Universidade Federal de Santa Maria
Centro de Tecnologia
Engenharia Aeroespacial

Apostila de MATLAB

Kátia Maier dos Santos
Promovido por Grupo de Sistemas Aeroespaciais e Controle (GSAC)
Apoio: Curso de Engenharia Aeroespacial
Recursos do programa FIEn da PROGRAD/UFSM

Santa Maria, 2017

Sumário

1 Introdução

1.1 MATLAB

MATLAB tem seu nome derivado de *Matrix Laboratory*, que quer dizer Laboratório de Matrizes. Foi desenvolvido para facilitar a programação matemática com acesso direto a algoritmos envolvendo matrizes e álgebra linear. Mas, com o tempo, tornou-se um dos principais softwares para modelagem e simulação de sistemas dinâmicos em Engenharia e Ciências Exatas.

O MATLAB possui mais de 1.000 funções, além de ferramentas e bibliotecas adicionais (as *Toolboxes*) que ampliam os recursos com muitas outras funções mais específicas. Uma das vantagens do uso desse software é a facilidade na utilização. Seu ambiente de trabalho não apresenta complicações, pois os problemas são escritos em linguagem matemática.

2 Funções e Comandos Básicos

Janela de Comando (*Command Window*): Janela onde os comandos para o programa serão feitos. São realizados instantaneamente. As operações feitas na janela de comando não são salvas. Para isso, usa-se o *script*.

Área de Trabalho (*Workspace*): Local em que as variáveis salvas aparecem. O MATLAB salva as variáveis escritas na janela de comando automaticamente e seu respectivo valor e classe.

Pasta Atual (*Current Folder*): Mostra a pasta que está sendo acessada no momento.

Comando *clc* limpa a janela de comando.

Os programas salvos no MATLAB possuem a extensão ".m".

2.1 Linha de comando

A linha de comando é a linha na qual as operações são realizadas. Está localizada na janela de comando e é iniciada pelo sinal >>. Alguns comandos básicos podem ser visualizados na tabela 1.

Tabela 1 – Comandos básicos

Operação	Símbolo
Adição	+
Subtração	−
Multiplicação	*
Divisão	/
Potenciação	^

As operações feitas seguem a ordem: Potenciação, Multiplicação e Divisão, Adição e Subtração.

2.2 Matrizes e Vetores

Todas as variáveis do MATLAB são vetores multidimensionais, não importa qual classe de dados. Uma matriz, por exemplo, é um vetor de duas dimensões.

Para criar um vetor com os elementos em uma linha, separa-se os elementos com uma vírgula (,) ou simplesmente um espaço.

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

Para criar uma matriz, separa-se as linhas com ponto e virgula (;).

$$A = [\, 1\ 2\ ;\ 3\ 4\,] \tag{2.1}$$

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

Note que o MATLAB reconhece o comando por linhas, ou seja, os números antes do ponto e vírgula pertecem à primeira linha e os números após o ponto e vírgula pertecem à segunda linha e assim por diante.

Outra maneira de criar matrizes e vetores é usando os comandos `ones` (cria um vetor unitário), `zeros` (cria um vetor nulo), `rand` (cria números aleatórios entre 0 e 1) ou `randi` (cria números inteiros aleatórios). Quando essas funções são usadas, o primeiro número é o número de linhas e o segundo é o número de colunas.

```
>> A = ones(2, 1)
A =
    1
    1

>> A = zeros(2, 2)
A =
    0   0
    0   0

>> A = rand(3, 3)
A =
    0.9058   0.6324   0.5469
    0.1270   0.0975   0.9575
    0.9134   0.2785   0.9649
```

2.2.1 Operações com Matrizes e Vetores

O MATLAB permite uma série de operações com matrizes, como nos exemplos abaixo.

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

>> a +  10
a =
    11   12   13
    14   15   16
    17   18   19
```

Para calcular a matriz transposta utiliza-se o operador apóstrofe (').

```
>> a'
ans =
       1   4  7
       2   5  8
       3   6  9
```

Também é possível realizar outras operações, como achar a matriz inversa A^{-1} e calcular o produto entre matrizes $A * B$.

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

>> inv(A)
ans =
    -2.0000   1.0000
     1.5000  -0.5000
```

A notação para calcular multiplicação de matrizes é a mesma para escalares.

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

>> B = [5 6; 7 8]
B =
    5   6
    7   8

>> A*B
ans =
    19   22
    43   50
```

Para multiplicar elemento a elemento, usa-se o operador ".∗".

```
>> A.*B
ans =
    5   12
    21   32
```

Percebe-se a diferença entre os resultados. O ponto "." antes do símbolo da operação matemática também serve para a potenciação e divisão.

A concatenação de vetores é o processo de juntar dois vetores para formar um maior. O operador de concatenação são os colchetes [ ].

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

>> A = [a, a]
A =
    1   2   3   1   2   3
```

Como visto acima, o operador concatenação apenas juntou os dois vetores um após o outro. O processo é igual para concatenar matrizes.

Outro comando muito útil é o comando `max`. Ele retorna o valor máximo dentro de um vetor ou matriz e pode ser usado de várias maneiras.

Usando apenas `max(A)`, retorna-se o maior valor dentro do arranjo A. Já `max(A, B)` retorna um vetor com os valores mais altos de A e B. A

```
>> A = [1 8 4]
A =
    1   8   4

>> B = [5 2 9]
B =
    5  2   9

>> max(A)
ans =
    8

>> max(A, B)
ans =
    5   8   9
```

O MATLAB possui uma série de comando matriciais que facilitam as operações. Os mais comuns estão listados na Tab. 2

Tabela 2 – Funções matriciais

Comando	Descrição
`eig`	Autovalores e autovetores
`inv`	Inversa
`expm`	Matriz exponencial
`det`	Determinante
`size`	Tamanho da matriz
`lu`	Fatorização triangular LU
`sqrtm`	Matriz raiz quadrada
`poly`	Polinômio característico
`diag(A)`	Elementos da diagonal principal
`[V D] = eig(A)`	V: autovetores, D: autovalores

Operações vetoriais são úteis para construção de tabelas, como pode ser visto no exemplo a seguir.

```
>> n = (0:3)'
n =
    0
    1
    2
    3

>> p = [n n.^2 n.^3]
p =
    0   0   0
    1   1   1
    2   4   8
    3   9   27
```

2.3 Variáveis

Caso seja feita uma operação na linha de comando que não tenha sido atribuída a uma variável, o MATLAB retorna o valor na variável `ans`, de "resposta"e a salva com esse nome na área de trabalho. Caso seja atribuída uma variável para a expressão, ela aparecerá com o nome dado na área de trabalho.

É possível declarar uma variável como simbólica, ou seja, não possui valor definido. Para isso escreve-se `syms`, como pode ser visto no exemplo abaixo.

```
>> syms x
>> y = [1 2 3]
y =
    1   2   3

>> z = x + 2*y
z =
    [x + 2, x + 4, x + 6]
```

Como visto acima, pode-se realizar operações normalmente com as variáveis simbólicas.

O MATLAB guarda os números como valores `float`, mostrando apenas quatro casas decimais. Caso haja necessidade alterar como o programa mostra a variável, usa-se o comando `format` antes da operação. A tabela e o exemplo abaixo exemplificam os os tipos de `format` que o MATLAB possui e como utilizar.

```
>> format long
>> 100/7

ans =
    14.285714285714286

>> format short
>> 100/7

ans =
    14.2857
```

Tabela 3 – Comando `format`

Comando	Descrição
`format short`	Mostra 4 casas após a vírgula
`format long`	Mostra 15 casas após a vírgula
`format short e`	Mostra 4 casas após a vírgula com exponencial
`format long e`	Mostra 15 casas após a vírgula com exponencial
`format short g`	Mostra 5 dígitos
`format long g`	Mostra 15 dígitos
`format hex`	Formato hexadecimal
`format bank`	Formato bancário
`format rat`	Formato racional

Esse comando altera apenas como o programa mostra os números, não como eles são salvos na memória.

2.4 Uso do `help` e `lookfor`

Todas as funções do MATLAB tem uma documentação que inclui explicação e exemplos. Esse comando se chama `help`. Existem algumas maneiras de usar esse comando.

- `doc` "função": abre uma janela separada com a documentação da função.
- `help` "função": mostra uma versão resumida da explicação.

O comando `lookfor` é útil caso não se saiba o tópico exato. Esse comando procura a palavra chave digitada em todos os tópicos de ajuda e retorna as informações.

3 Fundamentos de Linguagem

3.1 Texto e caracteres

É possível declarar uma variável com texto. Ela é declarada da mesma forma, apenas com o texto entre aspas simples, como pode ser visto abaixo.

```
>> a = 'Hello, World'
>> a =
        Hello, World
```

A variável `a` declarada no exemplo acima são como qualquer outra variável do MATLAB, porém do tipo **char** que é abreviação de `character`.

Usando o comando `whos`, é possível ver uma tabela com as propriedades do vetor ou matriz como tamanho, classe e quantidade de bytes.

É possível realizar operações com variáveis textuais, como concatenação e transformação números em caracteres. Para o primeiro caso, define-se uma nova variável, como pode ser visto a seguir.

```
>> a = 'Olá'
>> a =
        Olá

>> b = 'Tudo bem?'
>> b =
        Tudo bem?

>> c = [a, '! ', b]
c =
    Olá! Tudo bem?
```

Para converter valores numéricos em caracteres, usa-se as funções `num2str` ou `int2str`. Ela transforma números com exponenciação em números extensos. Um exemplo pode ser visto na figura abaixo.

```
>> a = [2e13 2^7]
a =
    1.0+13  *
    2.000   0.0000

>> num2str(a)
ans =
    20000000000000  128
```

A diferença é que a função `int2str` considera apenas números inteiros, caso haja um número não inteiro, ele é arredondado.

4 Álgebra Linear

Precisamente, uma matriz é um vetor bidimensional que representa uma transformação linear. As operações matemáticas realizadas em matrizes são conteúdos da álgebra linear.

É possível realizar uma série de operações no MATLAB com as matrizes sabendo o básico de álgebra linear. Por exemplo, somando uma matriz com a sua transposta produz uma matriz simétrica, como visto abaixo.

```
>> A = [16 3 2 13
        5 10 11 8
        9 6 7 12
        4 15 14 1]
A =
    16 3 2 13
    5 10 11 8
    9 6 7 12
    4 15 14 1

>> A + A'
ans =
    32  8   11  17
    8   20  17  23
    11  17  14  26
    17  23  26  2
```

Como já visto antes, o símbolo * significa multiplicação envolvendo os produtos entre linhas e colunas. Multiplicando a transposta de uma matriz por ela mesma também produz uma matriz simétrica.

```
>> A =
     16 3  2  13
     5  10 11 8
     9  6  7  12
     4  15 14 1

>> A'*A
ans =
    378 212 206 360
    212 370 368 206
    206 368 370 212
    360 206 212 378
```

Se o determinante de uma matriz é igual a zero, essa matriz é singular. O determinante é calculado com o comando `det(A)`. Se uma matriz for singular, ela não possui inversa, e caso o comando `inv(A)` for usado, aparecerá uma mensagem de erro.

Os autovalores da matriz são calculados com o comando `eig(A)`. Usando `[V, D] = eig(A)`, é calculado uma matriz diagonal D com os autovalores e uma matriz V onde as colunas são os autovetores correspondentes. Abaixo esses comandos estão exemplificados.

```
A =
    1   2
    2   1

>> eig(A)
ans =
    -1
    3

>> [V, D] = eig(A)
V =
    -0.7071 0.7071
    0.7071  0.7071

D =
    -1  0
    0   3
```

Com a função `poly(A)`, encontra-se os coeficientes do polinômio característico.

```
>> A
A =
    1   2
    2   1

>> poly(A)
ans =
    1   -2  -3
```

Assim, o polinômio característico é $1\lambda^2 - 2\lambda - 3$. Esses valores vem de `det(A -λI)`.

5 Funções

O MATLAB encontra as raízes das funções com o comando `roots`. A sua entrada é um vetor com os coeficientes polinomiais. A figura abaixo ilustra esse comando.

```
>> p = [3 4 1]
p =
    3   4   1

>> roots(p)
ans =
    -1.0000
    -0.3333
```

Onde `p` é o vetor com os coeficientes da equação $3x^2 + 4x + 1$. A resposta são os valores que zeram a equação.

O MATLAB também encontra os zeros de funções, ou seja, valores de x onde $y = 0$. Para isso usa-se o comando `fzero`. Esse comando é um pouco mais complicado, primeiro é preciso criar um `script` com a função desejada, como é mostrado a seguir.

```
function y = f(x)
y = x.^3 - 2*x - 5;
```

Após, é necessário criar um novo `script` chamando a função e utilizando o comando.

```
fun = @func;
z = fzero(fun, 2)
```

No qual `func` é como a função $y = f(x)$ foi salva, `fun` é a variável que está recebendo a função, e o valor 2 é um chute de valor inicial, ou seja, o MATLAB irá buscar o zero da função mais próximo desse valor.

A resposta dada pelo software pode ser vista abaixo.

```
z =
    2.0946
```

Também é possível encontrar máximos e mínimos das funções. Para isso, utiliza-se o comando `fminbnd`. Ele é mostrado abaixo.

```
>> xmin = fminbnd('sin(x)', 0, 2*pi)
xmin =
    4.7124

>> [xmin ymin] = fminbnd('sin(x)', 0, 2*pi)
xmin =
    4.7124

ymin =
   -1.0000
```

Caso o comando seja utilizado da primeira maneira, ele retorna apenas o valor mínimo de x, caso seja feito da segunda maneira, retorna tanto o valor mínimo de x como o de y. Os valores utilizados no comando, nesse caso 0 e $2 * pi$, são o intervalo da função.

6 Gráficos

6.1 Funções básicas

O MATLAB possui uma grande variedade de técnicas para mostrar dados graficamente. Existem ferramentas interativas que permitem manipular os gráficos. A Tabela 4 mostra os comandos mais usados para plotar gráficos.

