code>	Alterar a cor	'\color{red} texto'
<code>\color[rgb]{valor}</code>	Alterar a cor	'\color[rgb]{0.2, 0, 0.5} texto'

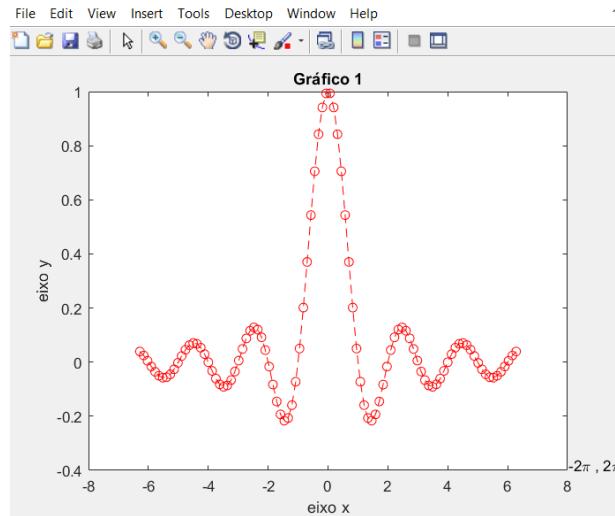
Caracteres especiais:

Sequência	Caractere	Sequência	Caractere
<code>\alpha</code>	α	<code>\upsilon</code>	υ
<code>\angle</code>	\angle	<code>\phi</code>	ϕ
<code>\ast</code>	$*$	<code>\chi</code>	χ
<code>\beta</code>	β	<code>\psi</code>	ψ
<code>\gamma</code>	γ	<code>\omega</code>	ω

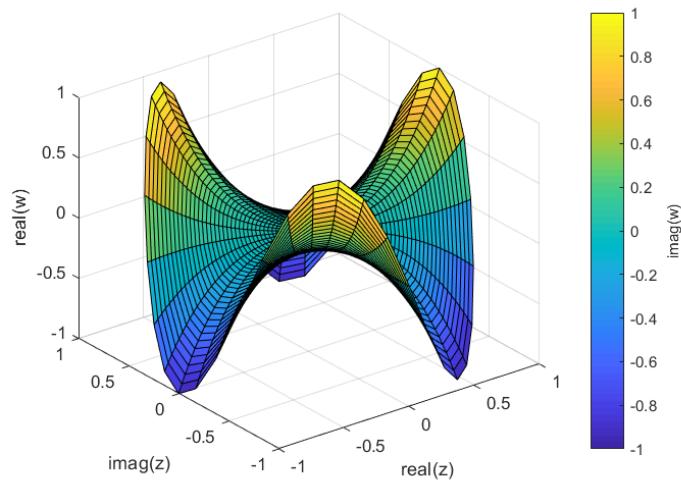
\delta	δ	\Gamma	Γ
\epsilon	ϵ	\Delta	Δ
\zeta	ζ	\Theta	Θ
\eta	η	\Lambda	Λ
\theta	θ	\Xi	Ξ
\vartheta	ϑ	\Pi	Π
\iota	ι	\Sigma	Σ
\kappa	κ	\Upsilon	Υ
\lambda	λ	\Phi	Φ
\mu	μ	\Psi	Ψ
\nu	ν	\Omega	Ω
\xi	ξ	\forall	\forall
\pi	π	\exists	\exists
\rho	ρ	\ni	\ni
\sigma	σ	\cong	\cong
\varsigma	ς	\approx	\approx
\tau	τ	\Re	\Re
\equiv	\equiv	\oplus	\oplus
\Im	\Im	\cup	\cup
\otimes	\otimes	\subseteq	\subseteq
\cap	\cap	\in	\in
\supset	\supset	\lceil	\lceil
\int	\int	\cdot	\cdot
\rfloor	\rfloor	\neg	\neg
\lfloor	\lfloor	\times	\times
\perp	\perp	\surd	\surd
\wedge	\wedge	\varpi	ϖ
\rceil	\rceil	\rangle	\rangle
\vee	\vee	\langle	\langle
\sim	\sim	\propto	\propto
\leq	\leq	\partial	∂
\infty	∞	\bullet	\bullet
\clubsuit	\clubsuit	\div	\div
\diamondsuit	\diamondsuit	\neq	\neq
\heartsuit	\heartsuit	\aleph	\aleph
\spadesuit	\spadesuit	\wp	\wp
\leftrightarrow	\leftrightarrow	\oslash	\oslash
\leftarrow	\leftarrow	\supseteq	\supseteq
\Leftarrow	\Leftarrow	\subset	\subset
\uparrow	\uparrow	\circ	\circ
\rightarrow	\rightarrow	\nabla	∇
\Rightarrow	\Rightarrow	\dots	\dots
\downarrow	\downarrow	\prime	\prime
\circ	\circ	\emptyset	\emptyset
\pm	\pm	\mid	\mid
\geq	\geq	\copyright	\copyright

Exemplo:

```
>>X=linspace (-2*pi, 2*pi, 100); Y=sinc(X);
>>plot(X, Y, 'r--o')
>>xlabel('eixo x'); ylabel('eixo y');
>>title('Gráfico 1')
>>gtext('-2\pi , 2\pi')
```



```
>> clear, clc
>> r = (0:0.025:1)'; theta = pi*(-1:0.05:1);
>> z = r*exp(i*theta); w = z.^3;
>> surf( real(z), imag(z), real(w))
>> cb = colorbar;
>> xlabel('real(z)'); ylabel('imag(z)'); zlabel('real(w)');
cb.Label.String = 'imag(w)';
```



Exercícios

- 1) O MATLAB possui funções prontas para se plotar algumas figuras tridimensionais. Utilize o comando help, caso necessário e use as seguintes funções para se plotar sólidos já definidos:

- a) sphere
- b) cylinder
- c) ellipsoid

- 2) Utilizando a função plot, plote, em um mesmo gráfico, a função seno de azul, e a função cosseno de vermelho. Ambas, no intervalo de $(0, 2\pi)$.

- 3) Coloque na imagem da questão anterior:

- a) Título “Seno e Cosseno”
- b) Legenda no eixo x “0 a 2π ”
- c) Legenda no eixo y “0 a 1”

- 4) Realize a sequência de comandos e observe o gráfico:

```
k = 1:20;
k_str = num2str(k);
f = ['sinc([' k_str, ']*x)'];
fplot(f, [-1 1])
```

- 5) Plote três gráficos em três figuras diferentes com as seguintes especificações:

- a) $y = x^2 - 2x + 5$, de -10 a 10, de azul
- b) $y = 2^x / x$, de -5 a 5, de vermelho
- c) $y = \text{sinc}(x)$, de -50 a 50, de verde

- 6) A superfície cone é definida como:

$$x^2/a^2 + y^2/b^2 = z^2/c^2$$

Plote o gráfico dessa função utilizando a função mesh, sendo $a=2$, $b=3$ e $c=5$. Adicione ainda, nesse mesmo gráfico, o contorno 2D.

Dica: isole z, plote o gráfico positivo, congele a imagem, plote a parte negativa.

- 7) Plote o gráfico de uma superfície conhecida como paraboloide hiperbólico definido por:

$$f(x, y) = (x - 3)^2 - (y - 2)^2, 2 \leq x \leq 4, 1 \leq y \leq 3$$

Note que sua variável z será a própria $f(x, y)$.

- 8) A função exponencial e^x pode ser interpretada como um somatório de termos infinitos $\sum_{n=0}^{\infty} \frac{x^n}{n!}$. Esse processo é chamado de expansão em série de Taylor. Utilizando linhas tracejadas e o intervalo de -2 a 5, para facilitar a visualização, plote, em uma mesma figura o somatório com n variando de 0 a:

- a) 2
- b) 3
- c) 4
- d) 5

Em seguida, plote a própria função exponencial e^x , em linha contínua e no intervalo de -2 a 5, e observe como a função converge a medida em que somamos o próximo termo. Insira legenda.

Dica: expandindo o somatório teremos: $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$

- 9) Repita o processo da questão anterior com a função $\cos(x)$, usando o intervalo de 0 a 2π .

Sua expansão em série é descrita por: $\sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!} (-1)^n$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

- 10) Utilize a função `randn` para introduzir ruídos a uma função de primeiro grau: Uma função de primeiro grau é gerada pela forma geral $f(x) = ax + b$. Tome $a=2$ e $b=1$. Utilize um intervalo de $-2, 10$. Pinte em uma mesma figura dois gráficos: o gráfico da função original e o gráfico da função com ruídos ($f_2(x) = ax + b + \text{ruídos}$).

Polinômios

O MATLAB também é uma boa ferramenta para se trabalhar com polinômios. Eles são representados por um vetor contendo os coeficientes do polinômio em ordem decrescente, por exemplo, $x^3 + 2x - 5$ será representado por $p = [1, 0, 2, -5]$. Algumas das funções para polinômios já foram expostas anteriormente, e serão revistas e melhores detalhadas.

8.1 Raízes

Podemos calcular as raízes a partir de um polinômio, e também um polinômio através de raízes com os comandos:

`roots(Polinômio)` retorna as raízes do polinômio;
`poly(Raízes)` retorna os coeficientes do polinômio em que as raízes são os números do arranjo de parâmetro;

Exemplo:

```
>> P=[ 1, 1, 4, 0];
>> roots(P)
ans =
    0.0000 + 0.0000i
   -0.5000 + 1.9365i
   -0.5000 - 1.9365i
>> P2=poly(ans)
P2 =
    1.0000    1.0000    4.0000         0
```

Se o argumento de `poly()` for uma matriz quadrada, irá retornar os coeficientes do polinômio característico dessa matriz, portanto, vamos usar esses artifícios para se calcular os autovalores de uma dada matriz.

$$\det([A] - \lambda[I]) = \begin{vmatrix} 1-\lambda & 2 & 3 \\ 4 & 5-\lambda & 6 \\ 7 & 8 & 9-\lambda \end{vmatrix} = \lambda^3 - 15\lambda^2 - 18\lambda$$

```
>> B = [2, 2, 2; 0, 0, 0; 1, 2, 3];
>> pp = poly(B)
pp =
    1      -5      4      0
>> roots(pp)
ans =
    0
    4
    1
```

8.2 Produto e Divisão

Para tais cálculos, temos as funções:

`conv(Polinômio1, Polinômio2)` retorna os coeficientes da multiplicação dos polinômios;

`deconv(Polinômio1, Polinômio2)` retorna dois vetores, o primeiro é o quociente da divisão entre os polinômios, e o segundo o resto, normalmente usa-se [q, r] = `deconv(y1, y2)`;

No caso da divisão, também é retornado o resto, vejamos o exemplo:

```
>>p1 = [1 -5 4 0]; p2 = [1 -4];
>>conv(p1, p2)
ans =
    1     -9     24    -16      0
>> [q, r] = deconv(p1, p2)
q =
    1     -1      0
r =
    0     0      0      0
```

Observe que o p2 é exatamente uma raiz de p1, portanto ao dividirmos p1 por p2, estamos fatorando e retirando uma raiz = 4 (e também por isso, o resto é 0), veja:

```
>> roots(p1), roots(q)
ans =
    0
    4
    1
ans =
    0
    1
```

8.3 Avaliação

Utilizando `polyval` podemos avaliar o polinômio:

`polyval(Polinômio, Valor(es))` retorna o resultado das avaliações do polinômio nos valores fornecidos;

Exemplo:

```
>> P=[4 4 0 2 -5]; x=0:0.5:2;
>> polyval(P, x)
ans =
-5.0000    -3.2500     5.0000    31.7500    95.0000
```

8.4 Frações Parciais

Transformação em frações parciais é utilizada em várias aplicações, como as transformadas de Laplace e Fourier, ou qualquer outro problema que envolva divisão de polinômios. A ideia é basicamente transformar uma razão entre dois polinômios B/A, em uma soma de outras frações, para facilitar os cálculos. Como por exemplo:

$$\frac{B(s)}{A(s)} = \frac{10s^3 + 80s^2 + 177s + 95}{s^3 + 8s^2 + 19s + 12}$$

$$\frac{B(s)}{A(s)} = \frac{9}{s+4} + \frac{-7}{s+3} + \frac{-2}{s+1} + 10$$

Para isso temos a função residue:

[r, p, k] = residue(n, d) sendo n e d, o numerador e denominador, respectivamente, a função retorna r=resíduo, p=polo, k=termo direto;

[n, d] = residue(r, p, k) sendo r=resíduo, p=polo, k=termo direto, a função retorna n e d, o numerador e denominador, respectivamente;

Exemplo:

```
>>B=[10 80 177 95]; A=[1 8 19 12];
>> [res, pol, k]= residue(B, A)
res =
    9.0000
   -7.0000
   -2.0000
pol =
   -4.0000
   -3.0000
   -1.0000
k =
    10
```

