

4300331

Métodos Computacionais em Física

Prof. Luis Gregório Dias da Silva

Depto. Física Materiais e Mecânica – IF – USP

Ed. Alessandro Volta, bloco C, sala 214

luisdias@if.usp.br

Página do curso ([Stoa -> Cursos -> IF -> 430 -> 4300331](#))

Avisos via Twitter: <https://twitter.com/ProfLuisDias>

Objetivos da disciplina

- Apresentar *métodos computacionais* utilizados para a *simulação de sistemas físicos* e resolver numericamente os problemas que surgem em física, astronomia, engenharias, bem como em outras áreas afins.

- Método-base: aprendizado indutivo.

Utilização da solução de **problemas** na construção do aprendizado.

A necessidade nos leva à assimilação de conceitos e à construção de novos conhecimentos!

- A pesquisa científica funciona desta forma!
-

Conceitos básicos

Algoritmo vs Programa (ou “código”):

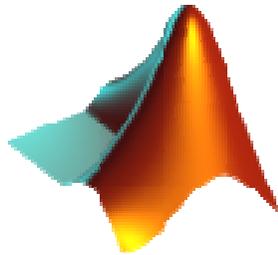
- **Algoritmo:** processo passo-a-passo para realizar uma tarefa (cálculo, processo, etc.)
- **Programa:** uma série de instruções ou comandos passados a uma unidade de processamento (computador, tablet, smartphone, etc...)
- Um algoritmo não necessita necessariamente de um programa!

Exemplos: Como chegar à Ala I? → algoritmo.

Multiplique $2344 \times 435 = ?$ → algoritmo. Fatore 432 → algoritmo.