Tabela 4 – Comandos para gráficos

Comando	Descrição	Exemplo
`plot`	Gráfico linear	`plot(x, y)`
`ezplot`	Gráfico com variáveis simbólicas	`ezplot(x, y)`
`loglog`	Gráfico em escala logarítmica	`loglog(x, y)`
`fill`	Desenha polígono 2D	`fill(x, y, 'r')`
`polar`	Gráfico em coordenadas polares	`polar(x, y)`
`bar`	Gráfico de barras	`bar(x, y)`
`stem`	Gráfico em tempo discreto	`tem(x, y)`
`stairs`	Gráfico em degrau	`stairs(x, y)`
`comet`	Gráfico com trajetória de cometa	`comet(x, y)`

Dentre esses, o mais usado para gráficos bidimensionais é o comando `plot`.

```
>> a = 0:pi/20:2*pi;
>> b = sin(a);
>> plot(a, b)
```

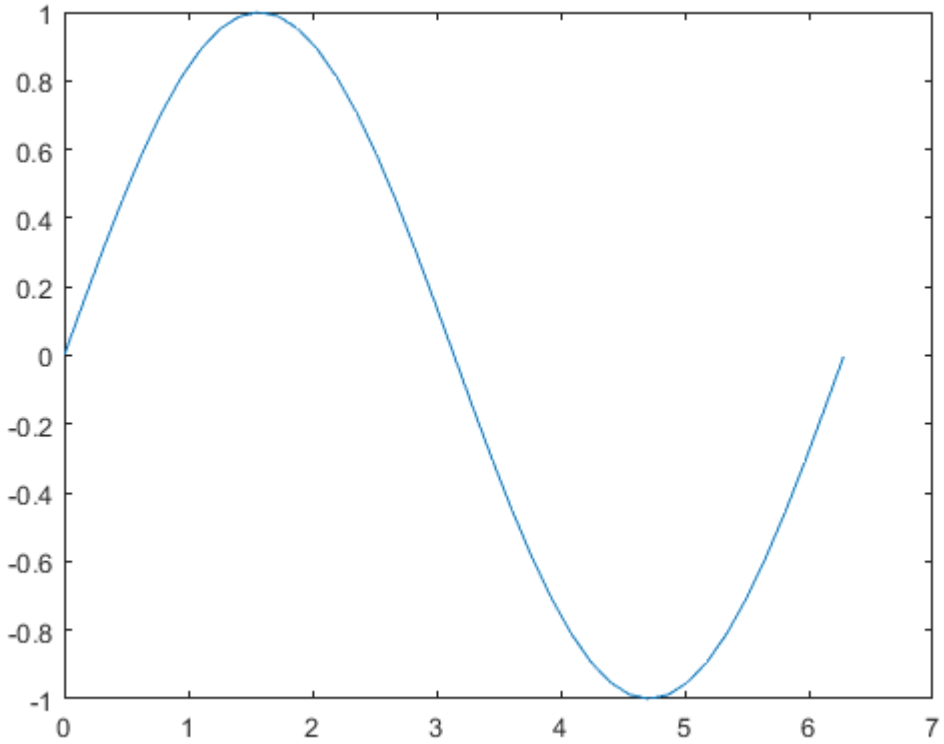


Figura 1 – Gráfico gerado com o comando `plot`.

As tabelas abaixo mostram alguns comandos de manipulação dos gráficos junto com exemplos. Com os comandos mais básicos é possível nomear e limitar os eixos, alterar a linha do gráfico e a cor.

Tabela 5 – Comandos para manipulação de gráficos

Comando	Descrição	Exemplo
`xlabel`	Nome do eixo x	`xlabel('Tempo (s)')`
`ylabel`	Nome do eixo y	`ylabel('Velocidade (m/s)')`
`title`	Título	`title('Gráfico 1')`
`legend`	Legenda	`legend('sin', 'cos')`
`xlim`	Limitar eixo x	`xlim([min max])`
`ylim`	Limitar eixo y	`ylim([min max])`
`grid on`	Grade	`grid on`
`grid minor`	Grade menor	`grid minor`
`FontSize`	Tamanho da letra	`xlabel('Tempo', 'FontSize', 5)`

Tabela 6 – Estilo de linha

Símbolo	Descrição	Exemplo
`-`	Linha sólida	`plot(x, '-')`
`--`	Linha tracejada	`plot(x, '- -')`
`:`	Linha pontilhada	`plot(x, ':')`
`-.`	Linha ponto-traço	`plot(x, '-.')`

Tabela 7 – Marcadores

Símbolo	Descrição	Exemplo
`*`	Linha com asteriscos	`plot(x, '*')`
`+`	Linha com +	`plot(x, '+')`
`o`	Linha com circunferência	`plot(x, 'o')`
`x`	Linha com x	`plot(x,'x')`
`s`	Linha com quadrado	`plot(x, 's')`
`v`	Linha com triângulo para baixo	`plot(x, 'v')`
$\wedge$	Linha com triângulo para cima	`plot(x,'^')`
$<$	Linha com seta	`plot(x, '<')`
$>$	Linha com seta	`plot(x, '>')`
`P`	Linha com estrela	`plot(x, 'P')`
`d`	Linha com diamante	`plot(x, 'd')`
`h`	Linha com hexágono	`plot(x, 'h')`
`MarkerSize`	Tamanho do marcador	`plot(x, '^', 'MarkerSize', 6)`

Tabela 8 – Cores

Símbolo	Descrição	Exemplo
`r`	Vermelho	`plot(x, 'r')`
`g`	Verde	`plot(x, 'g')`
`b`	Azul	`plot(x, 'b')`
`c`	Ciano	`plot(x, 'c')`
`w`	Branco	`plot(x, 'w')`
`k`	Preto	`plot(x, 'k')`
`m`	Magenta	`plot(x, 'm')`
`y`	Amarelo	`plot(x, 'y')`
`LineWidth`	Largura da linha	`plot(x, 'LineWidth', 0.5)`

Utilizando as Tabelas 6, 7 e 8, é possível juntá-las da seguinte maneira `plot(x, ':bs')` para um gráfico com linha pontilhada, azul e com quadrados. Um exemplo é mostrado abaixo.

```
>> a = 0:pi/20:2*pi;
>> b = sin(a);
>> plot(b,'cP:', 'MarkerSize', 10)
```

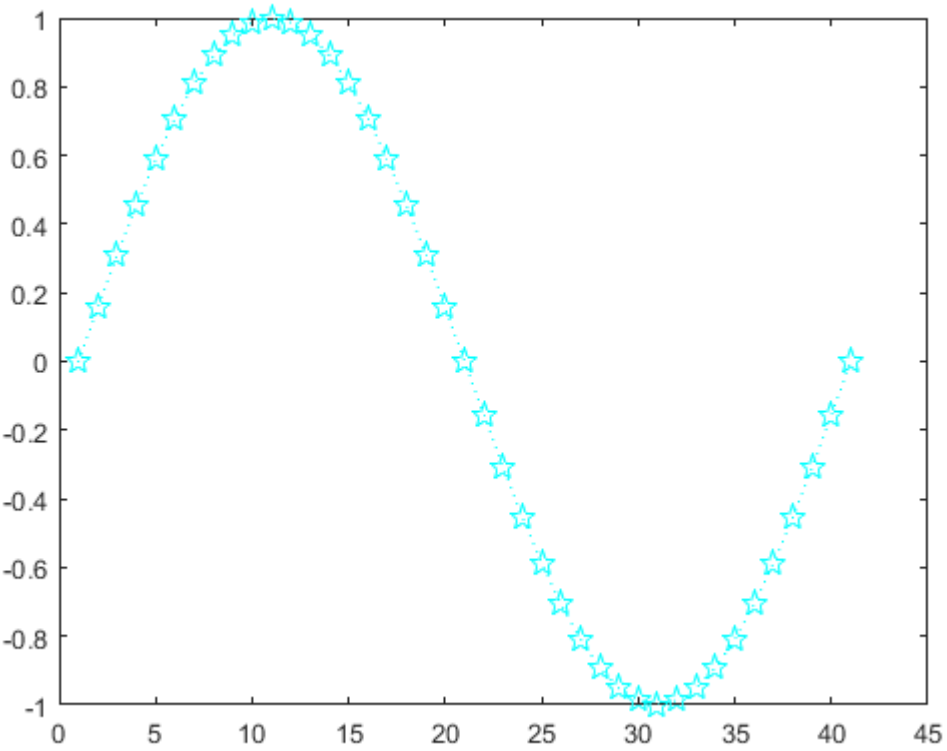


Figura 2 – Gráfico gerado usando comandos das tabelas.

Ainda, é possível escolher a posição da legenda no gráfico usando `legend('leg1', 'leg2', posição)`, na qual posição é um número entre -1 e 4. Seus significados podem ser vistos na tabela abaixo.

Tabela 9 – Comando `legend`

Número	Descrição
0	Escolha automática
-1	Direita
1	Canto superior direito
2	Canto superior esquerdo
3	Canto inferior esquerdo
4	Canto superior direito

Também é possível adicionar texto ao gráfico com os comandos `text(x, y, 'texto')` e `gtext('texto')`. Onde no comando `text` x e y são as coordenadas na qual deseja-se escrever o texto, enquanto com o comando `gtext` o texto pode ser movido com o mouse.

6.1.1 Comando `fill`

Esse comando permite preencher os gráficos bidimensionais. Um exemplo é mostrado abaixo.

```
>> a = 0:pi/20:2*pi;
>> b = sin(a);
>> fill(a, b,'m')
```

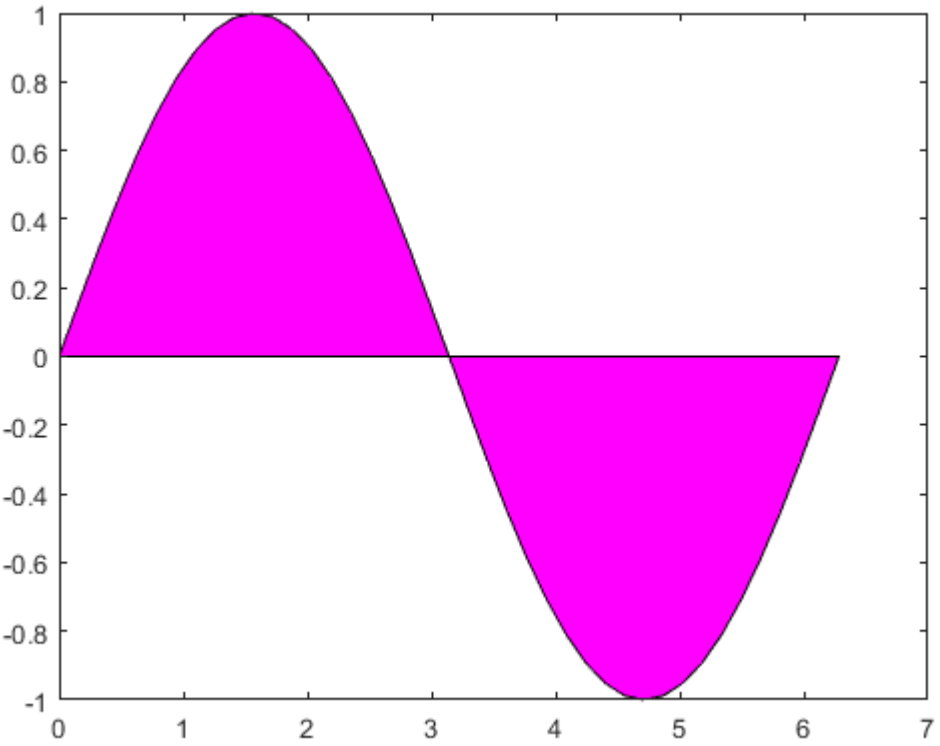


Figura 3 – Gráfico gerado com o comando `fill`.

6.1.2 Comando `axis`

Pode-se controlar os eixos horizontal e vertical dos gráficos com o comando `axis`. Eles podem ser vistos na tabela abaixo.

Tabela 10 – Comando `axis`

Comando	Descrição
`axis ([xmin xmax ymin ymax])`	Limita os eixos x e y
`axis image`	Ajusta os eixos para ficarem do tamanho do gráfico
`axis off`	Tira os eixos
`axis normal`	Deixa os eixos normais

6.1.3 Comando `hold on`

O comando `hold on` é usado quando deseja-se plotar mais de uma função na mesma figura. Esse exemplo pode ser visto na seguir.

```
a = 0:4*pi;
b = sin(a);
c = cos(a);

plot(a, b)
hold on
plot(a, c)
legend(' sin', 'cos')
```

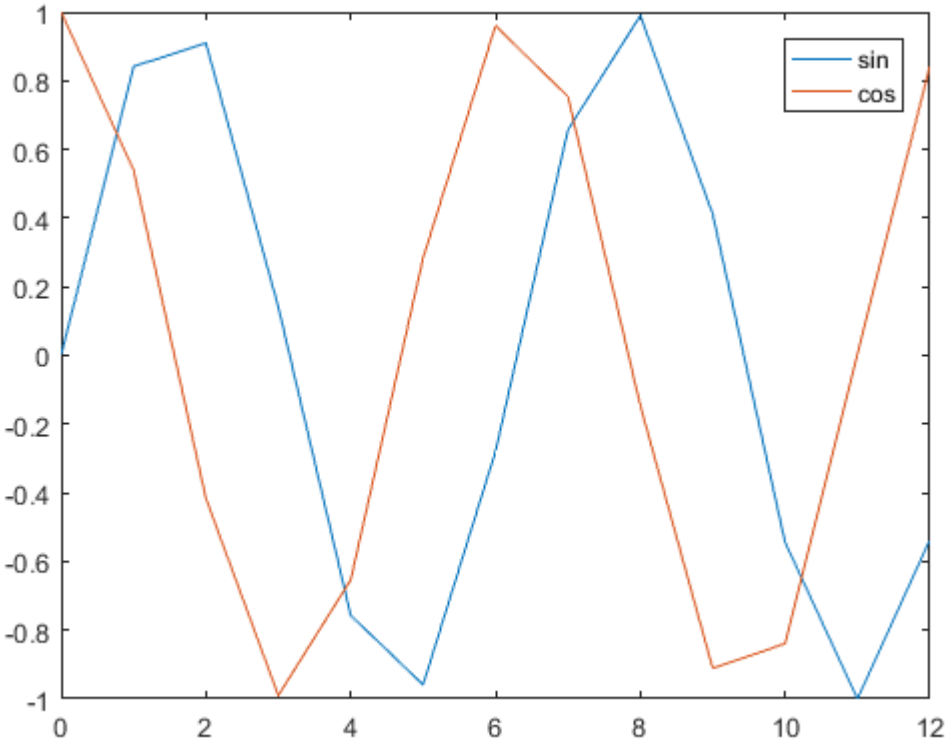


Figura 4 – Gráfico gerado com o comando `hold on`.