8.5 Ajuste Polinomial

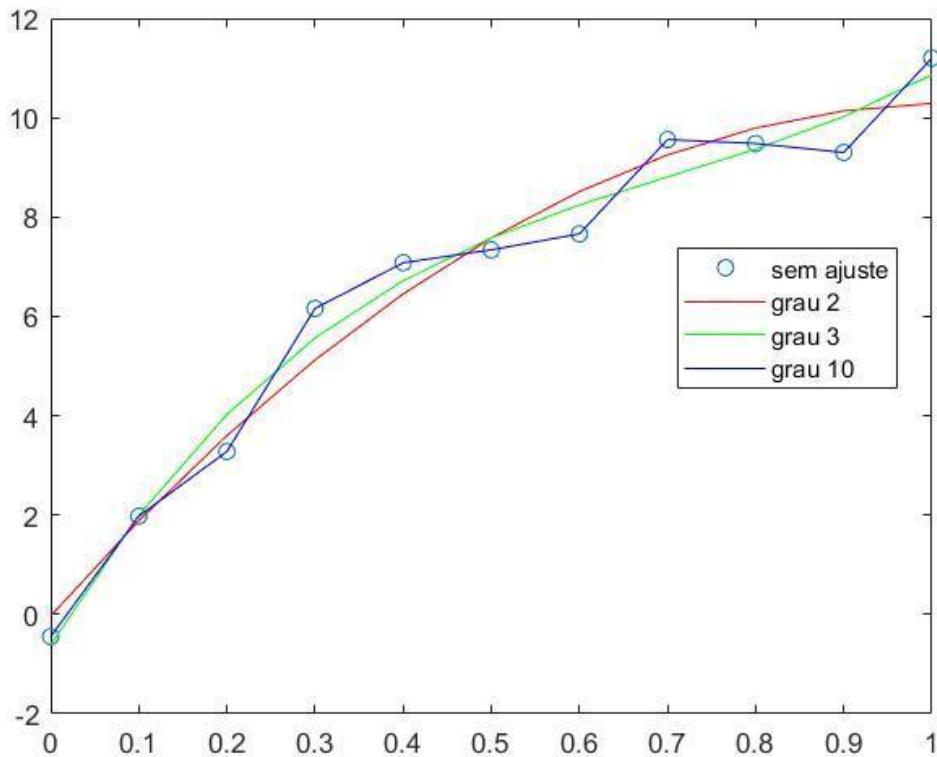
O ajuste ou regressão polinomial pode ser feito no MATLAB utilizando a função:

`polyfit(x, y, n)` retorna os coeficientes do polinômio ajustado de grau n nos valores de x e y;

Nesta função é utilizado o método dos quadrados mínimos, no qual busca os valores dos coeficientes do polinômio que fazem com que o quadrado da distância entre os pontos fornecidos e a curva gerada sejam as mínimas possíveis, calculado por meio da derivada.

Exemplo:

```
>> x=0:0.1:1; y = [-0.447 1.978 3.28 6.16 7.08 7.34 7.66
9.56 9.48 9.30 11.2];
>>pp2 = polyfit(x,y,2); pp3 = polyfit(x,y,3); pp10 =
polyfit(x,y,10);
>>yp2 = polyval(pp2, x); yp3 = polyval(pp3, x); yp10 =
polyval(pp10, x);
>>plot(x, y, 'o', x, yp2, 'r', x, yp3, 'g', x, yp10, 'b')
>> legend('sem ajuste', 'grau 2', 'grau 3', 'grau 10')
```



Plotagem referente ao exemplo

Note que possuímos 11 pontos, e pp10, o de azul, é o polinômio de grau 10. Pelo ajuste, este polinômio será o próprio polinômio interpolador, pois por $n+1$ pontos, passa somente um único polinômio de grau n .

Exercícios:

- 1) Em análise numérica, três polinômios auxiliares auxiliam a limitação das do intervalo das raízes do polinômio original, pois suas raízes tem propriedades em relação às raízes originais. Sendo $P(x)$ o polinômio, os auxiliares são: $P_1(x) = x^n P(1/x)$, $P_2(x) = P(-x)$, $P_3(x) = x^n P(-1/x)$. Utilizando a função roots, encontre as raízes dos seguintes polinômios e observe a sua relação:
 - a) $P(x) = x^5 - 15x^4 + 85x^3 - 225x^2 + 274x - 120$
 - b) $P_1(x) = x^5 P(1/x) = -120x^5 + 274x^4 - 225x^3 + 85x^2 - 15x + 1$
 - c) $P_2(x) = P(-x) = -x^5 - 15x^4 - 85x^3 - 225x^2 - 274x - 120$
 - d) $P_3(x) = x^5 P(-1/x) = 120x^5 + 274x^4 + 225x^3 + 85x^2 + 15x + 1$
- 2) Existe uma função no MATLAB, eig(Matriz) que retorna diretamente os autovalores de uma Matriz. Utilizando as funções poly e roots, ou seja, sem usar a função eig, encontre os autovalores das seguintes matrizes:
 - a)
$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

- b) 1 2 3
4 5 6
7 8 9
- c) 0 0
1 1
- d) 98 88 2.04
0.13 -12 9.08
11.7 11.7 2.22
- 3) O ajuste polinomial coincide com a interpolação polinomial quando o grau do polinômio ajustado é suficientemente grande. Sabendo que por $n+1$ pontos passa somente um polinômio interpolador de grau n e tomando $x = 0.0 \ 0.1 \ 0.2 \ 0.3 \ 0.4 \ 0.5$ e $y = 15.0317 \ 15.0234 \ 15.0710 \ 15.1970 \ 15.3796 \ 15.5509$ encontre:
- O polinômio interpolador, usando ajuste polinomial
 - O ajuste polinomial de grau 1
 - O ajuste polinomial de grau 2
- 4) Tendo os polinômios da questão anterior, crie um vetor H de 100 elementos igualmente espaçados e o avalie em cada um dos três polinômios encontrados, para plotagem. Em seguida, plote, na mesma imagem os seguintes gráficos:
- O polinômio interpolador
 - O polinômio de grau 1
 - O polinômio de grau 2
 - Os pontos x e y da questão anterior
- 5) Realize as seguintes operações com os polinômios a, b e c:
- $a = x^4 + 5x^3 - 5x^2 + 10$
 - $b = x^5$
 - $c = (x-1)(x-3)$
 - b/a
 - $a*c$
 - $10*b$
 - raízes ($5*a-c$)
- 6) Decomposição em frações parciais é um processo muito importante na resolução de algumas integrais e transformações de Laplace. Use a função residue para encontrar a decomposição em frações parciais da seguinte função racional:

$$f(x) = \frac{x+1}{x(x+2)(x+4)}$$

- 7) Faça o processo inverso da questão anterior: use a função residue e as frações parciais para chegar a função racional:

$$f(x) = -\frac{3}{8(x+4)} + \frac{1}{4(x+2)} + \frac{1}{8x} = \frac{x+1}{x(x+2)(x+4)}$$

- 8) Em um determinado experimento de física, deseja-se medir a constante elástica k de uma certa mola de 10cm. Para isso, estão à disposição seis pesos de 0.5kg e uma haste com uma régua. O tamanho da mola a cada quantidade de massa colocada foi medida, e foram montados os vetores $x = 0 \ 0.5000 \ 1.0000 \ 1.5000 \ 2.0000 \ 2.5000 \ 3.0000$ referente a massa em kg e $y = 0 \ 0.0550 \ 0.1010 \ 0.1630 \ 0.2000 \ 0.2520 \ 0.29203900$ referente à deformação da mola em m.

Sabendo que a força elástica de uma mola é dada segundo a equação $F = kx$, e que a força é o peso $P = mg$, encontre a constante k da mola. Use $g = 9.8\text{m/s}^2$. Dicas: compare a equação da força elástica com a função $y = ax + b$. Use o ajuste polinomial. Preste atenção em quais valores do ajuste considerar e porque.

- 9) Plote em uma mesma imagem os pontos do experimento e a função linear devidamente ajustada. Observe o resultado.
- 10) Refaça o processo a seguir utilizando polinômios:

```
>>syms x  
>> f = x^4 -3*x^3 + 4*x;  
>> solve(f)  
ans =  
-1  
0  
2  
2
```

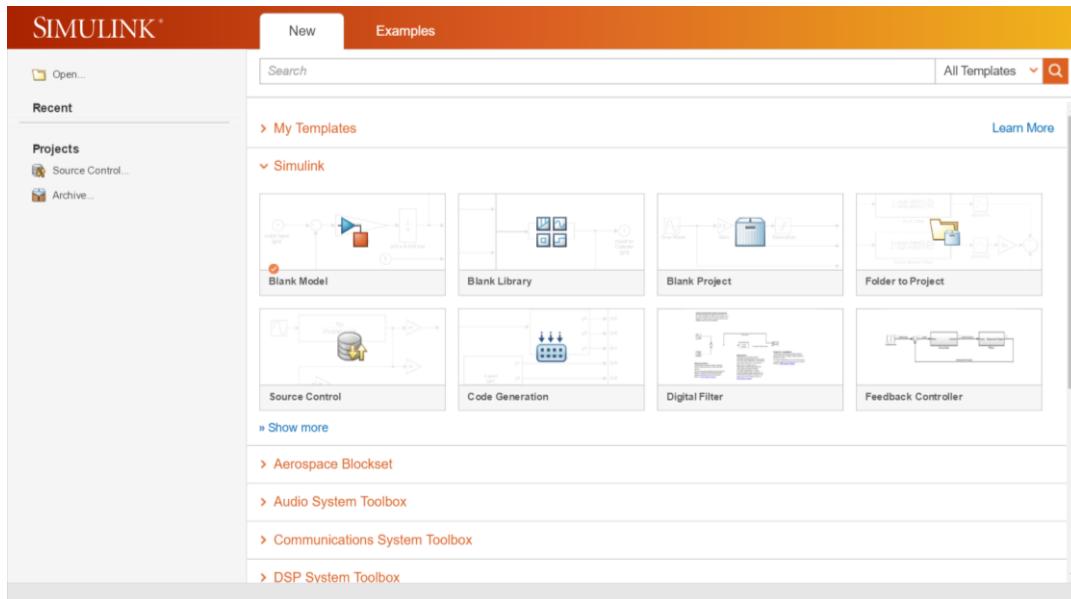
Simulink: Introdução

O MATLAB apresenta uma ferramenta poderosa e cheia de utilidades: o Simulink. Pode ser usado para modelar, analisar e simular sistemas dinâmicos, comportando também sistemas lineares e não-lineares. É possível simular diversas coisas, desde o mais simples como uma equação diferencial até mais complexos como grandes sistemas mecânicos e elétricos.

É integrado ao MATLAB, porque além de compartilharem funções e procedimentos, os dados de um podem ser facilmente exportados ou importados para o outro.

9.1 O Ambiente

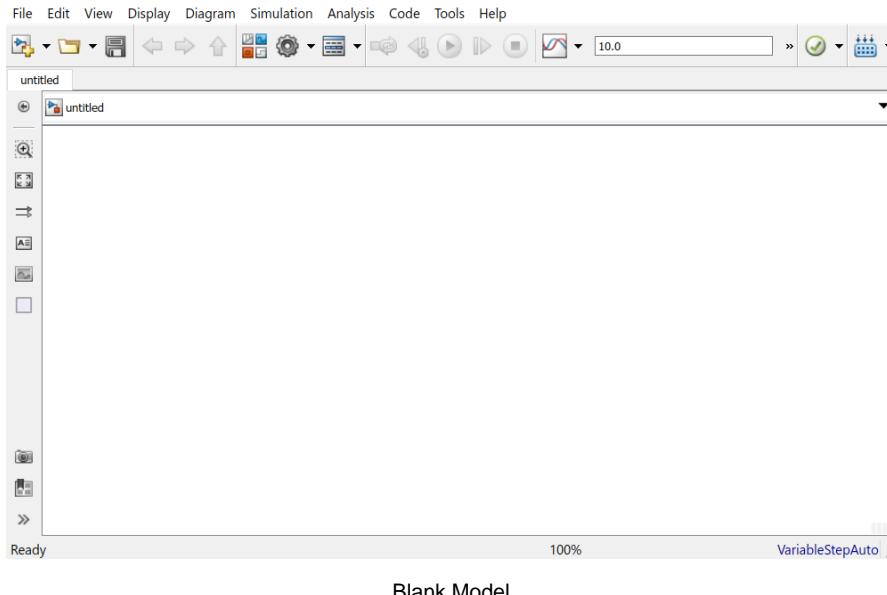
Para acessar o Simulink basta digitar “simulink” na Command Window ou clicar no ícone na barra de utilidades, na parte superior do ambiente MATLAB.



