

# Secure and reliable control systems: Matlab basics

Gianluca Bianchin  
gianluca@engr.ucr.edu  
Skype: giangi61978



Department of Mechanical Engineering  
University of California, Riverside

MSC 003 & Online - April 10, 2019

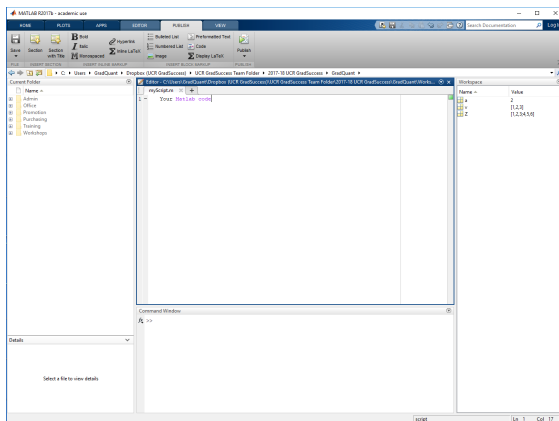
# Matlab: a brief description

- MATLAB = Matrix Laboratory, developed by MathWorks
- Includes a proprietary programming language
- Includes optional toolboxes for specific applications (Simulink, Computer Vision, SimBiology, Econometrics, ... )
- Great integration with Python, R, C++, L<sup>A</sup>T<sub>E</sub>X, ...

In short, MATLAB is an environment (programming language + desktop interface) to perform computations on vectors and matrices

UCR provides a free academic license to all students ([link](#))

# Desktop interface



- Current folder
- Editor
- Workspace
- Command window

# Outline of this lecture

- 1 Matlab fundamentals
- 2 State-space models in Matlab
- 3 Dynamical systems with Matlab Simulink

# Matlab fundamentals

# Operators, operations, and variables

We can type operations in the command window:

- Operators:  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $^$
- Operations:  $3*2$ ,  $5*2^3+4*(3)$ ,
- Variables assignment:  
 $a = 3$ ,  $b = 2$ ,  $c = a*b$ ,  $month = \text{'August'}$
- Variables can be visualized in the “Workspace” section
- Some notes:
  - ① No need to define variable types!!
  - ② All variables are handled by value (and not by reference)
  - ③ Variable names must begin with a letter
  - ④ Case sensitive
  - ⑤ Avoid names that correspond to functions

# Operators, operations, and variables

We can type operations in the command window:

- Operators:  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $^$
- Operations:  $3*2$ ,  $5*2^3+4*(3)$ ,
- Variables assignment:  
`a = 3, b = 2, c = a*b, month = 'August'`
- Variables can be visualized in the “Workspace” section
- Some notes:
  - 1 No need to define variable types!!
  - 2 All variables are handled by value (and not by reference)
  - 3 Variable names must begin with a letter
  - 4 Case sensitive
  - 5 Avoid names that correspond to functions

# Pre-defined functions and variables

- Pre-defined functions can be applied to a variable:  
`sqrt(x)`, `sin(x)`, `cos(x)`, `tan(x)`, `exp(x)`, `log(x)`  
`round(x)`, `floor(x)`, `ceil(x)`, ...
- Pre-defined variables: `pi`=3.14159, `i` = `j` =  $\sqrt{-1}$ , `Inf`, `NaN`
- To obtain function description: `help 'functionName'` or click “help” from the toolbar