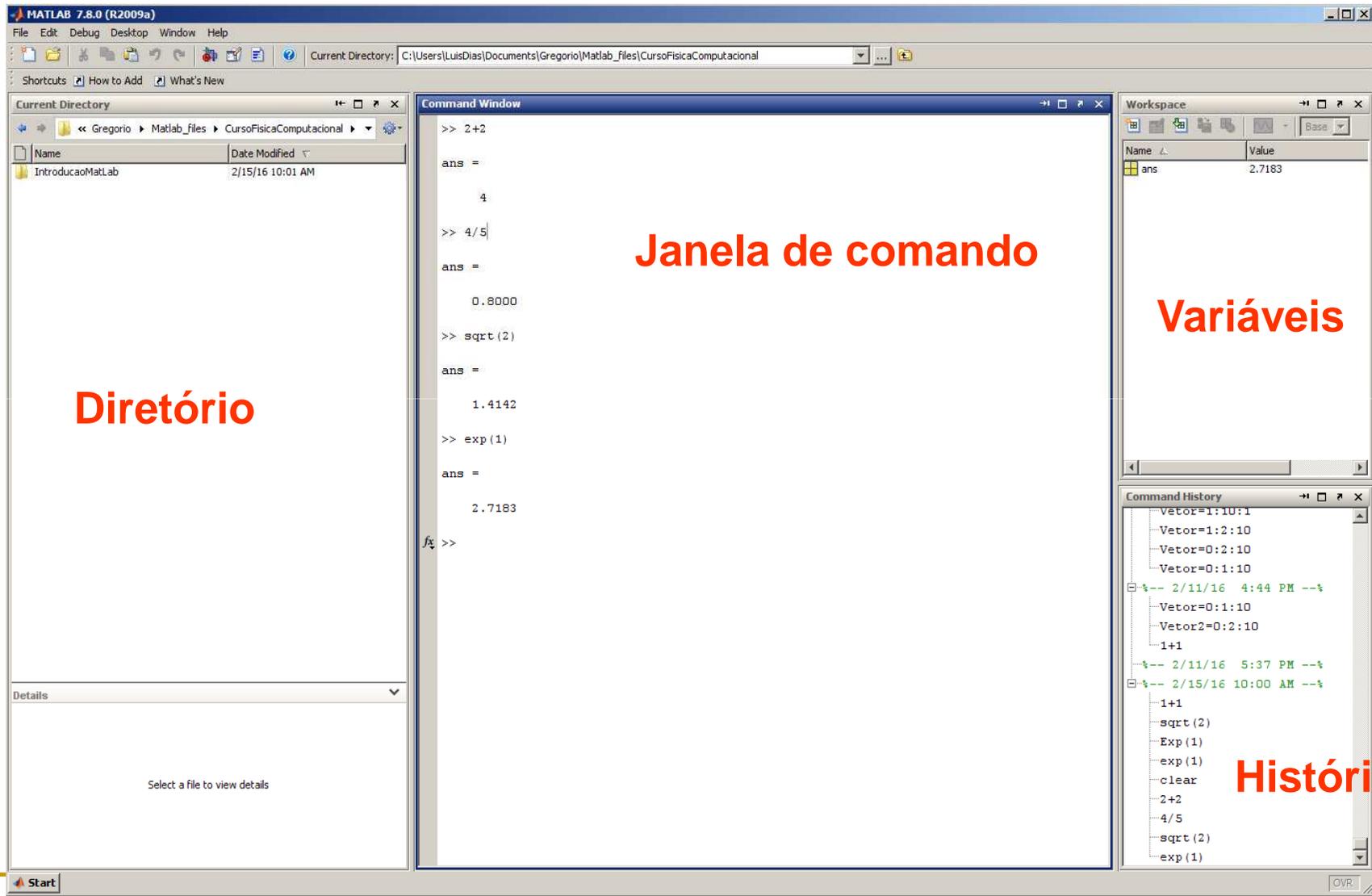
Introdução ao MatLab

O que é MatLab?