Ambiente Simulink

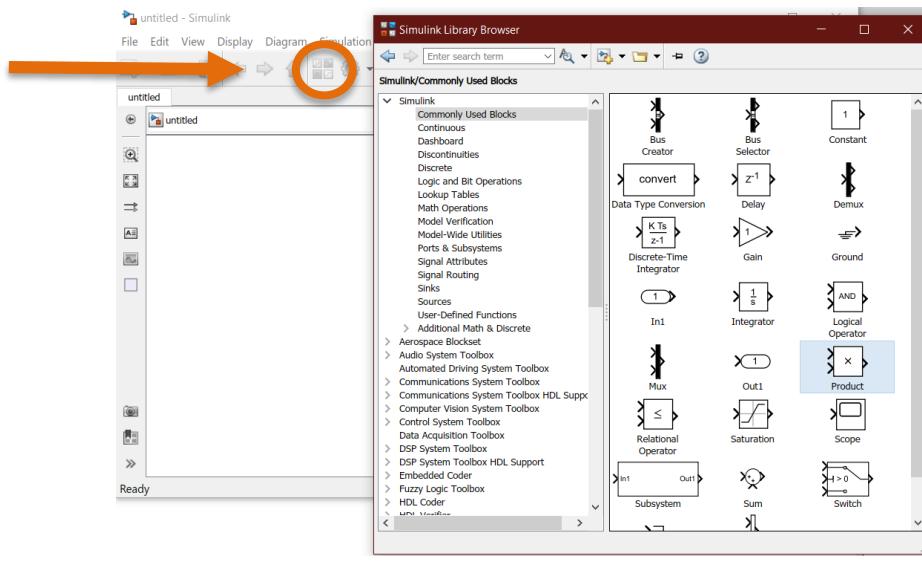
O Simulink possui interface gráfica do usuário, conhecido como GUI (Graphical User Interface). Com isso, o usuário pode modelar visualmente como se estivesse usando lápis e papel.

Na página inicial há vários modelos, templates e exemplos de sistemas. Clicando em “Blank Model” é aberto um novo modelo em branco para o usuário trabalhar.



Modelos podem ser salvos e abertos pela aba “File”. Clicando duas vezes sobre qualquer espaço branco do modelo em branco, pode-se adicionar uma anotação qualquer.

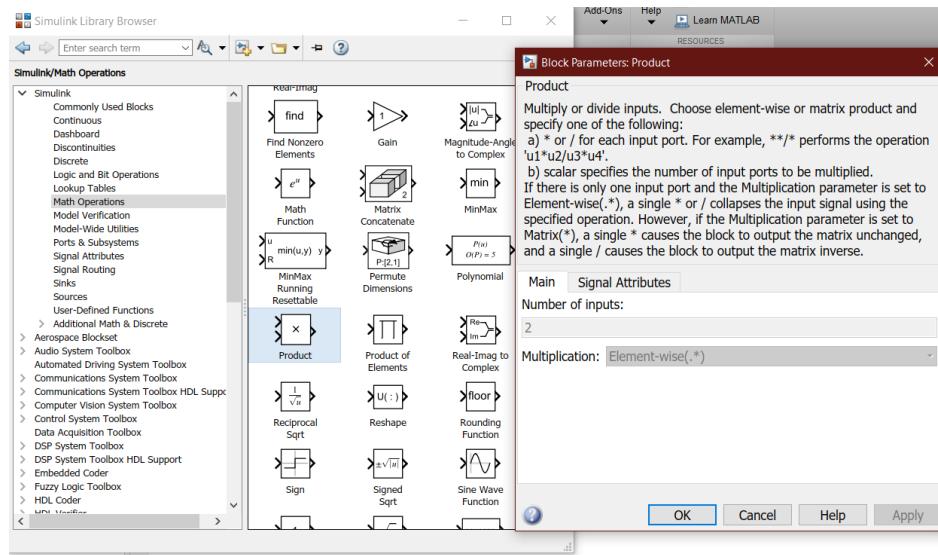
A barra superior apresenta opções úteis, como “run” – rodar a simulação, novo modelo, abrir, etc. Também há a opção “Library Browser”. Clicando nela, é aberta a biblioteca (com outras bibliotecas dentro) onde estão os blocos pré-definidos:



Aqui observa-se uma série de blocos: operações matemáticas; contínuos; discretos; fontes de sinais; atribuição de sinais; processamento de sinais; dentre vários outros.

Clicando com o botão direito sobre um determinado bloco e depois na opção “Block parameters” ou clicando diretamente duas vezes com o botão esquerdo sobre o bloco é aberto o guia de parâmetros do bloco. Caso o bloco estivesse em uso, alguns parâmetros seriam editáveis. Nessa aba, aparece também a descrição do bloco. Muito útil para sabermos o que o bloco faz, uma vez que

existem diversos deles. Exemplo de parâmetros do bloco “Product”, dentro de “Math Operations” da biblioteca “Simulink”:



Block Parameters: Product

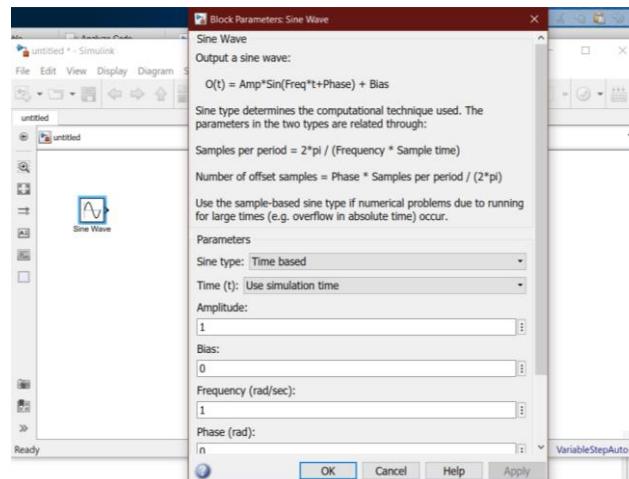
9.2 Modelagem

Um bloco pode ser adicionado a um modelo clicando com o botão direito sobre ele e em “Add block to model” ou simplesmente clicando e arrastando-o até o modelo. Os blocos possuem inputs e outputs, conectando-os montamos um sistema. Lembre-se também que alguns parâmetros dos blocos em uso podem ser alterados clicando em “Block parameters” ou duas vezes sobre o bloco.

9.2.1 Exemplo

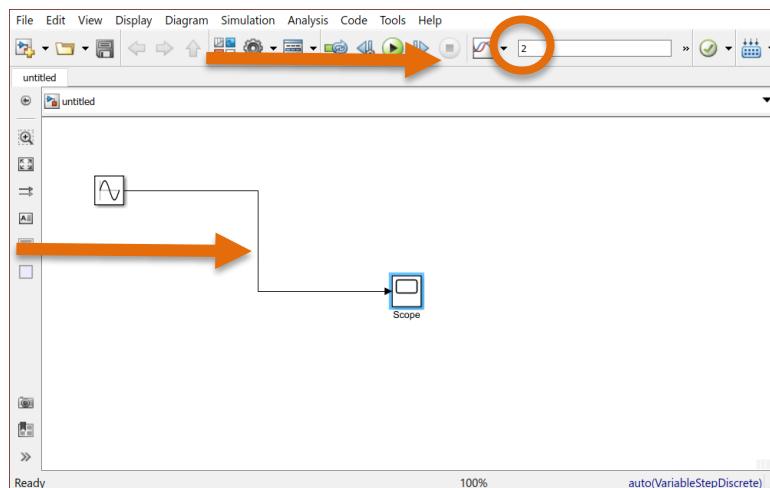
Vamos criar um exemplo para melhor compreensão:

Inicialmente, o objetivo é coletar dados de uma fonte senoidal. Para isso vamos adquirir o bloco “Sine Wave” de “Sources”. Adicionando-o ao modelo, podemos arrastá-lo com o mouse para qualquer posição, renomeá-lo clicando duas vezes sobre seu nome e editar seus parâmetros:

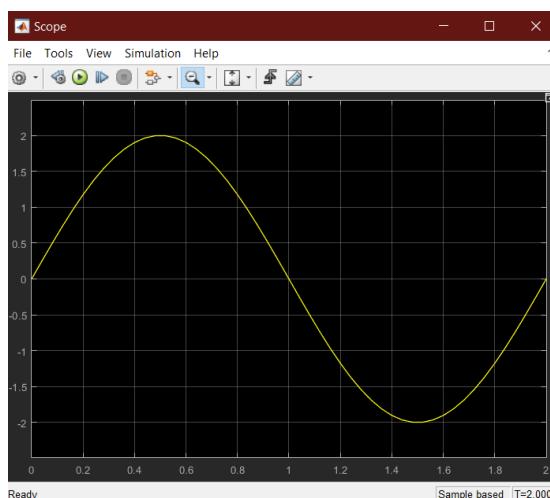


Vamos alterar a amplitude para 2 e a frequência para π . Note que podemos usar a mesma sintaxe que no MATLAB, logo π será a constante “pi”.

Agora, necessitamos de um mecanismo para verificar a onda. Iremos utilizar o bloco “Scope” de “Sinks”. O bloco “Display” mostra o valor recebido em tempo real, logo, visualizariamós, no fim do processo, apenas o último valor, e não a onda como um todo. Por isso que utilizaremos o osciloscópio “Scope”. Clicando na saída do “Sine Wave” e arrastando-o até “Scope” transmitimos o sinal do primeiro para o segundo. Vamos alterar o tempo do modelo de 10 para 2 segundos.



Clicando em “Run” (círculo verde com símbolo de play) nosso sistema é executado. Clicando duas vezes sobre o “Scope”, temos nosso resultado:



O resultado é como esperávamos: uma curva seno com 2 de amplitude definida entre 0 e 2π (frequência = π rad/s e 2 segundos de execução).

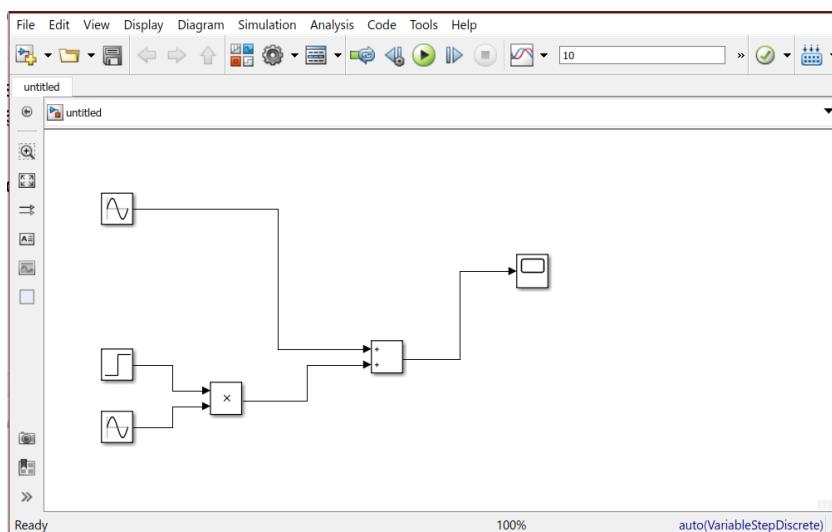
Agora, vamos incorporar o nosso sistema:

Vamos voltar o tempo de execução para 10 segundos, adicionar a onda uma perturbação devido a outra onda, porém apenas após 2 segundos.

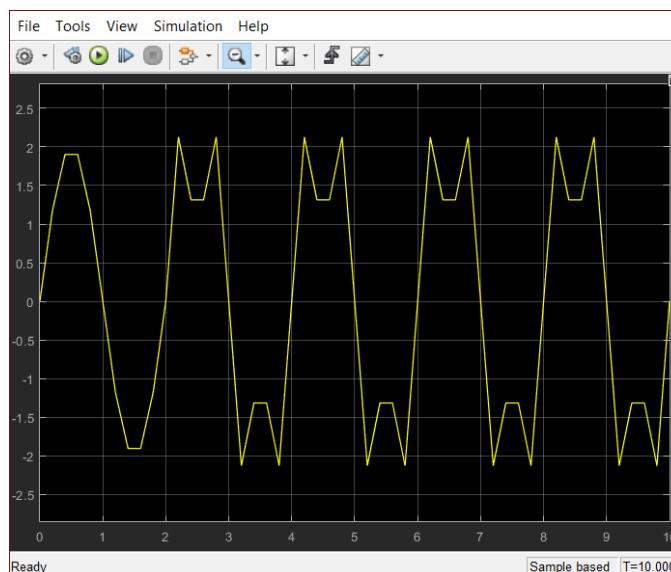
Para isso, vamos precisar de outro “Sine Wave” e dos blocos “Step” de “Sources”, “Product” e “Add” de “Math Operations”.

Vamos alterar somente a frequência do novo “Sine Wave” para 3π . O “Step” é um bloco que salta de um valor inicial para um valor final após um determinado tempo. Vamos alterar somente o tempo de salto (“Step time”) para 2 segundos, mantendo o valor inicial como 0 e o final como 1. Feito isso, vamos usar “Product” para multiplicar os dois blocos alterados anteriormente. Assim, o novo sinal só será considerado a partir de dois segundos.