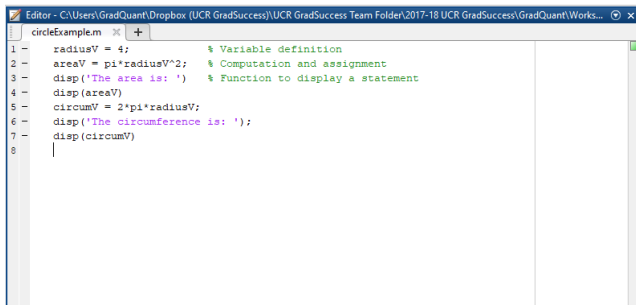


# Scripts and Functions

# Script files

A script file is a collection of commands that are executed in sequence

- Extension “.m”
- Click on the new script icon
- To run: Hit the green arrow in the toolbar



```
Editor - C:\Users\GradQuant\Dropbox (UCR GradSuccess)\UCR GradSuccess Team Folder\2017-18 UCR GradSuccess\GradQuant\Works...
circleExample.m
1 - radiusV = 4; % Variable definition
2 - areaV = pi*radiusV^2; % Computation and assignment
3 - disp('The area is: ') % Function to display a statement
4 - disp(areaV)
5 - circumV = 2*pi*radiusV;
6 - disp('The circumference is: ');
7 - disp(circumV)
8 -
```

# Functions

A function is a group of commands that together perform a certain task

Syntax:

```
function [outVariables] = myfun(invariables)
```

Example of function:

```
1 function [v1] = myFunction(v1, v2)
2 %Sets to zero the entries of vector v1 if the
3 %corresponding entry in v2 has value 1
4     for ind=1:length(v1)
5         if v2(ind)==1
6             v1(ind)=0;
7         end
8     end
9 end
```

# Functions: files

Functions in Matlab are typically saved in separate files

```
MAIN.m

clear
close all
clc

I = eye(2);
O = zeros(2);
n = 4;

% A = [O I; O O];
B = [O O; I O];
V = orth(rand(4));
A = inv(V) * diag(-rand(n,1)) * V;
```

```
MYFUNCTION.m

function [ out ] = myFunction( in )
%UNTITLED

end
```

NOTE: The name of the file and of the function name should be the same

# Calling Functions

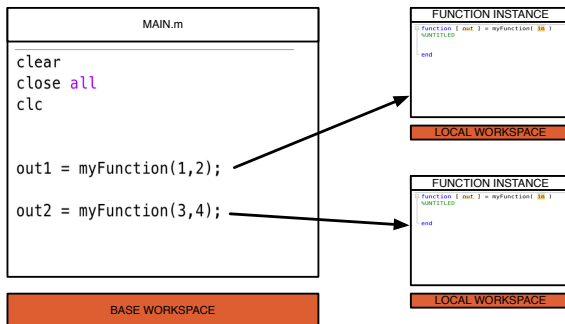
```
1 % "Main" script
2 v1 = [3 4 6];
3 v2 = [0 1 1];
4 v3 = myFunction(v1, v2)
```

```
1 function [v1] = myFunction(v1, v2)
2 %Sets to zero the entries of vector v1 if the
3 %corresponding entry in v2 has value 1
4     for ind=1:length(v1)
5         if v2(ind)==1
6             v1(ind)=0;
7         end
8     end
9 end
```

# An important difference with respect to C++ and Python

(Base and local workspaces)

Functions operate on variables within their own workspace, which is also called the **local workspace**, separate from the workspace you access at the MATLAB command prompt which is called the **base workspace**



- In MATLAB, all variables are referenced **by value**

## Vectors and matrices: storing data through arrays

# Arrays and Matrices

- One-dimensional arrays can be row vectors or column vectors

$$v = \begin{bmatrix} 1 \\ .9 \\ -3.7 \end{bmatrix} \quad w = [2 \quad -5 \quad 0.9 \quad 11.4]$$

- A matrix is represented through two-dimensional array

$$M = \begin{bmatrix} 3 & 1 \\ 2 & 3 \\ 4 & 2 \end{bmatrix}$$



# Arrays and Matrices

- One-dimensional arrays can be row vectors or column vectors

$$v = \begin{bmatrix} 1 \\ .9 \\ -3.7 \end{bmatrix} \quad w = [2 \quad -5 \quad 0.9 \quad 11.4]$$

- A matrix is represented through two-dimensional array

$$M = \begin{bmatrix} 3 & 1 \\ 2 & 3 \\ 4 & 2 \end{bmatrix}$$

# Arrays definition

- To create a column vector, separate the elements with semicolons:

