Software development for advanced controller design

Bc. Sofiia Serhiienko
Slovak University of Technology in Bratislava, Slovakia

Supervisor: Ing. Juraj Holaza, PhD.
Consultant: doc. Ing. Juraj Oravec, PhD.

STU FCHPT SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

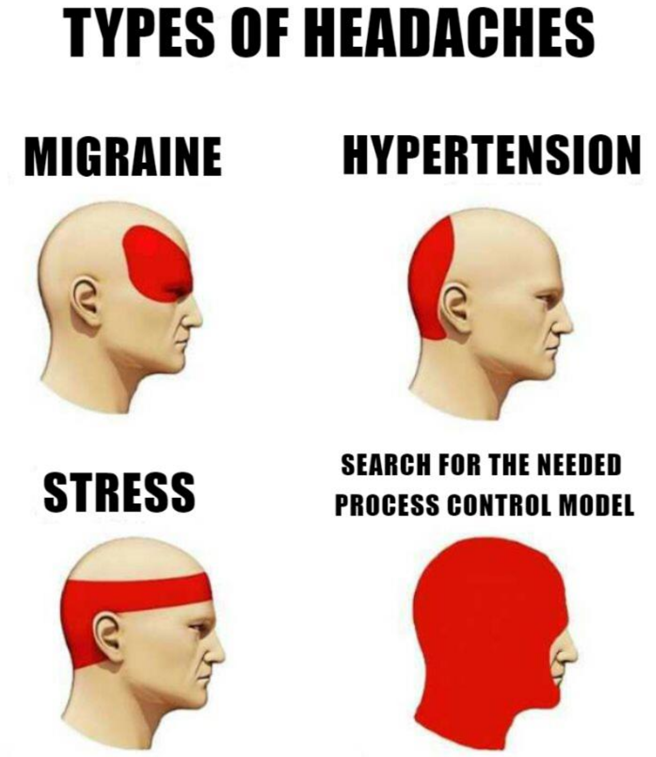
Introduction to Model Predictive Control (MPC)

$$
\begin{aligned}
\min \quad & \sum_{k=0}^{N} (x_k - x^s)^\top Q_x (x_k - x^s) \; + \sum_{k=0}^{N-1} (u_k - u^s)^\top Q_u (u_k - u^s) \\
\text{s.t.} \quad & x_{k+1} = A x_k + B u_k, \;\; k \in \mathbb{N}_0^{N-1} \\
& x_{\min} \leq x_k \leq x_{\max}, \;\; k \in \mathbb{N}_0^{N-1} \\
& u_{\min} \leq u_k \leq u_{\max}, \;\; k \in \mathbb{N}_0^{N-1} \\
& x_0 = x(t)
\end{aligned}
$$

Motivation

```
clear all, close all, clc
parameters.F1 = 1.2; parameters.F2 = 1.4;
parameters.F3 = 1.2; parameters.F4 = 1.3;
parameters.k11 =  0.8; parameters.k22 = 0.8;
parameters.k33 =  0.8; parameters.k44 =  0.8;
parameters.q01s = 1; parameters.q02s = 0.9;
parameters.h2s = ((parameters.q01s+parameters.q02s)/parameters.k22)^2;
parameters.h1s=parameters.h2s+(parameters.q01s/parameters.k11)^2;
parameters.h3s=((parameters.k22*sqrt(parameters.h2s))/(2*parameters.k33))^2;
parameters.h4s=((((parameters.k22*sqrt(parameters.h2s))/2)+(parameters.k33* ...
    sqrt(parameters.h3s)) )/parameters.k44)^2;

parameters.K1 = parameters.k11/(2*sqrt(parameters.h1s-parameters.h2s));
parameters.K2 = parameters.k22/(2*sqrt(parameters.h2s));
parameters.K3 = parameters.k33/(2*sqrt(parameters.h3s));
parameters.K4 = parameters.k44/(2*sqrt(parameters.h4s));
parameters.xs = [parameters.h1s;parameters.h2s;parameters.h3s;parameters.h4s];
parameters.us = [parameters.q01s;parameters.q02s];

dyn.sc.dif.A(1,1)=-parameters.K1/parameters.F1;
dyn.sc.dif.A(1,2)=parameters.K2/parameters.F1;
dyn.sc.dif.A(1,3)=0; dyn.sc.dif.A(1,4)=0;
dyn.sc.dif.A(2,1)=parameters.K1/parameters.F2;
dyn.sc.dif.A(2,2)=-(parameters.K1+parameters.K2)/parameters.F2;
dyn.sc.dif.A(2,3)=0; dyn.sc.dif.A(2,4)=0;
dyn.sc.dif.A(3,1)=0; dyn.sc.dif.A(3,2)=parameters.K2/(2*parameters.F3);
dyn.sc.dif.A(3,3)=-parameters.K3/parameters.F3; dyn.sc.dif.A(3,4)=0;
dyn.sc.dif.A(4,1)=0; dyn.sc.dif.A(4,2)=parameters.K2/(2*parameters.F4);
dyn.sc.dif.A(4,3)=parameters.K3/parameters.F4;
dyn.sc.dif.A(4,4)=-parameters.K4/parameters.F4;
dyn.sc.dif.B(1,1)= 1/parameters.F1; dyn.sc.dif.B(1,2)=0;
dyn.sc.dif.B(2,1)=0; dyn.sc.dif.B(2,2)= 1/parameters.F2;
dyn.sc.dif.B(3,1)=0;dyn.sc.dif.B(3,2)=0;
dyn.sc.dif.B(4,1)=0;dyn.sc.dif.B(4,2)=0;
dyn.sc.dif.C = [0 0 0 1];
dyn.sc.dif.D = 0;

dyn.sysc=ss(dyn.sc.dif.A,dyn.sc.dif.B,dyn.sc.dif.C,dyn.sc.dif.D);
lambda= eig(dyn.sysc.A);
dominant_lambda = max(abs(lambda));
T = 1/dominant_lambda;
dyn.Ts=T/4;% Ts =[T/5 T/2]
dyn.sysd=c2d(dyn.sysc,dyn.Ts );
dyn.sd.dif.A=dyn.sysd.A;
dyn.sd.dif.B=dyn.sysd.B;
dyn.sd.dif.C=dyn.sysd.C;
dyn.sd.dif.D=dyn.sysd.D;

properties.nx  = 4;
properties.nu = 2;

model= LTISystem('A',dyn.sd.dif.A,'B',dyn.sd.dif.B,'C',dyn.sd.dif.C,'Ts',dyn.Ts);

model.u.min = [0.1;0.1]-parameters.us;
model.u.max = [5;5]-parameters.us;
model.x.max=[10;10;10;10]-parameters.xs;
model.x.min=[0.5;0.5;0.5;0.5]-parameters.xs;

model.u.penalty = QuadFunction(eye(properties.nu));
model.x.penalty = QuadFunction(eye(properties.nx));

ctrl = MPCController(model, 6)
x0 = parameters.xs.*0;
Nsim = 101;
data = ctrl.simulate([1;1;1;1], 101);
```