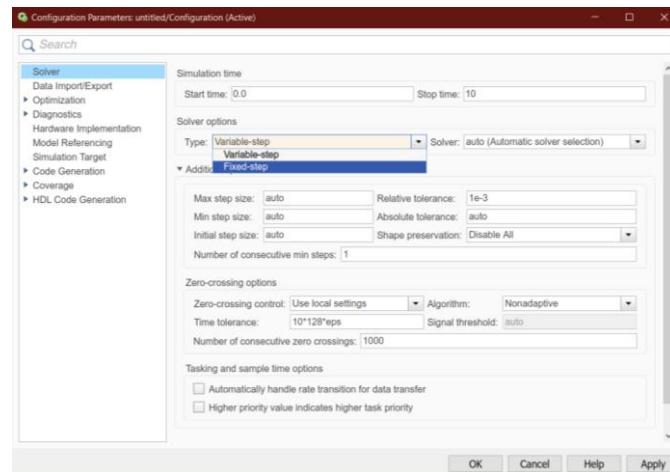
Por fim, vamos usar o bloco “Add” para somar a saída da multiplicação anterior e o sinal senoidal antigo. Daí, o resultado será transmitido até o osciloscópio (“Scope”). Nosso sistema ficará assim:



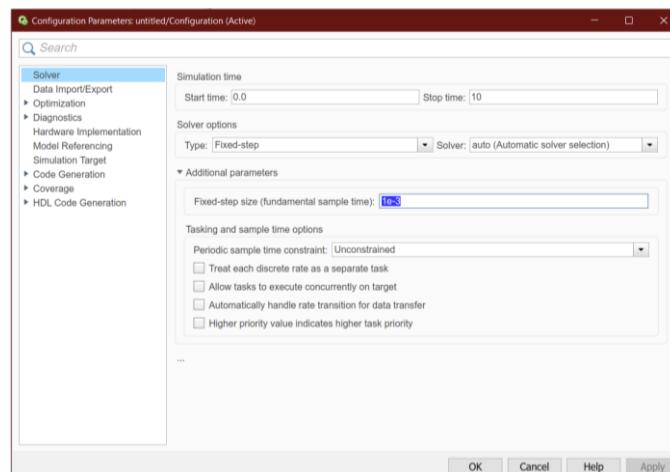
Ao executarmos e abrirmos o osciloscópio, temos o seguinte resultado:



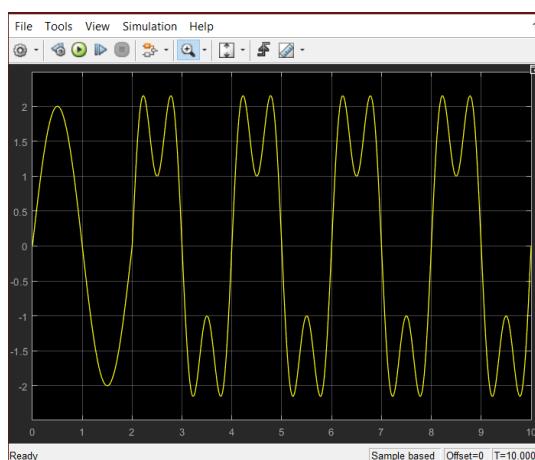
Podemos observar que a forma do gráfico não ficou bem definida devido ao intervalo entre os pontos. Para isso, no modelo, clicamos em configurações (“Model configuration parameters”) e alteramos o tipo de passo do modelo de “Variable-step” para “Fixed-step”:



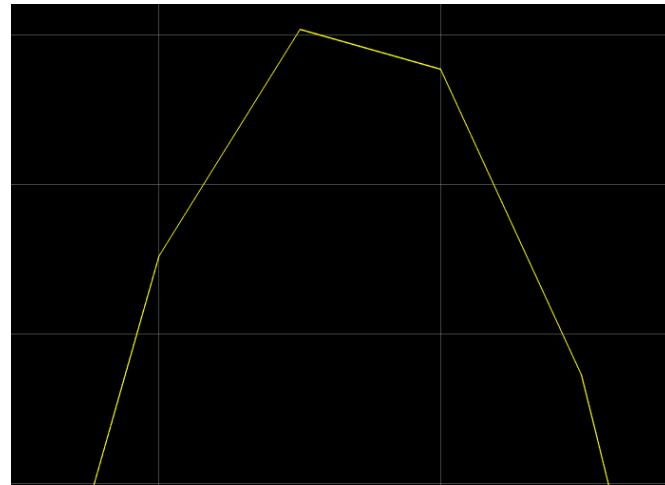
Após, em “Fixed-step size (fundamental sample size)” vamos diminuir o intervalo. Usemos 1e-3.



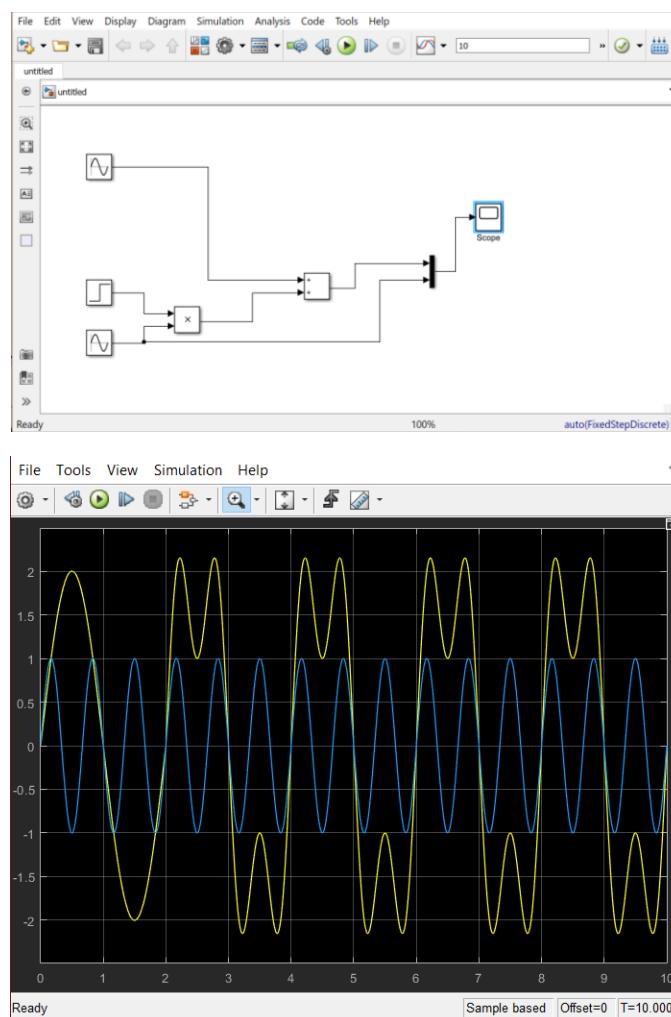
Agora, clicamos em Apply e em Ok, e verificamos os novos resultados após executar novamente o sistema:



O gráfico ficou melhor definido. Mas, se dermos zoom o suficiente, sempre iremos observar a aproximação por retas:



Vamos fazer uma última modificação ao nosso sistema: Vamos mostrar no osciloscópio a atual onda e a onda do segundo gerador de ondas, simultaneamente. Para isso, precisaremos do bloco "Mux" de "Singal Routing". Daí, puxamos de uma parte da seta que já existe entre o segundo gerador de onda e o multiplicador e ligamos até uma das entradas do "Mux". A outra entrada, será a atual entrada do osciloscópio, e a saída do "Mux", irá para o osciloscópio:



Exercícios:

- 1) Crie um simples sistema para realizar uma multiplicação entre dois valores arbitrários. Exiba o resultado em um “Display”.
- 2) Crie um modelo que recebe um sinal senoidal, multiplica por dois, arredonda para o inteiro mais próximo, e divide por dois. Exiba o resultado em um “Scope”.
- 3) Crie um novo modelo. Utilizando os blocos “Sine Wave”, “Mux” e “Scope”, mostre em um mesmo osciloscópio as funções, de 0 até 2π :
 - a) $f(x) = \sin(x)$
 - b) $f(x) = \sin(x + \frac{\pi}{2})$
 - c) $f(x) = \sin(\frac{x}{2})$
 - d) $f(x) = \frac{\sin(2x)}{2}$
- 4) No modelo anterior, utilize o bloco “To Workspace” como saída para poder exportar os dados coletados para a área de trabalho do MATLAB. Use como parâmetros, exportação como arranjo.
- 5) Utilizando os dados, no MATLAB, plote os quatro gráficos.
- 6) Em lançamentos oblíquos sem atrito, a posição em y em função do tempo é descrita como:

$$y(t) = y_0 + V_{0y}t + \frac{at^2}{2}$$

Crie um sistema para simular a variação de y em função do tempo de um determinado lançamento oblíquo. Use $a = g = -9.8$, $V_{0y} = 50$ e $y_0 = 0$.

- 7) Nos mesmos lançamos, a posição x em função do tempo é descrita como

$$x(t) = x_0 + V_{0x}t$$

No mesmo sistema anterior, altere $V_{0y} = 40$, e adicione o modelo de x em função do tempo. Transfira os dados coletados em forma de vetor para a área de trabalho do MATLAB. Considere $V_{0x} = 30$ e $x_0 = 0$. Por fim, plote um gráfico de x(t) por y(t), ajuste o eixo y para não mostrar valores negativos do seu lançamento.

- 8) Modele a seguinte equação diferencial e observe o gráfico:

$$\begin{aligned} y'' + 2y' + 5y &= 1 \\ y'(0) = y(0) &= 0 \end{aligned}$$

- 9) Modele a seguinte equação diferencial e observe o gráfico:

$$\begin{aligned} y'' + 4y' + 5y &= 3 \\ y'(0) = y(0) &= 0 \end{aligned}$$

- 10) Produza pulsos quadrados utilizando os blocos “Sine Wave” e “Sign”.

Resolução dos Exercícios

2. Estrutura de Dados

1)a)

```
>>5.17e-5 * 17
```

```
ans =
```

```
8.7890e-04
```

b)

```
>>13.711*13.711
```

```
ans =
```

```
187.9915
```

c)

```
>>ans/5.17e-5
```

```
ans =
```

```
3.6362e+06
```

d)

```
>>-1/-1.6022e-19
```

```
ans =
```

```
6.2414e+18
```

e)

```
>> 50/342.3
```

```
ans =
```

```
0.1461
```

2)

```
>>0/0
```

```
ans =
```

```
NaN
```

```
>>10/0
```

```
ans =
```

```
Inf
```

`NaN` e `Inf` são constantes do MATLAB que não representam valores numéricos. `NaN` é o valor que representa “não é um número”, e `Inf` representa infinito.

3)a)b)c)

```
>>NmrMatricula = 201900; Notas = [0 0 0 0 0]; Ocorrencias=
' O aluno não possui ocorrências';
```

d)

```
>>Aluno(1,1).nome = 'Jonatan'; Aluno(1,1).matricula =
NmrMatricula; Aluno(1,1).notas = Notas; Aluno(1,1).ocor =
Ocorrencias;
```

```
>>Aluno(2,1).nome = 'Patricia'; Aluno(2,1).matricula =
NmrMatricula; Aluno(2,1).notas = Notas; Aluno(2,1).ocor =
Ocorrencias;
```

4)

Variables - Aluno				
	Aluno	X		
	2x1 struct with 4 fields			
...	ch nome	grid matricula	grid notas	ch ocor
1	'Jonatan'	201900	[0 0 0 0 0]	'O aluno nã...
2	'Patricia'	201900	[0 0 0 0 0]	'O aluno nã...
3				
4				

5)

```
>>Aluno(1,1).matricula = NmrMatricula + 1; Aluno(1,1).notas
= [10 8 9.5 7 10];
```

```
>>Aluno(2,1).matricula = NmrMatricula + 2; Aluno(2,1).notas
= [10 10 3 9 7.8]; Aluno(2,1).ocor = 'Duas brigas em sala
de aula';
```

6)

Variables - Aluno				
	Aluno	X		
	2x1 struct with 4 fields			
...	ch nome	grid matricula	grid notas	ch ocor
1	'Jonatan'	201901	[10 8 9.500...]	'O aluno nã...
2	'Patricia'	201902	[10 10 3 9 ...]	'Duas brigas em sala de aula'
3				
4				

7)a)

```
>>A = [2+2i, 0, 7i, 13+2i];
>> B = [A(1)', A(2)', A(3)', A(4)']
B =
    2.0000 - 2.0000i    0.0000 + 0.0000i    0.0000 - 7.0000i
    13.0000 - 2.0000i
```

8)

```
>>syms x
>>y=2*x^2 - 30*x + 100;
>>subs(y, x, 10)
ans =
0
9)
>>subs(y, x, 0)
ans =
0
10)
>> subs(y, x, 0)
ans =
100
```

3. Comandos e Funções

1)

```
>>help cholesky
cholesky not found.

Use the Help browser search field to search the documentation, or
type "help help" for help command options, such as help for
methods.

>> lookfor cholesky
chol           - Cholesky factorization.
cholupdate     - Rank 1 update to Cholesky factorization.
ichol           - Sparse Incomplete Cholesky factorization
```

finchol - Cholesky factor for positive semi-definite correlation matrices.

covlamb - Covariance matrix of Cholesky factor of

cholcov - Cholesky-like decomposition for covariance matrix.

