Lecture notes for the course IIA 4117: Model Predictive Control

Roshan Sharma

October 2019

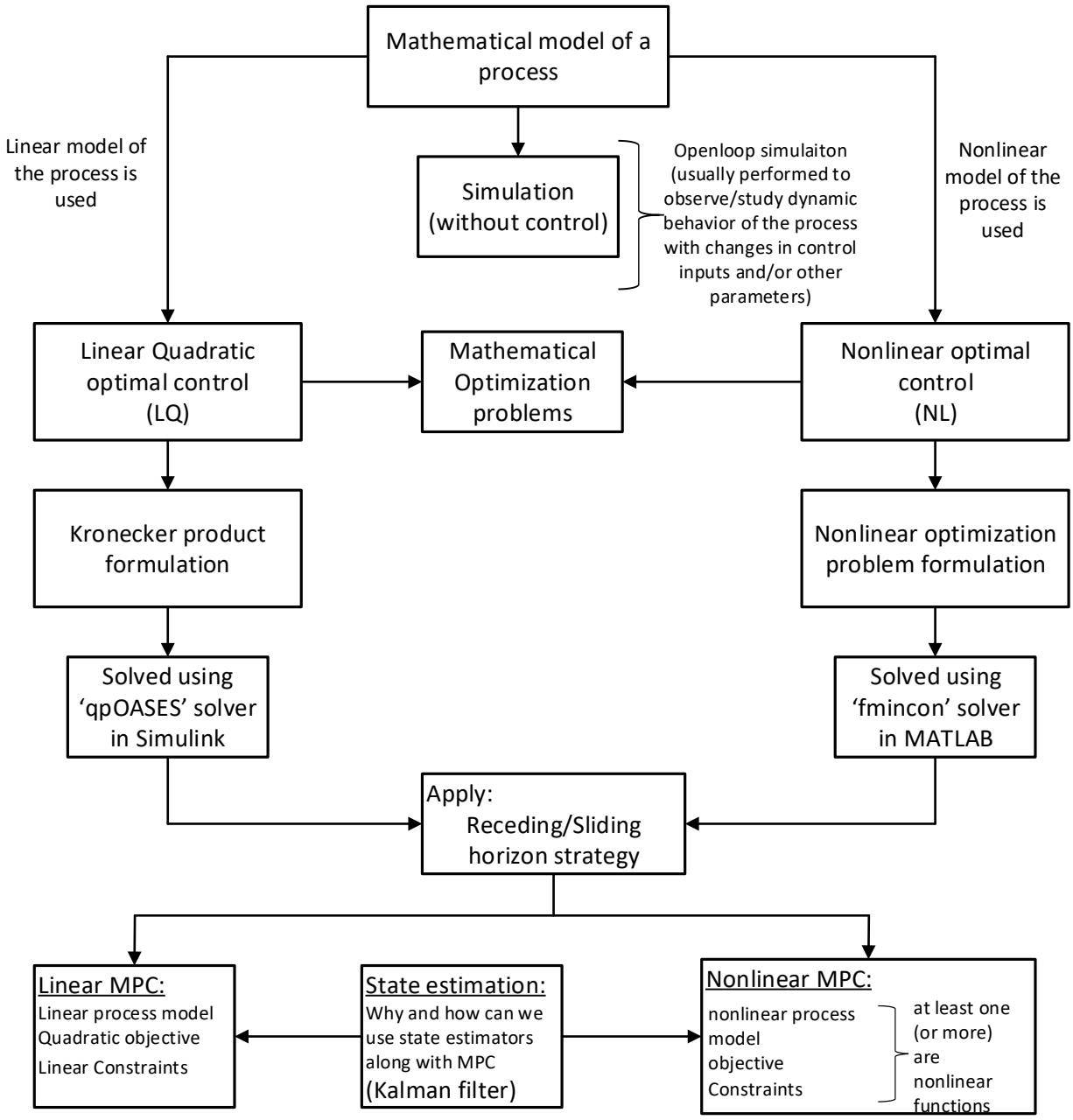
Foreword

This lecture note is intended for the master students for the course IIA4117, Model Predictive Control at the University College of Southeast Norway. I have tried to explain the main concepts in a simple way so that the students can follow them well. The lecture note is far from being complete and will be continuously updated throughout the semester. It is therefore highly recommended that students attend the lectures regularly to follow up the updates. It may contain errors and typos. If the students find any, it would be useful if you can point it out during the lecture or let me know by email.

Roshan Sharma

Associate Professor, Department of Electrical Engineering, IT and Cybernetics
University of South-Eastern Norway, Porsgrunn
Email: roshan.sharma@usn.no

Tree diagram of the main ingredients of the course



Project:

Linear MPC:
Quadratic objective
Linear Constraints
} Development and implementation of linear MPC to real process

Lecture 1

1.1 Fore Word:

If you split the term "Model based predictive control" into its meaningful parts, we obtain the following two distinctive meanings.

(i) Model based:

As the name implies, a model of the process is needed. The model of the process can be developed and represented in various forms suitable for control.

(ii) Predictive control:

Prediction of the future values of the process outputs and the states from the current time is performed. For the prediction, of course, the real plant/process cannot be made to operate in the future time steps from the current time, but the model of the process being controlled can be easily simulated to obtain the process outputs and the states for the future time steps.

Q) What to do with these future time step values?

The process outputs and the states predicted for the future time steps by using the process model are used to formulate an optimization problem which is an optimal control problem. The nature of the optimization problem can vary from tracking a set point to more complex economic objectives. This optimization problem is then solved to obtain the optimal values of the control actions. We can use these control actions to achieve the desired control objective.

With MPC, the procedure of looking into the future using the process model followed by formulation and solution of an optimization problem is repeated again and again for each time step. This creates a feedback action and is commonly known as a sliding horizon strategy.

In this course, we will talk about predictive control in detail throughout the semester. But at first, let us look at the different kinds/types of models that are more often used for predictive control.

1.2 Model Types:

The algorithm for MPC is generally implemented in digital devices like computers, microcontrollers and other forms of microprocessors. This ultimately leads us to the fact that the model of the process (being controlled) should also be in discrete form. In general, discrete time models (both linear and nonlinear) are expressed in many different ways.

For this course, we consider only a few of them. We do not go into details about the development of these models using for e.g. conservation laws like mass, momentum, energy, specie balances etc. (as this is not within the focus of the course) but simply see the possibilities of using these models for designing a model predictive controller. Thus, we begin with the assumption that the model of the process is already available to us.

A brief description of some of the model types that are well suited for MPC are given below.

a) Non-linear discrete time state space models:

$$x_{k+1} = f(t_k, x_k, u_k) \rightarrow \text{State Equation}$$
$$y_k = g(t_k, x_k, u_k) \quad \rightarrow \text{Measurment Equation}$$

Here, $x \epsilon \mathbb{R}^{n_x} = \left[x_1, x_2, x_3, \dots \dots \dots \dots \dots x_{n_x}\right]^T$ are the states of the system, $u \epsilon \mathbb{R}^{n_u} = \left[u_1, u_2, u_3, \dots \dots \dots \dots \dots u_{n_u}\right]^T$ are the control inputs and $y \epsilon \mathbb{R}^{n_y} = \left[y_1, y_2, y_3, \dots \dots \dots \dots \dots y_{n_y}\right]^T$ are the outputs of the system. The term n_x represents the number of states, n_u the number of control inputs and n_y the number of measurements of the system. For single input single output system (SISO), $n_u = 1$ and $n_y = 1$. For MIMO[1] (multi input multi output), $n_u > 1$ and $n_y > 1$. One of the biggest advantage of using MPC is that it can be used for a MIMO system effectively.

$f = \left[f_1, f_2, f_3, \dots \dots \dots \dots \dots f_{n_x}\right]^T$ are the nonlinear state equations and $g = [g_1, g_2, g_3, \dots \dots \dots \dots \dots g_n]^T$ are the nonlinear measurement equations expressed as vectors. The state equations and the measurement equations generally make up the mathematical model of the process.

b) Linear discrete time state space model:

The linear discrete time state space model is written as,

$$x_{k+1} = A_d x_k + B_d u_k + v_k \rightarrow \text{State Equation}$$
$$y_k = C_d x_k + D_d u_k + w_k \quad \rightarrow \text{Measurment Equation}$$

Here, A_d, B_d, C_d and D_d are the system matrices of appropriate sizes with the subscript d meaning discrete. $A_d \epsilon \mathbb{R}^{n_x \times n_x}$, $B_d \epsilon \mathbb{R}^{n_x \times n_u}$, $C_d \epsilon \mathbb{R}^{n_y \times n_x}$ and $D_d \epsilon \mathbb{R}^{n_y \times n_u}$. Here, $v_k \epsilon \mathbb{R}^{n_x}$ and $w_k \epsilon \mathbb{R}^{n_y}$ are random variables with zero mean and certain variance. v_k is the process noise and w_k is the measurement noise. In addition, it is assumed that v_k and w_k are uncorrelated (stochastically independent) i.e. $corr[v(k), w(j) = 0 \text{ for all } k \text{ and } j]$. In other words, this implies that the random disturbances affecting the measurements have nothing to do with the randomness in the states (or the process) itself.

For designing a linear MPC, the linear discrete time state space model is used. This is the model type that has the main focus in this course.

c) Input-output models/polynomial models.

Input-output models or the polynomial models are mathematical models for the outputs of a process. As the name implies, with the polynomial models, the current output is dependent on the values of the past outputs and/or the past inputs of the plant/process. The notion or concept of states of a process are not valid or used with the input-output models. They can further be categorized as:

[1] A single PID controller can control only one output by using one control input at a time. A single MPC can be used to control multiple outputs by using multiple inputs at a time.

(i) Non-linear input/output models:

The current output is some nonlinear combination of the past outputs and/or past inputs of the process as,

$$y_k = h\left(y_{k-1}, y_{k-2}, \dots\dots\dots\dots\dots\dots y_{k-m_y},\ u_{k-1}, u_{k-2}, \dots\dots\dots\dots\dots\dots u_{k-m_u}\right)$$

(ii) Linear input/output models:

The current output is a linear combination of the past outputs and/or past inputs of the process. An example is the ARX (Auto Regressive Exogenous) model which is written as,

$$y_k = -A_1 y_{k-1} - A_2 y_{k-2} - \dots\dots\dots - A_{m_y} y_{k-m_y} + B_1 u_{k-1} + \dots\dots\dots + B_{m_u} u_{k-m_u}$$

Here, m_y is the number past outputs and m_u is the number of past inputs. $A_1, A_2, \cdots, A_{m_y}$ and $B_1, B_2, \cdots, B_{m_u}$ are the coefficients of the past outputs and inputs respectively.

We can further express it in a compact way using a delay operator q such that

$q^{-1}y(k) = y_{k-1}$

$q^{-2}y(k) = y_{k-2}$

$\vdots$

and so on.

Similarly, $q^{-1}u(k) = u_{k-1}$, $q^{-2}u(k) = u_{k-2}$ and so on. Then we have,

$$y_k + A_1 y_{k-1} + A_2 y_{k-2} + \dots\dots\dots + A_{m_y} y_{k-m_y} = B_1 u_{k-1} + \dots\dots\dots + B_{m_u} u_{k-m_u}$$

$$y_k + A_1 q^{-1} y(k) + A_2 q^{-2} y(k) + \dots + A_{m_y} q^{-m_y} y(k) = B_1 q^{-1} u(k) + \dots + B_{m_u} q^{-m_u} u(k)$$

$$A(q)y(k) = B(q)u(k)$$

where,

$$A(q) = 1 + A_1 q^{-1} + A_2 q^{-2} + \cdots + A_{m_y} q^{-m_y}$$

$$B(q) = B_1 q^{-1} + B_2 q^{-2} + \cdots + B_{m_u} q^{-m_u}$$

d) Continuous time models

In general, if the model of the process is developed with the first principles modeling (using conservation laws like mass balance, momentum balance and energy balance), then in most of the cases, a non-linear continuous[2] time state space model is obtained as,

$$\frac{dx}{dt} = \dot{x}(t) = f(x, u, t) \rightarrow \text{State equations} \quad (1.1)$$

$$y = g\ (x, u, t)\ \rightarrow \text{Measurement equations} \quad (1.2)$$

Here, $x \epsilon \mathbb{R}^{n_x} = \left[x_1, x_2, x_3, \dots\dots\dots x_{n_x}\right]^T$ are the states, $u \epsilon \mathbb{R}^{n_u} = \left[u_1, u_2, u_3, \dots\dots\dots\dots\dots u_{n_u}\right]^T$ are the control inputs and $y \epsilon \mathbb{R}^{n_y} = \left[y_1, y_2, y_3, \dots\dots\dots\dots\dots y_{n_y}\right]^T$ are the outputs of the process.

[2] If the model of the process is in the continuous time domain, it should be discretized. One easy way of doing it in Simulink is to use the integrator block. In MATLAB the ode solvers can be used or you can even make one of your own integrator based on Euler backward/forward, Runge-Kutta 4 algorithm etc.

$f = [f_1, f_2, f_3 \dots \dots \dots f_{n_x}]^T$ is the vector with state equations and $g = [g_1, g_2, \dots \dots \dots \dots \dots g_{n_y}]^T$ is the vector with the measurement equations.

The continuous time linear state space model is of the form,

$$\frac{dx(t)}{dt} = \dot{x}(t) = A_c x(t) + B_c u(t) \rightarrow \text{State equations} \quad (1.3)$$

$$y(t) = C_c x(t) + D_c u(t) \rightarrow \text{Measurement equations} \quad (1.4)$$

Here, For a large part of this course, we will be using mechanistic models that is developed from the first principles modeling technique. For designing linear MPC, the nonlinear mechanistic model will be linearized to obtain a continuous time linear state space model. The continuous time linear state space model will then be discretized to obtain the discrete time linear state space model. Finally, model predictive control will be designed using the discrete time linear state space model.

Example:

e) Transfer function models

In addition, we can also have process models as transfer functions,

$$y(s) = H(s)u(s) \quad (1.5)$$

H(s) is the transfer function of the process with s as the Laplace operator. The model of the process given by (1.5) is a continuous time model in Laplace domain.

It is relatively easier and straightforward to handle a transfer function model. In MATLAB, the Control Toolbox offers a wide range of functions to handle the transfer function model, which is a linear model.

Example:

Let us consider the transfer function model of a distillation column as,

$$y = \begin{bmatrix} \frac{87.8}{194s+1}, & \frac{-(5227s+432)}{14550s^2+1045s+5} \\ \frac{108.2}{194s+1}, & \frac{-(9473S+548)}{14550s^2+1045s+5} \end{bmatrix} u \quad (1.6)$$

The system taken into consideration has two inputs and two outputs. The MATLAB function *'tf'* allows us to form the transfer function of the model. There are other functions such as *'step', 'impulse', 'ss', 'c2d', 'ssdata'* etc. in MATLAB. These can be used to handle and perform operations with the transfer function model.

Illustration:

In MATLAB command window, the following commands can be entered to generate a transfer function model.

```
>> num = {87.8, -[5227, 432]; 108.2, -[9473, 548]};
>> den = {[194, 1], [14550, 1045,5]; [194, 1], [14550, 1045, 5]};
>> dist = tf(num,den,'inputn',{'u1','u2'},'outputn',{'y1','y2'})

dist =

 From input "u1" to output...
       87.8
  y1:  ---------
     194 s + 1

       108.2
  y2:  ---------
     194 s + 1

 From input "u2" to output...
        -5227 s - 432
  y1:  ----------------------
     14550 s^2 + 1045 s + 5

        -9473 s - 548
  y2:  ----------------------
     14550 s^2 + 1045 s + 5

Continuous-time transfer function.
```

The step response of the transfer function model can be obtained as,

```
>> step(dist)
```

The step response plot is shown in Figure 1.

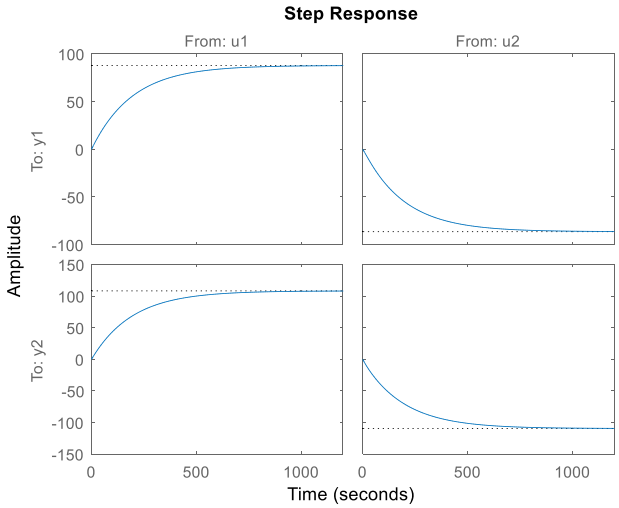


Figure 1: Step response plot of the transfer function model

From Figure 1, it can be understood that when there is a step change in the control input $u1$ (keeping $u2$ unchanged), both outputs $y1$ and $y2$ are increased. In contrast, when there is a step change in the control input $u1$ (keeping $u2$ unchanged), both outputs $y1$ and $y2$ are decreased.
The equivalent continuous time linear state space model of the transfer function model can be obtained as,

```
>> SSdist = ss(dist)

SSdist =

 A =
        x1       x2       x3
  x1  -0.005155        0        0
  x2         0   -0.07182   -0.02199
  x3         0    0.01563         0

 B =
     u1  u2
  x1  1   0
  x2  0   2
  x3  0   0

 C =
        x1      x2      x3
  y1   0.4526  -0.1796  -0.9501
  y2   0.5577  -0.3255   -1.205

 D =
     u1  u2
  y1  0   0
  y2  0   0

Continuous-time state-space model.
```

Note: The students should try to use these functions to observe the step response, impulse response and to convert continuous time model to discrete time model. Please see the MATLAB documentation regarding the syntax for these functions. In addition, there are plenty of tutorials and examples from Mathworks that the students can follow.

1.3 Conversion between Linear Model Representations:

For simpler cases or models, it is also possible to convert from one model representation to another model representation analytically. Few examples are shown below:
Consider the linear time invariant state space model,

$$x_{k+1} = \frac{1}{2}x_k + u_k \tag{1.7}$$

$$y_k = 2x_k \tag{1.8}$$

Let us consider a time shift operator q such that $qx := x_{k+1}$, $q^{-1}x = x_{k-1}$ and so on.
The equivalent ARX model of (1.7 and 1.8) can be written as,

$$qx = \frac{1}{2}x + u$$

$$x\left(q - \frac{1}{2}\right) = u$$

$$x = \frac{1}{q - \frac{1}{2}}u$$

$$\text{Then,} \quad y = 2x = \frac{2}{q - \frac{1}{2}}u \tag{1.9}$$

Equation (1.9) is an ARX model which can be easily converted to input-output model as,

$$\left(q - \frac{1}{2}\right)y = 2u$$

$$qy - \frac{1}{2}y = 2u$$

$$y_{k+1} - \frac{1}{2}y_k = 2u_k$$

$$y_{k+1} = \frac{1}{2}y_k + 2u_k$$

Similarly, an infinite impulse response model can also be developed from state-space model.
From (1.7) we have,

$x_k = \frac{1}{2}x_{k-1} + u_{k-1}$

$x_k = \frac{1}{2}\left(\frac{1}{2}x_{k-2} + u_{k-2}\right) + u_{k-1}$

$x_k = \left(\frac{1}{2}\right)^2 x_{k-2} + \frac{1}{2}u_{k-2} + u_{k-1}$

$x_k = \left(\frac{1}{2}\right)^2 \left(\frac{1}{2}x_{k-3} + u_{k-3}\right) + \frac{1}{2}u_{k-2} + u_{k-1} = \left(\frac{1}{2}\right)^3 x_{k-3} + \left(\frac{1}{2}\right)^2 u_{k-3} + \frac{1}{2}u_{k-2} + u_{k-1}$

$\vdots$

$\vdots$

$x_k = \left(\frac{1}{2}\right)^k x_0 + \sum_{i=0}^{k-1}\left(\frac{1}{2}\right)^{k-1-i} u_i$

When $k \to \infty$, $x_k \approx \sum_{i=0}^{k-1}\left(\frac{1}{2}\right)^{k-1-i} u_i$ and we get the infinite impulse response model.
From the output equation of (1.8) we have,

$y_k = 2x_k$

$y_k \approx 2\sum_{i=0}^{k-1}\left(\frac{1}{2}\right)^{k-1-i} u_i$

$$y_k \approx 2u_{k-1} + 2\left(\frac{1}{2}\right)u_{k-2} + 2\left(\frac{1}{2}\right)^2 u_{k-3} + \cdots + 2\left(\frac{1}{2}\right)^{k-1} u_0 \quad (1.10)$$

1.4 Linearization of continuous time nonlinear model

For most of the real world processes, the models are dynamic, continuous time and nonlinear. Let us first focus our attention to linear models. A nonlinear model can be linearized to obtain a linear model around an equilibrium point.

Let the nonlinear model of the process be written as,

$$\frac{dx(t)}{dt} = \dot{x}(t) = f\big(x(t), u(t)\big) \quad (1.11)$$

$$y(t) = g\big(x(t), u(t)\big) \quad (1.12)$$

Here, x are the states of the process, u are the control inputs and y are the outputs such that,

$x \epsilon \mathbb{R}^{n_x} = \big[x_1, x_2, \ldots\ldots\ldots\ldots\ldots x_{n_x}\big]^T$

$u \epsilon \mathbb{R}^{n_u} = \big[u_1, u_2, \ldots\ldots\ldots\ldots\ldots u_{n_u}\big]^T$

$y \epsilon \mathbb{R}^{n_y} = \big[y_1, y_2, \ldots\ldots\ldots\ldots\ldots y_{n_y}\big]^T$

$f = \big[f_1, f_2, \ldots\ldots\ldots\ldots\ldots f_{n_x}\big]^T$ are the state equations and $g = \big[g_1, g_2, \ldots\ldots\ldots\ldots\ldots g_{n_y}\big]^T$ are the measurement equations.

We linearize the model around an assumed/given/known equilibrium point or operating point (x_{op}, u_{op}, y_{op}). Here, x_{op} is the values of the states at the operating point, u_{op} is the corresponding control input values and y_{op} is the corresponding value of the outputs. Let us assume that the actual system dynamics are in the proximity of the nominal trajectories i.e. around the operating point.

i.e. $x(t) = x_{op} + \delta x(t)$, $u(t) = u_{op} + \delta u(t)$ and $y(t) = y_{op} + \delta y(t)$

Here, $\delta x(t), \delta u(t)$ and $\delta y(t)$ are small perturbations or changes.

Then, we have $\big(x(t) - x_{op}\big) = \delta x(t)$ and $\big(u(t) - u_{op}\big) = \delta u(t)$. The state equation is,

$$\frac{d}{dt}x(t) = \dot{x}(t) = f\big(x(t), u(t)\big) \quad (1.13)$$

Taylor series expansion of equation (1.13) about the operating point is,

$$\frac{d}{dt}x(t) = f\big(x_{op}, u_{op}\big) + \left.\frac{\partial f}{\partial x}\right|_{x_{op}, u_{op}} \big(x(t) - x_{op}\big) + \left.\frac{\partial f}{\partial u}\right|_{x_{op}, u_{op}} \big(u(t) - u_{op}\big) + HOT^3$$

$$\frac{d}{dt}x_{op} + \frac{d}{dt}\delta x(t) = f\big(x_{op}, u_{op}\big) + \left.\frac{\partial f}{\partial x}\right|_{x_{op}, u_{op}} \delta x(t) + \left.\frac{\partial f}{\partial u}\right|_{x_{op}, u_{op}} \delta u(t) + HOT$$

[3] HOT = Higher Order Terms

Here, $\frac{d}{dt}x_{op} = f(x_{op}, u_{op})$ i.e. functional values evaluated at the operating points. Neglecting the higher order terms (HOTs),

$$\frac{d}{dt}\delta x(t) = \left.\frac{\partial f}{\partial x}\right|_{x_{op},u_{op}} \delta x(t) + \left.\frac{\partial f}{\partial u}\right|_{x_{op},u_{op}} \delta u(t)$$

$$\delta\dot{x}(t) = A_c\delta x(t) + B_c\delta u(t)$$

where,

$$A_c = \left.\frac{\partial f}{\partial x}\right|_{x_{op},u_{op}} = A_c^{n_x\times n_x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}, \frac{\partial f_1}{\partial x_2}, & \dots\dots\dots & \frac{\partial f_1}{\partial x_{n_x}} \\ \frac{\partial f_2}{\partial x_1}, \frac{\partial f_2}{\partial x_2}, & \dots\dots\dots & \frac{\partial f_2}{\partial x_{n_x}} \\ & \vdots & \\ \frac{\partial f_{n_x}}{\partial x_1}, \frac{\partial f_{n_x}}{\partial x_2}, & \dots\dots\dots & \frac{\partial f_{n_x}}{\partial x_{n_x}} \end{bmatrix}_{|x_{op},u_{op}}$$

$$B_c = \left.\frac{\partial f}{\partial u}\right|_{x_{op},u_{op}} = B_c^{n_x\times n_u} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1}, \frac{\partial f_1}{\partial u_2}, & \dots\dots\dots & \frac{\partial f_1}{\partial u_{n_u}} \\ \frac{\partial f_2}{\partial u_1}, \frac{\partial f_2}{\partial u_2}, & \dots\dots\dots & \frac{\partial f_2}{\partial u_{n_u}} \\ & \vdots & \\ \frac{\partial f_{n_x}}{\partial u_1}, \frac{\partial f_{n_x}}{\partial u_2}, & \dots\dots\dots & \frac{\partial f_{n_x}}{\partial u_{n_u}} \end{bmatrix}_{|x_{op},u_{op}}$$

For the output equation:

Let us perform a Taylor series expansion of the output equation (1.12) about the operating points (x_{op}, u_{op}, y_{op}) such that the output dynamics is in close proximity of nominal trajectories, i.e. $y(t) = y_{op} + \delta y(t)$ and $y_{op} = g(x_{op}, u_{op})$.

We get,

$$y_{op} + \delta y(t) = g(x_{op}, u_{op}) + \left.\frac{\partial g}{\partial x}\right|_{x_{op},u_{op}} (x(t) - x_{op}) + \left.\frac{\partial g}{\partial u}\right|_{x_{op},u_{op}} (u(t) - u_{op}) + HOT$$

Neglecting HOT,

$$\delta y(t) = \left.\frac{\partial g}{\partial x}\right|_{x_{op},u_{op}} \delta x + \left.\frac{\partial g}{\partial u}\right|_{x_{op},u_{op}} \delta u$$

So,

$$\delta y(t) = C_c\delta x(t) + D_c\delta u(t)$$

where,

$$C_c = \frac{\partial g}{\partial x}\Bigg|_{x_{op},u_{op}} = C_c^{n_y \times n_x} = \begin{bmatrix} \frac{\partial g_1}{\partial x_1}, \frac{\partial g_1}{\partial x_2}, & \dots\dots\dots \frac{\partial g_1}{\partial x_{n_x}} \\ \frac{\partial g_2}{\partial x_1}, \frac{\partial g_2}{\partial x_2}, & \dots\dots\dots \frac{\partial g_2}{\partial x_{n_x}} \\ \vdots & \\ \frac{\partial g_{n_y}}{\partial x_1}, \frac{\partial g_{n_y}}{\partial x_2}, & \dots\dots\dots \frac{\partial g_{n_y}}{\partial x_{n_x}} \end{bmatrix}_{x_{op},u_{op}}$$

$$D_c = \frac{\partial g}{\partial u}\Bigg|_{x_{op},u_{op}} = D_c^{n_y \times n_u} = \begin{bmatrix} \frac{\partial g_1}{\partial u_1}, \frac{\partial g_1}{\partial u_2}, & \dots\dots\dots \frac{\partial g_1}{\partial u_{n_u}} \\ \frac{\partial g_2}{\partial u_1}, \frac{\partial g_2}{\partial u_2}, & \dots\dots\dots \frac{\partial g_2}{\partial u_{n_u}} \\ \vdots & \\ \frac{\partial g_{n_y}}{\partial u_1}, \frac{\partial g_{n_y}}{\partial u_2}, & \dots\dots\dots \frac{\partial g_{n_y}}{\partial u_{n_u}} \end{bmatrix}_{x_{op},u_{op}}$$

The linear model finally can be written as,

$$\delta\dot{x}(t) = A_c\delta x(t) + B_c\delta u(t) \quad (1.14)$$

$$\delta y(t) = C_c\delta x(t) + D_c\delta u(t) \quad (1.15)$$

It is very important to understand that the linear model given by Equation (1.14) and (1.15) is in deviation variable form i.e. deviation from the operating point. When you simulate this linear model, you will get the solution in the deviation variables $\delta x, \delta u$ and δy. To calculate the actual value of the variables, it is necessary to add the operating point values to the deviation variables i.e.

$$x(t) = \delta x + x_{op}$$

$$y(t) = \delta y + y_{op}$$

$$u(t) = \delta u + u_{op}$$

In special cases, if the operating point is at the origin i.e. $x_{op} = 0, u_{op} = 0$ & $y_{op} = 0$, then $\delta x = x(t), \delta y = y(t)$ & $\delta u = u(t)$. For such cases, the linear model in continuous time domain can be written as,

$$\frac{dx(t)}{dt} = \dot{x}(t) = A_c x(t) + B_c u(t) \quad (1.16)$$

$$y(t) = C_c x(t) + D_c u(t) \quad (1.17)$$

The Jacobians $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial u}, \frac{\partial g}{\partial x}, \frac{\partial g}{\partial u}$ can be calculated analytically for simple models. If analytical calculation becomes difficult, numerical approximations can be used (e.g., finite difference approximation).

Various operations can be performed to the linear model of by using the control system toolbox in MATLAB. As an example, the deviation form of the continuous linear model given by Equations (1.14) and (1.15) can be changed to a discrete time linear state space model in deviation form as,

$$\delta x_{k+1} = A_d\delta x_k + B_d\delta u_k \quad (1.18)$$

$$\delta y_k = C_d \delta x_k + D_d \delta u_k \tag{1.19}$$

Here, $\delta x_k = (x_k - x_{op}), \delta u_k = (u_k - u_{op})$ and $\delta y_k = (y_k - y_{op})$ are the deviation of the discrete time states, control inputs and outputs from the operating point respectively. If the values of the operating points are all at the origin i.e. if $x_{op} = 0, u_{op} = 0$ & $y_{op} = 0$, then $\delta x_k = x_k, \delta y_k = y_k$ & $\delta u_k = u_k$. For such special case, the linear model in discrete time domain can be written as,

$$x_{k+1} = A_d x_k + B_d u_k \tag{1.20}$$
$$y_k = C_d x_k + D_d u_k \tag{1.21}$$

Here A_d, B_d, C_d and D_d are the system matrices in discrete time domain. The lower superscript d denotes discrete time. These discrete time system matrices can be found by converting the continuous time domain system matrices A_c, B_c, C_c and D_c. For this, the MATLAB function *"c2dm"* can be used as shown below.

[A_d, B_d, C_d, D_d] = *c2dm*(A_c, B_c, C_c , D_c , T_s)

with T_s being the sampling time.

1.5 Introduction to simulation of process models

One basic requirement to be able to use the model of the process for predictive control is to be able to first simulate the model properly. Normally, the process model that we will encounter frequently is a set of ordinary differential equations (ODEs) in addition to some algebraic equations. To find the solution of ODEs, we need to integrate them i.e. $x(t) = \int \dot{x}(t)\, dt$. This means that the solution $x(t)$ is the integration of the differential equation $\dot{x}(t)$. A very basic introduction to some of the commonly used method for solving a set of ODEs is given below.

Let us consider a general continuous non-linear process model described by a set of ordinary differential equations as,

$$\frac{dx(t)}{dt} = \dot{x}(t) = f(x, t)$$

with x_0, the initial values of the states as being known or given. Remember that to solve an ODE, the initial value should be known.

We are interested in finding the solution $x(t)$ at discrete time steps for a given time interval when the initial values are known. Many methods are available to find its solution. Some of the numerical methods that can be used are given below. These methods for solving ODEs can be easily implemented as MATLAB scripts. They are fixed time step ODE solvers where the time step Δt is fixed.

a) **Forward Euler Method**

$\dot{x}(t) \approx \frac{x_{k+1} - x_k}{\Delta t}$ with Δt being the time step size.

$x_{k+1} = x_k + \Delta t\, \dot{x}(t)$

$x_{k+1} = x_k + \Delta t\, f(x_k, t_k)$ for $k = 0,1,2, \ldots \ldots$ and x_0 known

b) **Backward Euler:**

$\dot{x}(t) \approx \frac{x_k - x_{k-1}}{\Delta t}$ with Δt being the time step size.

$x_k = x_{k-1} + \Delta t\, f(x_k, t_k)$

$x_{k+1} = x_k + \Delta t\, f(x_{k+1}, t_{k+1})$ for $k = 0,1,2, \ldots \ldots$ and x_0 known

This is an implicit equation and has to be solved iteratively to obtain x_{k+1}. For example, Newtons method can be used to solve the implicit equation.

c) **Trapezoidal Method**

$x_{k+1} = x_k + \frac{1}{2}\Delta t\, [f(x_k, t_k) + f(x_{k+1}, t_{k+1})]$ for $k = 0,1,2, \ldots \ldots$ and x_0 known

This is an implicit equation and has to be solved iteratively to obtain x_{k+1}. For example, Newton method can be used to solve the implicit equation.

d) **Range-Kutta Method (4th order)**

$x_{k+1} = x_k + \Delta t \left(\frac{k_1 + 2k_2 + 2k_3 + k_4}{6}\right)$ for $k = 0,1,2, \ldots \ldots$ and x_0 known

where,

$$k_1 = f(x_k, t_k)$$

$$k_2 = f\left(x_k + k_1 \frac{\Delta t}{2}, t_k + \frac{\Delta t}{2}\right)$$

$$k_3 = f\left(x_k + k_2 \frac{\Delta t}{2}, t_k + \frac{\Delta t}{2}\right)$$

$$k_4 = f(x_k + k_3 \Delta t, t_k + \Delta t)$$

Example

Let us consider an inverted pendulum on a cart as shown in Figure 2. The position of the cart with mass m_2 is $(x_2, y_2 = 0)$, and the position of the pendulum mass m_1 at angle α and pendulum length l is (x_1, y_1). The cart is pushed by force F (control input), while the pendulum is influenced by gravity g and a friction torque $T_f = k_T l^2 |\omega| \omega$, where $\omega = \frac{d\alpha}{dt}$ is the angular velocity in rad/sec.

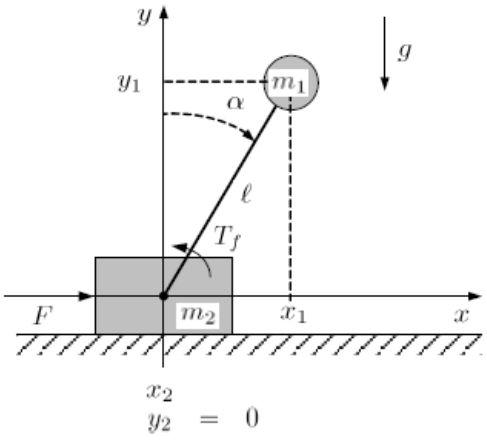


Figure 2: Inverted pendulum on a cart

A mechanistic model of the inverted pendulum system is given below.

$$\frac{d\alpha}{dt} = \omega$$

$$\frac{d\omega}{dt} = \frac{m_1+m_2}{m_1^2 l^2 cos^2\alpha - m_1^2 l^2 - m_1 m_2 l^2}(k_T l^2|\omega|\omega - m_1 g l\, sin\alpha) + \frac{cos\alpha}{m_1 l cos^2\alpha - m_1 l - m_2 l}(F + \omega^2 m_1 l\, sin\alpha)$$

$$\frac{dx_2}{dt} = v_2$$

$$\frac{dv_2}{dt} = \frac{1}{m_1 l\, cos\alpha}\left(m_1 g l\, sin\alpha - k_T l^2|\omega|\omega - m_1 l^2 \frac{d\omega}{dt}\right)$$

Here, α, ω, x_2 and v_2 are the states of the system. The parameters of the system are: $m_1 = 1$ kg, $m_2 = 2$ kg, $l = 1$ m, $k_T = 0.1$ kg/rad^2. Assume that the cart is not being controlled and hence $F = 0$.

In Simulink, to solve or simulate the ODEs that describes the pendulum model, various built-in blocks can be directly utilized. An easy way to do so is to use the user defined *MATLAB-function* block to write the right hand side of the differential equations. Then the integrator block can be used to integrate or solve the ODEs. For details, please look the video(s) in the link below.

https://web01.usn.no/~roshans/mpc/videos/lecture1/openloop-simulation-pendulum.mp4

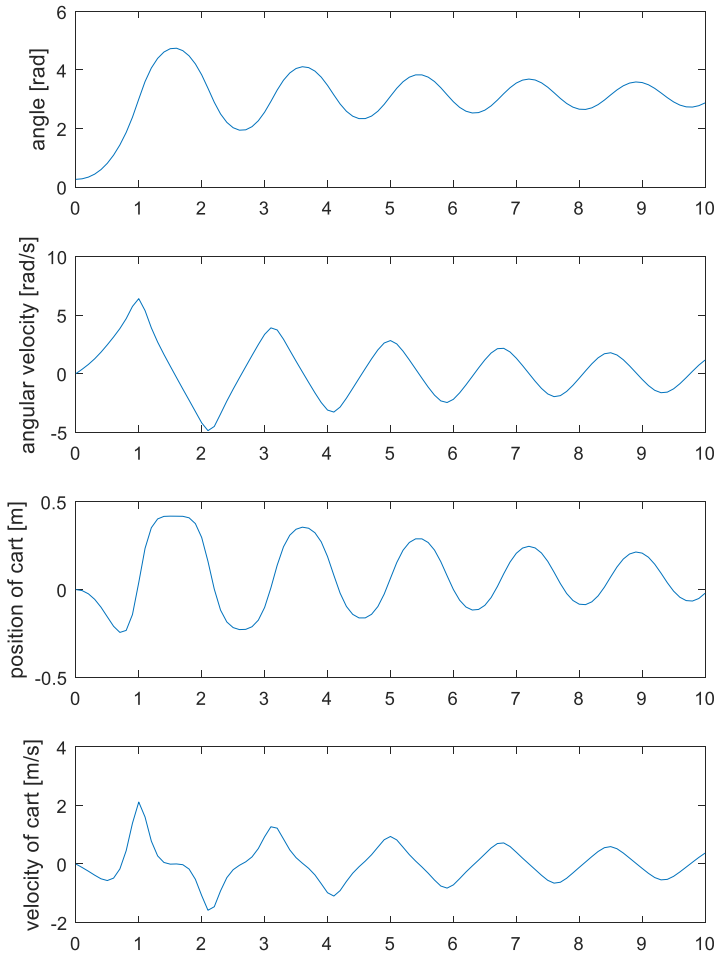


Figure 3: Openloop simulation of inverted pendulum with RK-4 fixed time step solver

Note:

Fixed time step $\Delta t = 0.1$ was used for the RK-4 method. We can also obtain the same results with the forward Euler method but the time step has to be decreased sufficiently i.e. $\Delta t = 0.001$. RK-4 method produces stable results even with a comparatively higher value of the time step as compared to the forward Euler method. A proper care about the choice of the time step value should be taken when forward Euler is used. With $\Delta t = 0.1$, the forward Euler produces incorrect results. You can play around with the Simulink simulator of the inverted pendulum that can be downloaded from the homepage of the course.

Real-time simulation of process models:

Normally, the model of the process is solved faster than the total length of time the model is simulated for. In other words, if you are simulating a process described by a set of ODEs from a starting time of 0 sec to an ending time of 200 sec, Simulink normally does not require 200 sec to solve it. In general, it will be able to solve the model much more faster than 200 sec. The solution is made available (for example for plotting) as soon as the model is solved. This way of simulating the model does not allow you to see the results or solutions in real-time.

In Simulink, it is possible to simulate the model in real time. This allows for the user to observe the solution in real time. In this course, we will use the *real time pacer* library for running the simulations in real time. The *real time pacer* library can be downloaded from the homepage of the course. Please refer to the video(s) (link below) to see and learn how it can be used in Simulink. The video is based on the example of the inverted pendulum model.
https://web01.usn.no/~roshans/mpc/videos/lecture1/realtime-simulation-pendulum.mp4

Important Note: To read data from sensors and to apply control inputs to a real process at specific time intervals (say for example every 0.1 seconds), the real time pacer library is a must and should be used in Simulink.

Built-in ODE solvers in MATLAB:
With MATLAB, we can also use the built-in ODE solvers for solving the process model given by a set of ODES. Various types of ODE solvers are available in MATLAB.
ode45, ode15s, ode23, ode23t etc. are some of the variable step ODE solvers available in MATLAB.
Tips: It is a good idea to start the simulation with *ode45* as a default choice. For stiff systems, *ode15s* is recommended.
Syntax example:
[T, X] = ode45(@your_ode_equations, time span, initialvalue, options, parameters)

The variable time step ODE solvers will automatically choose the step length Δt and discretize the process.
You can also implement you own MATLAB script for Euler and Runge-Kutta methods for solving or simulating the model of a process described by ODEs.
Important note: In case you are using MATLAB as a tool for developing/implementing MPC algorithm, it is highly recommended that you implement your own MATLAB script for Euler or Runge-Kutta methods. Using the built-in ODE solvers for MPC algorithm makes the iterations very slow.
In this course, however, we focus in the use of Simulink for simulating a process model.

Lecture 2

"MPC is nothing but a simple algorithm where an optimal control problem is solved at each time step and then the receeding/sliding horizon strategy is applied."
An optimal control problem is simply an optimization problem. The optimization problem is created with respect to process control point of view, for example for tracking process outputs to their desired setpoints. An optimal control problem is created by looking into the future from the current time step. To look into the future or in other words, to calculate how the process would behave, say from the current time and up to10 times steps ahead, the model of the process is used. At each new time step, such optimal control problem is created and solved. This means that at each time step, in an MPC, an optimization problem is solved.
So, "OPTIMIZATION" is the basic backbone for optimal control and hence for MPC.
In this lecture, we will learn the basics of optimization: the structure of an optimization problem, different types of optimization problems, how to create an optimization problem and how to solve them using MATLAB. We will also learn how we can create an optimization problem with process control point of view.

2.1 Basic introduction to optimization:

In an optimization problem, in general, we have two main ingredients:

a) **Objective(s) function** to be minimized or maximized :
   The objective function (also known as cost function or performance index) represents the main function that we would like to optimize. It is a function where we setup our main aim or goal for either minimizing or maximizing it. From process control point of view, an example of such an objective could be to track the set point i.e. to minimize the error between the reference/desired value and the actual process output. Other examples of the objective functions can be profit maximization, cost minimization, production maximization etc.

b) **Constraints** that should be obeyed if there are any and if it is possible. The presence of constraints generates a specific region (also called feasible region) and the solution of the optimization problem (also known as optimal solution) should always lie within the feasible region.
   An example of constraints could be limitations in the volumetric flow rate of a fluid through a pipeline i.e. the flow rate should be greater than a specified minimum value or/and less than a specified maximum value. Constraints could be on the input variables, output variables and on the process states as well. The valve opening (which usually is a control input for regulating flow) should be greater than 0% and cannot be more than 100% is yet another example of a constraint. The rate by which a valve can be opened, for example, valve opening should be less than 2% per time step, is another example of input constraint. Usually, the operating conditions of the process give rise to numerous constraints.

We will see a number of examples of objective function and constraints as we move forward with the course. If the optimization problem does not contain any constraints, it is called unconstrained optimization problem and if it does, it forms a constrained optimization problem.

In general, a constrained optimization problem is expressed as

$$\underset{x}{min/max} \quad J = f(x) \rightarrow \text{Objective function} \tag{2.1}$$

subject to,

$$h_i(x) = 0, \quad i = 1, 2, \dots\dots, m \rightarrow \text{Equality constraints} \tag{2.2}$$

$$g_j(x) \leq 0, \; j = 1, 2, \dots\dots, r \rightarrow \text{Inequality constraints} \tag{2.3}$$

$$x_L \leq x \leq x_U \quad x_L \epsilon \mathbb{R}^{n_x}, x_U \epsilon \mathbb{R}^{n_x} \rightarrow \text{Bounds} \tag{2.4}$$

Here, x are the unknown variables, sometimes also called decision variables. They are the variables that need to be optimized in order to either minimize or maximize the objective functions while still satisfying the constraints. $h_i(x) = 0$ is a general expression for the equality constraints and $g_j(x)$ is a general expression for the inequality constraints. They could be linear as well as nonlinear functions. x_L is a column vector with lower bounds on the decision variable i.e. the unknowns x should be equal to or greater than x_L. x_U is a column vector with upper bounds on the decision variable i.e. the unknowns x should be equal to or less than x_U.

Note: Bounds can also be treated as general inequality constraints. For example, we can write $x_L \leq x \leq x_U$ as two separate inequality constraints.

$$x \leq x_U$$
$$-x \leq -x_L$$

Types:

i. If $f(x), h_i(x)$ and $g_j(x)$ are all linear functions, it forms a **Linear Programming (LP)** problem or linear optimization problem. The term *"programming"* here simply refers to optimization (but not to a programming language or something else).

The standard structure of an LP problem is:

$$\underset{x}{min/max} \quad J = f(x) = c^T x \quad x \epsilon \mathbb{R}^{n_x} \rightarrow \text{Linear objective function} \tag{2.5}$$

subject to,

$$A_e x = b_e \rightarrow \text{Linear equality constraints} \tag{2.6}$$

$$A_i x \leq b_i \rightarrow \text{Linear inequality constraints} \tag{2.7}$$

$$x_L \leq x \leq x_U \quad x_L \epsilon \mathbb{R}^{n_x}, x_U \epsilon \mathbb{R}^{n_x} \rightarrow \text{Bounds} \tag{2.8}$$

Here c is a column vector (c^T is a row vector) that contains the coefficient of the unknowns x in the linear objective function. A_e is a matrix containing the coefficients of the unknowns in the linear equality constraints, b_e is a column vector of the constant term in the equality constraints. Similarly, A_i is a matrix containing the coefficients of the unknowns in the linear inequality constraints, b_i is a column vector of the constant term in the inequality constraints

ii. If either $f(x), h_i(x)$ and $g_j(x)$ (any one of them or all of them) is(are) nonlinear function(s), it forms a **NonLinear Programming (NLP) problem**. The standard structure of the NLP is the same as listed in Equations 2.1 – 2.4.

iii. If $f(x)$ is a quadratic[4] function and $h_i(x)$ and $g_j(x)$ are both linear functions, it forms **Quadratic Programming (QP) problem**. In this course, we will have more focus on QP problems.
The standard structure of a QP problem is

$$\underset{x}{min/max} \quad J = f(x) = \frac{1}{2}\, x^T\, Hx + c^T x \quad x\epsilon\mathbb{R}^{n_x} \rightarrow \text{Quadratic objective function} \tag{2.9}$$

subject to,

$$A_e x = \mathrm{b_e} \rightarrow \text{Linear equality constraints} \tag{2.10}$$

$$A_i x \leq b_i \rightarrow \text{Linear inequality constraints} \tag{2.11}$$

$$x_L \leq x \leq x_U \quad x_L\epsilon\mathbb{R}^{n_x}, x_U\epsilon\mathbb{R}^{n_x} \rightarrow \text{Bounds} \tag{2.12}$$

Here H is a symmetric diagonal matrix containing the coefficients of the quadratic terms on its diagonal. c^T is a row vector containing the coefficient of the linear terms.

iv. If $h_i(x)$ and $g_j(x)$ are absent, it forms an **unconstrained optimization problem**.

Note: Maximizing an objective function $f(x)$ can equivalently be written as the minimization of $-f(x)$ as shown in Figure 2.1. In other words, maximizing a function $f(x)$ is the same as minimizing the negative of the function. The structure of many optimization solvers are designed to accept only the minimization of an objective function but they may not have the feature of maximizing an objective function. In such a case, the negative of the objective function can be minimized in order to actually perform the desired maximization.

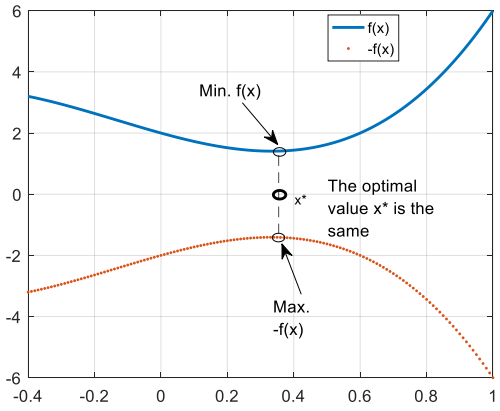


Figure 2.1: Illustration for min. f(x) = max -f(x)

[4] A quadratic function contains one or more variables where the highest degree terms are of second order. For example, $x_1^2 + 3x_2^2 - 5x_3^2 + 2x_4 - 3x_5$ is an example of a quadratic function. Here the highest degree terms x_1, x_2 and x_3 have powers raised to 2. Please note that a quadratic function in addition can also contain terms of first order (x_4 and x_5) in this example.

2.2 Oil Refinery Example

To give you a basic understanding of optimization, an example of optimal production in an oil refinery is taken as an example for illustration. We will formulate an LP and QP optimization problems and solve it.
But at first the problem description:

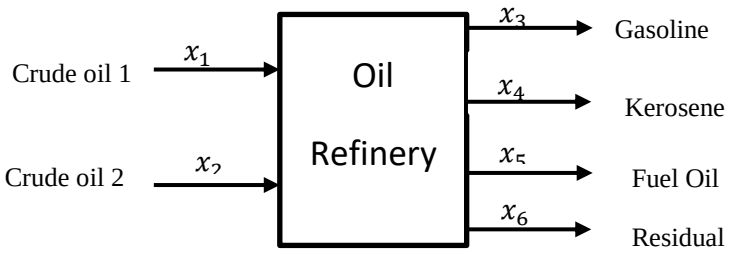


Figure 2.2: Block diagram showing raw materials and products of an oil refinery.

As shown in Figure 2.2, in an oil refinery, the raw material consists of two types of crude oil. The oil refinery purchase these crude oils from vendors. The crude oils are refined in the refinery and the end products are gasoline, kerosene, fuel oil and other residual. The refinery then sells these products. x_1 and x_2 denote the quantity of crude oils used in one day (units: barrel/day). x_3 to x_6 are the production rates of the products in barrel/day.
To produce one barrel (bbl.) of gasoline, 0.8 bbl. of crude oil 1 and 0.44 bbl. of crude oil 2 are required. To produce one barrel (bbl.) of kerosene, 0.05 bbl. of crude oil 1 and 0.1 bbl. of crude oil 2 are required. For one barrel of fuel oil, it requires 0.1 bbl. of crude oil 1 and 0.36 bbl. of crude oil 2. For one barrel of residual, it requires 0.05 bbl. of crude oil 1 and 0.1 bbl. of crude oil 2.

The price of purchasing the raw materials (crude oil) is $p_1 = \$24/\text{bbl}$ for crude oil 1 and $p_2 = \$15/\text{bbl}$ for crude oil 2. The production costs are related to the consumption of crude oil, and these are $c_1 = \$0.5/\text{bbl}$ for crude oil 1 and $c_2 = \$1.00/\text{bbl}$ for crude oil 2. The revenue are related to the selling price of the products: $s_3 = \$36/\text{bbl}$ for gasoline, $s_4 = \$24/\text{bbl}$ for kerosene, $s_5 = \$21/\text{bbl}$ for fuel oil and $s_6 = \$10/\text{bbl}$ for the residual.
The constraints on the system are related to the quantity of the products that can be produced in a day. The maximum amount of gasoline that can be produced in a day is 24000 bbl/day, for kerosene it is 2000 bbl/day and for the fuel oil it is 6000 bbl/day.

Objective: Maximize the profit by production by taking into consideration the constraints in the system.

Formulation of Linear Programming (LP) problem:
Profit is given by (total income – total expense).

$$\text{Total Income} = \text{selling price} * \text{quantity} = 36x_3 + 24x_4 + 21x_5 + 10x_6 \quad [\$/\text{day}]$$

$$\text{Total expense} = \text{buying price} * \text{quantity} + \text{production costs} * \text{quantity}$$
$$= 24x_1 + 15x_2 + 0.5x_1 + 1x_2 = 24.5x_1 + 16x_2 \quad [\$/\text{day}]$$

Now the profit (which is our objective function) is,

$$\text{Profit} = (\text{total income} - \text{total expense})$$
$$= 36x_3 + 24x_4 + 21x_5 + 10x_6 - 24.5x_1 - 16x_2 \quad [\$/\text{day}]$$

The profit is constrained to satisfy the steady state mass balance for the refinery and it gives rise to the following equality constraints:

$$\text{Gasoline: } x_3 = 0.8x_1 + 0.44x_2$$
$$\text{Kerosene: } x_4 = 0.05x_1 + 0.1x_2$$
$$\text{Fuel oil: } x_5 = 0.1x_1 + 0.36x_2$$
$$\text{Residual: } x_6 = 0.05x_1 + 0.1x_2$$

The production constraints are:

$$x_3 \leq 24000$$
$$x_4 \leq 2000$$
$$x_5 \leq 6000$$

Note: The production constraints for x_6 is not available. Similarly no constraints have been stated for x_1 and x_2. Keeping in mind that $x_1, x_2, \dots, x_6$ cannot have negative values (since they are quantities), we can formulate the inequality constraints as:

$$0 \leq x_1 \leq \infty$$
$$0 \leq x_2 \leq \infty$$
$$0 \leq x_3 \leq 24000$$
$$0 \leq x_4 \leq 2000$$
$$0 \leq x_5 \leq 6000$$
$$0 \leq x_6 \leq \infty$$

Note that since no constraints have been stated for x_1 and x_2, we simply assume that they can take values anywhere between 0 and ∞.

Finally the optimization problem can be formulated as,

$$\max_{(x_1, \dots, x_6)} \; J = f(x) = 36x_3 + 24x_4 + 21x_5 + 10x_6 - 24.5x_1 - 16x_2 \rightarrow \text{Objective function} \quad (2.13)$$

subject to,

$$\left.\begin{aligned} x_3 &= 0.8x_1 + 0.44x_2 \\ x_4 &= 0.05x_1 + 0.1x_2 \\ x_5 &= 0.1x_1 + 0.36x_2 \\ x_6 &= 0.05x_1 + 0.1x_2 \end{aligned}\right\} \text{Linear equality constraints} \quad (2.14)$$

$$\left.\begin{aligned} 0 \leq x_1 \leq \infty \\ 0 \leq x_2 \leq \infty \\ 0 \leq x_3 \leq 24000 \\ 0 \leq x_4 \leq 2000 \\ 0 \leq x_5 \leq 6000 \\ 0 \leq x_6 \leq \infty \end{aligned}\right\} \text{Linear inequality constraints} \quad (2.15)$$

2.2.1 Manual Solution of LP problem for oil refinery:

For smaller sized problems, it is possible to manually solve an LP optimization problem. Manual solution is discussed here simply to increase the understanding. We can use the equality constraints to simplify and eliminate variables x_3, x_4, x_5 and x_6. This can be done by inserting the equality constraints of equation 2.14 into the objective function of equation 2.13. We then get,

$$f(x) = 36(0.8x_1 + 0.44x_2) + 24(0.05x_1 + 0.1x_2) + 21(0.1x_1 + 0.36x_2) + 10(0.05x_1 + 0.10x_2)$$

$$f(x) = 8.1x_1 + 10.8x_2$$

Inserting the equality constrains into the inequality constraints for x_3, x_4 and x_5 we get,

$$\begin{aligned} 0.8x_1 + 0.44x_2 &\leq 24000 \\ 0.05x_1 + 0.1x_2 &\leq 2000 \\ 0.1x_1 + 0.36x_2 &\leq 6000 \end{aligned} \tag{2.15a}$$

The above inequality equations of 2.15a can be arranged as,

$$\begin{aligned} x_2 &\leq 54545 - 1.8182x_1 \\ x_2 &\leq 20000 - 0.5x_1 \\ x_2 &\leq 16667 - 0.27778x_1 \end{aligned} \tag{2.15b}$$

These inequality constraints of equation 2.15b defines the region where the solution should lie or be present. Such a region is also known as feasible region. To draw the region or boundary, the following equations of the lines (which defines the boundary) should be drawn.

$$\begin{aligned} x_2 &= 54545 - 1.8182x_1 \\ x_2 &= 20000 - 0.5x_1 \\ x_2 &= 16667 - 0.27778x_1 \end{aligned} \tag{2.15c}$$

The plot of these straight lines is shown in Figure 2.3.

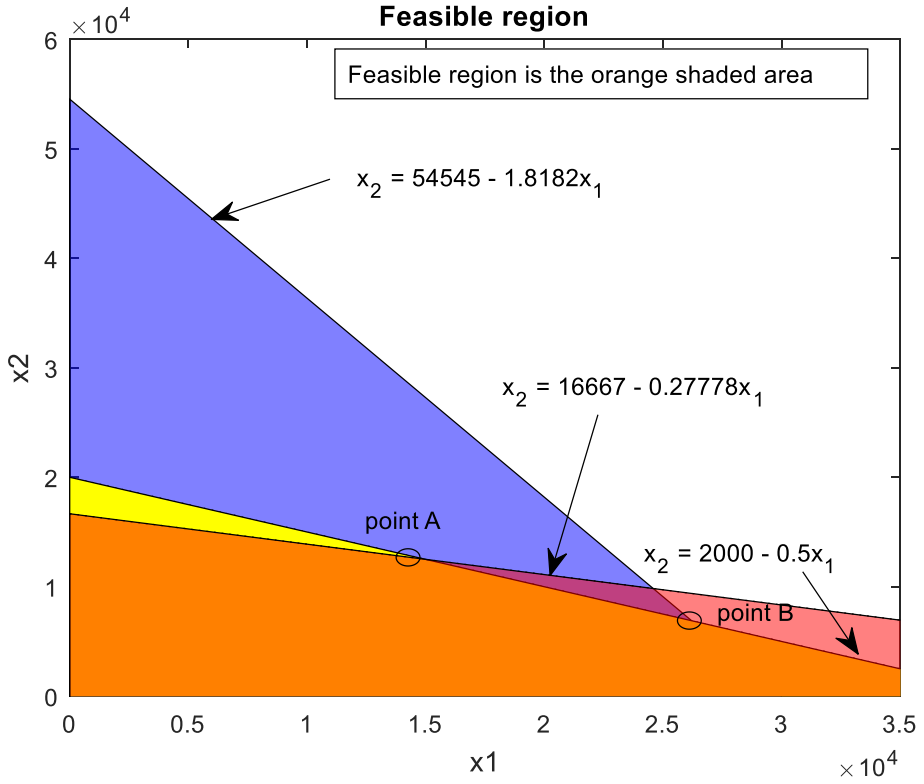


Figure 2.3: Feasible region and the optimal point

The region where all the three inequality constraints will be satisfied is the shaded area in orange color. The solution are the points where the lines intersect each other (in the feasible region). There are two solution points: point A and point B as shown in Figure 2.3. However, the optimal point where the objective function will attain the maximum value while still satisfying all the constraints is point B. Thus, the optimal values of x_1 and x_2 is the point B.

The optimal solution is found out by solving the two steepest inequality constraints as,

$$54545 - 1.8182x_1^* = 20000 - 0.5x_1^*$$
$$x_1^* = 26206$$

Then x_2 is given by

$$x_2^* = 20000 - 0.5x_1^* = 6897$$

Thus the optimal profit is,

$$J^* = f(x^*) = 8.1x_1^* + 10.8x_2^* = 2.8676 \times 10^5 \ \$/\text{day}$$

Then we can find the optimal production values in barrels per day of the products as

$$x_3^* = 0.8 \cdot 26206 + 0.44 \cdot 6897 = 24000$$

$$x_4^* = 0.05 \cdot 26206 + 0.10 \cdot 6897 = 2000$$

$$x_5^* = 0.10 \cdot 26206 + 0.36 \cdot 6897 = 5103.5$$

$$x_6^* = 0.05 \cdot 26206 + 0.10 \cdot 6897 = 2000$$

2.2.2 **Using solver for Linear programming (LP) for oil refinery example:**

It is usually less cumbersome and more easy to use solvers for solving an LP optimization problem. The optimization toolbox in MATLAB includes built-in routines that can be used to solve such problems. One such routine is the *'linprog'* with *'interior-point'* as the default algorithm. Other algorithms available are *'dual-simplex'* and *'active-set'* methods. *'linprog'* accepts the standard form or structure of LP problems as,

$$\min_x \ J = f(x) = c^T x \quad x \epsilon \mathbb{R}^{n_x} \ \rightarrow \text{Linear objective function} \tag{2.16}$$

subject to,

$$A_e x = b_e \ \rightarrow \text{Linear equality constraints} \tag{2.17}$$
$$A_i x \leq b_i \ \rightarrow \text{Linear inequality constraints} \tag{2.18}$$
$$x_L \leq x \leq x_U \quad x_L \epsilon \mathbb{R}^{n_x}, x_U \epsilon \mathbb{R}^{n_x} \rightarrow \text{Bounds} \tag{2.19}$$

An example of the syntax is,

[x, fval]= linprog(c,A_i ,b_i,A_e,b_e,x_L,x_U,x_0,options)

Here, $c, A_e, b_e, A_i, b_i, x_L, x_U$ are the matrices (or vectors) associated with the standard formulation of Equations 2.16 - 2.19. x_0 is the initial values of the unknowns x. Before the optimizer can start

looking for the optimal solution, it needs a point from where it can start the iterative algorithm. This is provided as the initial values of the unknowns x_0 by the user[5]. *Options* are passed to the *linprog* routine which provides us with the choice of the algorithm, tolerance error etc. For details about the different options that can be passed into *linprog* refer to MATLAB documentation.

