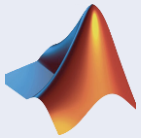
UNIVERSITY OF PAVIA

FACULTY OF ENGINEERING

Industrial Control

Prof. Lalo Magni



BRUSH UP ON MATLAB

ICDS Lab

Chiara Toffanin, Assistant Professor

Gian Paolo Incremona, Ph.D.

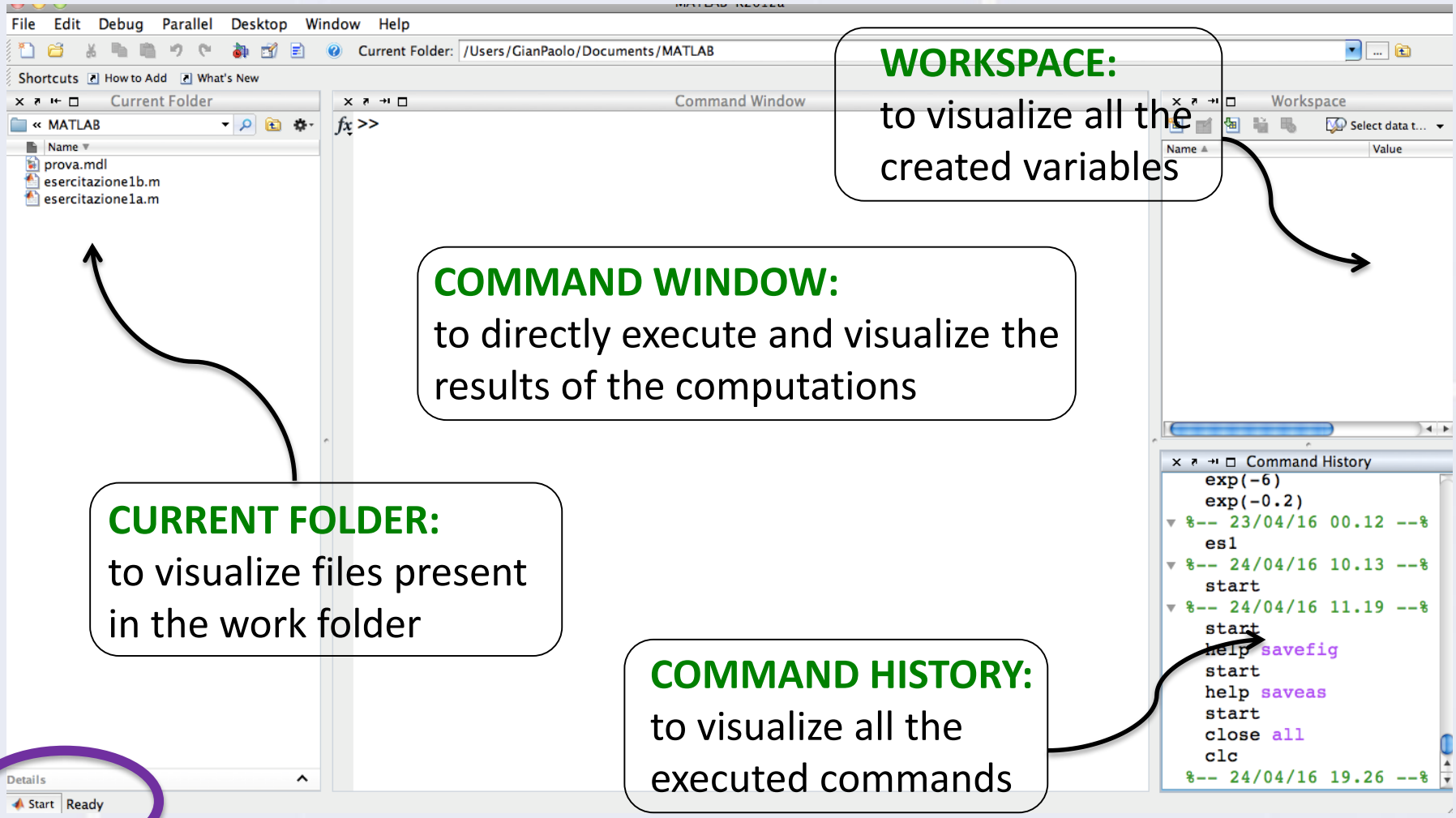
MATLAB: What is it?