Esse comando também é muito usado quando a função varia de comportamento ao longo do tempo, por exemplo, quando existe uma descontinuidade. Da mesma forma, pode ser usada para plotar duas funções que não iniciam e terminam no mesmo tempo. Esse exemplo pode ser visto abaixo.

```
t1 = pi:pi/20:2*pi;
t2 = 0:pi/20:pi;

y1 = cos(t1);
y2 = sin(t2);

plot(t1, y1,'b')
hold on
plot(t2, y2,'r')
```

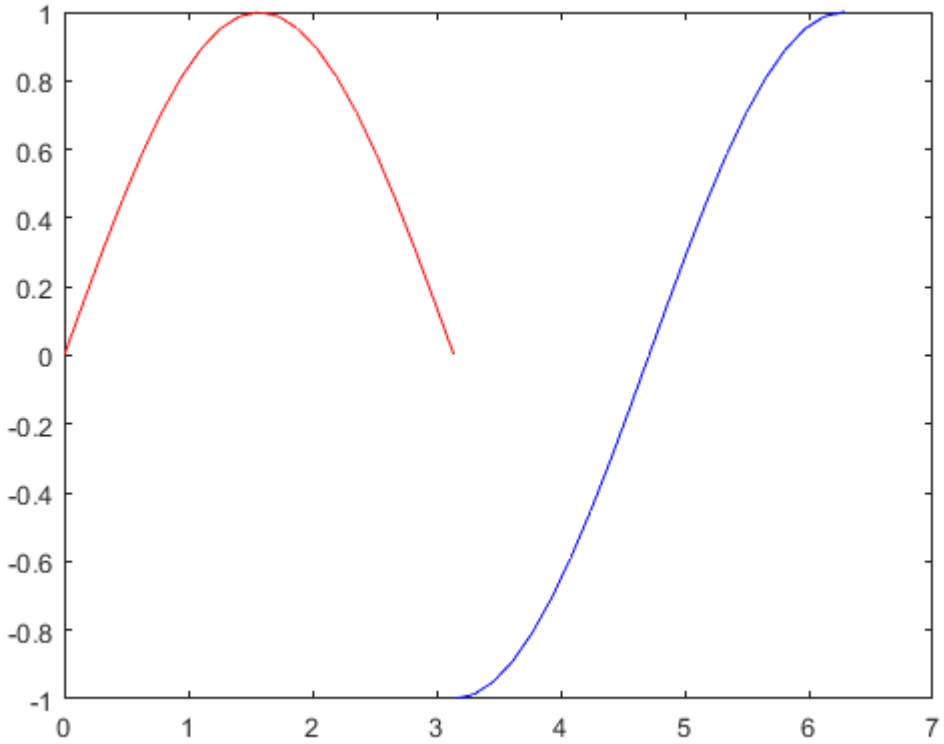


Figura 5 – Gráfico gerado com o comando `hold on`.

Como pode ser visto na figura 5, criou-se quatro variáveis, dividindo o tempo total em dois, cada um com sua função.

6.1.4 Comando `subplot`

Esse comando é usado quando se deseja colocar mais de um gráfico na mesma figura, mas em diferentes eixos, criando assim, múltiplos diagramas.

Ele é usado da maneira: `subplot(m, n, p)`. Onde **m** é o número de linhas, **n** é o número de colunas e **p** é a posição do gráfico que está sendo tratado. A figura abaixo mostra um exemplo.

```
x = pi:pi/20:2*pi;

subplot(2, 1, 1)
plot(sin(x))

subplot(2, 1, 2)
plot(cos(x))
```

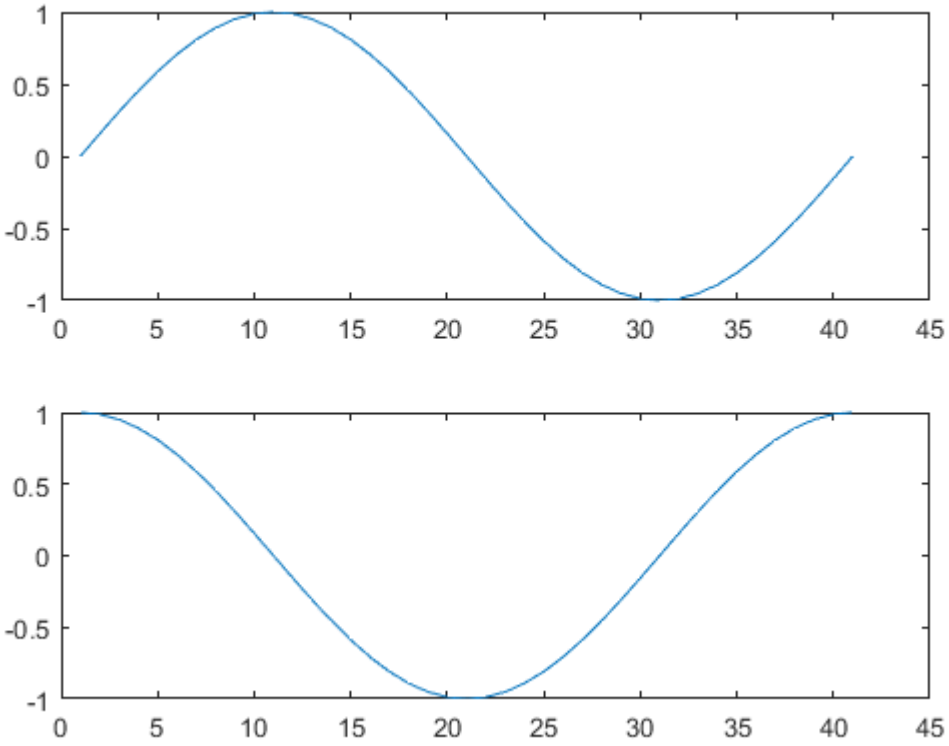


Figura 6 – Gráfico gerado com o comando `subplot`.

6.1.5 Comando `polar`

É possível desenhar gráficos usando coordenadas polares com o comando `polar`. Primeiro, é preciso definir o vetor θ e a função r para então, usar o comando. O exemplo pode ser visto na fig. 7.

```
theta = pi:pi/20:2*pi;

r = 2*(1+cos(theta));
polar(theta, r)
```

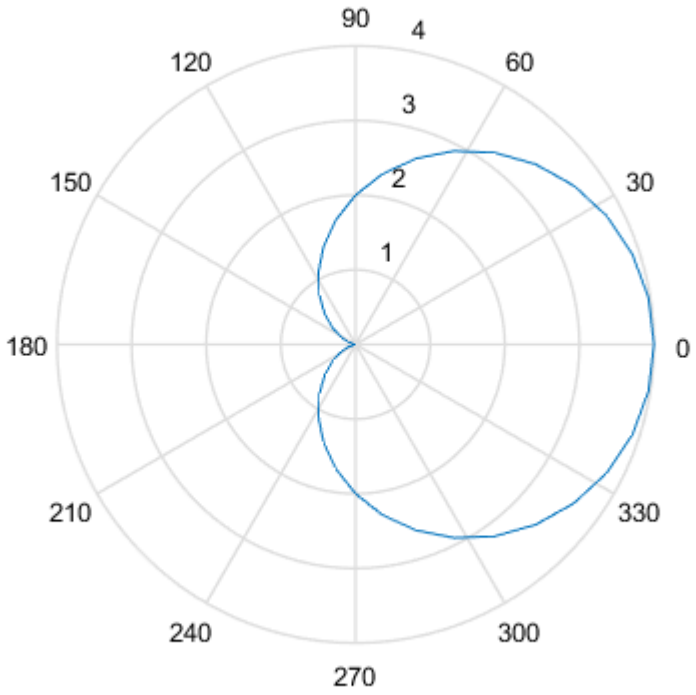


Figura 7 – Gráfico gerado com o comando `polar`.

Esse gráfico acima se chama cardióide.

6.1.6 Comando `ezplot`

O comando `ezplot` é usado para plotar gráficos com variáveis simbólicas. Um exemplo disso está na figura abaixo.

```
syms x
ezplot(sin(x))
ezplot(3*x + 1)
```

6.2 Diagramas

O MATLAB possui comando que permitem criar diagramas bidimensionais. A tabela a seguir sintetiza os comandos mais utilizados.

Tabela 11 – Diagrama

Comando	Descrição
`bar(x, y)`	Diagrama de barras vertical, x: rótulo, y: altura
`barh(x, y)`	Diagrama de barras horizontal, x: rótulo, y: altura
`pie(x)`	Diagrama de pizza
`pie(x, explode)`	Diagrama de pizza destacando dados
`pie3(x)`	Diagrama de pizza 3D
`stairs(y)`	Gráfico em escada
`stem(y)`	Gráfico em tempo discreto

Abaixo mostra-se exemplos de os comandos na Tabela 11.

Primeiro, diagrama de barras vertical. Esse exemplo é de uma pesquisa fictícia feita desde 1900 até 2000, de 10 em 10 anos. Esses valores são o x do diagrama, enquanto os números obtidos são o vetor y.

```
x = 1900:10:2000;
y = [10 5 2 6 11 7 2 4 5.6 8 3.2]
bar(x, y)
```

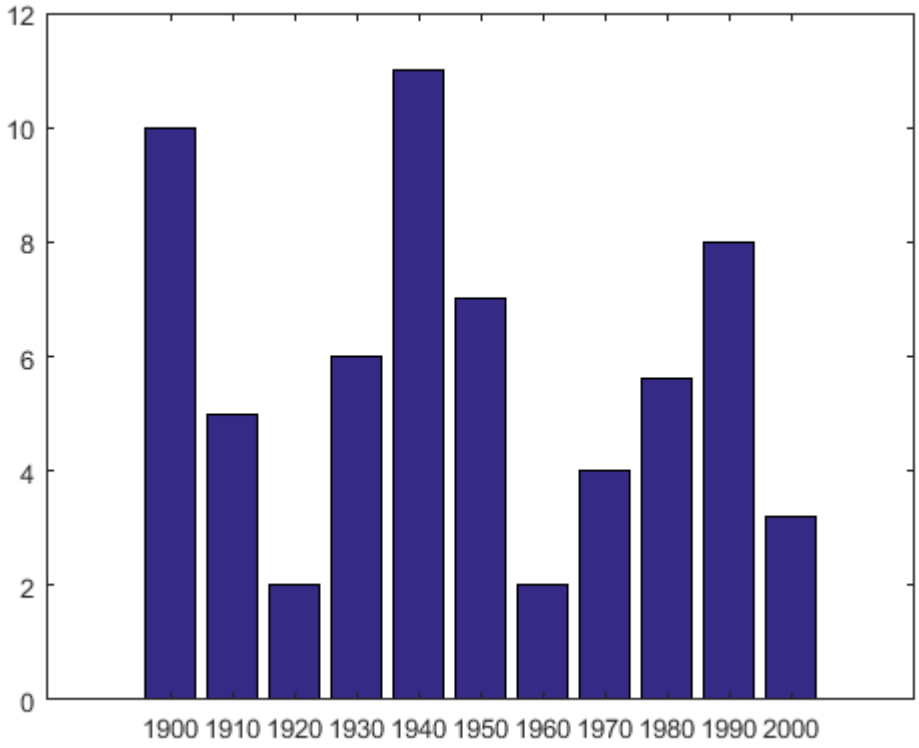


Figura 8 – Gráfico gerado com o comando `bar`.

Para diagrama de barras horizontal basta usar `barh`.

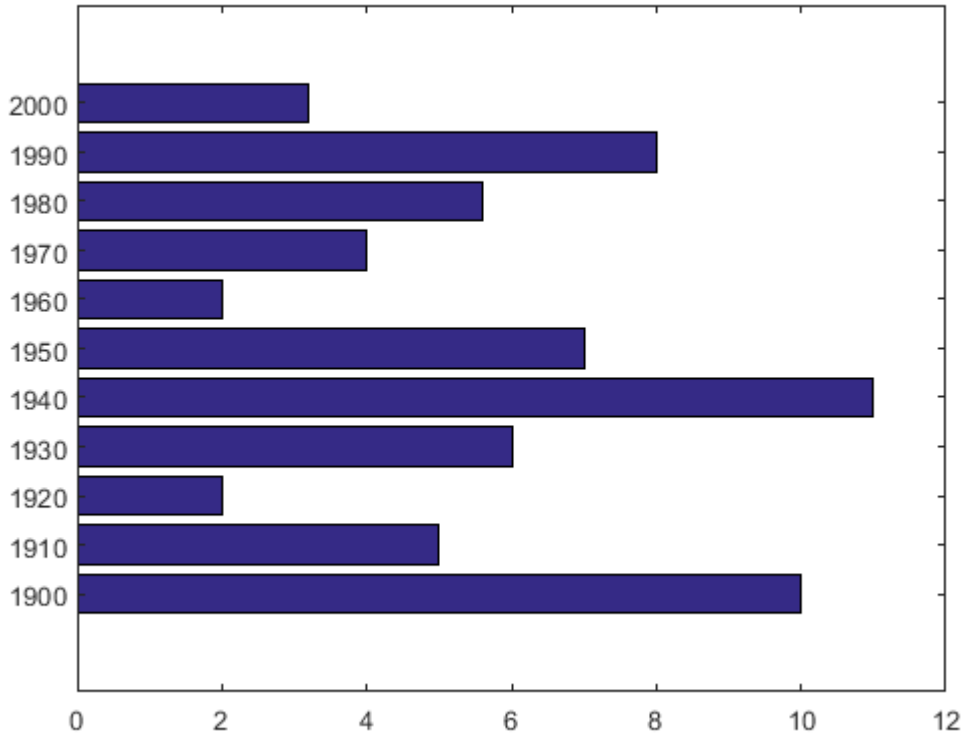


Figura 9 – Gráfico gerado com o comando `barh`.

O diagrama de pizza é chamado de diagrama *pie* em inglês, que significa torta. Para montar o gráfico é necessário definir um vetor. Diz-se que a soma de todos os valores do vetor é 100% e, assim, faz-se uma regra de 3 para descobrir quanto cada valor do vetor vale em porcentagem.

```
x = [10 5 2 8]
pie(x)
```

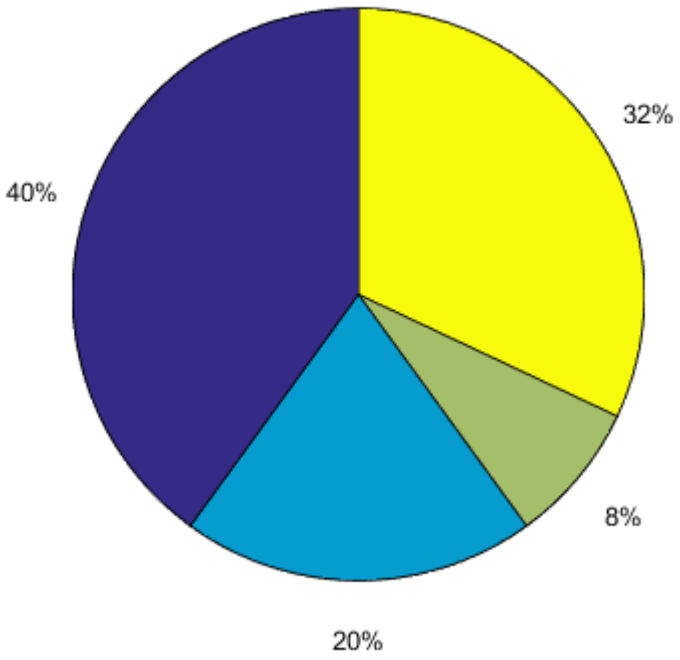


Figura 10 – Gráfico gerado com o comando `pie`.