For the oil refinery example, let us first rewrite the LP problem given by Equations 2.13 - 2.15 into the standard LP problem form/structure given by Equations 2.16 - 2.19. The objective function of Equation 2.13 is a maximization function, however the standard form is the minimization of the objective function. This can be achieved by minimizing $-f(x)$ and can be written in standard form as,

$$\underset{(x_1,\ldots,x_6)}{min} \quad J = -f(x) = \underbrace{[24.5 \quad 16 \quad -36 \quad -24 \quad -21 \quad -10]}_{c^T} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}}_{x} \tag{2.20}$$

The linear equality constraints of Equation 2.14 can be written in standard form as,

$$\underbrace{\begin{bmatrix} 0.8 & 0.44 & -1 & 0 & 0 & 0 \\ 0.05 & 0.1 & 0 & -1 & 0 & 0 \\ 0.1 & 0.36 & 0 & 0 & -1 & 0 \\ 0.05 & 0.1 & 0 & 0 & 0 & -1 \end{bmatrix}}_{A_e} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{b_e} \tag{2.21}$$

The inequality constraints of Equation 2.15 are actually the bounds on the unknown variables. They can be expressed in standard form as,

$$\underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{x_L} \leq \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}}_{x} \leq \underbrace{\begin{bmatrix} \infty \\ \infty \\ 24000 \\ 2000 \\ 6000 \\ \infty \end{bmatrix}}_{x_U} \tag{2.22}$$

Let us choose the initial values of the unknowns as $x_0 = [1000 \quad 1500 \quad 3000 \quad 4000 \quad 2500 \quad 1800]^T$ and use the *active-set* method as the algorithm for solving the LP problem in MATLAB.

>> options = optimoptions('linprog','Algorithm','active-set')

Then call the *linprog* routine as,

>> x = linprog(c,[],[],Ae,be,x_L,x_U,x0,options)

Optimization terminated.

x =

2.6207e+04
6.8966e+03
2.4000e+04
2.0000e+03
5.1034e+03

[5] The latest releases of MATLAB contains algorithms where the initial value is not required to be supplied by the user.

2.0000e+03

Note that any parameters that are not applicable for the optimization problem being solved is replaced by empty matrices or vectors *[]*. Here we have expressed the inequality constraints as bounds, thus $A_i = []$ and $b_i = []$ i.e. matrices related to the linear inequality constraints are taken as empty matrices/vectors.

Note: The algorithms *dual-simplex, simplex, interior-point* and *interior-point legacy* automatically choose the internal starting point x_0. Thus it is not necessary to supply the initial values of the unknowns when these algorithms are chosen. For LP problems, the solution is always located on the vertices (corner points where the constraints intersect) formed by the inequality constraints.

2.2.3 **Quadratic Programming (QP) and oil refinery example:**

Quadratic programming/optimization problems are the base stones for a linear MPC. Thus it is important to look into an example on how a QP problem can be formulated and solved.

Let us look back into oil refinery example of Section 2.2 in order to have a quadratic programming problem.

Let us assume that the price of purchasing the raw materials (crude oil) is $p_1 = \$24/\text{bbl}$ for crude oil 1 and $p_2 = \$15/\text{bbl}$ for crude oil 2. The production costs are related to the consumption of crude oil, and these are $c_1 = \$0.5/\text{bbl}$ for crude oil 1 and $c_2 = \$1.00/\text{bbl}$ for crude oil 2. The revenue or income are related to the selling price of the products: i.e. $s_3 = (36 - \frac{x_3}{5000})$ \$/bbl for gasoline, $s_4 = (24 - \frac{x_4}{5000})$ \$/bbl for kerosene, $s_5 = \$21/\text{bbl}$ for fuel oil and $s_6 = \$10/\text{bbl}$ for the residual.

Now the profit is given by (total income – total expense).

$$\text{Total Income} = \text{selling price} * \text{quantity}$$
$$= \left(36 - \frac{x_3}{5000}\right) x_3 + \left(24 - \frac{x_4}{5000}\right) x_4 + 21x_5 + 10x_6 \quad [\$/\text{day}]$$
$$\text{Total expense} = \text{buying price} * \text{quantity} + \text{production costs} * \text{quantity}$$
$$= 24x_1 + 15x_2 + 0.5x_1 + 1x_2 = 24.5x_1 + 16x_2 \quad [\$/\text{day}]$$

Now the profit (which is our objective function) is,

$$\text{Profit} = (\text{total income} - \text{total expense})$$
$$= 36x_3 - \frac{1}{5000}x_3^2 + 24x_4 - \frac{1}{5000}x_4^2 + 21x_5 + 10x_6 - 24.5x_1 - 16x_2 \qquad [\$/\text{day}]$$

Also let us assume that in order to produce one barrel (bbl.) of gasoline, 0.8 bbl. of crude oil 1 and 0.44 bbl. of crude oil 2 are required. To produce one barrel (bbl.) of kerosene, 0.05 bbl. of crude oil 1 and 0.1 bbl. of crude oil 2 are required. For one barrel of fuel oil, it requires 0.1 bbl. of crude oil 1 and 0.36 bbl. of crude oil 2. For one barrel of residual, it requires 0.05 bbl. of crude oil 1 and 0.1 bbl. of crude oil 2.

This gives rise to the steady state mass balance for the refinery and thus to the following equality constraints:

$$\text{Gasoline: } x_3 = 0.8x_1 + 0.44x_2$$
$$\text{Kerosene: } x_4 = 0.05x_1 + 0.1x_2$$
$$\text{Fuel oil: } x_5 = 0.1x_1 + 0.36x_2$$
$$\text{Residual: } x_6 = 0.05x_1 + 0.1x_2$$

The constraints on the system are related to the quantity of the products that can be produced in a day. The maximum amount of gasoline that can be produced in a day is 24000 bbl/day, for kerosene it is 2000 bbl/day and for the fuel oil it is 6000 bbl/day.
Thus, the production constraints are:

$$x_3 \leq 24000$$
$$x_4 \leq 2000$$
$$x_5 \leq 6000$$

Note: The production constraints for x_6 is not available. Similarly no constraints have been stated for x_1 and x_2. Keeping in mind that $x_1, x_2, \dots, x_6$ cannot have negative values (since they are quantities), we can formulate the inequality constraints as:

$$0 \leq x_1 \leq \infty$$
$$0 \leq x_2 \leq \infty$$
$$0 \leq x_3 \leq 24000$$
$$0 \leq x_4 \leq 2000$$
$$0 \leq x_5 \leq 6000$$
$$0 \leq x_6 \leq \infty$$

Note that since no constraints have been stated for x_1 and x_2, we simply assume that they can take values anywhere between 0 and ∞.
Finally the quadratic optimization problem of maximizing the profit can be formulated as,

$$\underset{(x_1, \dots, x_6)}{max} \quad J = f(x) = 36x_3 - \frac{1}{5000}x_3^2 + 24x_4 - \frac{1}{5000}x_4^2 + 21x_5 + 10x_6 - 24.5x_1 - 16x_2 \tag{2.23}$$

subject to,

$$\left.\begin{aligned} x_3 &= 0.8x_1 + 0.44x_2 \\ x_4 &= 0.05x_1 + 0.1x_2 \\ x_5 &= 0.1x_1 + 0.36x_2 \\ x_6 &= 0.05x_1 + 0.1x_2 \end{aligned}\right\} \text{Linear equality constraints} \tag{2.24}$$

$$\left.\begin{aligned} 0 \leq x_1 \leq \infty \\ 0 \leq x_2 \leq \infty \\ 0 \leq x_3 \leq 24000 \\ 0 \leq x_4 \leq 2000 \\ 0 \leq x_5 \leq 6000 \\ 0 \leq x_6 \leq \infty \end{aligned}\right\} \text{Linear inequality constraints} \tag{2.25}$$

Optimization problem given by Equations 2.23 - 2.25 is a QP problem (the objective function is quadratic and the constraints are linear). It can be solved using QP solvers. But most of the QP solvers are designed to solve the standard QP problem formulation given by Equations 2.9 – 2.12. The oil refinery QP problem given by Equation 2.23 – 2.25 is not in the standard form. Before we

can use QP solvers to solve it, the QP problem given by 2.23 – 2.25 must be first converted to the standard QP problem formulation given by Equation 2.9 – 2.12.

For the oil refinery example, the objective function of Equation 2.23 is a maximization function. The *qpOASES*[6] solver that we will be using to solve a QP problem in Simulink accepts only minimization problems. We have to adjust the objective function of the oil refinery example such that it becomes a minimization problem. This can be achieved by minimizing $-f(x)$ and can be written in standard form as,

$$\min_{(x_1,\dots,x_6)} \quad J = -f(x) = \frac{1}{2}\underbrace{[x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6]}_{x^T}\underbrace{\frac{2}{5000}\begin{bmatrix}0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\end{bmatrix}}_{H}\underbrace{\begin{bmatrix}x_1\\x_2\\x_3\\x_4\\x_5\\x_6\end{bmatrix}}_{x} + \underbrace{[24.5 \quad 16 \quad -36 \quad -24 \quad -21 \quad -10]}_{c^T}\underbrace{\begin{bmatrix}x_1\\x_2\\x_3\\x_4\\x_5\\x_6\end{bmatrix}}_{x} \tag{2.26}$$

The linear equality constraints of Equation 2.24 can be written in standard form as,

$$\underbrace{\begin{bmatrix}0.8&0.44&-1&0&0&0\\0.05&0.1&0&-1&0&0\\0.1&0.36&0&0&-1&0\\0.05&0.1&0&0&0&-1\end{bmatrix}}_{A_e}\underbrace{\begin{bmatrix}x_1\\x_2\\x_3\\x_4\\x_5\\x_6\end{bmatrix}}_{x} = \underbrace{\begin{bmatrix}0\\0\\0\\0\end{bmatrix}}_{b_e} \tag{2.27}$$

The inequality constraints of Equation 2.25 are actually the bounds on the unknown variables. They can be expressed in standard form as,

$$\underbrace{\begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix}}_{x_L} \leq \underbrace{\begin{bmatrix}x_1\\x_2\\x_3\\x_4\\x_5\\x_6\end{bmatrix}}_{x} \leq \underbrace{\begin{bmatrix}\infty\\\infty\\24000\\2000\\6000\\\infty\end{bmatrix}}_{x_U} \tag{2.28}$$

In addition, it is very important to understand that the *qpOASES* solver accepts a special structure of the Quadratic Programming problem as,

$$\min_{(x)} \quad J = f(x) = \frac{1}{2}x^T H x + c^T x, \qquad x \epsilon \mathbb{R}^{n_x} \quad \text{Objective function} \tag{2.29a}$$

subject to,

$$b_L \leq A_e x \leq b_U \quad \text{Linear equality/inequality constraints} \tag{2.29b}$$

$$x_L \leq x \leq x_U \qquad x_L \epsilon \mathbb{R}^{n_x}, x_U \epsilon \mathbb{R}^{n_x} \qquad \text{Bounds} \tag{2.29c}$$

The *qpOASES* solver uses the same notation for both the linear equality and the inequality constraints as shown in Equation 2.29b. For linear inequality constraints, the vectors b_L and b_U take different values. The good point is that we can use the same notation of equation 2.29b to express the linear equality constraints as well. For this the vectors b_L and b_U take the same value i.e. they are the same.

Thus for expressing linear equality constraints in *qpOASES*,

[6] Visit projects.coin-or.org/qpOASES for details about *qpOASES* solver.

$$b_L = b_U = b_e$$

Then we have,

$$b_e \leq Ax \leq b_e$$

This is equivalent to the linear equality equation of the standard QP form

$$A_e x = b_e$$

Refer to the following video for the formulation of the QP problem for the oil refinery and for the structure accepted with *qpOASES*

https://web01.usn.no/~roshans/mpc/videos/lecture2/problem-formulation.mp4

***qpOASES* solver for Quadratic Programming (QP) and oil refinery example**

Before we can start using the *qpOASES* solver in Simulink, we have to first install the solver in computer. Please look into the videos for lecture 2 in the homepage of this course. The following link explains how to install the *qpOASES* solver in your computer.

https://web01.usn.no/~roshans/mpc/videos/lecture2/qpoases-installation.mp4

The *qpOASES* solver is written in plain C++ language. Thus it has to be complied to a mex file before we can use it in Simulink. In this course we will be using TDM-GCC compiler. The following link explains where to download the installer and then how to install the C++ compiler.

https://web01.usn.no/~roshans/mpc/videos/lecture2/tdm64-gcc-installation.mp4

To check whether your compiler and the solver have been successfully installed, see the following video.

https://web01.usn.no/~roshans/mpc/videos/lecture2/compiling-qpoases.mp4

Now the Quadratic optimization problem with the oil refinery example can be solved using *qpOASES* solver in Simulink. The details of the implementation is described in the following video.

https://web01.usn.no/~roshans/mpc/videos/lecture2/refinery-qpoases.mp4

It is also possible to solve a QP problem in MATLAB. The optimization toolbox in MATLAB includes built-in routines that can be used to solve such problems. One such routine is the *'quadprog'* with *'interior-point-convex'* as the default algorithm. Other algorithms available are *'trust-region-reflective'* and *'active-set'* methods. *'quadprog'* accepts the standard form of QP problems in MATLAB as,

$$\min_x \; J = f(x) = \frac{1}{2} x^T H x + c^T x \quad x \epsilon \mathbb{R}^{n_x} \rightarrow \text{Quadratic objective function} \tag{2.30}$$

subject to,

$$A_e x = b_e \rightarrow \text{Linear equality constraints} \tag{2.31}$$

$$A_i x \leq b_i \rightarrow \text{Linear inequality constraints} \tag{2.32}$$

$$x_L \leq x \leq x_U \quad x_L \epsilon \mathbb{R}^{n_x}, x_U \epsilon \mathbb{R}^{n_x} \rightarrow \text{Bounds} \tag{2.33}$$

An example of the syntax is,

[x, fval]= quadprog(H,c,A_i ,b_i,A_e,b_e,x_L,x_U,x_0,options)

Here, $H, c, A_e, b_e, A_i, b_i, x_L, x_U$ are the matrices (or vectors) associated with the standard formulation of Equations 2.30 - 2.33. x_0 is the initial values of the unknowns x. Before the optimizer can start looking for the optimal solution, it needs a point from where it can start the iterative algorithm. This is provided as the initial values of the unknowns x_0 by the user. *Options* are passed to the *quadprog* routine which provides us with the choice of the algorithm, tolerance error, maximum iterations etc. For details about the different options that can be passed into *quadprog* refer to MATLAB documentation.

Let us choose the initial values of the unknowns as $x_0 = [1000 \quad 1500 \quad 3000 \quad 4000 \quad 2500 \quad 1800]^T$ and use the *active-set* method as the algorithm for solving the LP problem.

>> options = optimoptions('quadprog','Algorithm','interior-point-convex')

Then call the *linprog* routine as,

>> x = quadprog(H,c,[],[],Ae,be,xL,xU,x0,options)

The interior-point-convex algorithm does not accept an initial point.
Ignoring X0.

Minimum found that satisfies the constraints.
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the optimality tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.
<stopping criteria details>
x =

1.5000e+04
1.2500e+04
1.7500e+04
2.0000e+03
6.0000e+03
2.0000e+03

Note: The default algorithm *interior-point-convex* does not need an initial point and it generates it internally automatically. If *active-set* is used, then initial point of the unknowns should be specified. Any parameters that are not applicable for the optimization problem being solved is replaced by empty matrices *[]*. Here we have expressed the inequality constraints as bounds, thus $A_i = []$ and $B_i = []$. The complete source code is available in the Fronter.

2.3 Dynamic Optimal Control and Performance Indices:

The oil refinery posed as a Quadratic optimization problem is an example of static optimization where the dynamic model of the process is not used. They are also in reality not control problems. These optimization examples are defined by algebraic equations so they are in a sense steady state optimization problems. With a model predictive control, the dynamic model of the process is used for formulating the optimization problem, which is normally a control problem (e.g. tracking a set point) and the control action is dynamic. This gives rise to dynamic optimal control: dynamic because the model of the process captures its dynamics (and the generated control action acts on

the dynamics), optimal because the control actions/signals are obtained by solving an optimization problem. Let us define a dynamic process with input/output and states as shown in Figure 2.4.

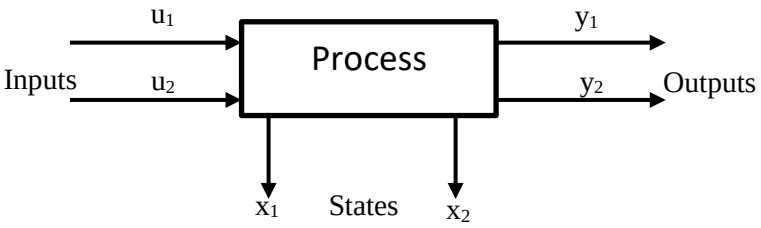


Figure 2.4: Dynamic process with inputs, outputs and states

For simplicity, two outputs and two states are shown but they could be more than two. In this course, we do not fouls on the development of the models of the processes. We assume that the model of the process is known or available. The model can be simulated from any start time to any end time (given that the initial values at the start time is known).

Now to formulate an optimization problem for creating an optimal control problem, what we do is: *we use the model i.e. we simulate the model from a given start time (with known/estimated initial values) to N steps forward in time and obtain the future values of the states and the outputs (from given start time to N steps ahead of this time, with dt being the time step). We then use these future values to create an optimal control problem (that has an objective function and constraints), solve this control problem by using optimization solvers (e.g. qpOASES for QP problem) and then find out a set of N control actions/control inputs (the optimal values that are found by solving the optimization problem). These optimal values of the control actions are then applied to the process.*

The set of *N* control moves should be calculated based on usually minimizing a criterion or an objective and at the same time obeying all the constraints. This criterion is also called performance index. The most general/common performance index with respect to control point of views is to track the output of the process to a given/desired set point value by using the optimal control inputs. The performance index can be both linear and non-linear. But we will start by first looking into the objective functions or performance index that is quadratic in nature. This is also the most common structure of the performance index or objective function for optimal control problems.

2.3.1 Dynamic optimal control problem formulation for tracking

Let us consider that we have a process and we would like to control the output of the process to a setpoint (or reference line) as shown in Figure 2.5. We would like to control the process output from the starting time step (0 in Figure 2.5) to *N* time steps ahead in the future. The reference line (written as Refline in Figure 2.5) is also defined for all these *N* time steps.

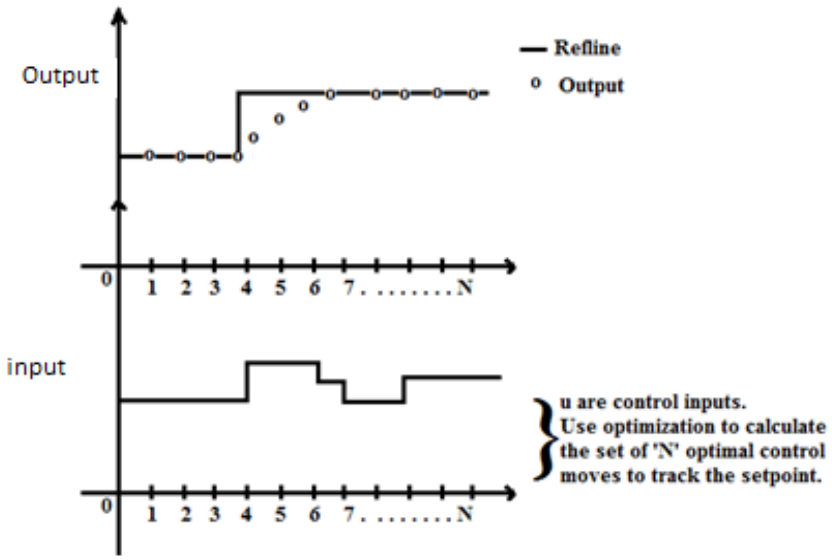


Figure 2.5: Dynamic optimal control

Now let us consider that the model of the process is given by discrete time linear state space representation as

$$x_{k+1} = Ax_k + Bu_k \quad \text{with } x_0 \text{ known} \quad \text{State equation} \quad (2.34)$$

$$y_k = Cx_k \quad \text{Measurement equation} \quad (2.35)$$

Here, k represents the discrete time steps. It is necessary that the values of the states at the initial time step of $k = 0$ i.e. x_0 is known.

From control point of view, our objective is to make the output of the process (y_k) be as close as possible to the set point or the Refline (denoted as r_k). In other words, our objective from control point of view is to reduce (minimize) the difference between the set point value and the actual process output value. If we define this difference as the error (e_k) in set point tracking, then we have,

$$e_k = r_k - y_k \quad \text{Error equation} \quad (2.36)$$

In addition, we would like to minimize this error (e_k) from the discrete time step $k = 1$ to $k = N$. The total number of time steps that we look ahead in the future starting from the initial time step is called the prediction horizon. In Figure 2.5, we are looking *N* time steps into the future from the current time step. **Thus *N* is the prediction horizon.**

If we want to minimize error (e_k) from $k = 1$ to $k = N$, Equation 2.35 says that we have to know the values of the process outputs for the whole prediction horizon length i.e. from $k = 1$ to $k = N$. For this, the model of the process given by equations 2.34 and 2.35 is utilized to predict the future time steps behavior of the process. To do so, the process states at the initial or current time step should be known i.e. x_0 should be known. For example: If x_0 is known then we can use equation 2.34 to calculate x_1 for time step $k = 1$ with a certain value of the control input u_0. Then we can use equation 2.35 to calculate y_1 for time step $k = 1$. Further, we can then use x_1 and control input u_1

to calculate x_2 for time step $k = 2$ using equation 2.34. Then we can again use equation 2.35 to calculate y_2 for time step $k = 2$ and so on for the whole prediction horizon length.
Another important thing to understand is that, in order to minimize the error e_k throughout the prediction horizon, the only way possible is by changing the control input signals (u_k).
Thus our final objective function for an optimal control problem for set point tracking is,

$$\underset{(u)}{min} \quad J = \frac{1}{2}\sum_{k=1}^{N}(e_k^T\, Q_k e_k + u_{k-1}^T\, P_{k-1}\, u_{k-1}) \tag{2.37}$$

where,

e_k = error in setpoint tracking = $y_k - r_k$

r_k = reference/setpoint value which are defined for the whole prediction horizon.

The $\sum$ symbol in equation 2.37 indicates that we would like to minimize the sum of the errors for the whole prediction horizon length. We achieve the setpoint or the reference point at the expense of the control signals or inputs (which give rise to the 2nd term $u_{k-1}^T\, P_{k-1}\, u_{k-1}$). At the same time we also want this expenses of the control inputs to be small so we minimize it.
Here, Q_k = weighting matrix for the error signal for each time step within the prediction horizon. It should be positive definite i.e. $e_k^T\, Q_k e_k > 0$ for $e_k \neq 0$. It defines the weight that you want to put in minimizing the error term.

P_k = weighting matrix on control input variables for each time step within the prediction horizon. It should be positive semi definite i.e. $u_k^T P_k u_k > 0$ for $u_k \neq 0$. It defines the weight that you want to put to limit the expense of the control input signals. With lower weight on control inputs, they usually become aggressive i.e. they are sacrificed heavily/their expense is higher.

Note: u_{k-1} is considered instead of u_k because the current output is the result of the previous input or present input will be used to calculate the next time step output.

In an optimal control problem formulation, generally the weighting matrices are taken to the same for the whole prediction horizon i.e. $Q_1 = Q_2 = \cdots = Q_N = Q$. Similarly, $P_0 = P_1 = \cdots = P_{N-1} = P$.

Some more examples of the performance indices:
Another variant for formulating the objective or performance index of an optimal control problem could be to take into account the **deviation or rate of change of control inputs** Δu_k where,

$$\Delta u_k = u_k - u_{k-1}$$

Then the performance index can be written as,

$$\underset{(\Delta u)}{min} \quad J = \sum_{k=1}^{N}(e_k^T\, Q e_k + \Delta u_{k-1}^T P_{k-1} \Delta u_{k-1})$$

We minimize this criterion with respect to Δu but apply $u_k = \Delta u_k + u_{k-1}$ as the control action.

e.g.: Motor, valves and other types of actuators may have limits on their dynamic performance i.e. the amount with which they can be changed in a single time step may be limited. For example in many operations, control valves cannot change from fully closed position to fully open position in one time step. In other words, if we want to restrict the rate of change of control inputs this formulation is suitable for optimal control problem formulation.

It is also possible to have an **economic profit criteria** in the objective function. As for an example,

$$\max \; J \; = \sum_{i=1}^{N} \|p_i y_i\|_p - \; \|c_{i-1} u_{i-1}\|_p \; \rightarrow \text{objective function}$$

where, $p_i \; \geq 0$ is the price of product $y_{i,} \;\; c_i \; \geq 0$ is the cost of raw material u_i and $\|(.)\|_p$ is the norm *p*. Usually *p=1* means simple addition of the elements of the vectors.
Another example of a performance index could be **production maximization** as,

$$\max \; J = \sum_{i=1}^{N} \lambda_p \left\| y_{pr}(x, u, \theta) \right\|_2$$

where, $y_{pr}(x, u, \theta)$ is some function for calculating the production and λ_p is weighting factor.

In this course, we will limit ourselves with the formulation of an optimal control problem from a control point of view i.e. we will focus only on optimal control problem for set point tracking.
One way of writing the complete formulation of an optimal control problem for setpoint tracking is,

$$\min_{(u)} \; J = \frac{1}{2} \sum_{k=1}^{N} (e_k^T \; Q_k e_k + u_{k-1}^T \; P_{k-1} \; u_{k-1}) \tag{2.38}$$

subject to,

$$\begin{aligned}
& x_{k+1} = \; Ax_k + \; Bu_k \, , \qquad k = \; 0, 1, 2, \ldots \ldots \ldots \ldots, N-1 \quad x_0 \text{ , } known \\
& y_k = Cx_k \\
& e_k = r_k - y_k \quad \text{with } r_k \text{ known or defined for whole prediction horizon length} \\
& x_L \leq \; x_k \leq x_U \\
& u_L \leq \; u_k \leq u_U \\
& \Delta u_L \leq \; \Delta u_k \leq \Delta u_U \\
& Q_k \geq 0 \\
& P_k \geq 0 \\
& \Delta u_k = \; u_k - u_{k-1}
\end{aligned} \tag{2.39}$$

Here, $x_L, u_L \; \& \; \Delta u_L$ are the lower limits for the states, control inputs and the rate of change of control inputs. Similarly, $x_U, u_U \; \& \; \Delta u_U$ are the upper limits for the states, control inputs and the rate of change of control inputs.
This formulation has *"Quadratic criteria or objective"* with linear process model (that forms the linear constraints). This type of optimal control problem is well known as ***"Linear Quadratic (LQ)"*** **optimal control.**

In general, the standard formulation for the quadratic programming (QP) problem is written as,

$$\min_{z} \quad \frac{1}{2} z^T Hz + c^T z \qquad \text{Quadratic objective} \tag{2.40}$$

subject to,

$$\begin{aligned} A_e z &= b_e && \text{Equality constraints} \\ A_i z &\leq b_i && \text{Inequality constraints} \\ z_L &\leq z \leq z_U && \text{Bounds} \end{aligned} \tag{2.41}$$

Here z is the vector of decision variable. The control input signals can be one of the decision variables in addition to other variables.
Many QP solvers including the *qpOASES* solver accept the standard formulation for the QP problems. This is also the structure that MATLAB's function/solver *'quadprog'* accepts.

Thus, the LQ optimal control problem for set point tracking formulated in equations 2.38 and 2.39 should be first expressed as a standard QP problem of equations 2.40 and 2.41. Then we can solve the standard QP problem and obtain the solution of the LQ optimal control problem.

LQ optimal control problem of Equations (2.38) and (2.39) can be efficiently formulated as standard QP problem of Equations (2.40) and (2.41) by using Kornecker product. With this formulation we will obtain a spare QP structure of the corresponding LQ optimal control problem.

Lecture 3

Goal: Efficient formulation of LQ optimal control problem as Standard Quadratic Programming (QP). Use of Kronecker products.

In this chapter, the setpoint tracking LQ optimal control problem given by Equations (2.46) and (2.47) from lecture 2 is considered. This control problem is expressed or formulated as a standard Quadratic Programming (QP) given by Equations (2.48) and (2.49) in lecture 2. The reason that we should formulate the LQ optimal control problem into a standard QP optimization problem is that the control problem becomes well structured and we can use the optimization solvers (which usually are based on the standard QP formulation) to solve the control problem.
The given LQ optimal control problem can be efficiently formulated as QP optimization problem by constructing well structured matrices by using the Kronecker product. But before we jump into the formulation, let us first look into some simple examples.

3.1 First some simple examples

(i)

a) **Quadratic Objective:**

$$J = f(x) = -5x_3^2 + 3x_1^2 + 2.5x_1 + 7x_2$$

b) **Linear Constraints:**

$$2x_1 + 3x_2 + 4x_3 = 5$$
$$-3x_1 - 8x_3 = -6$$
$$-4x_1 - 3.2x_2 + 1.8x_3 = 2$$

Express (a) and (b) in the standard form

$$J = f(x) = \frac{1}{2} x^T Hx + c^T x$$
$$A_\epsilon x = b_\epsilon$$

Let $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, $x^T = [x_1 \quad x_2 \quad x_3]$

Then, $J = f(x) = [x_1 \quad x_2 \quad x_3] \begin{bmatrix} 3 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + [2.5 \quad 7 \quad 0] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$

$$J = f(x) = \frac{1}{2} \underbrace{[x_1 \quad x_2 \quad x_3]}_{x^T} \underbrace{\begin{bmatrix} 2 \times 3 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -5 \times 2 \end{bmatrix}}_{H} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} + \underbrace{[2.5 \quad 7 \quad 0]}_{c^T} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x}$$

$$\therefore J = f(x) = \frac{1}{2} x^T Hx + c^T x$$

Now express the linear constraints in the for $A_\epsilon x = b_\epsilon$

$$\underbrace{\begin{bmatrix} 2 & 3 & 4 \\ -3 & 0 & -8 \\ -4 & -3.2 & 1.8 \end{bmatrix}}_{A_e} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 5 \\ -6 \\ 2 \end{bmatrix}}_{b_e}$$

(ii)

$$J = f(x) = 36x_3 - \frac{1}{5000}x_3^2 + 24x_4 - \frac{1}{5000}x_4^2 + 21x_5 + 10x_6 - 24.5x_1 - 16x_2$$

Let,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \text{ or } x^T = [x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6]$$

Then $J = f(x) = 0x_1^2 + 0x_2^2 - \frac{1}{5000}x_3^2 - \frac{1}{5000}x_4^2 - 0x_5^2 - 0x_6^2 - 24.5x_1 - 16x_2 + 36x_3 + 24x_4 + 4x_5 + 10x_6$

$$f(x) = \underbrace{[x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6]}_{x^T} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{5000} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{5000} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

$$+ \underbrace{[-24.5 \quad -16 \quad 36 \quad 24 \quad 21 \quad 10]}_{c^T} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

$$f(x) = \frac{1}{2}x^T \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{2}{5000} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{2}{5000} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{H} x + c^T x$$

$$\therefore f(x) = \frac{1}{2}x^T Hx + c^T x$$

3.2 Useful matrices and their structures

Please see also the video in the homepage of the course for some examples in MATLAB.
http://web01.usn.no/~roshans/mpc/videos/lecture3/useful_matrices_structures.mp4

a) **Kronecker product** (Symbol ⊗)

Let $A \in R^{m \times n}$ and B be any size matrix, then Kronecker product of A and B is,

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix} = \text{kron (A,B) in MATLAB}$$

Here, each element of the matrix A is being multiplied with the whole matrix B.

b) **Identity matrix : I**

$$I_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}_{nxn} = \text{eye (n) in MATLAB}$$

The diagonal elements are all unity.

c) **Identity matrix whose diagonal elements are offset by position 'k'** ($I_{n,k}$):

$I_{n,k} = \text{diag}\left(1_{n-|k| \times 1}, k\right)$ → Diagonal elements or 1's are offset by 'k'

$= \text{diag (ones}(n - \text{abs}(k), 1), k) \text{ in MATLAB}$

Example:

n = 4, k = 1

$$I_{4,1} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

n = 4, k = -1

$$I_{4,-1} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

d) **Block diagonal**

blkdiag ($A_{11}, A_{22}, A_{33}, A_{44}, \ldots\ldots\ldots A_{nn}$) in MATLAB will produce

$$\begin{bmatrix} A_{11} & 0 & 0 & \cdots\cdots \\ 0 & A_{22} & 0 & \cdots\cdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{nn} \end{bmatrix}$$

The matrices (or vectors) $A_{11}, A_{22}, A_{33}, A_{44}, \ldots\ldots\ldots A_{nn}$ are placed as the diagonal elements in a matrix.

e) **Kronecker product with an identity matrix:**

$$I_n \otimes B = \begin{bmatrix} B & 0 & 0 & \cdots & \cdots \\ 0 & B & 0 & \cdots & \cdots \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \cdots & B & \end{bmatrix} = \text{blkdiag}\underbrace{(B, B, \ldots, B)}_{\text{n times}}$$

f) **Block diagonal matrix offset by position 'k':**

$$I_{n,k} \otimes A = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & 0 \\ A & 0 & 0 & \ldots & 0 & 0 \\ 0 & A & 0 & \ldots & 0 & 0 \\ 0 & 0 & A & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & A & 0 \end{bmatrix}$$

e.g. $I_{3,-1} \otimes A = \begin{bmatrix} 0 & 0 & 0 \\ A & 0 & 0 \\ 0 & A & 0 \end{bmatrix}$, $I_{3,1} \otimes B = \begin{bmatrix} 0 & B & 0 \\ 0 & 0 & B \\ 0 & 0 & 0 \end{bmatrix}$

g) **Identity matrix + a block diagonal matrix offset by position 'k':**

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \ldots & 0 & 0 \\ A & 1 & 0 & \ldots & 0 & 0 \\ 0 & A & 1 & \ldots & 0 & 0 \\ 0 & 0 & A & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & A & 1 \end{bmatrix} = I_n + (I_{n,k} \otimes A)$$

e.g. $I_3 + (I_{3,-1} \otimes A) = \begin{bmatrix} 1 & 0 & 0 \\ A & 1 & 0 \\ 0 & A & 1 \end{bmatrix}$ $\qquad$ $I_3 - (I_{3,-1} \otimes A) = \begin{bmatrix} 1 & 0 & 0 \\ -A & 1 & 0 \\ 0 & -A & 1 \end{bmatrix}$

3.3 Efficient formulation of LQ optimal control problem

In general, let us consider the following for LQ optimal control problem with the future prediction horizon of N time steps.

a) **Quadratic Objective function:**

$$min \quad J = \frac{1}{2} \sum_{k=1}^{N} e_k^T \, Q_k e_k + u_{k-1}^T \, P_{k-1} u_{k-1} \tag{3.1}$$

b) **Linear Constraint (given by the process model)**

$x_{k+1} = Ax_k + Bu_k$ with $x \epsilon \mathbb{R}^{n_x \times 1}$, $u \epsilon \mathbb{R}^{n_u \times 1}$ (3.2)

$y_k = Cx_k$ with $y \epsilon \mathbb{R}^{n_y \times 1}$ and x_0 is given/known (3.3)

The error in tracking the set point r_k is,

$e_k = r_k - y_k$ (3.4)

Here, n_x= no. of states, n_u = no. of control inputs, n_y = no. of outputs
Q_k = weighting matrix for e_k and is positive definite. It is a diagonal matrix with n_y number of weighting elements on the diagonal corresponding to each output.
P_k = weighting matrix for u_k and is positive semi definite. It is a diagonal matrix with n_u number of weighting elements on the diagonal corresponding to each inputs.
r_k = reference signal or setpoint value

We want to express LQ optimal control problem of equations (3.1) - (3.4) in a standard QP form i.e. into the form given by equations 3.5 and 3.6,

$$\min_{z} \ \frac{1}{2} z^T Hz + c^T z \tag{3.5}$$

subject to,

$$\begin{aligned} A_e z &= b_e && \rightarrow \text{equality constraint} \\ A_i z &\leq b_i && \rightarrow \text{inequality constraint} \\ z_L \leq z &\leq z_U && \rightarrow \text{bounds} \end{aligned} \tag{3.6}$$

Let us first define the vector of unknowns z. This vector contains the variables to be optimized. For control purpose, obviously one of the unknowns that needs to be optimized are the control inputs. Why? because we try to fulfill the objective (setpoint tracking) and the constraints by finding the very best (optimal) values of the control inputs. However, in addition to the control inputs, other variables like the process states, outputs, error signal etc. can also be included in the vector of unknowns. In this sense, we can say that we have the flexibility in defining z, and its contents can vary.
Let us for example define the vector of unknowns z as follows,

$$z = \begin{bmatrix} u \\ x \\ e \\ y \end{bmatrix} \quad i.e. \quad z^T = (u^T, x^T, e^T, y^T) = \begin{bmatrix} u \\ x \\ e \\ y \end{bmatrix}^T$$

Remember that we are trying to control the process output throughout the whole prediction horizon (from the current time step and N steps into the future). Thus the unknown variables (which also includes the control inputs) should be optimized for the whole prediction horizon. We then have,

$$\begin{aligned} u^T &= (u_0^T, u_1^T, \ldots\ldots\ldots\ldots u_{N-1}^T) && , u^T \epsilon \ \mathbb{R}^{(1\times N.n_u)} \\ x^T &= (x_1^T, x_2^T, \ldots\ldots\ldots\ldots x_N^T) && , x^T \epsilon \ \mathbb{R}^{(1\times N.n_x)} \\ e^T &= (e_1^T, e_2^T, \ldots\ldots\ldots\ldots e_N^T) && , e^T \epsilon \ \mathbb{R}^{(1\times N.n_y)} \\ y^T &= (y_1^T, y_2^T, \ldots\ldots\ldots\ldots y_N^T) && , y^T \epsilon \ \mathbb{R}^{(1\times N.n_y)} \end{aligned}$$

Note:

For processes with single input, u_{k-1} at each time step i.e. $u_0, u_1, u_2, \ldots u_{N-1}$ for k = 1 to k = N will be each a scalar number. However, for processes with multiple inputs, u_{k-1} at each time step i.e. $u_0, u_1, u_2, \ldots u_{N-1}$ for k = 1 to k = N will be each a vector. For example, for a

system with $n_u = 2$ number of control inputs, the control inputs at time $k = 1$ is $u_0^T = (u_0^1, u_0^2)$. Here the superscript 1 means the first control input and superscript 2 means the second control input (out of the multiple inputs available in the process)
At time $k = 2$, the control inputs $u_1^T = (u_1^1, u_1^2)$ and so on for other time steps.
The same applies for the outputs also. If a system has more than one (multiple) outputs, then y_k at each time step i.e. $y_1, y_2, y_3, \dots y_N$ for $k = 1$ to $k = N$ will be each a vector. For example, for a system with $n_y = 3$ number of outputs, the outputs at time $k = 1$ is $y_1^T = (y_1^1, y_1^2, y_1^3)$. Here the superscript 1 means the first process output, superscript 2 means the second process output and superscript 3 means the third process output (out of the multiple inputs available in the process)
At time $k = 2$, it is $y_2^T = (y_2^1, y_2^2, y_2^3)$ and so on for other time steps.

Based on our choice, the total number of unknowns (n_z) in vector z for the whole prediction horizon length (N) is,

$$n_Z = N\,(n_u + n_x + n_y + n_y)$$

If we expand the objective function J of equation 3.1 for $k = 1$ to $k = N$ we will get,

$$J = \frac{1}{2}[e_1^T Q_1 e_1 + e_2^T Q_2 e_2 + \cdots + e_N^T Q_N e_N + u_0^T P_0 u_0 + u_1^T P_1 u_1 + \cdots + u_{N-1}^T P_{N-1} u_{N-1}] \tag{3.7}$$

Now with our choice of vector z, the standard quadratic objective function of equation 3.5 can be written as,

$$J = \frac{1}{2}\, z^T Hz + c^T z$$

$$J = \frac{1}{2}\underbrace{\begin{bmatrix} u \\ x \\ e \\ y \end{bmatrix}^T}_{z^T} \underbrace{\begin{bmatrix} H_{11} & 0 & 0 & 0 \\ 0 & H_{22} & 0 & 0 \\ 0 & 0 & H_{33} & 0 \\ 0 & 0 & 0 & H_{44} \end{bmatrix}}_{H} \underbrace{\begin{bmatrix} u \\ x \\ e \\ y \end{bmatrix}}_{z} + \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}^T}_{c^T} \underbrace{\begin{bmatrix} u \\ x \\ e \\ y \end{bmatrix}}_{z} \tag{3.8}$$

Here we have arranged the symmetric diagonal matrix with **four** elements H_{11}, H_{22}, H_{33} and H_{44} on the diagonal because the vector z contains **four** different types of elements (u, x, e and y). Similar argument holds also for the vector c.
Now if we multiply the matrices in equation 3.8 we get,

$$J = \frac{1}{2}[u^T H_{11} u + x^T H_{22} x + e^T H_{33} e + y^T H_{44} y] + c_1^T u + c_2^T x + c_3^T e + c_4^T y \tag{3.9}$$

Comparing (3.7) with (3.9),

$$u^T H_{11} u = u_0^T P_0 u_0 + u_1^T P_1 u_1 + \cdots + u_{N-1}^T P_{N-1} u_{N-1}$$

In matrix form,

$$u^T H_{11} u = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}^T \begin{bmatrix} P_0 & 0 & \cdots & 0 \\ 0 & P_1 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & P_{N-1} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \tag{3.10}$$

Comparing both sides of equation 3.10 we finally can write,

$$H_{11} = \begin{bmatrix} P_0 & 0 & \cdots & 0 \\ 0 & P_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & P_{N-1} \end{bmatrix}$$

If, $P_0 = P_1 = \ldots\ldots\ldots\ldots = P_{N-1} = P$ with $P \epsilon \mathbb{R}^{n_u \times n_u}$, i.e. taking the same value of the weighting matrix (the weighting matrix P should be a diagonal matrix, with weights for each control input placed on the diagonal) for the whole prediction horizon length, then we have

$$\boldsymbol{H_{11}} = \begin{bmatrix} \boldsymbol{P} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{P} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{P} \end{bmatrix} = \boldsymbol{I_N \otimes P}$$

Note:

I_N = Identify matrix of size 'N'

$\otimes$ = Kronecker product

Again Comparing (3.7) with (3.9) we see that there is no 'x' term in equation (3.7). So we have,

$$x^T H_{22} x = x_1^T 0 x_1 + x_2^T 0 x_2 + \cdots + x_N^T 0 x_N$$

In matrix form,

$$x^T H_{22} x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}^T \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \tag{3.11}$$

Comparing both sides of equation 3.11 we get,

$$\boldsymbol{H_{22}} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} = \boldsymbol{0_{N.n_x \times N.n_x} = I_N \otimes 0_{n_x \times n_x}}$$

Similarly, for the error term, comparing (3.7) with (3.9) we have,

$$e^T H_{33} e = e_1^T Q_1 e_1 + e_2^T Q_2 e_2 + \cdots + e_N^T Q_N e_N$$

In matrix form,

$$e^T H_{33} e = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix}^T \begin{bmatrix} Q_1 & 0 & \cdots & 0 \\ 0 & Q_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & Q_N \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} \tag{3.12}$$

Comparing both sides of equation 3.12 we get,

$$H_{33} = \begin{bmatrix} Q_1 & 0 & \cdots & 0 \\ 0 & Q_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & Q_N \end{bmatrix}$$

If, $Q_1 = Q_2 = \ldots\ldots\ldots\ldots = Q_N = Q$ with $Q \epsilon R^{n_y \times n_y}$, i.e. taking the same value of the weighting matrix (the weighting matrix Q should be a diagonal matrix) for the whole prediction horizon length, we get,

$$\boldsymbol{H_{33}} = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{Q} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{Q} \end{bmatrix} = \boldsymbol{I_N \otimes Q}$$

Again Comparing (3.7) with (3.9) we see that there is no 'y' term in equation (3.7). So we have,

$$y^T H_{44} x = y_1^T 0 y + y_2^T 0 y_2 + \cdots + y_N^T 0 y_N$$

In matrix form,

$$y^T H_{44} y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}^T \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{3.13}$$

Comparing both sides of equation 3.13 we get,

$$\mathbf{H_{44}} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} = \mathbf{0_{N.n_y \times N.n_y}} = \mathbf{I_N \otimes 0_{n_y \times n_y}}$$

Note: $0_{\mathrm{N.n_y \times N.n_y}} = \mathrm{zeros\ (N.\,n_y, N.\,n_y)}$ in MATLAB

Then the H matrix of the standard quadratic objective function can be written as,

$$H = \mathrm{blkdiag}\ (H_{11}, H_{22}, H_{33}, H_{44})$$
$$= \mathrm{blkdiag}\ (I_N \otimes P,\ 0_{N.n_x \times N.n_x},\ I_N \otimes Q,\ 0_{N.n_y \times N.n_y})$$

In addition, we can clearly see that in equation (3.7) we do not have any linear term (term where variables have order 1, i.e. power raised to 1). Then comparing equations 3.7 and 3.9 we get,

$$c_1^T u = 0u_0 + 0u_1 + \cdots + 0u_{N-1} \tag{3.14}$$

In matrix form,

$$c_1^T u = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}$$

Comparing both sides of equation 3.14 we get,

$$c_1^T = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}$$

Thus we have,

$$c_1 = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = 0_{N.n_u \times 1}$$

In a similar manner, comparing equations 3.7 and 3.9 for the remaining linear terms we get,

$$c_2^T x = 0x_1 + 0x_2 + \cdots + 0x_N \tag{3.15}$$
$$c_3^T e = 0e_1 + 0e_2 + \cdots + 0e_N \tag{3.16}$$
$$c_4^T y = 0y_1 + 0y_2 + \cdots + 0y_N \tag{3.17}$$

In matrix form,

$$c_2^T x = [0 \quad 0 \quad \cdots \quad 0] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \tag{3.18}$$

$$c_3^T e = [0 \quad 0 \quad \cdots \quad 0] \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} \tag{3.19}$$

$$c_4^T y = [0 \quad 0 \quad \cdots \quad 0] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{3.20}$$

Comparing both sides of equation 3.18, 3.19 and 3.20 respectively, we get,

$$c_2^T = [0 \quad 0 \quad \cdots \quad 0] \tag{3.21}$$
$$c_3^T = [0 \quad 0 \quad \cdots \quad 0] \tag{3.22}$$
$$c_4^T = [0 \quad 0 \quad \cdots \quad 0] \tag{3.23}$$

Thus we have,

$$c_2 = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = 0_{N.n_x \times 1} \qquad c_3 = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = 0_{N.n_y \times 1} \qquad c_4 = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = 0_{N.n_y \times 1}$$

Putting them together in vector c we get,

$$c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0_{N.n_u} \\ 0_{N.n_x} \\ 0_{N.n_y} \\ 0_{N.n_y} \end{bmatrix} = 0_{(n_z \times 1)} \qquad n_Z = \text{ no. of total unknowns}$$

Note: $0_{(n_z \times 1)}$ = zeros (n_Z , 1) → in MATLAB

Now let us express the equality constraints of the LQ optimal control problem given by equation (3.2) - (3.4) in the standard QP form $A_e z = b_e$ given by equation 3.6. Let us first organize the matrix A_e and vector b_e. From the optimal control problem, we have in total **three** equality constraints (equations 3.2, 3.3 and 3.4), therefore, let us group them into **three** groups (three rows). For each group (or each row), it is easier to find the structured matrices if we separate them out (in columns) according to the elements in the unknown vector z. Then we have,

$$\underbrace{\begin{bmatrix} A_{e,1u} & A_{e,1x} & A_{e,1e} & A_{e,1y} \\ A_{e,2u} & A_{e,2x} & A_{e,2e} & A_{e,2y} \\ A_{e,3u} & A_{e,3x} & A_{e,3e} & A_{e,3y} \end{bmatrix}}_{A_e} \begin{bmatrix} u \\ x \\ e \\ y \end{bmatrix} = \underbrace{\begin{bmatrix} b_{e,1} \\ b_{e,2} \\ b_{e,3} \end{bmatrix}}_{b_e} \tag{3.24}$$

Each row of A_e in equation (3.24) corresponds to each equality constraint of equations (3.2) - (3.4). Each column of A_e represents the an element of the unknown vector z i.e. first column corresponds to the first element of z which is the control inputs and so on. We will now find the structures for each of the elements of A_e in equation (3.24).

Let us first consider the equality constraint given by equation (3.2)

$$x_{k+1} = Ax_k + Bu_k$$
$$x_{k+1} - Ax_k - Bu_k = 0$$
$$x_k - Ax_{k-1} - Bu_{k-1} = 0$$

We should obey the constraints for the whole prediction horizon length. Since we consider a prediction horizon from $k = 1$ to $k = N$ we have,

$$\begin{aligned} x_1 - Ax_0 - Bu_0 \rightarrow \quad x_1 - Bu_0 &= Ax_0 && \text{for } k = 1 \\ x_2 - Ax_1 - Bu_1 &= 0_{(n_x \times 1)} && \text{for } k = 2 \\ \vdots \qquad\qquad\quad \vdots & \qquad\quad \vdots && \qquad\quad \vdots \\ x_N - Ax_{N-1} - Bu_{N-1} &= 0_{(n_x \times 1)} && \text{for } k = N \end{aligned} \tag{3.25}$$

Note: We take Ax_0 to the right hand side because x_0 is known and hence the term Ax_0 is known. Remember to take all the known terms to the right hand side and all the unknown terms to the left hand side.

Arranging the set of equations 3.25 obtained from the first equality constraints into a matrix form we get the first row of A_e in equation (3.24) as,

$$\underbrace{\begin{bmatrix} -B & 0 & 0 & \dots & 0 \\ 0 & -B & 0 & \dots & 0 \\ 0 & 0 & -B & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & -B \end{bmatrix}}_{A_{e,1u}} \vdots \underbrace{\begin{bmatrix} I & 0 & 0 & \dots & 0 \\ -A & I & 0 & \dots & 0 \\ 0 & -A & I & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & -A & I \end{bmatrix}}_{A_{e,1x}} \vdots \underbrace{\begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & & \dots & \vdots \\ 0 & 0 & & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & \dots & 0 \end{bmatrix}}_{A_{e,1e}} \vdots \underbrace{\begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & & \dots & 0 \\ 0 & 0 & & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & \dots & 0 \end{bmatrix}}_{A_{e,1y}}_{(N.n_x \times n_z)} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \\ x_1 \\ x_2 \\ \vdots \\ x_N \\ e \\ y \end{bmatrix}_{(n_z \times 1)} = \underbrace{\begin{bmatrix} Ax_0 \\ 0_{n_{x\times 1}} \\ 0_{n_{x\times 1}} \\ \vdots \\ \vdots \\ \vdots \\ 0_{n_{x\times 1}} \end{bmatrix}}_{b_{e,1}}$$

So, we have from the first linear equality constraint,

$$\boldsymbol{A_{e,1u}} = \begin{bmatrix} \boldsymbol{-B} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{-B} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{-B} \end{bmatrix} = \boldsymbol{-I_N \otimes B}$$

$$\boldsymbol{A_{e,1x}} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{-A} & \boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{-A} & \boldsymbol{I} & \cdots & \boldsymbol{0} \\ \vdots & & & \ddots & \\ \boldsymbol{0} & \cdots & \cdots & \boldsymbol{-A} & \boldsymbol{I} \end{bmatrix} = \boldsymbol{I_{N.n_x} - (I_{N,-1} \otimes A)}$$

In MATLAB, $I_{N,-1}$ can be written as *diag(ones((N-abs(-1), 1), -1)*.

Similarly,

$$\boldsymbol{A_{e,1e}} = \boldsymbol{0_{(N.n_x \times N.n_y)}}$$

$$\boldsymbol{A_{e,1y}} = \boldsymbol{0_{(N.n_x \times N.n_y)}}$$

and $$\boldsymbol{b_{e,1}} = \begin{bmatrix} \boldsymbol{Ax_0} \\ \boldsymbol{0_{((N-1).n_x \times 1)}} \end{bmatrix}_{(\boldsymbol{N.n_x \times 1})}$$

Now considering the second equality constraint of equation (3.3) we get,

$$y_k = Cx_k$$

$$y_k - Cx_k = 0$$

This equality constraint must also be satisfied over the whole prediction horizon length. Since we consider a prediction horizon from $k = 1$ to $k = N$ we get,

$$\begin{aligned} y_1 - Cx_1 &= 0_{(n_y \times 1)} \quad \text{for } k = 1 \\ y_2 - Cx_2 &= 0_{(n_y \times 1)} \quad \text{for } k = 2 \\ &\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ y_N - Cx_N &= 0_{(n_y \times 1)} \quad \text{for } k = N \end{aligned} \tag{3.26}$$

Arranging the set of equations 3.26 obtained from the second equality constraints into a matrix form we get the second row of A_e in equation (3.24) as,

$$\left[\underbrace{\begin{matrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & 0 \end{matrix}}_{A_{e,2u}} \vdots \underbrace{\begin{matrix} -C & 0 & \cdots & 0 \\ 0 & -C & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & -C \end{matrix}}_{A_{e,2x}} \vdots \underbrace{\begin{matrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & 0 \end{matrix}}_{A_{e,2e}} \vdots \underbrace{\begin{matrix} I & 0 & \cdots & 0 \\ 0 & I & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & I \end{matrix}}_{A_{e,2y}}\right]_{(N.n_y \times n_z)} \begin{bmatrix} u \\ x_1 \\ x_2 \\ \vdots \\ x_N \\ e \\ y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}_{(n_z \times 1)} = \underbrace{\begin{bmatrix} 0_{n_{y\times 1}} \\ 0_{n_{y\times 1}} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 0_{n_{y\times 1}} \end{bmatrix}}_{b_{\in,2}}$$

So we have,

$$\boldsymbol{A_{e,2u}} = \boldsymbol{0_{(N.n_y \times N.n_u)}}$$

$$\boldsymbol{A_{e,2x}} = \begin{bmatrix} \boldsymbol{-C} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{-C} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{-C} \end{bmatrix} = \boldsymbol{-I_N \otimes C}$$

$$\boldsymbol{A_{e,2e}} = \boldsymbol{0_{(N.n_y \times N.n_y)}}$$

$$\boldsymbol{A_{e,2y}} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{I} \end{bmatrix} = \boldsymbol{I_{N.n_y}}$$

$$\boldsymbol{b_{e,2}} = \boldsymbol{0_{(N.n_y \times 1)}}$$

Finally, from the last equality constraint of equation (3.4) we get,

$$e_k = r_k - y_k$$

$e_k + y_k = r_k$ (we are taking r_k to the right because the reference or setpoint is known or defined for the whole prediction horizon length)

This equality constraint must also be satisfied over the whole prediction horizon length. Since we consider a prediction horizon from $k = 1$ to $k = N$ we get,

$$\begin{aligned} e_1 + y_1 &= r_1 \quad \text{for } k = 1 \\ e_2 + y_2 &= r_2 \quad \text{for } k = 2 \\ &\vdots \\ e_N + y_N &= r_N \quad \text{for } k = N \end{aligned} \tag{3.27}$$

Arranging the set of equations 3.27 obtained from the third equality constraints in a matrix form we get the third row of A_e in equation (3.24) as,

$$\underbrace{\left[\begin{array}{c|c|c|c} \underbrace{\begin{matrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & 0 \end{matrix}}_{A_{e,3u}} & \underbrace{\begin{matrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & 0 \end{matrix}}_{A_{e,3x}} & \underbrace{\begin{matrix} I & 0 & \cdots & 0 \\ 0 & I & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & I \end{matrix}}_{A_{e,3e}} & \underbrace{\begin{matrix} I & 0 & \cdots & 0 \\ 0 & I & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & I \end{matrix}}_{A_{e,3y}} \end{array}\right]}_{(N.n_y \times n_z)} \begin{bmatrix} u \\ x \\ e_1 \\ e_2 \\ \vdots \\ e_N \\ y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}_{(n_z \times 1)} = \underbrace{\begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix}}_{b_{e,3}}$$

So we have,

$$\boldsymbol{A_{e,3u}} = \boldsymbol{0_{(N.n_y \times N.n_u)}}$$

$$\boldsymbol{A_{e,3x}} = \boldsymbol{0_{(N.n_y \times N.n_x)}}$$

$$\boldsymbol{A_{e,3e}} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{I} \end{bmatrix} = \boldsymbol{I_N \otimes I_{n_y} = I_{N.n_y}}$$

$$\boldsymbol{A_{e,3y}} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{I} \end{bmatrix} = \boldsymbol{I_{N.n_y}}$$

$$\boldsymbol{b_{e,3}} = \begin{bmatrix} \boldsymbol{r_1} \\ \vdots \\ \boldsymbol{r_N} \end{bmatrix}_{(\boldsymbol{N.n_y \times 1})}$$

Finally, all the three equality constraints can be written in the standard form of $A_e\, z = b_e$ where,

$$A_e = \begin{bmatrix} -I_N \otimes B & I_{N.n_x} - (I_{N,-1} \otimes A) & 0_{N.n_x \times N.n_y} & 0_{N.n_x \times N.n_y} \\ 0_{N.n_y \times N.n_u} & -I_N \otimes C & 0_{N.n_y \times N.n_y} & I_{N.n_y} \\ 0_{N.n_y \times N.n_u} & 0_{N.n_y \times N.n_x} & I_{N.n_y} & I_{N.n_y} \end{bmatrix}$$

and

$$b_e = \begin{bmatrix} b_{e,1} \\ b_{e,2} \\ b_{e,3} \end{bmatrix} = \begin{bmatrix} Ax_0 \\ 0_{(N-1).n_x \times 1} \\ 0_{N.n_y \times 1} \\ r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix}$$

3.4 Handling bounds and inequality constraints:

The standard QP formulation of equations 3.6 also contains the bounds i.e. $z_L \leq z \leq z_H$. Thus, bounds on the unknown variables should also be properly formulated. Since no values to the lower and upper bound are provided in the statement of the LQ control problem of equations (3.1 – 3.4), we will assume that they are unbounded i.e. we assume that they can take any values between $-\infty$ and $+\infty$. Thus for the given LQ optimal control problem, the bounds are,

$$z_L = -\infty \times ones\,(n_z, 1) = -\infty \times 1_{n_z \times 1}$$
$$z_U = +\infty \times ones\,(n_z, 1) = +\infty \times 1_{n_z \times 1}$$

In the statement of the LQ optimal control problem equations (3.1 – 3.4), only the equality constraints are present. Due to the absence of the inequality constraints for this particular problem i.e. when $A_i z \leq b_i$ is not considered or present, we simply represent it as,

$$A_i = [\ ] = empty\ matrix$$
$$b_i = [\ ] = empty\ vector$$

Some comments:

If the control input signals have bounds such that $u_L \leq u_k \leq u_U$ then, the bounds on the unknown vector z should be adjusted or modified. The vector $u_L = [u_L^1, u_L^2, \dots, u_L^{n_u}]$ contains the value of the lower bound for each control inputs. The vector $u_U = [u_U^1, u_U^2, \dots, u_U^{n_u}]$ contains the value of the upper bound for each control inputs. Since for this optimal control problem, the control input signals are the first unknown variable in the vector z, the first $N.n_u$ elements of z_L and z_U should be modified as,

$$z_L = \begin{pmatrix} 1_{N\times 1} \otimes u_L \\ -\infty_{(n_z - N.n_u)\times 1} \end{pmatrix}, \qquad z_H = \begin{pmatrix} 1_{N\times 1} \otimes u_U \\ +\infty_{(n_z - N.n_u)\times 1} \end{pmatrix}$$

The same approach can also be taken to add bounds to the other unknown variables in z.

In the statement of the LQ optimal control problem equations (3.1 – 3.4), only the equality constraints are present. However, in many other control problems, inequality constraints could also be present. In general, the inequality constraints can also be treated in the same way as the equality constraints i.e. by constructing the structured matrices for inequality constraints as was done for the equality constraints.

Finally, we have created all the matrices that are needed to formulate the given LQ optimal control problem into a standard QP (Quadratic optimization) problem. We are now ready to solve this QP. For this course, we will be using the *qpOASES* solver in Simulink. An example of its use is explained in the next section.