MATLAB (from **MAT**rix **LAB**oratory) is a multi-paradigm numerical computing environment and fourth-generation programming language

MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages

MATLAB is intended primarily for numerical computing and includes several **Toolboxes**, that is additional functions to solve specific classes of problems

Work Environment



WORKSPACE:
to visualize all the created variables

COMMAND WINDOW:
to directly execute and visualize the results of the computations

CURRENT FOLDER:
to visualize files present in the work folder

COMMAND HISTORY:
to visualize all the executed commands

Basics

•The simplest way to use MATLAB is doing simple computations such as (+, -, *, /, ^)

```
>> (18*2+19)/88
```

•If you use ; the result of the computation is not visualized,

• If you use % a comment is indicated

•It is possible to use predefined functions or write customized functions

•All the variables are stored into the **workspace**

Eigenvalues and eigenvectors

Given the matrix A, $n \times n$, the eigenvalues can be found as

```
a=eig(A)
```

It allows one to obtain only the eigenvalues, while

```
[V,D]=eig(A)
```

gives as output the matrix V, $n \times n$ of the eigenvectors and the diagonal matrix D, containing all the eigenvalues of the matrix A

To compute the determinant, rank, inverse and norm of A

```
det(A)
```

```
rank(A)
```

```
inv(A)
```

```
norm(A)
```

Other utilities

`A(i,j)`	select the entries of i-th row and j-th column
`A(:,j)`	select the j-th column of A
`A(i,:)`	select the i-th row of A
`A(:,end)`	select the last column of A
`A(end,:)`	select the last row of A
`A=[]`	create an empty vector or matrix
`v(i)`	select the i-th element of vector v
`[m,n]=size(A)`	determine the size of A (m rows, n columns)
`m=length(v)`	determine the number of elements of v

Polynomial functions

In order to represent a polynomial function, it is necessary to write a vector containing the coefficients in a decreasing order. For instance $p=t^3 -6t+3$ is

```
p=[1 0 -6 3]
```

while

```
r=roots(p)
```

allows one to find the root values of p

Given two polynomials a and b, their product is

```
>> a=[1 2 3]; b=[4 5 6];
>> c=conv(a,b);
```

Math functions

•sin	`sin(z)`	`sind(z)`
•cosin	`cos(z)`	`cosd(z)`
•tangent	`tan(z)`	`tand(z)`
•arctan	`atan(y)`	`atand(y)`
•exponent	`exp(x)`	
•natural logarithm	`log(x)`	
•10 logarithm	`log10(x)`	
•root square value	`sqrt(x)`	

The element-wise computations are executed as `.+` `.*` `.^` `./`

Plots

plot is the function used to draw graphics

```
>> figure(1)
plot(t,q,'b','linewidth',1.1);
title('Position');
xlabel('time (s)');
ylabel('$q$ (rad)');
grid on
xlim([-5,20])
ylim([0,0.6])
```

To open a new figure

Plot figure with line width 1.1

Define labels for the axes

Draw the grid

Define the limit for the x y axes

m-files

All the files containing instructions which can be executed by MATLAB are called m-files. (e.g. namefile.m)
If they contain only a procedure, they are called **script files**, if they contain functions with some inputs and outputs they are called **function files**.

```
function y=linear(x,alfa,beta)
y= alfa + beta *x;
>> x=[1 4 5 9];
>> linear(1,4,x)
ans =
5 8 9 13
```

Outputs, Inputs: they can have generic names

Cycles

It is possible to realize typical cycles (**for, while, if**). Note that it is necessary to close each cycle by using the instruction **end**.

