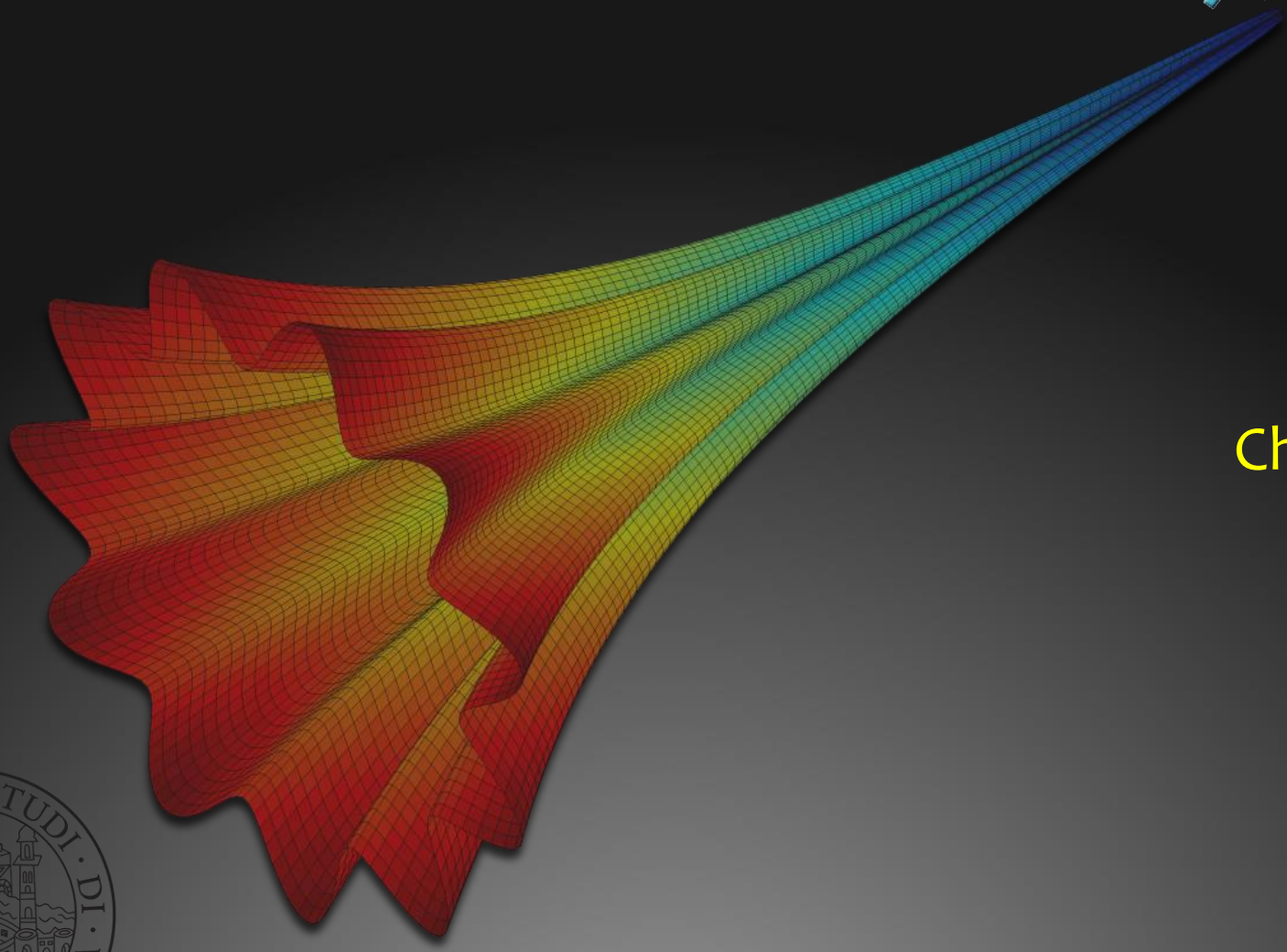


Chapter 1

Linguaggio Programmazione Matlab-Simulink (2017/2018)

