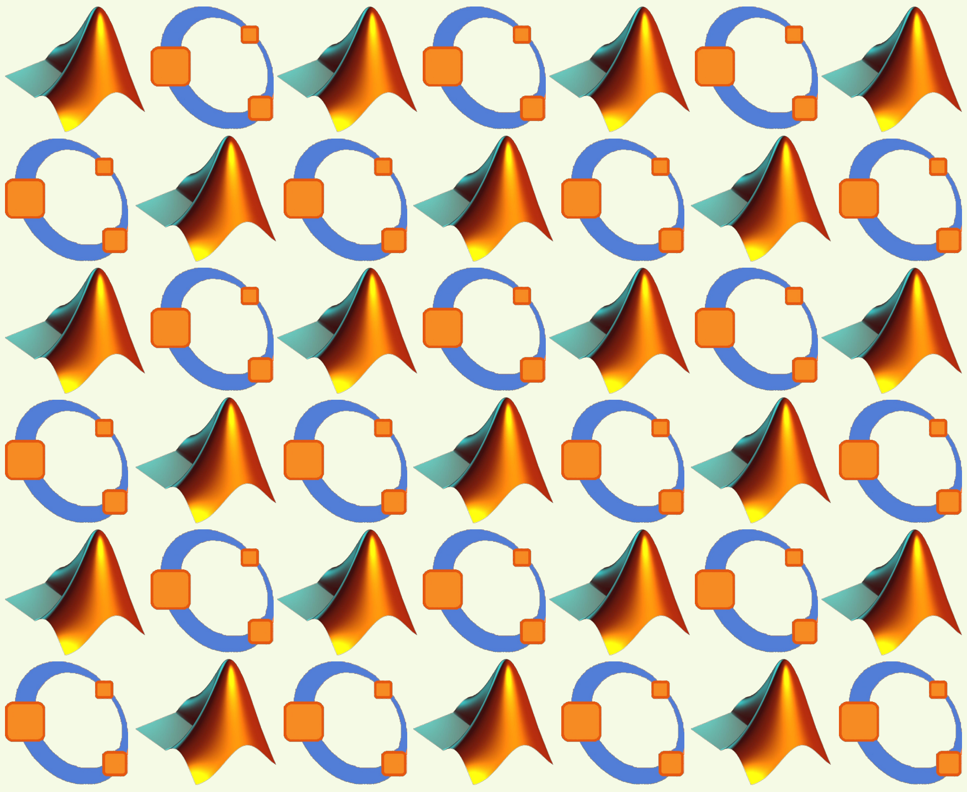


MATLAB & GNU Octave
reference guide

Alexandre de Siqueira

Programando Ciência

Alexandre Fioravante de Siqueira

MATLAB & GNU Octave: reference guide[1]

Programando Ciência

[1] Version 2.0: May 2016.

MATLAB & GNU Octave: reference guide / Alexandre Fioravante de Siqueira. Campinas: Programando Ciência, 2015
41 p.

Available at: http://www.programandociencia.com/

1. Scientific computing. 2. GNU Octave. 3. Mathworks MATLAB. I. Programando Ciencia. II. Title.

When you cite this material, please use the reference format below:

de Siqueira, A.F. MATLAB & GNU Octave: reference guide. Campinas: Programando Ciencia, 2015. Available at: http://www.programandociencia.com/.

CONTENTS

INTRODUCTION

In this introduction to the Matlab programming language, you will see some mathematical functions, variables, plots of two and three dimensions, scripts, functions and flow control expressions. The course will be based on MATLAB and GNU Octave softwares. At the end of this course, you will know how to use the language to solve basic scientific problems and you will be able of searching answers to more elaborate questions.

MATLAB is a programming language and an interactive environment aimed to numerical calculus. It contains algorithms for matrix calculations, numerical analysis, data visualization, signal and image processing, computational finance, among several others.

GNU Octave is a high level language similar to Matlab, distributed under the GNU GPL license. It is capable of numerically solve linear and non-linear problems and have tools to visualization and manipulation of data.

The source codes of the examples presented on the course are available on GitHub.

Happy coding!