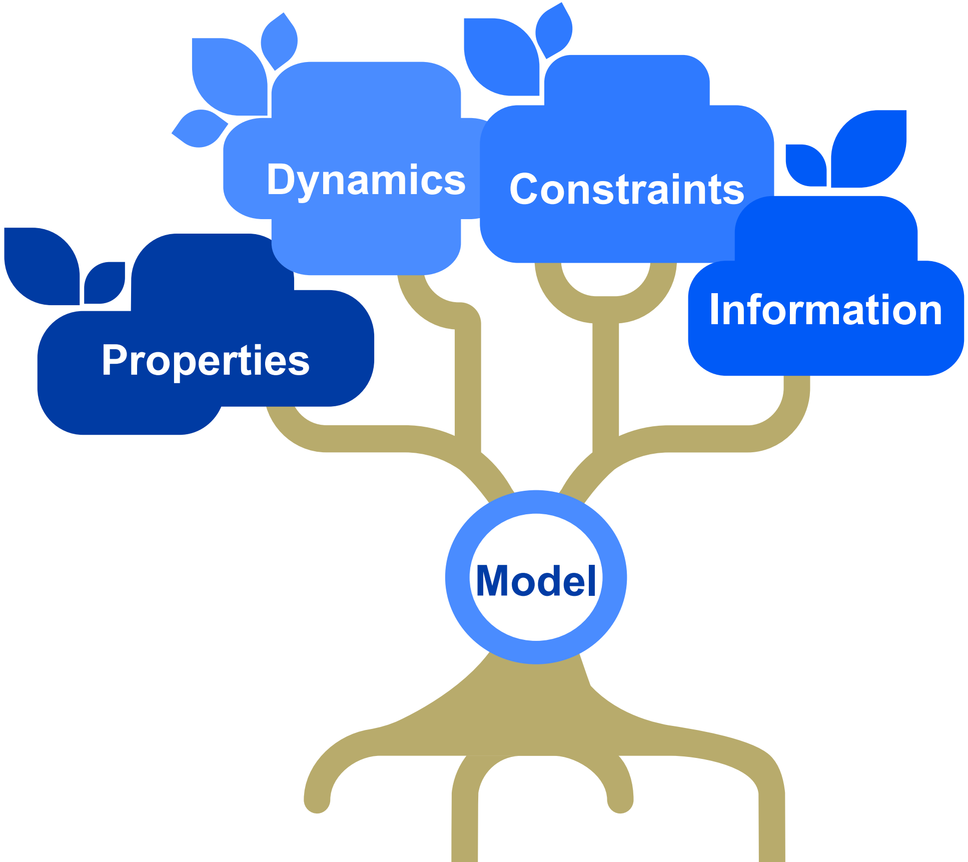
Problem Statement



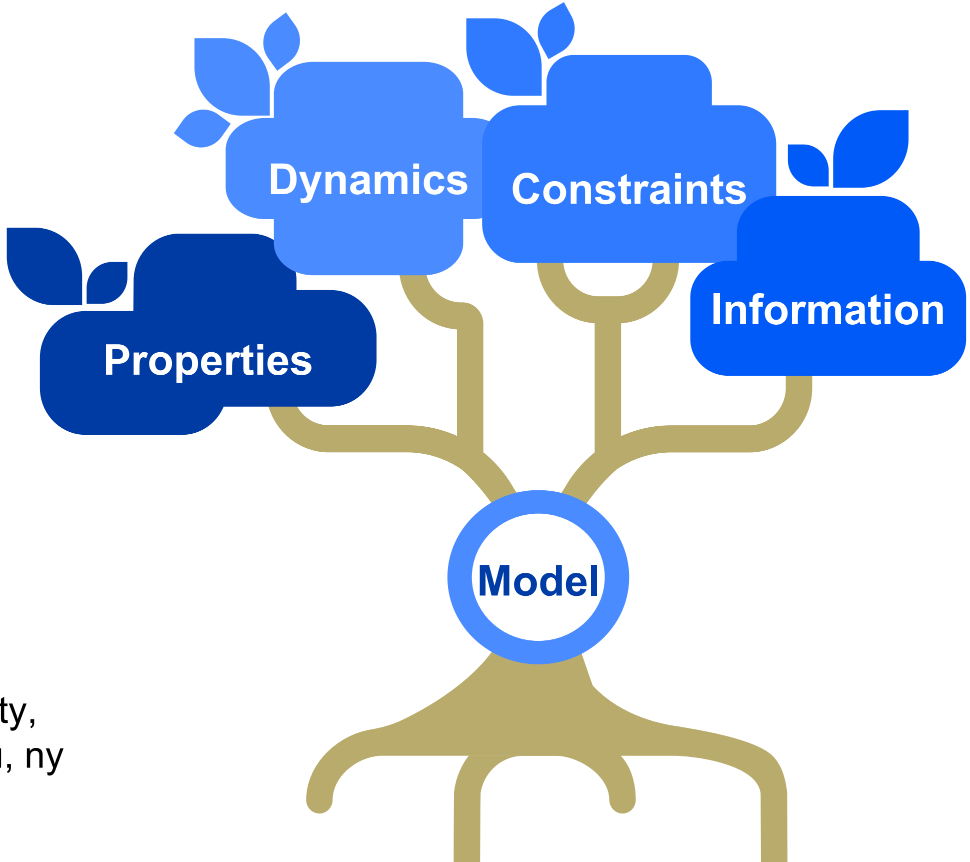
Moli

Maximize efficiency and save time with our innovative modeling toolbox

Model's Key Characteristics



Model's Key Characteristics



Dynamics

dyn.parameters,
dyn.sysc, dyn.sydc,
dyn.Ts etc.