Info Corso MATLAB

- Docente: Maris Bogdan Mihai
- Ufficio: Ca' Vignal 2, Piano 1, Stanza 61, lab. Altair (-2)
- E-mail: bogdan.maris@univr.it
- Lezioni:
  - giovedì dalle 14:30 alle 17:30
  - 8 lezioni da 3 ore ciascuna
- Modalità d'esame:

  (frequenza al corso non è obbligatoria)
  - Prova finale: scritto || test al calcolatore
  - Idoneo|| non idoneo

Info Corso MATLAB

- Testo di riferimento disponibile in biblioteca:

"Matlab: A Practical Introduction to Programming and Problem Solving" third edition by Stormy Attaway (in inglese)

- Materiale on-line sul sito del corso: slide delle lezioni,

esercizi, codice MATLAB,…

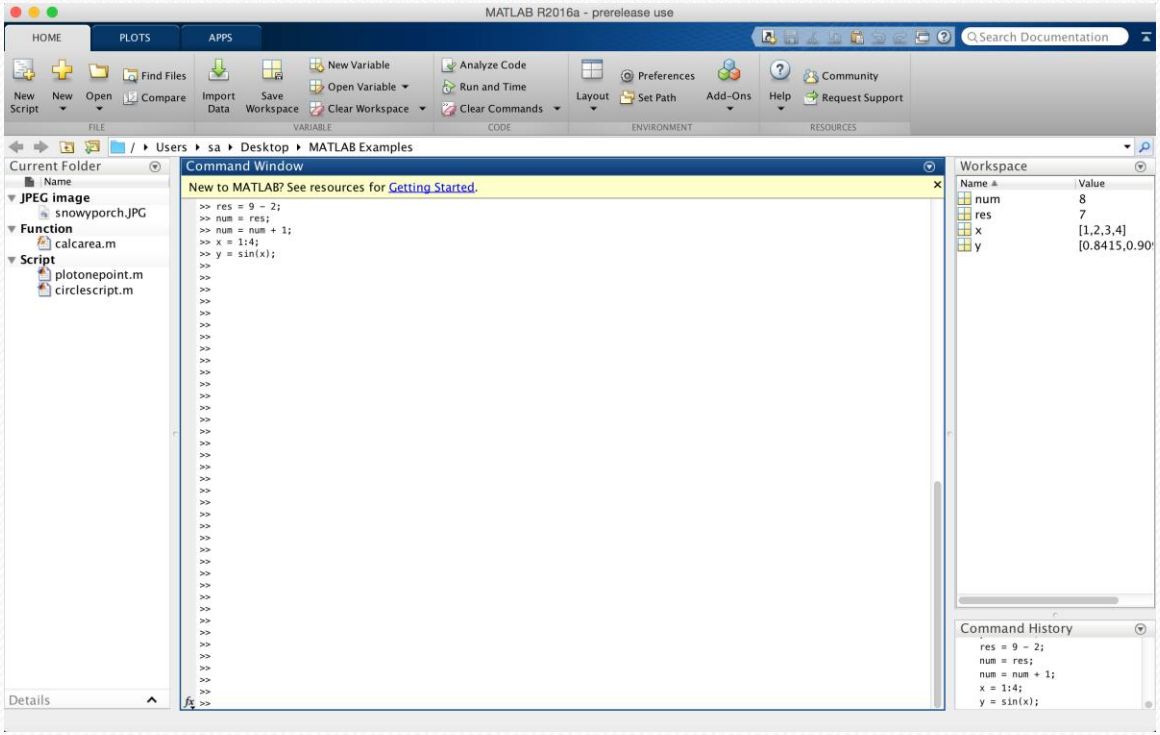
http://www.di.univr.it/?ent=oi&aa=2017%2F2018&codiceCs=S24&codins=4S007126&cs=420&discr=&discrCd=&lang=it

Introduction to MATLAB

- MATrix LABoratory
- Many mathematical and graphical applications
- Has programming constructs but not a programming language
- Also has many built-in functions
- Can use interactively in the Command Window, or write your own programs
- In the Command Window the >> is the prompt
  - At the prompt, enter a command or expression
  - MATLAB will respond with a result