$v = [1; 2; 3; 4]$

- To create a row vector, separate the elements with either a comma or a space:

$w = [1, 2, 3, 4]$

Notice that  $v \neq w$ . In particular,  $v = w'$

- To create a multidimensional array, combine the two notations:

$M = [3, 1; 2, 3; 4, 2]$

# Arrays definition

- To create a column vector, separate the elements with semicolons:  
 $v = [1; 2; 3; 4]$
- To create a row vector, separate the elements with either a comma or a space:

$$w = [1, 2, 3, 4]$$

Notice that  $v \neq w$ . In particular,  $v = w'$

- To create a multidimensional array, combine the two notations:  
 $M = [3, 1; 2, 3; 4, 2]$

# Arrays definition

- To create a column vector, separate the elements with semicolons:  
 $v = [1; 2; 3; 4]$
- To create a row vector, separate the elements with either a comma or a space:  
 $w = [1, 2, 3, 4]$

Notice that  $v \neq w$ . In particular,  $v = w'$

- To create a multidimensional array, combine the two notations:  
 $M = [3, 1; 2, 3; 4, 2]$

# Arrays definition

- To create a column vector, separate the elements with semicolons:  
 $v = [1; 2; 3; 4]$
- To create a row vector, separate the elements with either a comma or a space:  
 $w = [1, 2, 3, 4]$

Notice that  $v \neq w$ . In particular,  $v = w'$

- To create a multidimensional array, combine the two notations:  
 $M = [3, 1; 2, 3; 4, 2]$

## Arrays definition (2)

Other common ways to define arrays are:

- `v = ones(m,n)` ( $m$  by  $n$  array of all ones)
- `v = zeros(m,n)` ( $m$  by  $n$  array of all zeros)
- `v = rand(m,n)` ( $m$  by  $n$  array of random  $[0, 1]$ )
- `v = start:step:end` (equally spaced entries)

Arrays can be combined as blocks:

- `A = [A11, A12; A21, A22]`

# Array slicing

- To “read” a certain entry of an array:

`v(1), M(3,1)`

- “Slicing” allows to read groups of entries of an array:

`M(1:2,1), M([1 3],1:end), M([1 3],:)`

```
1 v = [1, 2, 3, 4, 5];  
2 w = v(3:end)           % w = [3, 4, 5]  
3  
4 w(1) = 10              % w = [10, 4, 5]  
5                         % v = [1, 2, 3, 4, 5]
```

Note:

- 1 Array indices start from 1
- 2 Slicing assignments are handled **by value**

# Array slicing

- To “read” a certain entry of an array:

`v(1), M(3,1)`

- “Slicing” allows to read groups of entries of an array:

`M(1:2,1), M([1 3],1:end), M([1 3],:)`

```
1 v = [1, 2, 3, 4, 5];  
2 w = v(3:end)           % w = [3, 4, 5]  
3  
4 w(1) = 10              % w = [10, 4, 5]  
5                        % v = [1, 2, 3, 4, 5]
```

Note:

- 1 Array indices start from 1
- 2 Slicing assignments are handled **by value**



# Operations on arrays

While other programming languages mostly work with a single array entry at a time, functions and operations in Matlab are optimized for fast processing on entire arrays

- $v + 10$ ,  $v+w$ ,  $2.1*v$ ,  $v/6$
- Function computed over arrays **return an array** with the output  
`sqrt(v)`, `sin(v)`, ...
- Some functions operate on the array and **return a scalar**  
`max(v)`, `mean(v)`, ...