>>help chol

chol Cholesky factorization.

chol(A) uses only the diagonal and upper triangle of A.

The lower triangle is assumed to be the (complex conjugate) transpose of the upper triangle. If A is positive definite, then R = **chol**(A) produces an upper triangular R so that $R^*R = A$. If A is not positive definite, an error message is printed

L = **chol**(A, 'lower') uses only the diagonal and the lower triangle of A to produce a lower triangular L so that $L^*L' = A$. If A is not positive definite, an error message is printed. When A is sparse, this syntax of **chol** is typically faster.

[R,p] = **chol**(A), with two output arguments, never produces an error message. If A is positive definite, then p is 0 and R is the same as above. But if A is not positive definite, then p is a positive integer. When A is full, R is an upper triangular matrix of order q = p-1 so that $R^*R = A(1:q,1:q)$. When A is sparse, R is an upper triangular matrix of size q-by-n so that the L-shaped region of the first q rows and first q columns of R^*R agree with those of A.

[L,p] = **chol**(A, 'lower'), functions as described above, only a lower triangular matrix L is produced. That is, when A is full, L is a lower triangular matrix of order q = p-1 so that $L^*L' = A(1:q,1:q)$. When A is sparse, L is a lower triangular matrix of size n-by-q so that the L-shaped region of the first q rows and first q columns of L^*L' agree with those of A.

[R,p,S] = **chol**(A), when A is sparse, returns a permutation matrix S which is a preordering of A obtained by AMD. When p = 0, R is an upper triangular matrix such that $R^*R = S^*A*S$. When p is not zero, R is an upper triangular matrix of size q-by-n so that the L-shaped region of the first q rows and first q columns of R^*R agree with those of S^*A*S . The factor of S^*A*S tends to be sparser than the factor of A.

[R,p,s] = **chol**(A, 'vector') returns the permutation information as a vector s such that $A(s,s) = R^*R$, when p =

0. The flag 'matrix' may be used in place of 'vector' to obtain the default behavior.

`[L,p,s] = chol(A,'lower','vector')` uses only the diagonal and the lower triangle of A and returns a lower triangular matrix L and a permutation vector s such that $A(s,s) = L*L'$, when $p = 0$. As above, 'matrix' may be used in place of 'vector' to obtain a permutation matrix.

For sparse A, CHOLMOD is used to compute the Cholesky factor.

See also [cholupdate](#), [ichol](#), [ldl](#), [lu](#).

[Reference page for chol](#)

[Other functions named chol](#)

```
>>A=[9 6 -3 3; 6 20 2 22; -3 2 6 2; 3 22 2 28];
>>chol(A)

ans =
    3     2    -1     1
    0     4     1     5
    0     0     2    -1
    0     0     0     1
```

2)a)b)

```
>>sqrt(187.9915)
ans =
    13.7110
>>round(sqrt(187.9915))
ans =
    14
>>fix(sqrt(187.9915))
ans =
    13
```

c)d)

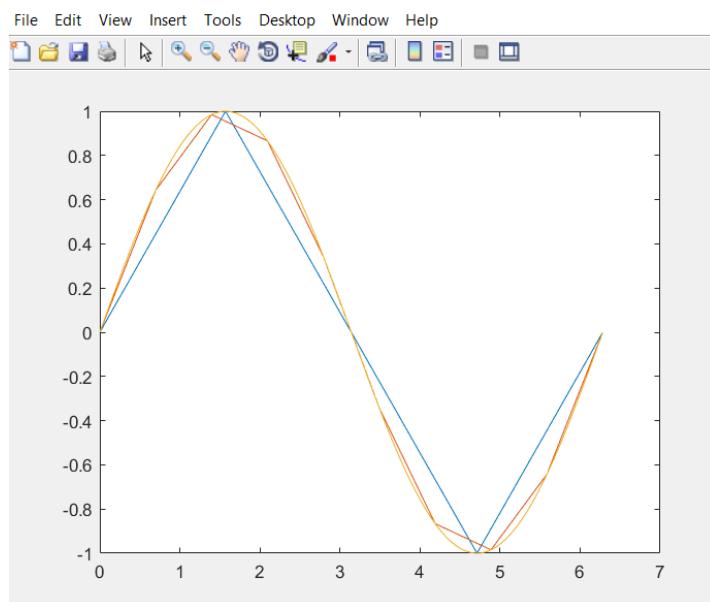
```
>> exp(3)
ans =
    20.0855
>> round(exp(3))
ans =
```

```
20  
>> ceil(exp(3))  
ans =  
21  
e)f)  
>> log(3)  
ans =  
1.0986  
>> round(log(3))  
ans =  
1  
>> ceil(log(3))  
ans =  
2  
3)a)b)c)d)e)  
>>A = [1 2 3];  
>>B = [A; A; A];  
>>P = [2 2];  
>>Q = [P; P];  
>> R=cat(3, Q, Q);  
4)  
>>size(R)  
ans =  
2 2 2  
>>trace(B)  
ans =  
6  
>>det(B)  
>>trace(Q)  
ans =  
4  
>>det(Q)
```

```

ans =
0
5)
>>display('Hello World')
Hello World
>>x=input('insira o valor de x\n')
>> x=input('insira o valor de x\n')
insira o valor de x
4
x =
4
>> fprintf('Olá, programador, x = %i\n', x)
Olá, programador, x = 4
6)
>>a1=linspace(0, 2*pi, 5); b1=sin(a1);
>>a2=linspace(0, 2*pi, 10); b2=sin(a2);
>>a3=linspace(0, 2*pi, 100); b3=sin(a3);
7)
>>plot(a1, b1, a2, b2, a3, b3)

```



8) O gráfico 1 é o de azul, o 2 o de amarelo e o 3 o de laranja. Podemos tirar essa conclusão pois os três espaços ai vão de 0 até 2π , o que difere um do outro é a quantidade de termos. Quanto menos termos, menor a precisão do gráfico seno, logo, o menos preciso é o 1, e o mais preciso o 3.

9)

```
>> abs(13 + 13i)  
ans =  
18.3848  
>> angle(13 + 13i)  
ans =  
0.7854  
>> rad2deg(ans)  
ans =  
45
```

10)

```
>> abs(20i)  
ans =  
20  
>> angle(20i)  
ans =  
1.5708  
>> rad2deg(ans)  
ans =  
90
```

4. Operadores

1)a)

```
>> 13.711^2  
ans =  
187.9915
```

b)

```
>> 13.711^3  
ans =
```

2.5776e+03

c)

```
>> [4, 1, 2]'
```

```
ans =
```

```
4
```

```
1
```

```
2
```

2)

```
>> 10/5, 10\5
```

```
ans =
```

```
2
```

```
ans =
```

```
0.5000
```

A inclinação da barra determina quem ficará em cima e em baixo na fração, sendo / a forma mais usual.

3) Não. Pois, $10 > x$ retornará 1 (verdadeiro) ou 0 (falso) de acordo com o valor de x , mas $1 > 2$ ou $0 > 2$ sempre retornará 0 (falso).

4) Irá retornar 1 para valores de x menores do que 11, pois $(2 < 10 == 1) + 10$ é igual a 10.

5) $((a > 10) == 0) + b > c == \neg d$

```
d=0, a = 11
```

```
( 1 == 0 ) + b > c == 1
```

```
b > c == 1
```

```
b=1; c=0
```

```
1 == 1
```

```
1
```

6) $((a > 10) == 0) + b > c == \neg d$

```
d=0, a = 11
```

```
( 1 == 0 ) + b > c == 1
```

```
b > c == 1
```

```
b=0; c=1
```

```
0 == 1
```

```
0
```

7)a)b)c)d)

```
>>A = [1 1 1; 0 0 0; 1 1 1]; B = [1 6 1; 2 9 0; 1 1 1]; C = [2 4 5]; D=7;
```

8)a)

```
>> A*B  
ans =  
4 16 2  
0 0 0  
4 16 2
```

b)

```
>> A.*B  
ans =  
1 6 1  
0 0 0  
1 1 1
```

c)

```
>> D*A  
ans =  
7 7 7  
0 0 0  
7 7 7
```

d)

```
>> C.^D  
ans =  
128 16384 78125
```

e)

```
>> D.^C  
ans =  
49 2401 16807
```

9)

```
>> A*B  
ans =  
4 16 2  
0 0 0  
4 16 2
```

```
>> A.*B
```

```
ans =
    1     6     1
    0     0     0
    1     1     1
```

O operador “.” indica que a operação será ponto a ponto, ou seja, $A_{ij} \times B_{ij}$, e não uma multiplicação matricial comum.

10) Como é aleatório de distribuição uniforme é de se esperar que provavelmente tenha metade de 0s e metade de 1s, ou seja, 3 e 3.

```
>> A=[rand>0.5 rand>0.5 rand>0.5 rand>0.5 rand>0.5
rand>0.5]
A =
1×6 logical array
1     0     1     0     0     1
```

5. Controle de Fluxo

1)

```
>>a=3; b=2;
>> operador=input('Operação: 1(soma)\n');
Operação: 1(soma)
1
>> if operador == 1
a+b
end
ans =
5
```

2)

```
>> if operador == 2
a-b
end
```

3)

```
>> if operador == 3
a*b
end
```

4)

```
>> if operador == 4  
a/b  
end  
5)  
>> operador=input('Operação: 1(soma)\n');  
>> switch operador  
case 1  
a+b  
case 2  
a-b  
case 3  
a*b  
case 4  
a/b  
end
```

6)

```
>> Nota=input('Digite a nota do aluno(0 a 5)')  
if Nota == 0  
display('Procure o professor para ajuda')  
end  
if Nota == 1  
display('Precisa se dedicar mais')  
end  
if Nota == 2  
display('Ainda um pouco mais')  
end  
if Nota == 3  
display('Bom, 60 é 100')  
end  
if Nota == 4  
display('Parabéns')
```

```
end

if Nota == 5
    display('Sensacional!!!')
end

7)
>> contador=10;
>> for i = 1:10
G(contador)=i^2;
contador=contador - 1;
end
>> G
G =
1          100          81          64          49          36          25          16          9          4

8)
>> contador = 1; valor = 10;
>> while contador < 10
G(contador)=valor^2;
Contador=contador + 1; valor=valor - 1;
end
>> G
G =
1          100          81          64          49          36          25          16          9          4

9)
>> contador=1;
>> for i = [3, 3, 1, 2, 3]
H(contador)=i^3;
contador=contador + 1;
end
>> H
H =
27          27          1          8          27
```

10)

```
>> contador=1; valor=[3 3 1 2 3];
>> while contador < 5
H(contador)=valor(contador)^3;
contador=contador+1;
end
>> H
H =
27      27      1      8      27
```

6. Scripts e arquivos .m

- 1) Clique em New Script no canto superior esquerdo para iniciar um novo script. Código:

```
clear, clc %Limpar a Command Window e o Workspace
Nota=input('Digite a nota do aluno(0 a 5)')
if Nota == 0
display('Procure o professor para ajuda')
end
if Nota == 1
display('Precisa se dedicar mais')
end
if Nota == 2
display('Ainda um pouco mais')
end
if Nota == 3
display('Bom, 60 é 100')
end
if Nota == 4
display('Parabéns')
end
if Nota == 5
display('Sensacional!!!')
end
```

Clique em Save, nomeie e tente executar o script. Ao tentar executá-lo pela primeira vez, clique em Change Folder ou Add to Path.

2) Clique em New Script no canto superior esquerdo para iniciar um novo script. Código:

```
clear, clc %Limpar a Command Window e o Workspace  
contador=10;  
  
for i = 1:10  
G(contador)=i^2;  
contador=contador - 1;  
end  
  
G  
  
G =  
  
100      81      64      49      36      25      16      9      4  
1
```

Clique em Save, nomeie e tente executar o script. Ao tentar executá-lo pela primeira vez, clique em Change Folder ou Add to Path.

3) Clique em New Script no canto superior esquerdo para iniciar um novo script. Código:

```
clear, clc %Limpar a Command Window e o Workspace  
contador=1;  
  
for i = [3, 3, 1, 2, 3]  
H(contador)=i^3;  
contador=contador + 1;  
end  
  
H  
  
H =  
  
27      27      1      8      27
```

Clique em Save, nomeie e tente executar o script. Ao tentar executá-lo pela primeira vez, clique em Change Folder ou Add to Path.

4) Clique em New Script no canto superior esquerdo para iniciar um novo script. Código:

```
function c = ex1log(x,y)  
c = log(x)/log(y);  
end
```

Clique em Save, nomeie como ex1log.m e tente executar o script. Ao tentar executá-lo pela primeira vez, clique em Change Folder ou Add to Path.

5) A função precisa ser modificada apenas em um detalhe: a divisão necessita ser ponto a ponto:

```
function c = ex1log(x,y)
c = log(x)./log(y);
end
```

6) Clique em New Script no canto superior esquerdo para iniciar um novo script. Código:

```
function ordenado = ex1ord(x)
for i = 1:size(x,2)
    for j = (i+1):size(x,2)
        if x(i) > x(j)
            aux = x(i);
            x(i) = x(j);
            x(j) = aux;
        end
    end
end
ordenado = x;
end
```

Clique em Save, nomeie como ex1ord.m e tente executar o script. Ao tentar executá-lo pela primeira vez, clique em Change Folder ou Add to Path.