Properties

Observability, stability,
controllability, nx, nu, ny

Constraints

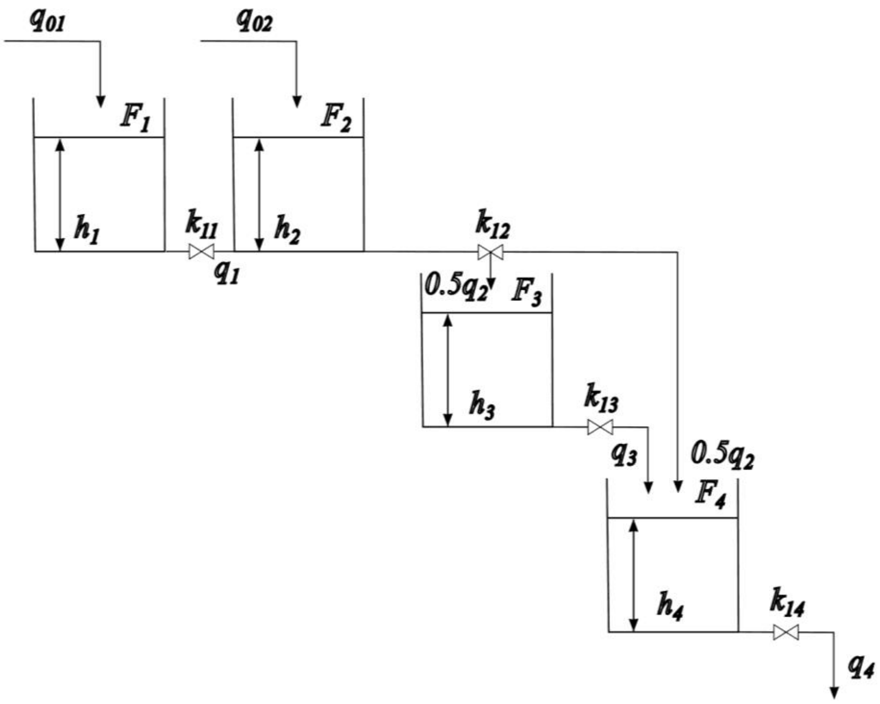
con.x.max/min,
con.u.max/min,
con.y.max/min

Information

Model description,
source (link), picture

Constructed Models

1 Moli_tanks

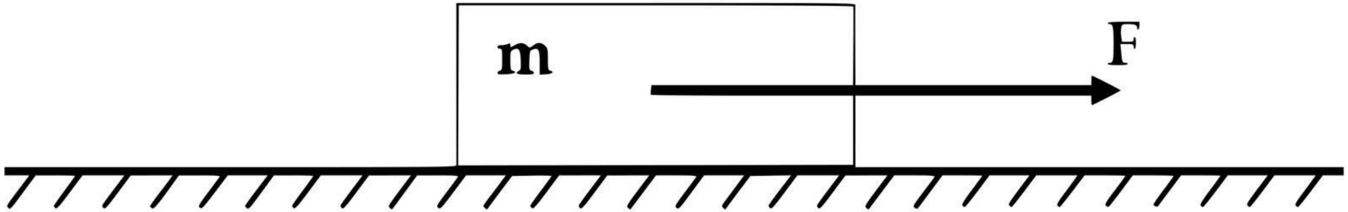



```
function [properties,dyn,con, info] = Moli_tanks()
```

Constructed Models

1 Moli_tanks

2 Moli_double_integrator



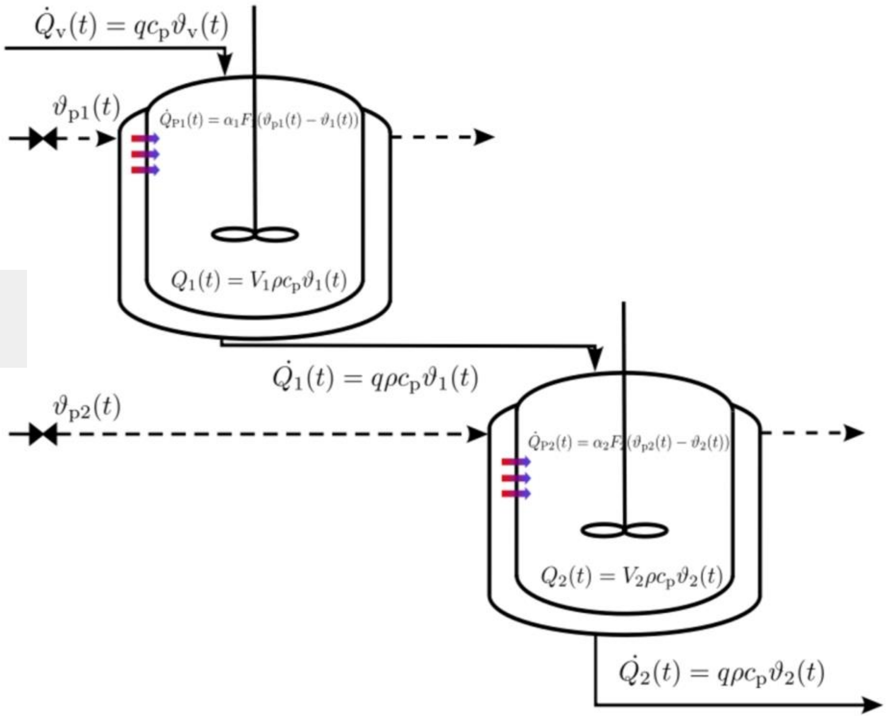

```
function [properties,dyn,con, info] = Moli_double_integrator()
```