# Efficient functions that operate on entire arrays

- Some useful functions for arrays:

`max()`, `min()`

`mean()`, `median()`, `cov()`, `var()`

`sum()`, `diff()`, `cumsum()`

- Sorting: `sort(v)`
- Find: `find(v==3)`, `find(v>1)`
- Size: `length(v)`, `size(M)`

# Matrix multiplication

- Given two matrices  $A = [a_{ij}] \in \mathbb{R}^{m \times n}$  and  $B = [b_{ij}] \in \mathbb{R}^{n \times p}$ , matrix multiplication  $X = [x_{ij}] = A \cdot B$  produces a  $m \times p$  matrix, where

$$x_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$

- Example:

$$\begin{array}{c} 4 \times 2 \text{ matrix} \\ \begin{bmatrix} a_{11} & a_{12} \\ \cdot & \cdot \\ a_{31} & a_{32} \\ \cdot & \cdot \end{bmatrix} \end{array} \begin{array}{c} 2 \times 3 \text{ matrix} \\ \begin{bmatrix} \cdot & b_{12} & b_{13} \\ \cdot & b_{22} & b_{23} \end{bmatrix} \end{array} = \begin{array}{c} 4 \times 3 \text{ matrix} \\ \begin{bmatrix} \cdot & x_{12} & x_{13} \\ \cdot & \cdot & \cdot \\ \cdot & x_{32} & x_{33} \\ \cdot & \cdot & \cdot \end{bmatrix} \end{array}$$

$$x_{12} = a_{11}b_{12} + a_{12}b_{22}$$

$$x_{33} = a_{31}b_{13} + a_{32}b_{23}$$

- In Matlab:  $X = A * B$

## Matrix multiplication (2)

- Not to be confused with entry-wise multiplication  $E = A \circ B$ , that can be applied to two matrices of identical dimensions  $A = [a_{ij}] \in \mathbb{R}^{m \times n}$  and  $B = [a_{ij}] \in \mathbb{R}^{m \times n}$ , where

$$e_{ij} = a_{ij} \cdot b_{ij}$$

- In Matlab:  $E = A .* B$
- Similarly, matrix exponential:  $A^2$
- Entry-wise exponential:  $A.^2$

# Useful matrix operations for this class

- Identity matrix of size  $n$ : `eye` ( $n$ )
- Rank of matrix  $A$ : `rank` ( $A$ )
- Eigenvalues of matrix  $A$ : `eig` ( $A$ )
- Null space of  $A$ : `null` ( $A$ )
- Inverse of matrix  $A$ : `inv` ( $A$ )
- Pseudoinverse of matrix  $A$ : `pinv` ( $A$ )
  - 1 If the columns of  $A$  are linearly independent,  $A^+$  is a left inverse
  - 2 If the rows of  $A$  are linearly independent,  $A^+$  is a right inverse

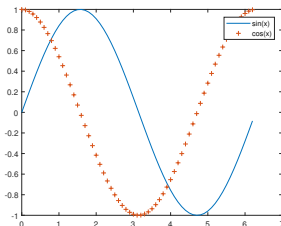
# Plots

# Plots

Two-dimensional line and points can be plotted with the command:

`plot(xdata,ydata)`

```
1 t = 0 : 0.1 : 2*pi;      % Time data (xdata)
2 y1 = sin(t);             % First set of ydata
3 y2 = cos(t);             % Second set of ydata
4 figure                   % Create a figure window
5 plot(t, y1, '-')         % Plot in the same figure window
6 hold on                  % Plot in the same figure window
7 plot(t, y2, '+')
```



# Linestyle, markers, and colors

We can specify our own colors, markers, and linestyles by giving `plot` a third argument

Symbol	Color	Symbol	Marker	Symbol	Linestyle
b	Blue	.	Point	-	Solid line
g	Green	o	Circle	:	Dotted line
r	Red	x	Cross	-.	Dash-dot line
c	Cyan	+	Plus sign	--	Dashed line
m	Magenta	*	Asterisk		
y	Yellow	s	Square		
k	Black	d	Diamond		
w	White	v	Triangle (down)		
		^	Triangle (up)		
		<	Triangle (left)		
		>	Triangle (right)		
		p	Pentagram		
		h	Hexagram		

```
plot(x,y,'g-'), plot(x,y,'k+'), plot(x,y,'b:s')
```