7) Clique em New Script no canto superior esquerdo para iniciar um novo script. Código:

```
function confere = ex1cpf(x)
aux=1e10;

for i = 1:9 % Extrair os 9 dígitos em forma de arranjo
    dig(i)= floor( x/aux );
    x = x - dig(i)*aux;
    aux = aux/10;

end % Aqui, x vale exatamente os dígitos verificadores
% Início do cálculo dos dígitos verificadores

verifica = 0; s
resto = 0;
aux=10;
for i = 1:9
    resto = resto + aux*dig(i);
    aux = aux - 1;
```

```
end
resto = rem(resto, 11);

if resto == 0 | resto == 1
    verifica = 0;
else
    verifica = (11 - resto);
end
%Já calculado o primeiro verificador, inciando o cálculo do
segundo

resto = 0;
aux=11;

for i = 1:9
    resto = resto + aux*dig(i);
    aux = aux - 1;
end
resto = resto + aux*verifica;

resto = rem(resto, 11);
verifica = verifica*10;

if resto == 0 | resto == 1
else
    verifica = verifica + (11 - resto);
end
if verifica == x % verificando e atribuindo 1 ou 0
    confere = 1;
else
    confere = 0;
end
end
```

Clique em Save, nomeie como ex1cpf.m e tente executar o script. Ao tentar executá-lo pela primeira vez, clique em Change Folder ou Add to Path.

8) Podemos fazer o script utilizando a função criada na questão anterior, ou até mesmo no mesmo script. Código:

```
x = input ('Insira o cpf');

if ex1cpf(x)
display('CPF correto')

else
display('CPF incorreto')
end
```

Clique em Save, nomeie e tente executar o script. Ao tentar executá-lo pela primeira vez, clique em Change Folder ou Add to Path.

9) Clique em New Script no canto superior esquerdo para iniciar um novo script. Código:

```
N=input('Quantos alunos são?\n');
media=0;
for i = 1:N
    media = media + input('nota:\n');
end
media = media/N;
display(['A nota média é ', num2str(media)])
```

Clique em Save, nomeie e tente executar o script. Ao tentar executá-lo pela primeira vez, clique em Change Folder ou Add to Path.

10) Clique em New Script no canto superior esquerdo para iniciar um novo script. Código:

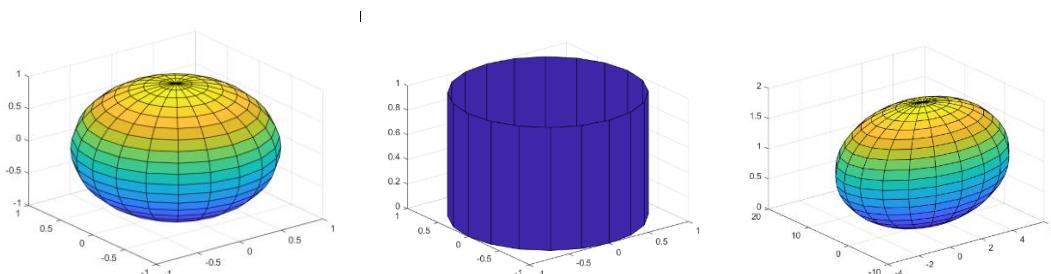
```
x=input('Insira o vetor x\n');
y=input('Insira o vetor y\n');
opera=input('O que deseja?(1 ou 2)\n');
if opera == 1
x*y'
end
if opera == 2
x'*y
end
```

Clique em Save, nomeie e tente executar o script. Ao tentar executá-lo pela primeira vez, clique em Change Folder ou Add to Path.

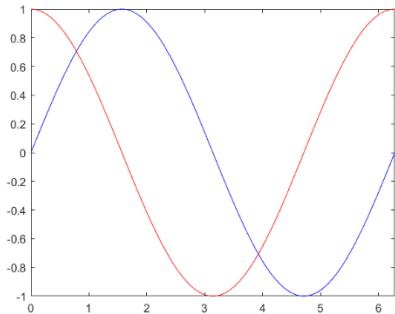
7. Gráficos

1) a)b)c)

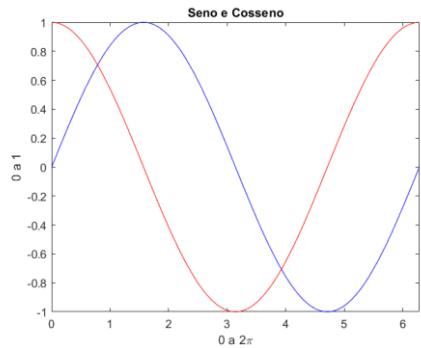
```
>> figure; sphere; figure; cylinder; figure;
ellipsoid(1,1,1,5,10,1);
```



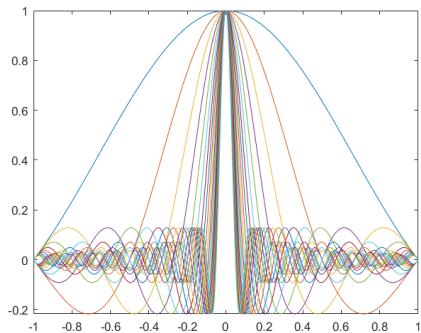
2) >> fplot('sin', [0 2*pi], 'b'); hold on; fplot('cos', [0 2*pi], 'r')



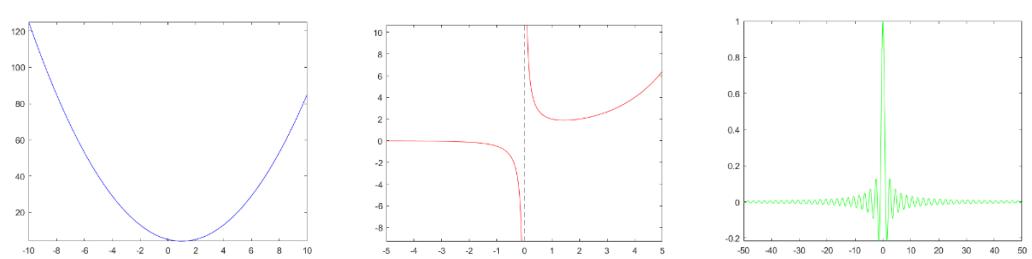
3) >> title('Seno e Cosseno'); xlabel('0 a 2\pi'); ylabel('0 a 1');



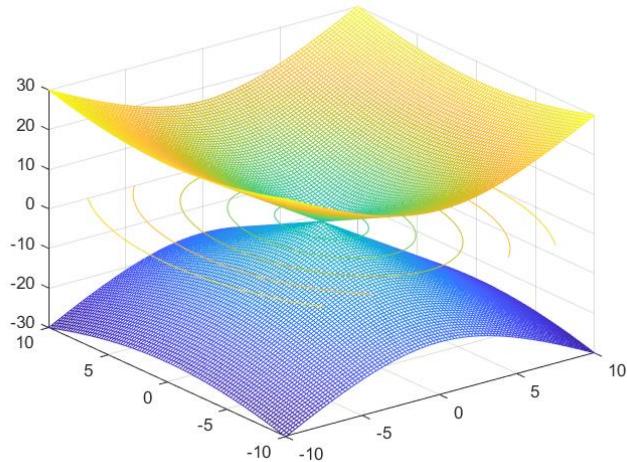
4) >> k = 1:20; k_str = num2str(k); f = ['sinc([', k_str, ']*x)']; fplot(f, [-1 1])



5) >> syms x; y1 = x^2 - 2*x + 5; y2 = 2^x/x; y3 = sinc(x);
>> figure; fplot(y1, [-10 10], 'b'); figure; fplot(y2, [-5 5], 'r'); figure; fplot(y3, [-50 50], 'g');

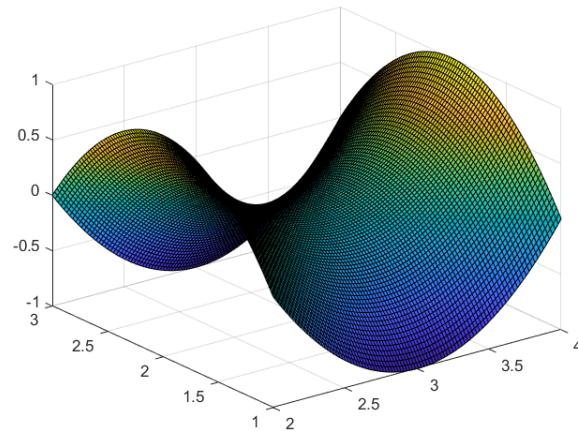


```
6) >> x=linspace(-10, 10, 100); [x, y]=meshgrid(x);
>> z=5*sqrt(x.^2/4 + y.^2/9);
>> mesh(x, y, z); hold on; mesh(x, y, -z)
>> contour(x, y, z)
```



7)

```
>> x = linspace(2,4); y = linspace(1,3);
>> [X, Y] = meshgrid(x, y);
>> Z = (X-3).^2 - (Y-2).^2;
>> surf(X, Y, Z)
```



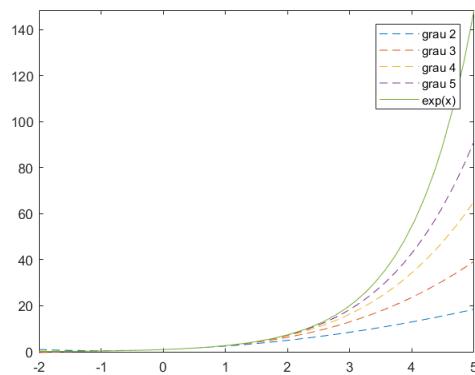
8)

```
>> syms x;
>> s2 = 1 + x + x^2/factorial(2);
>> s3 = s2 + x^3/factorial(3);
```

```

>> s4 = s3 + x^4/factorial(4);
>> s5 = s4 + x^5/factorial(5);
>> fplot(s2, [-2, 5], '--'); hold on;
>> fplot(s3, [-2, 5], '--'); fplot(s4, [-2, 5], '--');
fplot(s5, [-2, 5], '--');
>> fplot(exp(x), [-2, 5]);
>> legend('grau 2', 'grau 3', 'grau 4', 'grau 5',
'exp(x)');

```

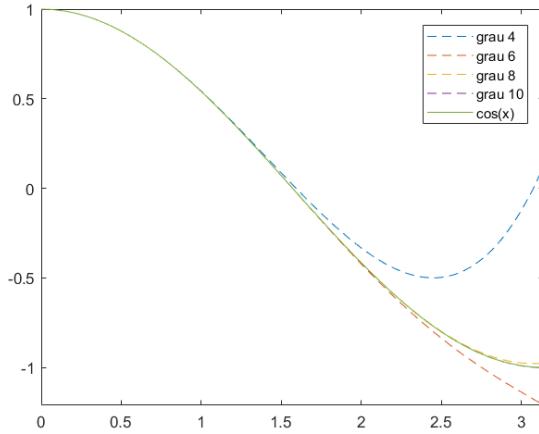


9)

```

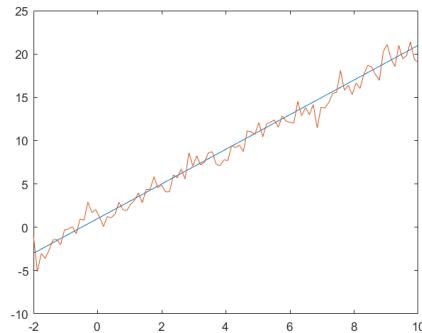
>> syms x;
s2= 1 - x^2/factorial(2) + x^4/factorial(4);
s3 = s2 - x^6/ factorial(6);
s4 = s3 + x^8/factorial(8);
s5 = s4 - x^10/factorial(10);
>> fplot(s2, [0, pi], '--'); hold on;
fplot(s3, [0, pi], '--'); fplot(s4, [0, pi], '--');
fplot(s5, [0, pi], '--');
>> fplot(cos(x), [0, pi])
>> legend('grau 4', 'grau 6', 'grau 8', 'grau 10',
'cos(x)')

```



10) Sabendo que a função randn retorna um arranjo com valores em distribuição normal padrão, iremos soma-la a função y para adquirir os ruídos.

```
>>x=linspace(-2, 10);
>> y=2*x + 1;
>> y2 = y+rand(1, 100); %ruídos
>> plot(x, y, x, y2);
```



8. Polinômios

1)

```
>> P = [1 -15 85 -225 274 120]; P = [1 -15 85 -225 274 -
120]; P1 = [-120 274 -225 85 -15 1]; P2 = [-1 -15 -85 -225
-274 -120]; P3 = [120 274 225 85 15 1];
>> roots(P)
ans =
5.0000
4.0000
3.0000
2.0000
```

```
1.0000
>> roots(P1)
ans =
1.0000
0.5000
0.3333
0.2500
0.2000
>> roots(P2)
ans =
-5.0000
-4.0000
-3.0000
-2.0000
-1.0000
>> roots(P3)
ans =
-1.0000
-0.5000
-0.3333
-0.2500
-0.2000
```

Podemos observar que as raízes de P_1 são o inverso das raízes de P , enquanto que as de P_2 são o oposto, e as de P_3 o inverso e oposto.