Caso seja necessário chamar atenção para um determinado dado do gráfico, usa-se um novo vetor para indicar qual dado se quer destacar. Nesse vetor, 0 significa deixar o dado como está e 1 significa destacar o dado.

```
x = [10 5 2 8]
explode = [0 1 0 1]
pie(x, explode)
```

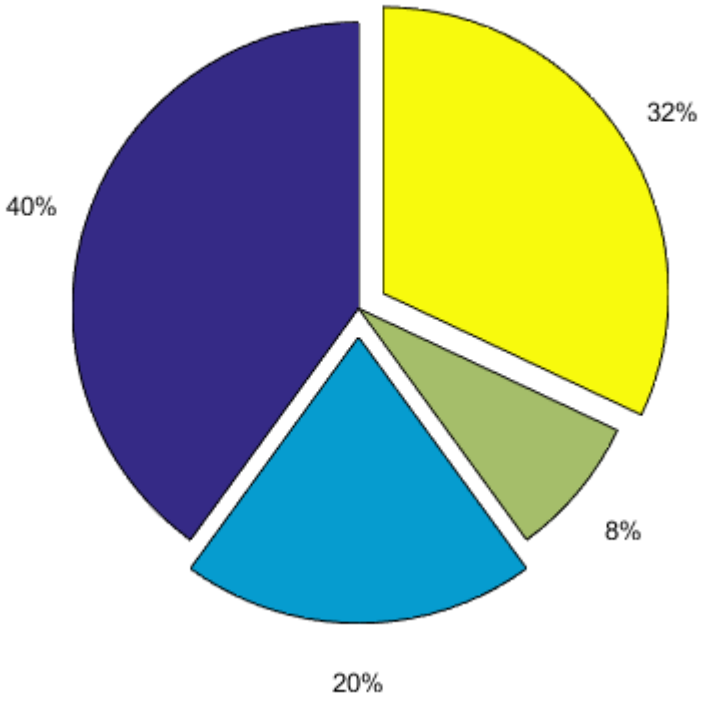


Figura 11 – Gráfico gerado com o comando `pie`.

O comando `stairs` plota o gráfico com a linha em escada. A figura abaixo exemplifica isso.

```
x = 0:pi/20:2*pi;
y = sin(x);
stairs(y)
```

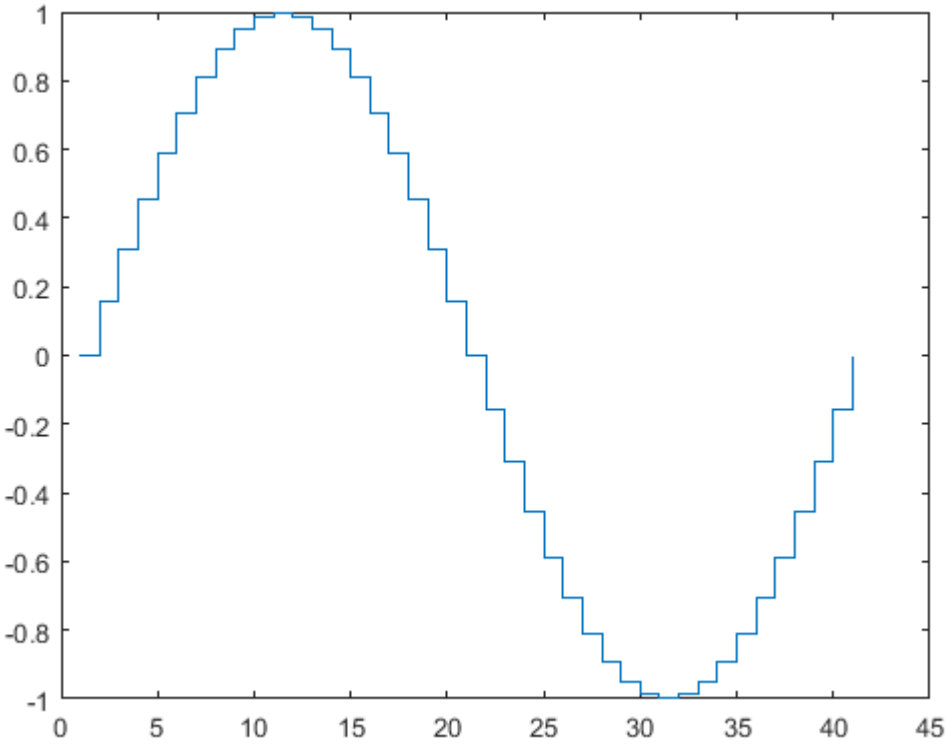


Figura 12 – Gráfico gerado com o comando `stairs`.

O comando `stem` é usado para plotar gráficos em tempo discreto, onde `'filled'` é um comando para preencher o círculo do gráfico.

```
x = pi:pi/20:2*pi;
y = sin(x);

subplot(2, 2, 1)
stem(y)

subplot(2, 2, 2)
stem(y, 'k', 'filled')
```

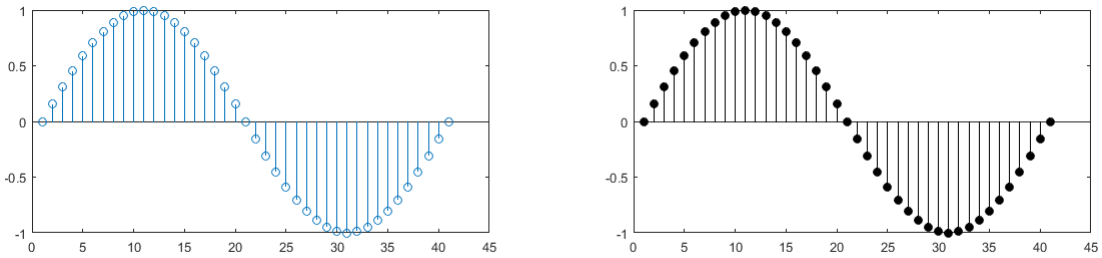


Figura 13 – Gráficos gerados com o comando `stem`.

6.3 Gráficos 3D

Com o comando `plot3` é possível criar gráficos em 3 dimensões. Para isso, basta definir uma função para cada eixo, como mostram as figuras abaixo.

```
x = rand(100, 1);
y = rand(100, 1);
z = rand(100, 1);
plot3(x, y, z)
```

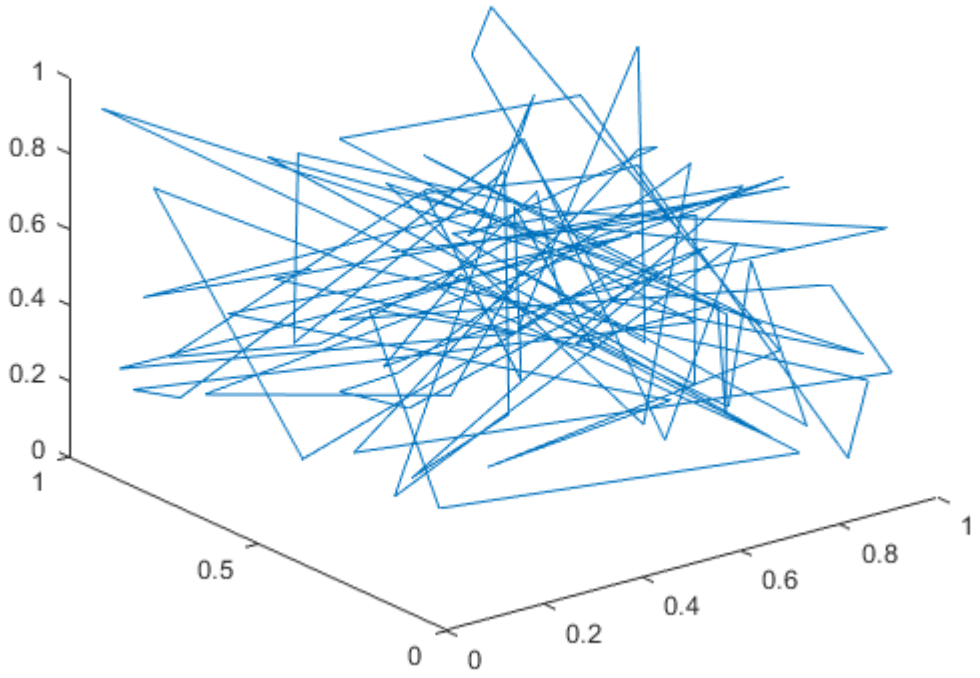


Figura 14 – Gráfico gerado com o comando `plot3`.

Também é possível utilizar o comando `meshgrid` para criar gráficos 3D. É preciso definir uma função de duas variáveis `z = f(x, y)`.

Primeiro, cria-se intervalos de pontos onde o gráfico será desenhado, ou seja, uma malha, a intersecção de dois intervalos, um no eixo x e outro no eixo y.

```
x = x0 : delta x : x1
y = y0 : delta y : y1
```

Após utiliza-se o comando `meshgrid`, que cria matrizes X e Y que serão usadas para plotar o gráfico. Finalmente, define-se uma função Z que depende de X e Y para, então, plotar o gráfico com o comando `surf`.

```
x = 1:0.5:50;
y = 1:0.2:25;
[X, Y] = meshgrid(x, y);
Z = sin(X) + cos(Y);
surf(X, Y, Z)
```

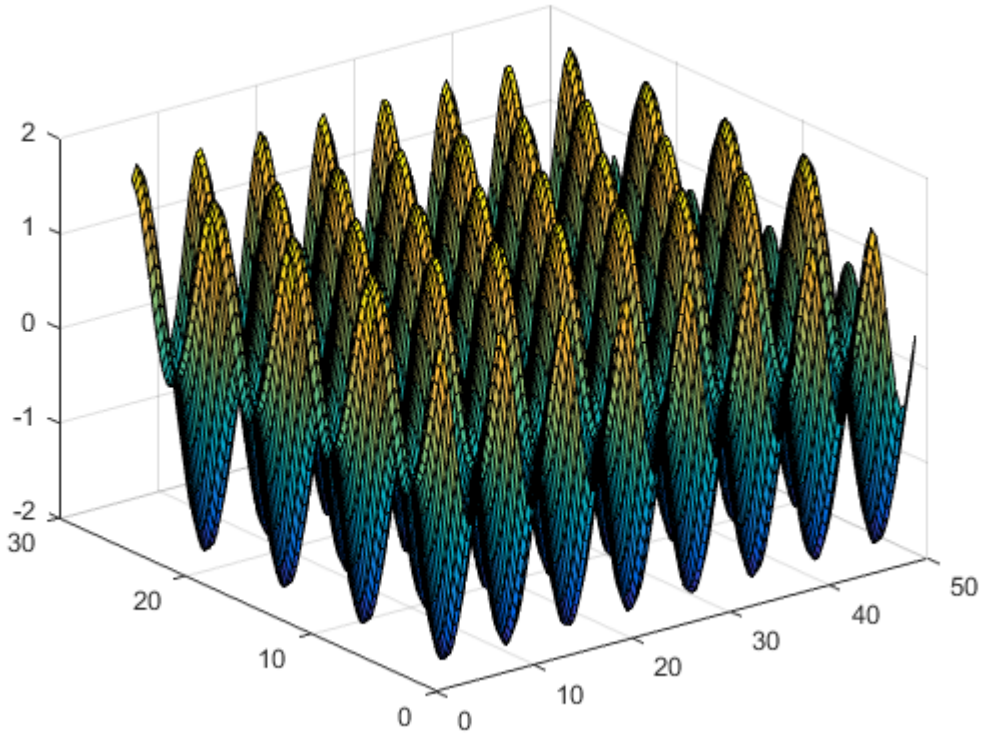


Figura 15 – Gráfico gerado com os comandos `meshgrid` e `surf`.

Se em vez de usar o comando `surf` para plotar o gráfico, usar o comando `mesh`, o gráfico será o mesmo, porém vazado, como mostra a figura abaixo.

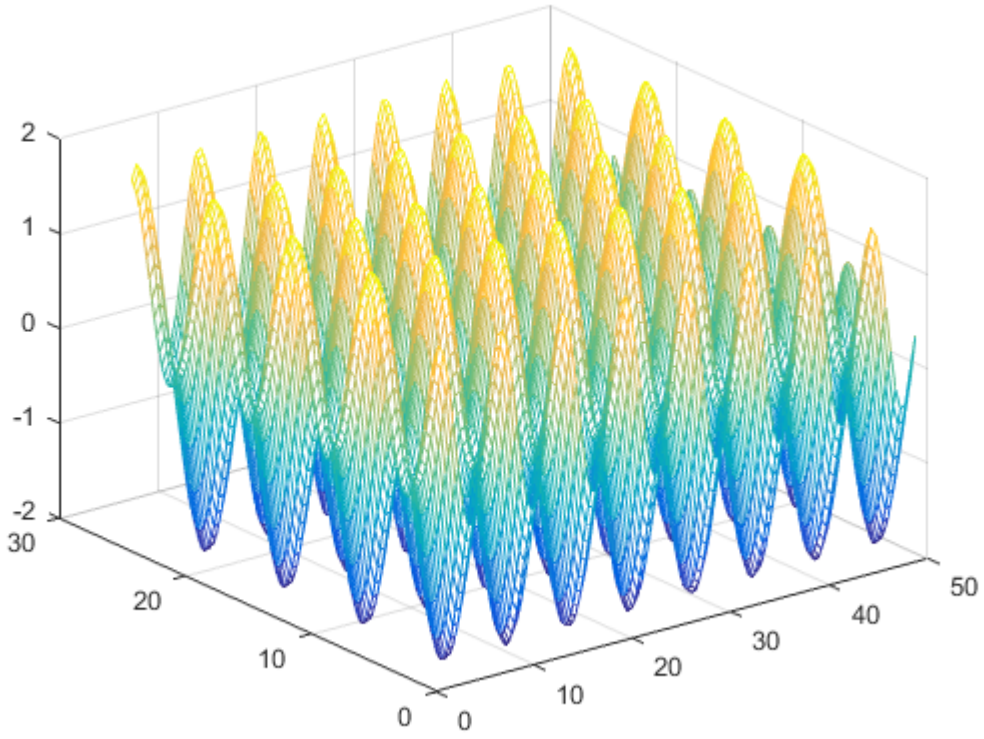


Figura 16 – Gráfico gerado com os comandos `meshgrid` e `mesh`.

O comando `meshz` é análogo ao `mesh`, porém cria uma espécie de cortina em torno da superfície.