`for` condition …instructions ... `end`

`while` condition …instructions ... `end`

`if` condition …instructions ... `else` ... instructions `end`

```
% a1+a2+ … + a9 with ai=i^2/(i+1)
s=0;
for i=1:9
   s=s+i^2/(i+1);
end
>> s =
37.9290
```

Other utilities

`who` to visualize which variables are present in the workspace

`clear all` clear all the variables in the workspace

`clear variable_name` clear only variable_name

`save file_name` save the content of the workspace

`load file_name` load all the variables contained in file_name

`cd` to change directory

`dir` or `ls` to see the content of the directory

`help command` to see the details of the considered command

`doc command` to see the documentation of the command

Control Systems Toolbox

Toolbox for the analysis and simulation of dynamical systems

- `tf([num],[den])` creates a transfer function with the coefficients of the numerator and denominator polynomials specified in num and den vectors, respectively
- `s = tf('s')` denotes the **s** laplace variable in Matlab workspace
- `sys=ss(A,B,C,D)` creates a LTI system starting from the state-space matrices A,B,C,D.
- `[A, B, C, D] = ssdata(sys)` given a LTI system **sys** (or transfer function), returns the state spaces representation matrices A, B, C, D.

Some usefull instruction…

- `trim` calculates an equilibrium point of a Simulink model
- `linmod` linearize a Simulink model around a specified equilibrium point
- `sigma(sys)` plot of the minimum and maximum singular values of the system **sys**
- `norm(A,p)` calculates the p-norm of the A matrix (p can be equal to 1, 2 or inf)
- `K=lqr(A,B,Q,R)` synthetizes a continuous time LQR controller
- `K=dlqr(A,B,Q,R)` synthetizes a discrete time LQR controller

…some other

`kalman` synthetizes a continuous/discrete time Kalman predictor

`sysdis=c2d(sys,Ts)` discretizes a continuous time LTI system (sys) with sampling time Ts

`ctrb(sys)` calculates the controllability matrix of the linear system sys

`obsv(sys)` calculates the observability matrix of the linear system sys

`pzmap(sys)` plot of the poles and zeros of the linear system sys in the complex plane

`zero(sys)` calculates zeros of the linear system sys

`pole(sys)` calculates poles of the linear system sys

`minreal` minimal realization or pole-zero cancelation

Block diagonal matrix

To create a block-diagonal matrix with blocks X and Y , use

```
Z = blkdiag(X,Y)
```

```
kron(eye(N),X)
```

To create a block matrix , the easiest way is concatenation

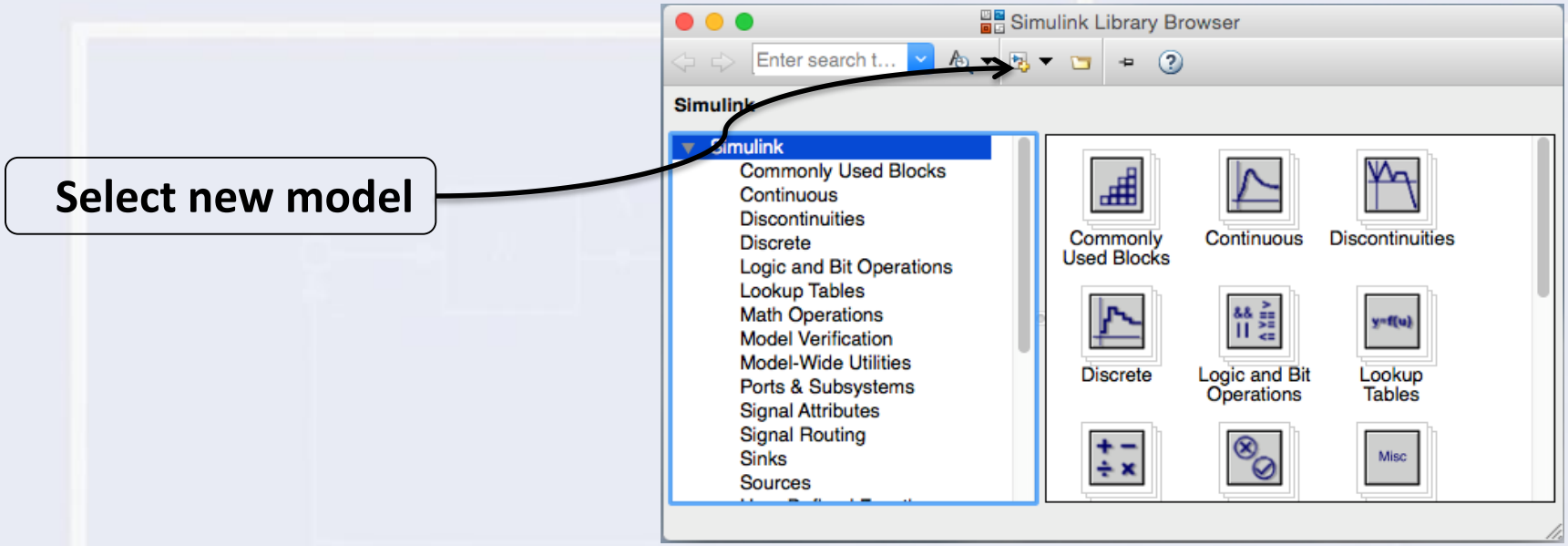
```
F = [];
for i = 1:N
   temp = [];
   for j = 1:M
      temp = [temp Q];
   end
   F = [F;temp];
end
```