2) a)

```
>> poly([1 2; 2 1])
ans =
1      -2      -3
>> roots(ans)
ans =
3.0000
-1.0000
```

b)

```
>> poly([1 2 3; 4 5 6; 7 8 9])  
ans =  
1.0000 -15.0000 -18.0000 -0.0000
```

```
>> roots(ans)  
ans =  
16.1168  
-1.1168  
-0.0000
```

c)

```
> poly([0 0; 1 1])  
ans =  
1 -1 0  
>> roots(ans)  
ans =  
0  
1
```

d)

```
>> poly([98 88 2.04; 0.13 -12 9.08; 11.7 11.7 2.22])  
ans =  
1.0e+03 *  
0.0010 -0.0882 -1.1266 3.4090  
>> roots(ans)  
ans =  
99.2277  
-13.5442  
2.5365
```

3) Como x e y formam um conjunto que seis pontos, o polinômio interpolador via ajuste polinomial será de grau 5.

a)b)c)

```
>> p5 = polyfit(x, y, 5), p1 = polyfit(x, y, 1), p2 =  
polyfit(x, y, 2)  
p5 =
```

```

-1.5000    -16.9583     14.3000     -0.2854     -0.1803
15.0317
p1 =
1.0830    14.9382
p2 =
2.5679    -0.2009    15.0238

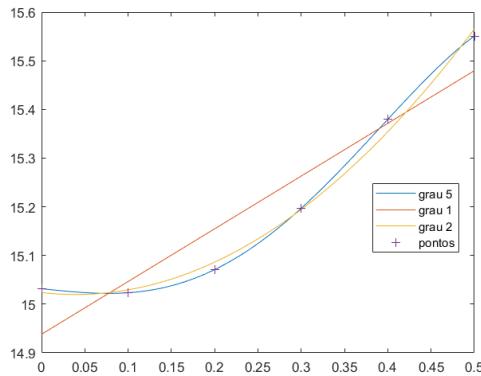
```

4)

```

>> H = linspace(0, 0.5);
>> y5 = polyval(p5, H); y1 = polyval(p1, H); y2 =
polyval(p2, H);
>> plot(H, y5, H, y1, H, y2, x, y, '+')
>> legend('grau 5', 'grau 1', 'grau 2', 'pontos')

```



5) a)b)c)d)e)f)g)

```

>> a = [1 5 -5 0 10]; b = [1 0 0 0 0 0]; c = conv([1 -1], [1
-3]);
>> deconv(b, a)
ans =
1      -5
>> conv(a, c)
ans =
1      1    -22     35     -5    -40     30
>> 10*b
ans =
10      0      0      0      0      0
>> c1 = [0 0 c]; %ajustando a dimensão de c para a subtração
>> roots(5*a-c1)

```

```
ans =
-5.8635 + 0.0000i
0.9180 + 0.8977i
0.9180 - 0.8977i
-0.9725 + 0.0000i
```

6) Primeiro precisamos escrever os polinômios P e Q, para Q, vamos usar a função conv:

```
>> P = [1 1];
>> Q = conv([1 0], [1 2]); Q = conv(Q, [1 4]);
>> [r p k] = residue(P, Q)
r =
-0.3750
0.2500
0.1250
p =
-4
-2
0
k =
[]
```

Assim, obtemos as frações parciais:

$$f(x) = -\frac{3}{8(x+4)} + \frac{1}{4(x+2)} + \frac{1}{8x}$$

7) Agora, só usar a função utilizando os parâmetro r, p e k. No lugar de k, podemos também usar 0, uma vez que não há termo direto:

```
>> [A B] = residue(r, p, k)
```

```
A =
0      1      1
```

```
B =
1      6      8      0
```

8) Basta usarmos os vetores x e y para um ajuste polinomial de grau 1. Deve ser usado grau 1, pois a força elástica é linear em x. Feito isso, teremos a aproximação linear do gráfico de deformação (m) x massa (kg). O segundo

termo é referente à deformação da mola quando a massa sob ela é 0, e portanto, apenas o primeiro termo nos interessa. Por fim, basta fazermos gravidade/(primeiro termo).

(I) $P = aX + b$, expressão polinomial de grau 1

(II) $kx = mg$, x = deformação, m = massa

$$x = \frac{g}{k} m$$

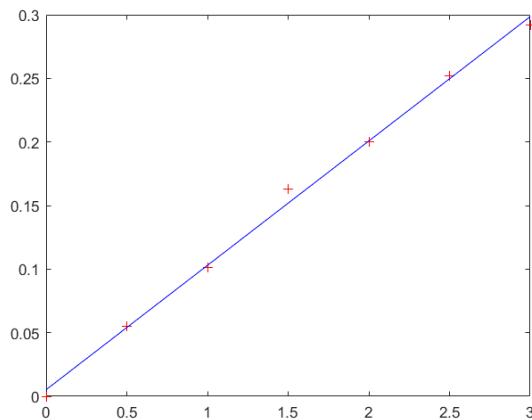
$$\frac{g}{k} = a$$

$$k = \frac{g}{a}$$

```
>> x = 0:0.5:3; y = [0      0.0550      0.1010      0.1630
0.2000    0.2520    0.2920];
>> a = polyfit(x, y, 1);
>> k = 9.8/a(1)
k =
100.2191
```

9)

```
>> y2 = polyval(a, x);
>> plot(x, y, 'r+', x, y2, 'b')
```



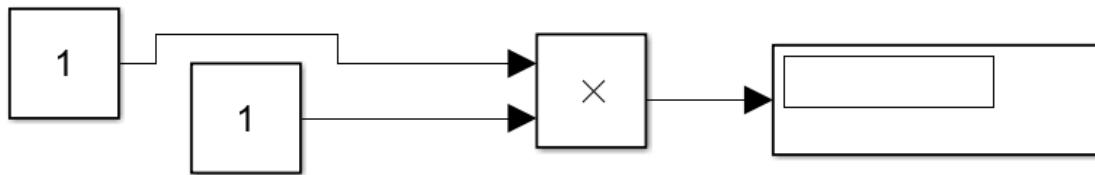
10)

```
>> F = [1 -3 0 4 0];
>> roots(F)
ans =
0
2.0000
```

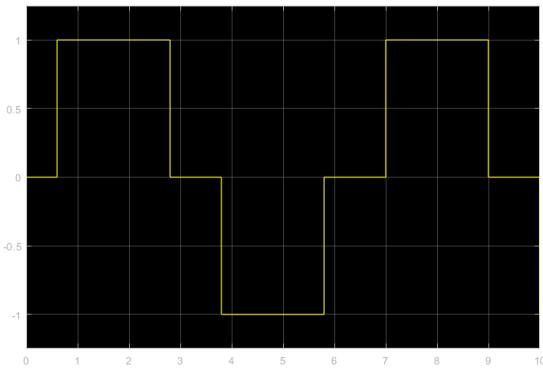
2.0000
-1.0000

9. Simulink: Introdução

1) Basta adicionarmos duas constantes, um multiplicador e um display:

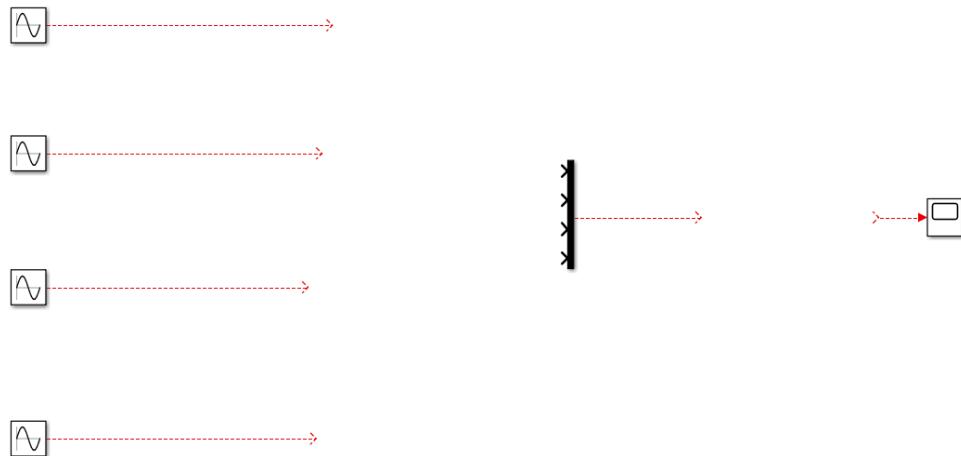


2) Para o sinal senoidal iremos usar “Sine Wave”. Para a multiplicação e divisão, iremos usar os blocos “Gain”. Para arredondar iremos usar o bloco que realiza a função com o mesmo nome no MATLAB: “Round”. Montaremos os blocos assim:



3)a)b)c)d)

Para esse sistema, utilizaremos quarto blocos “Sine Wave”, um bloco “Mux” para juntarmos os valores e um bloco “Scope”, para coletarmos os dados e vermos as curvas devidamente.



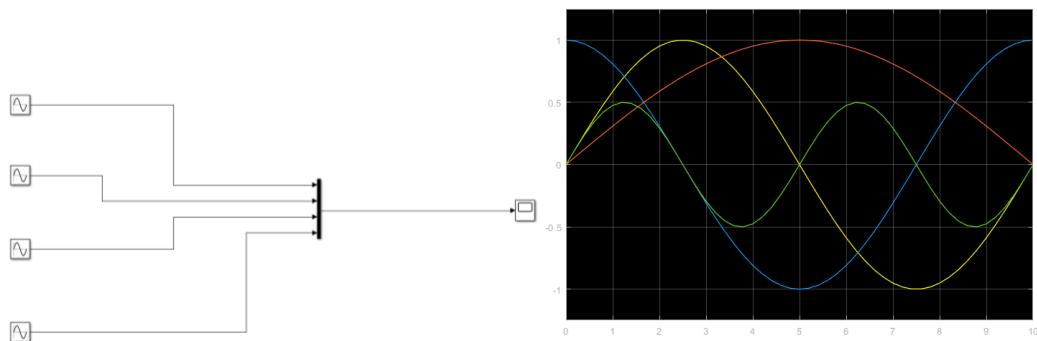
Devemos modificar os parâmetros das fontes senoidais para podermos obtermos os resultados desejados. O primeiro bloco será a função padrão e a referência para as outras três. Vamos utilizar, arbitrariamente, o tempo de execução de 10 segundos.

O bloco senoidal emite valores da seguinte forma:

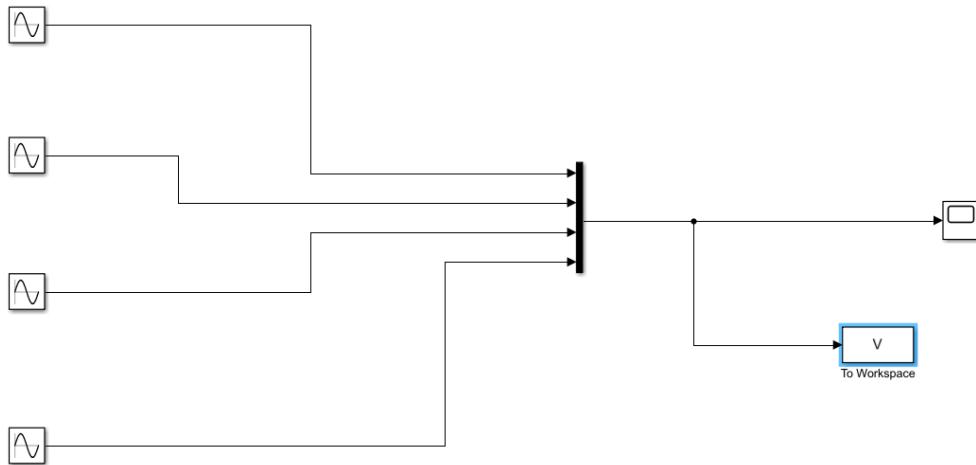
$$O(t) = Amp * \sin(Freq * t + Phase) + Bias$$

Com isso, para obtermos, do primeiro bloco, a função seno padrão de 0 a 2π , devemos ajustar a frequência para “ $0.2*\pi$ ”, mantendo os outros parâmetros. Para o segundo, devemos colocar o mesmo valor de frequência que o primeiro, e mudar a “Phase” para “ $\pi/2$ ”. Para o terceiro, a frequência deve ser metade da do primeiro, logo “ $0.1*\pi$ ”. Para o último, a frequência deve ser o dobro da do primeiro ($0.4*\pi$), e a amplitude a metade (0.5).