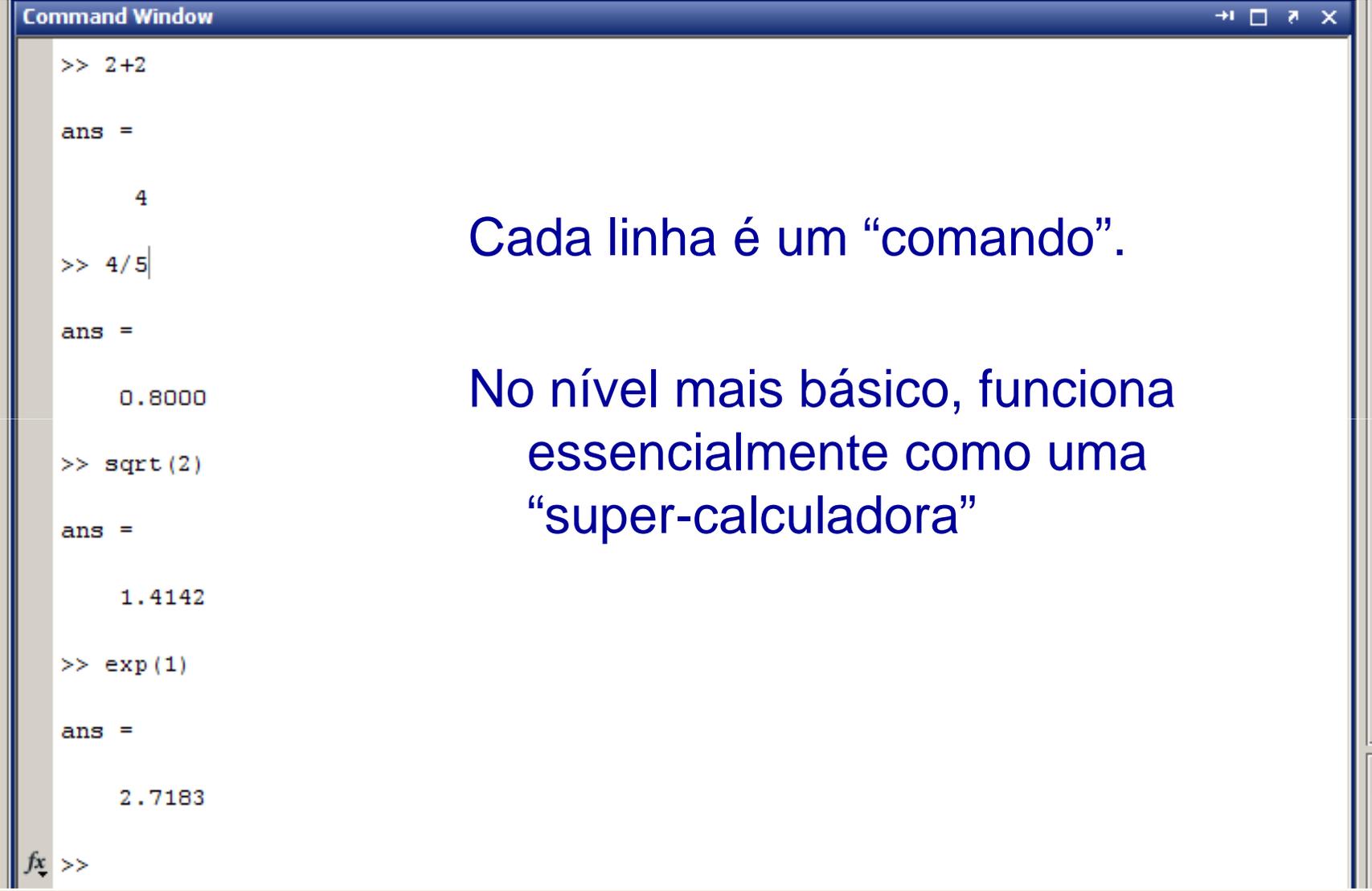


- MatLab (=“Matrix laboratory”) é um *ambiente* para computação científica e técnica desenvolvido pela empresa MathWorks.
 - O ambiente integra uma **linguagem de programação** (de alto nível, tipo script) e **ferramentas de visualização** (gráficos, etc.) além de **pacotes computacionais** pré-instalados (bibliotecas, pacotes de manipulação simbólica, etc.).
-

Abrindo o MatLab: Elementos básicos



Básico do MatLab: Janela de comando

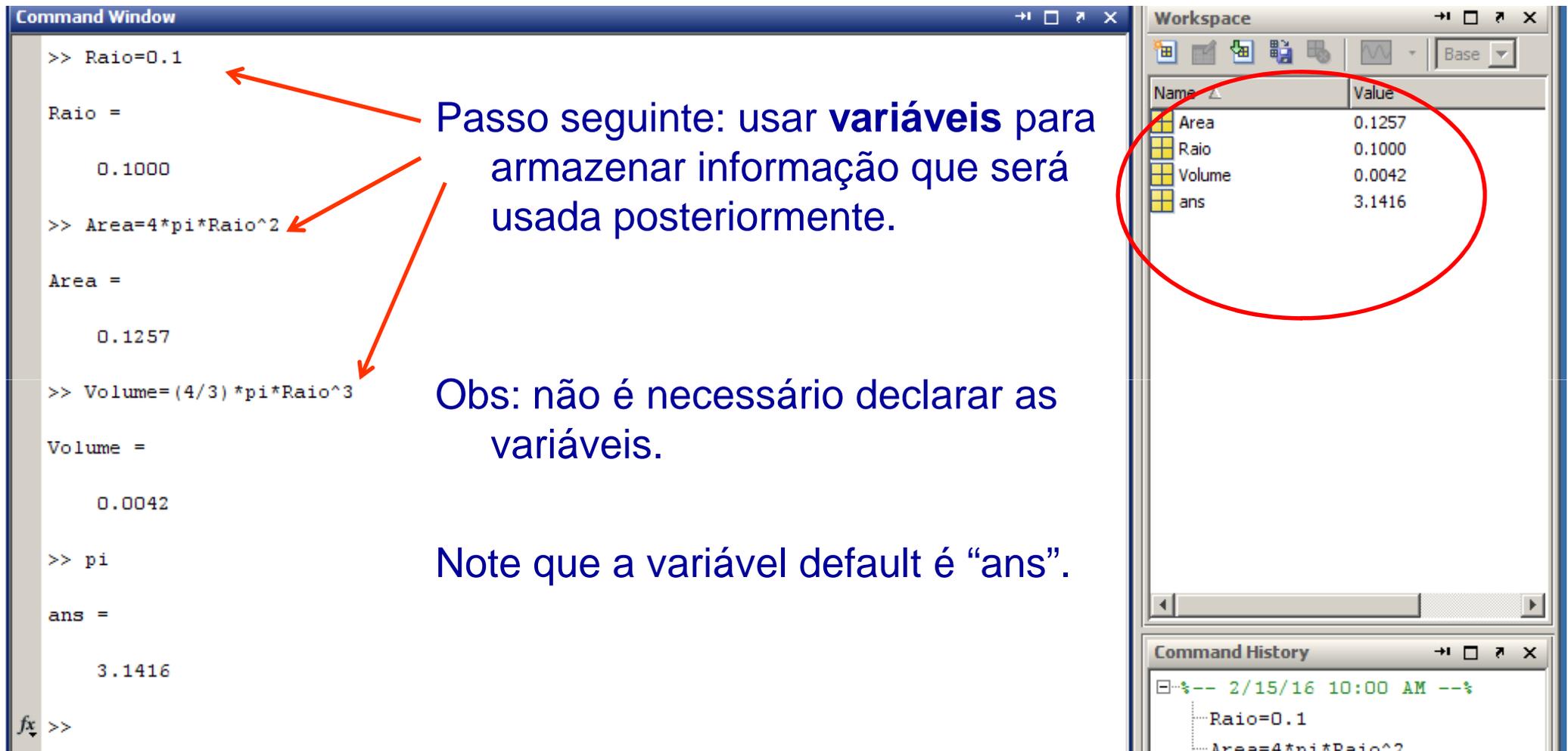


```
Command Window
>> 2+2
ans =
    4
>> 4/5
ans =
    0.8000
>> sqrt(2)
ans =
    1.4142
>> exp(1)
ans =
    2.7183
fx >>
```