Simulink

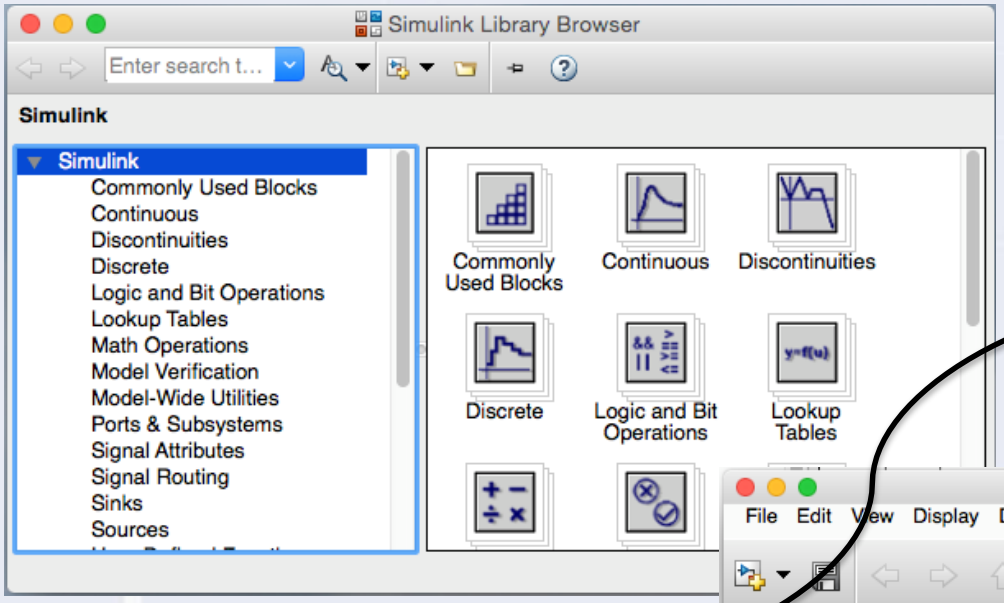
Simulink (from **Sim**ulation and **Link**) is a **Toolbox** to simulate dynamical systems, which uses a graphical interface to realize the model through predefined blocks. Write