1

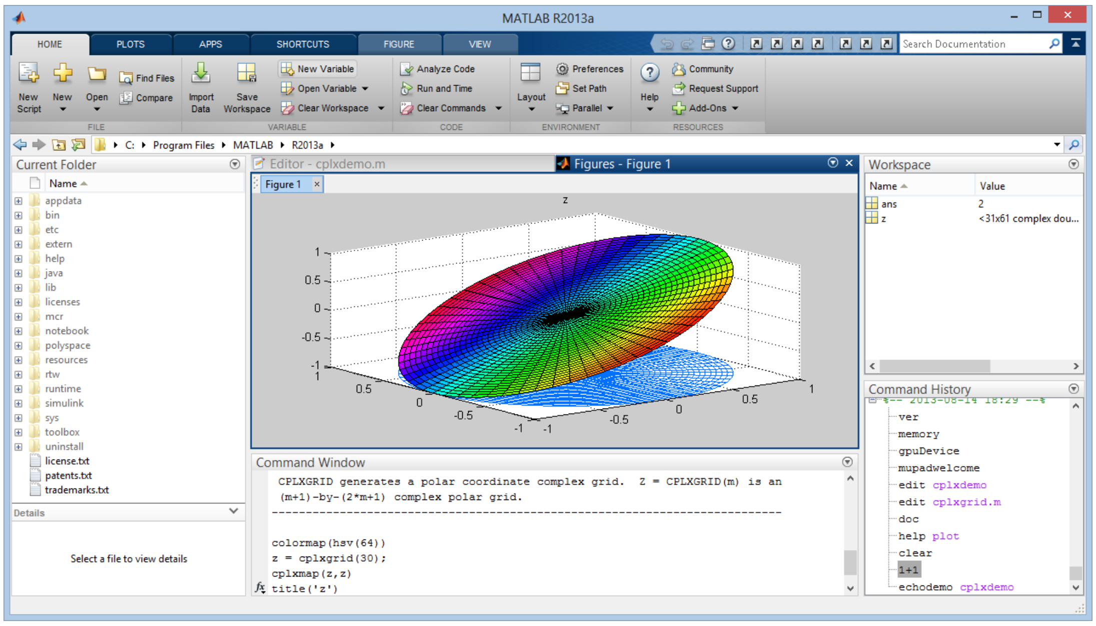
BASIC COMMANDS

In this chapter we will see:

- MATLAB and GNU Octave interfaces;
- Ways to receive and show data, and how to ask help;
- Basic mathematical functions.

1.1 MATLAB and GNU Octave interfaces

1.1.1 MATLAB graphical interface



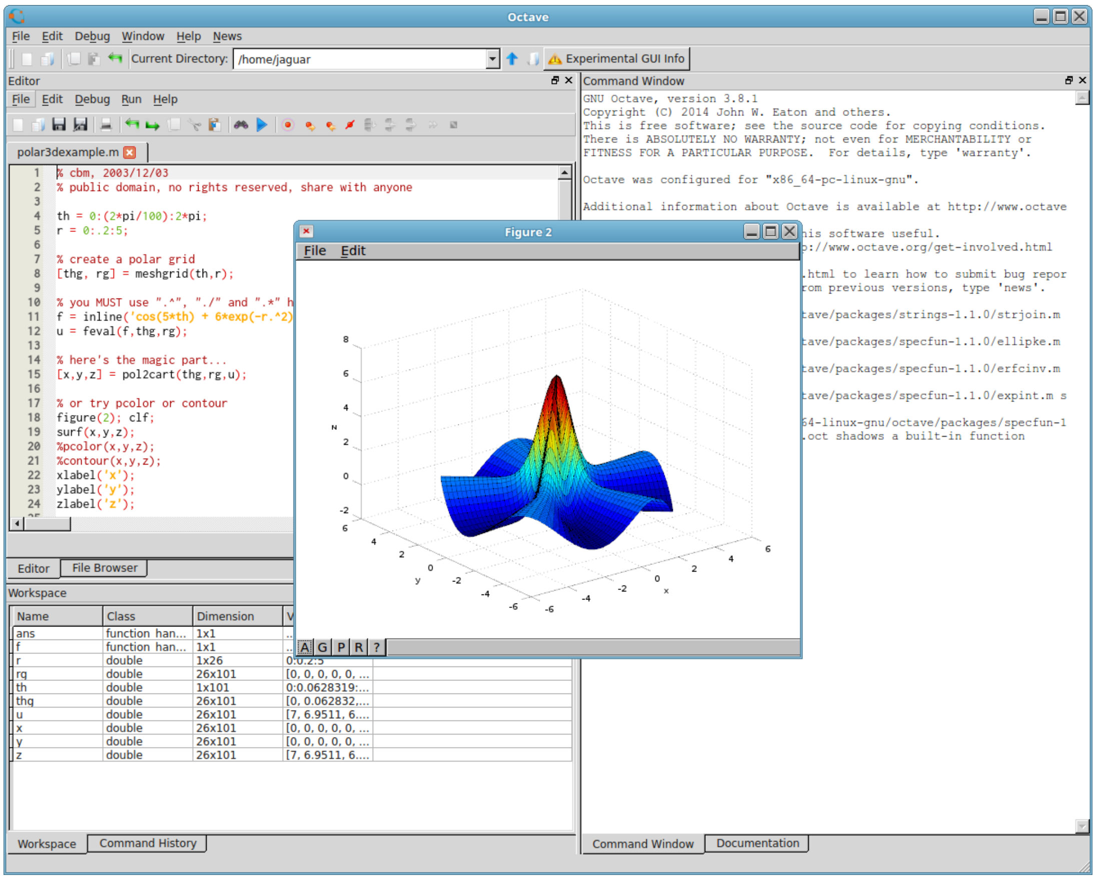
Name	Value
ans	2
z	<31x61 complex dou...

```
  CPLXGRID generates a polar coordinate complex grid.  Z = CPLXGRID(m) is an
  (m+1)-by-(2*m+1) complex polar grid.
 ----------------------------------------------------------------------------

colormap(hsv(64))
z = cplxgrid(30);
cplxmap(z,z)
title('z')
```

```
%-- 2013-08-14 18:29 --%
ver
memory
gpuDevice
mupadwelcome
edit cplxdemo
edit cplxgrid.m
doc
help plot
clear
1+1
echodemo cplxdemo
```

Figure 1 – MathWorks MATLAB interface (https://goo.gl/H2HBhg).