Constructed Models

1 Moli_tanks

2 Moli_double_integrator

3 Moli_heat_exchangers




```
function [properties,dyn,con, info] = Moli_heat_exchangers()
```

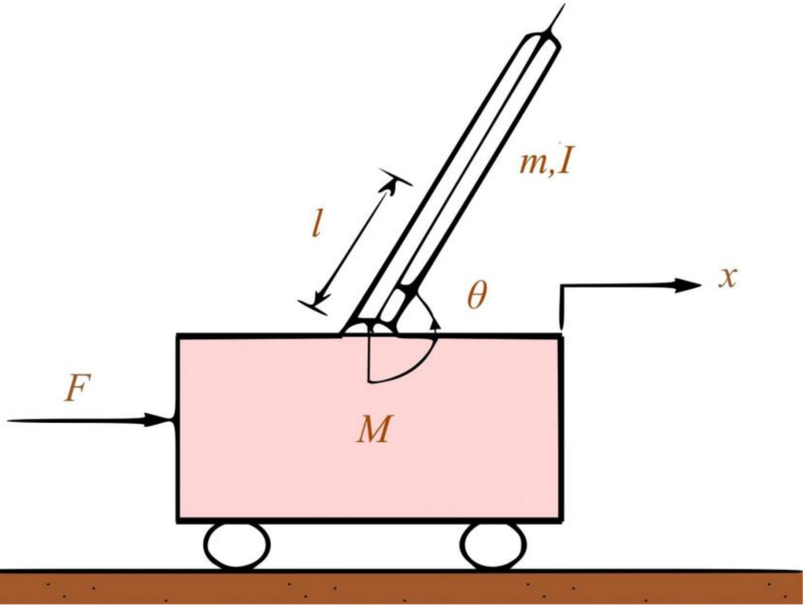
Constructed Models

1 Moli_tanks

2 Moli_double_integrator

3 Moli_heat_exchangers

4 Moli_inverted_pendulum




```
function [properties, dyn, con, info] = Moli_inverted_pendulum()
```

Constructed Models

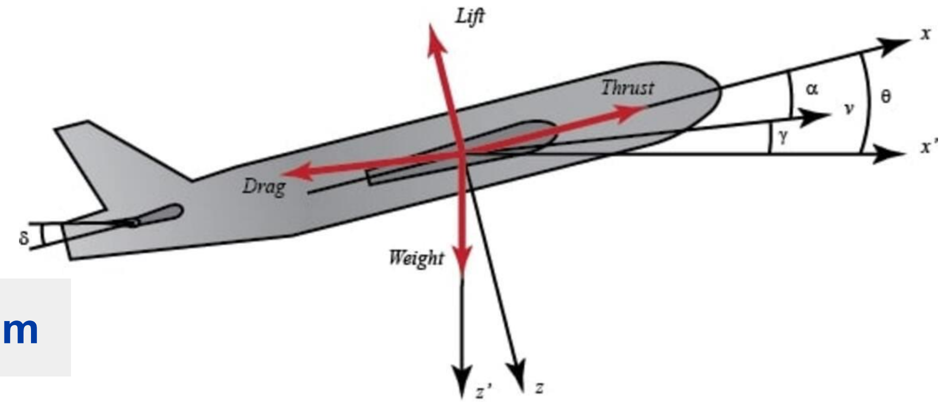
1 Moli_tanks

2 Moli_double_integrator

3 Moli_heat_exchangers

4 Moli_inverted_pendulum

5 Moli_aircraft_pitch




```
function [properties, dyn, con, info] = Moli_aircraft_pitch()
```

Effortless Access to Mathematical Models

```
M = Moli()
M.getListOfModels()
```

```
ans =

  5×5 table
```

id	Name_of_list	Number_of_states_nx	Number_of_inputs_nu	Number_of_outputs_ny
1	"Moli_aircraft_pitch"	3	1	1
2	"Moli_double_integrator"	2	1	1
3	"Moli_heat_exchangers"	2	2	1
4	"Moli_inverted_pendulum"	4	1	2
5	"Moli_tanks"	4	2	2

Model Filtering Process

```
>> Choose what to filter:
   s - States
   i - Inputs
   o - Outputs
   If you want to finish, please press Enter to exit.
   Enter your choice (s for States, i for Inputs, o for Outputs):
```

Model Filtering Process

```
Choose what to filter:
s - States
i - Inputs
o - Outputs
If you want to finish, please press Enter to exit.
Enter your choice (s for States, i for Inputs, o for Outputs): o
Outputs filtering...
Enter the desired number of outputs ny (choose one option from 1 to 5):
1: ny >=
2: ny <=
3: ny =
4: ny >
5: ny <
Enter your choice:
```

```
Enter your choice (s for States, i for Inputs, o for Outputs): o
Outputs filtering...
Enter the desired number of outputs ny (choose one option from 1 to 5):
1: ny >=
2: ny <=
3: ny =
4: ny >
5: ny <
Enter your choice: 3
Enter the desired number of outputs ny (more than 0): 2
    id           Name_of_list           Number_of_states_nx    Number_of_inputs_nu    Number_of_outputs_ny
    __    ________________________    ___________________    ___________________    ____________________

    4     "Moli_inverted_pendulum"             4                      1                      2
    5     "Moli_tanks"                         4                      2                      2

Choose an option (from 1 to 4):
1 - Filter outputs again
2 - Return to beginning
3 - The next round of filtering
4 - Exit
Enter your choice:
```

```
Choose an option (from 1 to 4):
1 - Filter outputs again
2 - Return to beginning
3 - The next round of filtering
4 - Exit
Enter your choice: 3
Choose what to filter:
s - States
i - Inputs
If you want to finish, please press Enter to exit.
Enter your choice (s for States, i for Inputs): i
Inputs filtering...
Enter the desired number of inputs nu (choose one option from 1 to 5):
1: nu >=
2: nu <=
3: nu =
4: nu >
5: nu <
Enter your choice: 3
Enter the desired number of inputs nu (more than 0): 1
```

```
Enter your choice: 3
Enter the desired number of inputs nu (more than 0): 1
```

id	Name_of_list	Number_of_states_nx	Number_of_inputs_nu	Number_of_outputs_ny
4	"Moli_inverted_pendulum"	4	1	2