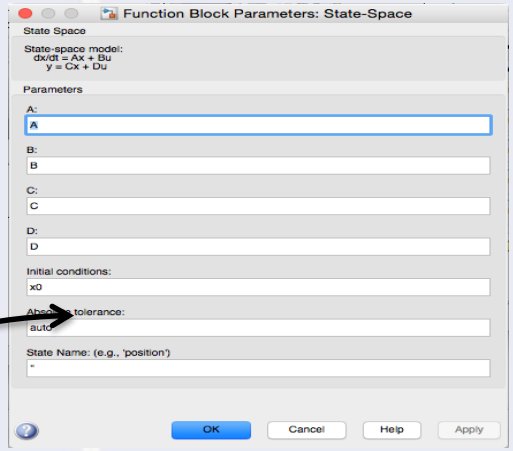
```
simulink
```

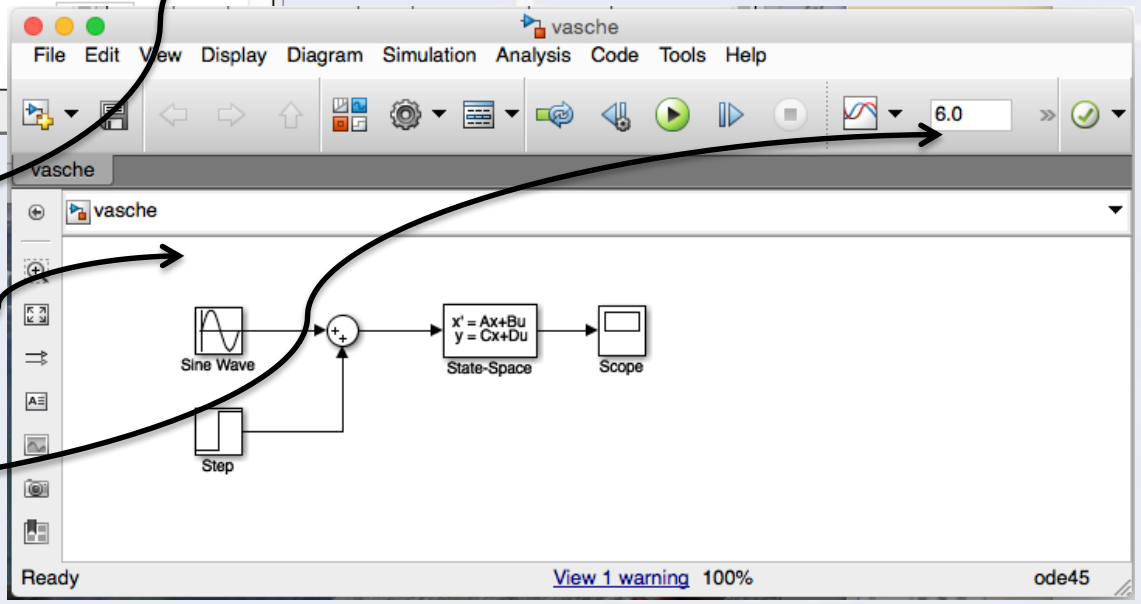
to open the library with all the blocks.



How to proceed?