1.1.2 GNU Octave: graphical and text mode interfaces

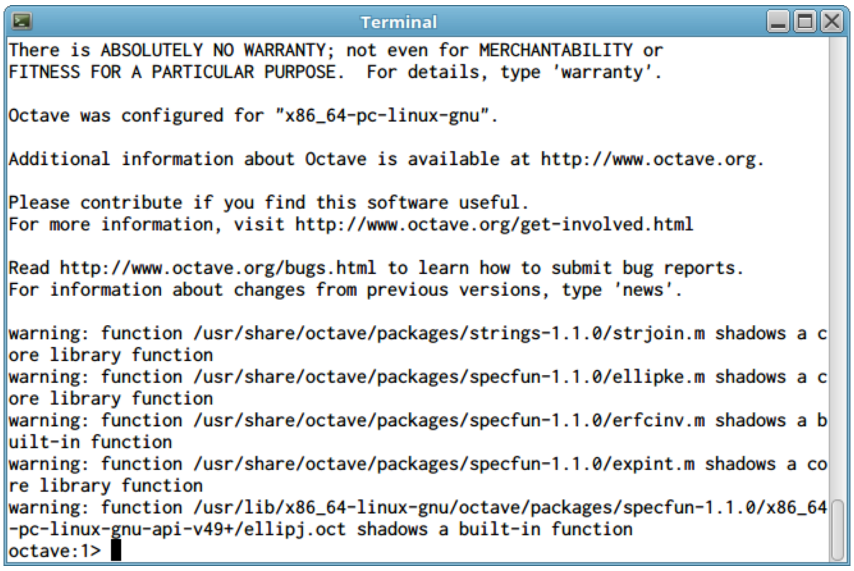


```
% cbm, 2003/12/03
% public domain, no rights reserved, share with anyone

th = 0:(2*pi/100):2*pi;
r = 0:.2:5;

% create a polar grid
[thg, rg] = meshgrid(th,r);

% you MUST use ".^", "./" and ".*" h
f = inline('cos(5*th) + 6*exp(-r.^2)
u = feval(f,thg,rg);

% here's the magic part...
[x,y,z] = pol2cart(thg,rg,u);

% or try pcolor or contour
figure(2); clf;
surf(x,y,z);
%pcolor(x,y,z);
%contour(x,y,z);
xlabel('x');
ylabel('y');
zlabel('z');
```

Name	Class	Dimension	V
ans	function han...	1x1	..
f	function han...	1x1	..
r	double	1x26	0:0.2:5
rg	double	26x101	[0, 0, 0, 0, 0, ...
th	double	1x101	0:0.0628319:...
thg	double	26x101	[0, 0.062832,...
u	double	26x101	[7, 6.9511, 6....
x	double	26x101	[0, 0, 0, 0, 0, ...
y	double	26x101	[0, 0, 0, 0, 0, ...
z	double	26x101	[7, 6.9511, 6....

```
GNU Octave, version 3.8.1
Copyright (C) 2014 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-pc-linux-gnu".

Additional information about Octave is available at http://www.octave

his software useful.
p://www.octave.org/get-involved.html

.html to learn how to submit bug repor
rom previous versions, type 'news'.

tave/packages/strings-1.1.0/strjoin.m

tave/packages/specfun-1.1.0/ellipke.m

tave/packages/specfun-1.1.0/erfcinv.m

tave/packages/specfun-1.1.0/expint.m s

64-linux-gnu/octave/packages/specfun-1
.oct shadows a built-in function
```

Figure 2 – GNU Octave graphical interface. The source code of the plot is available at http://goo.gl/W7liGB.



```
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-pc-linux-gnu".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

warning: function /usr/share/octave/packages/strings-1.1.0/strjoin.m shadows a c
ore library function
warning: function /usr/share/octave/packages/specfun-1.1.0/ellipke.m shadows a c
ore library function
warning: function /usr/share/octave/packages/specfun-1.1.0/erfcinv.m shadows a b
uilt-in function
warning: function /usr/share/octave/packages/specfun-1.1.0/expint.m shadows a co
re library function
warning: function /usr/lib/x86_64-linux-gnu/octave/packages/specfun-1.1.0/x86_64
-pc-linux-gnu-api-v49+/ellipj.oct shadows a built-in function
octave:1> █
```

Figure 3 – GNU Octave running on text mode.

1.2 The prompt

Appears on *Command Window*. It is presented by two “higher than” symbols:

```
>>
```

1.3 disp()

Shows info on the screen.

Exemplo:

```
>> disp('Hey Freiberg! Here we are!');
```

1.4 input()

Gets information from the user.

Example 1: receiving characters.

```
>> your_name = input('Tell me your name, please: ', 's');
```

Example 2: receiving numbers.

```
>> your_age = input('Now, tell me your age: ');
```

*** A semicolon omits the presentation of results.**

1.5 help

Provides information about a command.