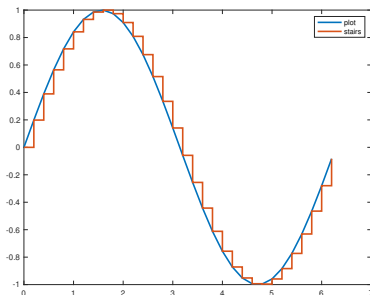
## Note on plot function

When the `plot` function is called with only one argument, e.g.,  
`plot(y)`  
the variable `y` is plotted versus an index of its values, that is, the command is interpreted as follows  
`plot(1:1:length(y), y)`

When `y` is a matrix, MATLAB plots its **columns** as individual signals

# Plotting discrete-time functions

`stairs(yData)` draws a stairstep graph of the elements in `yData`



# State-space models in Matlab

# Modeling

The state-space model for a **continuous-time** linear system is given by

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

The state-space model for a **discrete-time** linear system is given by

$$x[k+1] = Ax[k] + Bu[k]$$

$$y[k] = Cx[k] + Du[k]$$

- $x$  is an  $n \times 1$  vector representing the system's state variables
- $u$  is a  $m \times 1$  vector representing the input
- $y$  is a  $p \times 1$  vector representing the output

In Matlab state-space systems can be defined through the command:

```
sys = ss(A,B,C,D,Ts);
```

- $T_s$  is the sampling time (0 for continuous, 1 for discrete)

# Modeling

The state-space model for a **continuous-time** linear system is given by

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

The state-space model for a **discrete-time** linear system is given by

$$x[k+1] = Ax[k] + Bu[k]$$

$$y[k] = Cx[k] + Du[k]$$

- $x$  is an  $n \times 1$  vector representing the system's state variables
- $u$  is a  $m \times 1$  vector representing the input
- $y$  is a  $p \times 1$  vector representing the output

In Matlab state-space systems can be defined through the command:

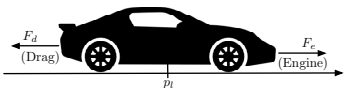
```
sys = ss(A,B,C,D,Ts);
```

- $T_s$  is the sampling time (0 for continuous, 1 for discrete)

# Modeling: longitudinal vehicle dynamics

## Example: longitudinal vehicle dynamics

$$m\ddot{p}_l = F_e - F_d$$



- $F_e$  : longitudinal engine force
- $F_d$  : drag force
- Assume linear friction:

$$F_d = \alpha \dot{p}_l$$

- Define:  $x_1 = p_l$  (position),  $x_2 = \dot{p}_l$  (velocity),

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\alpha/m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} F_e$$

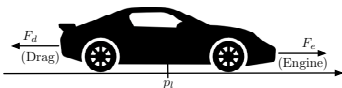
- If we can measure the position and velocity of the vehicle:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

# Modeling: longitudinal vehicle dynamics

## Example: longitudinal vehicle dynamics

$$m\ddot{p}_l = F_e - F_d$$



- $F_e$  : longitudinal engine force
- $F_d$  : drag force
- Assume linear friction:

$$F_d = \alpha \dot{p}_l$$

- Define:  $x_1 = p_l$  (position),  $x_2 = \dot{p}_l$  (velocity),

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\alpha/m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} F_e$$

- If we can measure the position and velocity of the vehicle:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

