Pyomo Notes

Brittany Hall

1 Introduction

A typical implementation of a (N)MPC controller requires seven steps:

1. Define a dynamic model that represents accurately enough the process
2. Define an objective function for the controller and (maybe) a tracking quadratic function
3. Discretize the model so the OCP can be solved using QP or NLP algorithms which can exploit the structure of the model
4. Read the current states of the system and process the information
5. Solve the OCP for the current time interval
6. Process the OCP solution and define the value of the manipulated variables that are feed-back to the process
7. Move to the next sampling period.

Step 5 is critical for the closed loop performance of the controller because it can introduce a significant delay produced by the time required for solving the dynamic optimization problem. An alternative would be to use NLP sensitivity to quickly update online the values of the MVs. This integration of NMPC with a fast estimation of the optimal solution is called advanced step NMPC (asNMPC). The NLP sensitivity theory is a powerful tool that can provide fast and accurate estimation of the solution of a nonlinear optimization problem when the parameters of the problem are perturbed in a small region. It can also be used in the state estimation problem.

We will now look at a NMPC framework which has been developed in Pyomo, an open source-tool for modelling optimization problems and applications in Python. Pyomo has been developed in a modular structure so that users can easily access different parts of the software or customize them without the risk of altering core functionalities. Users can also customize modelling components or the optimization process according to their needs for certain applications.

2 Design Goals

Currently the NMPC framework uses Pyomo, NumPy, and the pseudo solver gjh for the sensitivity calculations. The design goals for this framework are outlined below:

- Open-source:
- Avoid communication between programs:

- Automation of the controller set-up: It is expected that the user knows about defining models in AMLs. This is the main input of the framework and afterwards, the additional steps are done automatically. The framework makes extensive use of the DAE module of Pyomo. This allows user to incorporate differential equations within the model. The pyomo.dae module includes several important modelling classes. It can also transform a dynamic model defined in continuous time to a model in discrete time, where the user can select the discretization scheme. In summary, the proposed NMPC framework discretizes the whole model after a discretization scheme has been selected, it defines the control horizon constraints, it defines the manipulated variables as piece-wise functions for each sampling time, it does sensitivity calculations to correct model plant mismatch if necessary, it communicates directly with the solver to pass or extract the results and it presents the results dynamically for a better visualization and testing.
- Compatibility with different models: The user can define any kind of model as long as it is stated as a mathematical program. The only requirement is that the user has installed an adequate solver for the type of modeling that is solving. Pyomo can be integrated with any optimization solver.

3 Description of the framework for asNMPC in Pyomo

This is an extension of Pyomo and can thus be used in any Python script by import the module and it also inherits all the functions of Pyomo.

3.1 Advanced Step NMPC (asNMPC)

The idea of asNMPC is to solve the OCP problem in advance, which means that at the current time t_k and with the measurements of the current states and manipulated variables, the prediction model is used to predict the states at the next time period t_{k+1} and the OCP for this period is solved between t_k and t_{k+1}. Using this strategy the delays generated due to the computational time are avoided, but there can still be a mismatch so rapid correction using NLP sensitivity has to be done. Note that at each sampling instant the problem is very similar to the previous one, the only parameters that change are the initial condition of the states (x_0) and the value of the disturbances (w).

Within the NMPC framework two new classes are introduce for defining the initial condition and the disturbance classes. if the parameters (p) of an optimization problem are associated with artificial constraints of the form $x_p - p = 0$ where x_p is a variable used to store the value of the parameters, the NLP sensitivity calculations can be reduced to solve the linear system shown in Equation 1. In this linear equation W is the Hessian matrix of the Lagrange function L, A is the Jacobian matrix of the equality constraints, V is a diagonal matrix which contains the dual variables ν, Z is a diagonal matrix which contains all the variables of the problem and λ are the Lagrange multipliers for the equality constraints. Note that this equation is valid when the the problem is reformulated for solving it using an interior point strategy (IPOPT) and also that the LHS matrix corresponds to the KKT matrix at the optimal solution which is available after solving the nominal optimization problem. Additionally the artificial constraints and variables can be automatically defined within the NMPC framework when the classes *InitialCondition* and

Disturbance are used to declare these elements of the model.

$$\begin{bmatrix} W & \nabla_{zx_p}\mathcal{L}(z,x_p,\lambda,\nu) & A & -I & 0 \\ \nabla_{x_pz}\mathcal{L}(z,x_p,\lambda,\nu) & \nabla_{x_px_p}\mathcal{L}(z,x_p,\lambda,\nu) & \nabla_{x_p}c(z,x_p) & 0 & I \\ A^T & \nabla_{x_p}c(z,x_p)^T & 0 & 0 & 0 \\ V & 0 & 0 & Z & 0 \\ 0 & I & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta x_p \\ \Delta\lambda \\ \Delta\nu \\ \Delta\bar{\lambda} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & \Delta p \end{bmatrix} \tag{1}$$

Injecting the updated manipulated variables using NLP sensitivity to the process may cause the active set of constraints to change, which means that inactive inequality constraints are violated in practice and active inequality constraints may become inactive. These changes of active sets, affect the accuracy and stability of the controller if not handled properly. This framework using clipping in the first interval (CFI) which basically updates the value of the manipulated variables with only a fraction of the sensitivity step, so the active constraints do not change.

This framework uses NLP/QP strategies for solving the optimization problem and for large scale problems these alternatives are particularly efficient. In other words, the differential equations and all the variables have to be fully discretized so the problem can be formulated as a large-scale NLP. However, the discretization can be done automatically using the DAE module from Pyomo and it also makes the problem compatible for using NLP sensitivity.