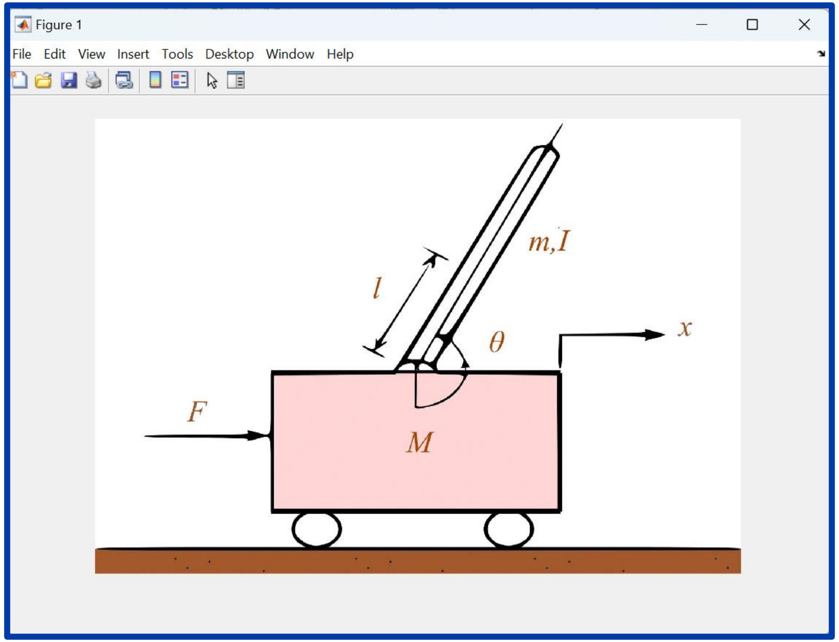
```
Choose an option (from 1 to 4):
1 - Filter inputs again
2 - Return to beginning
3 - Exit
4 - The next round of filtering
Enter your choice: 3
Exiting...

ans =

  1×5 table
```

id	Name_of_list	Number_of_states_nx	Number_of_inputs_nu	Number_of_outputs_ny
4	"Moli_inverted_pendulum"	4	1	2

Customized Model Based on Required Inputs and Outputs

Visualization of Desired Model

```
M.getModel('Moli_inverted_pendulum')
figure, M.ShowPicture()
```