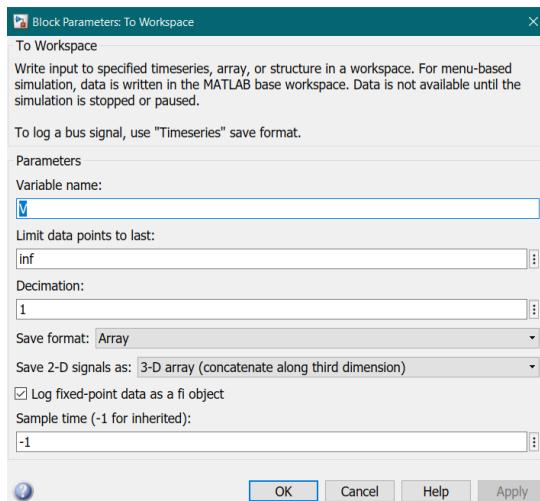
Após ligarmos nosso sistema e coletarmos os dados:



4) Vamos adicionar como saída o bloco “To Workspace”

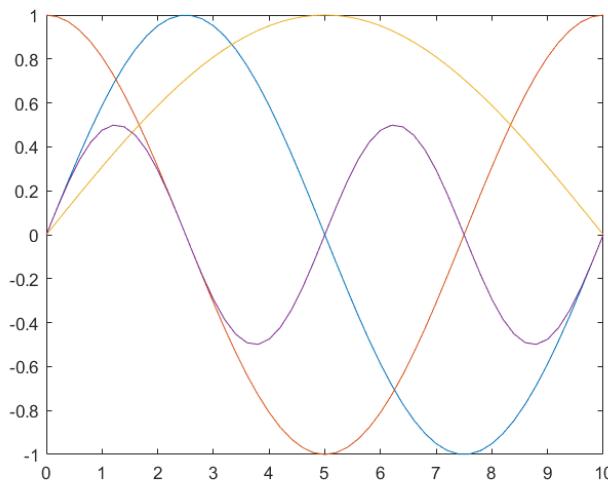


Mudamos o nome da variável para V e o tipo de saída para arranjo:

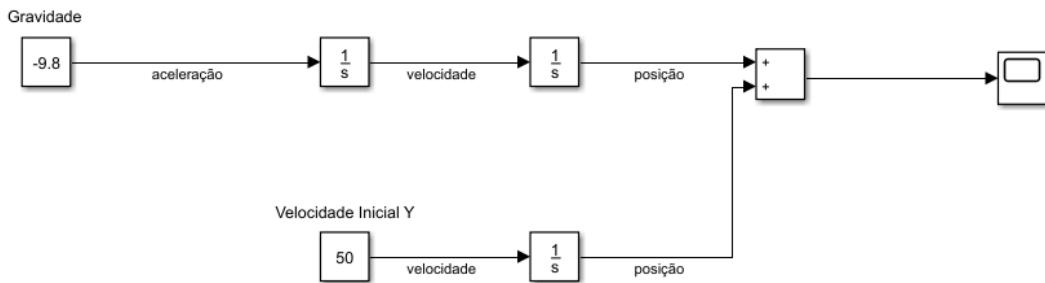


5) Observe que também foi exportado a variável tout, referente a sequência dos tempos de saída. Agora, para plotar, basta fazermos:

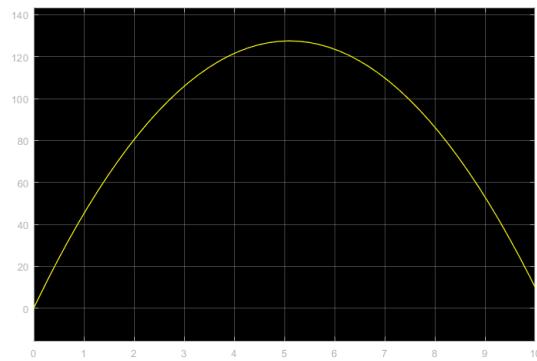
```
>>plot(tout, V(:,1), tout, V(:,2), tout, V(:,3), tout, V(:,4))
```



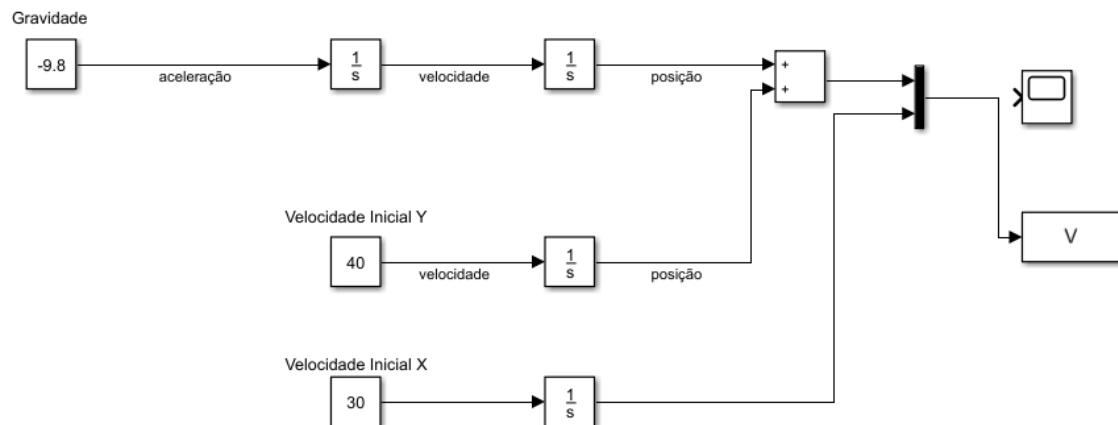
6) Utilizaremos dois blocos “Constant”, três blocos “Integrator”, um bloco “Add” e um bloco “Scope”.



Como a aceleração é a derivada da velocidade, e a velocidade a derivada da posição, podemos integrar a gravidade duas vezes, e a velocidade inicial uma. Somando, teremos a variação de y com o tempo.

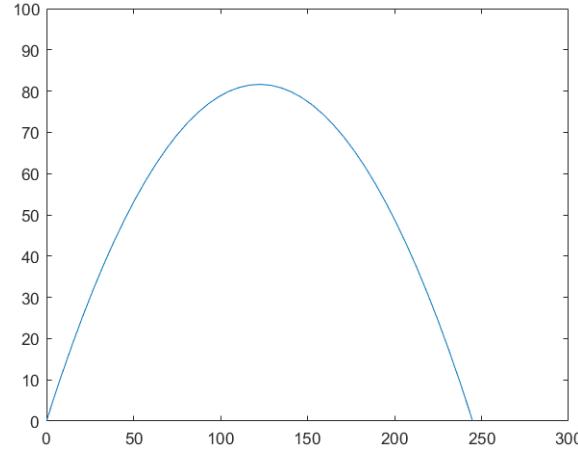


7) Vamos adicionar um “Mux” e usar para a parte do $x(t)$, uma parte igual a da velocidade inicial de Y. Os ados serão transmitidos para um bloco “To Workspace” Após ajustar o valor da velocidade inicial de X e Y, e configurar os parâmetros do bloco que enviará os dados para a área de trabalho para arranjo, basta plotarmos e configurarmos o eixo:

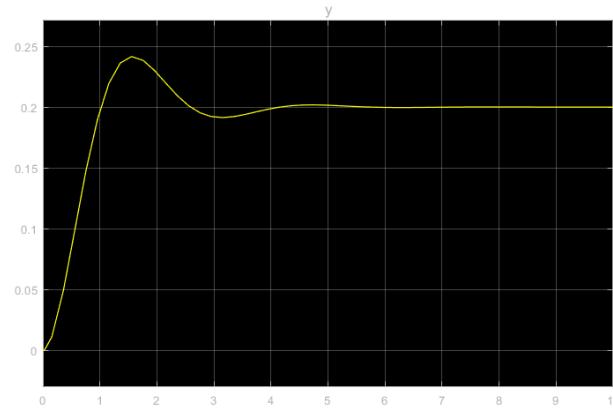
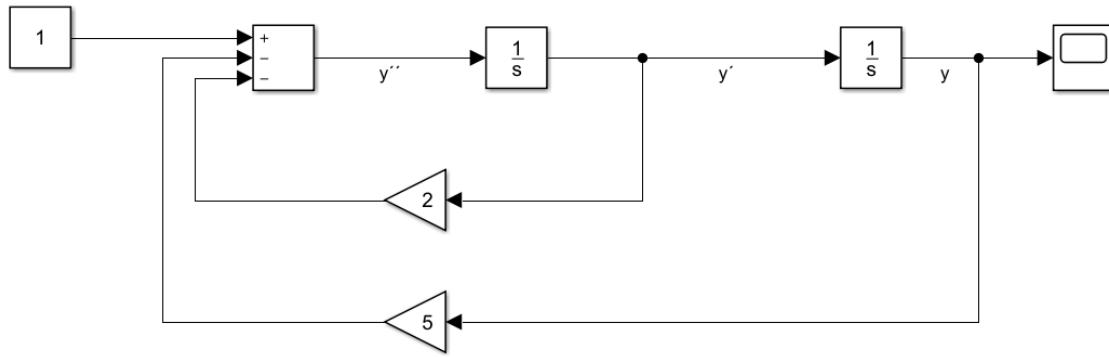


```
>> plot(V(:,2), V(:,1))
```

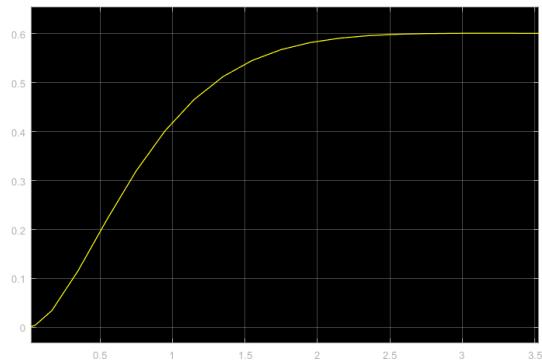
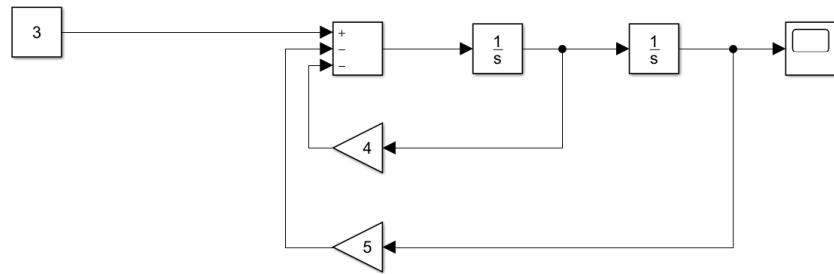
```
>> axis([0 300 0 100])
```



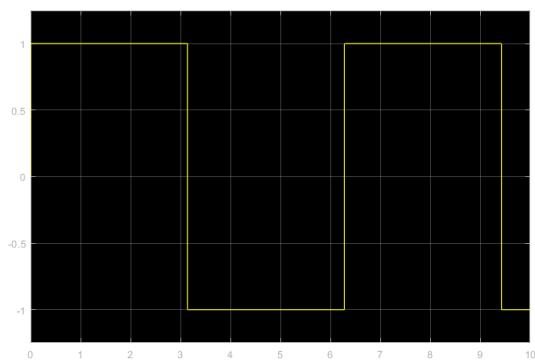
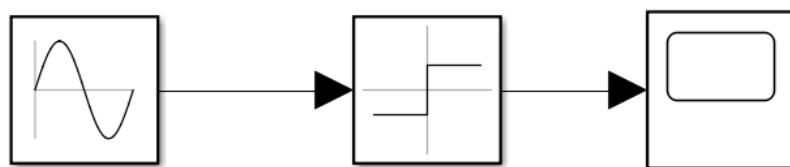
8) Primeiro isolamos a segunda derivada e modelamos o sistema da seguinte forma:



9) Primeiro isolamos a segunda derivada e modelamos o sistema da seguinte forma:



10) Basta montarmos o seguinte sistema:



Bibliografia

- ❖ <https://www.mathworks.com>
- ❖ Reginaldo de Jesus Santos, "Introdução ao Matlab," Departamento de Matemática, ICEX, UFMG <http://www.mat.ufmg.br/~regi>
- ❖ Frederico Ferreira Campos Filho, "Apostila de Matlab," Departamento de Ciência da Computação, ICEX, UFMG
- ❖ Frederico F. Campos, Introdução ao MATLAB
- ❖ Grupo PET, "Curso de MATLAB," Engenharia Elétrica - UFMS <http://www.del.ufms.br/tutoriais/matlab/apresentacao.htm>
- ❖ B.P. Lathi, Sinais e Sistemas lineares
- ❖ Pedro Henrique Galdino Silva, Minicurso de MATLAB - Consultoria e Projetos Elétricos Jr.
- ❖ Introdução ao Simulink, Apostila adaptada de MANUAL DE INTRODUÇÃO AO MATLAB/SIMULINK, prof. Luis Filipe Baptista

Apostila

Minicurso de MATLAB 1.0





U F *m* G



PROGRAMA DE EDUCAÇÃO TUTORIAL – ENGENHARIA ELÉTRICA – UNIVERSIDADE
FEDERAL DE MINAS GERAIS