Cada linha é um “comando”.

No nível mais básico, funciona essencialmente como uma “super-calculadora”

Básico do MatLab: Variáveis



The image shows a MATLAB Command Window on the left and a Workspace window on the right. The Command Window contains the following code and output:

```
>> Raio=0.1
Raio =
    0.1000
>> Area=4*pi*Raio^2
Area =
    0.1257
>> Volume=(4/3)*pi*Raio^3
Volume =
    0.0042
>> pi
ans =
    3.1416
fx >>
```

The Workspace window shows the following variables and their values:

Name	Value
Area	0.1257
Raio	0.1000
Volume	0.0042
ans	3.1416

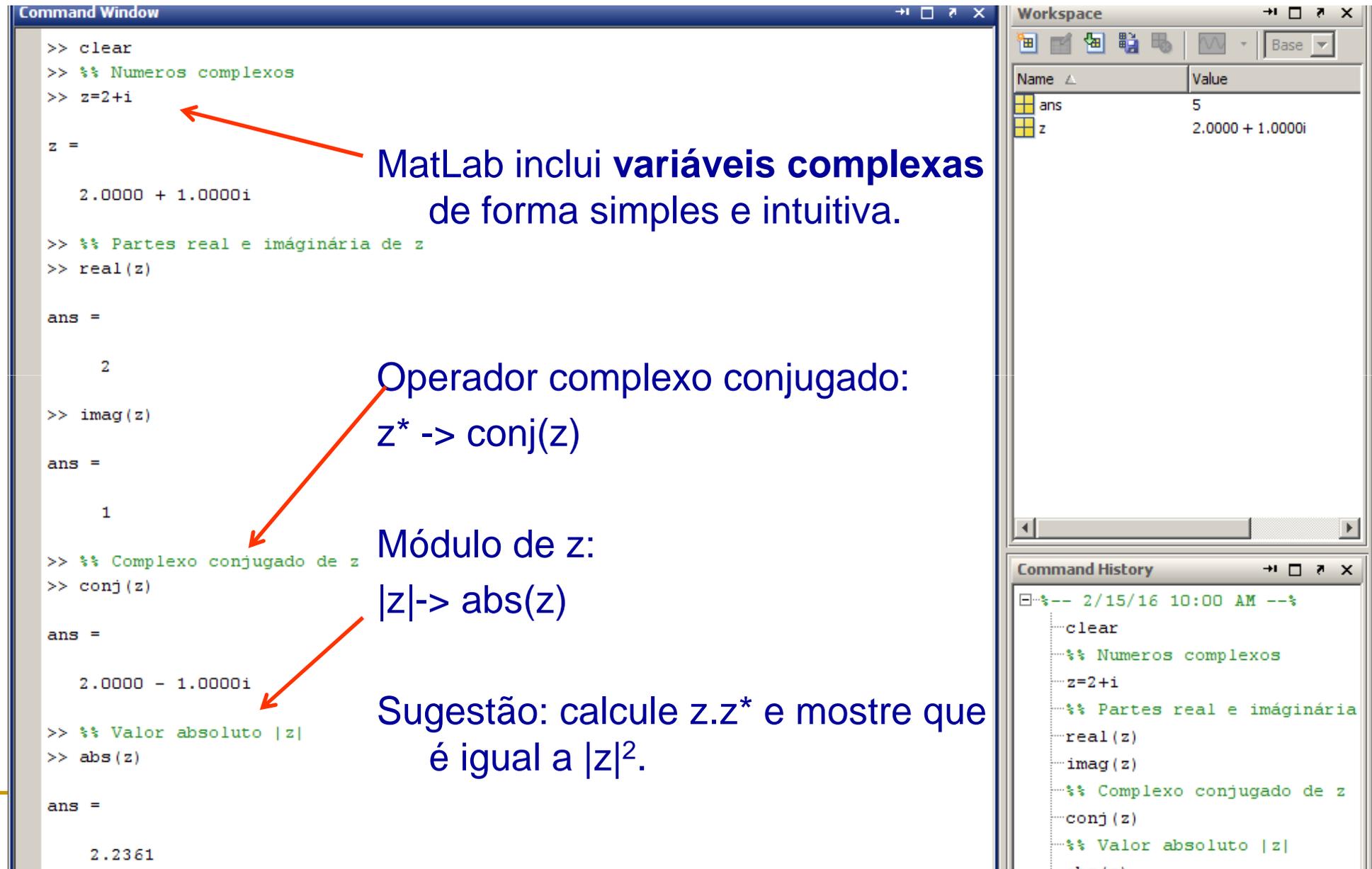
Annotations in the image include red arrows pointing from the text to the corresponding code lines in the Command Window. A red circle highlights the Workspace window.