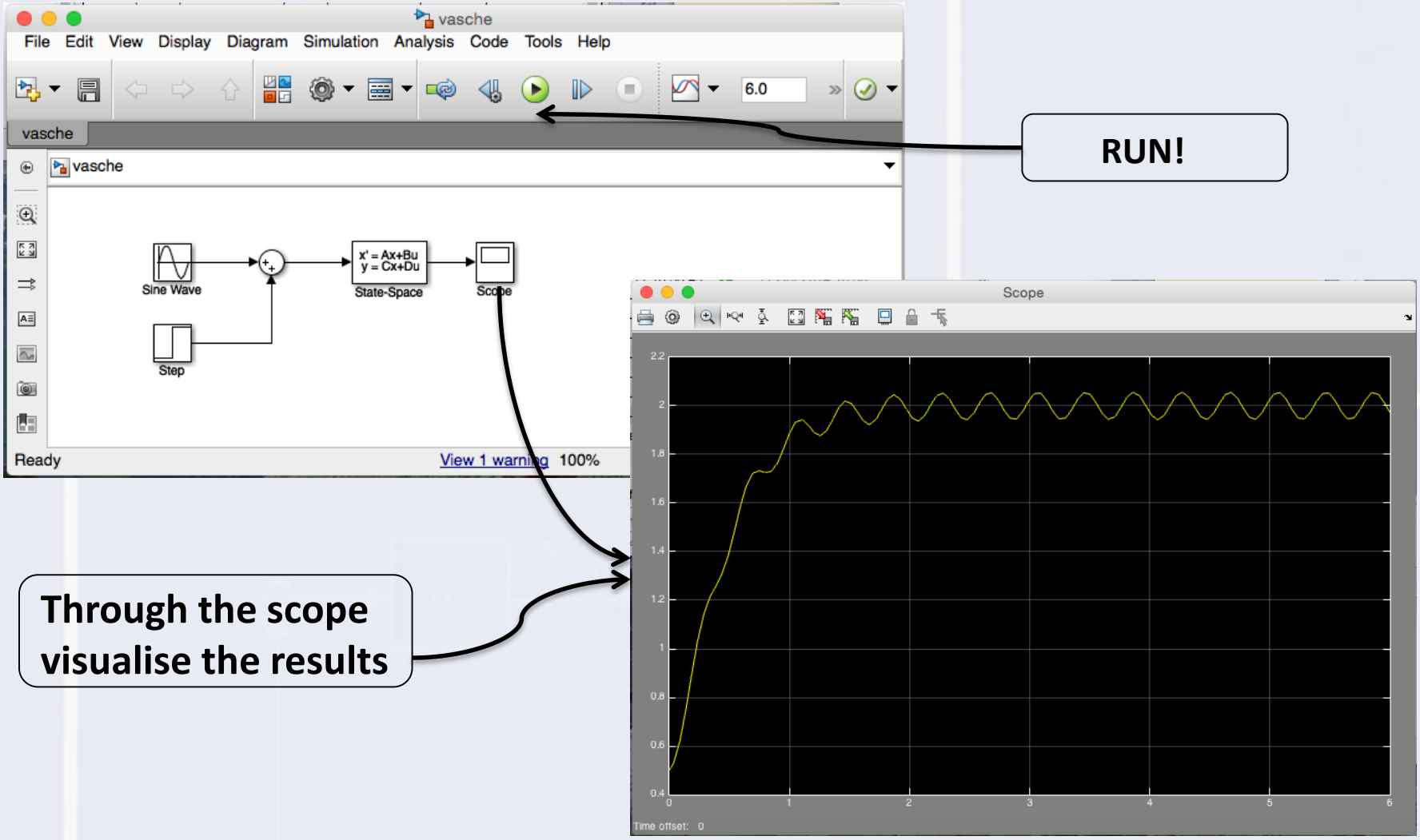


Set the parameters

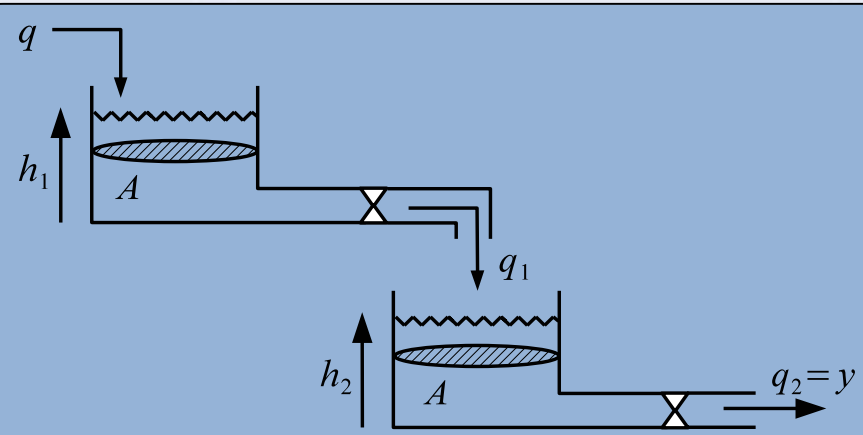
Drags the blocks

Set the time

Run the simulation



Try by yourself and good luck!



DYNAMICAL SYSTEM

DATA

$$A = 1\,m^2$$

$$q_1 = 3h_1\ [m^3/s]$$

$$q_2 = 5h_2\ [m^3/s]$$

MODEL

$$A\dot{h}_1 = q - 3h_1$$

$$A\dot{h}_2 = 3h_1 - 5h_2$$

$$y = 5h_2$$

$$x_1 = h_1,\ x_2 = h_2,\ u = q$$

STATE SPACE

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -3 & 0 \\ 3 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$