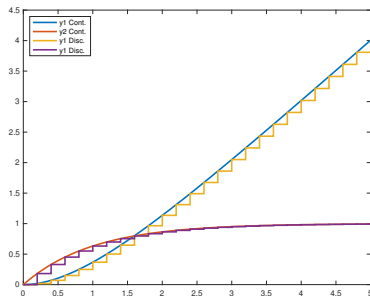
## Modeling: longitudinal vehicle dynamics (2)

```
1 alpha = 1;
2 m      = 1;
3
4 A = [0 1; 0 -alpha/m];           % System matrix
5 B = [0; 1/m];                   % Input matrix
6 C = [1 0; 0 1];                 % Output matrix
7 D = [0; 0];                     % Feedthrough matrix
8
9 sysC = ss(A,B,C,D,0);            % Continuous-time system
10
11 [Yc, Tc] = step(sysC,5);         % Step response with final time=5
12 figure                               % Create figure window
13 plot(Tc, Yc, 'Linewidth',2)
14
15 sysD = c2d(sysC,.2);             % Discretize the system
16
17 [Yd, Td] = step(sysD,5);         % Step response
18 hold on                             % Plot in the same figure window
19 stairs(Td, Yd, 'Linewidth',2)
```



# Modeling: longitudinal vehicle dynamics (3)

Step response:

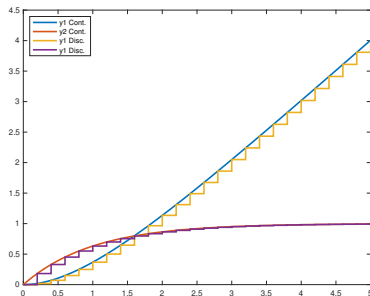


Discrete-time models are often obtained by discretizing continuous time physical equations

- Q: How do we check if the discrete-time system is stable?
- A: `abs(eig(Ad))`

# Modeling: longitudinal vehicle dynamics (3)

Step response:



Discrete-time models are often obtained by discretizing continuous time physical equations

- Q: How do we check if the discrete-time system is stable?
- A: `abs(eig(Ad))`

# Useful commands for state-space models

- Obtain step response of the system:  
`[Y, T] = step(sysD, x0, tFinal)`
- Obtain impulse response of the system:  
`[Y, T] = impulse(sysD)`
- Obtain impulse response to initial conditions:  
`[Y, T] = initial(sysD, x0, tFinal)`
- Discretize a continuous-time system:  
`sysD = c2d(sysC, .2)`
- Obtain transfer function of the system:  
`TrFcn = tf(sys)`

# Controllability, Observability, and State Feedback

## Controllability

A discrete-time linear system is controllable if, for any initial state  $x[0]$  and any desired state  $x_f$ , there is a nonnegative integer  $T$  and a sequence of inputs  $u[0], u[1], \dots, u[T]$  such that  $x[T + 1] = x_f$

## How do we check for controllability?

The system is controllable if and only if the rank of the controllability matrix is equal to the system size  $n$

In Matlab:

- `ctrb(A, B)` (returns the controllability matrix)
- `rank(ctrb(A, B))`

## Controllability

A discrete-time linear system is controllable if, for any initial state  $x[0]$  and any desired state  $x_f$ , there is a nonnegative integer  $T$  and a sequence of inputs  $u[0], u[1], \dots, u[T]$  such that  $x[T + 1] = x_f$

## How do we check for controllability?

The system is controllable if and only if the rank of the controllability matrix is equal to the system size  $n$

In Matlab:

- `ctrb(A, B)` (returns the controllability matrix)
- `rank(ctrb(A, B))`

## Observability

A discrete-time linear system is controllable if, for any initial state  $x[0]$  and any known sequence of inputs  $u[0], u[1], \dots$ , there is a nonnegative integer  $T$  such that  $x[0]$  can be recovered from the outputs  $y[0], y[1], \dots, y[T]$

## How do we check for observability?

The system is observable if and only if the rank of the observability matrix is equal to the system size  $n$

In Matlab:

- `obsv(A, C)` (returns the observability matrix)
- `rank(obsv(A, C))`

## Observability

A discrete-time linear system is controllable if, for any initial state  $x[0]$  and any known sequence of inputs  $u[0], u[1], \dots$ , there is a nonnegative integer  $T$  such that  $x[0]$  can be recovered from the outputs  $y[0], y[1], \dots, y[T]$

## How do we check for observability?

The system is observable if and only if the rank of the observability matrix is equal to the system size  $n$