MATLAB Desktop Environment

- Command Window is large window in middle; Current Folder Window to left, Workspace and Command History to right



Desktop Environment

- Current Folder window shows files; the folder set as the Current Folder is where files will be saved
- Workspace Window: shows variables (discussed next)
- Command History Window: shows commands that have been entered and on what date
- Toolstrip on top has tabs for HOME (the default), PLOTS, and APPS
- HOME tab is divided into functional sections FILE, VARIABLE, CODE, ENVIRONMENT, RESOURCES
  - Under ENVIRONMENT, Layout allows for customization of the Desktop Environment

Variables and Assignments

- To store a value, use a *variable*
- one way to put a value in a variable is with an *assignment statement*
- general form:

  variable = expression

- The order is important
  - variable name on the left
  - the assignment operator "=" (Note: this does NOT mean equality)
  - expression on the right

Variables and Assignments

- For example, in the Command Window at the prompt:

```
>> mynum = 6
mynum =
        6
>>
```

- This assigns the result of the expression, 6, to a variable called *mynum*
- A semicolon suppresses the output but still makes the assignment

```
>> mynum = 6;
>>
```

- If just an expression is entered at the prompt, the result will be stored in a default variable called *ans* which is re-used every time just an expression is entered

```
>> 7 + 4
ans =
    11
>>
```

Modifying Variables

- *Initialize* a variable (put its first value in it)

```
mynum = 5;
```

- Change a variable (e.g. by adding 3 to it)

```
mynum = mynum + 3;
```

- *Increment* by one

```
mynum = mynum + 1;
```

- *Decrement* by two

```
mynum = mynum – 2;
```

NOTE: after this sequence, *mynum* would have the value 7 (5+3+1-2)

Variable names

- Names must begin with a letter of the alphabet
- After that names can contain letters, digits, and the underscore character _
- MATLAB is case-sensitive
- the built-in function **namelengthmax** tells what the limit is for the length of a variable name
- Names should be mnemonic (they should make sense!)
- The commands **who** and **whos** will show variables
- To delete variables: **clear**

Types

- Every expression and variable has an associated *type*, or *class*
  - Real numbers: **single**, **double**
  - Integer types: numbers in the names are the number of bits used to store a value of that type
    - Signed integers: **int8**, **int16**, **int32**, **int64**
    - Unsigned integers: **uint8, uint16, uint32, uint64**
  - Characters and strings: **char**
  - True/false: **logical**
- The default type is **double**

Expressions

- Expressions can contain values, variables that have already been created, operators, built-in functions, and parentheses
- Operators include:
  - `+` addition
  - `-` negation, subtraction
  - `*` multiplication
  - `/` division (divided by e.g. 10/5 is 2)
  - `\` division (divided into e.g. 5\10 is 2)
  - `^` exponentiation (e.g. 5^2 is 25)
- Operator precedence:
  - `()` parentheses
  - `^` exponentiation
  - `-` negation
  - `*, /, \` all multiplication and division
  - `+, -` addition and subtraction

Formatting

- **format** command has many options, e.g:
  - long, short
  - loose, compact
- Continue long expressions on next line using ***ellipsis***:

```
>> 3 + 55 - 62 + 4 - 5 ...
   + 22 - 1
ans =
    16
```

- Scientific or exponential notation: use e for exponent of 10 raised to a power
  - e.g. 3e5 means 3 * 10^5

Operator Precedence

- Some operators have precedence over others
- Precedence list (highest to lowest) so far:

  ( ) parentheses

  ^ exponentiation

  - negation

  *, /, \ all multiplication and division

  +, - addition and subtraction

- Nested parentheses: expressions in inner parentheses are evaluated first

Built-in functions and help

- There are many, MANY built-in functions in MATLAB
- Related functions are grouped into help topics
- To see a list of help topics, type "help" at the prompt:

  >> help
- To find the functions in a help topic, e.g. elfun:

  >> help elfun
- To find out about a particular function, e.g. sin:

  >> help sin
- Can also choose the Help button under Resources to bring up the Documentation page

Using Functions: Terminology

- To use a function, you *call* it
- To call a function, give its name followed by the *argument(s)* that are *passed* to it in parentheses
- Many functions calculate values and *return* the results
- For example, to find the absolute value of -4

```
>> abs(-4)
ans =
    4
```

- The name of the function is "abs"
- One argument, -4, is passed to the **abs** function
- The **abs** function finds the absolute value of -4 and returns the result, 4

Functional form of operators

- All operators have a functional form
- For example, an expression using the addition operator such as 2 + 5 can be written instead using the function **plus**, and passing 2 and 5 as the arguments:

```
>> plus(2,5)
ans =
     7
```

Constants

- In programming, variables are used for values that could change, or are not known in advance
- *Constants* are used when the value is known and cannot change
- Examples in MATLAB (these are actually functions that return constant values)

pi	3.14159….
i, j	$\sqrt{-1}$
inf	infinity
NaN	stands for “not a number”; e.g. the result of 0/0

Random Numbers

- Several built-in functions generate random (actually, pseudo-random) numbers
- Random number functions, or random number generators, start with a number called the ***seed***; this is either a predetermined value or from the clock
- By default MATLAB uses a predetermined value so it will always be the same
- To set the seed using the built-in clock:

  rng('shuffle')

Random Real Numbers

- The function **rand** generates uniformly distributed random real numbers in the open interval `(0,1)`
- Calling it with no arguments returns one random real number
- To generate a random real number in the open interval `(0,N)`:

  rand * N

- **randn** is used to generate normally distributed random real numbers

Random Integers

- Rounding a random real number could be used to produce a random integer, but these integers would not be evenly distributed in the range

- The function **randi(imax)** generates a random integer in the range from 1 to imax, inclusive
  - A range can also be passed:
    - randi([m,n],1) generates one integer in the range from m to n

Characters and Strings

- A ***character*** is a single character in single quotes
- All characters in the computer' s character set are put in an order using a ***character encoding***
- The character set includes all letters of the alphabet, digits, punctuation marks, space, return, etc.
- Character strings are sequences of characters in quotes, e.g. 'hello and how are you?'
- In the character encoding sequence, the letters of the alphabet are in order, e.g. 'a' comes before 'b'
- Common encoding ASCII has 128 characters, but MATLAB can use a much larger encoding sequence

Relational Expressions

- The relational operators in MATLAB are:

>	greater than
<	less than
>=	greater than or equals
<=	less than or equals
==	equality
~=	inequality

- The resulting type is **logical** 1 for true or 0 for false
- The logical operators are:

||	or for scalars
&&	and for scalars
~	not

- Also, **xor** function which returns logical true if only one of the arguments is true

Truth Table

- A truth table shows how the results from the logical operators for all combinations

x	y	~x	x || y	x && y	xor(x,y)
true	true	false	true	true	false
true	false	false	true	false	true
false	false	true	false	false	false

- Note that the logical operators are commutative (.e.g., x|| y is equivalent to y || x)

Expanded Precedence Table

- The precedence table is expanded to include the relational and logical operators:

Operators	**Precedence**
parentheses: ( )	highest
power ^	
unary: negation (-), not (~)	
multiplication, division *, /, \	
addition, subtraction +, -	
relational <, <=, >, >=, ==, ~=	
and &&	
or ||	
assignment =	lowest

Range and Type Casting

- Range of integer types found with **intmin/intmax**
  - e.g. **intmin('int8')** is -128, **intmax('int8')** is 127
- Converting from one type to another, using any of the type names as a function, is called *casting* or *type casting*, e.g:

```
>> num = 6 + 3;
>> numi = int32(num);
>> whos
  Name        Size              Bytes  Class     Attributes

  num         1x1                   8  double
  numi        1x1                   4  int32
```