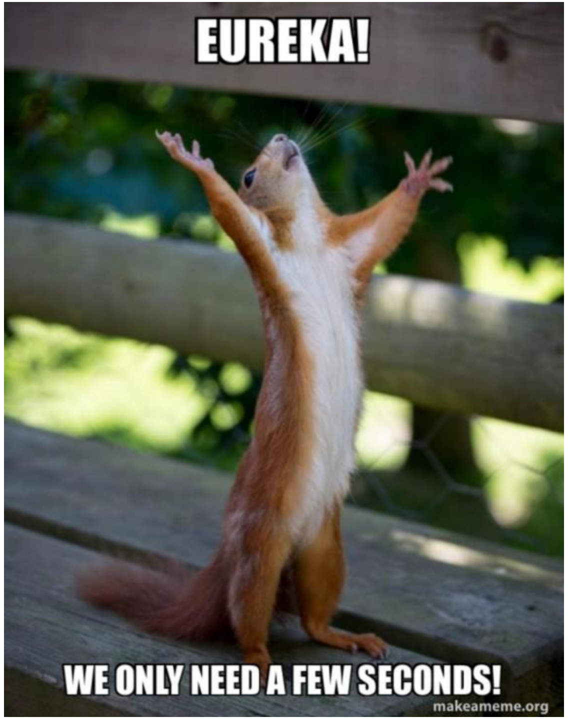


65 Lines of Code

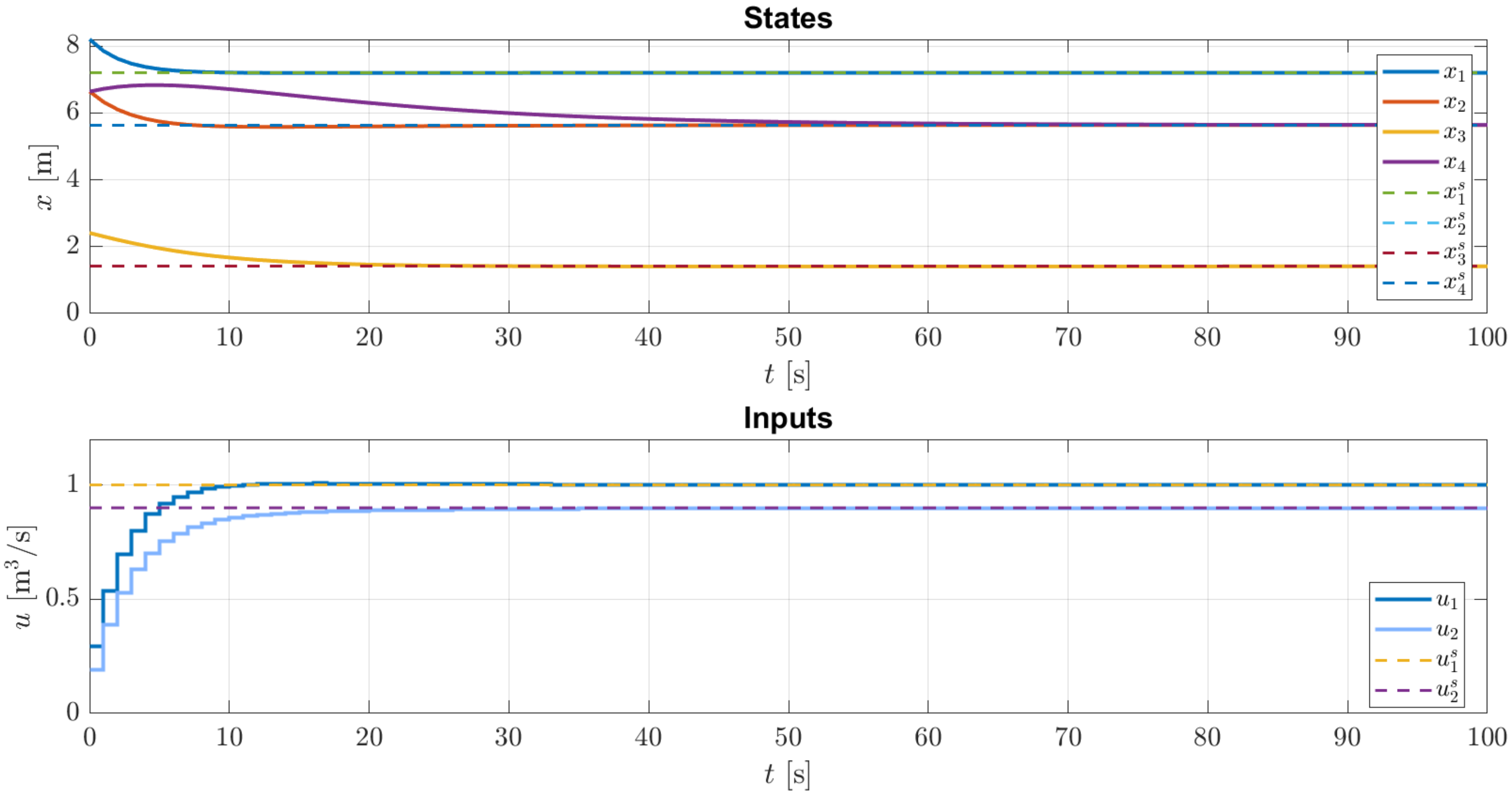
```
clear all, close all, clc
parameters.F1 = 1.2; parameters.F2 = 1.4;
parameters.F3 = 1.2; parameters.F4 = 1.3;
parameters.k11 =  0.8; parameters.k22 = 0.8;
parameters.k33 =  0.8; parameters.k44 =  0.8;
parameters.q01s = 1; parameters.q02s = 0.9;
parameters.h2s = ((parameters.q01s+parameters.q02s)/parameters.k22)^2;
parameters.h1s=parameters.h2s+(parameters.q01s/parameters.k11)^2;
parameters.h3s=((parameters.k22*sqrt(parameters.h2s))/(2*parameters.k33))^2;
parameters.h4s=((((parameters.k22*sqrt(parameters.h2s))/2)+(parameters.k33* ...
    sqrt(parameters.h3s)) )/parameters.k44)^2;

parameters.K1 = parameters.k11/(2*sqrt(parameters.h1s-parameters.h2s));
parameters.K2 = parameters.k22/(2*sqrt(parameters.h2s));
parameters.K3 = parameters.k33/(2*sqrt(parameters.h3s));
parameters.K4 = parameters.k44/(2*sqrt(parameters.h4s));
parameters.xs = [parameters.h1s;parameters.h2s;parameters.h3s;parameters.h4s];
parameters.us = [parameters.q01s;parameters.q02s];

dyn.sc.dif.A(1,1)=-parameters.K1/parameters.F1;
dyn.sc.dif.A(1,2)=parameters.K2/parameters.F1;
dyn.sc.dif.A(1,3)=0; dyn.sc.dif.A(1,4)=0;
dyn.sc.dif.A(2,1)=parameters.K1/parameters.F2;
dyn.sc.dif.A(2,2)=-(parameters.K1+parameters.K2)/parameters.F2;
dyn.sc.dif.A(2,3)=0; dyn.sc.dif.A(2,4)=0;
dyn.sc.dif.A(3,1)=0; dyn.sc.dif.A(3,2)=parameters.K2/(2*parameters.F3);
dyn.sc.dif.A(3,3)=-parameters.K3/parameters.F3; dyn.sc.dif.A(3,4)=0;
dyn.sc.dif.A(4,1)=0; dyn.sc.dif.A(4,2)=parameters.K2/(2*parameters.F4);
dyn.sc.dif.A(4,3)=parameters.K3/parameters.F4;
dyn.sc.dif.A(4,4)=-parameters.K4/parameters.F4;
dyn.sc.dif.B(1,1)= 1/parameters.F1; dyn.sc.dif.B(1,2)=0;
dyn.sc.dif.B(2,1)=0; dyn.sc.dif.B(2,2)= 1/parameters.F2;
dyn.sc.dif.B(3,1)=0;dyn.sc.dif.B(3,2)=0;
dyn.sc.dif.B(4,1)=0;dyn.sc.dif.B(4,2)=0;
dyn.sc.dif.C = [0 0 0 1];
dyn.sc.dif.D = 0;

dyn.sysc=ss(dyn.sc.dif.A,dyn.sc.dif.B,dyn.sc.dif.C,dyn.sc.dif.D);
lambda= eig(dyn.sysc.A);
dominant_lambda = max(abs(lambda));
T = 1/dominant_lambda;
dyn.Ts=T/4;% Ts =[T/5 T/2]
dyn.sysd=c2d(dyn.sysc,dyn.Ts );
dyn.sd.dif.A=dyn.sysd.A;
dyn.sd.dif.B=dyn.sysd.B;
dyn.sd.dif.C=dyn.sysd.C;
dyn.sd.dif.D=dyn.sysd.D;

properties.nx  = 4;
properties.nu = 2;

model= LTISystem('A',dyn.sd.dif.A,'B',dyn.sd.dif.B,'C',dyn.sd.dif.C,'Ts',dyn.Ts);

model.u.min = [0.1;0.1]-parameters.us;
model.u.max = [5;5]-parameters.us;
model.x.max=[10;10;10;10]-parameters.xs;
model.x.min=[0.5;0.5;0.5;0.5]-parameters.xs;

model.u.penalty = QuadFunction(eye(properties.nu));
model.x.penalty = QuadFunction(eye(properties.nx));

ctrl = MPCController(model, 6)
x0 = parameters.xs.*0;
Nsim = 101;
data = ctrl.simulate([1;1;1;1], 101);
```

Simulation of the System Controlled by an MPC Controller in Just 6 Lines of Code

```
M = Moli();
M.getListOfModels();
M.getModel('Moli_tanks');
modelLTI = M.getMPTModel();
ctrl = MPCController(modelLTI,5);
data = ctrl.simulate([1;1;1;1], 101);
```



Control Outcome Using MPC Controller



Future Goals

- Enhanced Filtering Options
- Model Monasteries
- User-Created Models
- GUI Development
- Developing Observers and Controllers

The greatest value of your work is not what you achieve, but how you save time for others.