Examples:

```
>> help disp;
>> help input;
```

1.6 Basic operators

• **Addition:**	`+`
• **Subtraction:**	`-`
• **Multiplication:**	`*`
• **Division:**	`/`
• **Potentiation:**	`power()` or ∧
• **Root extraction (square root):**	`sqrt()`

*** Use `nthroot()` for higher roots.**

Examples:

```
>> 2+4-10
>> 3*9
>> 100/(4*3)
>> sqrt(100)
>> 6^(4-2)
>> nthroot(64,3)
```

*** Use parentheses to change the order of operations.**

1.7 `factorial()`

Factorial of a natural number.

Examples:

```
>> factorial(4)
>> factorial(3)*9
>> 100/factorial(4*3)
```

1.8 `abs()`

Absolute value of a number.

Examples:

```
>> abs(37)
>> abs(-37)
```

1.9 Trigonometric functions

	Radians	Degrees
• **Sine:**	`sin()`	`sind()`
• **Cosine:**	`cos()`	`cosd()`
• **Tangent:**	`tan()`	`tand()`
• **Secant:**	`sec()`	`secd()`
• **Cosecant:**	`csc()`	`cscd()`
• **Cotangent:**	`cot()`	`cotd()`

*** Pi is given by `pi`.**

Examples:

```
>> sin(pi/6)+cos(0)
>> sind(90)+cosd(0)
>> sec(pi/4)*cot(pi/4)
>> secd(45)*cotd(90)
>> tan(csc(2*pi/3))
>> tand(csc(120))
```

1.10 Inverse trigonometric functions

	Radians	Degrees
• **Arcsine:**	`asin()`	`asind()`
• **Arccosine:**	`acos()`	`acosd()`
• **Arctangent:**	`atan()`	`atand()`
• **Arcsecant:**	`asec()`	`asecd()`
• **Arccosecant:**	`acsc()`	`acscd()`
• **Arccotangent:**	`acot()`	`acotd()`

Examples:

```
>> asin(1/2)
>> asind(1/2)
>> sec(45)*cot(90)
>> secd(45)*cotd(90)
>> tan(csc(120))
>> tand(csc(120))
```

1.11 Hyperbolic trigonometric functions

• **Hyperbolic sine:**	`sinh()`
• **Hyperbolic cosine:**	`cosh()`
• **Hyperbolic tangent:**	`tanh()`
• **Hyperbolic secant:**	`sech()`
• **Hyperbolic cosecant:**	`csch()`
• **Hyperbolic cotangent:**	`coth()`

Examples:

```
>> sinh(pi/6)+cosh(pi/6)
>> tanh(2*pi)/csch(3*pi/2)
>> sech(coth(pi/4))
```

1.12 Inverse hyperbolic trigonometric functions

• **Hyperbolic arcsine:**	`asinh()`
• **Hyperbolic arccosine:**	`acosh()`
• **Hyperbolic arctangent:**	`atanh()`
• **Hyperbolic arcsecant:**	`asech()`
• **Hyperbolic arccosecant:**	`acsch()`
• **Hyperbolic arccotangent:**	`acoth()`

Examples:

```
>> asinh(pi^3)
>> asech(pi/10)
>> acsch(pi/4)
```

1.13 Exponential and logarithm

• **Exponential:**	`exp()`
• **Natural logarithm:**	`log()`
• **Base 2 logarithm:**	`log2()`
• **Base 10 logarithm:**	`log10()`

Examples:

```
>> exp(10)
>> log(e)
>> log2(64)
>> log10(1000)
```

1.14 Complex numbers

• **Real part:**	`real()`
• **Imaginary part:**	`imag()`
• **Conjugate:**	`conj()`
• **Angle:**	`angle()`

*** The imaginary unit is given by `i` or `j`.**

Examples:

```
>> real(3+4i)
>> imag(3+4j)
>> conj(3+4i)
>> angle(3+4j)
```

2

MATRICES AND VECTORS

In this chapter we will see:

- How to create vectors and matrices;
- Special matrices;
- Advanced operations and functions.

2.1 Vectors

Use brackets to create vectors.

Examples:

```
>> vector_1 = [1 2 3];
>> vector_2 = [sin(pi) cos(pi) 0];
>> vector_3 = [2^3, 6i, e^pi];
```

*** Commas are optional.**

Use parentheses to refer to an element of the vector.

Examples:

```
>> vector_1(1)
>> vector_2(3)
>> vector_3(end)
```

*** `end` points to the last element of the vector.**

2.2 Vectors by increment

Use *start point* : *step* : *end*. If omitted, *step* receives 1.

Examples:

```
>> vector_inc1 = [1:10];
>> vector_inc2 = [-pi:0.1:pi];
>> vector_inc3 = 10:-0.1:1;
>> vector_inc4 = e^-pi:0.1:e^pi;
```

*** Brackets are optional.**

2.3 `linspace()`

Linearly spaced vector. Arguments: *start point*, *final point*, *quantity of points*.

Examples:

```
>> vector_lin1 = linspace(0,50,100);
>> vector_lin2 = linspace(-25,25,90);
>> vector_lin3 = linspace(-10,10,40);
```

*** To verify the larger length, use `length()`; the number of elements is given by `numel()`.**

2.4 `logspace()`

Logarithmically spaced vector. Arguments: *start point*, *final point*, *quantity of points*.