The standard QP problem can also be solved using the '*quadprog'* solver in MATLAB for. The syntax for using the *quadprog* solver is,

$$z^* = \text{ quadprog } (H, c, A_i\ , B_i\ , A_e\ \ , b_e\ \ , z_L\ \ , z_U\ , z_O)$$

where z_O is the initial starting point for the optimizer, $z_0 \epsilon \mathbb{R}^{n_z \times 1}$. All the matrices and vectors i.e. $H, c, A_i\ , B_i\ , A_\epsilon\ \ , b_\epsilon\ \ , z_L\ \ , z_H\ , z_O$ that are needed for the '*quadprog'* solver should be known or constructed.

Both the *qpOASES* solver in Simulink and the *'quadprog'* solver in MATLAB will solve the QP problem (the inner details of how the solver function operates is NOT within the scope of this course) and return back the optimal values (the best values) of the variables contained in the unknown vector z. The optimal values of the unknown variables are denoted as z^* where $z^* \epsilon \mathbb{R}^{n_z \times 1}$. For the z variables that we previously chose as unknowns, the structure of the z^* is as follows,

$$z^* = \begin{bmatrix} u_0^* \\ u_1^* \\ \vdots \\ u_{N-1}^* \\ x_1^* \\ \vdots \\ x_N^* \\ e_1^* \\ \vdots \\ e_N^* \\ y_1^* \\ \vdots \\ y_N^* \end{bmatrix}_{(n_z \times 1)}$$

Imp

For MIMO system and with multiple states, for each k^{th} sample, u, x, e and y are vectors (instead of just scalars)

e.g. If there are two control inputs in the system i.e. $n_u = 2$ then,

$u_k^* = \begin{bmatrix} u_k^1 \\ u_k^2 \end{bmatrix}^*$ for $k = 0, 1, \ldots\ldots\ldots\ldots\ldots, N-1$ (whole prediction horizon)

The superscript denotes the two control inputs present in the process respectively.

Another example: If there are three outputs present in the system,
i.e. $n_y = 3$ then,

$y_k^* = \begin{bmatrix} y_k^1 \\ y_k^2 \\ y_k^3 \end{bmatrix}^*$ for $k = 1, 2, \ldots\ldots, N$ (through out the prediction horizon)

Here, the superscript denotes the three outputs present in the system respectively

As an example of a complete structure of z^* for a MIMO system, let us consider that $n_u = 2, n_x = 2, n_y = 2$ and $N = 3$. To save space, let us write the structure of the transpose of z^* where the subscripts denote the time k and the superscript denote the individual inputs, states, error signals and the outputs present in the system. The solution returned by the *qpOASES* solver (also by *quadprog* solver) will have the following structure:

$$(z^*)^T = [u_0^1, u_0^2, u_1^1, u_1^2, u_2^1, u_2^2, x_1^1, x_1^2, x_2^1, x_2^2, x_3^1, x_3^2, e_1^1, e_1^2, e_2^1, e_2^2, e_3^1, e_3^2, y_1^1, y_1^2, y_2^1, y_2^2, y_3^1, y_3^2]^*$$

This means that the variables in z* vector are not arranged in "ready to plot form". You have to first properly extract the needed signals before you can plot them.

To better understand this, let us consider that the prediction horizon $N = 3$. With this horizon length we will have the time steps as $k = 1, 2, 3$. Let us also consider that the process has three outputs $n_y = 3$, two states $n_x = 2$ and three control inputs $n_u = 3$. The structure of the optimal solution will be (again using transpose to save space),

$$(z^*)^T = [u^T, x^T, e^T, y^T]^*$$

where,

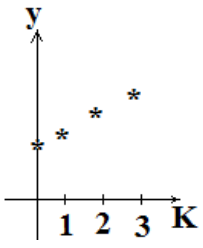
$$u^T = [u_0^1, u_0^2, u_0^3, u_1^1, u_1^2, u_1^3, u_2^1, u_2^2, u_2^3]^*$$

$$x^T = [x_1^1, x_1^2, x_2^1, x_2^2, x_3^1, x_3^2]^*$$

$$e^T = [e_1^1, e_1^2, e_1^3, e_2^1, e_2^2, e_2^3, e_3^1, e_3^2, e_3^3]^*$$

$$y^T = [y_1^1, y_1^2, y_1^3, y_2^1, y_2^2, y_2^3, y_3^1, y_3^2, y_3^3]^*$$

Let us take the output signal as an example here (but the structure is the same for all other elements in the vector z).



$$y = \begin{bmatrix} y_1^1 \\ y_1^2 \\ y_1^3 \\ \dots \\ y_2^1 \\ y_2^2 \\ y_2^3 \\ \dots \\ y_3^1 \\ y_3^2 \\ y_3^3 \end{bmatrix}$$

for the first time step $k = 1$ all the three outputs of the process are present.

for the second time step $k = 2$ all the three outputs of the process are present.

For the third time step $k = 3$ all the three outputs of the process are present.

Same structure also for u, x and e

Now, if we want to plot the first output signal y_k^1 for all the time steps i.e. for $k = 1, 2$ and 3, we should plot y_1^1, y_2^1 and y_3^1. As we can see that these values are not placed one after the other. They are rather placed in specific places in the z^* vector. Thus, we should make sure that we pick them up properly from the z^* vector. The same thing applies also for all the other unknown variables present in the z^* vector.

In MATLAB (and MATLAB function block in Simulink), you can use the function called *"reshape"* to pick up or choose the right signals from the optimal solution. Explore this function yourself and also look at the example!!

Can you do it yourself?

Let us consider the following quadratic performance criteria

$$J = \frac{1}{2}\sum_{k=1}^{N} e_k^T\, Q_k . e_k + \Delta u_{k-1}^T\, P_{k-1} \Delta u_{k-1} + u_{k-1}^T\, R_{k-1} u_{k-1}$$

where,

$$e_k = r_k - y_k$$

$$\Delta u_k = u_k - u_{k-1}$$

For the given process model,

$$x_{k+1} = Ax_k + Bu_k + v_k$$

$$y_k = Cx_k + w_k$$

where,

$$x_k \epsilon \mathbb{R}^{n_x}, u_k \epsilon \mathbb{R}^{n_u}, y_k \epsilon \mathbb{R}^{n_y}, v_k \epsilon \mathbb{R}^{n_x}, w_k \epsilon \mathbb{R}^{n_y}$$

v_k and w_k are the process noise and the measurement noise respectively and it is assumed that they are known or can be modeled.

Express the above in the standard QP from

$$J = \frac{1}{2} z^T H z + c^T z$$

$$A_\epsilon z = b_\epsilon$$

Where,

$$z^T = (u^T, \Delta u^T, x^T, e^T, y^T)$$

3.5 Example: LQ optimal control of inverted pendulum

In this example, we will further work with the inverted pendulum system (as described in lecture 1). If we recall, the inverted pendulum system has 4 states: the angle(α), angular velocity(ω), position of cart(x_2) and velocity of cart(v_2). A mechanistic model of the inverted pendulum system is given by the following set of nonlinear ODEs.

$$\frac{d\alpha}{dt} = \omega$$

$$\frac{d\omega}{dt} = \frac{m_1+m_2}{m_1^2 l^2 cos^2\alpha - m_1^2 l^2 - m_1 m_2 l^2}(k_T l^2 |\omega| \omega - m_1 g l\, sin\alpha) + \frac{cos\alpha}{m_1 l cos^2\alpha - m_1 l - m_2 l}(F + \omega^2 m_1 l\, sin\alpha)$$

$$\frac{dx_2}{dt} = v_2$$

$$\frac{dv_2}{dt} = \frac{1}{m_1 l\, cos\alpha}\left(m_1 g l\, sin\alpha - k_T l^2 |\omega|\omega - m_1 l^2 \frac{d\omega}{dt}\right)$$

The parameters of the system are: $m_1 = 1$ kg, $m_2 = 2$ kg, $l = 1$ m, $k_T = 0.1$ kg/rad^2.

We will control the angle of the inverted pendulum so that it remains inverted (i.e. it stands upright). At the same time we will also control the position of the cart. In order to control α and x_2, the force acting on the cart(F) can be adjusted/manipulated. Thus the control input of the system is F.

Since, we will be designing an LQ optimal controller, we need a linear model of the inverted pendulum system. The nonlinear ODEs can be linearized around the equilibrium point ($\alpha_s = 0, w_s = 0, x_{2s} = 0$ and $v_{2s} = 0$). We will then obtain a continuous time linear model of the system as,

$$\frac{dx}{dt} = A_c x(t) + B_c u(t) \tag{3.28}$$

$$y = C_c x(t) \tag{3.29}$$

where,

$$x = \begin{bmatrix} \alpha \\ \omega \\ x_2 \\ v_2 \end{bmatrix}, A_c = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 14.715 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -4.905 & 0 & 0 & 0 \end{bmatrix}, B_c = \begin{bmatrix} 0 \\ -0.5 \\ 0 \\ 0.5 \end{bmatrix}, C_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Note: Since the measurements (α and x_2) are directly the states of the system, measurement matrix C_c is obvious to construct.

For LQ optimal control, we need the discrete time linear model of the system. The continuous time model of equations (3.28) and (3.29) can be discretized (for e.g. in MATLAB) with a chosen sampling time dt as,

```
% Discrete time model
dt = 0.1; %sampling time
sys = ss(Ac,Bc,Cc,0); %there is no D matrix, so set it as 0
ds = c2d(sys,dt);
A = ds.a; B = ds.b; C = ds.c; %discrete time system matrices
```

Precaution and information:

In Simulink, the function *ss* is not supported for code generation (or compilation) as shown in Figure 3.1. So you cannot use the above code snippet with MATLAB function block in Simulink. The workaround here is to use the above code snippet with MATLAB alone, execute

it and then copy (or use) the **discrete** *A, B, C, D* matrices in MATLAB function block in Simulink.

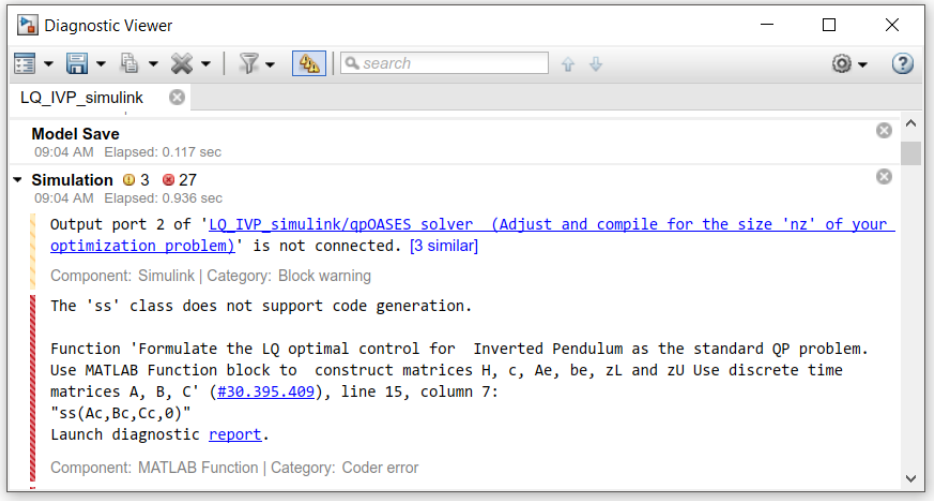


Figure 3.1: *ss* function does not support code generation in Simulink

We then have the discrete time model as,

$$x_{k+1} = Ax_k + Bu_k \tag{3.30}$$

$$y_k = Cx_k \tag{3.31}$$

where

$$x = \begin{bmatrix} \alpha \\ \omega \\ x_2 \\ v_2 \end{bmatrix}, A = \begin{bmatrix} 1.0745 & 0.1025 & 0 & 0 \\ 1.5079 & 1.0745 & 0 & 0 \\ -0.0248 & -0.0008 & 1 & 0.1 \\ -0.5026 & -0.0248 & 0 & 1 \end{bmatrix}, B_c = \begin{bmatrix} -0.0025 \\ -0.0512 \\ 0.0025 \\ 0.0504 \end{bmatrix}, C_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Problem Formulation:

Design an LQ optimal controller that tracks the angle of the pendulum (α) and the position of the cart (x_2) to their set points.

As a scenario for the simulation, let us simulate the system for 8 seconds with a sampling time of 0.1 sec. Then the length of the horizon will be $N = \frac{8}{0.1} = 80$ samples. Let us define the set point for the angle to be 0 radian throughout the horizon. For the position of the cart, let us define the set point to be:

0 for $N \leq 40$ i.e. $t \leq 4$ seconds

1 for $N > 40$ i.e. $t > 4$ seconds

In a MATLAB function block in Simulink, it can be done as,

```
N = 80;  % prediction horizon length
% set up the references for the whole prediction horizon length
r = [zeros(1,N);
    zeros(1,N/2) ones(1,N/2)];
```

The LQ optimal control problem can be formulated as,

$$min \quad J = \frac{1}{2}\sum_{k=1}^{N} e_k^T\, Q_k e_k + u_{k-1}^T\, P_{k-1} u_{k-1} \tag{3.32}$$

subject to,

$$x_{k+1} = Ax_k + Bu_k \qquad \text{with } x\epsilon\mathbb{R}^{n_x\times 1}, u\epsilon\mathbb{R}^{n_u\times 1} \tag{3.33}$$

$$y_k = Cx_k \qquad \text{with y}\epsilon\mathbb{R}^{n_y\times 1} \text{ and } x_0 \text{ is given/known} \tag{3.34}$$

$$e_k = r_k - y_k \qquad \text{with } r_k \text{ being the set points} \tag{3.35}$$

Here, n_x= no. of states = 4, n_u = no. of control inputs=1, n_y = no. of outputs=2, Q_k and P_k are the weighting matrices

Problem Solution:

To solve the LQ optimal control problem given by equations (3.32 – 3.35), we first need to express the problem in a standard QP form as,

$$\min_{z} \quad \tfrac{1}{2}\, z^T Hz + c^T z \tag{3.36}$$

subject to,

$$\left.\begin{array}{ll} A_e z = b_e & \rightarrow \text{equality constraint} \\ A_i z \leq b_i & \rightarrow \text{inequality constraint} \\ z_L \leq z \leq z_U & \rightarrow \text{bounds} \end{array}\right\} \tag{3.37}$$

Here we are free to choose the vector of unknowns z. Let us define it to be $z = [u, x, e, y]^T$.

This problem is similar to what we learnt in lecture 3. Following the same procedure we can construct the structured matrices $H, c, A_e, b_\epsilon, A_i, b_i, z_L, z_U$. They will be the same as discussed in lecture 3 (thus not shown here but note that they may be different for different formulations and different choices of the vector z).

In Simulink, we can make use of the *qpOASES* solver to solve the LQ optimal control problem for the inverted pendulum. A suggested block diagram for implementing it in Simulink is shown in Figure 3.2. Please note that here we use "Array plot" for plotting and not "scope" in Simulink. The "scope" can only plot variables vs. time. Here we need to plot array of numbers, thus we use "Array Plot".

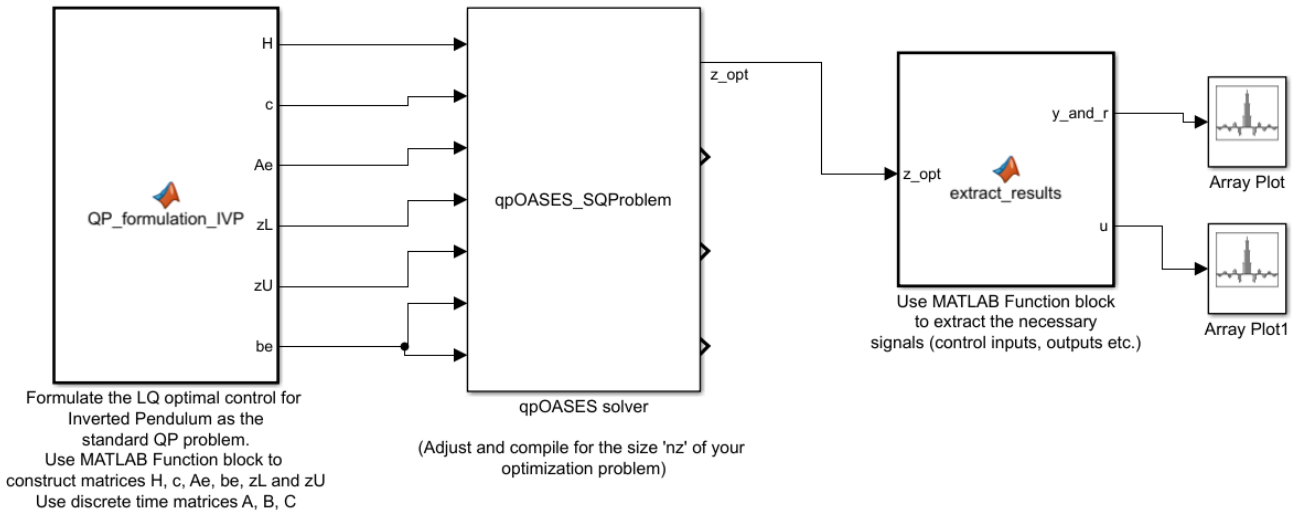


Figure 3.2: Implementation of LQ optimal control for inverted pendulum in Simulink

In the first block (QP_formulation_IVP), you can use the MATLAB function block in Simulink to construct matrices needed for standard QP formulation such as H, c, Ae, be, zL, zU. To do so you should use the discrete time matrices A, B, C.

A code snippet of the first MATLAB function block in Simulink is shown below:

```
function [H,c,Ae,zL,zU,be] = QP_formulation_IVP()
%discrete time model
A = [1.0745,0.1025,0,0; 1.5079,1.0745,0,0;-0.0248,-0.0008,1,0.1;-0.5026,
     -0.0248,0,1];
B = [-0.0025;-0.0512;0.0025;0.0504];
C = [1,0,0,0;0,0,1,0];


N = 80;  % prediction horizon length
% set up the references for the whole prediction horizon length
r = [zeros(1,N); % first row for first output (pendulum angle)
    zeros(1,N/2) ones(1,N/2)]; % second row for second output (cart position)

%initial values of the states should be known or estimated
x0 = [0; 0; 0; 0];

%size of matrices
nx = 4; ny = 2; nu = 1;

%size of the unknow vector z
nz = N*(nx + nu + 2*ny);

%weighting matrices
Q=diag([1.0 1.0]);      %tuning weight for error: there are 2
P=1e-3;                 %tuning weight for inputs: there is 1

% build matrices
H11 = kron(eye(N),P);
H22 = zeros(N*nx,N*nx);
H33 = kron(eye(N),Q);
```

```
H44 = zeros(N*ny,N*ny);
H_mat = blkdiag(H11,H22,H33,H44);
%qpOASES does not accept matrices, but only vectors
%we have to change H matrix to vector by stacking elements column wise
H = H_mat(:);
c = zeros(nz,1);

%constraints (from process model)
% from eq 3.33: state equation
Ae1u = -kron(eye(N),B);
Ae1x = eye(N*nx)-kron(diag(ones(N-abs(-1),1),-1),A);
Ae1e = zeros(N*nx,N*ny);
Ae1y = zeros(N*nx,N*ny);
be1 = [A*x0;zeros((N-1)*nx,1)];

%from eq 3.34: measurement equation
Ae2u = zeros(N*ny,N*nu);
Ae2x = -kron(eye(N),C);
Ae2e = zeros(N*ny,N*ny);
Ae2y = eye(N*ny);
be2 = zeros(N*ny,1);

%from eq 3.35: error equation
Ae3u = zeros(N*ny,N*nu);
Ae3x = zeros(N*ny,N*nx);
Ae3e = eye(N*ny);
Ae3y = eye(N*ny);
% since be3 contains the reference vector in specific order, we have to use
% the function "reshape" to put the reference values in the right order
be3 = reshape(r,N*ny,1);

Ae_mat=[Ae1u Ae1x Ae1e Ae1y;
    Ae2u Ae2x Ae2e Ae2y;
    Ae3u Ae3x Ae3e Ae3y];
%qpOASES does not accept matrices, but only vectors
%we have to change Ae matrix to vector by stacking elements column wise
Ae = Ae_mat(:); %stacking column wise


% make the standard be vector
be=[be1;be2;be3];

%bounds (not specified so assume between -inf to +inf)
ZL=(-Inf*ones(nz,1));
ZU=(Inf*ones(nz,1));
```

Some notes:

The *qpOASES* solver does not accept matrices but it can accept only vectors. Since H is a matrix, it should be converted to a vector by stacking the elements of the matrix column wise. The code snippet to be used in the first MATLAB function block in Simulink is

```
H_mat = blkdiag(H11,H22,H33,H44);
%qpOASES does not accept matrices, but only vectors
%we have to change H matrix to vector by stacking elements column wise
H = H_mat(:);
```

Similarly, A_e is a matrix, so it should be converted to a vector by stacking the elements of the matrix column wise. The code snippet to be used in the first MATLAB function block in Simulink is

```
Ae_mat=[Ae1u Ae1x Ae1e Ae1y;
    Ae2u Ae2x Ae2e Ae2y;
    Ae3u Ae3x Ae3e Ae3y];
%qpOASES does not accept matrices, but only vectors
%we have to change Ae matrix to vector by stacking elements column wise
Ae = Ae_mat(:); %stacking column wise
```

After defining and constructing the matrices in the first MATLAB function block in Simulink, you have to return back the matrices H, c, A_e, b_e, zL, zU. Then we are ready to solve the QP problem.

In the second block (qpOASES_SQProblem), you have to use the *qpOASES* solver (the compiled mex file) to solve the standard QP problem. Please make sure to compile the *qpOASES* solver such that it fits the size of your problem given by n_z. For this you should modify the .cpp file in the *qpOASES* installation folder. Details on how to do so was also discussed in lecture 2 when the oil refinery QP problem was solved in Simulink (look back the videos if necessary).

In the third block (extract_results), the optimal values of the unknowns in vector z, here donoted by *z_opt* is taken as the input in a MATLAB function block in Simulink. Since the *z_opt* vector does not contain data in a "ready to plot" order, we have to extract the results for the process outputs and control inputs. Then these extracted results can be plotted using *Array Plot* in Simulink. In addition, it is also useful to plot the reference line or the setpoints along with the process outputs (therefor we have `y_and_r`). The code snippet to do so is,

```
function [y_and_r,u] = extract_results(z_opt)
N = 80; %prediction horizon length
nx = 4; nu = 1; ny = 2; %no. of states, inputs and outputs
%extract results
Ua = z_opt(1+N*(0)          :N*(nu),1); %control inputs
Xa = z_opt(1+N*(nu)         :N*(nu+nx),:); %states
Ea = z_opt(1+N*(nu+nx)      :N*(nu+nx+ny),:); %error in tracking
Ya = z_opt(1+N*(nu+nx+ny)   :N*(nu+nx+ny+ny),:); %outputs

%we use the reshape function to rearrange the data
y_temp = reshape(Ya,ny,N);%arranged outputs(rows as signals, columns as data)

%the array plot in Simulink accepts signals in a format such that the columns
%represent the different signals and the rows represent the data for each
signal. So we need to transpose it.
%The first column is for output y1 (angle), second column for output y2
%(cart postion)
y = y_temp';
%Let us also plot the reference signals along with the output signals for
%better viewing.
r = [zeros(N,1),[zeros(N/2,1);ones(N/2,1)]];
```

```
%Put it as the third column (for r1) and fourth column (for r2).
y_and_r = [y,r(:,1),r(:,2)];

u_temp = reshape(Ua,nu,N); %arranged input(row as signals, columns as data)
u = u_temp';
```

You should now configure the simulation parameters. Click in the *model configuration parameter* (click gear like symbol in Simulink) as shown in Figure 3.3(a). Then it opens up configuration parameter window. You can choose 0.0 seconds for Start time and 0.1 second for Stop time. In solver options type, choose "Fixed-step". Choose ode4 (Runge-Kutta) as "Solver". Choose 0.1 as the fixed-step size under "additional options". See Figure 3.3(b) for the values to be chosen.

Note: Here for LQ optimal control, we choose Stop time = fixed-step size because we only need to solve the optimal control problem once. Later on in this course, you will learn that for MPC, your Stop time can be much larger, since with MPC we have to re-solve the optimal control problem at each time step.

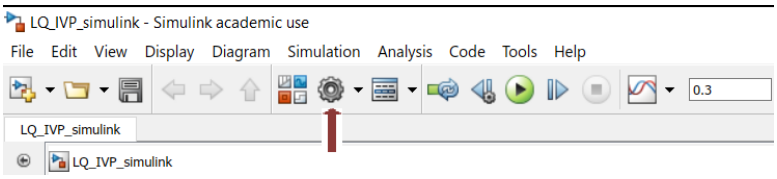


Figure 3.3 (a): Model configuration parameter icon

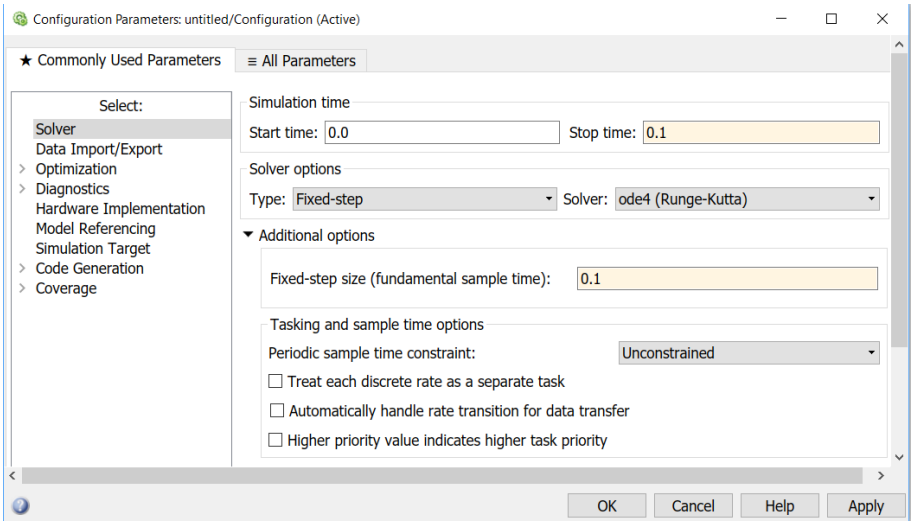


Figure 3.3 (b): Configuration of the simulation parameters

Click Apply and then OK to close the configuration parameter window. Then you can go ahead and click the run button in Simulink (on top middle) to start the simulation. The LQ optimal control of the inverted pendulum example is shown below in Figure 3.4.

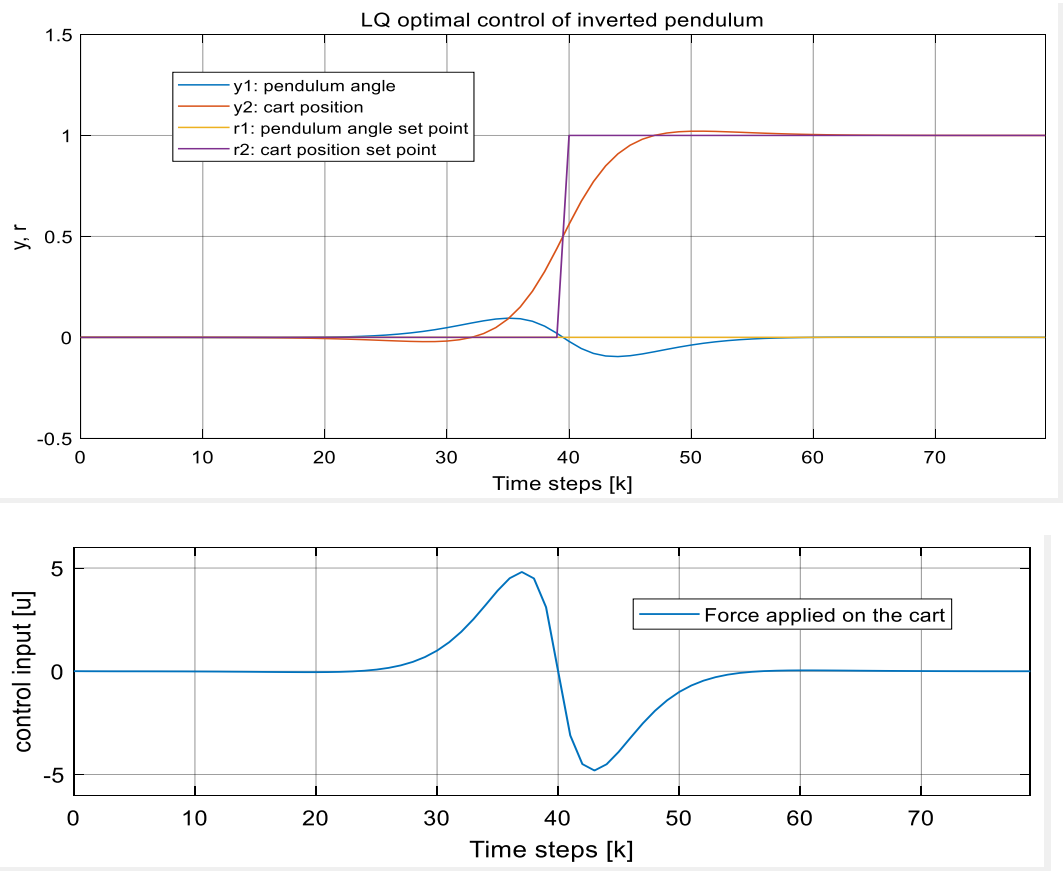


Figure 3.4: LQ optimal control of inverted pendulum in Simulink

In MATLAB (without Simulink) we can make use of the *quadprog* solver. The code snippet for solving LQ optimal control problem for inverted pendulum in MATLAB (is almost the same as for Simulink) is

```
%clear
clc,
clear all

%Continuous time model
Ac = [0 1 0 0; 14.715 0 0 0; 0 0 0 1;-4.905 0 0 0];
Bc = [0;-0.5;0;0.5];
Cc = [1 0 0 0;0 0 1 0];

dt = 0.1; %sampling time

%change to discrete time model
sys = ss(Ac,Bc,Cc,0); %there is no D matrix, so set it as 0
ds = c2d(sys,dt);
A = ds.a; B = ds.b; C = ds.c; D = ds.d;

N = 80;  % prediction horizon length
% set up the references for the whole prediction horizon length
r = [zeros(1,N); % first row for first output (pendulum angle)
    zeros(1,N/2) ones(1,N/2)]; % second row for second output (cart position)
```

```
%initial values of the states should be known or estimated
x0 = [0; 0; 0; 0];

%size of matrices
nx = 4; ny = 2; nu = 1;

%size of the unknow vector z
nz = N*(nx + nu + 2*ny);

%weighting matrices
Q=diag([1.0 1.0]);      %tuning weight for error: there are 2
P=1e-3;                 %tuning weight for inputs: there is 1

% build matrices
H11 = kron(eye(N),P);
H22 = zeros(N*nx,N*nx);
H33 = kron(eye(N),Q);
H44 = zeros(N*ny,N*ny);
H   = blkdiag(H11,H22,H33,H44);
c = zeros(nz,1);

%constraints (from process model)
% from eq 3.33: state equation
Ae1u = -kron(eye(N),B);
Ae1x = eye(N*nx)-kron(diag(ones(N-abs(-1),1),-1),A);
Ae1e = zeros(N*nx,N*ny);
Ae1y = zeros(N*nx,N*ny);
be1 = [A*x0;zeros((N-1)*nx,1)];

%from eq 3.34: measurement equation
Ae2u = zeros(N*ny,N*nu);
Ae2x = -kron(eye(N),C);
Ae2e = zeros(N*ny,N*ny);
Ae2y = eye(N*ny);
be2 = zeros(N*ny,1);

%from eq 3.35: error equation
Ae3u = zeros(N*ny,N*nu);
Ae3x = zeros(N*ny,N*nx);
Ae3e = eye(N*ny);
Ae3y = eye(N*ny);
% since be3 contains the reference vector in specific order, we have to use
% the function "reshape" to put the reference values in the right order
be3 = reshape(r,N*ny,1);

Ae =[Ae1u Ae1x Ae1e Ae1y;
    Ae2u Ae2x Ae2e Ae2y;
    Ae3u Ae3x Ae3e Ae3y];

% make the standard be vector
be=[be1;be2;be3];

%bounds (not specified so assume between -inf to +inf)
ZL=(-Inf*ones(nz,1));
ZU=(Inf*ones(nz,1));

% run quadprog to solve the QP problem
```

```
Z_opt = quadprog(H,c,[],[],Ae,be,Zl,Zh);
```

The solver calculates the optimal values of the unknowns of vector z (which also includes the control inputs). But as also discussed earlier, the calculated optimal values are of the vector z contains data which are not in "ready to plot" format. Thus we first need to extract data/results.

A code snippet for extracting data in MALTAB is shown below. Assuming that the optimal values are returned in the variable *z_opt,*

```
%% extract results
Ua = z_opt(1+N*(0)            :N*(nu),1); %control inputs
Xa = z_opt (1+N*(nu)          :N*(nu+nx),:); %states
Ea = z_opt (1+N*(nu+nx)       :N*(nu+nx+ny),:); %error in tracking
Ya = z_opt (1+N*(nu+nx+ny)    :N*(nu+nx+ny+ny),:); %outputs

%we use the reshape function to rearrange the data
u_opt = reshape(Ua,nu,N); %arranged control inputs
x_opt = reshape(Xa,nx,N); %arranged states
e_opt = reshape(Ea,ny,N); %arranged e
y_opt = reshape(Ya,ny,N); %arranged outputs
```

Finally we are ready to plot the results. However, first we need to define the x-axis in terms of time steps. We do this as,

```
%make x-axis
k = linspace(0,N-1,N);
```

The plots of the LQ optimal control are then obtained as,

```
% plot the figures,
figure,
subplot(211)
plot(k,r(1,:),'b-',k,r(2,:),'r-',k,y_opt(1,:),'g-',k,y_opt(2,:),'k-')
legend('r1: pendulum angle set point','r2: cart position set
point','y1:pendulum angle','y1: cart position')
ylabel('y, r'); title('LQ optimal control of Inverted Pendulum');

subplot(212)
plot(k,u_opt(1,:),'r-')
xlabel('time steps [k]'); ylabel('u [N]');
legend('u: Force applied on the cart');
```

The plots should look exactly like the one shown in Figure 3.4. Please check it out yourself.

Q) Can you invert the pendulum from its hanging position (i.e. initial angle $= \pi$ rad) to the upright position ($\alpha = 0\ rad$)?

Lecture 4

Receding horizon strategy for Model Predictive control

4.1 Recap

In the precious lectures, you learned about optimal control (for linear case). The optimal control problem was posed as dynamic optimization problem with a selected prediction horizon. The dynamic optimization problem was then solved using appropriate solvers and you computed the optimal control signal for the whole prediction horizon length of N time steps i.e. you calculated $u_0^*, u_1^*, \dots\dots\dots\dots\dots\dots, u_{N-1}^*$ and ***applied all of them*** to obtain the desired response.

You did the calculation (solved the optimal control problem) at the current time (say at $t = 0$) to compute the optimal control signals. The N number of optimal control moves/actions which were calculated at $(t = 0)$ were applied (all of them) to control the process up to N time steps ahead.

The optimal control sequence $(u_0^*, u_1^*, \dots\dots\dots\dots\dots\dots, u_{N-1}^*)$ was calculated by having a prior knowledge of the initial state (x_0) of the process, future reference values $(r_1, r_2, \dots\dots\dots\dots r_N)$ and future disturbance value $(w_0, w_1, \dots\dots\dots, w_{N-1})$. It also means that for N time steps ahead in the future, you already know what should be the desired reference and disturbance acting on the system.

4.2 The problem

So far so good. But there are various comments that needs to be considered.

a) Optimal control problem formulated as dynamic optimization problem is a *one time* optimization problem (one time in a sense that, the optimization is performed only once (at $t = 0$) and N number of optimal values of the control inputs are obtained and applied for $t = 0$ to $t = N - 1$). In other words, it is like an openloop solution. The disadvantage is that if any changes in the input disturbances or reference values occur as the system marches forward in time (by applying the N control moves), they are not updated to the controller. Therefore, if the input disturbances and references change after (say a few time steps), the controller has no idea or knowledge about this change since no re-computation is performed. Due to such changes, the controller, that is still using the optimal control inputs obtained from solving the control problem before disturbance occurred, may fail to produce the required performance.

   Fact: In real world systems/applications, disturbances are always present in the system and they may affect the system at any time.

b) Optimal control problems formed in the previous chapters as optimization problems use a finite horizon(N) i.e. we calculate a fixed number of optimal control actions that are used for N number of time steps. In reality, industrial processes run continuously

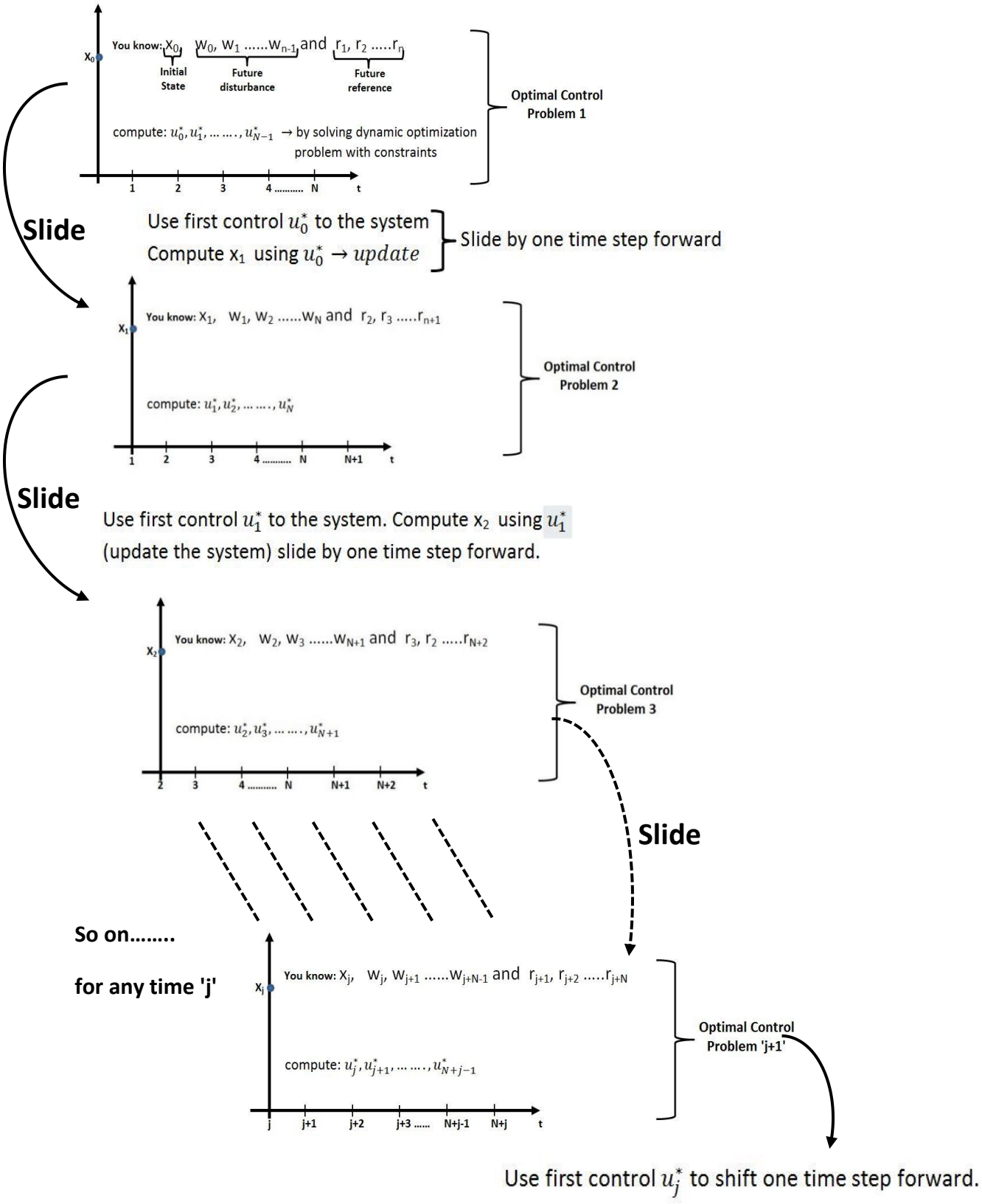


Figure 4.1: Receeding horizon or sliding horizon strategy

in time and this N number of time steps will easily be surpassed. With a larger prediction horizon, the size of the optimal control problem also becomes larger. With infinite prediction horizon, the size of the optimal control becomes infinite. If there were no disturbances and no model-plant mismatch, and if the optimization problem could be solved over an infinite horizon, the input signal found at $t = 0$ could be open-loop applied to the process for all $t \geq 0$. But in reality, disturbances and model-plant mismatch are always present due to which the actual system behavior is different from the predicted one. And also remember that mathematical model are simply representation of the real process, and usually they cannot represent the system fully or completely.

4.3 Receding horizon strategy

The solution to the above problem is that we have to introduce feedback to the open loop optimal controller. How?: By using the optimal open-loop input only until the next sampling instant and then re-computing/re-optimizing the optimal solution at each time step as the process marches forward in time. The idea is to use the receding horizon strategy (also known as sliding horizon strategy) to introduce feedback as shown in Figure 6.1. An optimal controller with the receding horizon strategy is the Model predictive Controller (MPC).
MPC also has other names such as moving horizon optimal control or receding horizon optimal control. If the optimal control problem is linear quadratic (LQ) where the objective function is quadratic and all the constraints are liner, with the sliding horizon strategy included it is called a linear MPC. If the optimal control problem is nonlinear where either objective and/or constraints are nonlinear function of the decision variables, with the sliding horizon strategy included it is called a nonlinear MPC.
The sliding horizon strategy is the same for both the linear as well as nonlinear MPCs.

4.3.1 How does sliding horizon strategy introduce feedback?

⟹With a sliding horizon strategy, a new optimal control problem is solved at every time step. For example, at any given time $t = 0$, an optimal control problem (say optimal control problem 1 in Figure 4.1) is solved to obtain N number of optimal control actions $u_0^*, u_1^*, \ldots\ldots\ldots\ldots\ldots\ldots, u_{N-1}^*$. But instead of applying all the N number of control actions for the whole prediction horizon length, **only the first control action $\boldsymbol{u_0^*}$ is applied** to slide or shift one time step forward. Remember for processes having more than one inputs, $\boldsymbol{u_0^*}$ will be a vector. Now let us assume that after you have slided one-step forward to $t = 1$, some changes had to be made to the reference values (say process operation had to be changed) or due to some disturbances acting on the system, the dynamic behavior of the process changed. Before you can take any further step forward in time, a completely new optimal control problem (say optimal control problem 2 in Figure 4.1) is created at $t = 1$. This new control problem will incorporate these changes in the reference values (and/or incorporate the new predicted plant behavior due to occurrence of disturbances) when the optimal control problem is re-created. With this new optimal control problem, the controller will calculate a new set of

optimal control inputs by solving the updated/new optimal control problem i.e. a new set of N number of optimal control actions $u_1^*, u_2^*, \ldots\ldots\ldots\ldots\ldots\ldots, u_N^*$ are obtained. Once again, only the first control input u_1^* is applied to the process to slide/shift one time step forward.
This process (of re-computing the optimal solution of a new control problem) is repeated at each time step. When a new optimal control problem is formulated at each time step, the most recent (updated) values of the states are used as the initial values (denoted by x_0, $x_1, \ldots, x_j$ … in Figure 4.1) to predict the plant future behavior. If the occurrence of any disturbance into the system has changed this "most recent/updated" states, it is accounted for when formulating a new optimal control problem and the controller will re-generate a new set of optimal control move to handle it. This introduces feedback to the optimal controller.

4.3.2 How does sliding horizon strategy introduce feedforward?

In forming and solving an optimal control problem at any time, say at $t = j$, future reference values $r_{j+1}, r_{j+2}, \ldots\ldots\ldots r_{j+N}$ and future known input disturbance values $w_j, w_{j+1}, \ldots\ldots\ldots w_{j+N-1}$ are included. This introduces feed forward action to the optimal controller i.e. the controller has a prior information of what these variables are going to be in the future. The controller then can appropriately react before a change in the input disturbance (or set point changes) affect the process output variable.

Note: You should have prior knowledge about the future references and disturbance values. It is recommended that you calculate all the value of the future known input disturbances or references offline and then use them while solving the optimization problem.

4.4 State feedback MPC

In the figure 4.1 you can see that in order to solve an optimal control problem at any time step j, we need to know the initial value of the state of the process x_j at the current time. The initial (or current) value of the state at each time step should be known in order to predict how the process behaves in the future (within the prediction horizon) by taking this initial value of state as the starting point for prediction.
A model based predictive control where we assume that all the states are exactly measured (or that the full state is available) at each time step is also known as state feedback MPC. A simple algorithm of a state feedback MPC is given below:

1. Start with a given initial values of the state of the process $x_j = x_0$ and set $t = j = 0$.
2. Considering a prediction horizon of N samples. Solve the optimal control problem (dynamic constrained optimization with desired objective function) over the prediction horizon from $t = j$ to $t = j + N$. Use x_j (full state) as the current states of the process and the system model for prediction of the future states. Compute $u_j^*, u_{j+1}^*, \ldots\ldots\ldots\ldots\ldots\ldots, u_{j+N-1}^*$ by solving this openloop optimization problem.

3. Use only the first control move u_j^* until the next sampling instant and discard the others i.e. compute the updated state of the system x_{j+1}^* for $t = j + 1$ by using the control input u_j^* (update the system using u_j^*).
4. Set $x_j = x_{j+1}^*$ and slide one time step forward to $t = j + 1$
5. Repeat steps (2) to (5) until the program terminates.

4.5 Warm start

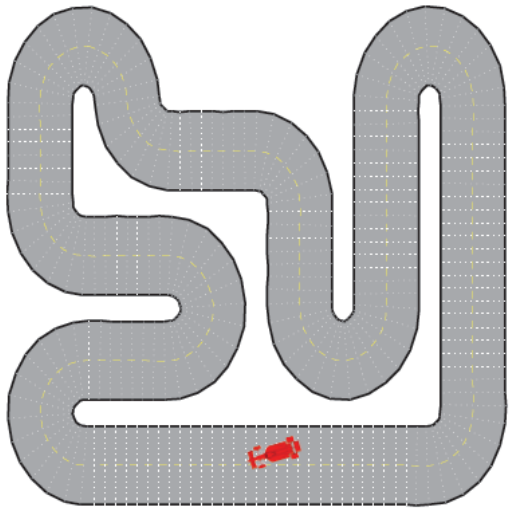
To solve an optimization problem, you need to tell the optimizer where to start looking for the optimal solution. i.e. initial starting point should be provided to the solver. Assuming that the control inputs are the variables to optimize, you should at first choose good starting values of the control inputs while solving "optimal control problem 1" (refer to Figure 4.1). But when solving "optimal control problem 2" at the next time step, you can wisely use $u_0^*, u_1^*, \ldots\ldots\ldots\ldots\ldots\ldots, u_{N-1}^*$ (the optimal results of the optimal control problem 1) as the initial starting points for solving "optimal control problem 2". For solving "optimal control problem 3" you can use $u_1^*, u_2^*, \ldots\ldots\ldots\ldots\ldots\ldots, u_N^*$ as initial values for optimizer & so on for others. This is called warm start.

You could have randomly chosen the starting point for each optimal control problem for each time step. But with warm start, the optimizer finds the solution to each optimal control problem probably more faster in less number of iterations.

4.6 Example to illustrate the main idea of MPC

This interesting example has been from Institu für Automatik. The example is about a racing car.

- **Objective:**
  - Minimize lap time
- **Constraints:**
  - Avoid other cars
  - Stay on road
  - Don't skid
  - Limited acceleration
- Approach: (as a driver/controller)
  - Look forward (with a prediction horizon) and plan the driving path based on:
    - Road Conditions
    - Upcoming corners
    - Ability of car
- Requirement for automatic controller: You need a model of the process to look forward throughout the horizon.

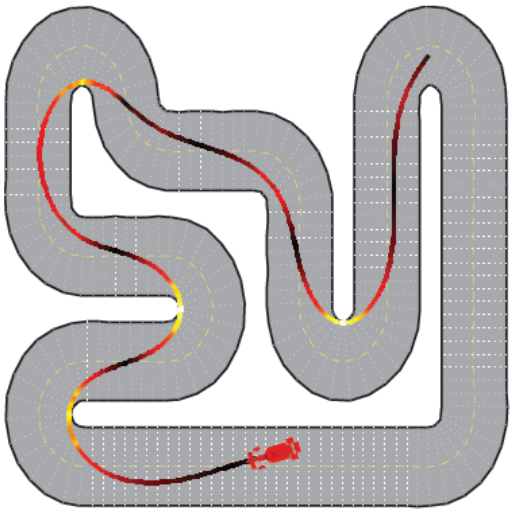


- **Back bone is OPTIMIZATION** (form an optimization problem to)
  - Minimize lap time

    while satisfying the constraints.
- Solve the optimization problem to:
  - Find minimum-time path
  - Find a collection of control actions to be taken over the horizon.

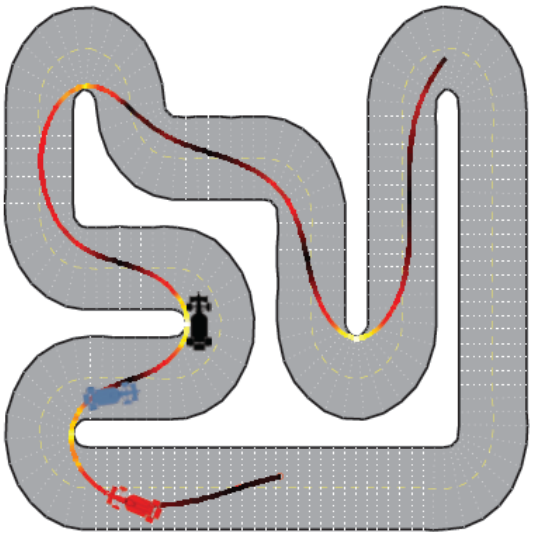
- Collection of control action is denoted by the coloured line.

- Yes, you could have used all the control actions for the whole prediction horizon.



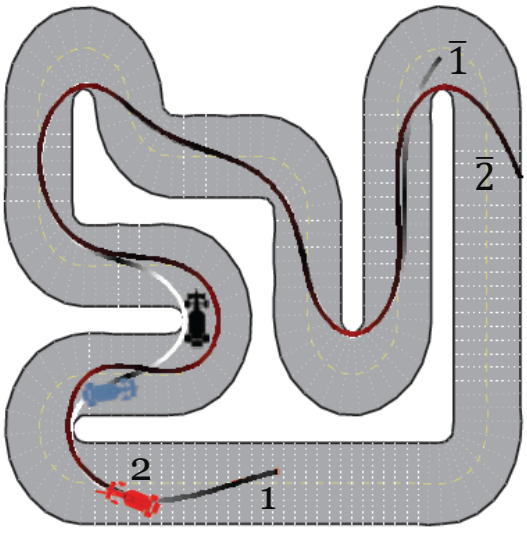
BUT…….

- **When there are unexpected disturbances and unknown model errors….**
  - E.g. another car suddenly drove inside your planned path after you started using the optimal control moves
  - didn't see cars around the corner etc.
- **What not to do?**
  - Don't be stupid to hit the other cars
  - Don't follow all the control actions of the horizon
- **Then what to do?**
  - Apply only the first control action
  - Introduce feedback



- Slide or move one time step ahead (from position 1 to position 2)
  - Use only the first control input to slide.
- Repeat the planning procedure:
- From position 2, form and solve a new optimal control problem with an updated information about the disturbances (cars at the corners etc.)
- The prediction horizon will slide from $1-\bar{1}$ to $2-\bar{2}$ at position 2.
- Again at position 2, use only the first control input to slide one step further.

- Repeat this until the racing is over.



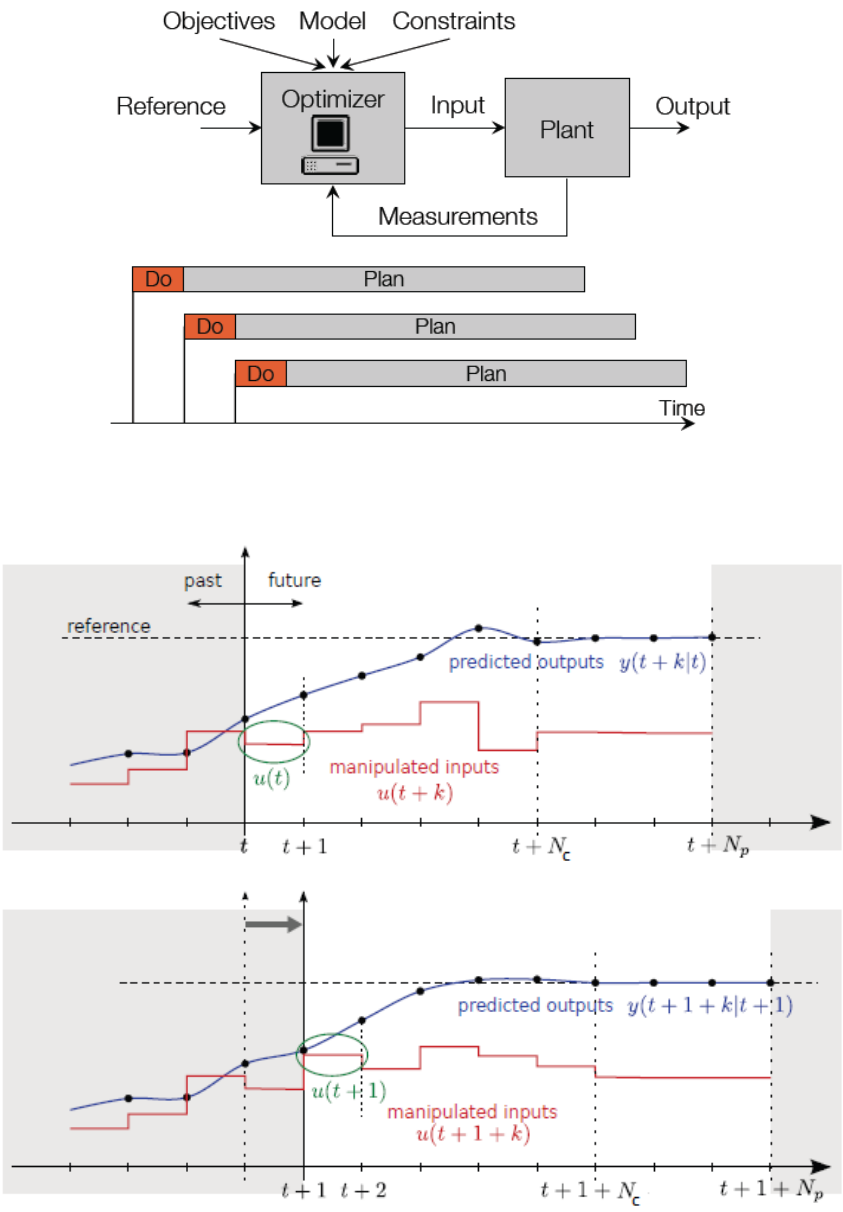


Figure 4.2: Receding horizon strategy for MPC

In an MPC, you need a model of the plant. You create an optimal control problem with objectives and constraints (linear or nonlinear) and solve it using an optimization solver. The optimal value of the first control action is fed to the plant for controlling it. Then, receding or sliding horizon strategy is applied as shown in Figure 4.2. If the states need to be estimated, the latest measurements from the plant are used.

Note: There is also another strategy called the *shrinking horizon strategy* which shrinks the prediction horizon at each time step instead of sliding the horizon. This strategy is useful for e.g. for batch processes where it doesn't make any sense to let the horizon extend beyond the end of the batch operation. If the optimization horizon is equal to the batch time, then we shrink the horizon by one time unit while solving the new optimal control problem at the new shifted time.

In the next section, an example of nonlinear MPC is given, but the principle of making the optimal control problem into a predictive control (by using sliding horizon strategy) also applies for a linear MPC. For linear MPC, the LQ optimal control (with quadratic objective and linear constraints) should be used with sliding horizon strategy.

4.7 Example of a linear model predictive control in simulink

Let us again consider the example of inverted pendulum. In the previous lectures, we have already formulated the LQ optimal controller to the inverted pendulum system. In this example, we will be using the sliding horizon strategy and convert the LQ optimal control problem to linear MPC. Formulating and solving the optimal control problem takes the biggest part in making a linear MPC. Applying receeding horizon strategy is relatively simpler. The control problem is to keep the pendulum in inverted position (angle $\alpha = 0$) and change the position of the cart while still keeping the pendulum in upright position. Simulink will be used as the platform for designing and implementing linear MPC. The MPC will be designed to be run in real time. We will make use of the "knobs" and/or "slider" in Simulink to change the set points in real time.

At first, for simplicity, let us re-write (from previous chapters) the LQ optimal control problem formulation for the inverted pendulum. Linear model of the inverted pendulum in deviation form is used (linearization part is skipped here, see details in the previous chapters). The LQ optimal control problem can be formulated as (see detail in the previous chapter),

$$min\ J = \frac{1}{2}\sum_{k=1}^{N} \delta e_k^T\, Q_k \delta e_k + \delta u_{k-1}^T\, P_{k-1} \delta u_{k-1}$$

subject to,

$$\delta x_{k+1} = A\delta x_k + B\delta u_k \quad \text{with } \delta x \epsilon \mathbb{R}^{n_x \times 1},\ \delta u \epsilon \mathbb{R}^{n_u \times 1}$$
$$\delta y_k = C\delta x_k \quad \text{with } \delta y \epsilon \mathbb{R}^{n_y \times 1} \text{ and } \delta x_0 \text{ is given/known}$$
$$\delta e_k = \delta r_k - \delta y_k \quad \text{with } \delta r_k = r_k - y_{op} \text{ being the deviation set points}$$

Note: Here y_{op} is simply the operating points for measured states i.e. α_{op} and $x_{2\,op}$

This LQ optimal control problem has to be converted to standard QP formulation using Kronecker product formulation (see previous chapter for details). Only the final structure is written here for simplicity.

Standard QP form is,

$$\min_{z} \frac{1}{2} z^T Hz + c^T z$$

subject to,

$$A_e z = b_e \rightarrow \text{equality constraint}$$
$$A_i z \leq b_i \rightarrow \text{inequality constraint (if any present)}$$
$$z_L \leq z \leq z_U \rightarrow \text{bounds}$$
$$z = [\delta u, \delta x, \delta e, \delta y]^T$$

Here, z is the vector of unknowns and contains the deviation variables for u, x, e and y. From the kronecker product formulation (for efficiently converting LQ optimal control problem to standard QP problem) we get the $H, c, A_e, b_e, A_i, b_i, z_L$ and z_U. These will be the same as discussed in lecture 3 (thus not shown here but note that they may be different for different formulations and different choices of the vector z).

For making a linear MPC: We have to solve this QP optimization problem (control problem) in Simulink using the *qpOASES* solver. Then select only the first control move (out of the N number of them available) for each control input of the system. For this example of the inverted pendulum, we have only one control input which is the force applied to the cart. For systems with more than one control inputs, you should select the first control move for each control input. Then apply it (them) to the system to go to the next time step (i.e. update the state of the process using the first control move of all the available control inputs). Using the new updated state at the new time step, re-formulate/re-solve the QP optimization problem. Again, select only the first control action for each control input, then apply it (them) to the process and go one time step further. Repeat this procedure at each time step until you stop the simulation.

The basic block diagram for the implementation of the linear MPC is shown in Figure 4.3.

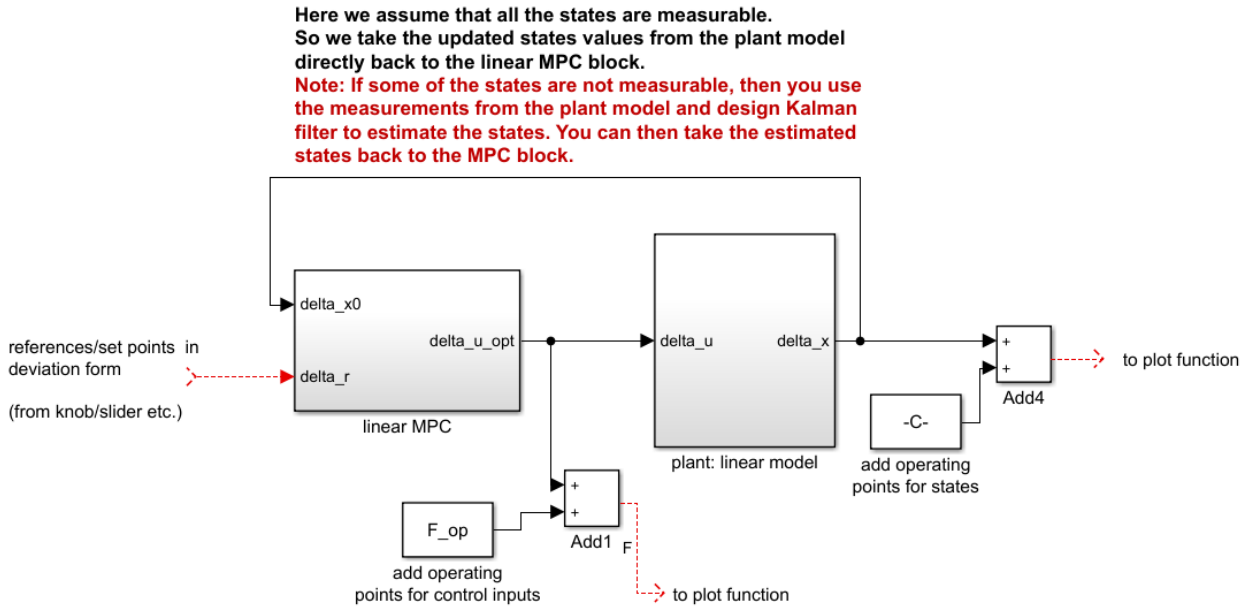


Figure 4.3: Basic block diagram for Linear MPC applied to a linear plant model

The first thing to note is that the "linear MPC" subsystem in Figure 4.3 generates the control action in deviation form (*delta_u_opt*) since the linear model used for making the MPC is in deviation form. This is then applied to the linear plant model. The linear plant model requires δu as control input signal.