Passo seguinte: usar **variáveis** para armazenar informação que será usada posteriormente.

Obs: não é necessário declarar as variáveis.

Note que a variável default é “ans”.

Básico do MatLab: Variáveis complexas



The screenshot shows the MATLAB Command Window and Workspace. The Command Window contains the following code and output:

```
>> clear
>> %% Numeros complexos
>> z=2+i

z =

    2.0000 + 1.0000i

>> %% Partes real e imaginária de z
>> real(z)

ans =

     2

>> imag(z)

ans =

     1

>> %% Complexo conjugado de z
>> conj(z)

ans =

    2.0000 - 1.0000i

>> %% Valor absoluto |z|
>> abs(z)

ans =

    2.2361
```

The Workspace window shows the following variables:

Name	Value
ans	5
z	2.0000 + 1.0000i

The Command History window shows the following commands:

```
clear
%% Numeros complexos
z=2+i
%% Partes real e imaginária de z
real(z)
imag(z)
%% Complexo conjugado de z
conj(z)
%% Valor absoluto |z|
abs(z)
```

Annotations in the image:

- An arrow points from the text "MatLab inclui **variáveis complexas** de forma simples e intuitiva." to the line `z=2+i` in the Command Window.
- An arrow points from the text "Operador complexo conjugado: $z^* \rightarrow \text{conj}(z)$ " to the line `conj(z)` in the Command Window.
- An arrow points from the text "Módulo de z: $|z| \rightarrow \text{abs}(z)$ " to the line `abs(z)` in the Command Window.
- An arrow points from the text "Sugestão: calcule $z \cdot z^*$ e mostre que é igual a $|z|^2$." to the line `abs(z)` in the Command Window.

Usando o "Help" do MatLab

ou digite "F1"

Pesquise!