Examples:

```
>> vector_log1 = logspace(0,50,100);
>> vector_log2 = logspace(0,15,10);
>> vector_log3 = logspace(pi,10,40);
```

*** The arguments are received as `10^arg`, except when *final point* is equal to `pi`.**

2.5 `transpose()`

Transposition of a vector.

Examples:

```
>> vector_1 = [1 2 3];
>> transpose(vetor_1)
>> vector_2 = [sin(pi) cos(pi) 0];
>> transpose(vetor_2)
>> vector_3 = [2^3, 6i, e^pi];
>> vector_3'
```

*** For real numbers, use also the “single quote”.**

2.6 Matrizes

Use brackets to create matrices. To define new lines, use the semicolon.

Examples:

```
>> matrix_1 = [1 2 3; 4 5 6];
>> matrix_2 = [sin(pi) 0 0; 0 cos(pi) 0; 0 0 tan(pi)];
>> matrix_3 = [2^3 sin(pi/3); 6i exp(10); e^pi tanh(10)];
```

*** To verify the size of rows and columns of a matrix, use `size()`.**

To refer to an element on a matrix, use parentheses and commas.

Examples:

```
>> matrix_1(1,2)
>> matrix_2(:,end)
>> matrix_3(2,:)
>> matrix_3(:,:)
```

*** Use colons to refer to an entire row or column.**

2.7 `zeros()`

Matrix of zeroes.

Examples:

```
>> mat_zero1 = zeros(2)
>> mat_zero2 = zeros(3)
>> mat_zero3 = zeros([3 2])
>> mat_zero4 = zeros(4,3)
```

*** Brackets are optional. If they are not used, it is mandatory to use the commas.**

2.8 ones()

Matrix of ones.

Examples:

```
>> mat_one1 = ones(2)
>> mat_one2 = ones(3)
>> mat_one3 = ones([3 2])
>> mat_one4 = ones(4,3)
```

2.9 eye()

Identity matrix.

Examples:

```
>> mat_id1 = eye(2)
>> mat_id2 = eye(3)
>> mat_id3 = eye([3 2])
>> mat_id4 = eye(4,3)
```

2.10 diag()

Diagonal matrix, defined by an argument vector.

Examples:

```
>> mat_diag1 = diag([1 2 3])
>> mat_diag2 = diag(4,3)
>> mat_diag3 = diag([1 1],2)
>> mat_diag4 = diag([3 2 3],-1)
```

*** The argument in front of the vector indicates how many rows will be inserted *before* (negative number) or *after* (positive number) the diagonal.**

2.11 rand()

Matrix with random numbers between zero and 1.

Examples:

```
>> mat_ran1 = rand(1,10)
>> mat_ran2 = rand(4,3)
>> mat_ran3 = rand('seed',1)
>> mat_ran4 = rand(3,2)
```

*** The argument `'seed',1` ensures that the same numbers can be generated again.**

2.12 Operations

We will consider an element as a number, vector, or matrix. Valid operations, except for mathematical rules:

• **Adition:**	`+`
• **Subtraction:**	`-`
• **Multiplication dot to dot:**	`.*`
• **Multiplication between elements:**	`*`
• **Division dot to dot:**	`./`
• **Division between elements:**	`/`
• **Power dot to dot:**	`.∧`
• **Power between elements:**	`∧`

*** Power between elements does not accept vector/matrix ∧ vector/matrix!**

Example 1: addition and subtraction.

```
>> vec_A = [2*pi -factorial(3) e];
>> vec_B = [factorial(4) pi/2 cos(pi)];
>> vec_A+vec_B
>> vec_A-vec_B
```

Example 2: multiplication.

```
>> mat_C = [sin(pi/6) cos(pi/6); cos(pi/6) sin(pi/6)];
>> mat_D = [sin(pi/3) cos(pi/3); cos(pi/3) sin(pi/3)];
>> 3.*mat_C
>> mat_C.*mat_D
>> mat_C*mat_D
>> vec_lin = [sin(pi/3) cos(pi/3) 0];
>> vec_col = [sin(pi/6); cos(pi/6); 0];
>> 10.*vec_col
>> vec_lin*vec_col
>> vec_col*vec_lin
```

Example 3: division.

```
>> vec_E = [linspace(-1,-0.5,3); linspace(-0.5,0,3); linspace(0,0.5,3)];
>> vec_E./0.1
>> 0.1./vec_E
>> mat_F = [1 2 3; 4 5 6; 7 8 9];
>> mat_F'/mat_F
```

Example 4: power.

```
>> vec_G = [exp(3); factorial(3); sin(pi/6)]
>> vec_G.^3
>> 2.^vec_G
>> mat_H = [1 3; 5 7];
>> mat_H^2
>> e^mat_H
```

2.13 Advanced functions

Related to Linear Algebra.

• **Trace:**	`trace()`
• **Determinant:**	`det()`
• **Inverse matrix:**	`inv()`
• **Eigenvalues and eigenvectors:**	`eig()`
• **Solving linear systems:**	`X = A\B`, for `A*X = B`.