Note: If you want to use a linear MPC for controlling the nonlinear plant model, then make sure that you add the operating point for control inputs before sending the control input to the nonlinear model. This is because, the nonlinear model would require u and not δu as control input signal. We will learn more about the use of linear MPC for controlling nonlinear plant models later on in this course when we talk about integral action and steady state offsets.

The second thing to note is that the updated values of the states from the "plant: linear model" subsystem (*delta_x* in Figure 4.3) after the control input was applied to it, is taken back to the "linear MPC" subsystem. This is because, we always need to know the initial values of the states (at any given time step) to formulate the LQ optimal control problem that sits inside the "linear MPC" subsystem. Remember: The $b_{e,1}$ vector (from kronecker product formulation) of the QP problem has the first element as $A\delta x_0$, and thus at each time step we need to know the value of δx_0 which is taken back from the linear plant model.

The third thing to note is that the setpoints or the reference values are also supplied to the "linear MPC" subsystem in Figure 4.3 in deviation form, since the linear model used for making the MPC is in deviation form. The reference values can be changed in real time, for e.g. using a slider or a knob in Simulink. The reference vector (or matrix for if there are more than one output to control) will use the chosen value of the setpoints for the whole prediction horizon, when formulating the LQ optimal control problem at any given time. In other words, if $delta\_r$ (look figure 4.3) contains the current chosen set points for both the pendulum angle and the cart position in deviation form (2 chosen scalar numbers, say 0 and -1 as setpoints for $\delta\alpha$ and δx_2), then to make reference matrix for the whole prediction horizon `delta_ref_N`, we can use the following code snippet:

```
delta_ref_N = [ones(1,N).*delta_r(1);
               ones(1,N).*delta_r(2)];
```

i.e. the same chosen scalar value for the setpoints, `delta_r(1)` and `delta_r(2)` is extended for the whole prediction horizon length of N.

Note: It is also possible to provide varying values for setpoints within the prediction horizon at any given time. But then you have to write up your own routine for it. In practice, for real time application on a real unit, where you would like to run the unit for hours, if not days, if not for all the time; planning for such varying values for setpoints simply becomes cumbersome. It is much easier just to use a knob or slider to change the setpoint in real time and consider to hold the same chosen value of the setpoints throughout the prediction horizon at any given time.

Inside the "linear MPC" subsystem in Figure 4.3, the LQ optimal control problem is formed, and then it is solved, and then the first control move for each control input is selected. The inside of the "linear MPC" subsystem is shown in Figure 4.4.

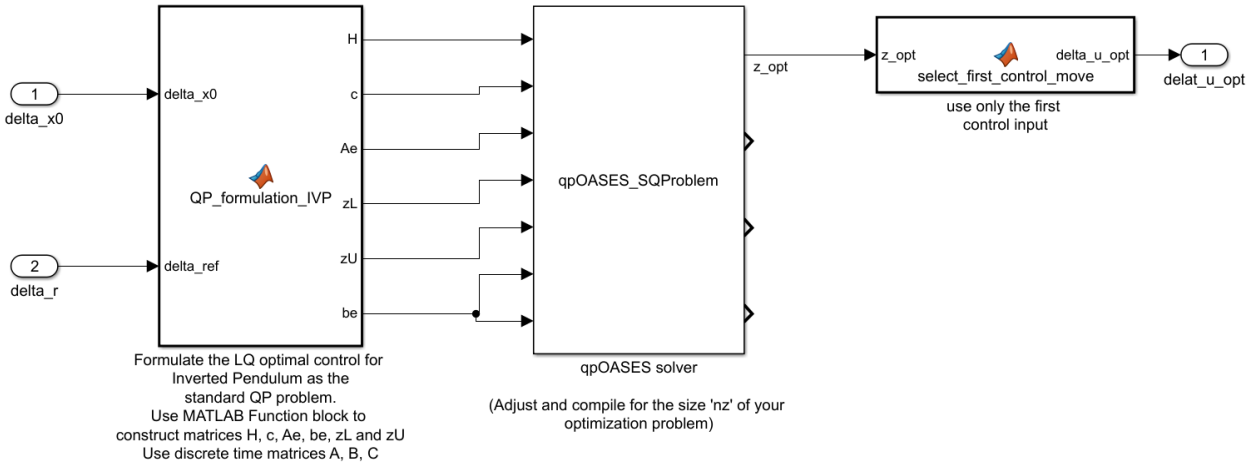


Figure 4.4: Inside the "linear MPC" subsystem, where LQ optimal control problem is formed, then solved using qpOASES solver, and then only the first control move for each control input is selected.

The first block in Figure 4.4 is the same as already explained in lecture 3, and hence not explained here. The LQ optimal control problem is formulated inside this *"QP_formulation_IVP"* MATLAB function block i.e. the standard matrices like H, c, A_e, b_e, z_L, z_U etc. for the standard QP problem are formed inside this block. The code snippet is shown here.

```
function [H,c,Ae,zL,zU,be] = QP_formulation_IVP(delta_x0,delta_ref,A, B, C)
%Here A, B, C are discrete time system matrices. They have been passed as
%parameters and not as inputs to this block.

%configuration
N = 20; %prediction horizon
del_x0 = delta_x0; %using the updated state values directly from the plant model

%setup the references for the whole prediction horizon length
delta_ref_N = [ones(1,N).*delta_ref(1); %first row for the pendulum angle, alpha
               ones(1,N).*delta_ref(2)]; %second row for the cart position, x2

%weighting matrices
Q=diag([1.0 1.0]);      %tuning weight of error
P=1e-3;                 %tuning weight of inputs

%standard QP problem formulation
%compute n's
nx = size(A,1); nu = size(B,2); ny = size(C,1); nz = N*(nx + nu + 2*ny);
% build matrices
H11 = kron(eye(N),P);
H22 = zeros(N*nx,N*nx);
H33 = kron(eye(N),Q);
H44 = zeros(N*ny,N*ny);
H_mat = blkdiag(H11,H22,H33,H44);
%qpOASES does not accept matrices, but only vectors
%we have to change H matrix to vector by stacking elements column wise
H = H_mat(:);

c = zeros(nz,1);

%constraints (from model)
Ae1u = -kron(eye(N),B);
Ae1x = eye(N*nx)-kron(diag(ones(N-abs(-1),1),-1),A);
```

```
Ae1e = zeros(N*nx,N*ny);              %eq 1 is states, hence nx rows, and errors are ny cols
Ae1y = zeros(N*nx,N*ny);
be1 = [A*del_x0;zeros((N-1)*nx,1)];

Ae2u = zeros(N*ny,N*nu);              %eq 2 is measurment eq, ny cols, nu inputs
Ae2x = -kron(eye(N),C);
Ae2e = zeros(N*ny,N*ny);
Ae2y = eye(N*ny);
be2 = zeros(N*ny,1);

Ae3u = zeros(N*ny,N*nu);              %eq3 is error, ny lines, and nu inputs
Ae3x = zeros(N*ny,N*nx);
Ae3e = eye(N*ny);
Ae3y = eye(N*ny);
be3 = reshape(delta_ref_N,N*ny,1);

Ae_mat=[Ae1u Ae1x Ae1e Ae1y;
    Ae2u Ae2x Ae2e Ae2y;
    Ae3u Ae3x Ae3e Ae3y];
%qpOASES does not accept matrices, but only vectors
%we have to change Ae matrix to vector by stacking elements column wise
Ae = Ae_mat(:); %stacking column wise
be=[be1;be2;be3];


%bounds
zL=(-Inf*ones(nz,1));
zU=(Inf*ones(nz,1));
```

In the code snippet above,

```
function [H,c,Ae,zL,zU,be] = QP_formulation_IVP(delta_x0,delta_ref,A, B, C)
```

the system matrices A, B, C in discrete form are passed to this block as parameters and not as inputs to the block. We can make use of a plain MALTAB script to create the A, B, C matrices, to define the operating points for linearization, to define the system parameters etc. For this example, a script in MATLAB called *"initialization_script.m"* is created for this purpose. This file is saved in the same location as the other Simulink files for this example. You should run this script first, before running your Simulink file. Inside the script, you see the following.

```
%file: initialization_script.m
clc
clear

%define model parameters
m1 = 1; %[kg]
m2 = 2; %[kg]
l = 1; %[m]
k_T = 0.1; %[kg/rad^2]
g = 9.81; %[m/s^2]

%perform linearization (e.g. using pen and paper) and type in the
%continuous time Ac, Bc and Cc matrices.
Ac = [0,1,0,0; (m1+m2)*g/(m2*l),0,0,0;0,0,0,1;-m1*g/m2,0,0,0];
Bc = [0;-1/(m2*l);0;1/m2];
Cc = [1,0,0,0;0,0,1,0]; % angle alpha and cart position are measured

%choose the oprerating points for the states
alpha_op = 0;
w_op = 0;
x2_op = 0;
v2_op = 0;

%calculate the operating point for control inputs by solving the plant
%nonlinear model at the steady state
F_op = 0;

% Discretize the continuous time model to get the discrete time linear model
dt = 0.1; %sampling time
```

```
sys = ss(Ac,Bc,Cc,0); %there is no D matrix, so set it as 0
ds = c2d(sys,dt);
A = ds.a; B = ds.b; C = ds.c; %discrete time system matrices
```

In the second block *"qpOASES_SQProblem"* in Figure 4.4, the standard QP is solved. This block has also been already explained in lecture 3, and hence skipped here. Note that, if you have changed the prediction horizon length (and hence the size of the optimal control problem for this example), then you should re-compile your *qpOASES* solver to match the size of your problem.

Finally, in the third block "select_first_control_move" in Figure 4.4, only the first control move for each control input is selected. For the inverted pendulum system, there is only one control input (force applied to the cart), thus we need to select the first control move for this control input only. The code snipped is as shown below.

```
function delta_u_opt = select_first_control_move(z_opt)

delta_u_opt = z_opt(1);
```

Since, δu is the first element of the $z = [\delta u, \delta x, \delta e, \delta y]^T$ vector, the first move of this control input (force applied to cart) is simply `z_opt(1)`.

Note: If you have a system/process, which has, let say 3 different control inputs i.e. if $n_u = 3$ for your system, then you need to extract the first control move for each of these three different control inputs of your system. In that case, you would simply write, `delta_u_opt = z_opt(1:nu)` to extract the first move of all the three control inputs present in your system. Why? Go back to lecture 3 and see the structure of the z^* vector.

In Figure, 4.3, the "plant: linear model" subsystem is the model of the plant which in this example is the linear state space model of the inverted pendulum system, which is in deviation form due to linearization. Inside this subsystem, the linear model in deviation form is formed and integrated as shown in Figure 4.5 below.

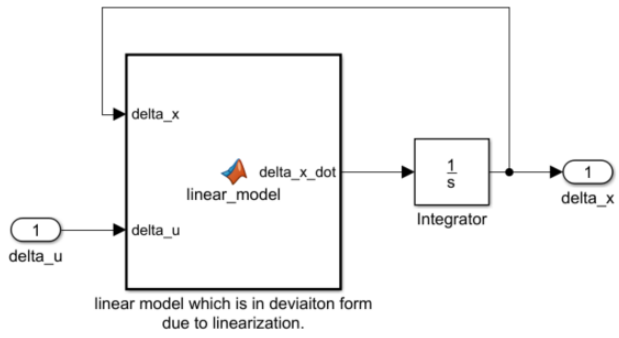


Figure 4.5: Inside the "plant: linear model" subsystem

As also stated before: We can use linear MPC for controlling nonlinear plant model. This is explained in detail in lecture 7.

The complete simulator for the simulation of linear MPC for linear model of the inverted pendulum system is shown in Figure 4.6. The simulator is also available for download in the homepage of the course. When you use the Simulink simulator in your own PC, make sure to first run the *"initialization_script.m"* file in MATLAB. This loads up the model parameters, the discrete time system matrices, the operating points for states and control inputs etc. into the workspace of MATLAB, which then can be used directly in Simulink. In the model configuration parameter (click the gear icon on top middle) in Simulink, you can configure the stop time in Simulink to be infinity by typing *inf*. Select the "Fixed-step" solver type, "ode4 (Runge-Kutta)" as solver and "fixed-step size" to be 0.1. After this, click on the run button in Simulink to start the simulation.

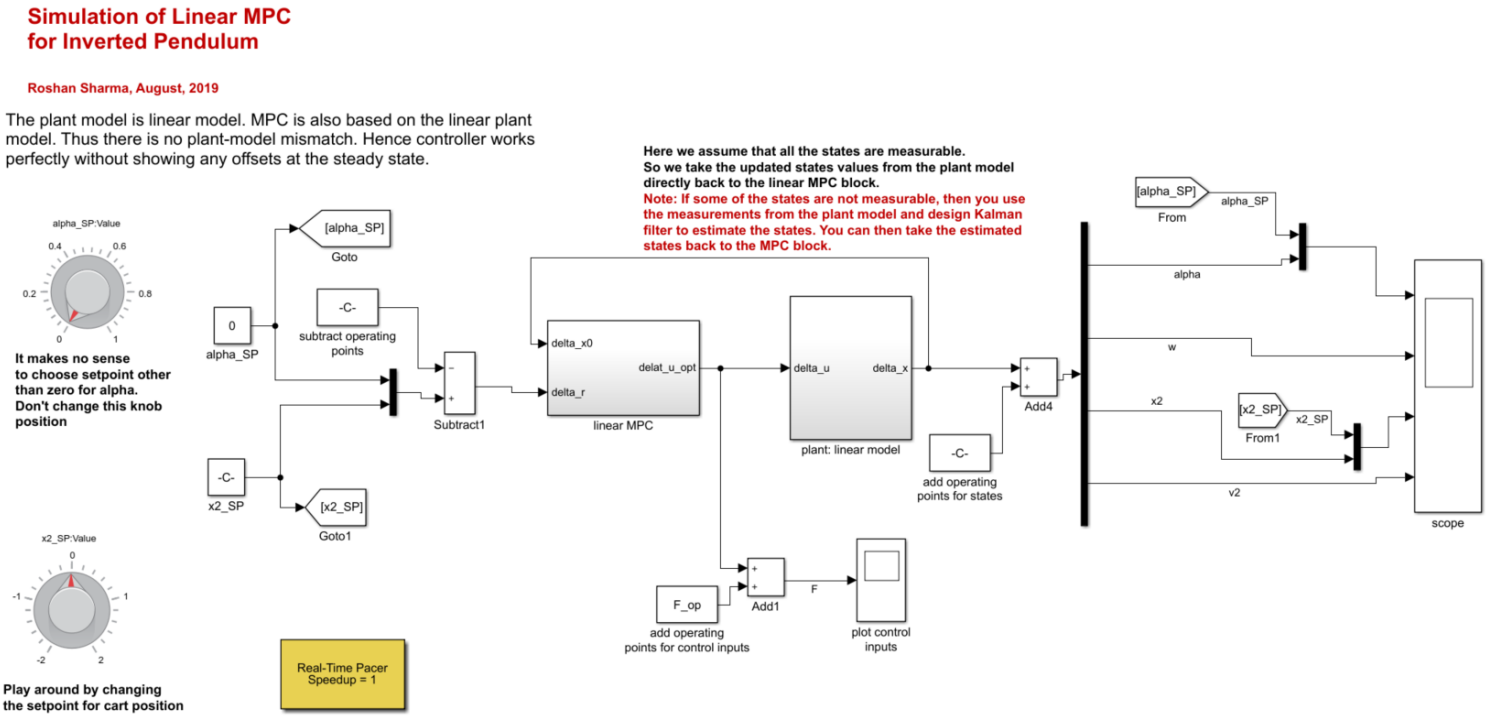


Figure 4.6: An example of implementing linear MPC for linear model of inverted pendulum system

The simulator can be run in real time using the real time pacer, and setpoint for cart position (x_2) can be changed in real time using knob. An example of running the simulator is shown in Figure 4.7(a).

As you can see, the setpoint for the cart position (x_2) is changed between $0, -1$ and 1 in real time while the simulator is running. The controller is able to keep track of the changed position of the cart. In doing so, the pendulum is still kept in the inverted position by the controller. It makes no sense to change the setpoint for pendulum angle α, since the whole idea of using the linear MPC is to keep the pendulum in the upright positon. So the setpoint for angle α is always kept at zero. To control this process, the linear MPC changes the force applied to the cart (F) which is the control input for the system. The snapshot of the control input produced by the linear MPC and used in the process is shown in Figure 4.7 (b).

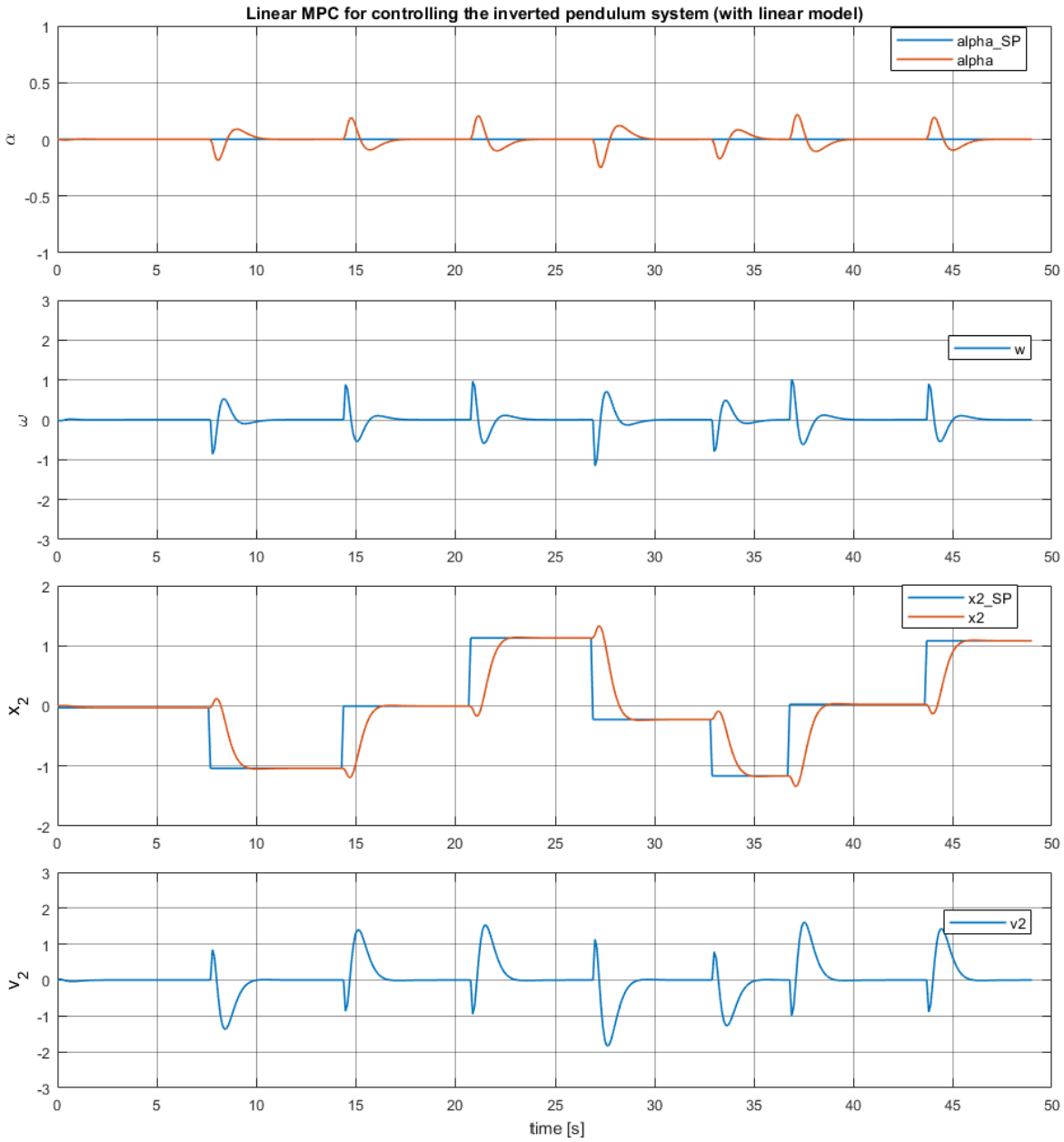


Figure 4.7 (a): An example of running the simulator and changing setpoint for x_2 in real time.

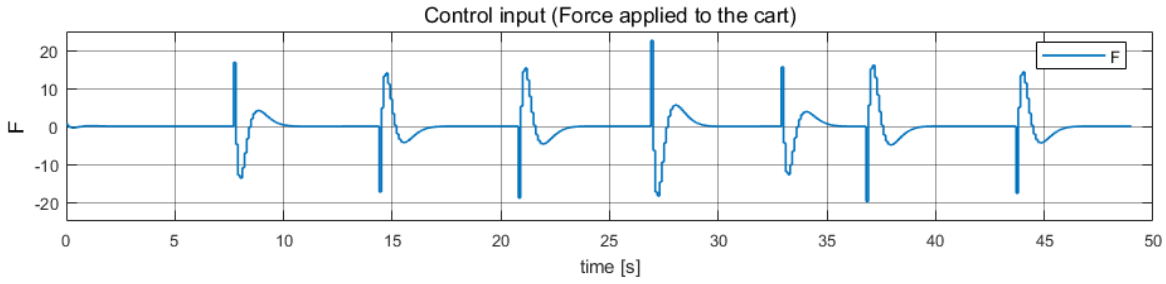


Figure 4.7 (b): Control input used to control the inverted pendulum

4.8 Linear MPC example in a scripting language like MATLAB + Execution time

It is also very interesting to study about the execution time need to run the linear MPC for the inverted pendulum. For this we will be using MATLAB (not Simulink) for implementing the linear MPC. The reasons for this are:

a) In this course, we use Simulink for real time simulation. Recording the exact time needed to solve the QP problem at each time step is much more precise and easier in MATLAB.
b) It also gives us an opportunity to see and understand how we can implement a linear MPC in a scripting (text based) language like MATLAB. Then you can easily implement the same in other text based languages like Python, Julia, etc.
c) We can compare the execution time for linear MPC with this sparse formulation of LQ optimal control problem with a more dense formulation by QR factorization (detail in lecture 5). We can see whether reduction of the size of the LQ optimal control problem increases the speed of execution.

Please look at the video https://web01.usn.no/~roshans/mpc/videos/lecture4/execution-time-linearMPC-MATLAB.mp4 for more details.

Note: The text in this section of the lecture note will be update by the end of this semester. However, the video covers all of it.

Lecture 5:

5.1 Review:

In lecture 3, we learned how to transform a given LQ optimal control problem to a standard QP problem using kronecker product formulation. In lecture 4, we learned how to introduce feedback to LQ optimal control problem by receeding horizon strategy and make a linear MPC.

A typical content of z (the vector of unknowns to be optimized) is,

$$z^T = (u_0^T, \dots, u_{N-1}^T, x_1^T, \dots, x_N^T, e_1^T, \dots, e_N^T, y_1^T, \dots, y_N^T)$$

Let us take an example where $N = 20, n_x = 15, n_u = 6$ and $n_y = 6$. Then the number of unknown variables are,

$$n_Z = N.(n_u + n_x + n_y + n_y) = 20.(6 + 15 + 6 + 6) = 660$$

The optimizer (for e.g. *qpOASES* in Simulink or *quadprog* in MATLAB) will try to calculate the optimal values of these 660 unknown variables (also called as decision variables).

If we consider an LQ optimal control problem as,

$$min\ J = \frac{1}{2}\sum_{k=1}^{N} e_k^T Q e_k + u_{k-1}^T P u_{k-1} \tag{5.1}$$

subject to,

$$x_{k+1} = Ax_k + Bu_k + v_k \quad \text{with } x_0 \text{ given} \tag{5.2}$$

$$y_k = Cx_k + Du_k + w_k \tag{5.3}$$

$$e_k = r_k - y_k \tag{5.4}$$

Equation (5.4) has $N.n_y = 20 \times 6 = 120$ equality constraints.

If we can eliminate the error variables by substituting equation (5.4) into the objective function (5.1) then we will have,

$$n_Z = N.(n_u + n_x + n_y) = 20.(6 + 15 + 6) = 540$$

i.e. there will be only 540 unknown variables to optimize instead of 660 unknown variables.

In this lecture, we will learn about reducing the size of the optimal control problems. We will do so in three different ways: a) by using Lagrangian functions b)by QR factorization c) by grouping of control inputs into two or more groups.

5.2 Reducing the size of optimal control problems:

5.2.1 Eliminating equality constraints using Lagrangian functions

Let us consider a quadratic cost function with linear equality constraint as,

$$\min_z\ J = f(z) = \frac{1}{2} z^T H z + c^T z$$

subject to

$$A_e z = b_e \tag{5.5}$$

The Lagrangian function $F(z, \lambda)$ is defined as,

$$F(z, \lambda) = f(z) + \lambda^T (A_e z - b_e) \tag{5.6}$$

where λ is known as Lagrange multiplier.
You can see that equation (5.5) which is a constrained optimization problem, has been changed to a reduced form of equation (5.6) without the equality constraints i.e. an unconstrained optimization problem. Therefore, we can now work further with this reduced problem of equation (5.6). However, the reduced problem has both z and λ as unknowns. This means that although the original problem of equation (5.5) has been reduced to equation (5.6), the number of unknowns to be optimized has been increased.
So we have,

$$\min_{(z,\lambda)} \quad F(z,\lambda) = \frac{1}{2} z^T H z + c^T z + \lambda^T (A_e z - b_e) \tag{5.7}$$

To find the minimum of unconstrained problem of equation (5.7), we simply take the first derivative and equate them to zero.
Note :- Since equation (5.7) has two unknown variables which are z and λ, we need to take partial derivative as,

$$\frac{\partial F(z,\lambda)}{\partial z} = Hz + c + A_e^T \lambda \tag{5.8}$$

$$\frac{\partial F(z,\lambda)}{\partial \lambda} = A_e z - b_e \tag{5.9}$$

Equating (5.8) and (5.9) to zero for minimum we get,

$$Hz + A_\epsilon^T \lambda = -c \tag{5.10}$$

$$A_e z + 0\lambda = b_e \tag{5.11}$$

Arranging (5.10) and (5.11) in a compact form,

$$\underbrace{\begin{bmatrix} H & A_\epsilon^T \\ A_e & 0 \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} z \\ \lambda \end{bmatrix}}_{\tilde{z}} = \underbrace{\begin{bmatrix} -c \\ b_e \end{bmatrix}}_{\tilde{b}_\epsilon} \tag{5.12}$$

The task has now reduced to solving equation (5.12) which is a linear algebraic equation to find the optimal solution of the original problem of equation (5.5) i.e. solve $M\tilde{z} = \tilde{b}_\epsilon$ to find $\tilde{z}^*$.
$\tilde{z}^*$ has the optimal solution z^* and λ^* . If M is invertible then $\tilde{z}^* = M^{-1}\tilde{b}_e$
You can also try :

$$M^T M \tilde{z} = M^T \tilde{b}_e$$

$$\tilde{z}^* = (M^T M)^{-1} M^T \tilde{b}_e$$

5.2.2 Eliminating equality constraints by QR factorization:

Background:

Let us consider the LQ optimization problem of the standard form *"(if original problem is not in the standard from, I assume that by this time, the students have mastered the transformation process of expressing it into the standard from)"* as,

$$\left.\begin{array}{l} \underset{z}{Min} \ \frac{1}{2} z^T H z + c^T z \\ \text{subject to} \\ \quad A_e z = b_e \\ \quad A_i z \leq b_i \\ \quad z_L \leq z \leq z_H \end{array}\right\} \tag{5.13}$$

Optimization problem of (5.13) can be converted into a quadratic problem of reduced form as,

$$\left.\begin{array}{l} \underset{z_2}{min} \ \frac{1}{2} z_2^T \tilde{H} z_2 + \tilde{c}^T z_2 \\ \quad \tilde{A}_i z_2 \leq \tilde{b}_i \end{array}\right\} \tag{5.14}$$

The reduced problem of (5.14) has only linear inequality constraints i.e. we have eliminated the linear equality constraints from the original optimization problem of (5.13)

IDEA:

The trick here is to use the linear equality constraint $A_e z = b_e$ present in the original problem to split the unknown variables z into two parts: *"basic variables"* and *"non-basic variables"*. Using the equality constraint, the basic variables are expressed in terms of the non-basic variables. The basic variables (which are the functions of non-basic variables) are then substituted in the objective function of equation (5.13). The objective function will then only have the non-basic variables. This will also result in the elimination of the linear equality constraint & we will obtain the reduced problem of equation (5.14).

Example: Let us consider a quadratic optimization problem as,

$$\underset{z}{min} \quad f(z) = 2x_1^2 + 5x_2^2 - 3x_3^2 + x_4^2 + 4x_1 + 2x_2 + 6x_3 - 2x_4 \tag{5.15}$$

subject to,

$$x_1 + 5x_2 - 2x_3 + x_4 = 2 \tag{5.16}$$
$$-3x_1 + x_2 - x_3 - x_4 = 2 \tag{5.17}$$
$$2x_1 + x_2 - 2x_3 - 2x_4 = 5 \tag{5.18}$$
$$x_1 - 3x_2 - 3x_3 + 2x_4 = 6 \tag{5.19}$$

Here, the vector of unknowns can be written as,

$$z = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

a) Divide the vector of unknowns into two parts.

$$z = \left.\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}\right\} \begin{array}{l} \text{basic variables, } 'z_1' \\ \text{non} - \text{basic variables, } 'z_2' \end{array}$$

i.e. x_1 & x_2 are basic variables x_3 & x_4 are non basic variables.

b) Express the basic variables as algebraic functions of non-basic variables. Use linear equality constraints for doing it.

From equation (5.16) and (5.19) you can express, (the order in which we choose linear equality constraints doesn't matter)

$$x_2 = -\frac{1}{8}x_3 + \frac{1}{8}x_4 - \frac{1}{2} \tag{5.20}$$

i.e. $x_2 = f(x_3, x_4)$ i.e. x_2 is expressed using non-basic variables. Similarly, from equation (5.17) and (5.18) we can write,

$$x_1 = \frac{1}{5}x_3 + \frac{1}{5}x_4 + \frac{3}{5} \tag{5.21}$$

$x_1 = f(x_3, x_4)$ i.e. x_1 is expressed using non-basic variables.

c) To eliminate the equality constraints completely, substitute equation (5.20) and (5.21) in the objective function of equation (5.15).

$$\min_{z_2} \; f(z_2) = 2\left(\frac{1}{5}x_3 + \frac{1}{5}x_4 + \frac{3}{5}\right)^2 + 5\left(-\frac{1}{8}x_3 + \frac{1}{8}x_4 - \frac{1}{2}\right)^2 - 3x_3^2 + x_4^2 + 4x_1 + 2x_2 + 6x_3 - 2x_4 \tag{5.22}$$

Here $f(z_2)$ consists of only the non-basic variables z_2 i.e. $f(z_2) = f(x_3, x_4)$

So, in this way, optimization problem of (5.15) -(5.19) is reduced to the problem of (5.22). We can then solve equation (5.22) to obtain optimal values z_2^* i.e. x_3^* & x_4^*. Further we can find the optimal values of x_1 and x_2 using equations (5.20) and (5.21) as,

$x_1^* = f(x_3^*, x_4^*)$ and $x_2^* = f(x_3^*, x_4^*)$.

Well, that was just an example. So, the question remains:

Q) Is there a way to generalize these steps i.e. steps (a) -(c)

Answer: Yes, by using QR factorization (for linear case)

Let us consider the linear equality of eq. (5.23)

$$A_e z = b_e \tag{5.23}$$

Let us assume equation (5.23) has n number of equations i.e. equation (5.23) is a compact form of n linear equality constraints. Then, $A_e \epsilon R^{n \times n_z}$, where n_z is the total number of unknown variables that are listed in vector z. Let the rank of A_e be r i.e. $r =$ rank(A_e). Rank of a matrix gives you the number of linearly independent rows of the matrix, here in this case, the number of independent linear equality constraints.

Now, let us decompose A_e into the product of Q and R as,

$$A_e = QR$$

where $Q =$ orthogonal matrix i.e. $Q^T Q = I$ and $Q \epsilon \mathbb{R}^{n \times n}$, $R =$ upper triangular matrix, $R \epsilon \mathbb{R}^{n \times n_z}$.

For example, let,

$$A_e = \begin{bmatrix} 2 & 3 & 1 \\ 1 & 5 & 3 \\ 3 & 7 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 0.53 & -0.5 & 0.6 \\ 0.26 & 0.86 & 0.42 \\ 0.8 & 0.04 & -0.59 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} 3.7 & 8.5 & 2.1 \\ 0 & 3.1 & 2.1 \\ 0 & 0 & 1.3 \end{bmatrix}}_{R}$$

Furthermore, R can be written as,

$$R = \begin{bmatrix} R_1 & R_2 \\ 0 & 0 \end{bmatrix} \tag{5.24}$$

where, $R_1 \epsilon \mathbb{R}^{r \times r}$ is full rank i.e. rank $(R_1) = r = \text{rank}(A_e)$ and $R_2 \epsilon \mathbb{R}^{r \times (n_z - r)}$ is the upper right submatrix of R.

Note that the 0's under R_1 has the size of $(n - r) \times r$ and the 0's under R_2 has the size of $(n - r) \times (n_z - r)$. So, we can write the linear equality constraints of equation (5.23) as,

$$A_e z = b_e$$
$$QRz = b_e$$

Multiplying both sides by Q^T we get,

$$\underbrace{Q^T Q}_{I} Rz = \underbrace{Q^T b_e}_{\bar{b}_e}$$
$$Rz = \bar{b}_e \tag{5.25}$$

Now let us split the vector of unknowns z into $z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$ or $z^T = (z_1^T, z_2^T)$

where, z_1 are the basic variables of size $(r \times 1)$ and z_2 are the nonbasic variables of size $(n_z - r) \times 1$.

Also let us split $\bar{b}_\epsilon$ into two parts as,

$$\bar{b}_e = \begin{bmatrix} \bar{b}_{e,1} \\ \bar{b}_{e,2} \end{bmatrix} \tag{5.26}$$

where, $\bar{b}_{e,1} \epsilon \mathbb{R}^{r \times 1}$ and $\bar{b}_{e,2} \epsilon \mathbb{R}^{(n-r) \times 1}$

Then we have from equations (5.24, 5.25 & 5.26) we get,

$$\begin{bmatrix} R_1 & R_2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \bar{b}_{e,1} \\ \bar{b}_{e,2} \end{bmatrix} \tag{5.27}$$

From equation 5.27 we get,

$$R_1 z_1 + R_2 z_2 = \bar{b}_{e,1}$$

Since R_1 is full ranked, it is invertible. Then we can express z_1 (the basic variables) in terms of z_2 (the non-basic variables) as,

$$z_1 = R_1^{-1} (\bar{b}_{e,1} - R_2 z_2) \tag{5.28}$$

So, in summary we have,

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} R_1^{-1} (\bar{b}_{e,1} - R_2 z_2) \\ z_2 \end{bmatrix} = \begin{bmatrix} R_1^{-1} \bar{b}_{e,1} - R_1^{-1} R_2 z_2 \\ 0 + z_2 \end{bmatrix}$$
$$\downarrow$$
$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} R_1^{-1} \bar{b}_{e,1} \\ 0 \end{bmatrix} + \begin{bmatrix} -R_1^{-1} R_2 \\ I \end{bmatrix} z_2 \tag{5.29}$$

The sizes of zeros and identity matrix in equation (5.29) are: $0 \rightarrow (n_z - r) \times 1$ and $I \rightarrow (n_z - r) \times (n_z - r)$.

Now let us substitute z from equation (5.29) in the original problem given by equation (5.13) i.e.

$\Rightarrow \frac{1}{2}\, z^T H z + c^T z$ can be written as,

$$\frac{1}{2}\left(\begin{bmatrix} R_1^{-1}\,\overline{b}_{e,1} \\ 0 \end{bmatrix} + \begin{bmatrix} -R_1^{-1}\,R_2 \\ I \end{bmatrix} z_2\right)^T H \left(\begin{bmatrix} R_1^{-1}\,\overline{b}_{e,1} \\ 0 \end{bmatrix} + \begin{bmatrix} -R_1^{-1}\,R_2 \\ I \end{bmatrix} z_2\right) + c^T\left(\begin{bmatrix} R_1^{-1}\,\overline{b}_{e,1} \\ 0 \end{bmatrix} + \begin{bmatrix} -R_1^{-1}\,R_2 \\ I \end{bmatrix} z_2\right) \tag{5.30}$$

We can see that equation (5.30) has only the non-basic variable z_2. After rearranging and solving equation (5.30) we can express it in the reduced order standard form as,

$$\underset{z_2}{min}\ ,\ \frac{1}{2}\, z_2^T\, \widetilde{H} z_2 + \tilde{c}^T z_2 + \widetilde{K} \tag{5.31}$$

where,

$$\widetilde{H} = \begin{bmatrix} -R_1^{-1}\,R_2 \\ I \end{bmatrix}^T H \begin{bmatrix} R_1^{-1} R_2 \\ I \end{bmatrix} \tag{5.32}$$

$$\tilde{c} = \begin{bmatrix} -R_1^{-1}\,R_2 \\ I \end{bmatrix}^T H \begin{bmatrix} R_1^{-1}\,\overline{b}_{e,1} \\ 0 \end{bmatrix} + \begin{bmatrix} -R_1^{-1}\,R_2 \\ I \end{bmatrix}^T c \tag{5.33}$$

$$\widetilde{K} = \frac{1}{2}\begin{bmatrix} R_1^{-1}\,\overline{b}_{e,1} \\ 0 \end{bmatrix}^T H \begin{bmatrix} R_1^{-1}\,\overline{b}_{e,1} \\ 0 \end{bmatrix} + c^T R_1^{-1} \begin{bmatrix} \overline{b}_{e,1} \\ 0 \end{bmatrix} = \text{constant} \tag{5.34}$$

Since $\widetilde{K}$ is constant, it can be safely removed from the optimization problem. Then the objective of the reduced problem is,

$$\underset{z_2}{min}\ \ \frac{1}{2}\, z_2^T\, \widetilde{H} z_2 + \tilde{c}^T z_2$$

At this stage we have eliminated the equality constraint. But the original optimization problem of equation (5.13) also has linear inequality constraints and bounds.

For the <u>inequality constraints</u> we have from equation (5.13),

$$A_i z \leq b_i$$

Substituting z from equation (5.29) in this above equation we get,

$$A_i\left(\begin{bmatrix} R_1^{-1}\,\overline{b}_{e,1} \\ 0 \end{bmatrix} + \begin{bmatrix} -R_1^{-1}\,R_2 \\ I \end{bmatrix} z_2\right) \leq b_i$$

$$A_i \begin{bmatrix} -R_1^{-1}\,R_2 \\ I \end{bmatrix} z_2 \leq b_i - A_i \begin{bmatrix} R_1^{-1}\,\overline{b}_{e,1} \\ 0 \end{bmatrix} \tag{5.35}$$

<u>For the bounds</u>, we have from equation (5.13),

$$z_L \leq z \leq z_H$$

Substituting z from equation (5.29) in this above equation we get,

$$z_L \leq \begin{bmatrix} R_1^{-1}\,\overline{b}_{e,1} \\ 0 \end{bmatrix} + \begin{bmatrix} -R_1^{-1}\,R_2 \\ I \end{bmatrix} z_2 \leq z_H$$

$$z_L - \begin{bmatrix} R_1^{-1}\,\overline{b}_{e,1} \\ 0 \end{bmatrix} \leq \begin{bmatrix} -R_1^{-1}\,R_2 \\ I \end{bmatrix} z_2 \leq z_H - \begin{bmatrix} R_1^{-1}\,\overline{b}_{e,1} \\ 0 \end{bmatrix} \tag{5.36}$$

Equation (5.36) can be written separately using two equations (i.e. bounds can be expressed as two inequality constraints) as,

$$-\begin{bmatrix} -R_1^{-1} R_2 \\ I \end{bmatrix} z_2 \leq -z_L + \begin{bmatrix} R_1^{-1} \overline{b}_{e,1} \\ 0 \end{bmatrix} \tag{5.37}$$

$$\begin{bmatrix} -R_1^{-1} R_2 \\ I \end{bmatrix} z_2 \leq z_H - \begin{bmatrix} R_1^{-1} \overline{b}_{e,1} \\ 0 \end{bmatrix} \tag{5.38}$$

Equations (5.35), (5.37) & (5.38) can be written in compact form as,

$$\tilde{A}_i z_2 \leq \tilde{b}_i$$

where,

$$\tilde{A}_i = \begin{bmatrix} A_i \begin{bmatrix} -R_1^{-1} R_2 \\ I \end{bmatrix} \\ -\begin{bmatrix} -R_1^{-1} R_2 \\ I \end{bmatrix} \\ \begin{bmatrix} -R_1^{-1} R_2 \\ I \end{bmatrix} \end{bmatrix}, \; \tilde{b}_i = \begin{bmatrix} b_i - A_i \begin{bmatrix} R_1^{-1} \overline{b}_{e,1} \\ 0 \end{bmatrix} \\ -z_L + \begin{bmatrix} R_1^{-1} \overline{b}_{e,1} \\ 0 \end{bmatrix} \\ z_H - \begin{bmatrix} R_1^{-1} \overline{b}_{e,1} \\ 0 \end{bmatrix} \end{bmatrix} \tag{5.39}$$

Thus, the original problem has been formulated as the reduced problem as,

$$\min_{z_2} \quad \frac{1}{2} z_2^T \tilde{H} z_2 + \tilde{c}^T z_2 \tag{5.40}$$

subject to,

$$\tilde{A}_i z_2 \leq \tilde{b}_i \tag{5.41}$$

where $\tilde{H}$ is given by equation (5.32), $\tilde{c}$ is given by equation (5.33), $\tilde{A}_i$ and $\tilde{b}_i$ are given by equation (5.39).

The reduced problem given by equations (5.40 & 5.41) does not have equality constraints but it is equivalent to the original problem of (5.13). We can solve the reduced problem of equations (5.40 & 5.41) using *qpOASES* solver in Simulink or *quadprog* solver in MATLAB. Let us assume that the optimal values returned by the solver be denoted by z_2^* . After finding the optimal values z_2^* of the reduced optimization problem of (5.40 & 5.41), we can find the optimal values of the basic variables as,

$$z_1^* = R_1^{-1}\left(\overline{b}_{e,1} - R_2 z_2^*\right) \tag{5.42}$$

Finally, we have the optimal values of the unknown variables as, $\mathrm{z}^* = \begin{bmatrix} \mathrm{z}_1^* \\ \mathrm{z}_2^* \end{bmatrix}$

Notes:

(1) If the reduced problem does not have inequality constraints (because it had no inequality constraints in the original problem) then the reduced problem is unconstrained in nature. Under such condition, we can then find z_2^* analytically without using any solvers as,

$\frac{\partial}{\partial z_2}\left(\frac{1}{2} z_2 \tilde{H} z_2 + \tilde{c}^T z_2\right) = 0$ → First order derevative of unconstrained problem equated to zero

$\frac{1}{2} \cdot 2\, \tilde{H} z_2^* + \tilde{c} = 0$

$z_2^* = -\tilde{H}^{-1} \tilde{c}$

(2) If we check the eigen value of $\widetilde{H}$ of the objective of the reduced problem (5.40) we can know the information about whether the reduced objective function is convex or not. If the eigen values of $\widetilde{H}$ are all positive, then the objective function of the optimization problem (5.39) is convex. This means that the solution z^* is the global solution.

(3) A typical content of the unknown vector z of unknowns are,

$$z^T = (u_0^T, \dots, u_{N-1}^T, x_1^T, \dots, x_N^T, e_1^T, \dots, e_N^T, y_1^T, \dots, y_N^T)$$

However, using the QR decomposition, the reduced problem will be expressed as the non-basic variable (z_2). Since for a control problem the unknowns that are of primary importance to compute are the control inputs $u_0,\ u_1, \dots\dots\dots, u_{N-1}$, it is a good idea to make $u_0,\ u_1, \dots\dots\dots, u_{N-1}$, as the non-basic variables. If we do so, we directly get the optimal values of the control inputs from the solvers i.e. we do not need to do the additional calculation using equation (5.28) to calculate z_1^*. We can make the control inputs $u_0,\ u_1, \dots\dots\dots, u_{N-1}$ as the non-basic variables by putting them at the end of the vector z.

(4) By Lagrangian or QR decomposition, we may have reduced the order of the optimization problem. But in doing so, we are paying some price. The penalty to be paid is that in the reduced problem, the matrices $\widetilde{H}, \tilde{c}, \tilde{A}_i$ & $\tilde{b}_i$ are dense and hence the sparse matrix structure that is present in the original problem is lost.

In a sparse matrix, the majority of the elements are zeros. In a dense matrix, the majority of the elements are non-zero. Good & efficient solvers use the advantage of sparsity for memory management by:

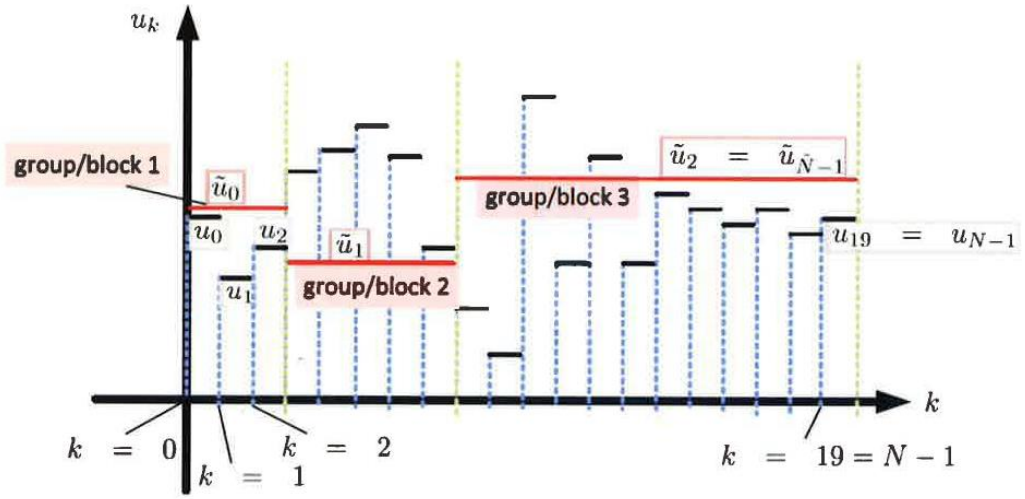
- Storing only the nonzero elements of the matrix, together with their indices
- Elimination of unnecessary low level math operations on zero elements of the matrix. For e.g. zero-adds ($x + 0$ is always x, and this zero-add operation can be eliminated).

5.2.3 Grouping control input variables:

Another way to reduce the size of the optimization problem is by grouping the control input variable.

IDEA:

Create blocks/groups over the prediction horizon. Within a block/group , keep the control signals constant. As an example: for a prediction horizon $= 20$, we can have the following grouping of control signals.



From $k = 0$ to $k = 3, \Rightarrow$ group 1/block 1
From $k = 3$ to $k = 8, \Rightarrow$ group 2/block 2
From $k = 8$ to $k = N = 20, \Rightarrow$ group 3/block 3

For group 1 $\Rightarrow u_0 = u_1 = u_2 = \tilde{u}_0$
For group 2 $\Rightarrow u_3 = u_4 = \cdots \ldots\ldots\ldots\ldots\ldots\ldots = u_7 = \tilde{u}_1$
For group 3 $\Rightarrow u_8 = u_9 = \cdots \ldots\ldots\ldots\ldots\ldots\ldots = u_{19} = \tilde{u}_2$

So after grouping the control signals into three groups, we have only three unknown control inputs to calculate ($\tilde{u}_0, \tilde{u}_1$ and $\tilde{u}_2$) instead of original 20 control inputs.
In general, due to grouping of signals, we introduce additional equality constraints,

$$\tilde{u}_0 = u_0 = u_1 = \ldots\ldots\ldots\ldots = u_{N_1-1}$$
$$\tilde{u}_1 = u_{N_1} = u_{N_1+1} = \ldots\ldots\ldots\ldots\ldots u_{N_2-1}$$
$$\tilde{u}_2 = u_{N_2} = u_{N_2+1} = \ldots\ldots\ldots\ldots\ldots u_{N_3-1}$$
$$\vdots \quad \vdots \quad \vdots \quad \vdots$$
$$\tilde{u}_{\tilde{N}-1} = u_{N_{\tilde{N}-1}} = u_{N_{\tilde{N}}+1} = \ldots\ldots\ldots\ldots\ldots u_{N-1}$$

Here, N_1 is the length of the first control group, N_2 the length of the second control group and so on. $\tilde{N}$ is the number of groups/blocks.
Suppose $n_u = 3 =$ three control signals of a process, $N = 20 =$ prediction horizon, then the optimizer have to calculate $20\times3 = 60$ unknown control inputs for the horizon length without grouping of signals.
Now, let $\tilde{N} = 3 =$ number of groups/blocks. Now we need to calculate $3\times 3 = 9$ unknown control inputs for the whole prediction horizon length. It reduces the number of unknown variables significantly.

Note:

(1) The length of each block can be made equal. However, variable length blocks can also be used such that second block is a little longer than the first block, third block is longer than second block and so on.

(2) The additional equality constraints that arise due to grouping of control inputs can also be eliminated for e.g. using QR factorization.

(3) At the first glance it looks like the additional equality constraints due to grouping of control inputs increase the size of control problem since there are now more equality constraints than the original problem. Yes this is true, but the execution time for solving an optimization problem is more dependent and affected by the number of variables to optimize. So the increase in the number of equality constraints due to input signal grouping has insignificant effect compared to the reduction of the execution time due to decrease in the total number of variables to optimize.

5.3 Execution time for Linear MPC for reduced LQ optimal control problem with QR factorization:

Look at the video https://web01.usn.no/~roshans/mpc/lecture5/linear-MPC-dense-speed.mp4 for more details.

In this video, you can see that the linear MPC with reduced LQ optimal control problem can be solved much faster than the original MPC. For this particular example of the inverted pendulum, the execution time speed gain is about 4.

Note: The text for this section of the lecture note will not be updated by the end of this semester. However, the video includes all of it.

5.4 Ensuring feasibility:

In an optimization problem, one question arises, "Does a feasible point or solution always exist?"

Answer :- No.

The problem is said to be infeasible, if there is no feasible solution or simply the solution does not exists.

What causes infeasibility?

Constraints, particularly inequality constraints on the output & states cause infeasibility. Remember that the constraints define the boundary/region where the solution should lie. If the solution does not lie in the feasible region then the constraints are not satisfied and infeasibility occurs. Now let us define the two types of constraints that form a background for feasibility studies.

5.4.1 Types of constraints

(i) *Hard constraint:*

Constraints that have to be always obeyed strictly are the hard constraints. A system must adhere to hard constraints. Usually constraints on the input variables can be posed as hard constraints. For example: the opening of a choke valve in a pipeline should be within 0 & 100% i.e. $0 \leq u \leq 100$ is a hard constraint. This constraint cannot be violated at any cost. The valve cannot be opened more than 100% & cannot be closed below 0% (the physical

structure & the operational condition of the choke valve strictly puts this limit). Any value of u outside $0 \leq u \leq 100$ is simply not possible/feasible.
Other examples of hard constraints can be: capacity of an equipment, limits on actuators, completion time of a project (let's say the group project of this course) etc.

(ii) ***Soft constraints:***
Constraints which are fulfilled if possible, but if it is not possible, disobeying or breaking the constraints is also allowed are the soft constraints. However, violating the constraint should be made as gentle as possible.
With soft constraints, the system tries to adhere or stick to it, but the system can violate the constraints if necessary in order to find a feasible solution (but of course a solution that complies with the hard constraints).
For example: Process outputs like flow rate, temperature, pressure etc. (unless they are too serious to disobey or too serious to violate) can be regarding as soft constraints depending on the operating conditions. Also note that when disobeying or breaking the constraints, you may have to compromise some other things like quality of the product.
Violating the soft constraint is also known as ***relaxing the constraints***. The goal during constraints relaxation is to minimize the total amount of violation of all the soft constraints.

Let us look at an example. Consider an optimization problem as,

$$\min_{u} \quad J = y^2 + 0.1u^2$$

subject to,

$$x_{k+1} = \frac{1}{2}x_k + u_k$$
$$y_k = x_k$$
$$0 \leq u_k \leq 100$$

For any value of u_k such that $0 \leq u_k \leq 100$, we can find a feasible solution. Now, let us add another inequality constraints to the problem such that,

$y_k \geq 300 \quad \rightarrow$ inequality constraints on output variables

At steady state,

$$x = \frac{1}{2}x + u$$
$$\downarrow$$
$$x = 2u$$

and

$$y = x = 2u$$

For any values of u (of course between 0 & 100), the value of y will never be greater than 200. Therefore, the inequality constraint $y_k \geq 300$ will never be satisfied and hence the

problem becomes infeasible. For this example, the infeasibility is due to the presence of the inequality constraints on the output variables.
$\Rightarrow$ Normally, inequality constraints on the control input variables cannot render the optimal control infeasible.

5.4.2 Relaxing the constraints

Taking the infeasible problem from the example presented above (section 5.4.1), the major question remains as: **How to find a feasible solution in this case?**
Answer:- If $y_k \geq 300$ is a soft constraint, then relax it.
Relaxation of inequality constraints if they are soft constraints is performed by using slack variables. For relaxation, the inequality constraint $y_k \geq 300$ can be written as,

$$y_k \geq 300 - S_k$$

where S_k is the slack variable.
To soften the constraint, remember to minimize the amount of violation. This can be done by adding the slack variables S_k to the unknown to be optimized. Then the relaxed optimization problem becomes,

$$\underset{(u,S)}{min} \quad J = y^2 + 0.1u^2 + \beta S_k$$

subject to,

$$x_{k+1} = \frac{1}{2}x_k + u_k$$
$$y_k = x_k$$
$$y_k \geq 300 - S_k$$

In general, an infeasible optimization problem can be relaxed by adding the slack variable to the unknowns z and then penalizing the degree of violation as,

$$\underset{(z, s_k^L, s_k^H)}{min} J = \frac{1}{2}\sum_{k=1}^{N} e_k^T Q e_k + u_{k-1}^T P u_{k-1} + \Delta u_{k-1}^T R u_{k-1} + \beta_L^T S_K^L + \beta_H^T S_K^H + (S_K^L)^T S_1 S_K^L + (S_K^H)^T S_2 S_K^H$$

subject to,

$x_{k+1} = Ax_k + Bu_K + \Gamma v_K$, $\quad x_0, u_{-1}$ given
$y_k = Dx_k + Eu_k$
$y_L - S_k^L \leq y_k \leq y_H + S_k^H$ $\quad \rightarrow$ additional of slack variables to relax the constraint
$u_L \leq u_k \leq u_H$
$\Delta u_L \leq \Delta u_k \leq \Delta u_H$

Note:

(1) Here S_k^L and S_k^H are the slack variable that are added to the output constraint. A suitable value of S_k^L and S_k^H (of course which has to be calculated by the optimizer) will break the violation in the most gentle way (with least violation error). Since you need to minimize the violation, you consider S_k^L and S_k^H as extra unknown variables and add them to the optimization problem along with z.

(2) S_1 and S_2 are the weighting matrices for the slack variables such that $S_1 \geq 0$ and $S_2 \geq 0$ and β_L & β_H are the tuning parameters.

(3) The slack variables S_k^L and S_k^H should be zero if the constraints are not violated. You don't need to relax the constraints if the constraints are already satisfied. The slack variables should be non zero if and only if the corresponding constraints are violated.

(4) This above example was shown for the inequality constraints in the output. The same relaxation technique can also be applied to other constraints in order to relax them.

Lecture 6
Output feedback MPC & state Estimation[7]

6.1 Need for state estimation:

In the Figure 4.1 (in lecture 4), you can see that in order to solve an optimal control problem at any time step j we need to know the initial value of the state of the process x_j at the current time. With the state feedback MPC in lecture 4, we made the assumption that all the states of the process/plant are available to us i.e. they are measurable.

But the question/problem is,

i. In a real process, sometimes it is not possible to measure all the states of the system. The reason could be simply that there are no sensors available for measuring some of the states of the process, or that the measurements are too noisy. Under such circumstances (which is very normal in real industrial processes), the assumption of having full state information simply becomes unrealistic.

ii. It is often very tempting to thing that, one way to calculate the states of the system x_j is to excite the process by the control input u^*_{j-1} (the first optimal control input obtained by solving optimal control problem at time $j-1$). We can then use the model of the process to calculate x_j by updating the process from *t=j* to *t=j+1* using the control input u^*_{j-1}.

   Computing x_j by using the model of the process (which is merely a mathematical representation of the physical process) does not account for errors in the model (e.g. modeling error and remember that models of a process are never 100% accurate). In addition, disturbances frequently occur in the system that may alter the states of the system, and in many case, the disturbances are not measurable and are unknown. Under such conditions, the predication of x_j using the model of the process may not be accurate and may deviate from the real value of the system states.

What is the solution to this problem?

$\Longrightarrow$ It is better and more accurate to ***estimate*** the states by using the latest available measurements from the real process. It makes sense to do so because the real measurements from the process are due to the actual dynamics occurring in the real process. The estimates of the states are usually computed using estimators and observers. Kalman filter is an example of a widely used optimal estimator. For linear MPC where the process model is linear, standard Kalman filter is used. For nonlinear systems, extended Kalman filter (EKF), unscented Kalman filter (UKF) etc. are used.

Note: If the model of the process is an input output model, then concept of state is irrelevant. The output of the system is a function of previous inputs and outputs. Quantities like y_0,

[7] In this course, we will not focus on the theoretical details of state estimators. This is outside the scope of this course. However, you will learn about how a state estimator can be used along with an MPC.

$y_{-1}, \ldots\ldots, u_{-1}, u_{-2} \ldots\ldots\ldots$, are measured directly. Therefore, there is no need for state estimation for such models.

6.2 Output feedback MPC

The algorithm in lecture 4 (state feedback MPC) assumed that the states of the system are perfectly known or measurable. However, in practice, it may not always be possible to know the state of the system perfectly. In many cases, we may not be able to measure the states of the system at all. In such cases, the states of the system should be "ESTIMATED". Estimation of the states can be performed by utilizing the measurement data up to and including time k. The use of the measurement data for state estimation along with predictive control will form an output feedback model predictive controller. The algorithm for output feedback MPC will be only a slight modification of the algorithm for state feedback MPC (i.e. with an addition of a state estimator). The algorithm for output feedback MPC is listed under:

1) Start with a given initial state of the process $x_k = x_0$ and set $t = k = 0$
2) Estimate the current state $\hat{x}_k$ for the current time k by using measurement data up to and including time k.
3) Considering a prediction horizon N, solve the optimal control problem (dynamic optimization). Use $\hat{x}_k$ as the initial state of the system.
   Compute $u_k^*, u_{k+1}^*, \ldots\ldots\ldots\ldots\ldots, u_{N+k-1}^*$
4) Use only the first control move u_k^* and discard the others. Update the system using u_k^*
5) Slide one time step forward, $k = k + 1$
6) Repeat steps (2) to (5) until the program terminates.

A general block diagram of the output feedback MPC is shown in Figure 6.1.

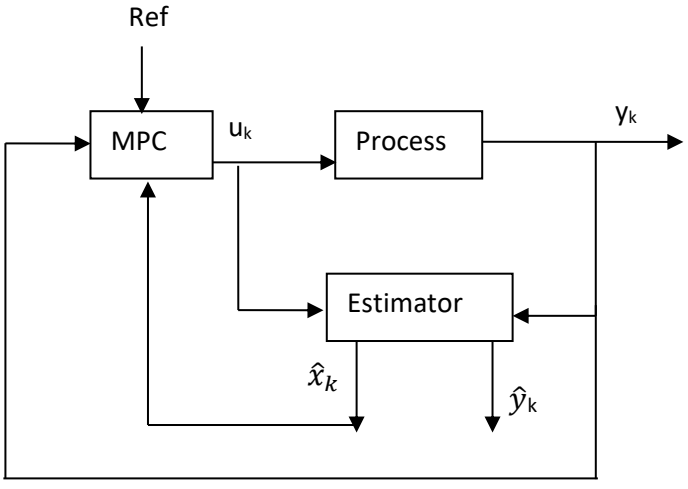


Figure 6.1 : Output feedback MPC

6.3 Brief introduction to estimators

Kalman filter can be used as optimal estimator for estimating the states of a system. In this course, use limit ourselves from the details of Kalman filter theories as it is not within the scope of this course. Therefore, we focus only on the implementation of the algorithms.