Vários exemplos

Help

File Edit View Go Favorites Desktop Window Help

Help Navigator

complex numbers

Contents Index Search Results Demos

Documentation Search Results (113)

Title	Section
Complex Numbers	Classes (Data Types)
Numbers	Matrices and Arrays
Working with Complex Numbers	Working with the Embedded MATLAB Subset
Complex Signals	Working with Signals
Validating User Input as Numbers	Examples of GUIDE GUIs
Handling Complex Data	Creating Fortran MEX-Files
Handling Complex Data	Creating C Language MEX-Files
Mapping Complex Values to C	Calling C Functions from the MATLAB Command Window
Matrix Generating Functions No Longer Available in Version 7.5 (R2007b) MATLAB Software	Version 7.5 (R2007b) MATLAB Software
New Warnings for Complex Inputs to cplxpair	Version 7 (R14) MATLAB Software
exp	Functions

Demo Search Results (3)

Title	Product
Adding User-Defined Wavelets	Wavelet
Basic Matrix Operations	MATLAB
Creating IDNLGREY Model Files	System Identification

Search Support Database on Web for [complex numbers](#)

Title: Embedded MATLAB Coding Style :: Working with the Embedded MATLAB Subset (Embedded MATLAB™)

Working with Complex Numbers

The Embedded MATLAB subset supports **complex numbers** and operations.

- Creating Local **Complex Variables By Assignment**
- Semantic Rules for **Complex Numbers**

Creating Local **Complex Variables By Assignment**

As in MATLAB, you create **complex** variables by assignment. Unlike in MATLAB, you must set complexity at the time of assignment, either by assigning a **complex** constant to the variable or using the **complex** function, as in these examples:

```
x = 5 + 6i; % x is a complex number by assignment.  
y = 7 + 8j; % y is a complex number by assignment.  
x = complex(5,6); % x is the complex number 5 + 6i.
```

Use the following rules to specify and use **complex** variables in Embedded MATLAB functions:

- Once you set the type and size of a variable, you cannot cast it to another type or size. In the following example, the variable **x** is declared **complex** and stays **complex**:

```
x = 1 + 2i; % x is declared a complex variable.  
y = int16(x); % Real and imaginary parts of y are int16.  
x = 3; % x now has the value 3 + 0i.
```

Conflicts can occur from operations with real operands that can have **complex** results. For example, the following code generates an error:

```
z = 3; % Sets type of z to double (real)  
z = 3 + 2i; % ERROR: cannot recast z to complex
```

The following is a possible workaround that you can use if you know that a variable can be assigned a **complex** number:

```
m = complex(3); % Sets m to complex variable of value 3 + 0i  
m = 5 + 6.7i; % Assigns a complex result to a complex number
```

- Cases in which a function can return a **complex** number for a real argument are handled individually for each

Tarefas da aula de hoje

Lembrando: toda aula haverá tarefas!! (20% da média final!!!)

- Tarefa 1: Calcule o seno, o cosseno, a tangente, a raiz quadrada e a raiz cúbica de $\pi/9$.
 - Tarefa 2: Dados dois números complexos $z_1=5+8i$ e $z_2=6+4i$, calcule:
 - $z_1 \cdot z_2$
 - $|z_1+z_2|^2$
 - o ângulo de fase de z_1+z_2 .
-

Básico do MatLab: Vetores e Matrizes

“**Tudo** (ou quase tudo) no MatLab são matrizes”

The image shows a MATLAB interface with three windows: Command Window, Workspace, and Command History.

Command Window:

```
>> %% Usando "%" antes do comando escrevemo um comentários (será ignorado...)  
>> %% "clear" limpa as variáveis e etc.  
>> %% Definindo um vetor (array)  
>> Vetor=[0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5]  
  
Vetor =  
  
Columns 1 through 8  
    0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000  
  
Columns 9 through 11  
    4.0000    4.5000    5.0000  
  
>> %% Acessando o i-ésimo elemento do vetor: Vetor(i)  
>> Vetor(7)  
  
ans =  
  
    3  
  
>> Vetor(10)  
  
ans =  
  
    4.5000
```

Workspace:

Name	Value
Vetor	<1x11 double>
ans	4.5000

Command History:

```
clear  
%% Um vetor em 1D (array)  
clear  
%% Usando "%" antes do com  
%% Definindo um vetor (ar  
Vetor=[0 1 2 3 4 5 6 7 8 9  
%% Acessando o i-ésimo el
```

Annotations:

- An arrow points from the text "Vetores: criados com “[]” e espaços entre os elementos." to the command `Vetor=[0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5]` in the Command Window.
- An arrow points from the text "Acessamos o i-ésimo elemento do vetor com “()”" to the command `Vetor(7)` in the Command Window.
- Another arrow points from the same text to the command `Vetor(10)` in the Command Window.
- A red circle highlights the Workspace window.