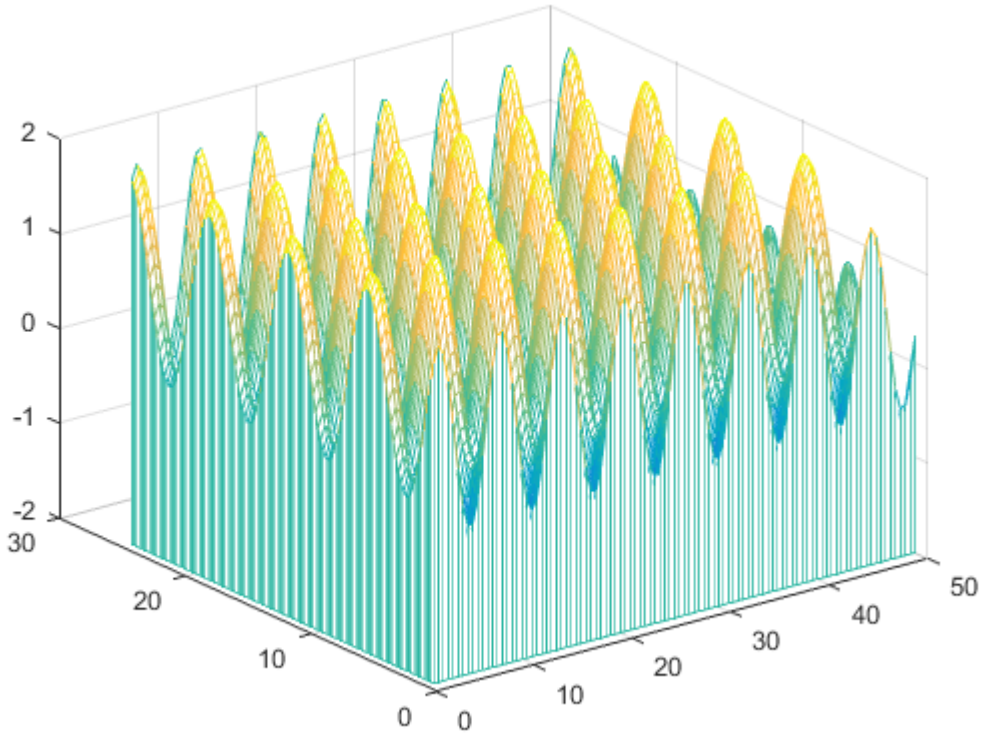


Figura 17 – Gráfico gerado com os comandos `meshgrid` e `meshz`.

O comando `contour` serve para criar curvas de níveis. Ele é definido usando o `meshgrid` da mesma maneira que o `mesh` e o `surf`, mas usando `contour` no lugar de `surf` ao final. A figura abaixo ilustra isso.

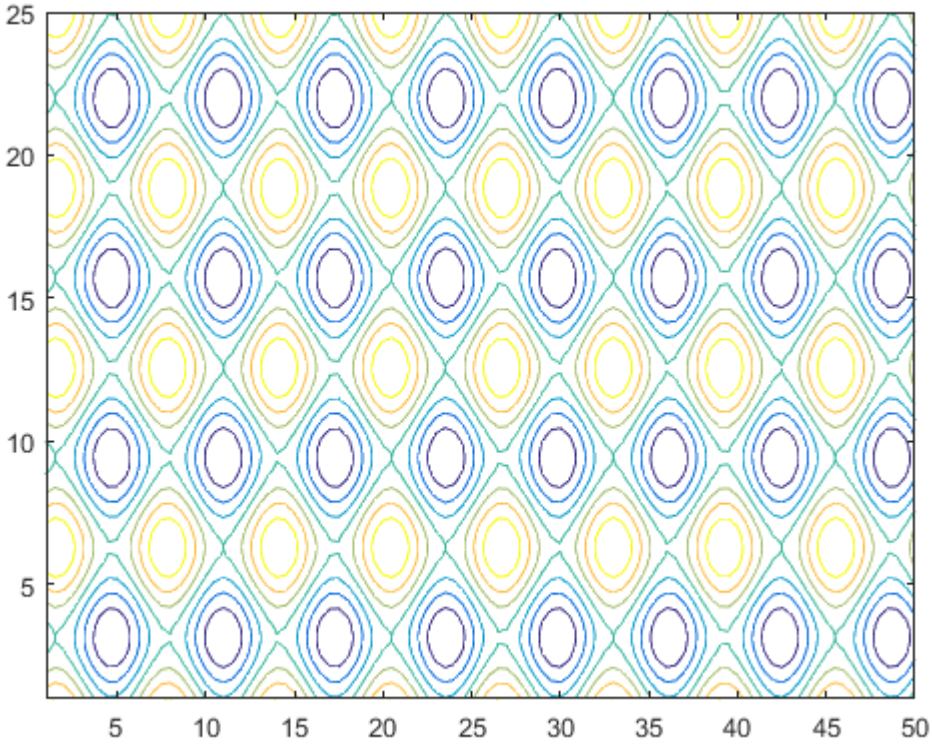


Figura 18 – Gráfico gerado com os comandos `contour`.

Usando `contour(X,Y,Z, n)` é possível especificar quantas curvas de níveis se deseja, onde **n** é o número desejado.

Para criar curvas de nivel preenchidas, utiliza-se `contourf` no lugar de `contour`, obtendo, assim, a figura abaixo.

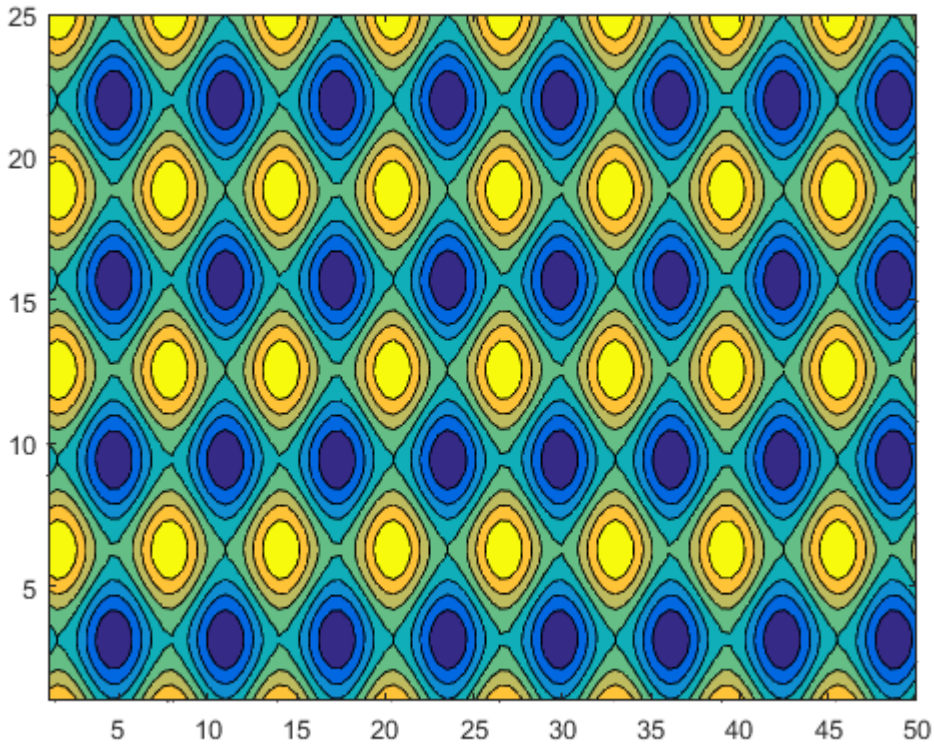


Figura 19 – Gráfico gerado com os comandos `contourf`.

Já o comando `contour3` cria as curvas de nível em três dimensões.

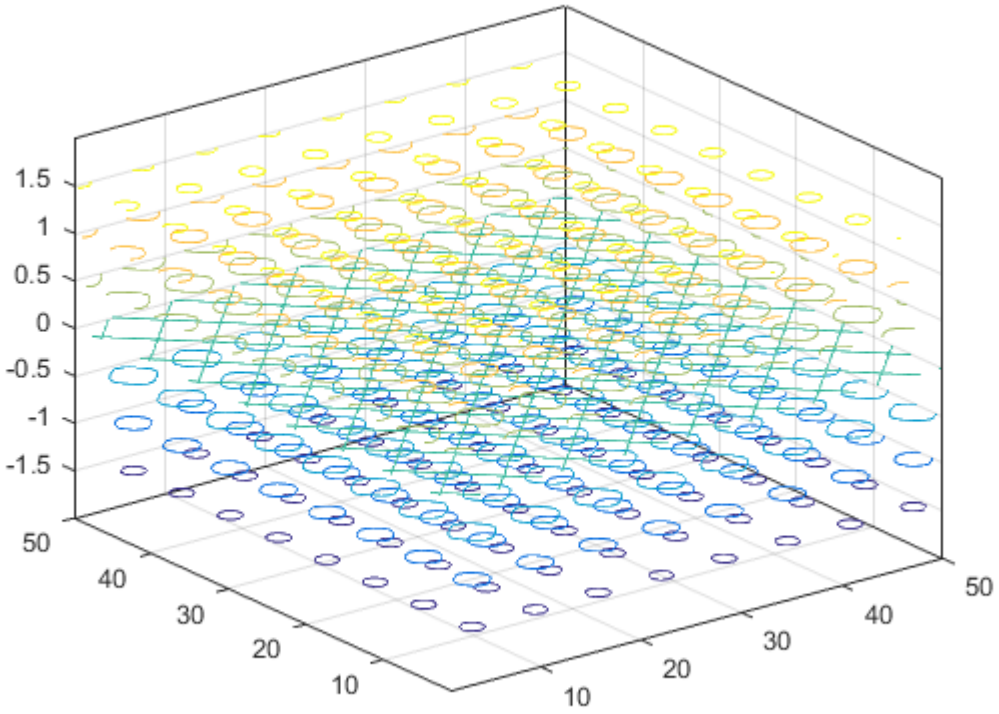


Figura 20 – Gráfico gerado com os comandos `contour3`.

A tabela abaixo mostra um resumo dos comandos que são usados com o `meshgrid`.

Tabela 12 – Comandos usando `meshgrid`

Comando	Descrição
`surf`	Gráfico 3D
`mesh`	Gráfico 3D vazado
`meshz`	Gráfico 3D vazado com "cortina"
`contour`	Curvas de nível
`contourf`	Curvas de nível preenchidas
`contour3`	Curvas de nível em 3D

6.3.1 Comando `sphere` e `cylinder`

É possível criar uma esfera no MATLAB com o comando `sphere`.

```
[x, y, z] = sphere;
surf(x, y, z)
hold on
surf(x+3, y-2, z)
surf(x, y+1, z-3)
```

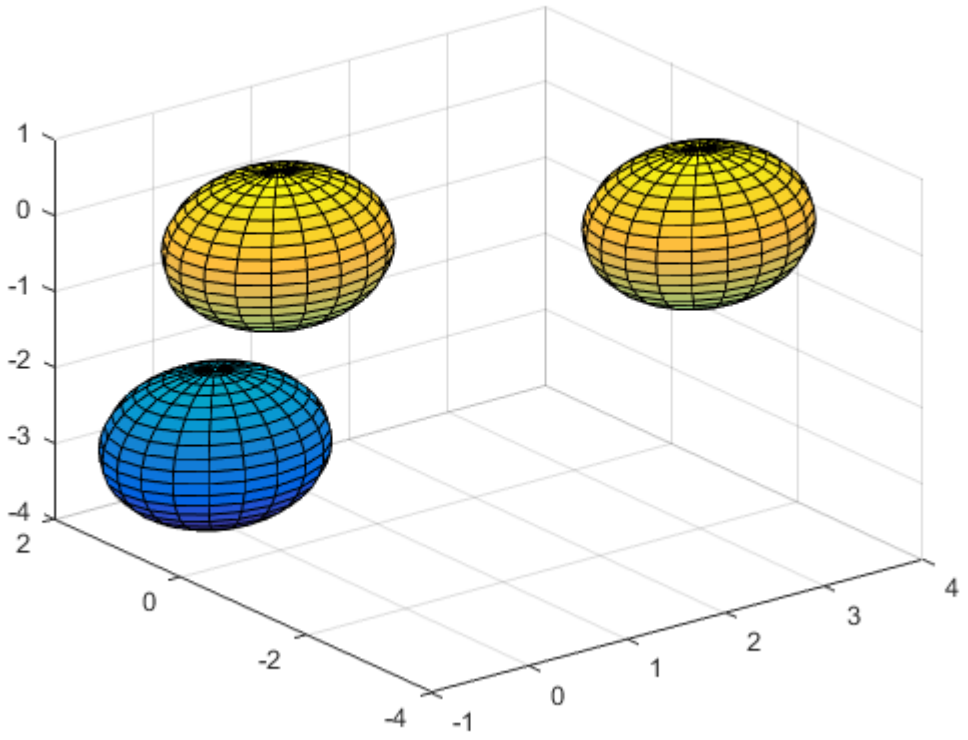


Figura 21 – Gráfico gerado com os comandos `sphere`.

O comando `cylinder` é semelhante, só que em vez de usarmos `[x, y, z] = sphere`, usa-se `[x, y, z] = cylinder`.

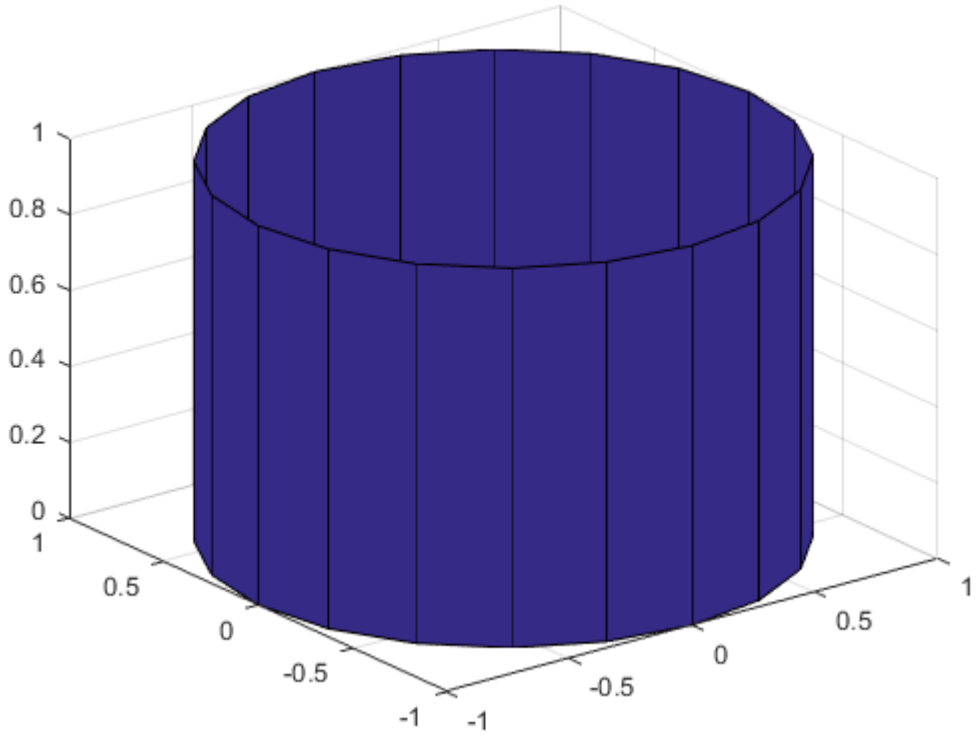


Figura 22 – Gráfico gerado com os comandos `cylinder`.