3.2 Framework Overview

When using this framework within dynamic models it is necessary for the user to call new classes that are not standard from Pyomo and that are developed exclusively to make the implementation of iNMPC/asNMPC controllers easy and intuitive. The classes ManipulatedVar, Disturbance, and InitialCondition are created especially for this framework and the user must use them the same way as other Pyomo objects to define the manipulated variables, the disturbances and the initial conditions of a differential variable. The class DerivativeVar is inherited from the DAE module and it used to define the derivative variables of the model. The ManipulatedVar class inherits the arguments and key words from the Pyomo Var class. The Disturbance and InitialCondition class inherit the key words from the Pyomo Param class with the exception that both of them only receives one argument, for Disturbance is a continuous set that represents the time of the model and for InitialCondition is the corresponding differential variable.

After defining the model using these new classes the user can create an object of the class iNMPC or asNMPC and then call the method *solve* with the adequate arguments. These two classes have the core functionalities of the framework and automate the whole implementation of NMPC controllers. The asNMPC class has additional methods which includes: a method for adding suffixes to the model that avoids the elimination of the artificial constraints by preprocessing instructions and allows for the storage of the sensitivity results; a method for getting and saving the KKT matrix at the optimal solution; a method for doing NLP sensitivity calculations solving the linear problem defined in Eq (1) and a method for doing CFI after the sensitivity calculations so active set changes can be handled.

asNMPC class receives the inputs from a generic function *Plant.* This function must return the states of the process at the current time and it has a flexible structure, so it could be a mathematical model, a call to another software or real measurements of a process. In addition, this framework does not handle the state estimation problem but the user can define a state estimator within the *Plant* function to address this issue.

Finally there is another class called *nl_interface* that interacts with the iNMPC and asNMPC classes and it is used to generate and modify a ".nl" file of the optimization problem. This type of file is used to communicate an AML with a solver and contain the whole representation of the optimization problem including variables, constraints, objectives, suffixes, expression trees and Jacobians (in code). Each ".nl" begins with a header that gives the problem statistics such as: number of continuous variables, number of discrete variables, number of constraints, number of objectives, number of logical constraints and nonzero elements in the Jacobian and gradients.

This "nl_interface" class is necessary in the implementation of NMPC controllers because Pyomo is not efficient writing this type of file for large-scale problems. Using this class allows for this delay to be avoided.

4 Examples and Discussion

We will now look at an example.

4.1 Batch reactor for penicillin production

This example is the temperature control of a batch reactor for the production of penicillin. The objective is to maximize the penicillin concentration (P) at the final time, thus the objective function of the NMPC is not a standard quadratic function of tracking but a final cost function. The OCP for this problem is shown in Equation (2).

$$\min \quad -P(t_k + \tau N_p) \tag{2a}$$

$$\text{s.t.} \quad \frac{dX}{dt} = b_1 X - \frac{b_1}{b_2} X^2, \qquad X(t_k) = X_0, \tag{2b}$$

$$\frac{dP}{dt} = b_3 X, \qquad P(t_k) = P_0, \tag{2c}$$

$$b_1 = 13.1\left[\frac{1 - 0.005(\theta - 30)^2}{1 - 0.005(25 - 30)^2}\right], \tag{2d}$$

$$b_2 = 0.94\left[\frac{1 - 0.005(\theta - 30)^2}{1 - 0.005(25 - 30)^2}\right], \tag{2e}$$

$$b_3 = 1.71\left[\frac{1 - 0.005(\theta - 30)^2}{1 - 0.005(25 - 30)^2}\right], \tag{2f}$$

$$20^\circ C \leq \theta \leq 30^\circ C \tag{2g}$$

All the variables are dimensionless except for the temperature (θ) which is in degrees Celsius. The differential equations come from mass balances that represent the change of the penicillin concentration (P) and the biomass concentration (X). The RHS are dimensionless reaction rate expressions. The manipulated variable temperature which has an upper and lower bound as shown in Equation (2g). The initial biomass concentration is 0.02, the initial concentration is 0.0, and the initial temperature is 27°C.

For this example we use a sampling time of 0.02, a control horizon of 25 and prediction horizon of 50. The differential equations are discretized using orthogonal collocation in finite elements with three Radau roots, meaning they are transformed into algebraic constraints. This is the default discretization for this framework. After discretization, the OCP problem has 755 variables and 730 equality constraints for a total number of 25 degrees of freedom which correspond to the reactor temperature in each sampling instant.

We will consider the asNMPC and introduce a model-plant mismatch using the same model but changing the proportional constant of the expressions to be 15.1,0.74, and 1.91 respectively.

```
from pyomo.environ import *
from pyomo.dae import *
from pyomo.asNMPC import *

#Model
m = ConcreteModel()
#Sets
m.t = ContinuousSet(bounds =(0,1.0))
#Variables
m.x = Var(m.t,bounds=(None,None),initialize=0.0)
m.y = Var(m.t,bounds=(None,None),initialize=0.0)
m.vx = ManipulatedVar(m.t,bounds=(None,None),initialize=0.02)
m.vy = ManipulatedVar(m.t,bounds=(None,None),initialize=0.02)

##NMPC
CONTROLLER = AdvancedStepNMPC(model=m, model_time=m.t,
                              control_horizon=25,
                              simulation_periods=50,
                              default_discretization=True)
CONTROLLER.solve('ipopt',
                              options=[('linear_solver', 'ma57'),
                              ('mu_init',1E-4)],
                              file_name='Penicillin_asNMPC',
                              real_plant = plant,
                              dynamic_plot=True)
```