Example 1: trace.

```
>> trace(mat_C)
>> trace(mat_H')
```

Example 2: determinant.

```
>> det(mat_D)
>> det(mat_F)
```

```
%% vander(): Vandermonde matrix.
%% More in:
% http://mathworld.wolfram.com/VandermondeMatrix.html
% http://www.mathworks.com/help/matlab/ref/vander.html
>> ent = 1:5;
>> mat_I = vander(ent);
>> det(mat_I)
```

Example 3: inverse matrix.

```
>> inv(mat_C)
>> inv(mat_D)
>> mat_J = [1 0 -3; 2 5 0; -2 8 -6];
>> inv(mat_J)
```

Example 4: eigenvalues and eigenvectors.

```
>> [avet, aval] = eig(mat_C)
>> aval = eig(mat_D)
>> [avet, aval] = eig(mat_I)
```

Example 5: linear systems.

```
>> A = [1 0; -2 3];
>> b = [3 -1]'
>> x = A\b
>> A*x
>> k = [-1 0 5]';
>> x2 = mat_J\k
```

3

TWO-DIMENSIONAL PLOTS

In this chapter we will see:

- Simple plots and customization options;
- Types of two-dimensional plots;
- How to plot several graphs in one window.

3.1 plot()

The swiss army knife for two-dimensional plots.

Examples:

```
>> x = linspace(-pi,pi,250);
>> plot(x,-x.^2)
>> plot(x,exp(x).*sec(x))
```

3.2 Labels

Title:	`title()`
X axis:	`xlabel()`
Y axis:	`ylabel()`

Examples:

```
>> x = -2:0.1:2;
>> plot(x,x.^2)
>> title('2D plot.')
>> xlabel('x')
>> ylabel('x^2')
```

3.3 legend()

Legends. Their location is given by the argument `'location'`:

- **Above:**
  - * **Right:** `'northeast'`
  - * **Center:** `'north'`
  - * **Left:** `'northwest'`
- **Center:**
  - * **Right:** `'east'`
  - * **Left:** `'west'`
- **Below:**
  - * **Right:** `'southeast'`
  - * **Center:** `'south'`
  - * **Left:** `'southwest'`

Examples:

```
>> x = -2:0.1:2;
>> plot(x,x.^2)
>> legend('x^2','location','northeast')
>> legend('boxoff')
>> figure;
>> plot(x,x.^3)
>> legend('x^3','location','southwest')
```

*** `legend('boxoff')` removes the legend box, while `legend('boxon')` restores it; `figure` opens a new Figure window.**

The text position related to the legend can be changed using `legend('left')` and `legend('right')`.

3.4 axis()

Sets the area where the plot is presented.

Example:

```
>> x = linspace(-5,5,250);
>> plot(x,1./x)
>> axis([-2 2 -2 2])
```

*** There is also `axis('off')`, which removes the plot axes. To reactivate the axes, use `axis('on')`.**

3.5 `grid`/`grid minor`

Inserts major and minor grids. Remove them with `grid off`).

Example:

```
>> x = linspace(-5,5,250);
>> plot(x,1./x)
>> grid on
>> grid minor on
>> grid off
```

3.6 Options for lines

• **Solid:**	`'-'`
• **Dots:**	`':'`
• **Dashes:**	`'--'`
• **Dashes and dots:**	`'-.'`

Examples:

```
>> x = -2:0.1:2;
>> plot(x,-x.^2,'--')
>> plot(x,sec(x),'-.')
>> plot(x,1./x,':')
```

3.7 Options for symbols

- **Ball:** 'o'
- **Cross:** 'x'
- **Star:** '*'
- **Hexagram:** 'h'
- **Diamond:** 'd'
- **More:** '+'
- **Pentagram:** 'p'
- **Dot:** '.'
- **Square:** 's'
- **Triangle pointing:**
  * **Down:** 'v'
  * **Up:** '∧'
  * **Right:** '>'
  * **Left:** '<'

Examples:

```
>> x = -5:0.1:5;
>> plot(x,-1./x,'d')
>> plot(x,-x.^2,'*')
>> plot(x,sec(x),'o')
```

3.8 Options for colors

- **Blue:** 'b'
- **Yellow:** 'y'
- **White:** 'w'
- **Cyan:** 'c'
- **Magenta:** 'm'
- **Green:** 'g'
- **Red:** 'r'

Example:

```
>> x = -5:0.1:5;
>> plot(x,-1./x,'y')
>> plot(x,-x.^2,'c')
>> plot(x,sec(x),'r')
```

*** You can use options for lines/symbols and colors together: `'*r'`, `'-.g'`.**

3.9 Arguments

The corresponding value is separated from the argument by a comma.