6.3.2 Comando `shading` e `FaceColor`

O comando `shading` serve para alterar o preenchimento da figura. Existem três tipos, o `default`, o padrão, o `shading flat` e o `shading interpol`.

```
subplot(2, 3, 1)
[x, y, z] = sphere;
surf(x, y, z)

subplot(2, 3, 2)
surf(x, y, z)
shading flat

subplot(2, 3, 3)
surf(x, y, z)
shading interp
```

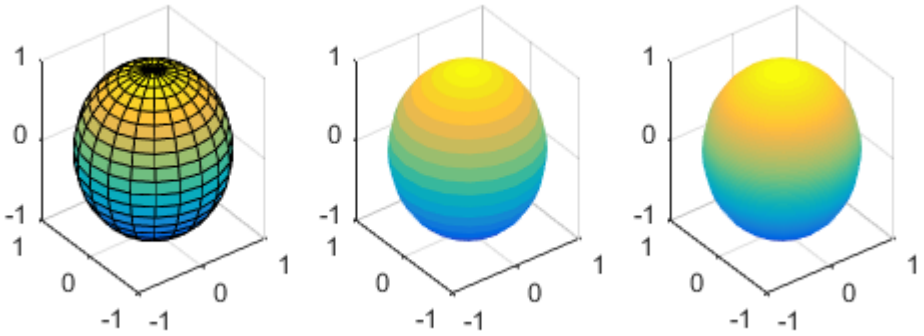


Figura 23 – Gráfico gerado com os comandos `shading`.

Já o comando `FaceColor` é parecido com o `fill` mas para gráficos em três dimensões. Ele é usado de forma semelhante.

```
x = 1:0.1:10;
y = 1:1:50;
[X, Y] = meshgrid(x, y);
Z = sin(X) + cos(Y)
surf(X, Y, Z, 'FaceColor', 'm')
```

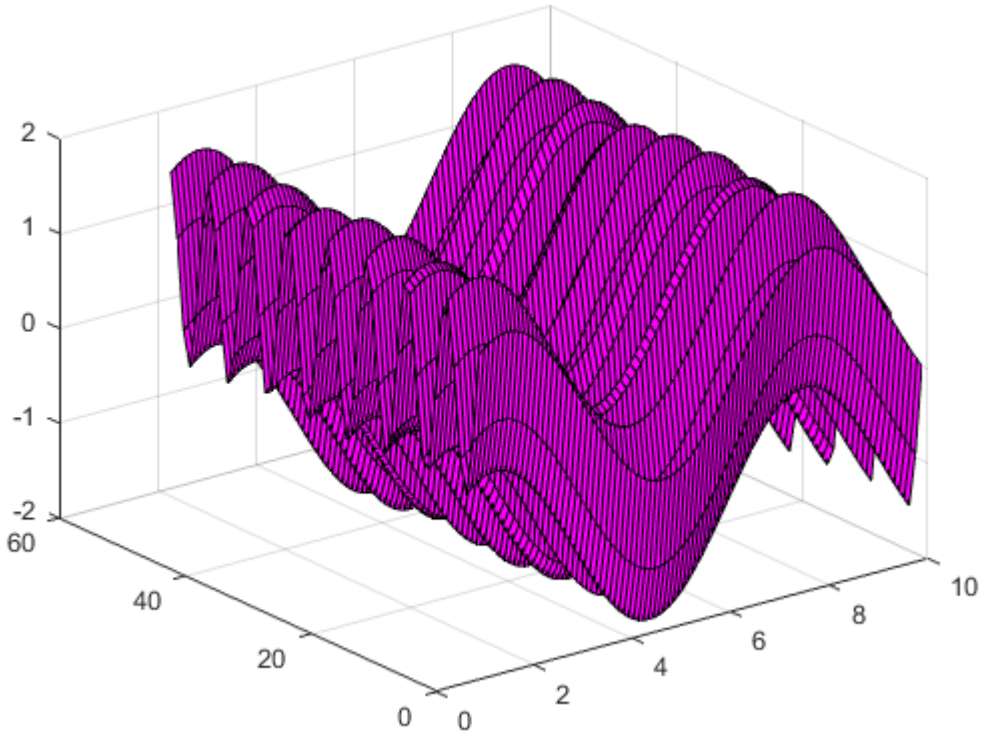


Figura 24 – Gráfico gerado com os comandos `FaceColor`.

7 Programação

O MATLAB é uma ferramenta muito útil para programação. Sua linguagem é muito parecido com a linguagem C, por isso, não apresenta muitas dificuldades para iniciantes.

A tabela abaixo mostra alguns comandos básicos na hora de programar.

Tabela 13 – Comandos de programação

Comando	Descrição	Exemplo
`fprintf`	Mostra na tela	`fprintf('Mensagem')`
`disp`	Mostra o valor da variável	`disp(x)`
`input`	Entra com um valor	`a = input('Digite um número')`
`warning`	Mostra um aviso	`warning('Aviso')`
`clear`	Apaga as variáveis	`clear all`

7.1 Controle de fluxo

O controle de fluxo se resume alguns comandos condicionais: `if`, `else`, `for`, `switch` e `while`.

7.1.1 Comando `if`

Esse é o comando condicional mais simples, significa *se*. Ele permite executar um comando caso uma condição seja satisfeita.

```
a = -3;
if a >= 5
    disp('a é maior que 5')
end
```

7.1.2 Comando `else`

O comando `else` é usado quando deseja-se adicionar mais alternativas no laço `if`. A diferença entre o `else` e o `elseif` é que quando usa-se o `elseif` é necessário dar uma segunda condição, já quando usa-se somente o `else` isso não é preciso.

```
n = input('Digite um número: ')

if n == 0
    fprintf('n é zero')
elseif n>0
    fprintf('n é positivo')
else
    fprintf('n é negativo')
end
```

7.1.3 Comando `for`

```
for c = 1:2
    X(c) = c^2;
end
```

7.1.4 Comando `switch`

O comando `switch` é usado quando se tem opções de escolha. Ele executa determinado comando baseando-se no valor da variável. A figura abaixo mostra o seu funcionamento.

```
x = [12 64 24];
a  = 'pie3';

switch a
case 'bar'
    bar(x)
    title('Gráfico de barras')
case 'barh'
    barh(x)
    title('Gráfico de barras horizontal')
case 'pie'
    pie(x)
    title('Gráfico de pizza')
case 'pie3'
   pie3(x)
    title('Gráfico de pizza 3D')
otherwise
    fprintf('Erro')
end
```

O `case` é executado assim que achar a afirmação em que se encaixa. Após, a função prossegue para `end`. Se nenhum dos `case` concordar com a expressão do `switch`, o comando `otherwise` é executado.

7.1.5 Comando `while`

O laço `while` repete um grupo de comandos um número de vezes, até obter-se uma resposta satisfatória ou até que se interrompa programa. Pode ser usado da seguinte maneira:

```
n = 10;
f = n;

while n>1
    n = n-1;
    f = f*n;
    disp(f)
end
```

O exemplo é um programa usando o laço `while` para calcular o fatorial, ou seja, enquanto $n < 1$, ele vai executando a função.

7.2 Funções

As funções são arquivos que podem aceitar argumentos de entrada e retornar argumentos de saída. Os nomes do arquivo e da função devem ser os mesmos. Funções operam em variáveis dentro de seu próprio espaço de trabalho, separado da `Workspace` do MATLAB.

Os textos abaixo mostram um exemplo de como criar e chamar uma função.

```
function y = mediana(x)
   y = sum(x)/length(x);
end

a =
    2   3   4

>> mediana(a)
ans =
    3
end
```

A primeira linha de uma função começa com a palavra `function`. Ela dá o nome da função e ordem dos argumentos.

7.2.1 Tipos de funções

7.2.1.1 Função anônima

Uma função anônima é uma forma simples de função MATLAB que não requer um arquivo **.m**. Consiste em uma única expressão MATLAB e qualquer número de argumentos de entrada e saída. É possível definir uma função anônima diretamente em uma linha de comando MATLAB ou dentro de uma função ou `script`.

```
>> quadrado = @(x) x.^2
>> a = 9
>> quadrado(a)

ans =
    3
```

O comando geral para criar uma função anônima é `nome_da_função = @(argumento) expressão`.

7.2.1.2 Funções primárias e subfunções

Qualquer função que não seja anônima deve ser definida dentro de um arquivo. Cada um desses arquivos de função contém uma função primária necessária que aparece primeiro e qualquer número de subfunções pode segui-la. As funções primárias têm um escopo mais amplo do que as subfunções, ou seja, as funções primárias podem ser chamadas de fora do arquivo que as define (por exemplo, a partir da linha de comando MATLAB ou de funções em outros arquivos) enquanto as subfunções não podem. As subfunções são visíveis apenas para a função principal e outras subfunções dentro de seu próprio arquivo.

7.2.1.3 Função privada

Uma função privada é um tipo de função primária. Sua característica única é que é visível apenas para um grupo limitado de outras funções. Esse tipo de função pode ser útil caso seja preciso limitar o acesso a uma função, ou quando escolhe-se não expor a implementação de uma função.

As funções privadas residem em subdiretórios com o nome especial privado. Eles são visíveis apenas para funções no diretório pai.

Como as funções privadas são invisíveis fora do diretório pai, elas podem usar os mesmos nomes que funções em outros diretórios. Isso é útil caso deseje-se criar sua própria versão de uma função específica, mantendo o original em outro diretório.

7.3 Variáveis globais

Se você deseja que mais de uma função compartilhe uma única cópia de uma variável, simplesmente declare a variável como global em todas as funções. Faça o mesmo no linha de comando se desejar que o espaço de trabalho base acesse a variável. A declaração de global deve ocorrer antes que a variável seja realmente usada em uma função.

7.4 Scripts

Quando se usa um *script*, o MATLAB executa os comandos contidos no arquivo. Os *scripts* podem operar em dados existentes na *Workspace* (espaço de trabalho), ou podem criar novos dados para operar. Embora os *scripts* não retornem argumentos de saída, quaisquer variáveis que eles criam permanecem na *Workspace* mesmo após a conclusão do arquivo. Além disso, os *scripts* podem produzir gráficos como saída.

Usando o caractere % é possível escrever comentários, ou seja, linhas que o MATLAB vai ignorar e que servem para explicar o conteúdo do *script*.

8 Cálculo

8.1 Limites

É possível calcular $lim_{x \to a} f(x)$ utilizando o comando `limit(f(x), x, a)`.

```
>> syms h n x
>> limit((cos(x+h) - cos(x))/h, h, 0)

ans =
    -sin(x)
```

8.1.1 Limites laterais

Para calcular o limite lateral usa-se `limit (f(x), x, a, 'left')` para o limite à esquerda e `limit (f(x), x, a, 'right')` para o limite à direita.

```
>> syms x
>> limit(x/abs(x), x, 0, 'left')

ans =
    -1

>> limit(x/abs(x), x, 0, 'right')

ans =
    1
```

8.2 Derivadas

É possível calcular a derivada de uma função utilizando o comando `diff(f(x), x, n)`, onde `n` é a ordem da derivação. O comando também funciona para matrizes e vetores.

```
>> syms x
>> f = sin(5*x);
>> diff(f)

ans =
    5*cos(5*x)

>> g = x^2*cos(x);
>> diff(g)

ans =
    2*x*cos(x) - x^2*sin(x)
>> syms x
>> g - x^2*cos(x);
>> diff(g)

ans =
    2*x*cos(x) - x^2*sin(x)

>> diff(g,2)

ans =
    2*cos(x) - x^2*cos(x) - 4*x*sin(x)

>> diff(g, 3)
ans =
    x^2*sin(x) - 6*sin(x) - 6*x*cos(x)
```

8.2.1 Derivadas parciais

Para diferenciar uma expressão que contenha mais de uma variável simbólica, é necessário especificar a variável que se deseja diferenciar em relação a ela. O comando `diff`, então, calcula a derivada parcial da expressão em relação a essa variável.

```
>> syms s t
>> f = sin(s*t);
>> diff(f, t)

ans =
    s*cos(s*t)

>> diff(f, s)

ans =
    t*cos(s*t)
```

8.3 Integrais

8.3.1 Integral indefinida

É possível calcular a integral indefinida de uma função utilizando o comando `int(f(x), x)` ou apenas `int(f(x))`.

```
>> syms y
>> int(y^(-1))

ans =
    log(y)
```

8.3.2 Integral definida

É possível calcular a integral definida de uma função utilizando o comando `int(f(x), x, a, b)`, onde `a` e `b` são os limites da integração.

```
>> syms x
>> int(x^7, 0 , 1)

ans =
    1/8
```

9 Solução de Equações

O comando `solve` soluciona equações algébricas simbólicas.

```
>> syms a x b c
>> s = a*x^2 + b*x + c
>> solve(s)

ans =
    -(b + (b^2 - 4*a*c)^(1/2))/(2*a)
    -(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```

Nesse caso, a equação foi resolvida tratando x como variável de interesse. Caso deseje-se resolver a equação para outra variável, é necessário escrever o comando como `solve(s, b)`, onde `s` é a equação e `b` é a variável de interesse.

```
>> syms a b c x
>> s = a*x^2 + b*x + c;
>> b = solve(s, b)

b =
    -(a*x^2 + c)/x
```

Também é possível resolver a equação para mais de uma variável, como visto abaixo.

```
>> syms a b c x y
>> [x, y] = solve(a*x^2 + b*x + c)

x =
    -(c + (c^2)^(1/2))/(2*a)
    -(c - (c^2)^(1/2))/(2*a)

y =
    0
    0
```

10 Equações Diferenciais

Já para equações diferenciais, usa-se o comando `dsolve`. É possível definir condições iniciais ou não.

```
>> syms y
>> dsolve('Dy = 1 + y^2')

ans =
    tan(C2 + t)
            1i
           -1i

>> dsolve('Dy = 1 + y^2', 'y(0) = 1')

ans =
    tan(t + pi/4)
```

Esse comando também serve para resolução de equações diferenciais não lineares, como mostra a figura a seguir.

```
>> x = dsolve('(Dx)^2 + x^2 = 1', 'x(0)=0')

x =
    -(exp (-t*1i - (pi*1i)/2*(exp(t*2i)- 1))/2
    -(exp (t*1i - (pi*1i)/2*(exp(-t*2i)- 1))/2
```

A figura abaixo mostra um exemplo do cálculo de uma equação diferencial de segunda ordem.

```
>> y = dsolve('(D2y = cos(2*x)-y', 'y(0)=1', 'Dy(0)=0', 'x')
>> simplify(y)

ans =
    1 - (8*sin(x/2)^4)/3
```

10.1 Comando ode23

É possível resolver equações diferenciais de primeira ordem em um intervalo com o comando `ode23`. É preciso definir o intervalo de tempo assim como uma condição inicial. Esse comando age chamando uma função anônima.

```
intervalo = [0 5];
y0 = 0;
[t, y] = ode23(@(t, y) 2*t, intervalo, y0);
plot(t, y, '-o')
```

11 Aproximação de polinômio

O comando `polyfit` permite ajustar os coeficientes da função polinomial para minimizar o erro entre a função aproximada e a função polinomial utilizada.

A função `polyfit` efetua o ajuste dos coeficientes da um polinômio de qualquer grau. Ela recebe dois vetores de entrada e um número que representa o grau do polinômio. A saída da função é um vetor com coeficientes.

Para verificar o resultado do ajuste usa-se a função `polyval`, que resolve o polinômio em um ou vários pontos desejados, caso a entrada de `polyval` seja um vetor de pontos.