Criando vetores com o operador ":"

The screenshot shows the MATLAB Command Window and Workspace. The Command Window displays the following code and output:

```
>> clear
>> %% O Operador ":" nos permite gerar vetores com valores igualmente espaçados
>> %% de forma mais prática.
>> %% Por exemplo:
>> Vetor2=0:0.5:5
```

Vetor2 =

Columns 1 through 8
0 0.5000 1.0000 1.5000 2.0000 2.5000 3.0000 3.5000

Columns 9 through 11

4.0000 4.5000 5.0000

```
>> %% Notacao: Vetor=Inicial:Passo:Final
>> Vetor2(7)
```

ans =

```
3
```

```
>> Vetor2(10)
```

ans =

```
4.5000
```

The Workspace window shows the following variables:

Name	Value
Vetor2	<1x11 double>
ans	4.5000

The Command History window shows the following commands:

```
clear
%% Usando "%" antes do com
clear
%% Usando "%" antes do com
%% "clear" limpa as variáv
%% Definindo um vetor (arr
Vetor=[0 0.5 1 1.5 2 2.5
%% Acessando o i-ésimo ele
```

Vetores regulares: podem ser criados com o operador ":"
"Vetor=Inicial:Passo:Final"

Obs: se o passo for omitido, o MatLab assume "1".

Acessamos o i-ésimo elemento do vetor com "("

Criando matrizes (2D)

The screenshot shows the MATLAB Command Window and Workspace. The Command Window contains the following code and output:

```
>> %% Criamos matrizes (2D) da mesma forma
>> %% Agora, separando as linhas com ";"
>> Mat1=[0 0.5 1 1.5; 2 3 4 5]

Mat1 =

    0    0.5000    1.0000    1.5000
    2    3.0000    4.0000    5.0000

>> %% ou, usando o operador ":"
>> Mat2=[0:0.5:1.5;2:5]

Mat2 =

    0    0.5000    1.0000    1.5000
    2    3.0000    4.0000    5.0000

>> %% Criando um vetor a partir de uma matriz
>> Vec1=Mat1(1,:)

Vec1 =

    0    0.5000    1.0000    1.5000

>> % Vec1 é a 1a linha de Mat1
>> Vec2=Mat1(:,2)

Vec2 =

    0.5000
    3.0000

>> % Vec2 é a 2a coluna de Mat1
>> Vec1(2)

ans =

    0.5000
```

The Workspace window shows the following variables:

Name	Value
Mat1	[0,0.5000,1,1.5000;2,3,4,5]
Mat2	[0,0.5000,1,1.5000;2,3,4,5]
Vec1	[0,0.5000,1,1.5000]
Vec2	[0.5000;3]
ans	3

The Command History window shows the following commands:

```
Mat2=[0:0.5:1.5;2:5]
%% Criando um vetor a partir de uma matriz
Vec1=Mat1(1,:)
% Vec1 é a 1a linha de Mat1
% Vec2 é a 2a coluna de Mat1
clear
%% Criamos matrizes (2D) da mesma forma
%% Agora, separando as linhas com ";"
Mat1=[0 0.5 1 1.5; 2 3 4 5]
%% ou, usando o operador ":"
Mat2=[0:0.5:1.5;2:5]
%% Criando um vetor a partir de uma matriz
Vec1=Mat1(1,:)
% Vec1 é a 1a linha de Mat1
% Vec2 é a 2a coluna de Mat1
Vec2=Mat1(:,2)
```

Annotations in blue text:

- Matrizes: podem ser criadas das mesmas formas, separando as linhas com “;”
- No caso, temos uma matriz 2x4
- Podemos também criar vetores a partir de matrizes!!

Red arrows point from the blue text to the corresponding code and output in the Command Window.

Tarefas da aula de hoje (cont)

Lembrando: toda aula haverá tarefas!! (20% da média final!!!)

- Tarefa 3: Gere uma sequência de números pares com início em 4 e término em 100.
 - Tarefa 4: Gere uma sequência que comece em $-\pi$ e acabe em $+\pi$ com um passo de $\pi/15$.
 - Tarefa 5: Gere uma matriz 20x20 (elementos à sua escolha)
-