- **Line width:** `'linewidth'`
- **Symbol color:** `'markerfacecolor'`
- **Symbol size:** `'markersize'`

Example:

```
>> x = -5:0.1:5;
>> plot(x,-1./x,'linewidth',3)
>> plot(x,-x.^2,'*','markerfacecolor','r')
>> plot(x,sec(x),'d','markersize',3)
```

3.10 Several plots, one window

- **Same area:**
  - **plot():** `plot(x1,y1,'arg1',x2,y2,'arg2',...)`
  - **hold:** `hold on / hold off`
- **Different areas:**
  - **subplot():** `subplot(lin,col,pos)`

Example 1: using `plot()`.

```
>> x = -5:0.05:5;
>> plot(x,-1./x,x,-x.^2,x,exp(x),x,sec(x))
```

Example 2: using `hold`.

```
>> x = -5:0.05:5;
>> hold on;
>> plot(x,-1./x)
>> plot(x,-x.^2)
>> plot(x,exp(x))
>> plot(x,sec(x))
>> hold off;
```

*** Use `hold all` instead of `hold on`; then, the arguments of the previous plot will be different from the next plot.**

Example 3: using `subplot()`.

```
>> x = -5:0.05:5;
>> subplot(221)
>> plot(x,-1./x)
>> subplot(222)
>> plot(x,-x.^2)
>> subplot(223)
>> plot(x,exp(x))
>> subplot(224)
>> plot(x,sec(x))
```

3.11 Another types of plots available

• **Bars:**	`bar()`
• **Dispersion:**	`scatter()`
• **Stairs:**	`stairs()`
• **Histogram:**	`hist()`
• **Discrete points:**	`stem()`
• **Pie:**	`pie()`

Examples:

```
>> mat_rand = 100*rand(3,4);
>> bar(mat_rand)  %% bars
>> vet_rand1 = 100*rand(1,50);
>> vet_rand2 = 100*rand(1,50);
>> scatter(vet_rand1,vet_rand2)  %% dispersion
>> stairs(vet_rand1)  %% stairs
>> hist(vet_rand1)  %% histogram
>> stem(vet_rand1)  %% discrete points
>> vet_rand3 = 100*rand(1,6);
>> pie(vet_rand3)  %% pie
```

*** Each type of graphic have several specific options. For more information, check** http://goo.gl/TRzJmK **and** http://www.mathworks.com/help/matlab/2-and-3d-plots.html**.**

4

THREE-DIMENSIONAL PLOTS

In this chapter we will see:

- Types of three-dimensional plots.
- How to use vectors to obtain meshs.

*** All customizations seen on chapter 3 are suitable with 3D plots. Besides, a new label is available for the Z axis: `zlabel()`.**

4.1 `plot3()`

The swiss army knife for three-dimensional plots.

Examples:

```
>> x = -2*pi:0.1:2*pi;
>> plot3(x,sin(exp(x)))
>> plot3(x,sin(x).^2,cos(x),x,cos(x).^2,sin(x));
>> figure;
>> y = 1:0.1:e;
>> plot3(y,y.^2,exp(y),y,y.^3,log(y));
```

4.2 `meshgrid()`

Returns a two-dimensional matrix (grid or mesh), created by copying a vector several times.

Examples:

```
>> x1 = [1 2 3 4 5];
>> meshgrid(x1)
>> y1 = [5 4 3 2 1];
>> [xx1, yy1] = meshgrid(x1,y1)
>> x2 = y2 = 0:0.1:pi;
>> [xx2, yy2] = meshgrid(x2,y2)
```

4.3 Another types of plots available

• **Contours:**	`contour()`
• **Filled contours:**	`contourf()`
• **Dispersion:**	`scatter3()`
• **Mesh:**	`mesh()`
• **Mesh with contours on X & Y plane:**	`meshc()`
• **Surface:**	`surf()`
• **Surface with contours on X & Y plane:**	`surfc()`

Example 1: contour and filled contour.

```
>> linx1 = 0:0.1:10;
>> contour(linx1'*linx1)
>> contourf(linx1'*linx1)
>> liny1 = -10:0.1:10;
>> contour(linx1'*-liny1)
>> contourf(linx1'*-liny1)
```

Example 2: dispersion.

```
>> randx1 = rand(1,100);
>> randy1 = rand(1,100);
>> randz1 = randx1+randy1;
>> scatter3(randx1,randy1,randz1,'filled')
>> randz2 = rand(1,numel(linx1));
>> scatter3(linx1,liny1,randz2)
```

Example 3: mesh and mesh with contours.

```
>> linx2 = liny2 = -pi:0.1:pi;
>> [xx, yy] = meshgrid(linx2,liny2);
>> mesh(xx,yy,xx.^2+yy.^2)
>> meshc(xx,yy,xx.^2+yy.^2)
>> contourf(xx,yy,xx.^2+yy.^2)
>> figure;
>> mesh(xx,yy,sin(xx)+cos(yy))
>> meshc(xx,yy,sin(xx)+cos(yy))
>> contourf(xx,yy,sin(xx)+cos(yy))
```

Example 4: surface and surface with contours.

```
>> surf(xx,yy,xx.^2+yy.^2)
>> surfc(xx,yy,xx.^2+yy.^2)
>> figure;
>> surf(xx,yy,sin(xx)+cos(yy))
>> surfc(xx,yy,sin(xx)+cos(yy))
```

*** Each type of plot have several specific options. For more information, check** http://goo.gl/Uc82oi **and** http://www.mathworks.com/help/matlab/2-and-3d-plots.html**.**

5

SCRIPTS AND FUNCTIONS

In this chapter we will see:

- How to create scripts and functions.
- How to comment code.

5.1 Scripts

A file with the '.m' extension which contains code.

5.2 Comments

Code or text line not executed, indicated by %.

Example:

```
% this line is not executed by the interpreter.
% nor this.
```

5.3 Block of comments

Several lines of text or code not executed, started by %{ and finished by %}.