```
x = 0:pi/20:2*pi;
y = sin(x) + cos(2*x);
p = polyfit(x, y, 2) %Aproxima o polinômio
ya = polyval(p, x) %Resolve o polinômio

plot(x, y, '*')
hold on
 plot(x, ya, 'red--')
```

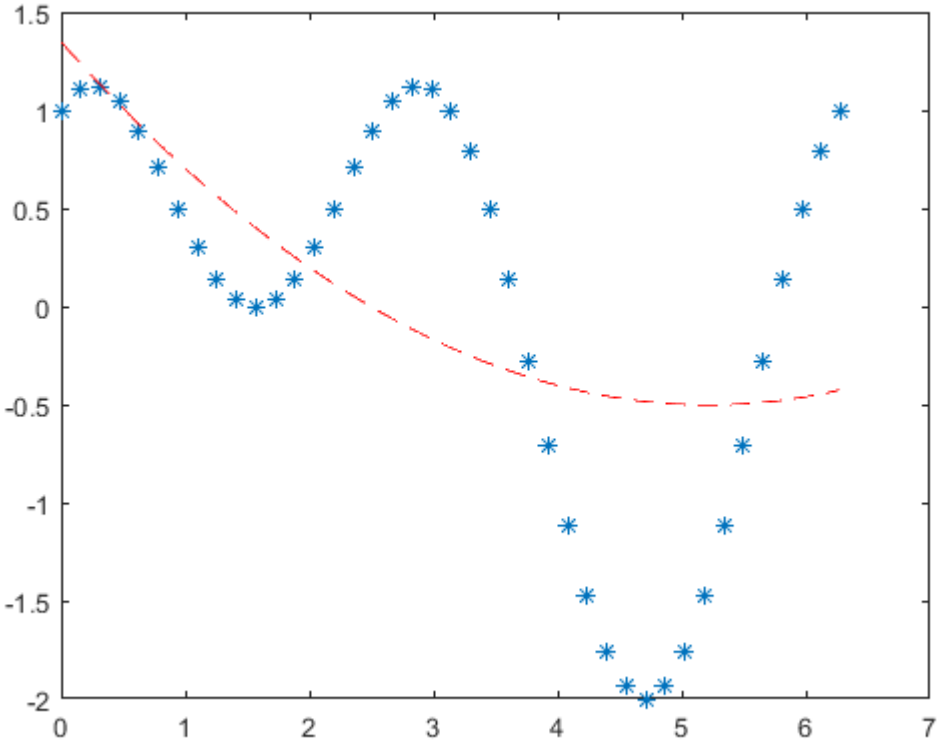


Figura 25 – Exemplo da comparação entre o polinômio e a sua aproximação.

12 Toolboxes

As *Toolboxes*, ou bibliotecas, são pacotes de funções específicas. Tanto o MATLAB quanto o Simulink possuem várias, cada uma com um propósito.

Cada uma dessas bibliotecas possuem vários exemplos prontos, e é possível acessá-los usando o comando `doc` seguido do nome da *toolbox* que deseja. Dessa maneira, uma página irá abrir com a explicação da toolbox, assim como alguns links para exemplos e explicações mais aprofundadas.

12.1 Aerospace Toolbox

A *Aerospace Toolbox* fornece padrões de referência, modelos ambientais e importação de coeficientes aerodinâmicos para realizar análises aeroespaciais avançadas para desenvolver e avaliar projetos. As opções para visualizar a dinâmica do veículo incluem um objeto de animação MATLAB® de seis graus de liberdade e interfaces para o simulador de voo *FlightGear* e o software *Simulink® 3D Animation*^{TM}. Essas opções permitem visualizar dados de voo em um ambiente tridimensional e reconstruir anomalias comportamentais nos resultados do teste de voo.

Abaixo estão listadas algumas características principais dessa biblioteca.

- Inclui modelos ambientais baseados em padrões para atmosfera, gravidade, altura de geoide, vento, corpos celestes e campo magnético.
- Converte unidades e transforma sistemas de coordenadas e representações espaciais, incluindo rotações de e para Euler-Rodrigues.
- Implementos de utilitários predefinidos para cálculos de parâmetros aeroespaciais, cálculos de tempo e matemática de quaternion.
- Importa coeficientes aerodinâmicos dos Dados Digitais da Força Aérea dos EUA (Datcom).
- Fornece opções para visualizar a dinâmica do veículo em um ambiente 3-D, incluindo uma interface para o simulador de vôo FlightGear.

12.2 MATLAB Coder

O MATLAB Coder gera código C e C ++ legível e portátil a partir do código MATLAB. Ele suporta a maioria do idioma MATLAB e uma ampla gama de caixas de ferramentas. Você pode integrar o código gerado em seus projetos como código fonte, bibliotecas estáticas ou bibliotecas dinâmicas. Você também pode usar o código gerado dentro do ambiente MATLAB para acelerar partes intensivas em computação de seu código MATLAB. O MATLAB Coder permite incorporar o código C inicial no seu algoritmo MATLAB e no código gerado.

Abaixo estão listadas algumas características principais dessa biblioteca.

- Geração de códigos C e C++ compatíveis com ANSI/ISO.
- Suporte de geração de código para caixas de ferramentas, incluindo o *Communication System Toolbox*^{TM}, *Computer Vision System Toolbox*^{TM}, *DSP System Toolbox*^{TM}, *Image Processing Toolbox*^{TM} e *Signal Processing Toolbox*^{TM}.
- Geração de função MEX para verificação de código.
- Integração do código Legacy C em algoritmos do MATLAB®.
- Geração de código *multicore* usando o *OpenMP*.
- Controle de alocação de memória estática ou dinâmica.
- Aplicação e funções de linha de comando equivalentes para gerenciar projetos de geração de código.

12.3 Control System Toolbox

O *Control System Toolbox* fornece algoritmos e aplicativos para analisar, projetar e ajustar sistemas de controle linear de forma sistemática. É possível especificar o sistema como uma função de transferência, espaço de estado ou modelo de resposta de frequência. Os aplicativos e funções como o gráfico de resposta ao degrau e o diagrama Bode permitem analisar e visualizar o comportamento do sistema nos domínios do tempo e frequência.

Abaixo estão listadas algumas características principais dessa biblioteca.

- Modelos de função de transferência, estado-espaço, zero/polo/ganho e resposta em frequência de sistemas lineares.
- Resposta ao degrau, diagrama de Nyquist e outras ferramentas de domínio do tempo e da frequência para analisar estabilidade e desempenho.

- Sintonização automática de PID, sistemas de controle SISO (*Single Input Single Output*) e MIMO (*Multiple Input Multipe Output*) arbitrários.
- Diagramas de Bode, LQR, LQG e outras técnicas de design clássico e de espaço de estado.
- Conversão de representação de modelos, discretização do modelo em tempo contínuo e aproximação de baixa ordem de sistemas de alta ordem.

12.4 Optimization Toolbox

A *Optimization Toolbox* resolve problemas de otimização linear, quadrática, inteira e não-linear. Fornece funções para encontrar parâmetros que minimizem ou maximizam os objetivos ao mesmo tempo que satisfazem restrições. A caixa de ferramentas inclui solucionadores para programação linear, programação linear mista, programação quadrática, programação não linear, mínimos quadrados lineares restritos, mínimos quadrados não lineares e equações não-lineares.

É possível definir o problema de otimização com funções e matrizes ou especificando expressões variáveis que refletem a matemática necessária.

Pode-se usar os solucionadores da caixa de ferramentas para encontrar soluções ótimas para problemas contínuos e discretos, realizar análises de *tradeoff* e incorporar métodos de otimização em algoritmos e aplicativos. A caixa de ferramentas permite executar tarefas de otimização de projeto, incluindo estimativa e ajuste de parâmetros e seleção de componentes. Pode ser usado para encontrar soluções ótimas em aplicações como otimização de portfólio, alocação de recursos e planejamento de produção.

Abaixo estão listadas algumas características principais dessa biblioteca.

- Otimização não linear e multiobjetiva de problemas restritos e sem restrições.
- Solver para mínimos quadrados não lineares, mínimos quadrados lineares restritos, ajustes de dados e equações não-lineares.
- Programação quadrática, programação linear e Programação linear mista inteira.
- Ferramentas de modelagem de otimização.
- Monitoramento gráfico do progresso da otimização.
- Aceleração da estimativa de gradiente (com Parallel Computing Toolbox ^{TM})

12.5 DSP System Toolbox

O *DSP System Toolbox* fornece algoritmos, aplicativos e escopos para projetar, simular e analisar sistemas de processamento de sinais em MATLAB e Simulink. Você pode modelar sistemas DSP em tempo real para comunicações, radar, áudio, dispositivos médicos, IoT e outras aplicações.

Com essa toolbox é possível projetar e analisar filtros. Pode-se transmitir sinais de variáveis, arquivos de dados e dispositivos de rede para o desenvolvimento e verificação do sistema. O *Time Scope*, o *Spectrum Analyzer* e o *Logic Analyzer* permitem visualizar e medir dinamicamente os sinais de transmissão. Para prototipagem e implantação de *desktop* para processadores embutidos, incluindo arquiteturas ARM Cortex, a caixa de ferramentas do sistema oferece suporte à geração de código C/C ++. Ele também suporta modelagem de ponto fixo, geração de código HDL a partir de filtros, FFT, IFFT e outros algoritmos.

- Processamento de sinal no MATLAB®.
- Processamento de sinal e blocos de álgebra linear no Simulink®.
- Design de filtros.
- *Time Scope*, *Spectrum Analyzer* e *Logic Analyzer* para visualizar e medir sinais de transmissão.
- Modelagem de ponto fixo e simulação de algoritmos de processamento de sinal Suporte para geração de código C e C ++.
- Suporte para geração de código HDL.

13 Exercícios

1. Criar uma matriz aleatória, achar a inversa, o determinante, os autovalores e autovetores.

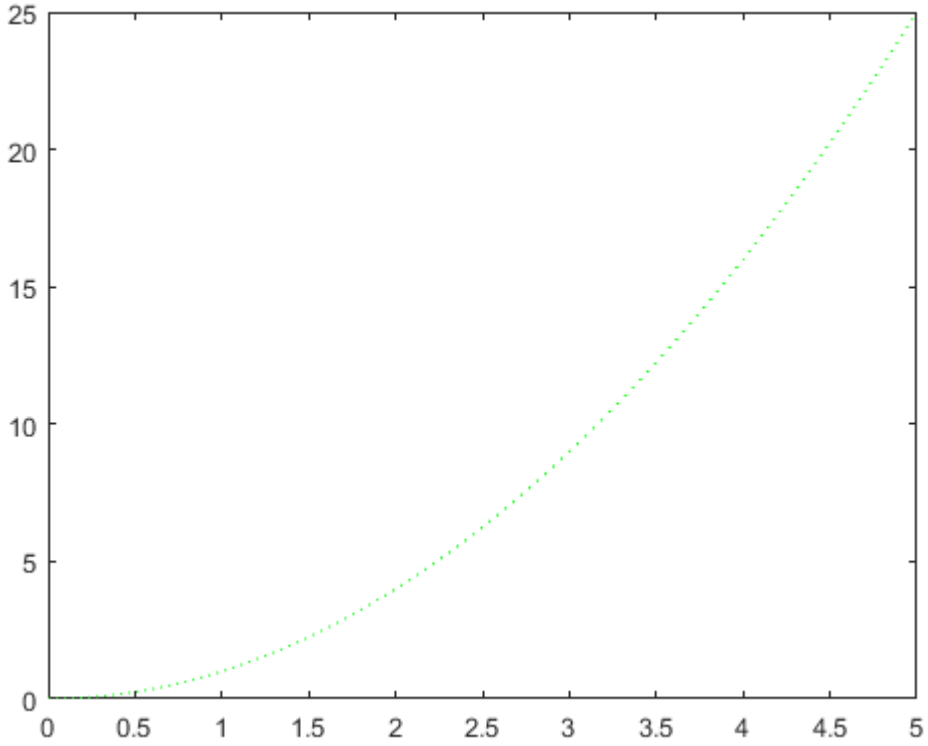
```
a = rand(3)
inv(a)
det(a)
[V, D] = eig(a)
```

2. Criar duas matrizes 2x2, calcular suas transpostas, multiplicá-las e multiplicá-las elemento a elemento (.*).

```
A = [1 2; 3 4];
B = [5 6; 7 8];
A'
B'
A*B
A.*B
```

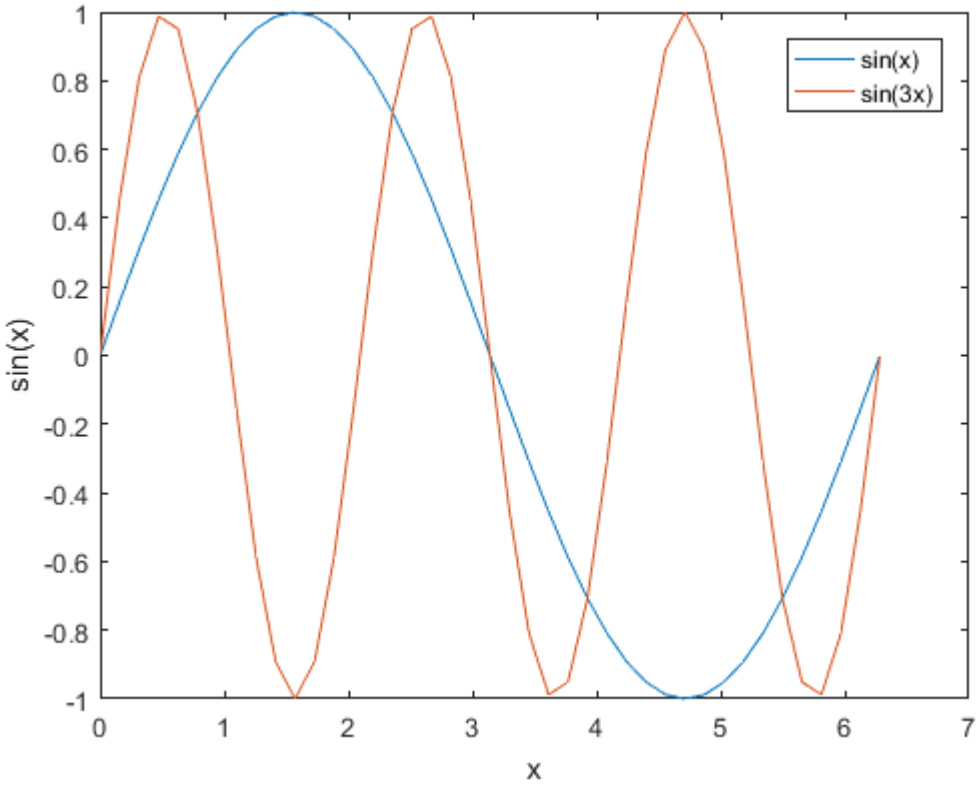
3. Plotar a função $y = x^2$ na cor verde e pontilhado para x variando de 0 até 5 com espaçamento de 0.1.