Kalman filter is a probabilistic state estimator, a type of Bayesian estimator. It is a minimum covariance estimator, thus optimal in this sense. In the algorithm of Kalman filter, we encounter terms such as "priori" and "posteriori" state estimates. Kalman filter describes the propagation of the mean of the states and propagation of covariance of the state estimation errors (hopefully minimum).

We proceed to estimate the states in two steps:

i) a priori estimate (predictor)

ii) a posteriori estimate (corrector)

Remember that we make use of the measurement data y_k's to calculate the estimates.

→ If the estimate is computed by using all measurement data up to time k (but not including time k), it is denoted as a priori estimate. We denote it by '–' in the superscript.

$$\hat{x}_k^- = E[x_k|y_1, y_2 \dots \dots \dots . y_{k-1}] = \textit{a priori estimate}.$$

Note that ^ denotes the estimated value. To obtain the apriori estimate, the mathematical model of the process is used. This is the **prediction step.**

→ Now we make use of the current measurement at time k obtained from the real plant, and then improve the apriori estimate obtained from the model. It is denoted as a posteriori estimate. We denote it by '+' in the superscript.

$$\hat{x}_k^+ = E[x_k|y_1, y_2 \dots \dots \dots . y_k] = \textit{a posteriori estamate}.$$

The estimate $\hat{x}_k^+$ is better than the estimate $\hat{x}_k^-$ because the current available measurement is also included when calculating $\hat{x}_k^+$ i.e. the latest new information about the measurement is utilized. This is the **correction step.**

The estimation error is calculated as,

$x_k - \hat{x}_k^-$ for a priori estimate

$x_k - \hat{x}_k^+$ for a posterior estimate

The covariance of the apriori estimation error (denoted by P_k^-) for, $\hat{x}_k^-$ is

$$P_k^- = E[\,(x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T]$$

The covariance of the aposteriori estimation error (denoted by P_k^+) for $\hat{x}_k^+$ is

$$P_k^+ = E[\,(x_k - \hat{x}_k^+)(x_k - \hat{x}_k^+)^T]$$

For easier understanding, see the time line of Figure 6.2 .

Figure 7.2: Time line for apriori and aposteriori state estimates

At the time step k, before using the measurement at that time k, we calculate an estimate of x_k (denoted by $\hat{x}_k^-$, the apriori estimate by using the mathematical model of the process) and covariance of the estimation error (denoted by P_k^-). The measurement data at the current time k is used to improve the state estimation. So, $\hat{x}_k^+$, the aposteriori estimate is calculated along with the covariance of the estimation error P_k^+ .

Note: What Kalman filter does is: Every time a measurement is available, it updates the predicted state and covariance of the state estimation error. As the time propagates, the estimated state will finally converge to the true value of the state. So Kalman filter is a recursive estimator.

6.4 Discrete time standard Kalman filter

Let us first study the algorithm of a Kalman filter for a linear state space model of the process given as,

$$x_{k+1} = Ax_k + Bu_k \ , \quad \text{with initial states } x_0 \text{ known} \tag{6.1}$$

$$y_k = Cx_k + Du_k \tag{6.2}$$

To make the process stochastic in nature, let us add process noise 'w_k' and measurement noise 'v_k' to the model.

$$x_{k+1} = Ax_k + Bu_k + w_k \ , \quad \text{with initial states } x_0 \text{ known} \tag{6.3}$$

$$y_k = Cx_k + Du_k + v_k \tag{6.4}$$

Assume that the process and the measurement noises are white noises with zero-mean, uncorrelated and have known covariance matrices W and V as,

$$W = E[w_k w_k^T]$$

$$V = E[v_k v_k^T]$$

These covariance matrices are used to tune the Kalman filter. The **algorithmic steps** of a discrete-time Kalman filter for a linear process are:

1. Initialize the Kalman filter for time $k = 0$. You can use the initial state of the system (known) for initialization.
   $\hat{x}_k^+ = \hat{x}_0^+ = E(x_0)$
   $P_k^+ = P_0^+ = E[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T]$
2. Compute the Kalman filter gain (K_f)
   $P_{k+1}^- = AP_k^+ A^T + W$
   $k_{f_k} = P_{k+1}^- C^T (CP_{k+1}^- C^T + V)^{-1}$

3. Compute a priori state estimate (**predictor**).
   $\hat{x}_{k+1}^{-} = A\hat{x}_k^{+} + Bu_k$

4. Compute output using a priori state estimate
   $\hat{y}_k^{-} = C\hat{x}_{k+1}^{-} + Du_k$
5. Compute a posteriori state estimate using the current measurement y_k at time k (**corrector**).
   $\hat{x}_{k+1}^{+} = \hat{x}_{k+1}^{-} + K_{f_k}\,(y_k - \hat{y}_k^{-})$

6. Update the covariance of the state.
   $P_{k+1}^{+} = \left(I - K_{f_k}\mathrm{C}\right) P_{k+1}^{-}\left(I - K_{f_k}C\right)^{T} + K_{f_k}VK_{f_k}^{T}$
7. Set $k := k+1$ i.e. set $P_k^{+} = P_{k+1}^{+}$ and $\hat{x}_k^{+} = \hat{x}_{k+1}^{+}$ & repeat step 2 to 6.

Important note:

In the above algorithm, as the iteration continues, the Kalman filter gain k_{f_k} will attain a steady state value and will remain constant. This is usually denoted as the steady state Kalman filter gain K_f. In many applications, it is okay to use the steady state Kalman filter gain right from the beginning. As a matter of fact it be calculated offline if you already know the A, B, C, D matrices of the linear state space model, and the process noise and measurement noise covariance matrics W and V. For example, in MATLAB, we can use the function *kalman* to calculate the steady state Kalman gain as,

```
[Kest,Kf,P]= kalman(plant,W,V);
```

Here, K_f is the steady state Kalman filter gain. $plant$ is the linear model of the process in discrete time domain. In MATLAB the following discrete linear model is used,

$$x_{k+1} = Ax_k + Bu_k + Gw_k \quad (6.5)$$
$$y_k = Cx_k + Du_k + Hw_k + v_k \quad (6.6)$$

To make the discrete time model used in MATLAB (equation 6.5 and 6.6) equivalent to our process model (equations 6.3 and 6.4), we have to set G to be unity and H to be zero as,

```
G = eye(nx);
H = zeros(ny,nx);
```

Here, $n_x =$ number of states and $n_y =$ number of outputs. We can then define $plant$[8] in MATLAB as,

```
Plant = ss(A,[B G],C,[D H],-1);
```

By using the steady state Kalman filter gain K_f, the algorithm reduces to the following,

$$\hat{x}_{k+1}^{+} = A\hat{x}_k^{+} + Bu_k + K_f\,(y_{meas} - C\hat{x}_k^{+} - Du_k), \text{ with initial value } \hat{x}_0^{+} \text{assumed}$$

[8] If the system matrix D is not present, simply put $D = zeros(n_y, n_u)$.

$\hat{x}_k^+ = \hat{x}_{k+1}^+$ updating the states.

Here, $\hat{x}_k^+$ is the estimated states, $\hat{x}_0^+$ is the initial values of the estimated states which you can assume something reasonable and y_{meas} is the measurement from the plant. So what you can see here is that a Kalman filter is infact a linear dynamic model (copy of the process model + the injection of the correction term). The correction term is the state estimation error which is then multiplied with the Kalman filter gain.

7.5 State estimation with nonlinear process model

If the system model is nonlinear, its state can be estimated using nonlinear versions of the Kalman filter. Extended Kalman filter (EKF) and Unscented Kalman Filter (UKF) are two widely used methods for estimating the states for nonlinear system.

EKF relies on linearization to propagate mean and covariance of the state. It assumes that a linearized transformation of means and covariance of the states are approximately equal to the true nonlinear transformation. However, the drawback of EKF is that the approximation could be unsatisfactory.

Therefore, in this lecture, we focus our attention to a form of Kalman filter which directly utilizes the nonlinear process model for state estimation. UKF is an example of such an estimator. Again we do not go into details about the theory of UKF but we limit ourselves to the algorithmic steps only (that can be coded for e.g. in MATLAB).

7.5.1 Unscented Kalman filter

Unscented Kalman filter is also probabilistic in nature and it is based on the concept of

Unscented transformation:

$\Longrightarrow$ Unscented transformation is performed using nonlinear process model (both for the state equations and for the measurement equations). A set of deterministic vector or points known as sigma points are used for transformation.

Let us assume that we know the mean of the states and the covariance of the state errors at the initial time of $k = 0$. (This is true, because at time k=0, we already know $\hat{x}_0^+$ and P_0^+ or at least we start with their assumed initial values)

The sigma points are generated in such a way that their ensemble mean and covariance at any time k is actually equal to $\hat{x}_k^+$ and P_k^+. These sigma points are then applied to the nonlinear process model, for e.g. $x_{k+1} = f(x_k, u_k)$ and $y_k = g(x_k, u_k)$ to obtain the transformed vectors.

Good thing is: Ensemble mean of the transformed vector will give a good estimate of the true mean. Covariance of the transformed vector will give a good estimate of the true covariance.

So, what we need to do is to:

a. Generate the sigma points.
b. Use the sigma points with nonlinear model of the process to find the transformed points or vectors.

c. Use the current/latest measurement from the process and the mean of the transformed points (or vectors) to find a good estimate of the states.

Details about sigma point generation is not within the scope of the course. Let us now look into the **algorithmic steps of UKF**.

First, let the nonlinear process model (stochastic) be given by,

$$x_{k+1} = f(x_k, u_k, \theta_k, t_k) + w_k \quad \text{nonlinear state equation with process noise } w_k \qquad (6.7)$$

$$y_k = g(x_k, u_k, \theta_k, t_k) + v_k \quad \text{measurement equation with measurment noise } v_k \qquad (6.8)$$

Here, $x \epsilon \mathbb{R}^{n_x}$ where n_x is the number of states. $f(.)$ is some nonlinear function of x_k (states), u_k (inputs), θ_k (parameters)and t_k (time). $g(.)$ is some non-linear function and v_k is the measurement noise at time k.

The process and the measurement noise are assumed to be white noises with zero mean, uncorrelated and that their covariances W_k and V_k are known. These covariances are used to tune the UKF. The **algorithmic steps** are:

1. Set the known initial mean $\hat{x}_k^+ = \hat{x}_0^+$ for time $k = 0$ and known initial covariance $P_k^+ = P_0^+$ for time $k = 0$ of the states of the system.
2. Generate $2n_x$ sigma points $\hat{x}_k^{(i)}$ as,

   $\tilde{x}^{(i)} = \left(\sqrt{n_x P_k^+}\right)_i^T , i = 1,2, \dots\dots\dots\dots, n_x$

   $\tilde{x}^{(n_x+i)} = -\left(\sqrt{n_x P_k^+}\right)_i^T , i = 1,2, \dots\dots\dots\dots, n_x$

   $\hat{x}_k^{(i)} = \hat{x}_k^+ + \tilde{x}^{(i)} , \ i = 1,2, \dots\dots\dots\dots, 2n_x$

   Note: Here $\left(\sqrt{n_x P_k^+}\right)_i$ is the i[th] row of $\sqrt{n_x P_k^+}$. Take the transpose after taking out the i[th] row.
3. Using the nonlinear model $f(.)$, perform the unscented transformation of the sigma points to find the transformed vector $\hat{x}_{k+1}^{(i)}$ of the states as,

   $$\hat{x}_{k+1}^{(i)} = f\left(\hat{x}_k^{(i)}, u_k, \theta_k, t_k\right) for \ i = 1,2, \dots\dots\dots\dots, 2n_x$$
4. Obtain a priori state estimate by finding the mean of the transformed vectors of the states,

   $$\hat{x}_{k+1}^- = \frac{1}{2n_x} \sum_{i=1}^{2n_x} \hat{x}_{k+1}^{(i)}$$
5. Obtain a priori error covariance and add the known process noise covariance matrix.

   $$P_{k+1}^- = \frac{1}{2n_x} \sum_{i=1}^{2n_x} \left(\hat{x}_{k+1}^{(i)} - \hat{x}_{k+1}^-\right)\left(\hat{x}_{k+1}^{(i)} - \hat{x}_{k+1}^-\right)^T + W_k \qquad (7.9)$$

   Note: Here W_k is added to take into account the process noise.
6. Using a priori state estimate $\hat{x}_{k+1}^-$ and a priori covariance P_{k+1}^-, generate $2n_x$ sigma points,

   $\tilde{x}^{(i)} = \left(\sqrt{n_x P_{k+1}^-}\right)_i^T , i = 1,2, \dots\dots\dots\dots, n_x$

$\tilde{x}^{(n_x+i)} = -\left(\sqrt{n_x P_{k+1}^-}\right)_i^T , i = 1,2, \ldots\ldots\ldots\ldots, n_x$

$\hat{x}_{k+1}^{(i)} = \hat{x}_{k+1}^- + \tilde{x}^{(i)} , \ i = 1,2, \ldots\ldots\ldots\ldots, 2n_x$

7. Using nonlinear measurement/output equation $g(.)$ of the model, transform the sigma points $\hat{x}_{k+1}^{(i)}$ to find transformed vector $\hat{y}_k^{(i)}$ of the outputs as,
   $\hat{y}_k^{(i)} = g\left(\hat{x}_{k+1}^{(i)}, u_k, \theta_k, t_k\right)$
8. Calculate the predicated output at time k by finding the mean of $\hat{y}_k^{(i)}$,
   $\hat{y}_k = \frac{1}{2n_x} \sum_{i=1}^{2n_x} \hat{y}_k^{(i)}$
9. Calculate the covariance of the predicated output and add the known measurement noise covariance matrix V_k.
   $$P_y = \frac{1}{2n} \sum_{i=1}^{2n_x} \left(\hat{y}_k^{(i)} - \hat{y}_k\right)\left(\hat{y}_k^{(i)} - \hat{y}_k\right)^T + V_k \quad (6.10)$$
   Note: Here V_k is added to take into account the measurement noises.
10. Obtain the cross covariance matrix between a priori states estimate $\hat{x}_{k+1}^-$, and measurements/output estimate $\hat{y}_k$.
    $P_{xy} = \frac{1}{2n_x} \sum_{i=1}^{2n_x} \left(\hat{x}_{k+1}^{(i)} - \hat{x}_{k+1}^-\right)\left(\hat{y}_k^{(i)} - \hat{y}_k\right)^T$
11. Use the current/latest measurement y_k from the real process to find the aposteriori states and convariance estimates.
    $K_{f_k} = P_{xy} P_y^{-1} \rightarrow Kalman\ gain$[9]
    $\hat{x}_{k+1}^+ = \hat{x}_{k+1}^- + K_{f_k}(y_k - \hat{y}_k)$
    $P_{k+1}^+ = P_{k+1}^- - K_{f_k} P_y K_{f_k}^T$
12. Set $k := k + 1$ i.e. i.e. set $P_k^+ = P_{k+1}^+$ and $\hat{x}_k^+ = \hat{x}_{k+1}^+$
13. Repeat steps (2) to (12) for $k = 1,2, \ldots \ldots$ until the end of simulation time.

6.6 Some comments

1. Process and measurement noises are assumed to be additive (linear) i.e. the process and measurement equations (equation (6.7) and equation (6.8)) are linear with respect to the noises. But noises may enter the process and measurement equations nonlinearly. If so, the nonlinear model of the stochastic system becomes,
   $$x_{k+1} = f(x_k, u_k, \theta_k, w_k, t_k)$$
   $$and\ y_k = g(x_k, u_k, \theta_k, v_k, t_k)$$
   For such models the UKF algorithm may not be very rigorous and accurate.

[9] For nonlinear Kalman filter, the steady state Kalman gain cannot be calculated offline or in advance.

2. **Augmentation**

   We can augment the noise with the state vector to handle such cases. Let us denote the augmented state (original system state + process and measurement noises) by $x_k^{(a)}$ → superscript (a) denotes augmented.

   $$x_k^{(a)} = \begin{bmatrix} x_k \\ w_k \\ v_k \end{bmatrix}$$

   Then for such augmentation, we can remove the terms W_k from equation (6.9) and the term V_k from equation (6.10). Rest of the algorithm remains the same as in section 6.5.1.

3. With an output feedback MPC, the tuning of MPC becomes more complicated. This is because of the presence of the estimation loop (e.g. Kalman filter for state estimation) which also has to be tuned. In general, the dynamics of the estimator should be significantly faster than the MPC loop to limit interaction between the estimator and the control loop.
4. There is also another class of state estimator known as *moving horizon estimator (MHE)* which has a structure very similar to an MPC structure. MHE estimates the states by solving an optimization problem formed by using the data upto M time steps backwards i.e at times $t_{k-M}, t_{k-M+1}, \ldots \ldots$ to current time t_k. The same solver that is used to solve MPC can also be used to solve MHE. Again, details are not included in this course.

6.7 Combined state and disturbance estimation

If disturbances act on the system we must ensure that the MPC will keep track of the reference despite the disturbance. For this, the information about the disturbance should be provided to the controller.

However, in many cases disturbances are not measurable and they are not known. In such cases, disturbances can be estimated together with the states of the system. The controller can then use these estimated disturbances.

One easy way to represent a disturbance is to assume that it is a constant signal acting on the process. Then a simple model of the disturbance can be of the form as,

$$d_{k+1} = d_k \tag{6.11}$$

where d denotes the disturbance acting on the system.

A linear state space model of the form

$$x_{k+1} = Ax_K + Bu_k + A_d d_k \tag{6.12}$$

$$y_k = Cx_k + C_d d_k \tag{6.13}$$

can be augmented with the disturbance model given by $d_{k+1} = d_k$ (a constant disturbance) as,

$$\underbrace{\begin{bmatrix} x_{k+1} \\ d_{k+1} \end{bmatrix}}_{\tilde{x}_{k+1}} = \underbrace{\begin{bmatrix} A & A_d \\ 0 & I \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x_k \\ d_k \end{bmatrix}}_{\tilde{x}_k} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{\tilde{B}} u_k$$

$$y_k = \underbrace{\begin{bmatrix} C & C_d \end{bmatrix}}_{\tilde{C}} \underbrace{\begin{bmatrix} x_k \\ d_k \end{bmatrix}}_{\tilde{x}_k}$$

Then the augmented state space model can be written as,

$$\tilde{x}_{k+1} = \tilde{A}\tilde{x}_K + \tilde{B}u_k \quad (6.14)$$

$$y_k = \tilde{C}\tilde{x}_k \quad (6.15)$$

In order to estimate the disturbance $\mathbf{d}_k$, the augmented state space model of equation (6.14) and (6.15) should be used with the Kalman filter algorithm of section 6.4 (but not the original state space model of equation (6.12) and (6.13)).

For nonlinear system, the disturbance model can be written as,

$$d_{k+1} = h(d_k, t_k) \quad \rightarrow \text{ non} - \text{linear function for input disturbance.} \quad (6.16)$$

Let the nonlinear process model be,

$$x_{k+1} = f(x_k, u_k, \theta_k, d_k, t_k) \quad (6.17)$$

$$y_k = g(x_k, u_k, \theta_k, d_k, t_k) \quad (6.18)$$

The disturbance and the system states can be augmented as,

$$x_{k+1}^{(a)} = \begin{bmatrix} x_{k+1} \\ d_{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} f(x_k, u_k, \theta_k, d_k, t_k) \\ h(d_k, t_k) \end{bmatrix}}_{\tilde{f}} \quad (6.19)$$

The augmented model is,

$$x_{k+1}^{(a)} = \tilde{f}\left(x_k^{(a)}, u_k, \theta_k, t_k\right) \quad (6.20)$$

$$y_k = g\left(x_k^{(a)}, u_k, \theta_k, t_k\right) \quad (6.20)$$

In order to estimate the disturbance $\mathbf{d}_k$, the augmented state space model of equation (6.20) and (6.21) should be used with the Unscented Kalman filter algorithm of section 6.5.1 (but not the original state space model of equation (6.17) and (6.18)).

6.8 Example and demonstration for state estimation:

Example and demonstration will be presented in the classroom using the real helicopter unit available at USN. The simulator for state estimation of the helicopter process is also available for download from the homepage of the course.

Students who were not present in the classroom, please see the video of the lecture to understand the example.

Lecture 7
Integral action and features of MPC

7.1 Offset free MPC: Integral action

When there is plant-model mismatch, i.e. if the model used for constructing the MPC is not the same as the plant model which is being controlled, then there will be some offset between the reference value and the actual plant output at the steady state. In other words, at the steady state, the output of the process being controlled will not become exactly equal to the setpoint value, but will show some deviation or offset. Two common scenarios where a plant-model mismatch may occur are given below:

a) When a linear MPC is used to control a nonlinear plant away from the point of linearization: In this case, the MPC is based on the linearized model of the plant. The linear plant model shows correct behavior only at or around the point of linearization. If the plant is operated at regions which lie away from the point of linearization, then the linearly approximated model may not be able to show the true nonlinear behavior of the plant, and hence the plant-model mismatch arises. Under such scenarios, linear MPC (based on the linearized model of the plant) when applied to the nonlinear plant may show finite steady state error or offset.

b) When an MPC (either linear or nonlinear) is applied to a plant having parametric uncertainty: In this case, the plant model (either linear or nonlinear) which is used to create the MPC (either linear or nonlinear) has parametric values that necessarily do not match the true parameters of the target system (the plant being controlled). Under such cases, the MPC will produce steady state error between the target system (actual plant) output and the reference value.

In an offset free MPC, steady state errors are driven to zero i.e. at the steady state the output of the process being controlled should become equal to the setpoint value. It is similar to adding integral action to the controller. Integral action is necessary to account for the plant-model mismatch or uncertainties in mathematical model of the process and/or to compensate for the effect of unknown disturbances acting on the system. We will see different ways of achieving integral action with MPC in this section.

Note: It may be very tempting to choose a large value for the weighting matrix for error i.e. a large value for the Q matrix in the objective function of the optimal control problem to reduce the offset due to plant-model mismatch. Well, with such a choice, the amount of offset may be decreased slightly, but it will never be able to eliminate the offset completely.

7.1.1 Augmentation with integrating constant nonzero disturbance model

One of the reasons why we may have a steady state error between the setpoint and the process output with an MPC is that various unknown disturbances may be affecting the process. The mathematical model of the process may not be sufficient to reflect the effect of these disturbances.

Thus a way of obtaining an offset free MPC for integral action is to augment the model of the process with an integrating constant nonzero disturbance model.

For simplicity, let us take the linear state space model of the process[10] as,

$$x_{k+1} = Ax_k + Bu_k \tag{7.16}$$

$$y_k = Cx_k \tag{7.17}$$

Now let us consider an integrating constant nonzero disturbance model as,

$$d_{k+1} = d_k \tag{7.18}$$

To incorporate the effect of disturbances, let us modify the linear state space model of the process as,

$$x_{k+1} = Ax_k + Bu_k + B_d d_k \tag{7.19}$$

$$y_k = Cx_k + C_d d_k \tag{7.20}$$

Now the process model of Equation 7.19 and 7.20 can be augmented with the disturbance model of Equation 7.18 as,

$$\underbrace{\begin{bmatrix} x_{k+1} \\ d_{k+1} \end{bmatrix}}_{\tilde{x}_{k+1}} = \underbrace{\begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x_k \\ d_k \end{bmatrix}}_{\tilde{x}_k} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{\tilde{B}} u_k \tag{7.21}$$

$$y_k = \underbrace{\begin{bmatrix} C & C_d \end{bmatrix}}_{\tilde{C}} \underbrace{\begin{bmatrix} x_k \\ d_k \end{bmatrix}}_{\tilde{x}_k} \tag{7.22}$$

The augmented model is in a standard linear state space form as,

$$\tilde{x}_{k+1} = \tilde{A}\tilde{x}_k + \tilde{B}u_k \tag{7.23}$$

$$y_k = \tilde{C}\tilde{x}_k \tag{7.24}$$

Here, $d_k \in \mathbb{R}^{n_d}$ with $n_d = n_y$ being the number of unmeasured disturbance variables and equal to the number of available measurement. The matrices $B_d \in \mathbb{R}^{n_x \times n_d}$ and $C_d \in \mathbb{R}^{n_y \times n_d}$ are chosen appropriately such that the following condition holds true for detectability.

$$rank \begin{bmatrix} I - A & -B_d \\ C & C_d \end{bmatrix} = n_x + n_y \tag{7.24}$$

Here n_x, n_y, n_d are the number of states, outputs and disturbance variables of the system.

The linear MPC (LQ optimal control structure + receeding horizon strategy) should be constructed with the augmented model of Equation 7.23 and 7.24. If the conditions for detectability holds true, then the resulting model predictive controller should produce offset free outputs i.e. integral action. Obviously, it is also clear that since the disturbances d_k are not measured, they should be estimated. Here the standard Kalman filter algorithm for linear system based on Equation 7.23 and 7.24 may be used to estimate the augmented states $\tilde{x}_k = \begin{bmatrix} x_k \\ d_k \end{bmatrix}$. This augmented state estimated at each time step should further be used as initial values for kronecker product formulation for LQ optimal control problems.

[10] In this lecture, I have dropped out deviation form (using δ) of the linear state space model for easiness in typing. But if your linear model is obtained by linearization of nonlinear model, then please make sure that your further calculations are based on the deviation form of your linear model.

7.1.2 $\Delta \boldsymbol{u}$ formulation for integral action

Another way to achieve integral action is the $\Delta \boldsymbol{u}$ formulation. It can be easily implemented to account for unknown disturbances which can be regarded as constant or slowly varying. The term $\Delta \boldsymbol{u}$ is defined as,

$$\Delta u_k = u_k - u_{k-1} \tag{7.25}$$

But at first, let us consider a linear system whose state space model[11] is given by,

$$x_{k+1} = Ax_k + Bu_k + v \tag{7.26}$$

$$y_k = Cx_k + w \tag{7.27}$$

Here, v and w account for the unknown disturbances (process and measurement noise respectively) which are either constant or slowly varying. To get rid of these unknown disturbances, we can define $\Delta x_k = (x_k - x_{k-1})$ or with one increment in k index as $\Delta x_{k+1} = (x_{k+1} - x_k)$. Then we have,

$$x_{k+1} - x_k = Ax_k + Bu_k + v - Ax_{k-1} - Bu_{k-1} - v \tag{7.28}$$

$$\Delta x_{k+1} = A(x_k - x_{k-1}) + B(u_k - u_{k-1}) \tag{7.29}$$

$$\Delta x_{k+1} = A\Delta x_k + B\Delta u_k \tag{7.30}$$

As, you can see in Equation 7.28, the effect of v gets cancelled out, i.e. $+v$ and $-v$ cancel out each other.

Similarly, for the output equation we have,

$$y_k - y_{k-1} = Cx_k + w - Cx_{k-1} - w \tag{7.31}$$

$$y_k - y_{k-1} = C(x_k - x_{k-1}) \tag{7.32}$$

$$y_k = y_{k-1} + C\Delta x_k \tag{7.33}$$

As, you can see in Equation 7.31, the effect of w gets cancelled out, i.e. $+w$ and $-$ w cancel out each other.

Now we can augment Equation 7.30 and Equation 7.33 as,

$$\underbrace{\begin{bmatrix} \Delta x_{k+1} \\ y_k \end{bmatrix}}_{\tilde{x}_{k+1}} = \underbrace{\begin{bmatrix} A & 0 \\ C & I \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} \Delta x_k \\ y_{k-1} \end{bmatrix}}_{\tilde{x}_k} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{\tilde{B}} \Delta u_k \tag{7.34}$$

$$y_k = \underbrace{\begin{bmatrix} C & I \end{bmatrix}}_{\tilde{C}} \underbrace{\begin{bmatrix} \Delta x_k \\ y_{k-1} \end{bmatrix}}_{\tilde{x}_k} \tag{7.35}$$

The augmented model is in a standard linear state space form as,

$$\tilde{x}_{k+1} = \tilde{A}\tilde{x}_k + \tilde{B}\Delta u_k \tag{7.36}$$

$$y_k = \tilde{C}\tilde{x}_k \tag{7.37}$$

The linear MPC (LQ optimal control structure + receeding horizon strategy) should be constructed with the augmented model of Equation 7.36 and 7.37. This way we achieve the integral action. Since the augmented linear model of the process now contains Δu_k instead of u_k, the formulation

[11] In this lecture, I have dropped out deviation form (using δ) of the linear state space model for easiness in typing. But if your linear model is obtained by linearization of nonlinear model, then please make sure that your further calculations are based on the deviation form of your linear model.

of the performance criteria for the MPC (i.e. the objective function) and the constraints should also be modified to contain Δu_k as,

$$\underset{(\Delta u_k)}{min} \quad J = \frac{1}{2}\sum_{k=1}^{N} e_k^T\, Q_k e_k + \Delta u_{k-1}^T\, P_{k-1}\Delta u_{k-1} \tag{7.38}$$

subject to,

$$\tilde{x}_{k+1} = \tilde{A}\tilde{x}_k + \tilde{B}\Delta u_k \quad \text{with } \tilde{x}_k \epsilon \mathbb{R}^{(n_x+n_y)\times 1}, \Delta u \epsilon \mathbb{R}^{n_u \times 1} \text{ and } \tilde{x}_0 \text{ known} \tag{7.39}$$

$$y_k = \tilde{C}\tilde{x}_k \quad \text{with } y_k \epsilon \mathbb{R}^{n_y \times 1} \tag{7.40}$$

$$e_k = r_k - y_k \tag{7.41}$$

$$\Delta u_L \leq \Delta u_k \leq \Delta u_U \tag{7.42}$$

$$u_L \leq u_k \leq u_U \tag{7.43}$$

If any of the augmented states $\tilde{x}_k = \begin{bmatrix} \Delta x_k \\ y_{k-1} \end{bmatrix}$ are not measurable then they should be estimated using state estimators like the Kalman filter and used further in kronecker product formulation of LQ optimal control problems.

Here the variables to be optimized are the rate of change of control input Δu_k. However, when applying the control input to the target system, $u_k = \Delta u_k + u_{k-1}$ should be used.

7.1.3 Adding integrators to the output (MPC + I control): A practical approach

In practice, it is much easier to achieve integral action (or zero steady state offset) by directly adding integrators to the measured outputs. The block diagram in Figure 7.4 illustrates the concept. Here, integrator have been added to the outputs such that,

$$u_{int} = K_i \int (y_{meas} - y_{SP})dt \tag{7.44}$$

In Equation 7.44, the error between the measured output (y_{meas}) and the setpoint (y_{SP}) is integrated over time. K_i is the integrator gain chosen by the user.

If u is the control inputs obtained from the linear MPC structure, then the actual control input applied to the process is,

$$u_{app} = u + u_{int} \tag{7.44}$$

$$u_{app} = u + K_i \int (y_{meas} - y_{SP})dt \tag{7.45}$$

The Kalman filter block is used to estimate the unknown states. The MPC block (either linear or nonlinear) contains the optimal control with receeding horizon strategy. An important thing to note here is that the output integrator is not a direct part during the formulation of the optimal control problem. Thus, it can be argued that the applied control inputs u_{app} are sub-optimal instead of being optimal. In practice this does not matter since in an MPC, the optimization is carried out not just once but at every time step thus obtaining feedback action. Given that the user chooses proper value of the integrator gain (K_i), the question of sub-optimal solution (at least for the set point tracking problems) becomes not very important.

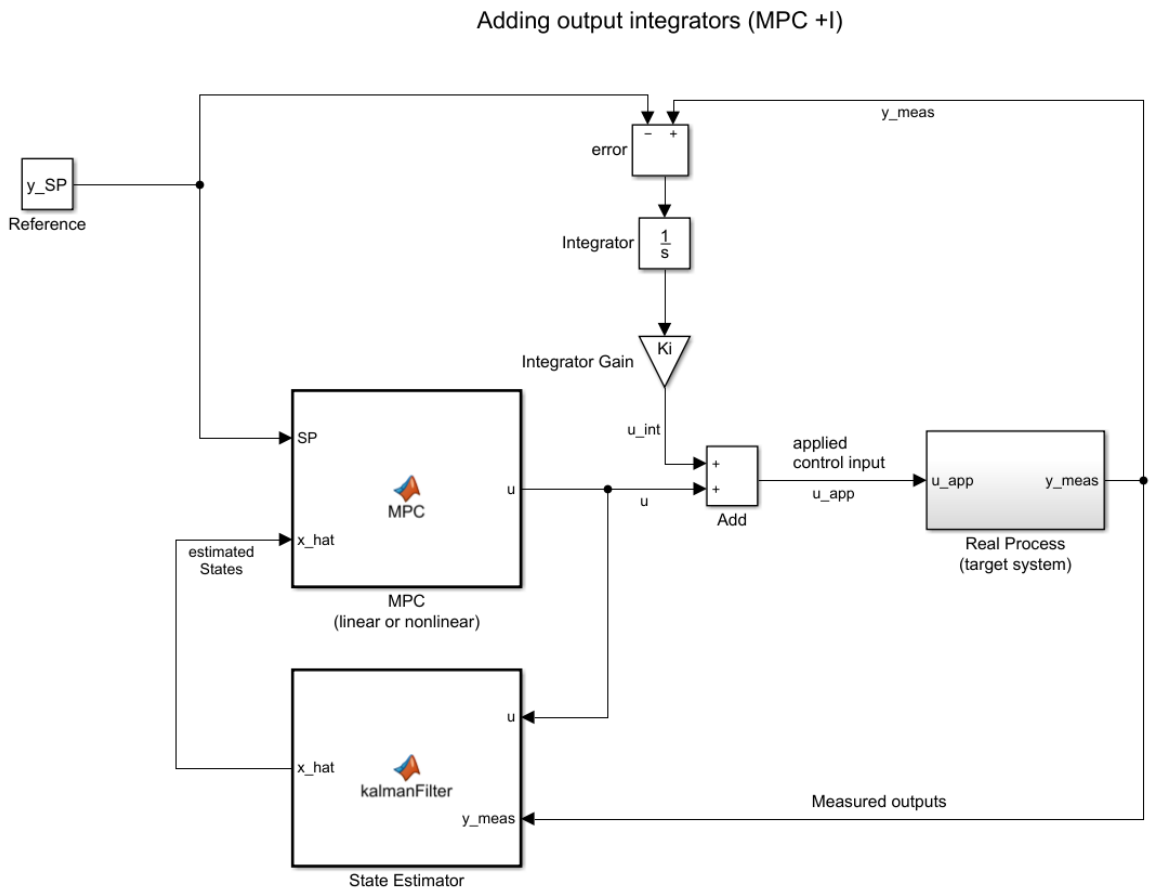


Figure 7.4: Adding integrator to the outputs with MPC for integral action

7.2 Features of MPC:

A big advantage of MPC is that a systematic approach for handling constraints is included in the design of MPC itself.

Constraints such as,

- Physical constraints, e.g. actuator limits.
- Safety constraints, e.g. temperature/pressure limits.
- Environmental constraints, quality constraints
- Performance constraints e.g. overshoot.

are integrated as a part of the optimal control problem of MPC. i.e. while formulating the optimization problem, you can in a natural way also include constraints.

MPC is an optimal controller (because we solve an optimization problem at each time step). The control actions generated by an MPC are optimal in nature. Upon the usage of these control actions, the process operates in an optimal manner.

Classical controller (e.g. PID controller) do not necessarily operate a process/plant in an optimal manner. The operation of the plant may be just suboptimal. Classical control methods may not be able to handle constraints efficiently, even if they do, it will be unsystematic of adhoc constraint management.

Another feature of MPC is that we can include different kinds of models such as,

- Linear
- nonlinear
- SISO (Single input Single output)
- MIMO (Multiple Input Multiple Output) with strong coupling
- Models with constraints
- Models with time delays.
- Models with inverse response

MPC can be formulated to include many different kinds of objectives (or performance criteria). They could for e.g. be:

- sum of squared error $\sum e_K^T Q e_k$,
- sum of absolute errors $\sum \lambda_e |e_k|$,
- economic objectives like profit maximizing ,
- minimizing loss or minimizing production downtime,
- multi objective function etc.

Note: MPC not only can be used for tracking a reference, but it can also be used as an optimizer. MPC can be designed to calculate optimal set points (for e.g. flow rates, pump speed, value opening, temperature, pressure etc.). These optimal set points can be utilized by lower level controllers (e.g. PID controllers) for operating the process. MPC providing set points to lower level controllers is also known as a supervisory MPC.

Therefore, in this regard, we can say that an MPC is a high performance controller. However, there are many challenges that arise when a finite horizon MPC is used as a controller (or as an optimizer). Some of the challenges are:

i. **Feasibility**

There is no guarantee that an optimization problem will have a feasible solution at all times. At any point of time, we may not find a solution to the optimization problem that will satisfy all the constraints. This could (for example) be because of a large disturbance acting on the system, say at time t_k that may cause infeasiblity at the time t_k and possibly also at future time steps t_{k+1}, t_{k+2} etc.

ii. **Stability**

In a finite horizon MPC, closed-loop stability is not guaranteed i.e. the closed-loop performance of the MPC may not converge. So, even if the process/plant is open-loop stable, the closed-loop stability with MPC is not guaranteed.

iii. **Robustness**

Under the presence of uncertainties or disturbances, the performance of MPC is not necessarily robust.

iv. **Implementation**

MPC problems are computationally demanding in general. If MPC is used for real time applications, the MPC problem has to be solved within the sampling interval of the process.

7.3 Stability of MPC

As also mentioned before, a finite horizon MPC cannot guarantee stability even if a feasible solution exists at every time step. Let us write a finite horizon MPC in general as,

$$(A)\begin{cases} \underset{(u_0,u_1 \dots \dots \dots u_{N-1})}{min} \quad J = \sum_{k=0}^{N-1} q(x_k,u_k) \\ s.t. \quad x_{k+1} = f(x_k,u_k)\ ,\ k = 0,1,2,\dots\dots\dots,N-1 \rightarrow nonlinear\ process\ model, \qquad x_0\ known \\ \quad y_k = g(x_k,u_k)\ ,\ k = 0,1,2,\dots\dots\dots,N-1 \rightarrow measurement\ equations \\ \quad h_i(x) - b_i = 0\ ,\ i = 1,2,\dots\dots\dots,m \rightarrow nonlinear\ equality\ constraints \\ \quad I_j(x) - c_j \leq 0\ ,\ j = 1,2,\dots\dots\dots,r \rightarrow nonlinear\ inequality\ constraints \end{cases}$$

Here, $q(x_k,u_k)$ is any linear or nonlinear objective. The stability of the finite horizon constrained optional control problem of (A) can be improved by making the horizon longer. As a rule of thumb, the prediction horizon should be at least equal to the dominant dynamics (for e.g. , the longest time constant of a stable system). However, to ensure stability, the prediction horizon can be extended to infinity, i.e. by considering an infinite horizon MPC[12]. So, what we would like the MPC to be is given by problem (B),

$$(B)\begin{cases} \underset{(u_0,u_1 \dots \dots \dots \dots \infty)}{Min} \quad J = \sum_{k=0}^{\infty} q(x_k,u_k) \\ s.t. \quad x_{k+1} = f(x_k,u_k)\ ,\ k = 0,1,2,\dots\dots\dots,\infty\ ,x_0\ known \\ \quad y_k = g(x_k,u_k)\ ,\ k = 0,1,2,\dots\dots\dots,\infty \\ \quad h_i(x) - b_i = 0\ ,\ i = 1,2,\dots\dots\dots,m \\ \quad I_j(x) - c_j \leq 0\ ,\ j = 1,2,\dots\dots\dots,r \end{cases}$$

But infinite horizon constrained optimal control problem of (B) is infinite dimensional optimization problem. It cannot be computed or solved using conventional method because there are infinite number of variables to optimize. At the same time, the objective function should also be bounded i.e. $J < \infty$.

The solution to the problem:

The idea is to split the infinite horizon of problem (B) into two parts:

$0,1,\dots\dots,N-1,$: part 1
$N,N+1,\dots\dots\dots,\infty$: part 2

Splitting is done in a way that the first part $(0,1,\dots\dots,N-1)$ will represent a finite horizon control problem. On the second part $(N,N+1,\dots\dots\dots,\infty)$, the problem is unconstrained. Then we have,

[12] The openloop trajectories of an infinite horizon optimal control problem is the same as the closed loop trajectories. So, if the problem is feasible, the closed loop trajectories will be always feasible. If the problem is stable, the closed loop stability is guaranteed.

$$(C)\begin{cases} \underset{(u_0, u_1 \dots\dots\dots\dots u_\infty)}{min} \quad J = \sum_{k=0}^{N-1} q(x_k, u_k) + \sum_{k=N}^{\infty} q(x_k, u_k) \\ S.t. \quad x_{k+1} = f(x_k, u_k) \ , \ k = 0,1,2, \dots\dots\dots, \infty \\ y_k = g(x_k, u_k) \ , \ k = 0,1,2, \dots\dots\dots, \infty \\ h_i(x) - b_i = 0 \quad , \quad i = 1,2, \dots\dots\dots, m \\ I_j(x) - c_j \le 0 \quad , \quad j = 1,2, \dots\dots\dots, r \\ x_0 \ known \end{cases}$$

N should be large enough to drive the system/process to a state $x_N \in \chi_f$ where χ_f is a feasible set from where the optimal trajectory is unconstrained. In other words, in the infinite horizon interval $(N, N+1, \dots\dots, \infty)$, none of the constraints are active (both in states and in control inputs). The objective function in the infinite horizon interval is for simplicity can be written in quadratic form as

$$\sum_{k=N}^{\infty} q(x_k, u_k) = p(x_N) = \frac{1}{2} x_N^T S x_N$$

where S is the weighting matrix for x_N. The term $p(x_N)$ is also referred to as terminal cost or terminal objective function. The term $x_N \in \chi_f$ will give rise to terminal constraints. Then problem (C) which is infinite horizon constrained problem can be reformulated as finite horizon constrained problem or finite dimensional control problem (D) as,

$$(D)\begin{cases} \underset{(u_0, u_1 \dots\dots\dots u_{N-1})}{Min} \ J = \sum_{k=0}^{N-1} q(x_k, u_k) + p(x_N) \\ S.t. \quad x_{k+1} = f(x_k, u_k) \ , \ k = 0,1,2, \dots\dots\dots, N-1 \\ y_k = g(x_k, u_k) \ , \ k = 0,1,2, \dots\dots\dots, N-1 \\ h_i(x) - b_i = 0 \quad , \quad i = 1,2, \dots\dots\dots, m \\ I_j(x) - c_j \le 0 \quad , \quad j = 1,2, \dots\dots\dots, r \\ x_0 \ known \\ x_N \in \chi_f \quad \rightarrow \ terminal \ constraint \end{cases}$$

Note that in the optimal control problem (D), $U \in [u_0, u_{1,} \dots\dots, u_{N-1}]$ i.e. we have finite number of variables to optimize. By introducing terminal cost and terminal constraint, stability is ensured (of course for a proper choice of N) provided that the problem is feasible at all times.

χ_f is a set or region where the optimal control problem becomes unconstrained in the infinite horizon interval. $p(x_N)$ and χ_f are chosen to mimic or approximate infinite horizon problem with finite horizon problem.

Note: For a proper choice of N, it can be proven that the terminal objective function $p(x_N)$ becomes a Lyapunov function for the whole horizon that decays toward the origin i.e. stability is guaranteed. The proof involves a good choice of N and the generation of χ_f is a difficult task (especially for nonlinear system) and it is out of the scope of the course. Generally, in practice, terminal sets are not used i.e. simply neglected as long as there is feedback in the system.

7.4 Handling computational time delay and burden

MPC algorithms are computationally demanding. The algorithm is suitable for systems whose sampling time is larger than the computational time required for solving the MPC problem at each time step i.e. for slow processes. For processes with fast system dynamics, real time application of MPC may be difficult to realize. Failing to properly address the computational time delay, may lead the process to become unstable.

7.4.1 As input and output delay

If it is possible[13] to define an upper bound 'τ' on the computation time (i.e. to define the maximum time required to solve MPC problem), then computational delay time ' τ ' can be handled in two ways.

(i) Consider the computational delay $'\tau'$ as an input delay to the model i.e.

$$u(t_k) \rightarrow u(t_k - \tau)$$

To do so, the model used in the MPC algorithm must be extended with the delay. This simply means that when the control signal has been computed for time t_k, it is injected or used into the system at time $t_k + \tau$. The model must be able to handle the input delay. This technique improves the closed loop response.

(ii) Consider the computational delay $'\tau'$ as an output delay:

If dt is the sampling time, then the number of samples corresponding to the computational delay can be calculated as,

$$n_\tau = \text{round down}\left(\frac{\tau}{dt}\right)$$

Figure 7.5 shows a simplified diagram of an output delay.

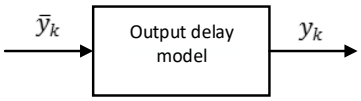


Figure 7.5: Output delay model

Without the output delay, the state space model of the process can be written as,

$$x_{k+1} = Ax_k + Bu_k \quad (7.46)$$

$$\bar{y}_k = Dx_k \quad (7.47)$$

Let us assume that the state space model from $\bar{y}_k$ to y_k (i.e. the delay model) be written as,

$$x^\tau_{k+1} = A^\tau x^\tau_k + B^\tau \bar{y}_k \quad (7.48)$$

$$y_k = D^\tau x^\tau_k \quad (7.49)$$

Illustration 1: For a computational delay of 2 samples i.e. $n_\tau = 2$,

[13] The difficulty in determining the computational delay offline lies in that the computational times for different optimization cycles are different even for the same optimization algorithm. Computational time is quite different in each nonlinear optimization procedure for different initial points and optimum.

$$\begin{bmatrix} x_{k+1}^1 \\ x_{k+1}^2 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}}_{A^\tau} \underbrace{\begin{bmatrix} x_k^1 \\ x_k^2 \end{bmatrix}}_{x_k^\tau} + \underbrace{\begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{B^\tau} \bar{y}_k \tag{7.50}$$

$$y_k = \underbrace{\begin{bmatrix} 0 & 1 \end{bmatrix}}_{D^\tau} \underbrace{\begin{bmatrix} x_k^1 \\ x_k^2 \end{bmatrix}}_{x_k^\tau} \tag{7.51}$$

Analysing equations (7.50) and (7.51) we get,

$$x_{k+1}^1 = \bar{y}_k$$

$$x_{k+1}^2 = x_k^1 = \bar{y}_{k-1}$$

$$y_k = x_k^2 = \bar{y}_{k-2}$$

i.e. there is an output delay of 2 samples.

<u>Illustration 2:</u> For a computational delay of 3 samples i.e. $n_\tau = 3$,

$$\begin{bmatrix} x_{k+1}^1 \\ x_{k+1}^2 \\ x_{k+1}^3 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{A^\tau} \underbrace{\begin{bmatrix} x_k^1 \\ x_k^2 \\ x_k^3 \end{bmatrix}}_{x_k^\tau} + \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}}_{B^\tau} \bar{y}_k \tag{7.52}$$

$$y_k = \underbrace{\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}}_{D^\tau} \underbrace{\begin{bmatrix} x_k^1 \\ x_k^2 \\ x_k^3 \end{bmatrix}}_{x_k^\tau} \tag{7.53}$$

Analysing equations (8.7) and (8.8) we get,

$$x_{k+1}^1 = \bar{y}_k$$

$$x_{k+1}^2 = x_k^1 = \bar{y}_{k-1}$$

$$x_{k+1}^3 = x_k^2 = \bar{y}_{k-2}$$

$$y_k = x_k^3 = \bar{y}_{k-3}$$

i.e. there is an output delay of 3 samples.

To handle the computational time delay, the process model without output delay (equation 7.46 & 7.47) should be augmented with the output delay model (equation 7.48 & 7.49) as,

$$\underbrace{\begin{bmatrix} x_{k+1} \\ x_{k+1}^\tau \end{bmatrix}}_{\tilde{x}_{k+1}} = \underbrace{\begin{bmatrix} A & 0 \\ B^\tau D & A^\tau \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x_k \\ x_k^\tau \end{bmatrix}}_{\tilde{x}_k} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{\tilde{B}} u_k \tag{7.54}$$

$$y_k = \underbrace{[0 \quad D^\tau]}_{\widetilde{D}} \underbrace{\begin{bmatrix} x_k \\ x_k^\tau \end{bmatrix}}_{\tilde{x}_k} \tag{7.55}$$

Augmented model is in a standard form as,

$$\tilde{x}_{k+1} = \widetilde{A}\tilde{x}_k + \widetilde{B}u_k \tag{7.56}$$
$$y_k = \widetilde{D}\tilde{x}_k \tag{7.57}$$

The draw back with this idea is that additional states in x_k^τ are involved which makes the size of the control problem bigger than before.

7.4.2 Sampled-data MPC

In sampled-data MPC, the open-loop optimal control problem is only solved at the *discrete* recalculation time instants[14] as shown in Figure 7.6. The recalculation time instants should be at least equal to (or greater than) the computational delay. In other words, the recalculation time instants can be many times larger than the sampling time dt of the system.

Let us assume that we are going to solve the following MPC problem with the objective function as,

$$\min_u \; J = \sum_{k=1}^{N} x_k^T Q x_k + u_{k-1}^T P u_{k-1} \tag{8.58}$$

When an open-loop optimal control problem is solved at any recalculation time t_k, it will take certain time (computational delay time, τ) to solve it i.e. the problem will be solved only at $t_k + \tau$. Between this time $[t_k \quad t_k + \tau]$, the solution (optimal control input) is not available because the optimal control problem is still being solved. Thus, it makes no sense to optimize the control profile between the time $[t_k \quad t_k + \tau]$ from time t_k. Instead, at time t_k, it makes sense to optimize the control sequence for the time $[t_k + \tau \quad t_k + \tau + N]$. If n_τ is the number of samples corresponding to the computational delay , then at time t_k, optimal control problem with the modified objective function is solved as,

$$\min_u \; J = \sum_{k=n_\tau}^{N} x_k^T Q x_k + u_{k-1}^T P u_{k-1} \tag{8.59}$$

To formulate this modified objective function, the initial value of the state at $t_k + \tau$ is needed and thus this is predicted as $x(t_k + \tau)$. Then this problem is started to be solved at t_k. However, the solution of this problem is only available at $t_k + \tau$. During this time $[t_k \quad t_k + \tau]$, the optimal input signal calculated at the previous recalculation time instant $t_k - \tau$ is applied to the system as shown in Figure 7.6.

[14] If the open-loop optimal control problem is solved at each time step, then it is called instantaneous MPC.

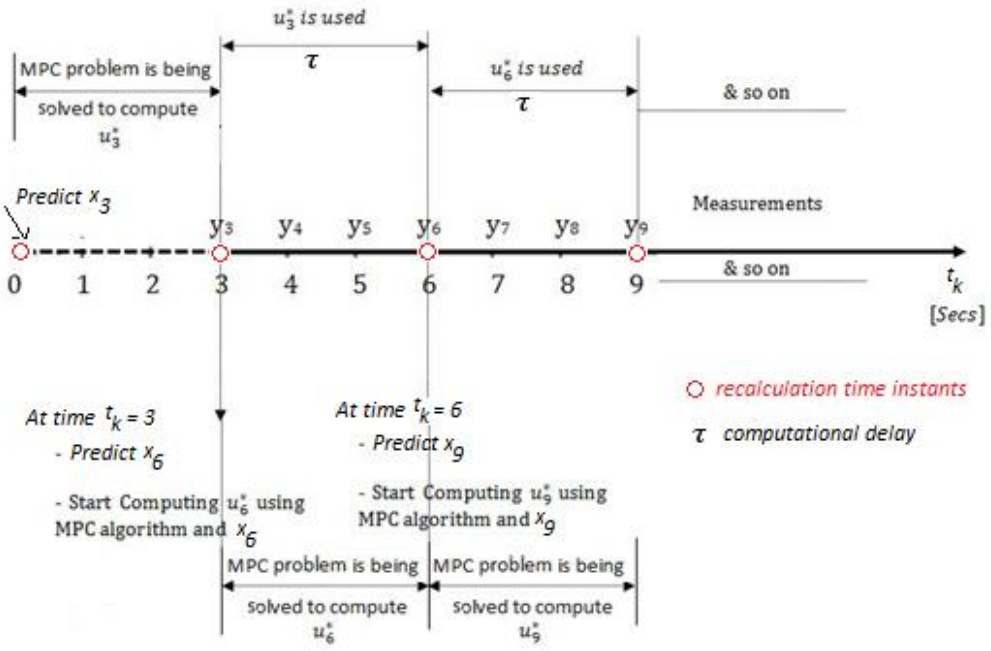


Figure 7.6: Sampled-data MPC timeline

For explanation, let, $\tau = 3\ seconds$ is the highest computational time delay. Let us for simplicity also assume that the recalculation time instant is equal to the computational time delay. Let $dt = 1$ second be the sampling time or step length.

Let us assume that the current time is $t_k = 3$ seconds. The value of the states at $t_k + \tau = 6$ second is predicted i.e. x_6 is predicted at the time $t_k = 3$ second. Using x_6 as the initial state, at current time $t_k = 3$ seconds, an optimal control problem is formulated to compute u_6^*. The optimal control inputs u_6^* will be available only at the next recalculation time instant ($t_k = 6$ second). While the calculation of u_6^* is still going on (for the time between 3 and 6 seconds), the optimal control input u_3^* computed at the previous recalculation time instant ($t_k = 0$ sec) is used. This is repeated at each recalculation time instant as shown in Figure 7.6.

7.5 Hierarchy of control and optimization system

There are many layers inter-communicating with each other that make up a hierarchy of a control and optimization system. Figure 7.7 shows such a hierarchy with the information and function of each layer.

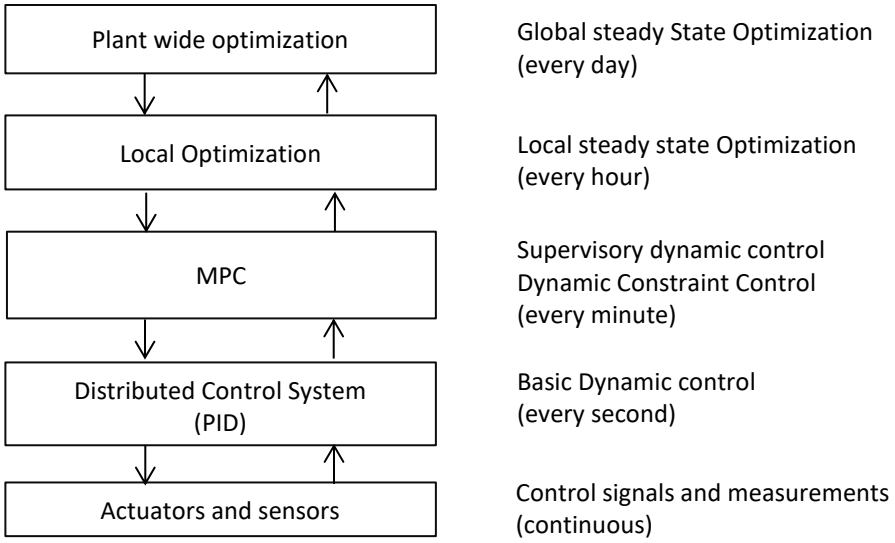


Figure 7.7: Control and optimization hierarchy of a system

The MPC layer (with is often a part of the Advanced Process Control layer) can serve as a supervisory dynamic control and dynamic constraint control. It can provide optimal setpoints to the layer below it. In the basic dynamic control layer, there are classical controllers (e.g. PID controllers) which control process variables like pressure, temperature, flow rates, speed etc. In industrial applications, the lower level classical controllers are a part of the distributed control system (DCS) or Programmable logic controllers (PLC). These controllers in the DCS or PLC system then send control signals to the actuators (e.g. valves, motor, pumps, compressor etc.). They also read measurements from the sensors and provide it to the upper levels. The local optimization layer which performs local steady state optimization (for example of a specific unit of the industrial process) can interact with the MPC to provide/receive information. The communication method between the layers can be for example the Open Process Control (OPC) protocols.

7.6 Commercial predictive controllers

It is difficult to give an exhaustive overview of commercially available model predictive controller. It is even more difficult to provide detailed information about where and how have they been used for industrial applications. Nevertheless a list of companies and their commercial MPC product have been listed in Table 1 (for linear MPCs) and Table 2 (for nonlinear MPC). These tables have been extracted from the article written by Qin and Badgwell [1].

Table 1: Companies and products included in Linear MPC technology

Company	Product Name	Description
Adersa	HIECON	Hierarchical constraint control
	PFC	Predictive functional control
	GLIDE	Identification package
Aspen Tech	DMC-plus	Dynamic matrix control package
	DMC-plus model	Identification package
Honeywell	RMPCT	Robust model predictive control technology
Shell Global solutions	SMOC-II	Shell multivariable optimizing control
Invensys	Connoisseur	Control and Identification package
ABB	3d MPC	Three dimensional model predictive control

Table 2: Companies and products included in Nonlinear MPC technology

Company	Product Name	Description
Adersa	PFC	Predictive functional control
Aspen Tech	Aspen Target	Nonlinear MPC package
Continental Controls Inc.	MVC	Multivariable control
DOT Products	NOVA-NLC	NOVA nonlinear controller
Pavilion Technologies	Process Perfecter	Nonlinear control
Statoil	SEPTIC	Statoil's Estimation and Prediction tool for identification and control

[1] Qin, S.J. and Badgwell, T. A., "A survey of industrial model predictive control technology", *Control Engineering Practice,* Vol. 11, pp. 733-764, 2003.

Lecture 8

Nonlinear optimal control, Nonlinear MPC

8.1: Goal

In this chapter, we will formulate a nonlinear optimal control problem as a general nonlinear optimization problem. We will then solve the nonlinear optimization problem using the solver *fmincon* in MATLAB[15]. The nonlinear model of the process will be directly utilized to make the nonlinear optimization problem i.e. there is no necessity for performing linearization. Thus, formulation and the solution of the nonlinear optimization problems is the backbone for nonlinear optimal control problems.

Secondly, to create a nonlinear MPC, receeding horizon strategy can be utilized to the nonlinear optimal control problem.

8.2 Well structured problem formulation?

It is difficult to create well structured matrices for nonlinear optimal control problem. In fact, it is very difficult (almost impossible) to generalize a well defined structure for nonlinear optimal control problems and to solve them analytically. This is due to the presence of nonlinearities in the system model, which cannot be generalized. So, unlike the linear case (in lecture 3) where Kronecker product formulation resulted in a specific well structured QP problem formulation, for the nonlinear case, such a structure is absent.

In other words, we cannot transform a nonlinear optimal control problem to a specific well structured NLP by making use of nonlinear algebra. Instead we have to treat a nonlinear optimal control problem as a general mathematical nonlinear optimization problem, and rely on nonlinear optimization solvers to obtain its solution. Since we rely heavily (almost completely) on the nonlinear solver, the accuracy of the solution and how fast we can solve a nonlinear MPC depends very much on the ability of the chosen solver/algorithm.

However, before we begin to formulate the nonlinear optimal control problem, let us first have the basic understanding about different terms associated with nonlinear optimization.

8.3 Nonlinear Programming (NLP)

8.3.1 Basic Introduction

A general non-linear optimization problem can be written as,

$$\text{(A)} \quad \begin{cases} \underset{x}{Min}\, f(x) \;,\; x \epsilon R^n & \rightarrow \text{Objective function} \\ h_i(x) = b_i \quad , i = 1, 2, \dots\dots\dots\dots, m & \rightarrow' m' \text{ number of equality constraints} \\ g_j(x) \leq c_j \quad , j = 1, 2, \dots\dots\dots\dots, r & \rightarrow 'r' \text{ number of inequality constraints} \end{cases}$$

[15] We will not use graphical tool like Simulink for solving Nonlinear optimization problems in this course. Instead we will use scripting language like MATLAB. You can then modify the MATLAB scripts for Python, Julia etc.

Here $x\epsilon\mathbb{R}^n$ is the n number of decision variables (the variable that should be optimized or the unknown variables that should be calculated by the optimizer).
Note: If the bounds on the decision variables i.e. $x_L \leq x \leq x_H$ is present, it can be easily expressed as two inequality constraints $x \leq x_H$ and $-x \leq -x_L$. So, it is assumed that such bounds are already the part of the general inequality constraint $g_j(x) \leq c_j$.
For a nonlinear programming problem one or more of the function $f(x)$, $h_i(x)$ & $g_i(x)$ are nonlinear. In other words if any of the $f(x)$, $h_i(x)$ & $g_i(x)$ are non-linear functions, then the problem (A) is a NLP problem (example: problem with a quadratic objective and one or more of the constraints being nonlinear is also a NLP).
By solving the optimization problem of (A), we try to find the optimal value of the decision variables x. We denote the optimal values as x^*.

A value of x that satisfy all the constraints of (A) is known as a feasible solution. A group of or a set of such feasible points will make a feasible region denoted by Ω .