In Matlab:

- `obsv(A, C)` (returns the observability matrix)
- `rank(obsv(A, C))`



# State feedback

## Controllability

We would like use the state of the system to construct a feedback input so that we can place the closed loop eigenvalues of the system at certain (stable) locations

$$u[k] = -Kx[k]$$

$$\begin{aligned}x[k+1] &= Ax[k] + Bu[k] = (A - BK)x[k] \\ y[k] &= (C - DK)x[k]\end{aligned}$$

## (Feedback control)

It is possible to arbitrarily place the closed loop eigenvalues via state feedback control if and only if the pair  $(A, B)$  is controllable

## State feedback (2)

The MATLAB commands `place` and `acker` can be used to find the matrix  $K$  such that the poles of  $A - BK$  have certain desired values

```
K = place(A,B,P)
```

```
K = acker(A,B,P)
```

```
1 K      = place(Ad,Bd,[.9 .8]); % Determine state feedback matrix
2
3 sysF    = ss(Ad-Bd*K, 0*Dd, Cd-Dd*K, 0*Dd); % Feedback system
4 [Yd, Td] = step(sysF,[1; 0], 10);           % Impulse response
5
6 figure
7 stairs(Td, Yd,'Linewidth',2)
```

# Dynamical systems with Matlab Simulink

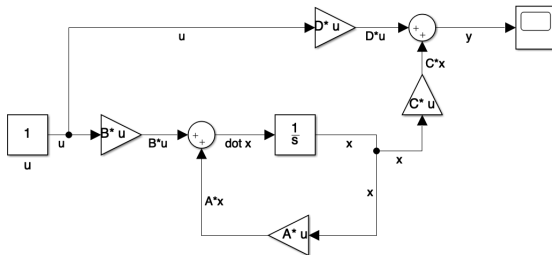
# Linear system in Simulink

Recall the expression of a continuous-time linear system:

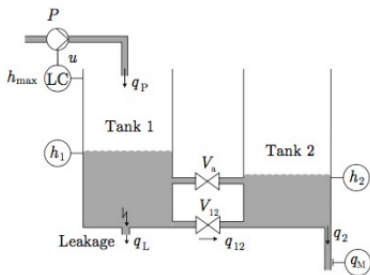
$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

We can study the step response by constructing its Simulink model:



# Two-tank system



Recall the dynamical equations of Tank 1:

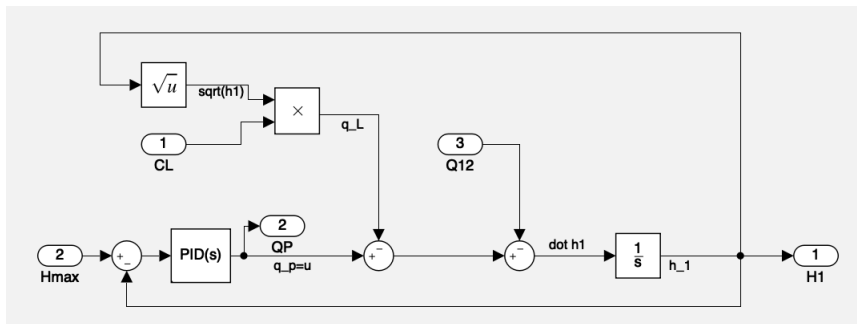
$$\dot{h}_1(t) = q_p(t) - q_L(t) - q_{12}(t)$$

$$q_L(t) = c_L(t) \sqrt{h_1(t)}$$

$$q_p(t) = u(t)$$

$$u(t) = \text{PID controller to regulate } h_1 \text{ to } h_{\max}$$

# Two-tank system: Simulink model



# Secure and reliable control systems: Matlab basics

Gianluca Bianchin  
gianluca@engr.ucr.edu  
Skype: giangi61978



Department of Mechanical Engineering  
University of California, Riverside

MSC 003 & Online - April 10, 2019