Do you have any questions?

xsergienko@stuba.sk

SCAN ME

Moli

Backup Slides

What will happen if the wrong input is entered?

```
Outputs filtering...
Enter the desired number of outputs ny (choose one option from 1 to 5):
1: ny >=
2: ny <=
3: ny =
4: ny >
5: ny <
Enter your choice: 6
Invalid choice. Please enter a number from 1 to 5.
Enter the desired number of outputs ny (choose one option from 1 to 5):
1: ny >=
2: ny <=
3: ny =
4: ny >
5: ny <
Enter your choice: u
Error using input
Unrecognized function or variable 'u'.
Error in filter_table_by_number (line 18)
        choice = input('Enter your choice: ');
Error in Moli/getListOfModels (line 351)
    filtered_table = filter_table_by_number(T0, 'Number_of_outputs_ny', 'outputs ny');
Error in LiveEditorEvaluationHelperE1301006382 (line 9)
M.getListOfModels(1)
Enter your choice: 3
Enter the desired number of outputs ny (more than 0): 5
Sorry, but no model with the specified number of outputs ny exists.
Choose an option (from 1 to 4):
1 - Filter outputs again
2 - Return to beginning
3 - The next round of filtering
4 - Exit
Enter your choice:
```

Initializing the Model

```
>> M = Moli();
M.getListOfModels();
M.getModel('Moli_tanks');
modelLTI = M.getMPTModel();
The system is fully observable!
The system is stable!
The system is fully controllable!
>> M.dynamics.params
ans =
  struct with fields:

      F1: 1.2000
      F2: 1.4000
      F3: 1.2000
      F4: 1.3000
     k11: 0.8000
     k22: 0.8000
     k33: 0.8000
     k44: 0.8000
    q01s: 1
    q02s: 0.9000
     h2s: 5.6406
     h1s: 7.2031
     h3s: 1.4102
     h4s: 5.6406
      K1: 0.3200
      K2: 0.1684
      K3: 0.3368
      K4: 0.1684
      xs: [4×1 double]
      us: [2×1 double]
```

```
>> M.dynamics.paramsT1
ans =
  10×4 table
```

Symbol	Description	Value	Unit
"q01"	"Set flow rate for tank 1"	1	"m^3/s"
"q02"	"Set flow rate for tank 2"	0.9	"m^3/s"
"F1"	"Cross-sectional area of tank 1"	1.2	"m^2"
"F2"	"Cross-sectional area of tank 2"	1.4	"m^2"
"F3"	"Cross-sectional area of tank 3"	1.2	"m^2"
"F4"	"Cross-sectional area of tank 4"	1.3	"m^2"
"k11"	"Valve constant for tank 1"	0.8	"m^2.5/s"
"k22"	"Valve constant for tank 2"	0.8	"m^2.5/s"
"k33"	"Valve constant for tank 3"	0.8	"m^2.5/s"
"k44"	"Valve constant for tank 4"	0.8	"m^2.5/s"

```
>> M.dynamics.paramsT2
ans =
  1×3 table
```

States				Inputs		Outputs
"h1"	"h2"	"h3"	"h4"	"q01"	"q02"	"h4"

Parameters of the System

```
%% Parameters of the system
parameters.F1 = 1.2; parameters.F2 = 1.4; parameters.F3 = 1.2; parameters.F4 = 1.3;
parameters.k11 =  0.8; parameters.k22 = 0.8; parameters.k33 =  0.8; parameters.k44 =  0.8;
parameters.q01s = 1; parameters.q02s = 0.9;
parameters.h2s = ((parameters.q01s+parameters.q02s)/parameters.k22)^2;
parameters.h1s=parameters.h2s+(parameters.q01s/parameters.k11)^2;
parameters.h3s=((parameters.k22*sqrt(parameters.h2s))/(2*parameters.k33))^2;
parameters.h4s=( ( ((parameters.k22*sqrt(parameters.h2s))/2)+(parameters.k33* ...
    sqrt(parameters.h3s)) )/parameters.k44)^2;

parameters.K1 = parameters.k11/(2*sqrt(parameters.h1s-parameters.h2s));
parameters.K2 = parameters.k22/(2*sqrt(parameters.h2s));
parameters.K3 = parameters.k33/(2*sqrt(parameters.h3s));
parameters.K4 = parameters.k44/(2*sqrt(parameters.h4s));
parameters.xs = [parameters.h1s;parameters.h2s;parameters.h3s;parameters.h4s];
parameters.us = [parameters.q01s;parameters.q02s];

data = {
    string('q01'),    string('Set flow rate for tank 1'), 1,    string('m^3/s');
    'q02', 'Set flow rate for tank 2', 0.9, 'm^3/s';
    'F1', 'Cross-sectional area of tank 1', 1.2, 'm^2';
    'F2', 'Cross-sectional area of tank 2', 1.4, 'm^2';
    'F3', 'Cross-sectional area of tank 3', 1.2, 'm^2';
    'F4', 'Cross-sectional area of tank 4', 1.3, 'm^2';
    'k11', 'Valve constant for tank 1', 0.8, 'm^2.5/s';
    'k22', 'Valve constant for tank 2', 0.8, 'm^2.5/s';
    'k33', 'Valve constant for tank 3', 0.8, 'm^2.5/s';
    'k44', 'Valve constant for tank 4', 0.8, 'm^2.5/s'};
T1 = cell2table(data, 'VariableNames', {'Symbol', 'Description', 'Value', 'Unit'});
%disp(T1)

states = {'h1'; 'h2'; 'h3'; 'h4'};  states = string(states);
inputs = {'q01';'q02'}; inputs = string(inputs);
outputs = {'h4'}; outputs = string(outputs);

T2 = table(states', inputs', outputs');
T2.Properties.VariableNames = {'States', 'Inputs', 'Outputs'};
%disp(T2)
```