Examples:

```
%{
 all these lines will not be executed:
 - not this.
```

```
     - not this too.
     - nor this.
     - not even this.
    %}
```

5.4 Functions

A piece of code which receives and returns variables.

Basic form:

```
function [out1,out2,out3,...] = nome_func(in1,in2,in3,...)
    command_block;
end
```

Example 1: função `quadraticdelta()`.

```
function delta = quadraticdelta(vala,valb,valc)
    %{
     QUADRATICDELTA receives the coefficients A, B and C
     (vala, valb and valc) from a quadratic equation and
     returns its discriminant (delta).
    %}

    delta = valb^2-4*vala*valc;
end
```

*** Variables created inside the function are *local*, and they are known only at the body of the function!**

Example 2: function `cubicdelta()`.

```
function delta = cubicdelta(vala,valb,valc,vald)
    %{
     CUBICDELTA receives the coefficients A, B, C e D (vala,
     valb, valc and vald) from a cubic equation and returns
     its discriminant (delta).
    %}

    delta = 18*vala*valb*valc*vald ...
            -4*valb^3*vald+valb^2*valc^2 ...
            -4*vala*valc^3-27*vala^2*vald^2;
end
```

*** Ellipses indicates that the command continues on the next line.**

Example 3: função `infocone()`.

```
function [area_lat,area_sup,vol] = infocone(radius,height)
    %{
     INFOCONE receives radius and height from a cone and returns
     its lateral area (area_lat), surface area (area_sup) and
     volume (vol).
    %}

    area_lat = pi*radius*sqrt(height^2+radius^2);
    disp('The cone lateral area is '); disp(area_lat);

    area_sup = pi*radius*(radius+sqrt(height^2+radius^2));
    disp('The cone surface area is '); disp(area_sup);

    vol = pi*radius^2*(height/3);
    disp('The cone volume is '); disp(vol);
end
```

6

FLOW CONTROL

In this chapter we will see:

- Boolean and relational operators.
- Flow control: conditional and repetition structures.

6.1 Operators

Operators indicate a relationship between elements. They return 1 when true, and zero when false.

6.2 Comparison operators

• **Equal:**	==
• **Different:**	=
• **Less than:**	<
• **Less or equal:**	<=
• **Greater than:**	>
• **Greater or equal:**	>=

Examples:

```
>> a = 2^2
>> b = 3^2
>> a == b
>> a != b
>> a < 4
>> sqrt(9) <= b
>> a > 10.^b
>> factorial(a) >= b
```

6.3 Boolean operators

• **AND:**	`&&`
• **OR:**	`||`
• **EXCLUSIVE OR:**	`xor()`
• **NOT:**	`~`

Examples:

```
>> a = 2^2; b = 3^2;
>> (a+5 == b) && (~a != b)
>> xor(a.^2 == b.^2, 3 > 2)
>> ~(~a || ~b)
```

6.4 Control structures

Coordinates the program execution.

6.5 Conditional structures

- **if:**

```
if(condition)
    commands1;
elseif
    commands2;
else
    commands3;
end
```

- **switch:**

```
switch(condition)
    case (case1)
        commands1;
    case (case2)
        commands2;
    case (case3)
        commands3;
    (...)
    otherwise
        commandsn;
end
```

Example 1: if.

```
lower = 21;
higher = 80;
age = input('Type your age: ');
if age < lower
    disp('Do not drink kid!');
elseif age > higher
    disp('Do not drive sir!');
else
    disp('You can do everything dude! Except for drive and drink!')
end
```

Example 2: switch.

```
idade = input('Type your age: ');
switch idade
    case num2cell([0:17])
        disp('Do not drink kid!');
    case num2cell([70:100])
        disp('Do not drive sir!');
    otherwise
        disp('You can do everything dude! Except for drive and drink!')
end
```

6.6 Repetition structures

- **for:**

```
for (var = interval)
    commands;
end
```

- **while:**

```
while (condition)
    commands;
end
```

Example 1: for.

```
fibo = ones(1, 20);
for term = 3:numel(ones)
    fibo(term) = fibo(term-1) + fibo(term-2);
end
```

Example 2: while.

```
term = 100;
total = 0;
iter = 1;
while (iter != term)
    total = total + (1/(iter.^2))  % equivalent to (pi^2)/6
    iter++
end
```

REFERENCES

Keep learning my friend.

- **MATLAB documentation:**

http://www.mathworks.com/help/matlab/index.html

- **MATLAB extension packages:**

http://www.mathworks.com/help/

- **GNU Octave documentation:**

http://www.gnu.org/software/octave/doc/interpreter/

- **GNU Octave extension packages:**

http://octave.sourceforge.net/

- **Questions about MATLAB and Octave on StackOverflow:**

http://stackoverflow.com/questions/tagged/matlab

http://stackoverflow.com/questions/tagged/octave

About the author

Alexandre Fioravante de Siqueira

* Alex is a Mathematician licensed by FCT - Unesp, Presidente Prudente, Brazil.
* Is a doctor in Materials Science and Technology, also by FCT.
* Works since 2005 with digital image processing using MATLAB and GNU Octave.
* Writes about scientific computation every week on *Programando Ciência*.

*** Contact:** http://www.programandociencia.com/about/