Example:
Consider the optimization problem,

$$\underset{x\epsilon\mathbb{R}^2}{Min}\ f(x) = x_1^2 + 3x_2$$

s.t.

$$h_1(x) = x_1^2 + x_2^2 - 1 = 0$$
$$g_1(x) = x_1 + x_2 \leq 0$$

The feasible region is,

$$\Omega = \{x\epsilon\mathbb{R}^2 | x_1^2 + x_2^2 = 1 \ \wedge \ x_1 + x_2 \leq 0\}$$

The feasible region Ω is half circle with radius 1 (shown with bold line in Figure 8.1)

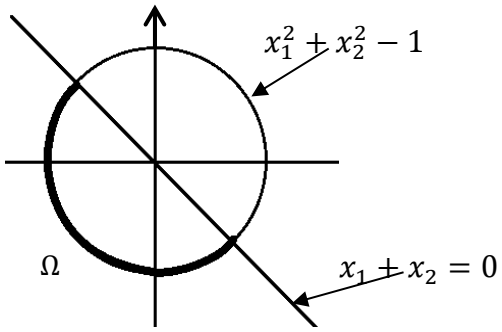


Figure 8.1: Illustration of a feasible region

8.3.2 Global and local minimum:

A point $x^* \epsilon\ \Omega$ is called a global solution if $f(x^*) \leq f(x)$ for all , i.e. for all x in the feasible region Ω . Point b is the global solution in Figure 8.2.

But many such minimum points or valley may be present in the entire space of . A point $x^* \epsilon\ \Omega$ is called a local solution if

$$f(x^*) \leq f(x) \text{ for all } x\ \epsilon\ \|x - x^*\| < \epsilon$$

i.e. for all x in the neighborhood around x^* rather than the whole feasible region.

Note: *fmincon* solver in MATLAB is a local solver i.e. it finds only the local optimal solutions and global solution is not guaranteed.

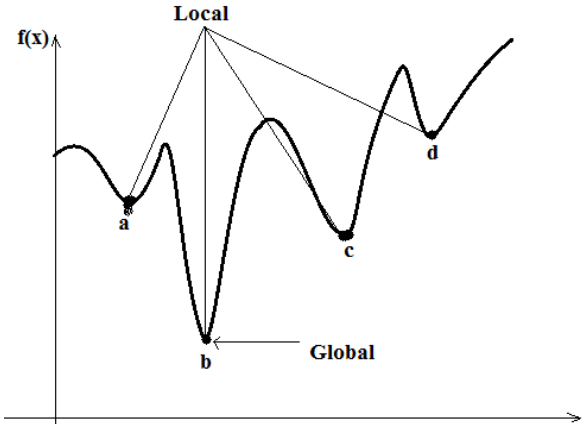


Figure 8.2: Illustration of local and global minima

8.3.3 Convexity

An optimization problem is convex iff:

a) The objective function is a convex function.

b) The feasible set or region Ω is a convex set.

A convex function cannot have any values larger than the values of a linear function (line) between two points *a* & *c* (any two points on the function).

For any point that lies between *a* and *c*, (for e.g. point *b*) in Figure 8.3,

$$f(b) \leq f_l(b)$$

where f(b) is the value of the convex function at point *b*. $f_l(b)$ is the value of the linear function (line) at point *b*.

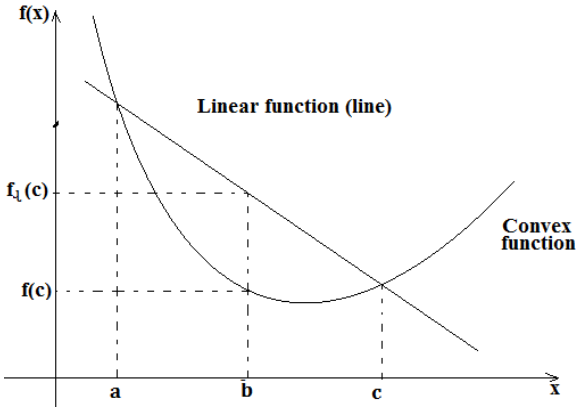


Figure 8.3: Example of a convex function

If the optimization problem is a convex optimization problem, there will be only one minimum point and this point is the global minimum point.

An example of a non-convex function is shown below in Figure 8.4:

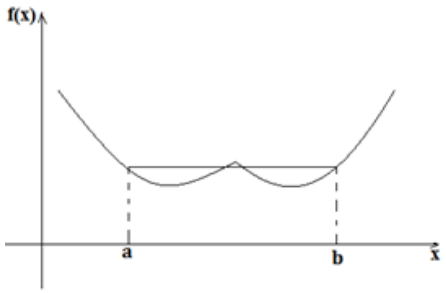


Figure 8.4: Example of a non-convex function

Q. How to determine whether a function is convex or concave?

The Hessian matrix of a function $f(x)$ can be analysed for convexity. Let us assume that the function $f(x)$ is continuous and twice differentiable on a feasible region $\Omega \in \mathbb{R}^n$. Let $H(x)$ be the Hessian matrix of a function $f(x)$, $x \in \mathbb{R}^n$ i.e. with n decision variables.

By definition :

$$H(x) \underset{\equiv}{def^n} \frac{\partial^2 f(x)}{\partial x_i^2}, \qquad i = 1, 2, \ldots\ldots, n$$

For a set of feasible points x, different cases can be listed:

a) If $H(x)$ is positive semi-definite $(i.e.\ iff\ x^T Hx \geq 0)$ for all $x \neq 0$,

or,

If all the eigen values of $H(x)$ are positive (zero included) i.e. if $eig\big(H(x)\big) \geq 0$, then $f(x)$ is convex on Ω. It means it has at most one minimum and this minimum is the global minimum.

b) If $H(x)$ is positive definite $(i.e.\ iff\ x^T Hx > 0)$ for all $x \neq 0$,

or,

If all the eigen values of $H(x)$ are positive i.e. if $eig\big(H(x)\big) > 0$, then $f(x)$ is strongly convex on Ω. It means it has a strict local minimum at x.

c) If $H(x)$ is negative semi-definite $(i.e.\ iff\ x^T Hx \leq 0)$ for all $x \neq 0$,

or,

If all eigen values of $H(x) \leq 0$, then $f(x)$ is concave. It means it has at most one maximum which is also the global maximum.

d) If $H(x)$ is negative definite $(i.e.\ iff\ x^T Hx < 0)$ for all $x \neq 0$,

or,

If all eigen values of $H(x) < 0$, then f(x) is strongly convex. It means it has a strict local maximum at x.

8.3.4 Active and Inactive constraints

In solving any optimization problem, all the equality constraint $h_i(x) = 0$ are active i.e. they are always 'ON' or active at a feasible point since they have to be exactly satisfied. However, inequality constraints may be either active or inactive (not active) at a feasible point.

For an example, consider an inequality constraint,

$y^2 + y \leq 20$

If, $y = 3$ then $y^2 + y = 12$

i.e. the constraint $y^2 + y \leq 20$ or $12 \leq 20$ is satisfied but is NOT active for this feasible point $y = 3$.

If $y = 4$, then $y^2 + y = 20$

i.e. the constraint $y^2 + y \leq 20$ or $20 \leq 20$ is satisfied and at the same time also active for this feasible point $y = 4$.

When an inequality constraint becomes active, then it changes into equality constraint.

8.3.5 Lagrangian function

The Lagrangian function of the nonlinear optimization problem of (A) can be written as,

$$L(x,\lambda,\mu) = f(x) + \sum_{i=1}^{m} \lambda_i [h_i(x) - b_i] + \sum_{j=1}^{r} \mu_j [g_j(x) - c_j]$$

Here, λ_i are the Lagrange multiplier for the m equality constraints.

Here, μ_j are the Lagrange multiplier for the r inequality constraints.

Then, the reduced form of the nonlinear optimization problem of (A) becomes,

$$\underset{(x,\lambda,\mu)}{Min}\ L(x,\lambda,\mu) = f(x) + \sum_{i=1}^{m} \lambda_i [h_i(x) - b_i] + \sum_{j=1}^{r} \mu_j [g_j(x) - c_j]$$

Conditions for optimality:

Assume that in the optimization problem of (A), $f(x), g_j(x)$ & $h_i(x)$ are continuous function and at least twice differentiable. We can use Karush-Kuhn-Tucker (KKT) conditions to define the conditions for optimality of optimization problem of (A).

First order necessary conditions:

Assume the x^* is a local solution of (A) and that $f(x), g_j(x)$ & $h_i(x)$ are differentiable and their derivatives are continuous. Further, assume that all the active constraint gradients are linearly independent at x^*. Then there exists Lagrange multipliers λ_i^* and μ_j^* for $i = 1,2,\ldots\ldots\ldots\ldots,m$ and $j = 1,2,\ldots\ldots\ldots,r$ such that the following conditions (KKT conditions) hold or are satisfied at $(x^*,\ \lambda^*,\ \mu^*)$:

$$\nabla L\ (x^*,\lambda^*,\mu^*) = \nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla h_i(x^*) + \sum_{j=1}^{r} \mu_j^* \nabla g_j(x^*) = 0$$

$$h_i(x^*) = b_i$$

and with complementary slackness holding true for inequality constraints such that,

$\mu_j^* \geq 0 \quad for\ g_j(x^*) = c_j \ \rightarrow for\ active\ inequality\ constraint.$

$\mu_j^* = 0 \quad for\ g_j(x^*) < c_j \ \rightarrow for\ inactive\ inequality\ constraint.$

The property that inactive inequality constraints will have zero multipliers is called complementary slackness. The reason why $\mu_j = 0$ for inactive inequality constraints is that,

when an inequality constraint is inactive, it is the same as considering that is was not present in the system i.e. it does not constraint the process operation when it is not active.

Second order sufficient conditions:

The first order necessary conditions does not say anything about whether it is minimum optimal or maximum optimal. So to guarantee an optimality (either maximum or minimum), the second order sufficient conditions are posed. Suppose that for some feasible point $x^* \epsilon\ \mathbb{R}^n$ there exists Lagrange multipliers such that KKT first order necessary conditions are satisfied. Also suppose that $f(x)$, $h_i(x)$ & $g_j(x)$ are twice differentiable and their derivatives are continuous. KKT second order sufficient conditions states that for a minima to exit,

$$\nabla^2 L(x^*, \lambda^*, \mu^*) > 0 \qquad (8.1)$$

i.e. the second order partial derivatives of the Lagrangian function of the original NLP problem should be positive definite. So, if (x^*, λ^*, μ^*) is a KKT point for problem (A) and 2nd order sufficiency conditions are satisfied at that point, optimality is guaranteed.

Note : If there are no active constraints (x^* is an unconstrained stationary point) and the Lagrange multipliers in equation (8.1) λ^* & μ^* are zero. Then condition given by (8.1) then reduces to the condition discussed in the section 8.3.3 (under topic "convexity": How to determine whether a function is convex or concave?)

8.3.6 Solution methods:

To solve the optimization problem of (A), most of the solvers follow iterative solution procedure. In this course, we do not go into details about the methods and steps involved in those methods. However, it will be interesting to understand the basic algorithmic steps used by many different methods and solvers.

1. Start with the chosen initial point x_0 and initial iteration number k,
2. Find or calculate the search direction, P_k
3. Calculate the step length $\propto_k$ to be taken in the search direction calculated in step 2. This step is also called "line search".
4. Compute the new solution using P_k & $\propto_k$ i.e. move one step forward in the search direction by stepping with a length of $\propto_k$,
$$x_{k+1} = x_k + \propto_K P_k$$
5. At this new point, check the termination criteria.
e.g.: Check if $f(x_{k+1}) \leq \varepsilon$ where ε is a tolerance number
6. If termination criteria is fulfilled stop the iteration otherwise go to step (2) & repeat the process.
7. Stop

Note: (1): There are many methods that can be used for calculating the search direction P_k. Examples are the steepest descent method (Newton's methods, Quasi-Newton method), conjugate gradient method etc.

Note (2): Gradients needed for finding the search direction P_k can also be calculated using finite difference methods. E.g. forward difference, central difference method etc. However, using these methods for approximating gradients may not always be very accurate.

Note: (3): Some methods are derivative free method where no gradient information is needed to calculate P_k. e.g. Pattern search method, mesh adaptive direct search etc.

8.4 Some widely used algorithms for solving NLPs

Below you will find the description of two widely used methods for solving nonlinear optimization problems. They are:

(a) SQP (Sequential Quadratic Programming).

(b) GRG (Generalized Reduced Gradient)

8.4.1. Sequential Quadratic Programming

Let us consider a general nonlinear constrained optimization problem as,

$$\begin{aligned} &\min && f(x) \\ &\text{subject to} \\ &&& h_i(x) = 0 \; i \in \varepsilon \\ &&& g_i(x) \leq 0 \; j \in I \end{aligned} \tag{8.2}$$

where $f(x)$ such that $x \in \mathbb{R}^n$is the objective function to be minimized, $h_i(x), i = 1,2, \dots \dots \dots, m$ are the m equality constraints and $g_j(x), j = 1,2, \dots \dots \dots, r$ are r inequality constraints. SQP is an iterative method which approximates a Quadratic Programming (QP) subproblem from the nonlinear constrained optimization problem for each given iterate x^k. The QP subproblem is solved and the solution is updated to a better solution x^{k+1}. The new iterate is used again to solve the approximated QP subproblem and the process is repeated to create a sequence of x^k which will converge to a local minimum x^* of the NLP of Equation (8.2) as $k \rightarrow \infty$.

Let us define the Lagrangian function of the NLP problem of Equation (8.2) as,

$$\mathcal{L}(x,\lambda,\mu) = f(x) + \sum_{i=1}^{m} \lambda_i \, h_i(x) + \sum_{j=1}^{r} \mu_j \, g_j(x) \tag{8.3}$$

where λ_i are the Lagrange multipliers for the equality constraints and μ_j are the Lagrange multipliers for the inequality constraints. For any local minimum x^*, Karush Kuhn-Tucker Conditions (KKT) or the first order necessary conditions are satisfied.

$$\nabla \mathcal{L}\,(x^*,\lambda^*,\mu^*) = \nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \, \nabla h_i(x^*) + \sum_{j=1}^{r} \mu_j^* \, g_j(x^*) = 0 \tag{8.4}$$

The complementary slackness hold true for the inequalities at x^*.

$$\begin{aligned} &\mu_j^* g_j(x^*) = 0, \quad j = 1,2, \dots \dots \dots, r \\ &\qquad \mu_j^* > 0, j \in I_{ac}(x^*) \end{aligned} \tag{8.5}$$

For any given $x \epsilon \mathbb{R}^n, \nabla f(x), \nabla h_i(x)\ and\ \nabla g_i(x)$ are the gradient of $f(x), h_i(x) and\ g_i(x)$ at x respectively. The QP subproblem is created by approximating the Lagrangian of Equation(8.3) using the first three terms of its Taylor's series. For a given current iterate (x^*,λ^*,μ^*), the quadratic approximation for the Lagrangian is written as,

$$\mathcal{L}\,(x^k,\lambda^k,\mu^k) = \nabla\mathcal{L}(x^k,\lambda^k,\mu^k)^T\Delta x + \frac{1}{2}\Delta x^T\nabla^2\nabla\mathcal{L}(x^k,\lambda^k,\mu^k)^T\Delta x \tag{8.6}$$

Similarly the nonlinear constraints can be linearized by using the first two terms of their Taylor's series at the given current iterate (x^k,λ^k,μ^k). The term $\nabla^2\mathcal{L}(x^k,\lambda^k,\mu^k)$ is the Hessian of the Lagrangian function which is given by,

$$\nabla^2\mathcal{L}\,(x^k,\lambda^k,\mu^k) = \nabla^2 f(x^k) + \sum_{i=1}^{m}\lambda_i^k\,\nabla^2 h_i(x^k) + \sum_{j=1}^{r}\mu_j^k\,\nabla^2 g_j(x^k) \tag{8.7}$$

To compute the Hessian $\nabla^2\mathcal{L}\,(x^k,\lambda^k,\mu^k)$ of the Lagrangian function, it is necessary to compute the second derivative of all the problem functions. Also, the Hessian matrix might not be positive definite making the QP subproblem difficult to solve. So, instead of calculating the Hessian, it is approximated by B^k and updated at each iteration. There are many methods available for approximating and updating B^k like Powel symmetric Broyden (PSB) update, Broyden Fletcher Goldfarb Shanno (BFGS) update, Powell SQP update, SALSA-SQP update etc. Later in this section, only a very brief description of the BFGS update is given. Then the QP subproblem to be minimized at the current iterate (x^k,λ^k,μ^k) using the approximated Hessian of the Lagrangian function can be written as,

$$\begin{aligned} &\underset{\Delta x}{minimize}\ \ \nabla\mathcal{L}(x^k,\lambda^k,\mu^k)^T\Delta x + \frac{1}{2}\Delta x^T\mathrm{B}^k\Delta x \\ \text{subject to}\quad & h_i(x^k) = \nabla h_i(x^k)^T\Delta x = 0, i \in \varepsilon \\ & g_j(x^k) = \nabla g_j(x^k)^T\Delta x \le 0, j \in I \end{aligned} \tag{8.8}$$

By solving the QP subproblem of (8.8), the optimal search direction Δx and the corresponding Lagrange multipliers λ^k and μ^k of the subproblem can be obtained. In the simple case, the optimal Lagrange multipliers for the QP subproblem can be used for the next iteration such that $\lambda^{k+1} = \lambda^k$ and $\mu^{k+1} = \mu^k$ The new iterate x^{k+1} can be obtained as,

$$x^{k+1} = x^k + \alpha^k\Delta x \tag{8.9}$$

where α^k is the search or step length for the k^{th} iteration. A suitable value of α^k can be calculated and the convergence properties of the SQP algorithm can be improved by using a line search which provides the choice of distance to be moved along the direction generated by the QP subproblem. Merit functions $M(x)$ are used to make sure that the reduction in $M(x)$ takes the iterates of the SQP algorithm eventually close to x^* such that

$$M(x^k + \alpha^k\Delta x) < M(x^k) \tag{8.10}$$

Two commonly used merit functions are the l_1 penalty merit function and the augmented Lagrangian merit function. For constrained problems, the interest lies in the next iterate to not only decrease the objective function but also to come closer to satisfying the constraints. The relative importance are weighted and penalized. The l_1 penalty merit function is written as,

$$M_1(x^k, \eta) = f(x^k) + \sum_{i=1}^{m} \eta_i |h_i(x^k)| + \sum_{j=1}^{r} \eta_j \max\left(g_j(x^k, 0)\right) \quad (8.11)$$

where $\eta_i > 0$ *and* $\eta_j > 0$ are the penalty parameters. If (x^k, λ^k, μ^k) satisfies the second order sufficiency condition, then x^* is a local minimum of M_1 if the penalty parameters are chosen such that $\eta_i > |\lambda_i|$ and $\eta_j > |\mu_j|$ For details about the l_1 penalty merit function and the augmented Lagrangian merit function, refer to other literature. The BFGS update for B^k is given as,

$$B^{k+1} = B^k + \frac{yy^T}{y^T s} - \frac{B^K s s^T B^k}{s^T B^k s}$$
$$s = x^{k+1} - x^k \quad (8.12)$$
$$\text{y} = \nabla \mathcal{L}(x^{k+1}, \lambda^k, \mu^k) - \nabla \mathcal{L}(x^{k+1}, \lambda^k, \mu^k)$$

If B^k is positive-definite and $y^T s > 0$, then B^{k+1} is also positive-definite. In case B^k is not sufficiently positive, Powell-SQP update which modifies y such that $y^T s > 0$ is always satisfied can be used.

To summarize, a simple and short algorithm of SQP is given below:

1. Set $k = 0$
2. Start with the user defined initial values $x^k = x_{ini}$, $B^k = B_{ini}$ (positive definite matrix) $\lambda^k = \lambda_{ini}$, $\mu^k = \mu_{ini}$
3. Form the approximated QP subproblem (Equation 8.8) $QP(x^k, \lambda^k, \mu^k, B^k)$ and solve it for the optimal Δx^*. Compute the optimal Lagrange multipliers λ^* *and* μ^*.
4. Is termination criteria satisfied? (e.g. Is $\Delta x^* < \in_x$? where ed is the specified tolerance. If yes goto step 9 else goto step 5.
5. Choose the step length α^k using the merit function such that $M(x^k + \alpha^k \Delta x) < M(x^k)$.
6. Update to find new iterate
$$x^{k+1} = x^k + \alpha^k \Delta x^*, \lambda^{k+1} = \lambda^*, \mu^{k+1} = \mu^* \quad (8.13)$$
7. Update the matrix using BFGS algorithm (Equation 11)
$$B^{k+1} = \emptyset(B^k, \Delta \mathcal{L}(x^k, x^{k+1}, \lambda^k, \mu^k), x^{k+1}, x^k) \quad (8.14)$$
8. Set $k \leftarrow k + 1$ and go to step 3
9. Optimal solution of the problem is $x^{opt} = x^k$ .Terminate the program.

8.4.2. Generalized Reduced Gradient Method

Generalized Reduced Gradient (GRG) is a method for solving general nonlinear optimization problem. It was first developed by Jean Abadie in 1969. As the name implies, the main idea of GRG method is to transform a general nonlinear problem with constraints and bounds to a reduced problem with only the upper and lower bounds. For this, the n decision variables are divided into two subsets of basic and nonbasic variables. Then the equality constraints are solved to express the basic variables in terms of nonbasic variables.

The general nonlinear Equation to be solved by the GRG method is written as,

$$Minimize \quad f(X) \quad (8.15)$$
$$Subject\ to \quad g_i(X) = 0\ , i = 1,2, \dots \dots, m \quad (8.16)$$
$$bounded\ by \quad l_i \leq X_i \leq u_i\ 0\ , i = 1,2, \dots \dots, n \quad (8.17)$$

Here, X is a vector of n decision variables and l_i and u_i are the lower and upper bounds to the decision variables. The independent decision variables are divided into m basic variable denoted by y and $(n - m)$ nonbasic variables denoted by vector x. The basic variables can be expressed in terms of the nonbasic variables by using the equality constraint. Remember that $g_i(x)$ is a general nonlinear function. So the basic variables will be expressed as nonlinear functions of non-basic variables. Let us denote the non basic variable as $y(x)$. The basic variables are then substituted in the objective function $f(X)$ and can be written as,

$$F(x) = f(y(x), x) \tag{8.18}$$

$$G(x) = g(y(x), x) = 0 \tag{8.19}$$

Now,

$$\frac{dG}{dx} = 0 = \frac{\partial g}{\partial y}\frac{\partial y}{\partial x} + \frac{\partial g}{\partial x} \tag{8.20}$$

$$\frac{dF}{dx} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial x} \tag{8.21}$$

From Equations 8.19 and 8.20 we get,

$$\frac{dF}{dx} = \frac{\partial f}{\partial x} - \underbrace{\frac{\partial f}{\partial y}\left(\frac{\partial g}{\partial y}\right)^{-1}}_{\text{Lagrange multiplier}}\frac{\partial g}{\partial x} \tag{8.22}$$

The term $\frac{\partial f}{\partial y}\left(\frac{\partial g}{\partial y}\right)^{-1}$ in Equation (8.22) is the Karush-Kuhn-Tucker multiplier vector. The nonlinear optimization problem is changed to a *reduced problem* that is unconstrained in nature with only the upper bound (u_i) and the lower bound (l_i) as,

Minimize

$$F(x) \tag{8.23}$$

subject

$$l_i \leq x \leq u_i \tag{8.24}$$

Note: The above optimization problem has only nonlinear equality constraints. If there are nonlinear inequality constraints, they can be expressed as nonlinear equality constraints using slack variables S. For e.g., consider the general nonlinear inequality constraint as,

$$h_j(X) \leq 0, j = 1,2, \ldots \ldots, r \tag{8.25}$$

The above inequality constraints can be written as equality constraints by adding slack variable S_j to each inequality constraint as,

$$h_j(X) + S_j = 0, j = 1,2, \ldots \ldots, r \tag{8.26}$$

$$S_j \geq 0, \quad j = 1,2, \ldots \ldots, r \tag{8.27}$$

Remember that you should also include the slack variables S_j as unknown decision variables to be optimized. So the number of decision variables will be increased from $n \text{ to } n + r$.

The procedure for solving the reduced problem of Equations 8.23 and 8.24 is explained below in steps. The general optimization problem could contain multiple local solutions (because it may not always be a convex optimization problem). In a non-convex optimization problem, to make sure that we obtain the best local solution, multiple starting points *(N* different initial points, but all feasible points) are used. The optimization problem is solved

for each of the initial starting point. We will then obtain N number of local solutions. The best one (the solution which minimizes the objective function the most) out of N solutions is then selected as the optimal solution. This procedure is called multi-start optimization.

1. Call the multi-start search routine to create 50 different feasible starting points and set the first starting point index $N = 1$.
2. Set the initial iteration number $k = 0$ and pick a starting point x_k^N which are the nonbasic variables and proceed to step 3.
3. Use the nonbasic variables vector x_k^N to solve the nonlinear equality constraints of Equation 8.19 and determine the values of the basic variable vector y_k^N.
   If the value of y_k^N violates the bounds, stop the iterative procedure and perform the *change of basis.* The basic variable which exceeded the bounds is made nonbasic and one of the nonbasic variable which is not on bound is made basic. The *change of basis* will produce a new set of nonbasic variables x_k^N, basic variable vector y_k^N and a new reduced function $F(x_k^N)$.
   If the *change of basis* was active, goto step 3. If the *change of basis* was not active, go to step 4.
4. For the set of nonbasic variables, calculate the direction of movement d_k^N.
   The equation for calculating the search direction is given by Goldfarb's algorithm and simplified for the special case of bounded variables as,
   $$d_k^N = -H_k^N \nabla F(x_k^N) \qquad (8.28)$$
   Here $H_k^N \in \mathbb{R}^{(n-m)\times(n-m)}$ is a positive semi-definite matrix whose diagonal element is set to zero if the corresponding nonbasic variable is at a bound. $\nabla F(x_k^N)$ is the derivatives of the reduced objective function given by Equation 8.22.
5. Perform line search to determine the step size α to be taken in the direction d_k^N by minimizing the one dimensional search problem $F(x_k^N + \alpha d_k^N)$. The next point is given by,
   $$x_{k+1}^N = x_k^N + \alpha d_k^N \qquad (8.29)$$
   α should be chosen such that $x_k^N + \alpha d_k^N$ satisfies the bounds on the nonbasic variables x.
   a. First calculate with a full Newton step ($\alpha = 1$) to get x_{k+1}^N. If the rate of descent from x_k^N to x_{k+1}^N is at least some prescribed fraction of the initial rate of descent in the search direction i.e. if $F(x_k^N + d_k^N) \leq F(x_k^N) + \gamma\alpha\nabla^T F(x_k^N) d_k^N$ with $\gamma = 10^{-4}$ and if x_{k+1}^N satisfies all the bounds then accept $\alpha = 1$, then goto step 6. If not goto step 5(b).
   b. Perform quadratic interpolation to get the value of α where the objective function $F(x_k^N + \alpha d_k^N)$ is minimized. With $\alpha = 0$, we know the values of $F(x_k^N)$ and $\nabla^T F(x_k^N) d_k^N$. With $\alpha = 1$ we know the value of $F(x_k^N + d_k^N)$. Using these three pieces of information we can perform the quadratic interpolation to get the value of α as,

$$\hat{\alpha} = \frac{\nabla^T F(x_k^N) d_k^N}{2[F(x_k^N + d_k^N) - F(x_k^N) - \nabla^T F(x_k^N)\, d_k^N)]} \quad (8.30)$$

If x_{k+1}^N found out using $\hat{\alpha}$ satisfy all the bounds, accept $\hat{\alpha}$. Then goto step 6. If not goto step 5(c).

c. As the last defense, use the scanning method to find α which minimizes $F(x_k^N + \alpha d_k^N)$ as well as satisfies all bounds in x_{k+1}^N. In the scanning method, iteration is started from a very small value of $\alpha = 10^{-4}$ and at the end of each iteration the value of α is increased by a small incremental value $d\alpha = 10^{-3}$. At each iteration, $F(x_k^N + \alpha d_k^N)$ is calculated and checked whether the bounds are satisfied. The iteration is stopped under two conditions:

- when one of the components of x_{k+1}^N violates the bound, say the i^{th} component $(x_k^N)^i$ then keep it at the bound (x_{bound}^i) and calculate the value of α as,

$$\alpha = \frac{x_{bound}^i - (x_k^N)^i}{(d_k^N)^i} \quad (8.31)$$

- when $F(x_k^N + \alpha d_k^N) > F(x_k^N)$, stop the iteration and calculate the value of α as,

$$\alpha = \alpha_{previous\ iteration} - d\alpha \quad (8.32)$$

6. Check for optimality by simply looking at absolute value of the difference of original objective functional values with $F(y_k^N + x_k^N)$ and $F(y_{k+1}^N + x_{k+1}^N)$. If
$$|f(y_{k+1}^N + x_{k+1}^N) - f(y_k^N + x_k^N)| < \epsilon_1, \quad \epsilon_1 = 10^{-6} \quad (8.33)$$
goto step 7 else put k = k + 1 and goto step 3.
7. Store the value of $(y(x_{k+1}^N),\ x_{k+1}^N)$ set $N = N + 1$ and goto step 2. If $N > 50$ goto step 8.
8. Compare the 50 local optimal values and choose the one for which the original objective function is maximum. This best solution will be closet to the global solution.
9. Terminate the program.

8.5 Solvers in MATLAB

In this course, you do not need to develop your own optimization routine. We will be using optimization solvers that are already developed and made available for use. There are many solvers for solving nonlinear optimization problem. Some of them are commercial (like the ones found in MATLAB optimization toolbox). Luckily, the open source community has also been active in this area. Therefore, there are also many efficient open-source solvers that are freely available to the users. A survey of optimization software has been listed at *ascend4.org/Survey_of_optimisation_software*. COIN-OR also has many open-source projects for the development of optimization solvers. You can see it at http://www.coin-or.org/projects/. One good solver (open-source) is the *IPOPT* solver. It can be interfaced with MATLAB using the OPTI toolbox which is a free MATLAB toolbox for optimization. It was developed and been

maintained by Industrial Information and Control Centre under Auckland University of Technology and The University of Auckland. OPTI toolbox also incorporate many other freely available solvers (in addition to *IPOPT)* for solving different types of optimization problem. You can read more about OPTI toolbox at:

http://www.i2c2.aut.ac.nz/Wiki/OPTI/index.php/Main/HomePage

MATALB optimization toolbox also has a good solver called *fmincon* for solving constrained nonlinear optimization problem. You can use *fmincon* solver with algorithms such as SQP, active set, interior point etc. for designing a Model Predictive Controller (MPC). In this course, we will use MATLAB and *fmincon* solver for solving the nonlinear constrained optimization problems. To understand the structure and input arguments of *fmincon* use MATLAB help or see the documentation from Mathworks. The general syntax is,

options = optimset ('Algorithm', 'sqp',);
obj_func = @ (z) your_obj_func (z); % function for calculating objective f(z)
cons_func = @ (z) your_constraint_func(z); % function for calculating constraints
$[z\_opt, fval] = fmincon(obj\_func, z_{ini}, A_i, b_i, A_{eq}, b_{eq}, z_L, z_U, cons\_func, options);$

to solve the problem of the form,

Minimize

$$z \qquad f(z) \qquad (8.34)$$

subject to,

$$A_i z \leq b_i \ (linear, inequality\ constraints) \qquad (8.35)$$
$$A_{eq} z = b_{eq} \ (linear\ equality\ constraints) \qquad (8.36)$$
$$z_L \leq z \leq z_U \ (bounds) \qquad (8.37)$$
$$g(z) \leq 0 (nonlinear, inequality\ constraints) \qquad (8.38)$$
$$h(z) = 0 \ (nonlinear, equality\ constraints) \qquad (8.39)$$

z_{ini} is the initial staring point for the solver to look for optimal solution of z. If any of the input arguments to $fmincon$ is not present, replace it by [ ]. For e.g. if linear inequality and linear equality constraints are not present in the optimization problem, use $A_i = [\,], b_i = [\,], A_{eq} = [] \ and \ B_{eq} = [\,]$.

obj_func = @ (z) your_obj_func (z) is the handle for defining your objective function. The handle *obj_func* is used as input argument to $fmincon$ to tell the optimizer that the objective function is defined in the function named *your_obj_func*.

Similarly, *cons_func = @ (z) your_constraint_func(z)* is the handle for defining the nonlinear constraints. The handle *cons_func* is used as input argument to $fmincon$ to tell the optimizer that the nonlinear constraints function is defined in the function named *your_obj_func*.

options = optimset ('Algorithm', 'sqp',) is the solver options. Here it tells the solver to choose the SQP algorithm for solving the nonlinear optimization problem.

The solver will find the optimal values of the decision variables and return it back to you as $[z\_opt, fval]$. Here, u_opt is the optimal value and $fval$ is the functional value of the objective function.

8.5.1 A general example:

Let us consider a continuous through- circulation dryer as shown in the Figure 8.5.

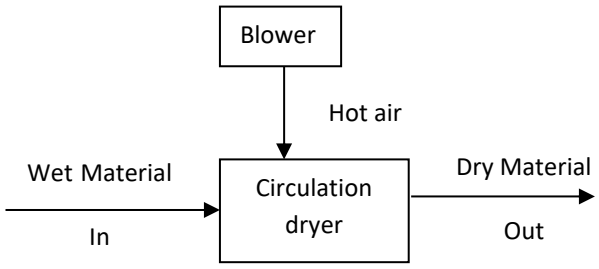


Figure 8.5: Block diagram of circulation dryer

The dryer takes in wet material as the input feed, dries it up and then the dry material come out of the dryer. This happens continuously i.e. the wet materials are continuously fed into the dryer and dried materials are obtained from the dryer continuously i.e. it is not a batch process.
Let x_1 be the velocity of the material/fluid fed into the dryer and x_2 be the depth of the dryer bed. For this example, the variables to be optimized are x_1 and x_2, i.e. $z = [x_1; x_2]$ The production rate R of the dryer is given by a nonlinear functions of x_1 and x_2 as,

$$R = 0.0064x_1\left[1 - e^{-0.184x_1^{0.3}x_2}\right]$$

For obtaining a maximum production rate, the motors, pumps and heater for the circulating blowers will be operating at the maximum allowed level and the moisture content of the dried material that comes out of the dryer will be tight. The power (P) consumed by the process is also a nonlinear function of x_1 and x_2 and is expressed as,

$$P = (3000 + x_1)\, x_1^2 x_2$$

The moisture content (M_c) of the dried material is measured using a complex exponential function which is also dependent on x_1 and x_2 as,

$$M_c = e^{0.184x_1^{0.3}x_2}$$

Let us formulate an optimization problem where we would like to maximize the production rate R when the power constraint P (due to pumps, motors and heaters) and the moisture content constraint M_c are specified. In other words, we would like to answer the question: For a given power consumption of 1.2×10^3 watt and a regulated moisture content of 4.1, what can be the maximum production rate?

The nonlinear optimization problem can be formulated as minimization problem (taking $-R$) as,

$$\min_{(x_1, x_2)} \; J = f(x) = -R = 0.0064x_1\left[1 - e^{-0.184x_1^{0.3}x_2}\right] \rightarrow \text{nonlinear objective function} \quad (8.40)$$

subject to,

$$(3000 + x_1)\, x_1^2 x_2 = 1.2 \times 10^{13} \rightarrow \text{Power constraint} \quad (8.41)$$

$$e^{0.184x_1^{0.3}x_2} = 4.1 \rightarrow \text{moisture content constraints} \quad (5.42)$$

The NLP problem given by Equations 8.40 - 8.42 can be solved by using *'fmincon'* solver in MATLAB. The *'fmincon'* solver accepts the nonlinear constraints in the form,

$$g(x) = 0 \rightarrow \text{nonlinear equality constraints}$$

$$h(x) \leq 0 \rightarrow \text{nonlinear inequality constraints}$$

So, the constraints of Equations 8.41 - 8.42 has to be written in the following form,

$$(3000 + x_1)\, x_1^2 x_2 - 1.2 \times 10^{13} = 0 \rightarrow \text{Power constraint} \quad (8.43)$$

$$e^{0.184 x_1^{0.3} x_2} - 4.1 = 0 \rightarrow \text{moisture content constraints} \quad (8.44)$$

The MATLAB code for solving the optimization problem of the dryer can be found in the home page of the course. The script *dryer_main.m* is the main file which should be run. The *interior-point* algorithm is chosen using *optimset.* The objective function is defined in the script *obj_func_dryer.m.* The nonlinear constraints is defined in the script *con_func_dryer.m.* Handle to the objective function *obj_func* and the handle to the constraints *con_func* are passed into the *fmincon* routine to solve the problem. The initial values of the unknowns $x_0 = [28000,1]^T$.

```
>>[x_opt,fx_opt]=fmincon(obj_func,x0,[],[],[],[],[],[],con_func,options)
```

```
x_opt =
  3.1766e+04
  3.4207e-01
fx_opt =
  1.5371e+02
```

Practice: Find the optimal solution to Rosenberg banana function subject to certain constraints using *fmincon* solver in MATLAB. ,

$$\min_{(x_1, x_2)} \; J = f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \rightarrow \text{nonlinear objective function} \quad (8.45)$$

subject to,

$$x_1 > 0 \quad (8.46)$$

$$x_2 > 0 \quad (8.47)$$

8.6 Nonlinear Optimal control

To make a nonlinear MPC, let us first create a nonlinear optimal control problem. Then, we can use the receeding horizon strategy to the nonlinear optimal control problem to create a nonlinear MPC.

For an optimal control problem, we need to look ahead into the future, say with a prediction horizon of N time steps and predict the future behavior of the process using the mathematical model of the process. The predicted behavior of the process is optimized to fulfill the control objective (here tracking the reference line *"Refline"* as shown in Figure 8.6). The concept is exactly the same as we discussed for LQ optimal control in Lecture 3. For simplicity the diagram showing optimal control concept is shown here as well.

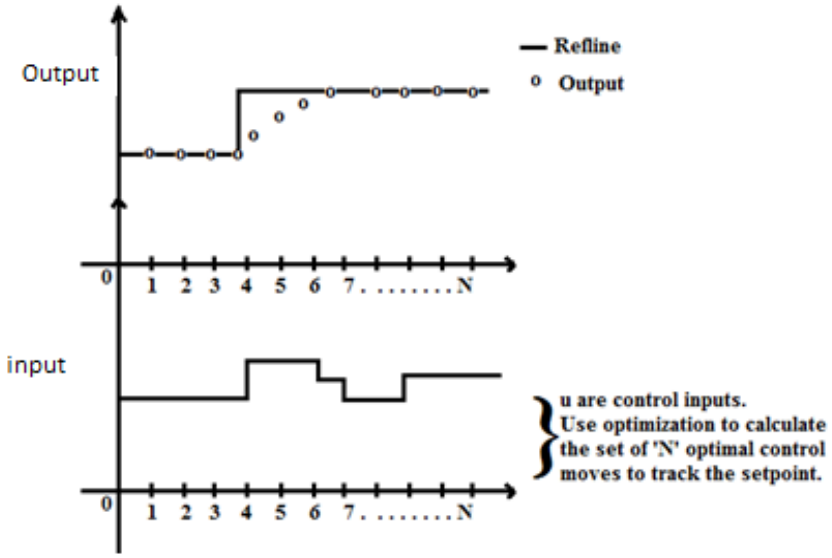


Figure 8.6: Conceptual diagram for nonlinear optimal control

The process model is used to predict the future states and the outputs, and this process model can be a nonlinear model. We then need to define the objective function to be minimized (or maximized) over the prediction horizon . The objective function can be any nonlinear function (scalar function though) denoted by $f(z)$. The quadratic objective covers most of the practical cases with respect to process control (e.g. set point tracking, profit maximization etc.). For the set point tracking optimal control problem, we can express the objective as,

$$\text{Minimize } f(z) = \frac{1}{2}\sum_{k=1}^{N_p} e_k^T\, Q e_k + u_{k-1}^T\, P u_{k-1} + \Delta u_{k-1}^T\, R \Delta u_{k-1} \tag{8.48}$$

subject to,

$$\mathrm{h}_i(z) - b_i = 0,\ i = 1,2,\dots\dots,m \tag{8.49}$$

$$\mathrm{g}_j(z) - c_j \le 0,\ j = 1,2,\dots\dots,r \tag{8.50}$$

Here, $f(z)$ is a quadratic objective function. It has $z \in \mathbb{R}^n$, i.e. n number of unknown variables to be optimized. Usually the variable to be optimized are the control inputs u_k (or sometimes rate of change of control inputs, $\Delta u_k = u_k - u_{k-1}$).

For calculating the objective function, we must at first calculate some variables over the prediction horizon of N_p from $k = 1\ to\ k = N_p$ (thus the $\sum$ sign in Equation 8.48). For example: In the objective function of Equation 8.48, there is the error term $e_k = r_k - y_k$. The reference signal r_k is known or defined for the whole prediction horizon length. But the process output is not known for the whole prediction horizon length. Thus y_k has to be calculated first before we can calculate the objective function. To calculate the process output y_k, we should solve the nonlinear model of the process. If the nonlinear model of the system is in continuous time form (with differential equations for the states and in addition some algebraic equations), then you should solve the continuous time dynamic model to calculate e_k for each time step from $k = 1$ to $k = N_p$.

You can use the built-in ODE solvers in MATLAB (*ode* 45, *ode15s etc.*) for each time step to calculate e_k or any other variables of interest. However, for nonlinear optimal control and for MPC, use of the built-in solver in MATLAB is not recommended[16]. It is advised that you implement the Runge-Kutta (or Euler) algorithm for solving the ODEs. And remember that for solving process model described by the ODEs, you must know the initial values of the states x_0.

$\mathrm{h}_i(z) - b_i = 0$ are the m number of nonlinear (could be linear also) equality constraints. $\mathrm{g}_j(z) - c_j \leq 0$ are the r number of nonlinear (could also be linear) inequality constraints. If any bounds on z are present like $z_L \leq z \leq z_H$, then it can be directly passed as the input arguments to the *fmincon* solver. It may also be necessary to solve the process model (the nonlinear ODEs) to calculate the equality and inequality constraints.

From a process control point of view, the control inputs u are the manipulated variables. Unlike the LQ optimal control problems where z could contain $z = [u, x, e, y]^T$, for the nonlinear optimal control problem, it is necessary for us to include only u as the element of z. This is because, no well structured matrices/vectors is present that aids in the optimization the other elements of the vector. Therefore for nonlinear optimal control, the **decision variable z can simply contain only u.**

To solve the optimal control problem of Equation (8.48 – 8.50) which actually is an optimization problem formed by predicting the process behavior *N* steps ahead of the current time, $fmincon$ of the MATLAB optimization toolbox or *IPOPT* of OPTI toolbox can be used. The optimization solver will calculate the unknown variables (or decision variables) not only for the current time step but for the whole prediction horizon length, for e.g. it will calculate *N* number of control inputs u, *N* number of control deviation Δu etc. We then apply these *N* number of optimal control moves to achieve the needed control behavior.

8.7 Example of nonlinear optimal control problem

Nonlinear optimal control of the pressure at the bottom of a tank

To get started, we will first look at a very basic and simple example. In this example, we will formulate a nonlinear optimal control problem to control the pressure at the bottom of a tank.

Process description:

The tank has an inlet pipe used to pour liquid into it (Q_{in}) using a pump as shown in Figure 8.7. It also has an outlet pipe with a choke valve. The choke valve can be opened/closed to regulate the amount of liquid flowing out (Q_{out}) of the tank. The pressure at the bottom of the tank (P_c) is due to the liquid column present inside the tank. This pressure is dependent on how much amount of

[16] A huge amount of time is lost i.e. there is a huge overhead in calling and executing built-in ode solver from inside a for loop.

liquid is present inside the tank. If the inflow/outflow varies, the amount of liquid present in the tank varies and hence the pressure at the bottom of the tank also varies.

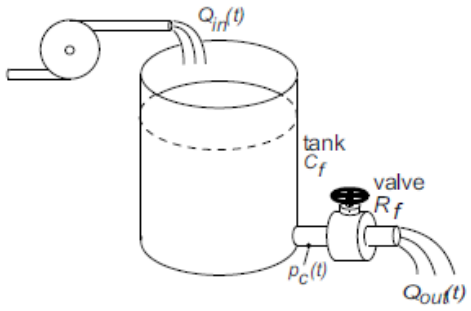


Figure 8.7: Liquid tank with inlet and outlet

For this example, let us fix the inflow and change only the outflow to change the pressure at the bottom of the tank. The outflow can be changed by opening/closing the choke valve. Thus for our case, the choke valve opening is the control input (u). The inflow (Q_{in}) which is kept constant is the input disturbance.

The dynamic model of the pressure at the bottom of the pipe is given the ordinary differential equation of 8.51.

$$\frac{dP_c}{dt} = \frac{1}{C_f} Q_{in}(t) - \frac{K_v u}{R_f C_f} P_c \qquad \text{with initial state } P_{c0} \text{ known} \tag{8.51}$$

Here, C_f, R_f are the parameters of the process and K_v is the gain of the outlet valve. The parameters/variables of the process are listed in Table 8.1.

Table 8.1: Parameter and variables

Parameters	Value
C_f	2.04×10^{-4}
R_f	1×10^{-6}
K_v	1.5
Q_{in}	1
u	[0 1]

The objective is to design a nonlinear optimal controller that will control the pressure at the bottom of the tank to their given set points.

The nonlinear optimal control problem can be written as,

$$\min_{u} \; J = \frac{1}{2} \sum_{k=1}^{N_p} \left(r_k - P_{c,k}\right)^T Q \left(r_k - P_{c,k}\right) + u_{k-1}^T P u_{k-1} \tag{8.52}$$

subject to,

$$0 \leq u_k \leq 1 \qquad 8.53(a)$$

where $P_{c,k}$ is the predicted pressure at the bottom of the tank for each time step k throughout the prediction horizon, which is obtained by solving the process model,

$$\frac{dP_c}{dt} = \frac{1}{C_f} Q_{in}(t) - \frac{K_v u}{R_f C_f} P_c \qquad \text{with initial state } P_{c0} \text{ known} \qquad 8.53(b)$$

Here r_k is the reference value of the pressure (which has to be pre-defined for the whole prediction horizon), u_k is the valve opening of the outlet valve, N_p is the prediction horizon, Q is the weighting matrix for the error (difference between setpoint and the bottom pressure) and P is the weighting matrix for the control inputs. The vector of unknowns to be optimized for this example are the N_p number of the control inputs u_k.

Important note:
To determine the objective function of Equation 8.52, the values of $P_{c,k}$ are needed for each time step from $k = 1$ to $k = N_p$. To calculate P_c for each time step k throughout the prediction horizon, the nonlinear ordinary differential equation (8.51) (which is the model of the process) should be solved. To solve this ODE, it is advised that that Runge-Kutta algorithm be used instead of using the built-in MATLAB solver.

In addition, let us also consider that the valve cannot be opened/closed by more than 0.1 at each time step. In other words, we have a constraint for the rate of change of valve opening as,

$$-0.1 \leq \Delta u_k \leq 0.1 \qquad (8.54)$$

The rate of change of valve opening is defined as, $\Delta u_k = u_k - u_{k-1}$. Constraints given by (8.53) and (8.54) can be written as inequality constrains as,

$$u_k - 1 \leq 0 \qquad (8.55)$$
$$-u_k + 0 \leq 0 \qquad (8.56)$$
$$\Delta u_k - 0.1 \leq 0 \qquad (8.57)$$
$$-\Delta u_k + (-0.1) \leq 0$$

Let us choose the prediction horizon $N_p = 136$ with a time step $dt = 6$ seconds. Let the initial state of the pressure be $10 \times 10^5\ N/m^2$. The dynamic optimization problem (i.e. the optimal control problem given by Equations 8.52, 8.55 – 8.57) is solved in MATLAB using the *fmincon* solver. The solver calculates the optimal values of 136 number of control moves/actions to be taken. With NLP optimal control problem, all of these 136 number of control actions are applied to the process to obtain the required controlled dynamics. Figure 8.8 shows the simulation results. The source code of the optimal control problem implemented in MATLAB with the *fmincon* solver is available in the homepage of this course for download. The main file to be run in MATLAB is "`main_file_NL_tank.m`". In general, the objective function and the constraints are calculated separately. But it is also possible to calculate the objective function and the constraints together at the same time. This reduces the number of calculations by half. The objective function and the

constraints are calculated in a single file "`compute_both.m`" by solving the model equations only once. It is advised that the students see the code and read the comments/notes in the codes (many concepts are explained using the comments in the MATLAB codes). The source codes (a complete set) for this example is also written below. For applying nonlinear optimal control to other processes, you can simply modify these source codes.

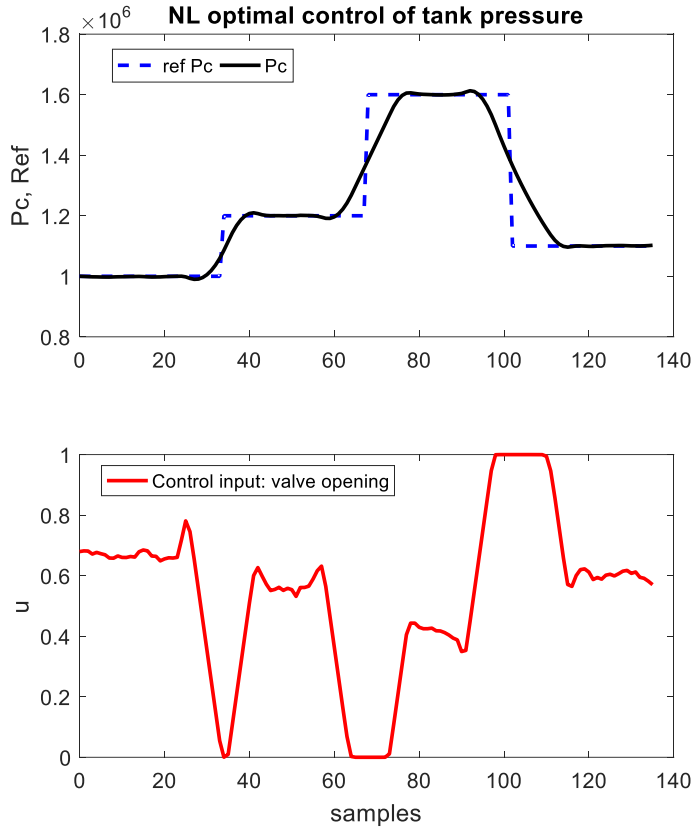


Figure 8.8: Nonlinear Optimal Control of the pressure at the bottom of a tank

The complete MATLAB source code is given below. The explanation of the source code follows after the codes.

<%the main file to be run> % main_file_NL_tank.m

```
clc
clear

%choose prediction horizon Np
Np = 136; %Np time steps ahead in the future
%sampling time
dt = 0.1*60; %sampling time in seconds %dt = 6 seconds

%initial values of the states
state_ini_values = 10e5; %intial pressure at the bottom of the tank

%initial value for optimizer
u_ini = ones(Np,1).*1; %the unknowns are initialized as full valve opening

%Reference
%make the reference vector offline (for the whole prediction horizon length).
Ref = [ones(Np/4,1)*10e5;ones(Np/4,1)*12e5;ones(Np/4,1)*16e5;ones(Np/4,1)*11e5];
```

```
%make the nonlinear optimization problem and solve it
u_k_ast = optimization_tank(u_ini,state_ini_values,dt,Ref,Np);


%we now have optimal values of 'u' to fulfill the objective function
%we can use the optimal values of 'u' to calculate the variables of interest
%to us.
%the ouput variables and the states are of interest. They should be
%calculated using the optimal values of 'u'. We use the model to calculate
%the output and the states.

%for storing
%storage place for the output
Pc = zeros(Np,1);

%Important note: We don't need this loop for MPC, only for NL opt. control
for i = 1:Np

    %calculate the outputs: for this example, outputs are the states
    %store the outputs
    Pc(i,1) = state_ini_values;

    %update the state with the optimal control inputs
    %with RK method
    x_next = my_runge_kutta(state_ini_values,dt,u_k_ast(i));
    state_ini_values = x_next;

end

%make time steps for plotting
tspan = linspace(0,Np-1,Np);
figure,
subplot(211)
plot(tspan,Ref,'b-',tspan,Pc,'k-')
legend('ref Pc','Pc','Orientation','horizontal')
ylabel('Pc, Ref'); title('NL optimal control of tank pressure');
subplot(212)
plot(tspan,u_k_ast,'r-')
xlabel('time [sec]'); ylabel('u');
legend('Control input: valve opening');
```

%### main file: main_file_NL_tank.m ends%################################

---------- x----------x----------

<function: optimization_tank.m: starts>%

```
function u = optimization_tank(u_ini,state_ini_values,dt,Ref,Np)
%In this function we:
%define functions for objective and constraints
%choose the slover types + other options to the solver

%######## Structure 2 ############################## %efficient
%reduces computation time by half
%We can cut the computational time by NOT repeating the same calculations
%for calculating the objective function and the nonlinear constraints
%let us try by making use of nested function

%options for chosing the type of method for optimization and other options
ops = optimset('Algorithm','sqp','Display','off','MaxIter',5000);
%ops =optimset('Algorithm','interior-point','Display','off','MaxIter',5000);
%ops = optimset('Algorithm','active-set','Display','off','MaxIter',20);


uLast = [];% Last place compute_both was called
myJ = [];% Use for objective at xLast
myG = [];% Use for nonlinear inequality constraint
myHeq = [];% Use for nonlinear equality constraint

%define the function where the objective function will be calculated
```

```
obj_func = @(u)objfun_tank(u,state_ini_values,dt,Ref,Np);
%define the function where the nonlinear constraints will be
%calculated (both equality and inequality constraints will be calculated in
%the same function
cons_func = @(u)confun_tank(u,state_ini_values,dt,Ref,Np);

%use the fmincon solver
[u,fval,exitflag,output,solutions] = fmincon(obj_func,u_ini,[],[],[],[],[],[],cons_func,ops);


    function J = objfun_tank(u,state_ini_values,dt,Ref,Np)
        if ~isequal(u,uLast) %check if computation is necessary
%             disp('button pressed: objective call');
%             pause;
            [myJ myG myHeq] = compute_both(u,state_ini_values,dt,Ref,Np);
            uLast = u;
        end
        %now compute objective function
        J = myJ;
    end

    function [G Heq] = confun_tank(u,state_ini_values,dt,Ref,Np)
        if ~isequal(u,uLast) %check if computation is necessary
%              disp('button pressed: constraint call');
%              pause;
            [myJ myG myHeq] = compute_both(u,state_ini_values,dt,Ref,Np);
            uLast = u;
        end
        %now compute constraints
        G = myG;
        Heq = myHeq;
    end
end

%########## Structure 2 ENDS #################################
```

<function: optimization_tank.m: ends>%

---------- x----------x----------

<function: compute_both.m: starts>%

```
function [myJ myG myHeq] = compute_both(u_ini,state_ini_values,dt,Ref,Np)
%this is the function where we compute the objective function and the
%constraints together at the same time.

%weighting matrices for error and control inputs
Qe = eye(Np).*1; %weighting matrix for the error
Pu = eye(Np).*1; %weighting matrix for the control inputs

%to store the output variable
Pc = zeros(Np,1);

%since we need to calculate the outputs for the whole prediction horizon we use a for loop and
%solve the ODE (model of the nonlinear process) using runge kutta. To solve the ODEs we need to
%know the initial values of the states. Thus they are passed into the "compute_both" function.
for i = 1:Np

    %find out which control input to use for each time step within the prediction horizon.
    u_k = u_ini(i,:);

    %use runge kutta to update the states
    x_next = my_runge_kutta(state_ini_values,dt,u_k);
    %use the states to calculate the output
    %in this case, the output is simply the state.
    %//Note: Sometimes outputs have to be calculated using algebriac
    %equations and the states. In such a case, please make a separate
    %function for calculating the output variables.
    Pc(i,1) = x_next;

    %update the state
    state_ini_values = x_next;
```

```
end

%now make the objective function
J = (Ref-Pc)'*Qe*(Ref-Pc) + u_ini'*Pu*u_ini;

myJ = J/2;

%if there are equaltiy constraints, it should be listed as a column vector
%here we don't have equality constraints so we use empty matrix
myHeq = [];

%For this example, there is also constraint on the rate of change of control input variables (du)
%such that -0.1<=du<=0.1
%since in the objective function, we don't have 'du' but only 'u', we have to calculate 'du'
ourself.
%find du from the input signals
du = u_ini(2:end)-u_ini(1:end-1);
%//Important Note: If the objective function was formulated such that it
%had " (du)'P du  ",then we don't have to calculate 'du' here because 'du' would be passed to
%this function "compute_both" by the optimizer and so we already would have it.

%list the inequality constraints as column vector
myG =[u_ini-1; % valve opening should be less than 1
      -u_ini+0; %valve opening should be greater than 0
      du - (0.1); % du should be less than 0.1 for each sample
      -du-(0.1);  % du should be greater than -0.1 for each sample
      ];
```

<function: compute_both.m: ends>%

---------- x----------x----------

<function: my_runge_kutta.m: starts>%

```
function x_next = my_runge_kutta(states_ini_values,dt,u_k_ast)
%the 4th oder runge kutta algorithm
K1 = tankPressure_equations([],states_ini_values,u_k_ast);
K2 = tankPressure_equations([],states_ini_values+K1.*(dt/2),u_k_ast);
K3 = tankPressure_equations([],states_ini_values+K2.*(dt/2),u_k_ast);
K4 = tankPressure_equations([],states_ini_values+K3.*dt,u_k_ast);

x_next = states_ini_values + (dt/6).*(K1+2.*K2+2.*K3 + K4);
```

<function: my_runge_kutta.m: ends>%

---------- x----------x----------

<function: tankPressure_equations.m: starts>%

```
function dP_dt = tankPressure_equations(t,x0,u)
%this is the function where we write the differential equations of the
%given model.

%needed parameters
Cf = 2.04e-4;
Rf = 1e6;

Qin = 1; %constant inflow
Kv = 1.5; %gain of the valve
dP_dt = Qin/Cf - (x0*u*Kv)/(Rf*Cf);
```

< tankPressure_equations.m: ends>%

---------- x----------x----------

8.7.1 Explanation of the source code for nonlinear optimal control

In this section, the source code has been explained in detail for all important steps involved in formulating and solving a nonlinear optimal control problem.

Step 1: Set a simulation environment

Define the prediction horizon length, the sampling time, initial values of the states of the process, and the initial guess for the variables to be optimized, which in this example is the valve opening.

```
clc
clear

%choose prediction horizon Np
Np = 136; %Np time steps ahead in the future
%sampling time
dt = 0.1*60; %sampling time in seconds %dt = 6 seconds

%initial values of the states
state_ini_values = 10e5; %intial pressure at the bottom of the tank

%initial value for optimizer
u_ini = ones(Np,1).*1; %the unknowns are initialized as full valve opening
```

Step 2: Define the reference for the whole prediction horizon length

The references are known variables and they should be defined for the whole prediction horizon length.

```
%Reference
%make the reference vector offline (for the whole prediction horizon length).
Ref = [ones(Np/4,1)*10e5;ones(Np/4,1)*12e5;ones(Np/4,1)*16e5;ones(Np/4,1)*11e5];
```

Here, the references are changed in step. So, specifying the changed reference values in a vector is simple and direct. If the references are to be changed in other ways (like ramped up/down, sinusoidal etc.) then you may also use a *for loop* (an offline loop) for generating the values of the references for the whole prediction horizon length.

Example:

```
%Reference
%make the reference vector offline (for the whole prediction horizon length).
for i = 1: N
       Ref(i,1) = sin(…);
      %or
       if i>= 1 && i<=5
          Ref(I,1) =  …
       elseif if i>5&& i<=10
          Ref(I,1) =  …
       else
          Ref(I,1) =  …
       end
end
```

Step 3: Make a nonlinear optimization problem and solve it using *fmincon*

The nonlinear optimization problem is created inside the user defined MATLAB script `optimization_tank.m`

```
%make the nonlinear optimization problem and solve it
u_k_ast = optimization_tank(u_ini,state_ini_values,dt,Ref,Np);
```

To this function, the initial guess for the optimizer `u_ini` , the initial values of the states `state_ini_values` , sampling time `dt` , the values of the reference `Ref` for the whole prediction horizon `Np` are passed. These information are needed to make the optimal control problem inside this function. If other variables are needed, please pass it into this function.