- The **class** function returns the type of a variable

Characters and Encoding

- standard ASCII has 128 characters; integer equivalents are 0-127
- any number function can convert a character to its integer equivalent

```
>> numequiv = double('a')
numequiv =
    97
```

- the function **char** converts an integer to the character equivalent (e.g. **char(97)**)
- MATLAB uses an encoding that has 65535 characters; the first 128 are equivalent to ASCII

Some Functions in elfun

- Trig functions, e.g. **sin**, **cos**, **tan** (in radians)
  - Also arcsine **asin**, hyperbolic sine **sinh**, etc.
  - Functions that use degrees: **sind**, **cosd**, **asind**, etc.
- Rounding and remainder functions:
  - **fix**, **floor**, **ceil**, **round**
  - **rem**, **mod**: return remainder
  - **sign** returns sign as -1, 0, or 1
- **sqrt** and **nthroot** functions
- **deg2rad** and **rad2deg** convert between degrees and radians

Log Functions

- MATLAB has built-in functions to return logarithms:
  - **log(x)** returns the natural logarithm (base e)
  - **log2(x)** returns the base 2 logarithm
  - **log10(x)** returns the base 10 (common) logarithm

- MATLAB also has a built-in function **exp(n)** which returns the constant e^n
  - Note: there is no built-in constant for e; use **exp** instead
  - Also, do not confuse with exponential notation e

Beware of Common Pitfalls

- Confusing the format of an assignment statement (make sure that the variable name is always on the left)
- Forgetting to use parentheses to pass an argument to a function (e.g., typing “fix 2.3” instead of “fix(2.3)”)
- Confusing || and xor
- Using = instead of == for equality
- Using an expression such as “5 < x < 10” – which will always be true, regardless of the value of the variable x (because the expression is evaluated from left to right; 5 < x is either true (1) or false (0); both 1 and 0 are less than 10)

Programming Style Guidelines

- Use mnemonic variable names (names that make sense; for example, *radius* instead of *xyz*)
- Although variables named *result* and *RESULT* are different, avoid this as it would be confusing
- Do not use names of built-in functions as variable names
- Store results in named variables (rather than using *ans*) if they are to be used later
- Make sure variable names have fewer characters than **namelengthmax**
- If different sets of random numbers are desired, set the seed for the random functions using **rng**

Exercises

1. Generate a:

- real number in the range (0,1)
- real number in the range (0, 100)
- real number in the range (20, 35)
- integer in the inclusive range from 1 to 100
- integer in the inclusive range from 20 to 35

Exercises

2. Think about what would be produced by the following expressions, and then type them in to verify your answers.

- >> 3 == 5 + 2 >> 'b' < 'a' + 1
- >> 10 > 5 + 2 >> (10 > 5) + 2
- >> 'c' == 'd' - 1 && 2 < 4
- >> 'c' == 'd' - 1 || 2 > 4
- >> xor('c' == 'd' - 1, 2 > 4) >> xor('c' == 'd' - 1, 2 < 4)
- >> 10 > 5 > 2

Exercises

3. Calculate the range of integers that can be stored in the types **int16** and **uint16**. Use **intmin** and **intmax** to verify your results.

4. Find the numerical equivalent of the character ‘x’.

5. Find the character equivalent of 107.

Exercises

6. Use the **help** function to find out what the rounding functions **fix**, **floor**, **ceil**, and **round** do. Experiment with them by passing different values to the functions, including some negative, some positive, some with fractions less than 0.5 and some greater.

Solutions

1.

- real number in the range (0,1) rand
- real number in the range (0, 100) rand*100
- real number in the range (20, 35) rand*(35-20)+20
- integer in the inclusive range from 1 to 100 randi(100)
- integer in the inclusive range from 20 to 35
  randi([20, 35])

Solutions

3.

- >> 2^16 ans = 65536
- >> 2^15 ans = 32768
- >> intmin('int16') ans = -32768
- >> intmax('int16') ans = 32767
- >> intmin('uint16') ans = 0
- >> intmax('uint16') ans = 65535

Solutions

4.

- >> double('x')

ans =

120

5.

- >> char(107)

ans =

k