Advanced Model Predictive Control

Recitation 2
Introduction to Code Framework

Alexandre Didier and Jérôme Sieber

ETH Zurich

Fall 2022

Programming Exercises

- All programming exercises (PEs) use a common MATLAB code framework
- You have two weeks to solve a PE (see tentative timetable)
- Hand in via Moodle (.zip file)
- You have to solve the PE individually (no groups allowed)
- Solving 5 out of 8 PEs correctly, awards you a bonus of 0.25 grade points

Programming Exercise	Hand Out	Hand In
PE 1	29.09.	12.10.
PE 2	06.10.	19.10.
PE 3	13.10.	26.10.
PE 4	03.11.	16.11.
PE 5	24.11.	07.12.
PE 6	01.12.	14.12.
PE 7	08.12.	21.12.
PE 8	15.12.	28.12.

MATLAB Code Framework for AMPC 2022

For all programming exercises we will use a unified MATLAB code framework, which consists of the following parts:

- **Parameter File:** `params_PE1.m` defines system, control, and simulation parameters; `get_params.m` loads these parameters to the main file.
- **Main File:** `main_PE1.m` initializes and simulates the complete control loop. You will only execute this file.
- **System Class:** `System.m` is an abstract parent class of all systems used in the programming exercises (PEs).
- **Control Class:** `Controller.m` is an abstract parent class of all controllers implemented in the PEs.
- **Parameter Estimator Class:** `Parameter_Estimator.m` is an abstract parent class of all parameter estimators implemented in the PEs.

```
code/
├── controllers/
│   ├── MPC.m
│   └── Nonlinear_MPC.m
├── parameters/
│   ├── get_params.m
│   └── params_PE1.m
├── systems/
│   └── NonlinearSystem.m
├── Controller.m
├── Parameter_Estimator.m
├── System.m
├── main_PE1.m
└── setup.m
```

Main File

- This file performs the following actions:
  - load system and control parameters from parameter file (`params_xy.m`),
  - initialize the system and controller objects,
  - simulate a defined number of closed-loop trajectories with a defined number of time steps,
  - plot the results of the closed-loop simulations.
- Execute only this file.

MATLAB Pseudocode:

```
% get parameters and define system ...
    & controller
params = get_params('params_PE1');
sys = NonlinearSystem(params.sys);
ctrl = Nonlinear_MPC(sys, ...
    params.ctrl);

% control loop
x(1) = params.sim.x_0;
for i=1:nrTraj
    for j=1:nrSteps
        u = ctrl.solve(x(j));
        x(j+1) = sys.step(x(j),u);
    end
end

% plot results
plot(x); plot(u);
```

System Class

- `System` implements an abstract class from which `LinearSystem` and `NonlinearSystem` inherit.
- The class defines the interfaces between the system and any controller derived from the `Controller` class.
- You should not modify this class.
- However, feel free to go through the code and play with it.

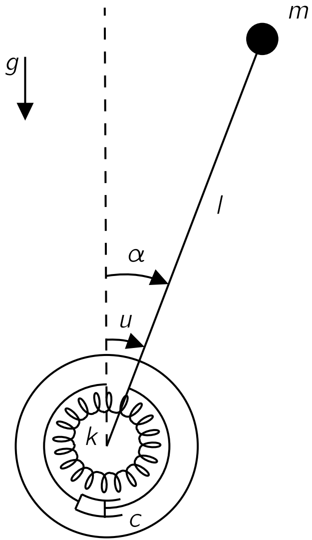
MATLAB Pseudocode:

```
classdef System
%System class for segway system
properties
  % class variables
end

methods
  % class methods
  function obj = System(params)
      %Class Constructor
  end
  function x1 = step(obj,x,u)
      %advance system from state x ...
          with input u
  end
  function y1 = get_output(obj,x,u)
      %evaluate output for state x ...
          and input u
  end
end
end
```

Segway System

[freepik.com]



We only consider the rotational dynamics of the segway and discretize the dynamics using Euler forward:

$$\begin{bmatrix} \alpha(k+1) \\ \dot{\alpha}(k+1) \end{bmatrix} = \begin{bmatrix} \alpha(k) + \delta t \cdot \dot{\alpha}(k) \\ \dot{\alpha}(k) + \delta t(-k\alpha(k) - c\dot{\alpha}(k) + {}^{g}/_{l} \cdot \sin\alpha(k) + u(k)) \end{bmatrix}$$

Controller Class

- All controllers you will implement in the programming exercises inherit from this class.
- We provide you with a nominal MPC (`MPC`) implementation as a reference.
- All controller classes need to define the property `prob`.
- The `solve` method works for both Yalmip and CasADi `prob` objects.

MATLAB Pseudocode:

```
classdef Controller
%Parent controller class
properties
  % class variables
  prob % optimizer/solver object
end

methods
  % class methods
  function obj = ...
      Controller(params)
    %Class Constructor
  end
  function [u, out, info] = ...
      solve(obj, x, vars, verbose)
    %solve optimization ...
        problem with initial ...
        state x and auxilliary ...
        variables vars
  end
end
end
```

Parameter Estimator Class

- All parameter estimators you will implement in the programming exercises inherit from this class.
- We will use this class only in the second part of the course.

MATLAB Pseudocode:

```
classdef Parameter_Estimator
%Parent parameter estimator class
properties
  % class variables
end

methods
  % class methods
  function obj = ...
        Parameter_Estimator(sys)
        %Class Constructor
  end
end
end
```

Programming Exercise 1 (Nominal Nonlinear MPC)

Nominal Nonlinear MPC Problem:

$$
\begin{aligned}
\min_{x,u} \quad & l_f(x_N) + \sum_{i=0}^{N-1} l(x_i, u_i) \\
\text{s.t.} \quad & \forall i = 0, \dots, N-1 \\
& x_{i+1} = f(x_i, u_i) \\
& x_i \in \mathcal{X} \\
& u_i \in \mathcal{U} \\
& x_N \in \mathcal{X}_f \\
& x_0 = x(k)
\end{aligned}
$$

First, make yourself familiar with the code base using the provided MPC example, i.e., running `main_MPC.m` and examining the files `main_MPC.m`, `params_MPC.m`, and `MPC.m`. Then, solve the programming exercise.

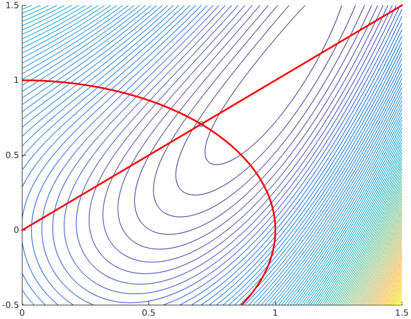
Nonlinear Programming using CasADi

Nonlinear Programming (NLP) Example:

$$\min_{x,y} \quad (1-x)^2 + \left(y - x^2\right)^2$$

$$\text{s.t.} \quad x^2 + y^2 = 1$$

$$x \leq y$$



MATLAB Code:

```matlab
% initialize Opti stack
prob = casadi.Opti();

% define decision variables
x = prob.variable();
y = prob.variable();

% objective
prob.minimize((1-x)^2+(y-x^2)^2);

% constraints
prob.subject_to(x^2+y^2==1);
prob.subject_to(y>=x);

% solve NLP
prob.solver('ipopt');
sol = prob.solve();
```

[Source: CasADi Opti Stack Documentation]