```
x = 0:0.1:5;
y = x.^2;
plot(x, y, 'g:')
```



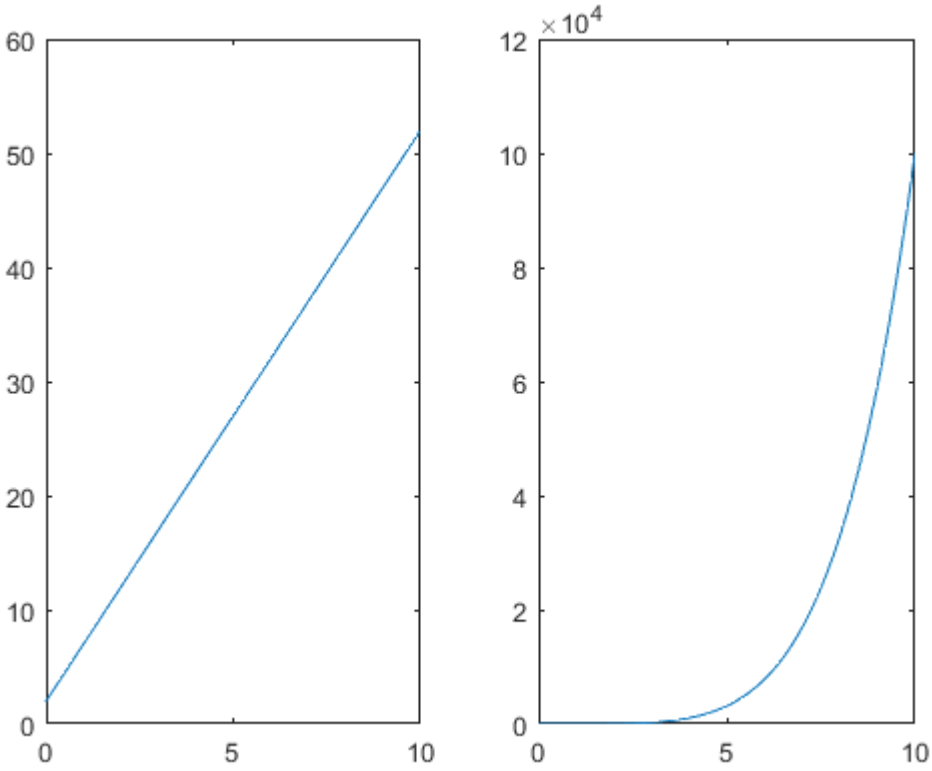
4. Plotar as funções $sin(x)$ e $sin(3x)$ na mesma janela para x variando de 0 até 2π dando nome aos eixos e colocando legenda.

```
x = 0:pi/20:2*pi;
y1 = sin(x);
y2 = sin(3*x);

plot(x, y1)
hold on
plot(x, y2)
xlabel('x')
ylabel('sin(x)')
legend('sin(x)', 'sin(3x)')
```



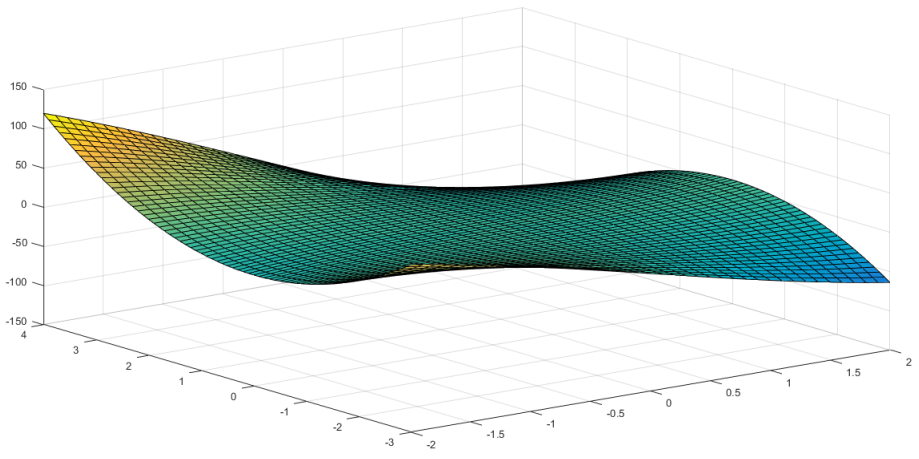
5. Plotar as funções $5x + 2$ e $x^5 + 2x$ usando o comando `subplot` para uma linha e duas colunas para x variando de 0 a 10.

```
x = 0:0.1*10;
y1 = 5.*x+2;
y2 = x.^5 + 2.*x;

subplot(1, 2, 1)
plot(x, y1)

subplot(1, 2, 2)
plot(x, y2)
```



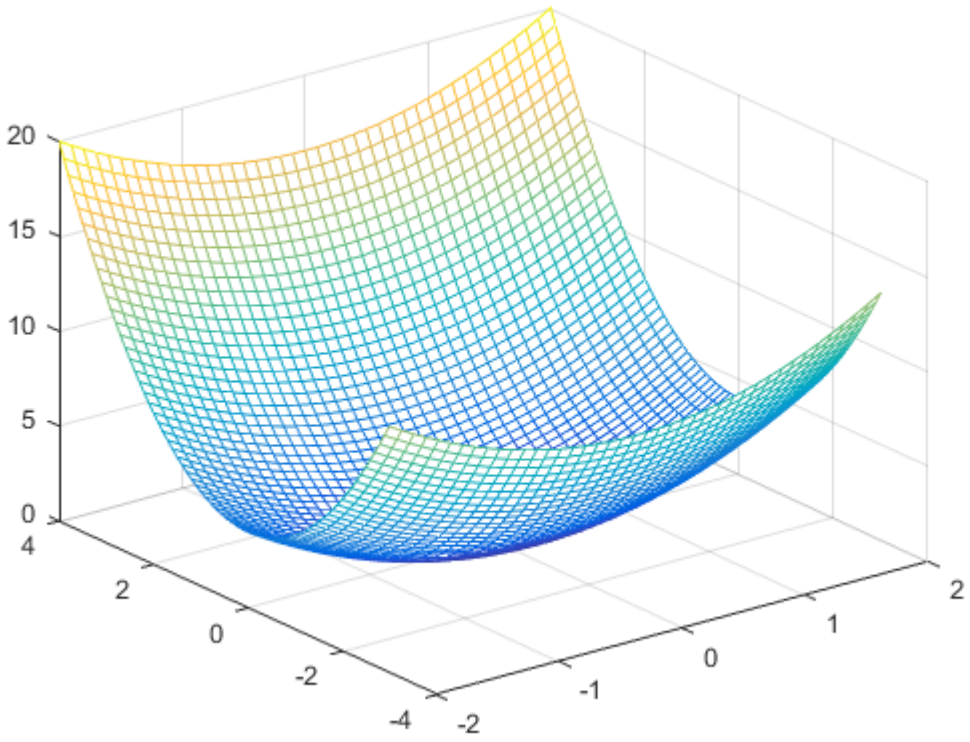
6. Fazer o gráfico 3D da função $z = f(x, y) = x^3 - 4xy^2$ em x de -2 a 2 e y de -3 a 4.

```
x = -2:0.1:2;
y = -3:0.1:4;
[X, Y] = meshgrid(x, y);
Z = X.^3 - 4*X.*Y.^2;
surf(X, Y, Z)
```



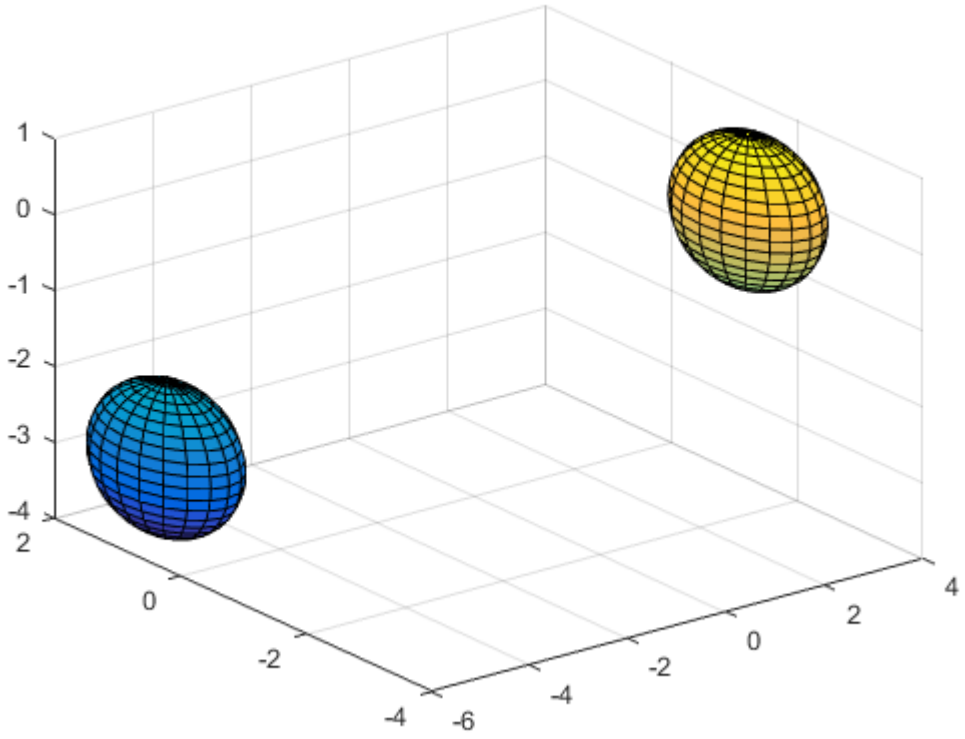
7. Fazer o gráfico da paraboloide $f(x, y) = x^2 + y^2$ usando o comando `mesh`.

```
x = -2:0.1:2;
y = -3:0.1:4;
[X, Y] = meshgrid(x, y);
mesh(X, Y, X.^2 + Y.^2)
```



8. Fazer o gráfico 3D de duas esferas, uma centrada em (3, -2, 0) e a segunda centrada em (-5, 1, 3).

```
[x, y, z] = sphere;
surf(x+3, y-2, z)
hold on
surf(x-5, y+1, z-3)
```

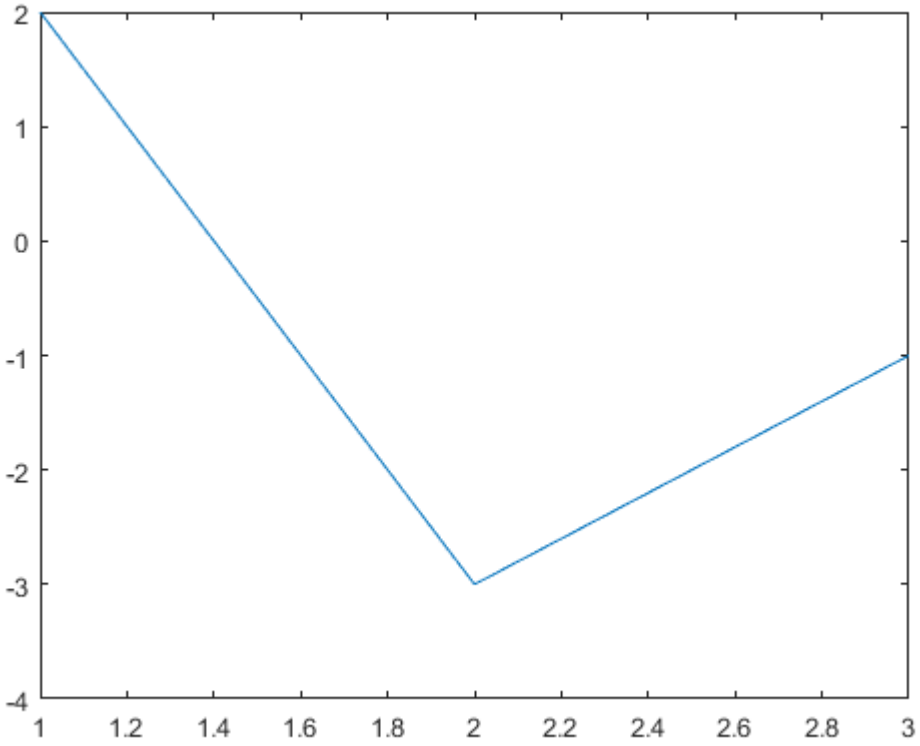


9. Encontrar uma raiz da equação $2x^2 - 3$ usando o comando `fzero` onde x é um vetor variando de 0 a 10.

```
x = 0:0.1:10;
fun = @(x) 2*(x^2) - 3;
fzero(fun, 0)

ans =
    -1.2247
```

10. Encontrar as raízes da equação $x^3 + 2x^2 - 5x - 6$ usando o comando `roots` e plotando o resultado.

```
p = [1 2 -5 -6]
roots(p)
ans =
    2.0000
   -3.0000
   -1.0000

plot(roots(p))
```



11. Criar um programa que pede para o usuário digitar dois números e criar um laço para verificar e mostrar na tela qual é o maior.

```
n1 = input('Digite o primeiro número: ');
n2 = input('Digite o segundo número: ');

if n1>n2
    fprintf('O primeiro número é maior');
elseif n1<n2
    fprintf('O segundo número é maior');
else
    fprintf('Os números são iguais');
```

12. Criar um programa usando o laço `for` para preencher duas matrizes 3x3 onde $A[i,j] = i + j$ e $B[i,j] = i - j$.

```
for i = 1:3
    for j = 1:3
        A(i, j) = i + j;
        B(i, j) = i - j;
    end
end

A
B
```

13. Criar um programa usando o laço `switch` para criar uma calculadora que pede dois números de entrada e realize as seguintes operações de acordo com a opção escolhida pelo usuário: soma, subtração, multiplicação e divisão.

```
fprintf('1: Soma 2: Subtração 3: Multiplicação 4: Divisão');
n1 = input('Digite um número: ');
n2 = input('Digite um outro número: ');
op = input('Entre a opção desejada ');

switch op
    case 1
        r = n1 + n2
    case 2
        r = n1 - n2
    case 3
        r = n1*n2
    case 4
        r = n1/n2
    otherwise
        disp('Operação inválida')
end
```

14 Referências

1. CHAPMAN, S.J. **Programação em MATLAB para Engenheiros**. 2ª ed., São Paulo, SP: Cengage Learning, 2011.

2. MATHWORKS. Learning MATLAB® 7. Release 14, 2005.

3. MATHWORKS. MATLAB® Primer. R2017a. Acesso: 15/08/2017. Disponível em: [https://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf].

4. PET Matemática UFSM. **Noções Básicas de Programação em MATLAB**. Universidade Federal de Santa Maria, Santa Maria, RS, 2010.