System Dynamics

```
%% Dynamics
% This code defines the dynamics of a system, specifically a linear
% time-invariant (LTI) system.

% State transition matrix. Describes how the state of the system evolves over time.
dyn.sc.dif.A(1,1)=-parameters.K1/parameters.F1; dyn.sc.dif.A(1,2)=parameters.K2/parameters.F1;
dyn.sc.dif.A(1,3)=0; dyn.sc.dif.A(1,4)=0;
dyn.sc.dif.A(2,1)=parameters.K1/parameters.F2; dyn.sc.dif.A(2,2)=-(parameters.K1+parameters.K2)/parameters.F2;
dyn.sc.dif.A(2,3)=0; dyn.sc.dif.A(2,4)=0;
dyn.sc.dif.A(3,1)=0; dyn.sc.dif.A(3,2)=parameters.K2/(2*parameters.F3);
dyn.sc.dif.A(3,3)=-parameters.K3/parameters.F3; dyn.sc.dif.A(3,4)=0;
dyn.sc.dif.A(4,1)=0; dyn.sc.dif.A(4,2)=parameters.K2/(2*parameters.F4);
dyn.sc.dif.A(4,3)=parameters.K3/parameters.F4; dyn.sc.dif.A(4,4)=-parameters.K4/parameters.F4;
% Input matrix. Describes how the inputs affect the state evolution.
dyn.sc.dif.B(1,1)= 1/parameters.F1; dyn.sc.dif.B(1,2)=0;
dyn.sc.dif.B(2,1)=0; dyn.sc.dif.B(2,2)= 1/parameters.F2;
dyn.sc.dif.B(3,1)=0;dyn.sc.dif.B(3,2)=0;
dyn.sc.dif.B(4,1)=0;dyn.sc.dif.B(4,2)=0;
% Output matrix. Describes how the states are mapped to the outputs.
dyn.sc.dif.C = [0 0 0 1];
% Feedthrough matrix. Describes direct feedthrough from inputs to outputs.
dyn.sc.dif.D = 0;

% This is an optional parameter representing any external disturbances
% acting on the system.
dyn.sc.dif.f=[];
% Another optional parameter representing any external disturbances acting
% on the output.
dyn.sc.dif.e=[];
% Similarly, dyn.sd.dif.f and dyn.sd.dif.e represent the same parameters
% for the system after it has been converted to discrete-time(sd).

% Sampling time
dyn.Ts=[0.5086];

% The continuous-time state-space model defined by matrices dyn.sc.dif.A,
% dyn.sc.dif.B, dyn.sc.dif.C, and dyn.sc.dif.D is converted to discrete-time
% using the function c2d.
dyn.sysc=ss(dyn.sc.dif.A,dyn.sc.dif.B,dyn.sc.dif.C,dyn.sc.dif.D);
dyn.sysd=c2d(dyn.sysc,dyn.Ts );
```

```
% The resulting discrete-time state-space model is stored in the matrices
% dyn.sd.dif.A, dyn.sd.dif.B, dyn.sd.dif.C, and dyn.sd.dif.D.
dyn.sd.dif.A=dyn.sysd.A;
dyn.sd.dif.B=dyn.sysd.B;
dyn.sd.dif.C=dyn.sysd.C;
dyn.sd.dif.D=dyn.sysd.D;

dyn.sd.dif.f=[];
dyn.sd.dif.e=[];

% dyn.Q, dyn.R, dyn.S: These are matrices used in the calculation of the
% cost function for optimal control. They define the importance of minimizing
% the state, input, and output deviations from desired values, respectively.
dyn.Q=[];
dyn.R=[];
dyn.S=[];

% dyn.gu, dyn.gy: These are optional parameters representing any additional
% terms in the cost function related to inputs and outputs.
dyn.gu=[];
dyn.gy=[];

dyn.params = parameters;
dyn.paramsT1 = T1;
dyn.paramsT2 = T2;
```

Properties of the System

```
%% Properties
% Check observability
obscon = rank(obsv(dyn.sc.dif.A, dyn.sc.dif.C)) == size(dyn.sc.dif.A, 1);
if obscon
    fprintf('The system is fully observable!\n');
    properties.observability = true;
else
    fprintf('The system is NOT fully observable!\n');
    properties.observability = false;
end

% Check stability
lambda_sys = eig(dyn.sc.dif.A);
stabcon = all(lambda_sys < 0);
if stabcon
    fprintf('The system is stable!\n');
    properties.stability = true;
else
    fprintf('The system is NOT stable!\n');
    properties.stability = false;
end

% Check controllability
C_0 = ctrb(dyn.sc.dif.A, dyn.sc.dif.B);
nx_ctrb = rank(C_0);
if nx_ctrb == size(dyn.sc.dif.A, 2)
    fprintf('The system is fully controllable!\n');
    properties.controllability = true;
else
    fprintf('The system is NOT fully controllable!\n');
    properties.controllability = false;
end

[properties.nx, properties.nu] = size(dyn.sc.dif.B);
properties.ny = size(dyn.sc.dif.C,1);
```

Constraints

```
%% Constraints
% Constraints on the states to the system.
con.x.max=[10;10;10;10];
con.x.min=[0.5;0.5;0.5;0.5];
% Constraints on the inputs to the system.
con.u.min=[0.1;0.1];
con.u.max=[5;5];
% Constraints on the outputs of the system.
con.y.min=[];
con.y.max=[];
```

Information

```
%% Info
%info.source= [''];
info.text = ['A system consisting of four sequentially connected tanks with' ...
    ' interdependencies is under investigation. Liquid flows to the first' ...
    ' tank with an adjustable flow-rate q01 and to the second tank with an' ...
    ' adjustable flow-rate q02It is presumed that the density of the liquid' ...
    ' remains uniform across the entirety of the system. Measurement of ' ...
    'the liquid level is solely feasible in the fourth tank.'];
%image_path = 'C:\Users\User\Documents\MATLAB\Toolbox\Moli\ModelLibrary\image\tanks4j.jpg';
temp = which('Moli_tanks');
temp = split(temp,'Moli_tanks.m');
PathModels = temp{1};
last_backslash_index = find(PathModels == '\', 1, 'last');
PathModels = PathModels(1:last_backslash_index-1);

image_path = [PathModels , filesep , 'image', filesep , 'tanks4j.jpg'];
info.image_data = imread(image_path);
%info.image= imshow(info.image_data);
```