Now inside this function `optimization_tank`, the handles to create the objective function and the constraints are defined.

```
%define the function where the objective function will be calculated
obj_func = @(u)objfun_tank(u,state_ini_values,dt,Ref,Np);
%define the function where the nonlinear constraints will be
%calculated (both equality and inequality constraints will be calculated in
%the same function
cons_func = @(u)confun_tank(u,state_ini_values,dt,Ref,Np);
```

`objfun_tank` is the function to create the objective, and `confun_tank` is the function to create the nonlinear constraints.
In addition, the options for the solver to select SQP as the algorithm are also set.

```
%options for chosing the type of method for optimization and other options
ops = optimset('Algorithm','sqp','Display','off','MaxIter',5000);
```

The process model has to be solved using a suitable ODE solver to make the objective. Now let us assume that the same process model has to be solved again to create the nonlinear constraints. In this particular example, we do not need to do so, but let us assume so for general explanation and concept. Under such condition, the same process model has to be solved twice (once for the objective function, and second time for the constraints) for the same control input sequence which is clearly a repetitive task. It is possible to create both the objective function and the constraints at the same time by solving the process model only once. To do so, we can make use of nested function. Then, both the objective and the nonlinear constraints can be computed in the `compute_both` function.

```
uLast = [];% Last place compute_both was called
myJ = [];% Use for objective at xLast
myG = [];% Use for nonlinear inequality constraint
myHeq = [];% Use for nonlinear equality constraint


function J = objfun_tank(u,state_ini_values,dt,Ref,Np)
        if ~isequal(u,uLast) %check if computation is necessary
%             disp('button pressed: objective call');
%             pause;
            [myJ myG myHeq] = compute_both(u,state_ini_values,dt,Ref,Np);
            uLast = u;
        end
        %now compute objective function
        J = myJ;
    end

    function [G Heq] = confun_tank(u,state_ini_values,dt,Ref,Np)
        if ~isequal(u,uLast) %check if computation is necessary
%             disp('button pressed: constraint call');
%             pause;
            [myJ myG myHeq] = compute_both(u,state_ini_values,dt,Ref,Np);
            uLast = u;
        end
        %now compute constraints
        G = myG;
        Heq = myHeq;
```

```
    end
```

Then the fmincon solver is called to solve the constrained nonlinear optimization problem.

```
%use the fmincon solver
[u,fval,exitflag,output,solutions] = fmincon(obj_func,u_ini,[],[],[],[],[],[],cons_func,ops);
```

All the actions needed to make the objective function and the nonlinear constraints happen inside the `compute_both` function.

```
function [myJ myG myHeq] = compute_both(u_ini,state_ini_values,dt,Ref,Np)
```

Inside this function, the weighting matrices have been defined as,

```
%weighting matrices for error and control inputs
Qe = eye(Np).*1; %weighting matrix for the error
Pu = eye(Np).*1; %weighting matrix for the control inputs
```

These matrices could also have been passed into this function from outside.
To make the objective function

$$\min_{u} \; J = \frac{1}{2}\sum_{k=1}^{N_p}\left(r_k - P_{c,k}\right)^T Q\left(r_k - P_{c,k}\right) + u_{k-1}^T P u_{k-1}$$

we need to know $r_k, P_{c,k}$ and u_{k-1} for the whole prediction horizon from $k = 1$ to $k = N_p$. The reference values r_k is known as this is passed into the `compute_both` function as `Ref` . Similarly, the control sequence $u_{k-1}$is known since this is passed into the `compute_both` function as `u_ini`. However, the process output $P_{c,k}$ is not known and this has to be predicted throughout the horizon using the model of the process. For this the ODE (model of the process) have to be solved by using the `state_ini_values` as the initial value for solving the ODE. This can be done using a *for loop* as,

```
for i = 1:Np

    %find out which control input to use for each time step within the prediction horizon.
    u_k = u_ini(i,:);

    %use runge kutta to update the states
    x_next = my_runge_kutta(state_ini_values,dt,u_k);
    %use the states to calculate the output
    %in this case, the output is simply the state.
    %//Note: Sometimes outputs have to be calculated using algebriac
    %equations and the states. In such a case, please make a separate
    %function for calculating the output variables.
    Pc(i,1) = x_next;

    %update the state
    state_ini_values = x_next;
end
```

To solve the model, we have to use the control input sequence that is passed into this function. During the internal iteration of the optimization, the optimizer will internally change the values of `u_ini` , i.e. the optimizer will try to adjust `u_ini` to minimize the objective function while still satisfying the constraints. So we excite the system with `u_ini` to predict the future values of the process outputs. Inside the loop, choose the control input for the corresponding time step from $k = 1$ to $k = N_p$

```
u_k = u_ini(i,:);
```
Then solve the ODE using Runge Kutta 4th order algorithm.
```
x_next = my_runge_kutta(state_ini_values,dt,u_k);
```

You can also use other ODE solvers like Euler method, trapezoidal methods, etc.
The goal is to predict the values of process output $P_{c,k}$. For this example, the process output is the state. So the values of the state at each time step within the prediction horizon can be directly stored.
```
Pc(i,1) = x_next;
```

If the measurement equation for the process output is some nonlinear function of the states, i.e. if $y_k = g(x_k, u_k)$ this has to be calculated inside this loop. Similarly, if there are constraints which are some nonlinear function of the states and control inputs, then they can also be calculated inside this loop. This way you can calculate the outputs and the nonlinear constraints (if any) for each time step within the prediction horizon and store it. The stored values of the constraints can later be used to make the constraints. For these purposes, you can make separate MATLAB scripts (separate .m files) and call the scripts from inside the loop.

Then we are ready to create the objective function.
```
%now make the objective function
J = (Ref-Pc)'*Qe*(Ref-Pc) + u_ini'*Pu*u_ini;

myJ = J/2;
```
The nonlinear constraints (both equality and inequality constraints) can then be formulated as,
```
%if there are equaltiy constraints, it should be listed as a column vector
%here we don't have equality constraints so we use empty matrix
myHeq = [];

%For this example, there is also constraint on the rate of change of control input variables (du)
%such that -0.1<=du<=0.1
%since in the objective function, we don't have 'du' but only 'u', we have to calculate 'du'
ourself.
%find du from the input signals
du = u_ini(2:end)-u_ini(1:end-1);
%//Important Note: If the objective function was formulated such that it
%had " (du)'P du  ",then we don't have to calculate 'du' here because 'du' would be passed to
%this function "compute_both" by the optimizer and so we already would have it.

%list the inequality constraints as column vector
myG =[u_ini-1; % valve opening should be less than 1
      -u_ini+0; %valve opening should be greater than 0
      du - (0.1); % du should be less than 0.1 for each sample
      -du-(0.1);  % du should be greater than -0.1 for each sample
      ];
```

In this particular example, the variable to be optimized is the control input u. Thus, the rate of change of control inputs are calculated as,
```
du = u_ini(2:end)-u_ini(1:end-1);
```

If the objective function was formulated such that it had Δu instead of u as the variable to be optimized, then we don't have to calculate the rate of change of control as shown above. In this case, would be passed into the function directly.

All the constraints should be constructed as a column vector. For example, `u_ini-1` means,

$$u \leq 1$$

Similarly, `-du-(0.1)` means,

$$-\Delta u \leq 0.1 \quad \text{i.e.} \quad -0.1 \leq \Delta u$$

If the nonlinear constrains were calculated and stored inside the loop for solving the process model, they can be used here to make the constraints.

Step 4: Use the optimal values of the control sequence returned by *fmincon*

The *fmincon* solver will calculate the optimal values of the control sequence `u_k_ast` for the whole prediction horizon. For optimal control problems (without receeding horizon strategy), we will apply all the control sequence obtained by solving the optimization problem to the process. Here a *for loop* is used to apply all the control sequence in order to see the controlled process outputs and states. Once again, Runge Kutta algorithm is used to solve the process model i.e. to excite the system with optimal control sequence and to update it.

```
%make the nonlinear optimization problem and solve it
u_k_ast = optimization_tank(u_ini,state_ini_values,dt,Ref,Np);


%we now have optimal values of 'u' to fulfill the objective function
%we can use the optimal values of 'u' to calculate the variables of interest
%to us.
%the ouput variables and the states are of interest. They should be
%calculated using the optimal values of 'u'. We use the model to calculate
%the output and the states.

%for storing
%storage place for the output
Pc = zeros(Np,1);

%Important note: We don't need this loop for MPC, only for NL opt. control
for i = 1:Np

    %calculate the outputs: for this example, outputs are the states
    %store the outputs
    Pc(i,1) = state_ini_values;

    %update the state with the optimal control inputs
    %with RK method
    x_next = my_runge_kutta(state_ini_values,dt,u_k_ast(i));
    state_ini_values = x_next;

end
```

Step 5: Plot the results

The results are then plotted as MATLAB figures.

```
%make time steps for plotting
tspan = linspace(0,Np-1,Np);
figure,
subplot(211)
plot(tspan,Ref,'b-',tspan,Pc,'k-')
legend('ref Pc','Pc','Orientation','horizontal')
ylabel('Pc, Ref'); title('NL optimal control of tank pressure');
subplot(212)
plot(tspan,u_k_ast,'r-')
xlabel('time [sec]'); ylabel('u');
legend('Control input: valve opening');
```

8.8 Nonlinear MPC

To create a nonlinear MPC, receeding horizon strategy is applied to the nonlinear optimal control problem which is already explained in detail in section 8.6. Similarly, the receeding horizon strategy has already been explained in detail in lecture 4. Thus, the details of the problem formulation (the nonlinear optimal control problem) + the receeding horizon are not explained in this section. In MATLAB script, a *for loop* can be used for implementing the receeding horizon strategy to the nonlinear optimal control problem.

The problem formulation remains the same as for the nonlinear optimal control problem. For simplicity it is re-written here for setpoint tracking optimal control problem by making use of a running variable i to denote the current prediction horizon length as,

$$\underset{z}{\text{Minimize}} \quad J = \frac{1}{2}\sum_{i=k}^{k+N_p} e_i^T\, Q e_i + u_{i-1}^T\, P u_{i-1} + \Delta u_{i-1}^T\, R \Delta u_{i-1} \tag{8.58}$$

subject to,

$$\mathrm{h}_l(z) - b_l = 0,\ l = 1,2,\dots\dots,m \tag{8.59}$$

$$\mathrm{g}_j(z) - c_j \leq 0,\ j = 1,2,\dots\dots,r \tag{8.60}$$

As also explained in detail in Section 8.6, for nonlinear MPC, the **decision variable z can simply contain only u or Δu.**

The main algorithmic steps for implementing a nonlinear MPC are:

a) Choose an initial value of the states x_0 and the initial guess u_{ini} of decision variables (control inputs) for the optimizer. Set the start time $t_k = 0$, for $k = 0$.
b) Using x_k at current time t_k and the nonlinear model of the process, create an optimal control problem for the given prediction horizon length N_p.
c) Solve the optimal control problem using for example *fmincon* solver in MATLAB and obtain the optimal control sequence u_i for the whole prediction horizon for $i = k$ to $i = k + N_p$.
d) Select the first control move u_k for $i = k$ at current time t_k.
e) Warm start: Update the initial guess for decision variables as, $u_{ini} = u_i$ for $i = k$ to $i = k + N_p$.
f) Apply the first control move u_k to the process and update the state $x_{k+1} = f(x_k, u_k)$.
g) Move one time step forward: Set, $k = k + 1$ and $x_k = x_{k+1}$.
h) Go to step (b) until the end of the simulation time.

Note:

Just like for the linear output feedback MPC, if some of the states of the system are not measurable, then to make an output feedback nonlinear MPC, a state estimator should be utilized to estimate the unmeasured states. The estimated states are then used further to create an optimal control problem. Please look lecture 6 for details.

8.8.1 Example of a Nonlinear MPC

Let us again consider the previous example of controlling the pressure at the bottom of a tank. In Section 8.7, a nonlinear optimal controller (openloop dynamic optimization as control problem) for this process was designed and implemented in MATLAB. Please look section 8.7 for the description of the process.

In this example, we will be introducing feedback to the nonlinear optimal controller by using the sliding horizon strategy. In MATLAB, we will make use of a *for loop* for sliding forward in time with a step size of dt. At each time step, we will use only the first control input to slide forward by one time step.

Since the only thing we are adding to the openloop dynamic optimization problem (of section 8.7) is the sliding horizon strategy, the formulation of the optimal control remains the same as before. For easiness it has been repeated here and a running variable i has been used to denote the whole prediction horizon length.

The nonlinear MPC problem for controlling the pressure at the bottom of the tank can be written as,

$$\min_{u} \; J = \frac{1}{2}\sum_{i=k}^{k+N}\left(r_i - P_{c,i}\right)^T Q\left(r_i - P_{c,i}\right) + u_{i-1}^T P u_{i-1} \tag{8.61}$$

subject to,

$$u_k - 1 \leq 0 \tag{8.62}$$
$$-u_k + 0 \leq 0 \tag{8.63}$$
$$\Delta u_k - 0.1 \leq 0 \tag{8.64}$$
$$-\Delta u_k + (-0.1) \leq 0 \tag{8.65}$$

Where $P_{c,i}$ is the predicted pressure at the bottom of the tank (please see the description of the process in section 8.7) for the whole prediction horizon from $i = k$ to $i = k + N_p$. $P_{c,i}$ is obtained by solving the process model given by the following ODE.

$$\frac{dP_c}{dt} = \frac{1}{C_f}Q_{in}(t) - \frac{K_v u}{R_f C_f}P_c \qquad \text{with initial state } P_{c0} \text{ known} \tag{8.66}$$

The complete source code for the nonlinear MPC for this example can be downloaded from the homepage of the course. The MATLAB code is also available at end of this section. Here, the main steps involved in making the nonlinear MPC will be provided.

Step 1: Set a simulation environment

Define the prediction horizon length, the sampling time, initial values of the states of the process, and the initial guess for the variables to be optimized, which in this example is the valve opening. Let us assume that the prediction horizon is $N_p = 20$ samples into the future, sampling time $dt = 6$ sec and ending time for simulation is `tend` $= 14$ min $= 14 \times 60$ sec. We can then

define the timespan (i.e. the time vector) as a vector and then calculate the length/size of this time vector to know how many number of times the sliding has to be performed to reach the end of simulation time.

```
%start time
tstart = 0;
tend = 14*60; %in sec
%sampling time
dt = 0.1*60; %sampling time in seconds %dt = 6 seconds
%time span
tspan = tstart:dt:tend;
%length of the sliding loop: how many times to slide
looplen = length(tspan);

%choose prediction horizon
Np = 20; %Np time steps ahead in the future
```

The ordinary differential equation that represents the model of the process needs an initial value `state_ini_values`. Let us assume that initial value of the pressure for solving the ODE is 10×10^5 N/m^2. Similarly, the optimizer also needs a guess/starting value `u_ini` for the variables to optimize Let us put the guess that the valve is at fully opened condition.

```
%initial values of the states
state_ini_values = 10e5; %intial pressure at the bottom of the tank

%initial value for optimizer
u_ini = ones(Np,1).*1; %full valve opening
```

Step 2: Define the reference for the whole simulation time

Since reference values are known variables, they can be calculated offline (i.e. before entering into the main sliding loop). You can make a separate .m function for calculating the known variables (like reference values and/or input disturbances) for the whole simulation time (and not just for prediction horizon length). For this example, a separate `make_reference.m` function is created.

```
%make the reference vector offline (for the whole simulation time).
Ref = make_reference(tspan);
```

Inside this function, the references for the whole simulation time length (which is the *tspan*) is defined as,

```
function Ref = make_reference(tspan)

Ref = zeros(length(tspan),1);
for i = 1:length(tspan)
    if tspan(i) <=2*60
        Ref(i,1) = 10e5;
    elseif tspan(i) >2*60 && tspan(i) <=6*60
        Ref(i,1) = 13e5;
    elseif tspan(i) >6*60 && tspan(i) <=10*60
        Ref(i,1) = 11e5;
    else
        Ref(i,1) = 15e5;
    end
end
```

Here, a simple, if-else statements have been used for defining references for the whole simulation time length as a simple example. Other ways can also be explored.

Note:

Let us look at the timeline of Figure 8.9. In this figure, tend is the end of the simulation time, and N_p is the prediction horizon length. Using the `make_reference` function, the values for the references from time $t = 0$ to $t =$ tend is already defined.

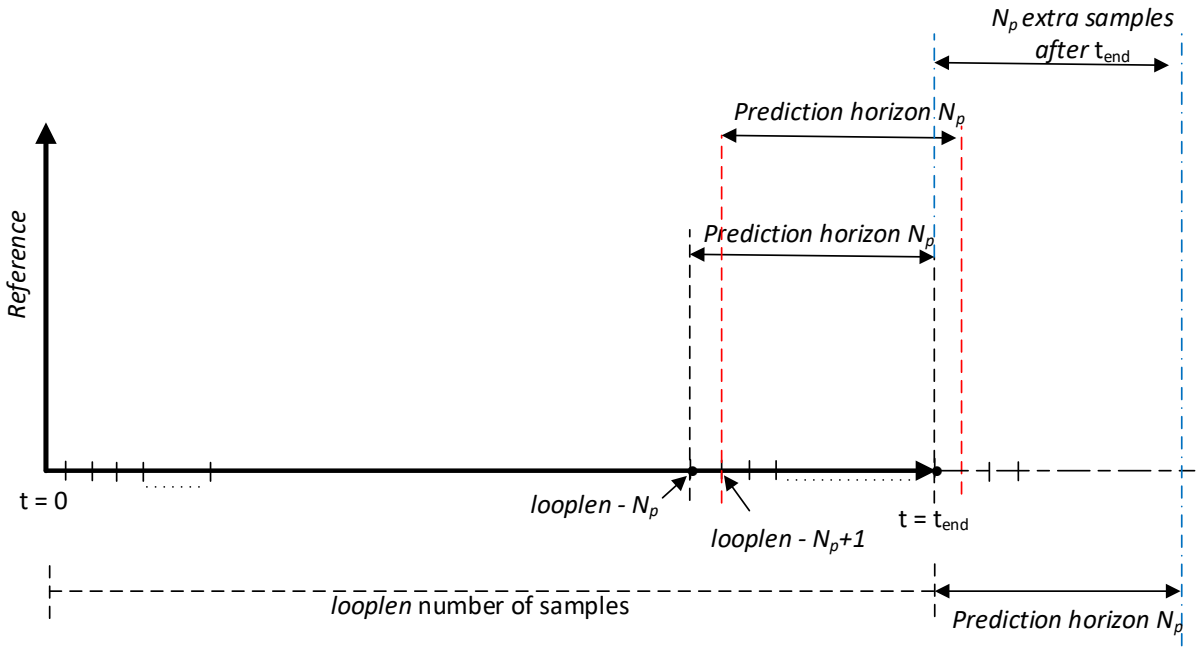


Figure 8.9: Timeline to explain why N_p extra samples are needed after tend.

However, when we reach to a time point which is at $(looplen - N_p)$ i.e. when we are exactly at N time steps before the end of the simulation time, we can see that the end of the prediction horizon from this point will coincide with the end of the simulation time. Now when you want to slide one time step further to $(looplen - N_p + 1)$ time point, you can see that the prediction horizon window extends one time step further from the end of the simulation time. But we have not defined any values for the references for this extra time step. We must define it.

In the same way, if would like to simulate/slide until the end of the simulation time tend, we can see that we need N_p extra/additional values for the references after the end of the simulation time. Thus we need to provide information about these extra N_p values for the references (and/or input disturbances if present). One way to do so is to use the value of the reference defined for the end of the simulation time (i.e. the last value of the reference) and extend it for the extra N_p values. For example,

```
%add the extra Np number of values for simulating it until "tend"
Ref = [Ref;ones(Np,1).*Ref(end)];
```

However, as an alternative, we could have stopped the simulation when we have reached the timepoint $(looplen - N_p)$. This way we could have only performed simulation from `tstart` $(t = 0)$ to $(\texttt{tend} - N_p * dt)$, and not completely until the defined `tend`.

In addition, as another alternative, we also could have stopped sliding further when we have already slid for $(looplen - N_p)$ number of times. If we do so, the last N optimal values of the control inputs are obtained as the solution of the optimal control problem which was solved at timepoint $(looplen - N_p)$. Then all these last N_p optimal values of the control inputs could also be applied to the process to simulate the remaining N number of steps so that we ultimately reach `tend`.

Which of these alternatives to use?: Well, the choice is yours.

In this example, I have implemented the first choice (adding the extra N_p values for the references by extending the last value of the references).

Step 3: Make a *for loop* for applying the receeding horizon strategy to the nonlinear optimization problem

The main sliding loop can be formed as,

```
%make a loop for sliding each time step forward
for i = 1:looplen
    %store the output
    %if output is not directly the states, make a separate .m file for calculating
    %it.
    Pc(i,1) = state_ini_values;


    %step (b) and step (c) of the algorithmic steps in Section 8.8
    %call the optimizer
    %make the nonlinear optimization problem and solve it
    u_k_ast = optimization_tank(u_ini,state_ini_values,dt,Ref(i:i+Np-1,1),Np);


    %step (d) of the algorithmic steps in section 8.8
    %use only the first control move
    u_first = u_k_ast(1);

    %store the optimal control input for plotting
    u(i,1) = u_first;


    %step (e)of the algorithmic steps in section 8.8
    %for warm start
    u_ini = u_k_ast;


    %step (f) of the algorithmic steps in section 8.8
    %use the first control move to slide one time step forward
    %with RK method
    x_next = my_runge_kutta(state_ini_values,dt,u_first);

    %step (g) of the algorithmic steps in section 8.8
    state_ini_values = x_next;
end
```

Inside this loop, all the algorithmic steps (from step b to step g) explained in Section 8.8 for nonlinear MPC are implemented. Let's look at these steps one by one.

At first, for the purpose of plotting later, the output variables can be saved inside this loop. For this example the output is the state so it can be saved directly as,

```
%store the output
    %if output is not directly the states, make a separate .m file for calculating
    %it.
    Pc(i,1) = state_ini_values;
```

If the output is some nonlinear functions of the state and control inputs, $y_k = g(x_k, u_k)$, then you should calculate the output. For this you can for example make a separate .m file/script.

Step (b) and step (c) of the algorithmic steps in section 8.8:

A nonlinear optimal control problem is formulated and solved using *fmincon* solver in MATLAB at each time step as,

```
%step (b) and step (c) of the algorithmic steps in Section 8.8
    %call the optimizer
    %make the nonlinear optimization problem and solve it
    u_k_ast = optimization_tank(u_ini,state_ini_values,dt,Ref(i:i+Np-1,1),Np);
```

When formulating the nonlinear optimal control problem at any given time step, make sure that you define the values for the reference for the whole prediction horizon starting from that particular time step. For this you can extract the values for the references belonging to that particular prediction horizon length from the variable `Ref`. Remember that the variable `Ref` contains the values for the references for all the simulation time + extra N_p for extending the horizon after the end of simulation. Selection of suitable values for reference at each time step has been done using proper indexing as,

```
Ref(i:i+Np-1,1)
```

Just note that:

> If some input disturbances are present in your system and they have been defined offline (just like for the output references) for the whole simulation time + extra N_p , then you also extract the values for the input disturbances (for the prediction horizon length) at any given current time step by using proper indexing (just like for the output references).

In addition, you need to supply the initial state values `state_ini_values` (current state values at any given time step) and the initial guess for the optimizer `u_ini`.

What happens inside the `optimization_tank` function has already been described in details in Section 8.7.1 (under step 3). This has not been repeated here since it is exactly the same i.e. a nonlinear optimal control problem is formed for a given prediction horizon length and solved using *fmincon* solver in MATLAB. When this `optimization_tank` function is placed inside the *for loop,* a nonlinear

optimal control problem is re-formed, re-optimized and re-solved at each time step as we slide forward (i.e. as the loop progresses).

The optimizer will return back N_p optimal values of the control inputs denoted here as `u_k_ast` .

Step (d) of the algorithmic steps in section 8.8:

However, instead of applying all these N_p optimal values, we only select the first control move (denoted here as `u_first`) for all the control inputs. They are then stored for plotting later on.

```
%step (d) of the algorithmic steps in section 8.8
%use only the first control move
u_first = u_k_ast(1);

%store the optimal control input for plotting
u(i,1) = u_first;
```

Step (e) of the algorithmic steps in section 8.8:

For warm start, the current optimal values (`u_k_ast`) returned by the optimizer is allocated as the starting point or initial guess (`u_ini`) for the optimizer for the next time step i.e. for re-solving an optimal control problem in the next time step or next iteration of the loop.

```
%step (e)of the algorithmic steps in section 8.8
    %for warm start
    u_ini = u_k_ast;
```

Step (f) of the algorithmic steps in section 8.8:

The first control move for all the control inputs are then applied to the nonlinear process to move one time step forward. Since the nonlinear process is described by ODE, we have to solve the model equation by using `u_first` and the known initial values `state_ini_values` of the system at the current time step. To solve the ODE, Runge Kutta 4th order algorithm have been used in this example as,

```
%step (f) of the algorithmic steps in section 8.8
    %use the first control move to slide one time step forward
    %with RK method
    x_next = my_runge_kutta(state_ini_values,dt,u_first);
```

You can also use other ODE solvers like Euler method, trapezoidal methods, etc. Just avoid using the built-in solvers in MATLAB inside a loop (it will make execution very slow).

Step (g) of the algorithmic steps in section 8.8:

The states are updated i.e. we move one time step forward by setting $k = k + 1$.

```
%step (g) of the algorithmic steps in section 8.8
    %update the system, set k = k+1
    state_ini_values = x_next;
```

After this step (g), the whole process of re-formulating, re-optimizing and re-solving of a nonlinear optimal control problem is repeated at each time step by the *for loop*. This creates

feedback to the system. This also turns a nonlinear optimal control problem into a nonlinear MPC.

Step 4: Plot the results

After the loop has terminated, we can finally plot the variables that are of interests to us. The results are plotted as MATLAB figures. Figure 8.10 shows the output variable, the reference and the control input.

```
tspan = tspan./60; %to plot in minutes as time axis
%plot figures,
figure,
subplot(211)
plot(tspan,Ref(1:looplen,1),'b-',tspan,Pc,'k-')
legend('ref Pc','Pc','Orientation','horizontal')
ylabel('Pc, Ref'); title('MPC for tank pressure');
subplot(212)
plot(tspan,u,'r-')
xlabel('time [sec]'); ylabel('u');
legend('Control input: valve opening');
```

As you may have observed, the major part of the code remains the same as for NLP optimal control (Section 8.7). The only change (difference) is that a *for loop* has been utilized to implement the sliding horizon strategy.

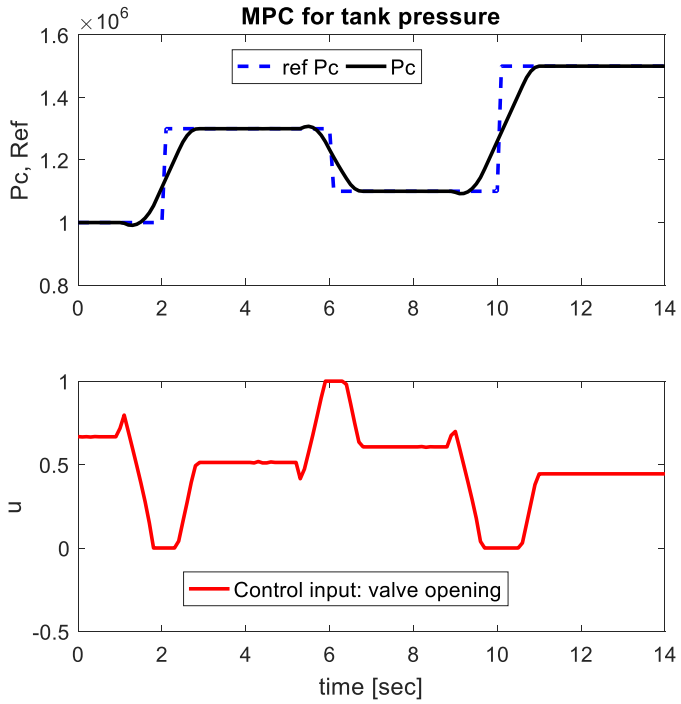


Figure 8.10: Nonlinear Model Predictive control of pressure at bottom of a tank

The complete MATLAB source code for this example of NMPC is given below:

<%the main file to be run> % main_file_tankPressure_MPC.m

```
%the main file to be run

clc
clear

%start time
tstart = 0;
tend = 14*60; %in sec
%sampling time
dt = 0.1*60; %sampling time in seconds %dt = 6 seconds
%time span
tspan = tstart:dt:tend;
%length of the sliding loop: how many times to slide
looplen = length(tspan);

%choose prediction horizon
Np = 20; %Np time steps ahead in the future

%initial values of the states
state_ini_values = 10e5; %intial pressure at the bottom of the tank

%initial guess for the optimizer
u_ini = ones(Np,1).*1; %full valve opening

%Reference
%make the reference vector offline (for the whole simulation time).
Ref = make_reference(tspan);
%add the extra Np number of values for simulating it until tend
Ref = [Ref;ones(Np,1).*Ref(end)];

%make space for storing variables of interest
Pc = zeros(looplen,1);
u = zeros(looplen,1);

%make a loop for sliding each time step forward
for i = 1:looplen
    %store the output
    %if output is not directly the states, make a separate .m file for calculating
    %it.
    Pc(i,1) = state_ini_values;


    %step (b) and step (c) of the algorithmic steps in Section 8.8
    %call the optimizer
    %make the nonlinear optimization problem and solve it
    u_k_ast = optimization_tank(u_ini,state_ini_values,dt,Ref(i:i+Np-1,1),Np);


    %step (d) of the algorithmic steps in section 8.8
    %use only the first control move
    u_first = u_k_ast(1);

    %store the optimal control input for plotting
    u(i,1) = u_first;


    %step (e)of the algorithmic steps in section 8.8
    %for warm start
    u_ini = u_k_ast;


    %step (f) of the algorithmic steps in section 8.8
    %use the first control move to slide one time step forward
    %with RK method
    x_next = my_runge_kutta(state_ini_values,dt,u_first);

    %step (g) of the algorithmic steps in section 8.8
    state_ini_values = x_next;
end

tspan = tspan./60; %to plot in minutes as time axis
%plot figures,
```

```
figure,
subplot(211)
plot(tspan,Ref(1:looplen,1),'b-',tspan,Pc,'k-')
legend('ref Pc','Pc','Orientation','horizontal')
ylabel('Pc, Ref'); title('MPC for tank pressure');
subplot(212)
plot(tspan,u,'r-')
xlabel('time [sec]'); ylabel('u');
legend('Control input: valve opening');
```

<function: main_file_tankPressure_MPC.m: ends>%

---------- x----------x----------

<function: make_reference.m: starts>%

```
function Ref = make_reference(tspan)

Ref = zeros(length(tspan),1);
for i = 1:length(tspan)
    if tspan(i) <=2*60
        Ref(i,1) = 10e5;
    elseif tspan(i) >2*60 && tspan(i) <=6*60
        Ref(i,1) = 13e5;
    elseif tspan(i) >6*60 && tspan(i) <=10*60
        Ref(i,1) = 11e5;
    else
        Ref(i,1) = 15e5;
    end
end
```

<function: make_reference.m: ends>%

---------- x----------x----------

<function: optimization_tank.m: starts>%

```
function u = optimization_tank(u_ini,state_ini_values,dt,Ref,Np)
%In this function we:
%define functions for objective and constraints
%choose the slover types + other options to the solver

%######## Structure 2 ############################## %efficient
%reduces computation time by half
%We can cut the computational time by NOT repeating the same calculations
%for objective function and the nonlinear constraints
%let us try by making use of nested function

%options for chosing the type of method for optimization and other options
ops = optimset('Algorithm','sqp','Display','off','MaxIter',5000);
%ops =optimset('Algorithm','interior-point','Display','off','MaxIter',5000);
%ops = optimset('Algorithm','active-set','Display','off','MaxIter',20);


uLast = [];% Last place compute_both was called
myJ = [];% Use for objective at xLast
myG = [];% Use for nonlinear inequality constraint
myHeq = [];% Use for nonlinear equality constraint

%define the function where the objective function will be calculated
obj_func = @(u)objfun_tank(u,state_ini_values,dt,Ref,Np);
%define the function where the nonlinear constraints will be
%calculated (both equality and inequality constraints will be calculated in
%the same function
```

```
cons_func = @(u)confun_tank(u,state_ini_values,dt,Ref,Np);

%use the fmincon solver
[u,fval,exitflag,output,solutions] = fmincon(obj_func,u_ini,[],[],[],[],[],[],cons_func,ops);


    function J = objfun_tank(u,state_ini_values,dt,Ref,Np)
        if ~isequal(u,uLast) %check if computation is necessary
%             disp('button pressed: objective call');
%             pause;
            [myJ myG myHeq] = compute_both(u,state_ini_values,dt,Ref,Np);
            uLast = u;
        end
        %now compute objective function
        J = myJ;
    end

    function [G Heq] = confun_tank(u,state_ini_values,dt,Ref,Np)
        if ~isequal(u,uLast) %check if computation is necessary
%             disp('button pressed: constraint call');
%             pause;
            [myJ myG myHeq] = compute_both(u,state_ini_values,dt,Ref,Np);
            uLast = u;
        end
        %now compute constraints
        G = myG;
        Heq = myHeq;
    end
end

%########## Structure 2 ENDS #################################
```

<function: optimization_tank.m: ends>%

---------- x----------x----------

<function: compute_both.m: starts>%

```
function [myJ myG myHeq] = compute_both(u_ini,state_ini_values,dt,Ref,Np)
%this is the function where we compute the objective function and the
%constraints.

%weighting matrices for error and control inputs
Qe = eye(Np).*1; %weighting matrix for the error
Pu = eye(Np).*1; %weighting matrix for the control inputs

%to store the output variable
Pc = zeros(Np,1);

for i = 1:Np

    %find out which control input to use for each time step within the prediction horizon.
    u_k = u_ini(i,:);

    %use runge kutta to update the states
    x_next = my_runge_kutta(state_ini_values,dt,u_k);
    %use the states to calculate the output
    %in this case, the output is simply the state.
    %//Note: Sometimes outputs have to be calculated using algebriac
    %equations and the states. In such a case, please make a separate
    %function for calculating the output variables.
    Pc(i,1) = x_next;

    %update the state
    state_ini_values = x_next;
end

%now make the objective function
J = (Ref-Pc)'*Qe*(Ref-Pc) + u_ini'*Pu*u_ini;

myJ = J/2;
```

```
%if there are equaltiy constraints, it should be listed as a column vector
%here we don't have equality constraints so we use empty matrix
myHeq = [];

%For this example, there is also constraint on the rate of change of control input variables (du)
%such that -0.1<=du<=0.1
%since in the objective function, we don't have 'du' but only 'u', we have to calculate 'du'
ourself.
%find du from the input signals
du = u_ini(2:end)-u_ini(1:end-1);
%//Important Note: If the objective function was formulated such that it
%had " (du)'P du  ",then we don't have to calculate 'du' here because 'du' is passed to this
%function by the optimizer and so we already would have it.

%list the inequality constraints as column vector
myG =[u_ini-1; % valve opening should be less than 1
      -u_ini+0; %valve opening should be greater than 0
      du - (0.1); % du should be less than 0.1 for each sample
      -du-(0.1);  % du should be greater than -0.1 for each sample
      ];
```

<function: compute_both.m: ends>%

---------- x----------x----------

<function: my_runge_kutta.m: starts>%

```
function x_next = my_runge_kutta(states_ini_values,dt,u_k_ast)

K1 = tankPressure_equations([],states_ini_values,u_k_ast);
K2 = tankPressure_equations([],states_ini_values+K1.*(dt/2),u_k_ast);
K3 = tankPressure_equations([],states_ini_values+K2.*(dt/2),u_k_ast);
K4 = tankPressure_equations([],states_ini_values+K3.*dt,u_k_ast);

x_next = states_ini_values + (dt/6).*(K1+2.*K2+2.*K3 + K4);
```

<function: my_runge_kutta.m: ends>%

---------- x----------x----------

<function: tankPressure_equations.m: starts>%

```
function dP_dt = tankPressure_equations(t,x0,u)
%this is the function where we write the differential equations of the
%given model.

%needed parameters
Cf = 2.04e-4;
Rf = 1e6;

Qin = 1; %constant inflow
Kv = 1.5; %gain of the valve
dP_dt = Qin/Cf - (x0*u*Kv)/(Rf*Cf);
```

< tankPressure_equations.m: ends>%

---------- x----------x----------

8.8.2 Grouping of control inputs for faster execution time

Perhaps the most efficient way of reducing the executing time for nonlinear MPC is by grouping the control inputs into blocks. Let the number of control signals available for controlling a process is denoted by n_u and the prediction horizon is denoted by N_p. Then for each time step, the MPC algorithm has to calculate $n_u N_p$ number of optimal control signals for the whole prediction horizon length. The greater the number of variables to optimize, the greater is the computational time needed by the optimization solver to solve the problem. However, it can be significantly reduced by grouping of control inputs into blocks.

Let us consider a process with three inputs as shown in Figure 8.9. For $n_u = 3$ & a prediction horizon of $N_p = 20$, the number of variables to optimize without grouping is $3 \times 20 = 60$ variables.

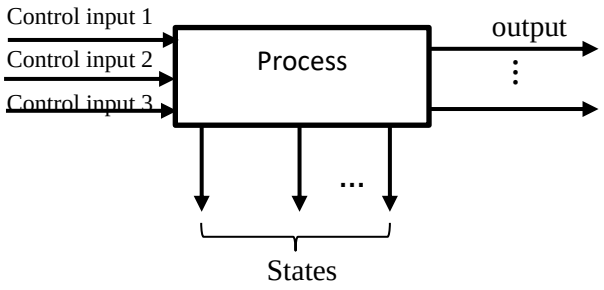


Figure 8.9: A MIMO system

Now, just for comparison, let us group each control input into two blocks[17]. For any time t_k and for the control input 1 as an example,

$$\text{Control input 1} = \left[\underbrace{\underbrace{u_k^1, u_{k+1}^1, u_{k+2}^1, \dots\dots\dots u_{k+8}^1, u_{k+9}^1}_{first\ 10\ signals\ (group\ 1)} \quad \underbrace{u_{k+10}^1, u_{k+11}^1, \dots\dots\dots u_{k+18}^1, u_{k+19}^1}_{remaining\ 10\ signals\ (group\ 2)}}_{Prediction\ horizon\ length\ N_p=20}\right]$$

Now suppose that for the first 10 time steps of the prediction horizon (group 1), we will use the same value for the control inputs. Denote it by u_{gr1}^1. Also suppose that for the remaining 10 time steps of the prediction horizon (group 2), let us use the same value for the control inputs. Denote it by u_{gr2}^1. Then in compact form we can write,

$$\text{Control input 1} = \left[u_{gr1}^1, \quad u_{gr2}^1\right]$$

Thus, 20 different values of control input 1 have been reduced to only 2 different values by grouping them into two blocks.

With the same logic, for the remaining control input signals,

$$\text{Control input 2} = \left[u_{gr1}^2, \quad u_{gr2}^2\right]$$

$$\text{Control input 3} = \left[u_{gr1}^3, \quad u_{gr2}^3\right]$$

[17] It is possible to group control inputs into more than two blocks. In general, the length of the latter blocks are shorter than the preceeding blocks.

Now after grouping, the total number of variables to optimize at any time t_k is (2 values for control input 1 + 2 values for control input 2 + 2 values for control input 3) i.e. 6 variables to optimize. Without grouping, there were 60 variables to optimize. Thus the total number of variables to be optimized by the solver has been reduced drastically. This results in quicker execution time when solving the optimization problem.
However, a question remains to be answered: Does the grouping of control inputs affect the closed loop response of the system? Well, the short answer is NO in general. Grouping of the control inputs into blocks does not severely affect the closed loop response. However, it is suggested that you test it for stability through simulations.

8.8.3 Example of Grouping of control inputs for faster execution time

To be consistent, let us once again look back into the previous example in section 8.8.2 for controlling pressure at the bottom of the tank. Here, a nonlinear MPC will be used. But this time, the control inputs will be grouped into 4 groups. If you recall from Section 8.8.2, without control input grouping, the total number of variables to optimize were 20 (given that that prediction horizon $N_p = 20$ and the number of control inputs present in the system $n_u = 1$).

If we group the sequence of control inputs within the prediction horizon into 4 groups, then we will have only 4 variables to optimize (instead of 20) at any given current time step. To implement the control input grouping, most of the code will remain the same as in Section 8.8.2 for nonlinear MPC without grouping. This section will only highlight those areas in the code implementation which needs to be changed or modified.

Let us group the control inputs within the prediction horizon of 20 samples as,

Grp1: from sample no. 1 to sample no. 2 (first part of the prediction horizon)

Grp 2: from sample no. 3 to sample no. 5 (second part of the prediction horizon)

Grp 3: from sample no. 6 to sample no. 11 (third part of the prediction horizon)

Grp 4: from sample no. 12 to sample no. 20 (fourth part of the prediction horizon)

At first, since with control input grouping we now have only 4 variables to optimize, the initial guess for the optimizer should be modified as,

```
%initial guess for the optimizer with control input grouping
Ngrp = 4;
u_ini = ones(Ngrp,1).*1; %full valve opening
```

The next change we have to make is during the use of process model for prediction i.e. when calculating $P_{c,i}$ for the whole prediction horizon from $i = k$ to $i = k + N_p$ at any given time t_k. $P_{c,i}$ is obtained by solving the process model given by the following ODE.

$$\frac{dP_c}{dt} = \frac{1}{C_f} Q_{in}(t) - \frac{K_v u}{R_f C_f} P_c \qquad \text{with initial state } P_{c0} \text{ known} \qquad 8.67$$

But when solving this model for the whole prediction horizon at any time, only 4 control inputs are available for 20 time steps of the prediction horizon (instead of $N_p = 20$ number of them). Thus we need to specify that:

> For the first 2 samples (from sample no. 1 to sample no. 2 within the prediction horizon), the control input corresponding to first group i.e. `u_ini(1,1)` should be used to solve the model (to excite the system). For the next 3 samples (from sample no. 3 to sample no. 5 within the prediction horizon), the control input corresponding to second group i.e. `u_ini(2,1)` should be used to solve the model (to excite the system). And similarly, for the next 6 samples (from sample no. 6 to sample no. 11 within the prediction horizon), the control input corresponding to third group i.e. `u_ini(3,1)` should be used to solve the model (to excite the system). Finally, for the last remaining 9 samples (from sample no. 12 to sample no. 20), the control input corresponding to the fourth group i.e. `u_ini(3,1)` should be used.

We can make use of *if-else* statement in MATLAB to specify the control inputs. To make this change, we can directly go to the `compute_both` function and specify the appropriate control inputs when solving the process model inside the *for lo*op. This can be performed as,

```
function [myJ myG myHeq] = compute_both(u_ini,state_ini_values,dt,Ref,Np)
%this is the function where we compute the objective function and the
%constraints.

%weighting matrices for error and control inputs
Qe = eye(Np).*1; %weighting matrix for the error
Pu = eye(4).*1; %weighting matrix for the control inputs

%to store the output variable
Pc = zeros(Np,1);

for i = 1:Np

    %find out which control input to use for each time step within the prediction horizon.
    %with control input grouping

    if i<=1   %for the first two samples within the prediction horizon
        u_k = u_ini(1,:);
    elseif i>1 && i<=3 %for the next three samples within the prediction horizon
        u_k = u_ini(2,:);
    elseif i>3 && i<=6 %for the next 6 samples within the prediction horizon
       u_k = u_ini(3,:);
    else               %for the remaining 9 samples witin the prediction horizon
        u_k = u_ini(4,:);
    end


    %use runge kutta to update the states
    x_next = my_runge_kutta(state_ini_values,dt,u_k);
    %use the states to calculate the output
    %in this case, the output is simply the state.
    %//Note: Sometimes outputs have to be calculated using algebriac
    %equations and the states. In such a case, please make a separate
    %function for calculating the output variables.
    Pc(i,1) = x_next;

    %update the state
    state_ini_values = x_next;
```

```
end

%now make the objective function
J = (Ref-Pc)'*Qe*(Ref-Pc) + u_ini'*Pu*u_ini;

myJ = J/2;

%if there are equaltiy constraints, it should be listed as a column vector
%here we don't have equality constraints so we use empty matrix
myHeq = [];

%For this example, there is also constraint on the rate of change of control input variables (du)
%such that -0.1<=du<=0.1
%since in the objective function, we don't have 'du' but only 'u', we have to calculate 'du'
ourself.
%find du from the input signals
du = u_ini(2:end)-u_ini(1:end-1);
%//Important Note: If the objective function was formulated such that it
%had " (du)'P du  ",then we don't have to calculate 'du' here because 'du' is passed to this
%function by the optimizer and so we already would have it.

%list the inequality constraints as column vector
myG =[u_ini-1; % valve opening should be less than 1
      -u_ini+0; %valve opening should be greater than 0
      du - (0.1); % du should be less than 0.1 for each sample
      -du-(0.1);  % du should be greater than -0.1 for each sample
      ];
```

A small change also has to be made when defining the weighting matrices for the control inputs in this function. Since after grouping, the number of control inputs (number of decision variables) are reduced to 3, the weighting matrix Pu has to be defined for these 3 control inputs as,

```
Pu = eye(3).*1; %weighting matrix for the control inputs with control input grouping
```

The remaining part of the code is the same as explained in section 8.8.2.

Figure 8.10 shows the computation time needed to complete the simulation both with and without grouping of the control inputs.

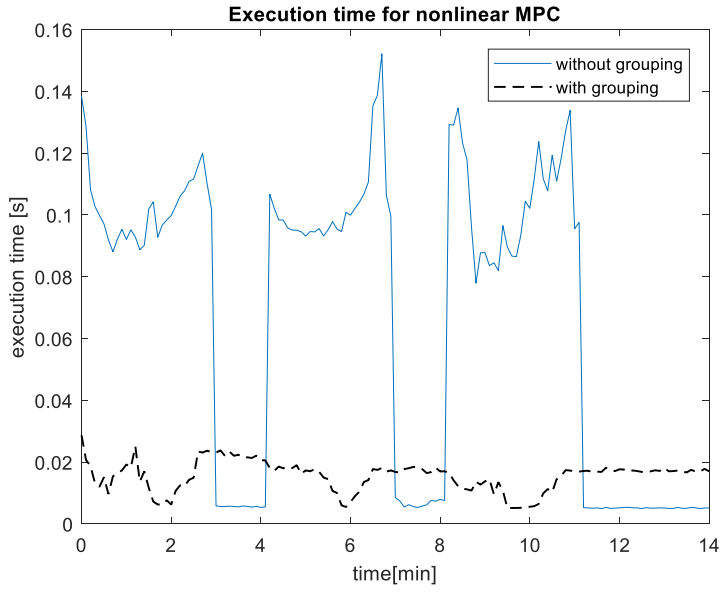


Figure 8.10: Comparison of the execution time with nonlinear MPC with/without control input grouping

Clearly, we can see that with grouping of control inputs, the nonlinear MPC can be executed relatively faster in general. Note that since the solver *fmincon* in MATLAB is optimized for performance, when the process reaches the steady state, the execution time for nonlinear MPC without grouping of control inputs is relatively smaller. However, optimizing the performance of solver may not be a service with other platforms. In general, grouping of control inputs usually results in faster execution time.

But how about the performance of the nonlinear MPC with grouping of control inputs? Will the performance be degraded with grouping of control inputs for this example? Figure 8.11 shows the simulation results.

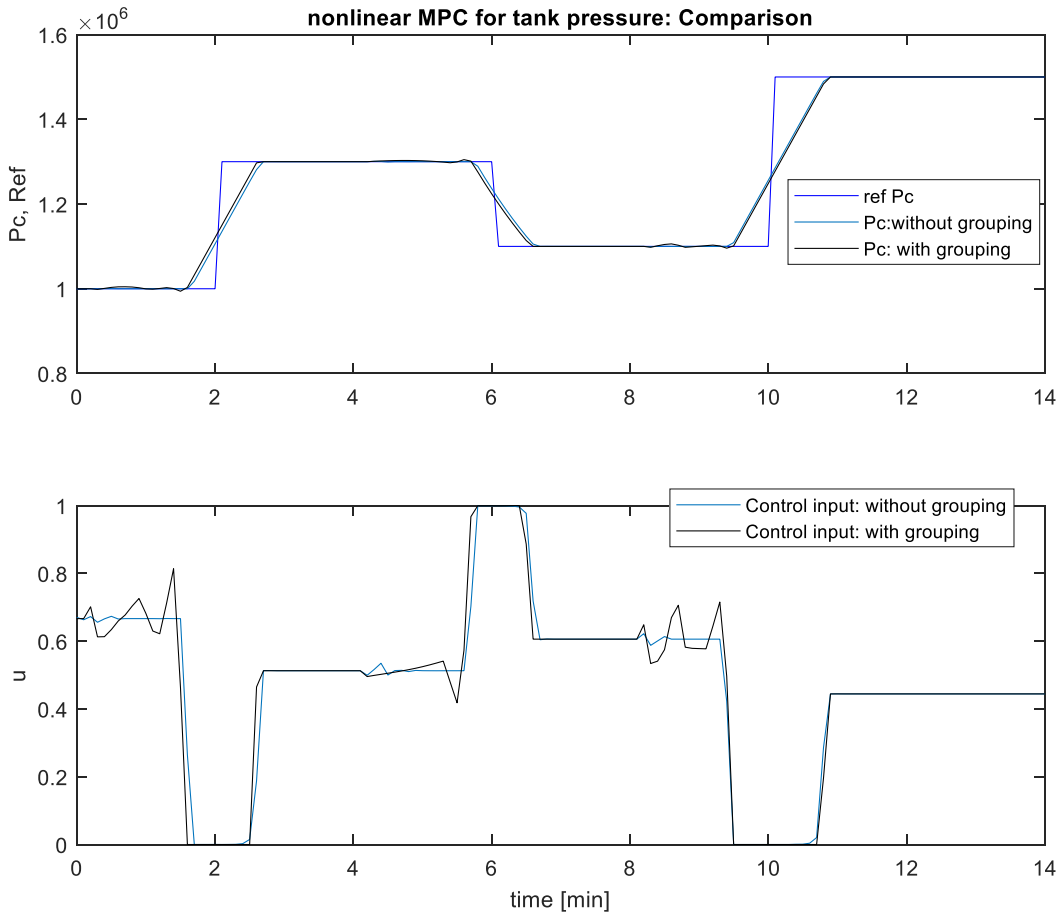


Figure 8.11: Comparison of nonlinear MPC with/without control input grouping

As can be seen from Figure 8.11, the performance of the controller are relatively similar. So we can say that grouping of control inputs does not severely degrade the performance.

Multi-objective and Pareto Optimization: Applications to MPC

Lecture supplement for

Model Predictive Control

University of South-Eastern Norway,

Porsgrunn, Norway.

Oct. 2018, Roshan Sharma

Lecture Content

- **Introduction**
  - Single objective optimization
    - Example of climbing a hill
  - Multi-objective optimization (MOO)
    - Example of buying a car
    - Dominance
    - Pareto optimal solutions, Pareto front
  - Different methods to solve MOO
    - Weighted sum
    - ε - Constraint method
    - Goal Attainment method
    - Hierarchical method
  - Applications
  - Utopia tracking MPC

Introduction: Single objective

Objective:

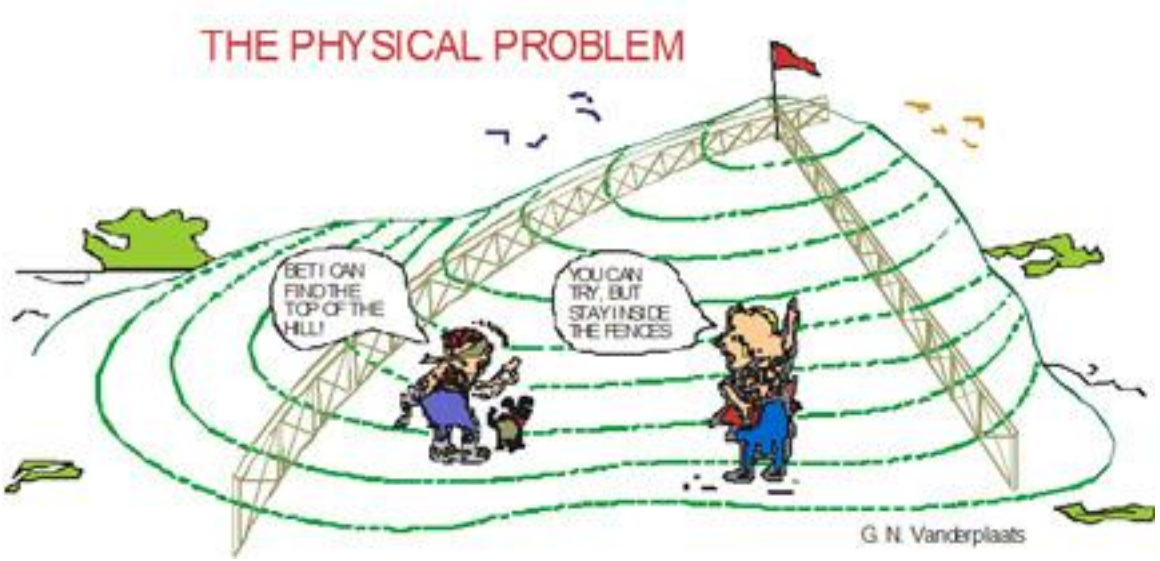
Min/Max $f(x)$
x

Constraints

$g_i(x) \leq 0, \quad i = 1,2,\ldots,m$

$h_j(x) = 0, \quad j = 1,2,\ldots,r$

$x = [x_1, x_2,\ldots,x_n]$



Courtesy: VR&D
www.vrand.com

- The objective is to reach to the top of the hill blind folded. (only a single objective)
- The person should walk up the hill by staying inside the fences (constraints)

Introduction: Single Objective

Objective:

$\min_{x} f(x)$

Constraints

$g_i(x) \leq 0, \quad i = 1,2,\ldots,m$

$h_j(x) = 0, \quad j = 1,2,\ldots,r$

$x = [x_1, x_2,\ldots,x_n]$

Courtesy: VR&D
www.vrand.com

- One possible path that the person can undertake to reach to the top of the hill.
- The single objective is fulfilled and the constraints are also satisfied or obeyed.

Introduction: Multi-objective

Objective:

$\underset{x}{\text{Min/Max}}$ $\quad \mathbf{J}(x) = [F_1(x), F_2(x),\ldots,F_k(x)]$

Constraints

$g_i(x) \le 0, \quad i = 1,2,\ldots,m$

$h_j(x) = 0, \quad j = 1,2,\ldots,r$

$x = [x_1, x_2,\ldots,x_n]$

$F_i(x) = [F_1(x), F_2(x),\ldots,F_k(x)], \quad i=1,2,\ldots,k$

Examples of objectives:

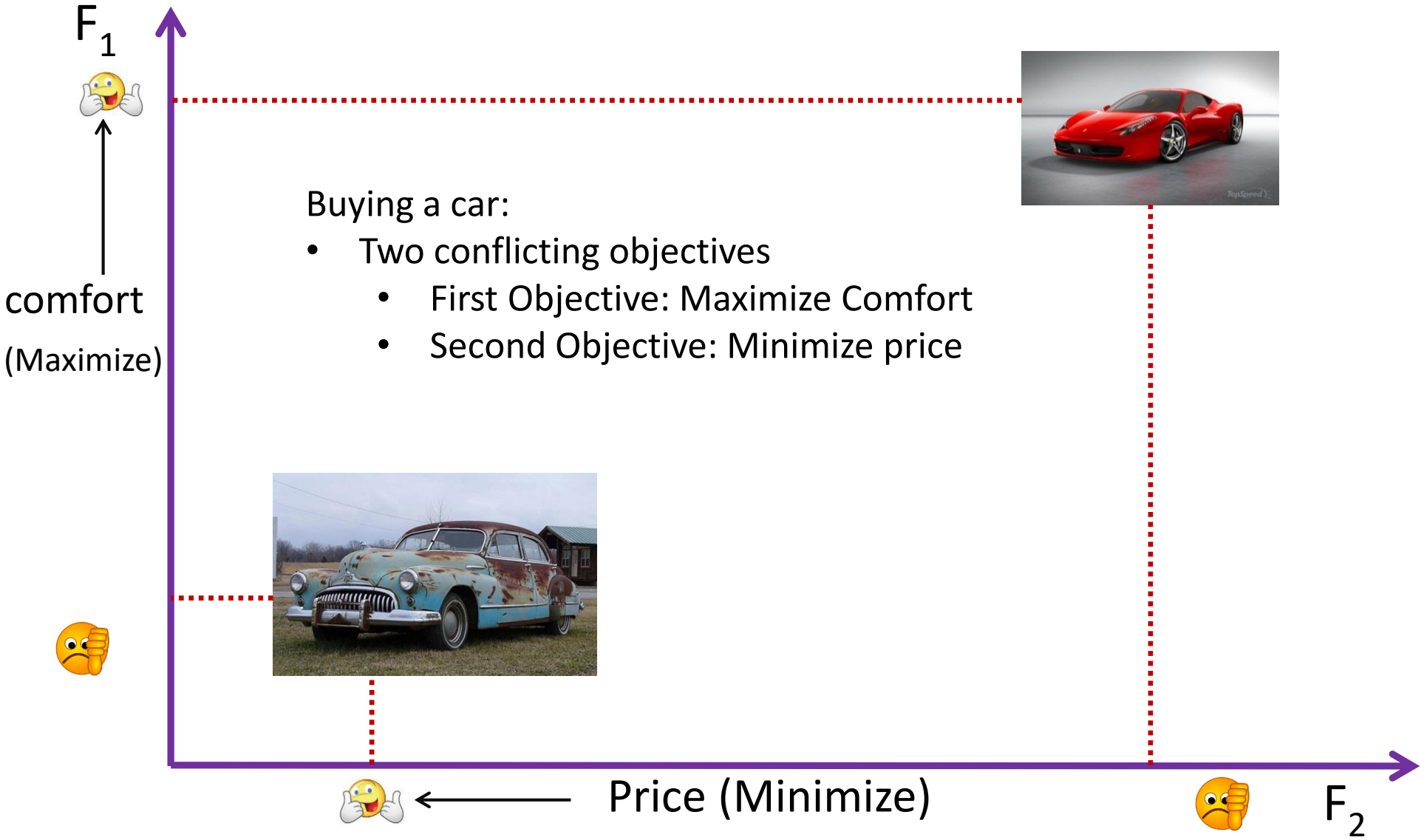
Revenue	Production
Cost	Safety
Energy	Environmental
Time	Range
efficiency	Speed

etc….

- More than one objectives $F_1(x), F_2(x),\ldots,F_k(x)$: k number of objectives
- All the objectives of the given problem should be addressed at once

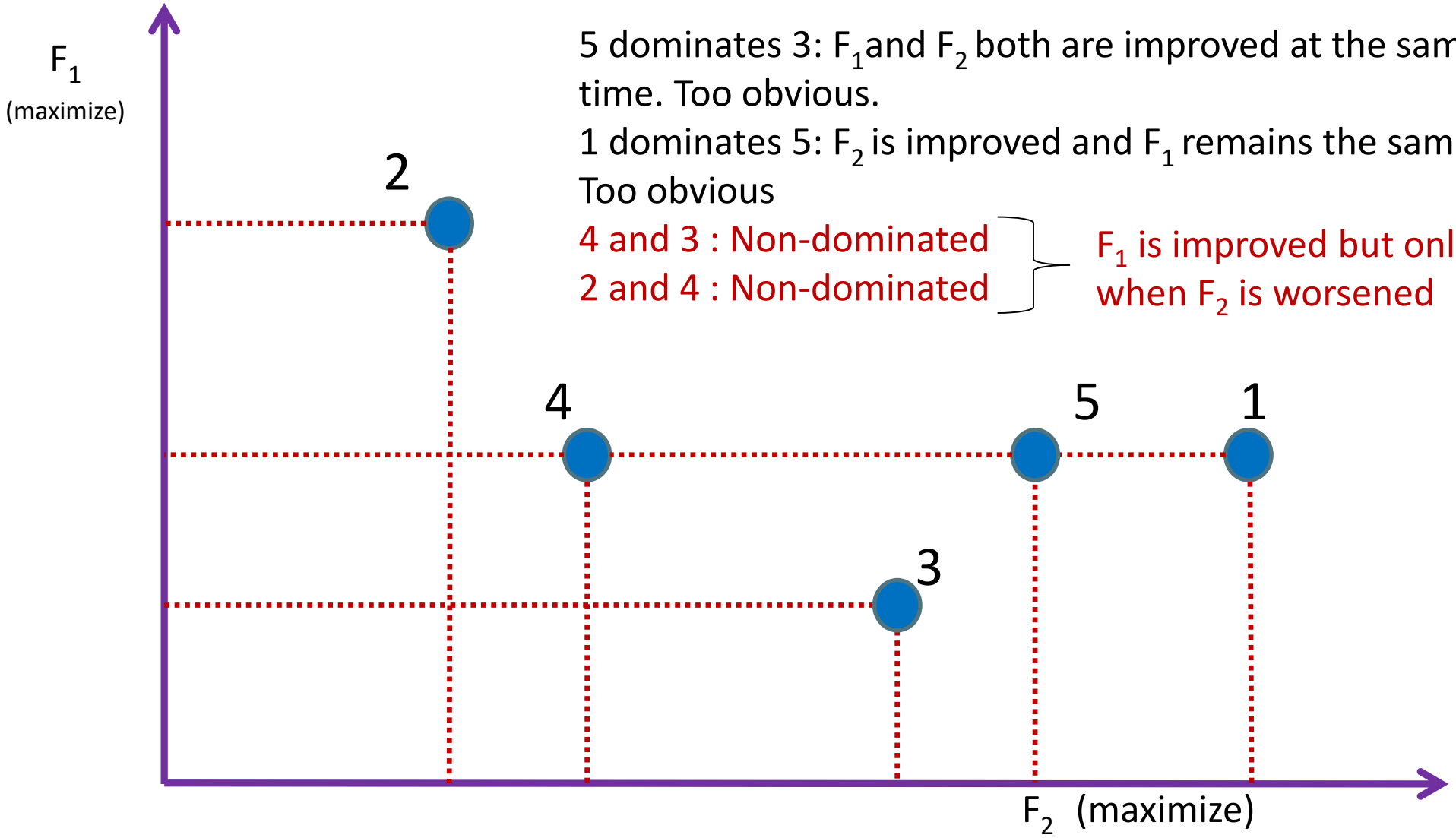
Introduction: Multi-objective, example

- Often the objective functions are conflicting to each other



Introduction: Dominance

- No solution is optimal for all objectives simultaneously.
- For compromised solution, non-dominating points are what we are looking after.



5 dominates 3: F_1 and F_2 both are improved at the same time. Too obvious.

1 dominates 5: F_2 is improved and F_1 remains the same. Too obvious

4 and 3 : Non-dominated

2 and 4 : Non-dominated

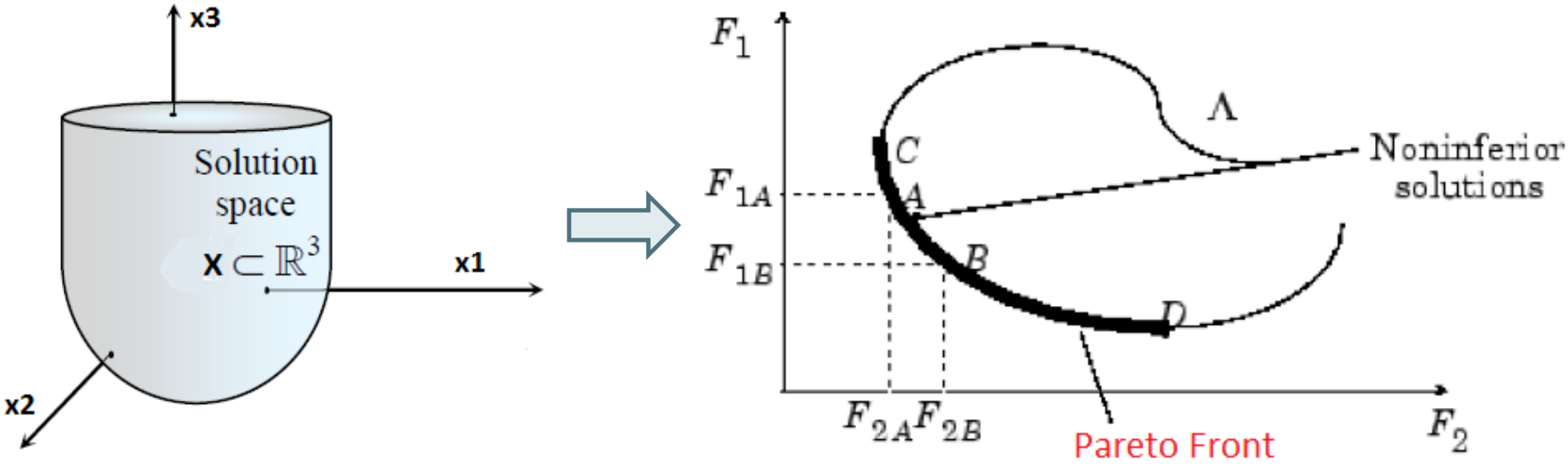
F_1 is improved but only when F_2 is worsened

Introduction: Pareto optimal solutions

Non- dominating x* are **Pareto optimal** solutions

In words: Improvement in one objective function is possible only by worsening of at least one other objective function.

For e.g: Points A, B, C, D are all solutions (non-dominating points). At point D (which is a solution), F_2 is improved but at the same time F_1 is worsened, when compared to point B.

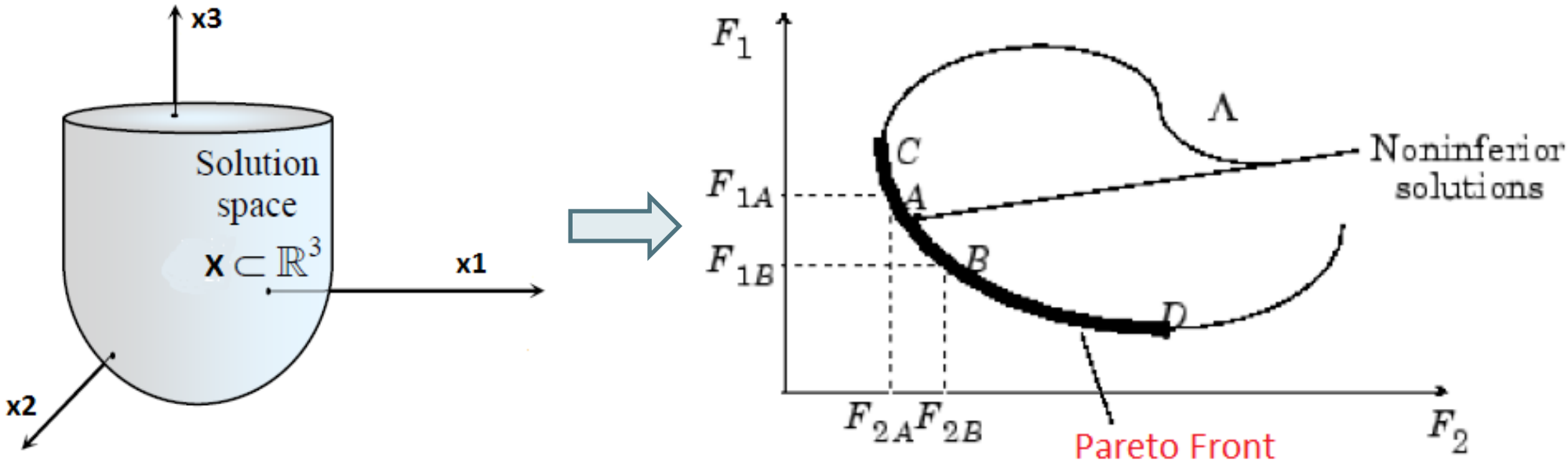


ref: Gambier (2008)

- In a given MOO, there can be more than one Pareto optimal solutions.
- Collection of Pareto Optimal solution is known as Pareto Front

Introduction: Pareto optimal solutions

- All points in Pareto front are equally acceptable solutions.
- Here, solutions A, B, C and D are all valid solutions.
- Decision maker decides the trade-off
  - For compromised solution
- Which one to choose among them? It depends on which objective you want to improve more by sacrificing the other objective.

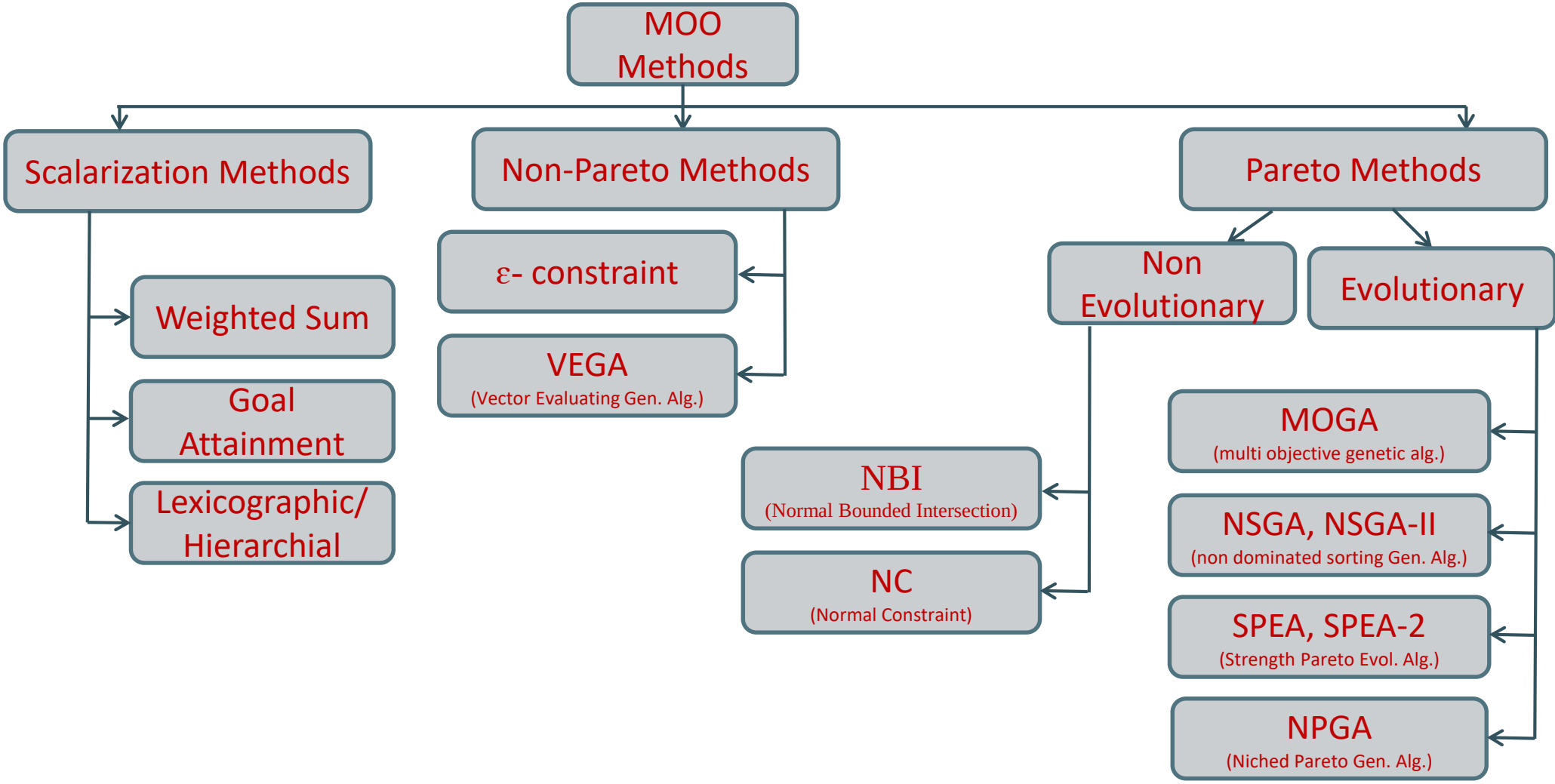


ref: Gambier (2008)

Introduction: Solution Methods, How to solve MOO?

- Methods to solve MOO.

MOO = Multi Objective Optimization

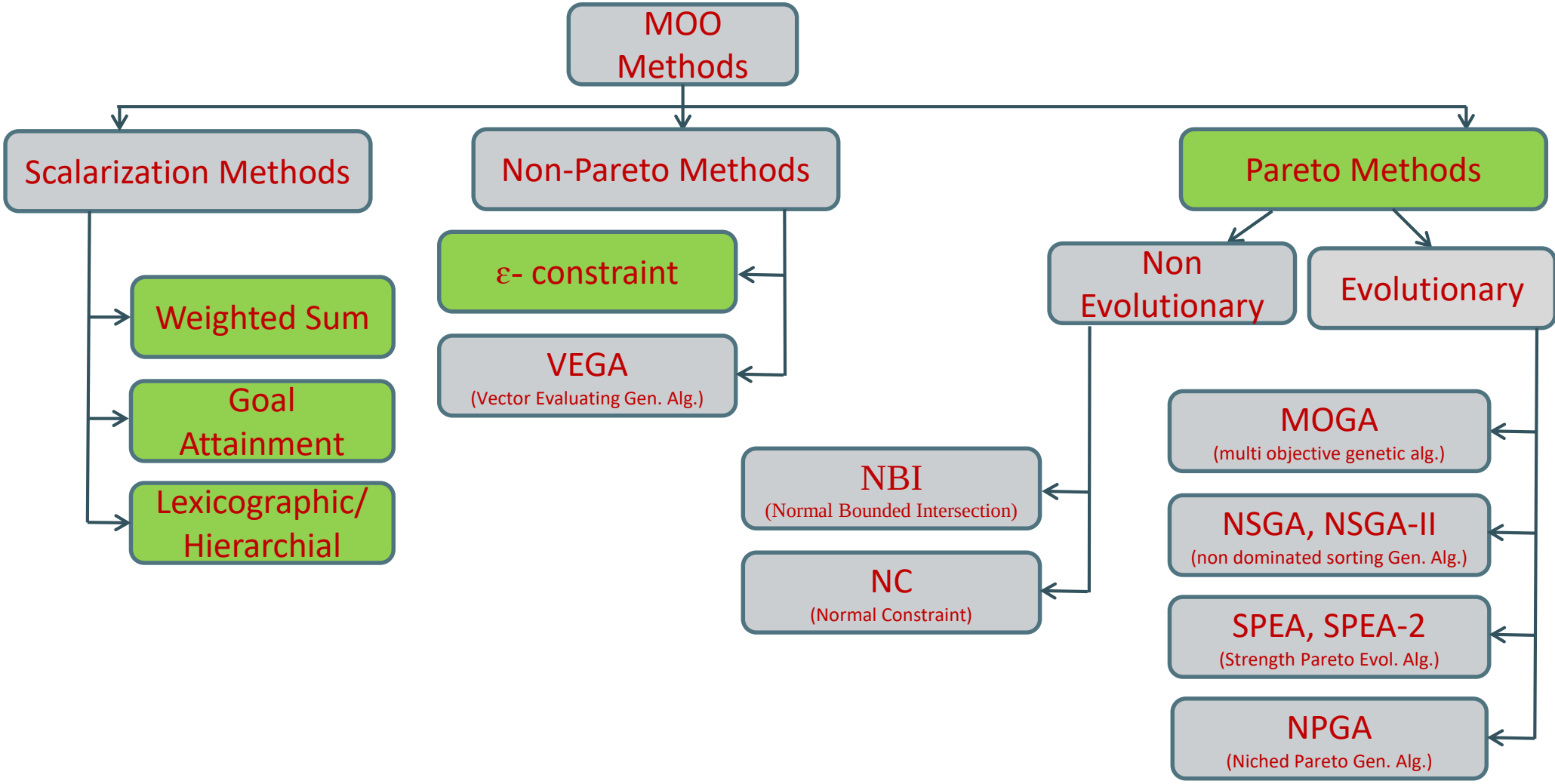


ref: Anderson (2000), Marler and Arora (2004) , Gambier (2008)

Introduction: Solution Methods, How to solve MOO?

- Methods to solve MOO.
- Only Green colored will be explained in brief.

MOO = Multi Objective Optimization



ref: Anderson (2000), Marler and Arora (2004) , Gambier (2008)

Solution Methods: Weighted Sum

Min $\quad J(x) = [F_1(x), F_2(x),\ldots,F_k(x)]$

s.t $\quad$ *Constraints*

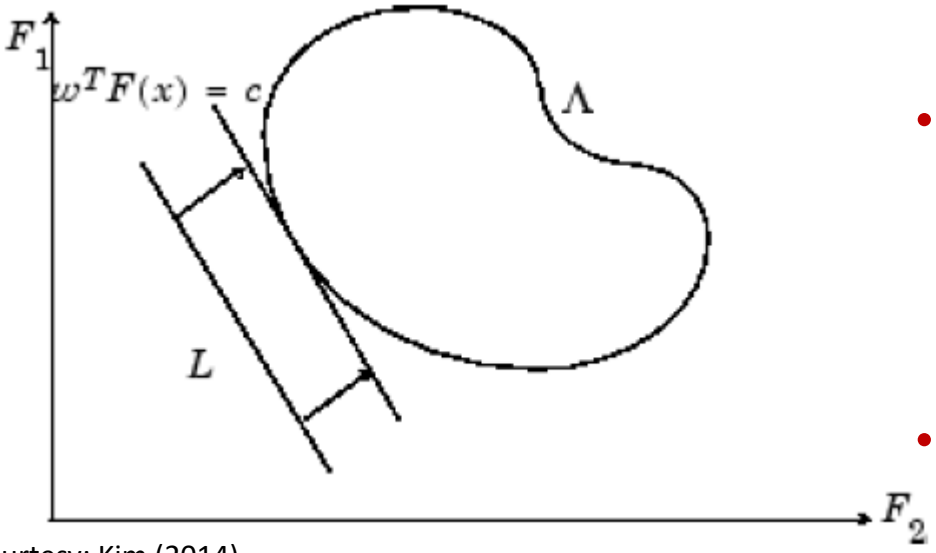
- Weighted Sum

Min $$J(x) = \sum_{i=1}^{k} w_i F_i(x)$$

s.t $\quad$ *Constraints*

- All the *k* objectives are added together
- Each objective is assigned a weighing factor $w_i > 0$

w_i = Weights assigned to the i^{th} objective



- The slope of the line *L* depends on the weighing factors.
- At each iteration while solving *J(x),* (an NLP with multi-objective), the value of *c* changes i.e. *J(x)* becomes better and better with iteration and the line slides towards the feasible region.
- The point where the line *L* touches the feasible region is the solution.

Courtesy: Kim (2014)

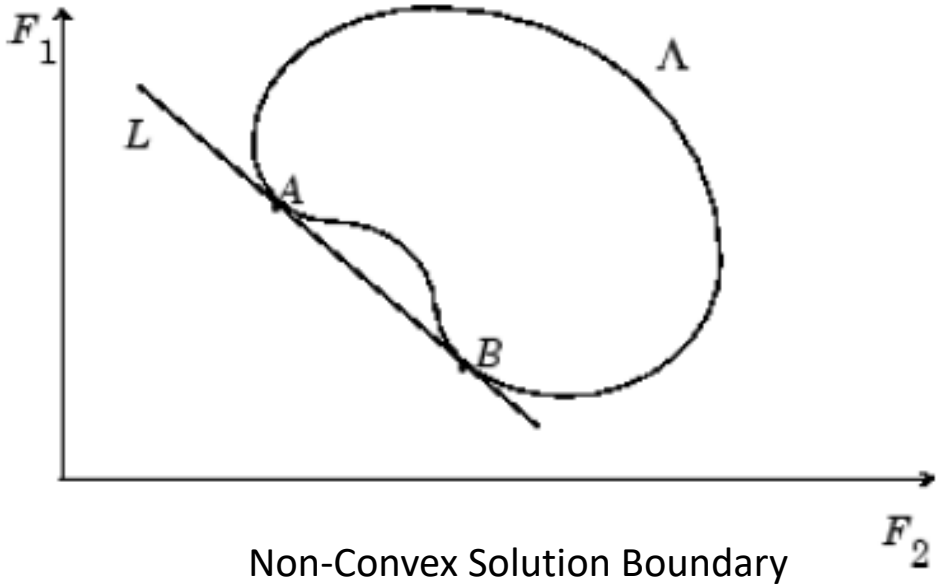
Ref: Marler and Arora (2004)

Solution Methods: Weighted Sum

- Weighted Sum (disadvantages)

  - Difficult to assign weights (based on experience or trial and error).

  - For non-convex problems, certain non-dominated solutions are not accessible.
    - Any point lying between A and B is not accessible by this method.



Non-Convex Solution Boundary

Ref: Marler and Arora (2004)

Solution Methods, Ɛ - constraint method

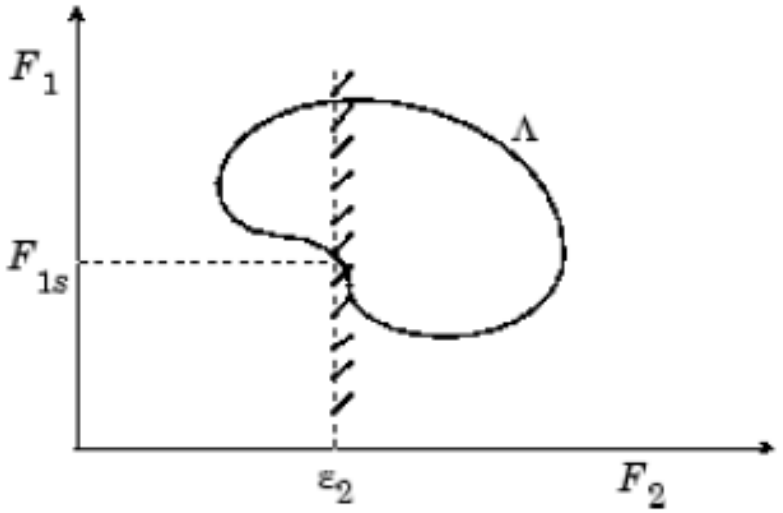
- ε - constraint method (can handle the disadvantage of weighted sum method)
  - Choose one of the objective as primary objective ($F_p(x)$)
  - Minimize the primary objective
  - Express other objectives as inequality constraints using ε such that the secondary objectives should be at least equal to or less than user defined ε.

$$\underset{x}{Min} \quad F_p(x)$$

$$s.t$$

$$F_i(x) \leq \varepsilon_i \qquad i = 1,2,\dots,k \qquad i \neq p$$

plus the constraints of the original optimization problem



For properly selected ε , it can also find non-convex solution.

But….
May have difficulty in the selection of suitable ε.

Ref: Marler and Arora (2004)

Solution Methods: Goal Attainment Method

- Goal Attainment Method (can be thought as relaxation of ε - constraint method)

-- Express a set of goals (F_i^*) for each objectives.
-- In MOO, it is difficult to achieve all goals simultaneously.
-- Use slack variables (λ) to violate the unachieved goals but minimize the violation.

$$\begin{aligned} & \underset{x,\lambda}{Min} \quad \lambda \\ s.t \quad & \\ & F_i(x) - w_i\lambda \leq F_i^* \qquad i = 1,2,\dots k \end{aligned}$$

F_i^* = goal or aspiration level for the i^{th} objective $F_i(x)$

λ = slack variable

- The idea is to achieve the value of each objective function $F_i(x)$ at least equal or less than the defined goal for the objective.
- The goals are under or over achieved by making use of weights.

w_i = weights to control the degree of under or over achievement. It allows user to express a measure of relative tradeoffs between the objectives.

$w_i\lambda$ introduces an element of slackness into the problem. If w_i is zero, it simply means that the goals has to be rigidly met.

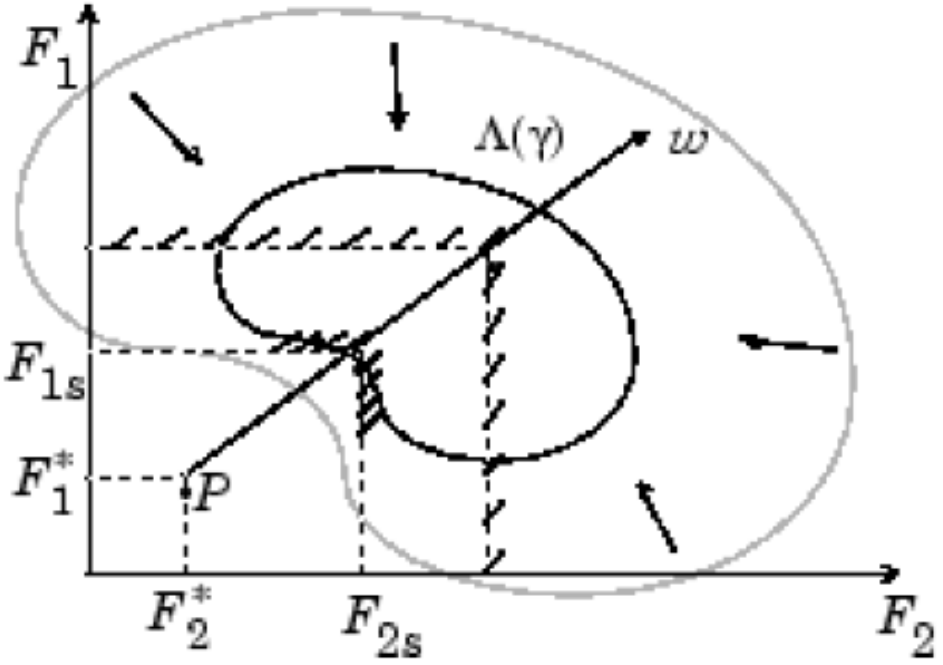
Ref: Wang (2003)

Solution Methods: Goal Attainment Method

- The choice of the goals F_i^* defines the goal point *P.*
- Weighting vector w_i for slack variables defines direction of search

- $slack\ variable\ \lambda$ is varied during optimization
  - Size of feasible region changes

- Converge to a unique solution point
  - (F_{1s} and F_{2s})



Courtesy: Kim (2014)

Solution Methods: Hierarchical method (Lexicographic method)

- Objectives functions are arranged in the order of importance with highest prioritized objective at the top.
- One after the other, each objective function is minimized starting with the most important one and proceeding according to the order of importance.
- At any given iteration, the prioritized objectives (appearing before the given iteration) form constraints inorder not to sacrifice its performance.

$$\underset{x}{Min} \quad F_i(x)$$

$$s.t$$

$$F_j(x) \leq F_j(x_j^*), \qquad j = 1,2,\dots,i-1, \qquad i > 1$$

- $F_j(x_j^*)$ is the optimum of the j^{th} objective function, found in the j^{th} iteration.
  - Optimal objective values of the higher prioritized objectives impose constraints.

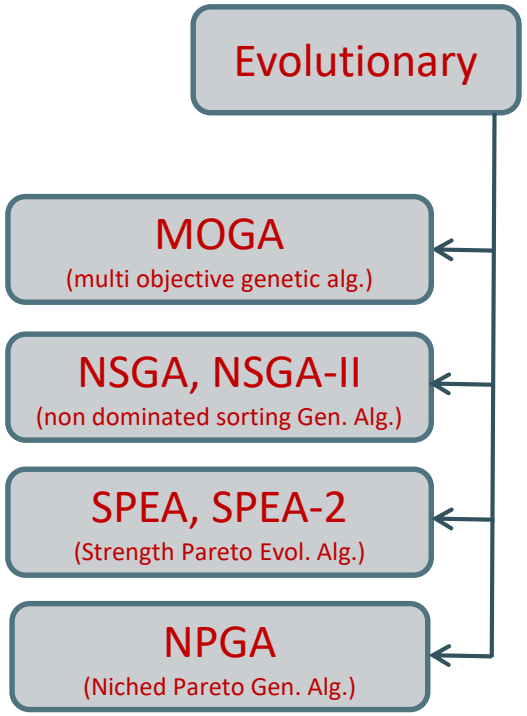
- The constraints $F_j(x) \leq F_j(x_j^*)$ ensures that the objective function $F_i(x)$ at current i^{th} iteration is minimized such that the higher prioritized objectives (from *j=1,2…,i-1*) are either equal to or less than their optimum values in the j^{th} iteration.

- We obtain one optimum for a given lexicographic order.
- Normally objectives with lower priorities will not be properly satisfied (disadvantage).

Ref: Gambier (2008) , Marler and Arora (2004), Wang (2003)

Introduction: Solution Methods, Brief

- Evolutionary Algorithms



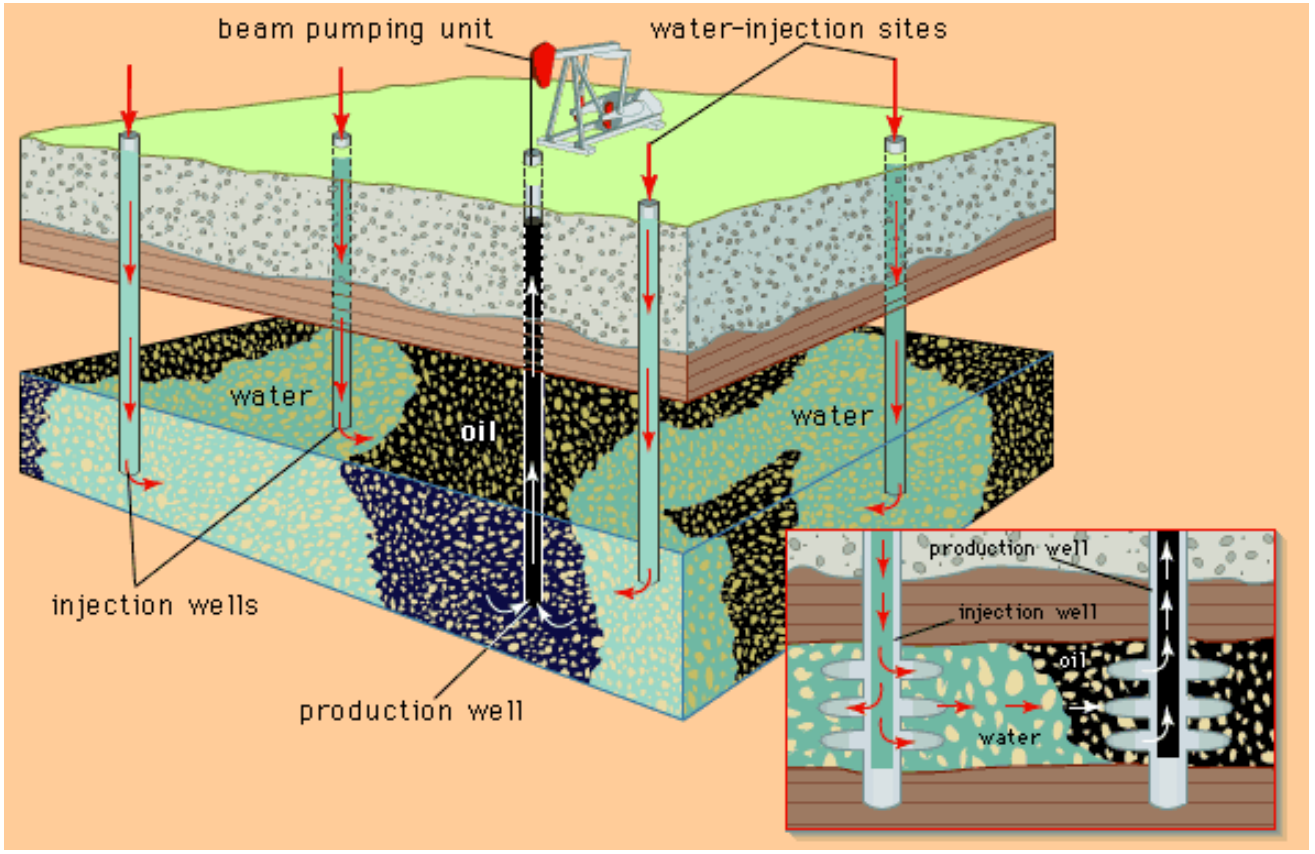
- Natural selection
  - Survival of the fittest

Applications: Multi-objective and Pareto

Applications to multi-objective optimization

Application: Reservoir planning

- Production optimization
  - Water flooding oil recovery



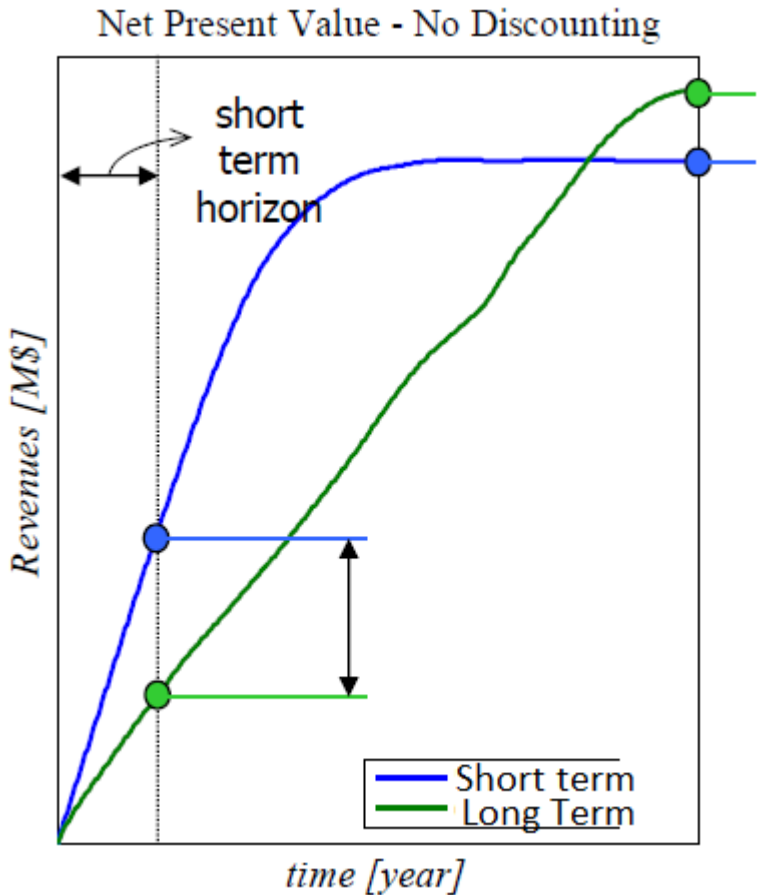
Courtesy: Adam (2009)
www.adamrlee.org

Application: Production planning

- Two objectives:
  - Maximize long term reservoir performance (life cycle)
  - Maximize short term reservoir performance (weeks)

- Net Present Value (NPV)

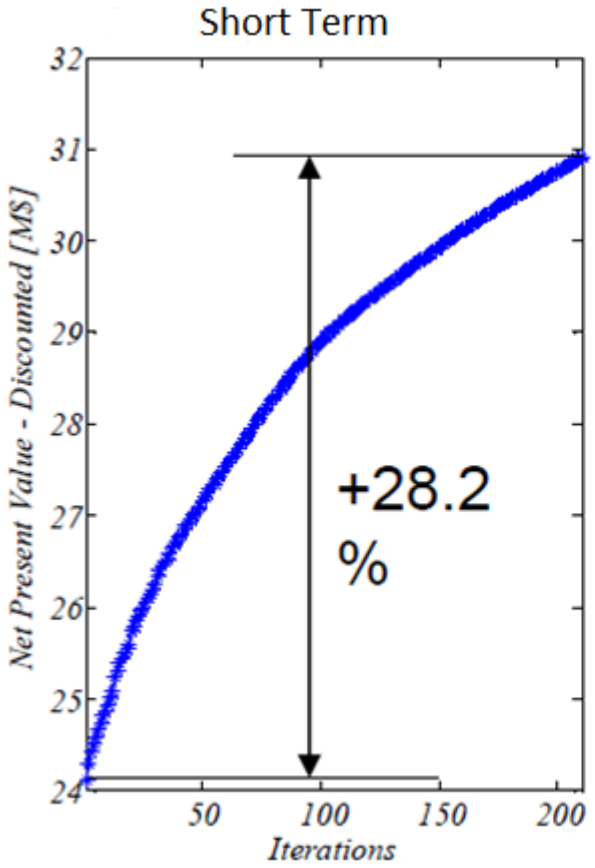
$$J = \sum_{k=1}^{K} \frac{\{[(q_{o,k}).r_o - (q_{wp,k}).r_{wp}] - [q_{wi,k}).r_{wi}]\}\Delta t_k}{(1+b)^{t_k/\tau_t}}$$

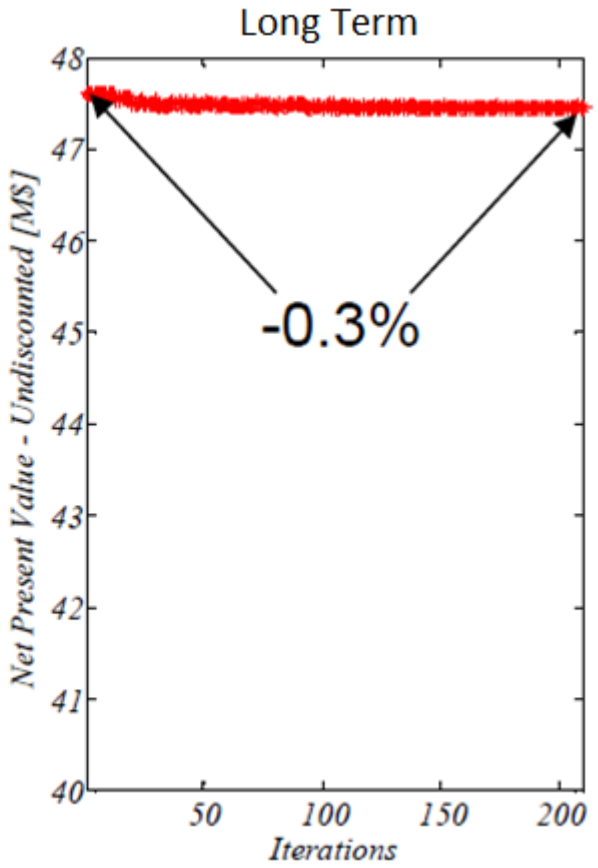


Ref: Essen et al. (2009), Fronseca et al. (2012)

Application: Production planning

- For a given well configuration and reservoir model:
  - Solved using Hierarchical method



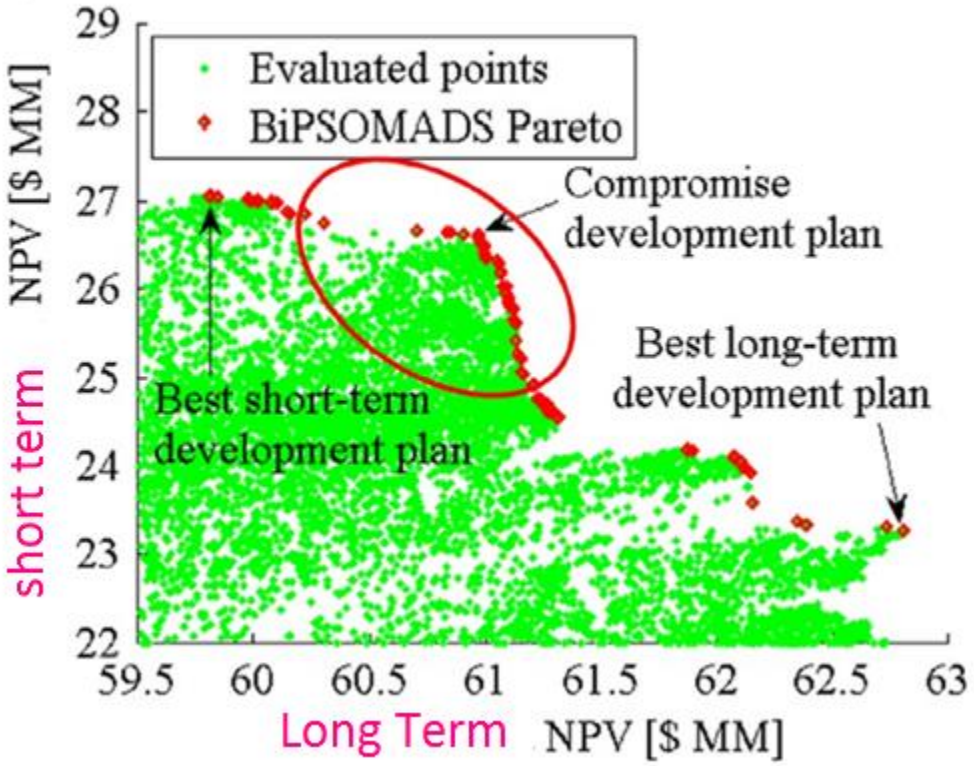


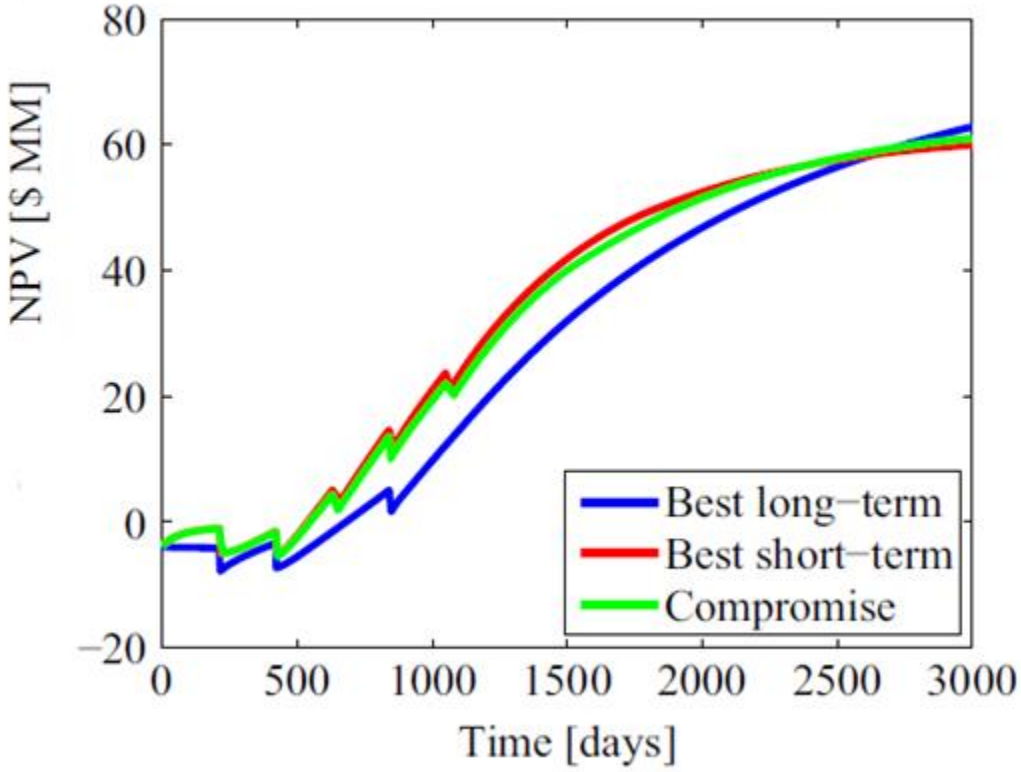
Best Compromise

Ref: Hof (2013)

Application: Reservoir planning

- More detailed study: Field Development Plan
  - PSO-MADS method (Evolutionary algorithm)





Ref: Isebor and Durlofsky (2014)

PSO-MADS = Particle Swarm Optimization- Mesh Adaptive Direct Search

Applications: Multi-objective and Pareto

Applications to multi-objective MPC

Application: MPC

Olive Oil extraction Mill

- Multi-Objective MPC formulated in three different ways

  (i) Weighted MPC (weighted sum method)

  (ii) Prioritized MPC (Hierarchical method)

  (iii) Structured MPC

Application: MPC, Olive oil extraction

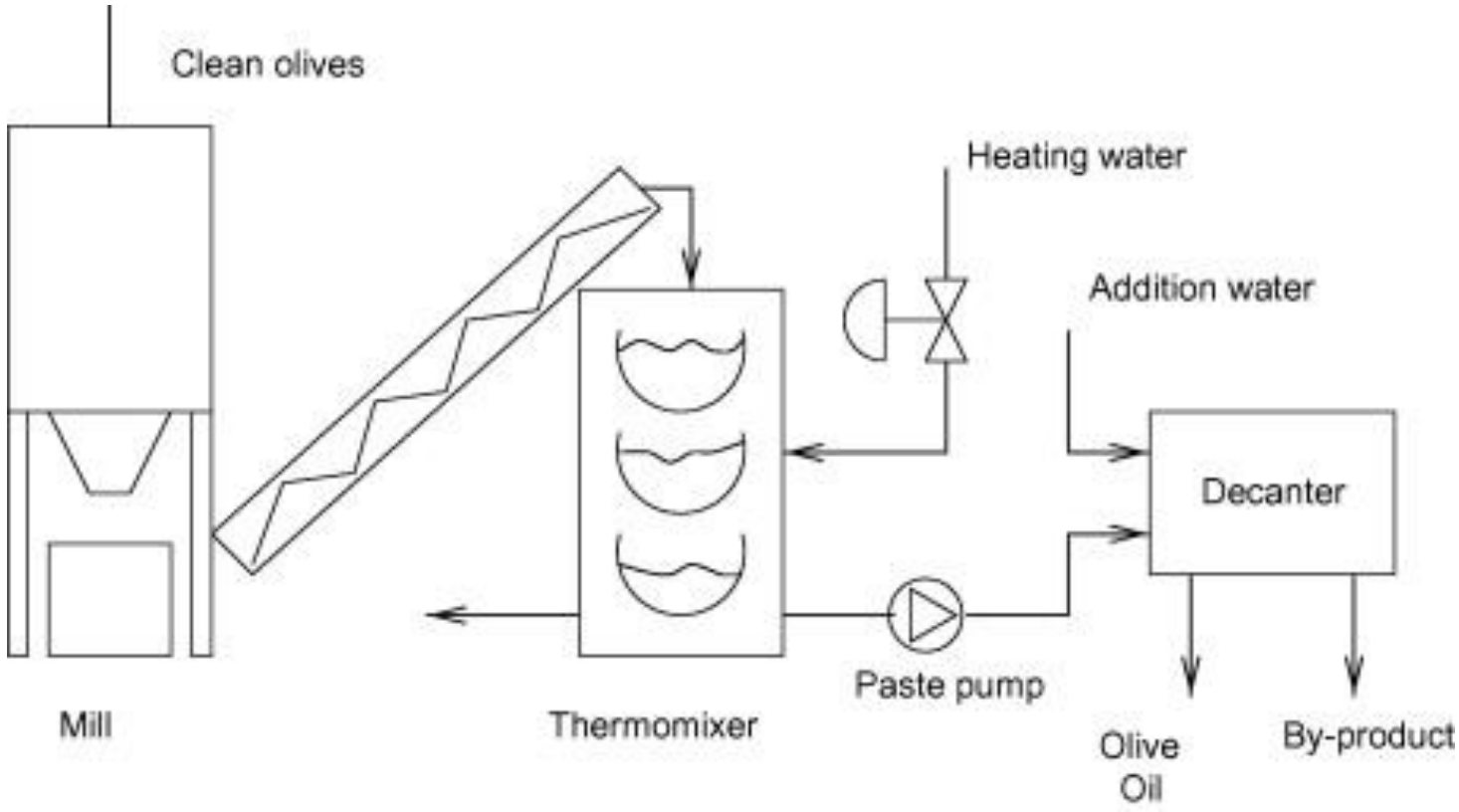
Multi-Objectives:

i) Maximize extracted oil $O_1 = |y-y_{op}| \leq \varepsilon_y$

ii) Keep Thermomixer temperature (Quality): $O_2 = |u_1-u_{1op}| \leq \varepsilon_{u1}$

iii) Reduce water flow: $O_3 = |u_3-u_{3op}| \leq \varepsilon_{u3}$

iv) Keep paste flow: $O_4 = |u_2-u_{2op}| \leq \varepsilon_{u2}$



Ref: Reyes et al. (2002)

Application: MPC, Olive oil extraction

Multi-Objective MPC:

i) Maximize extracted oil $O_1 = |y-y_{op}| \leq \varepsilon_y$

ii) Keep Thermomixer temperature (Quality): $O_2 = |u_1-u_{1op}| \leq \varepsilon_{u1}$

iii) Reduce water flow: $O_3 = |u_3-u_{3op}| \leq \varepsilon_{u3}$

iv) Keep paste flow: $O_4 = |u_2-u_{2op}| \leq \varepsilon_{u2}$

Weighted MPC

$$\min \quad J = \sum_{i=1}^{m} \beta_i O_{i,k}$$

Subject to

$$R_i u \leq a_i \qquad i = 1,2,\ldots,m$$

- Difficult to assign weights.
- Control objectives may be qualitative.

e.g.

$$J = \delta\left(y - y_{op}\right)^2 + \lambda(\Delta u)^2 + \gamma\left(u - u_{op}\right)^2$$

Ref: Reyes et al. (2002)

Application: MPC, Olive oil extraction

Prioritized MPC

i) Keep Thermomixer temperature: $O_1 = |u_1 - u_{1op}| \leq \varepsilon_{u1}$

ii) Maximize extracted oil: $O_2 = |y - y_{op}| \leq \varepsilon_y$

iii) Keep paste flow: $O_3 = |u_2 - u_{2op}| \leq \varepsilon_{u2}$

iv) Reduce water flow: $O_4 = |u_3 - u_{3op}| \leq \varepsilon_{u3}$

Reduced priority of the objective functions

Objectives are prioritized

- Prioritization problem is formed
  - Combining propositional logic using integer variables
    details at Tayler and Morari (1998)

- Due to integer: Mixed Integer Problem is formed.

Ref: Reyes et al. (2002)

Application: MPC, Olive oil extraction

Prioritized MPC

- For each time in the prediction horizon
  - Prioritization problem (Mixed Integer) is solved.
  - Optimal values from prioritization problem
    - Used as desirable setpoints to a MPC controller.

Ref: Reyes et al. (2002), Tyler and Morari (1998)

Application: MPC, Olive oil extraction

Structured MPC:

- Decision list based in a set of if-then statements

$$J_1 = \delta\left(y - y_{op}\right)^2 + \lambda(\Delta u)^2 + \gamma_1\left(u - u_{op}\right)^2$$

$$J_2 = \delta\left(y - y_{op}\right)^2 + \lambda(\Delta u)^2$$

$$J_3 = \delta\left(y - y_{op}\right)^2 + \lambda(\Delta u)^2 + \gamma_2\left(u - u_{op}\right)^2$$

$$J_4 = \delta\left(y - y_{op}\right)^2 + \lambda(\Delta u)^2 + \gamma_3\left(u - u_{op}\right)^2$$

- γ changes according to desirable level of priorities
  - Change of function: logic dependent (e.g. comparison of values)

Select current objective function → supply to the MPC

Ref: Reyes et al. (2002)

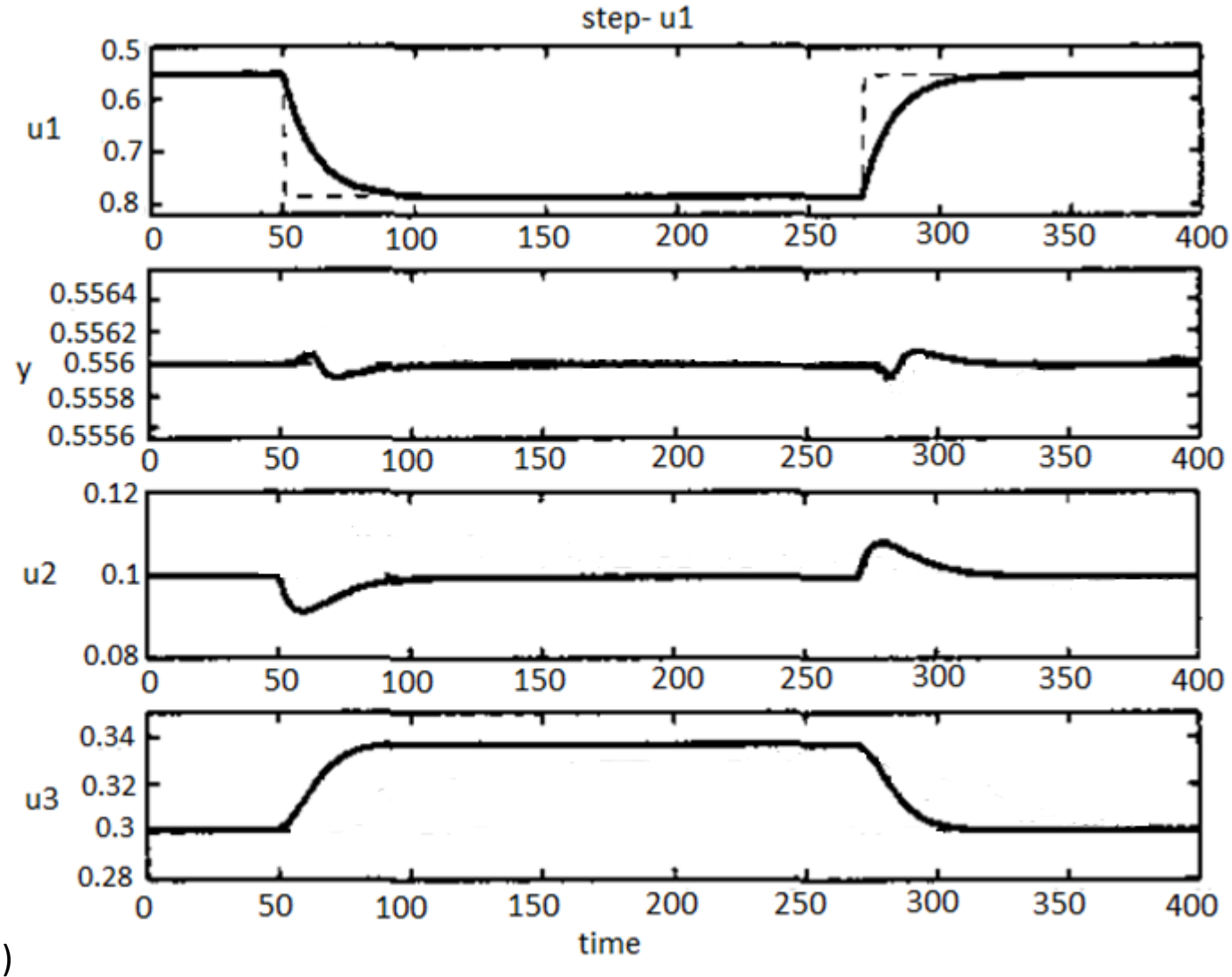
Application: MPC, Olive oil extraction

Comparison:

Highest Priority

Prioritized MPC

Lowest Priority



Ref: Reyes et al. (2002)

Application: MPC, Olive oil extraction

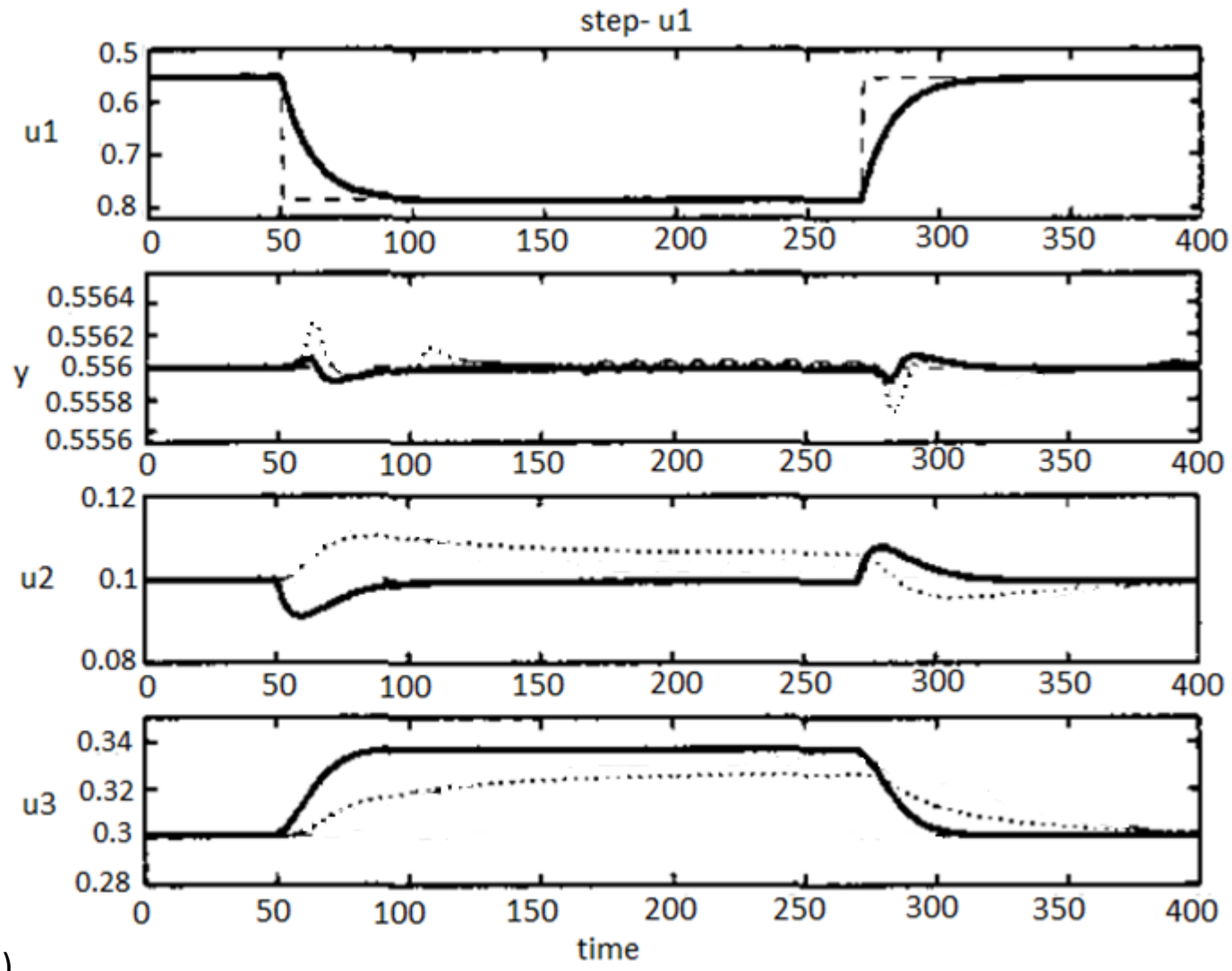
Comparison:

Highest Priority

Prioritized MPC ——

Weighted MPC .....

Lowest Priority



Ref: Reyes et al. (2002)

Application: MPC, Olive oil extraction

Comparison:

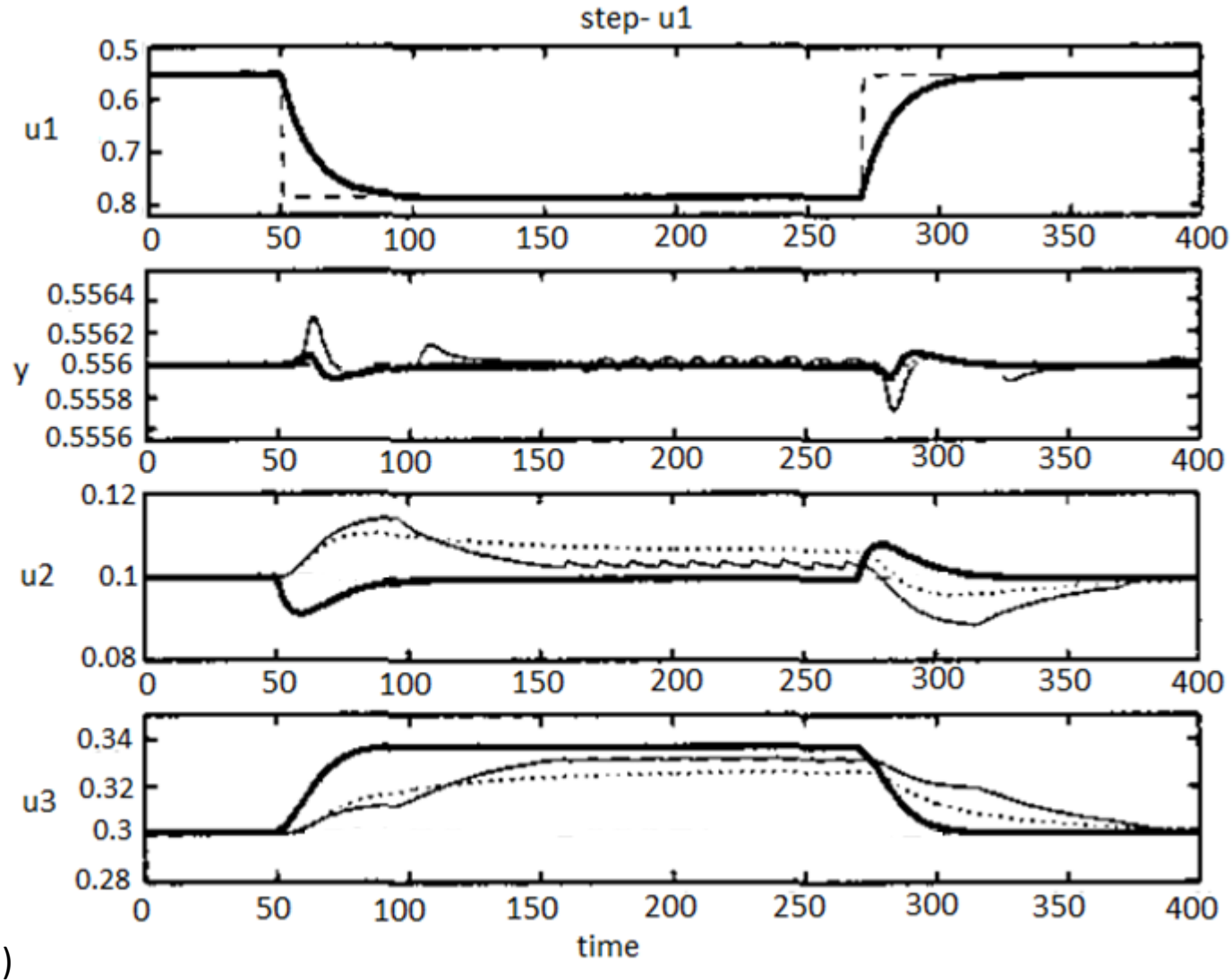
Highest Priority

Prioritized MPC

Structured MPC

Weighted MPC

Lowest Priority



Ref: Reyes et al. (2002)

Utopia Tracking MPC

Application: Utopia Tracking MPC

- Multi-Objective problem

$$\min_{x,u} \quad \boldsymbol{\phi}(x,u) = [\phi_1(x,u), \phi_2(x,u), \dots, \phi_M(x,u)]$$

$$s.t. \quad x^+ = f(x,u) \quad \text{Dynamic process model}$$
$$g(x,u) \leq 0$$
$$x \,\mathcal{E}\, X, u \,\mathcal{E}\, U$$

- Utopia Point (steady state)

Find the minimum of each individual objective separately using steady state optimization.

$$\phi_i^L = \min_{x,u} \quad \phi_i(x,u)$$

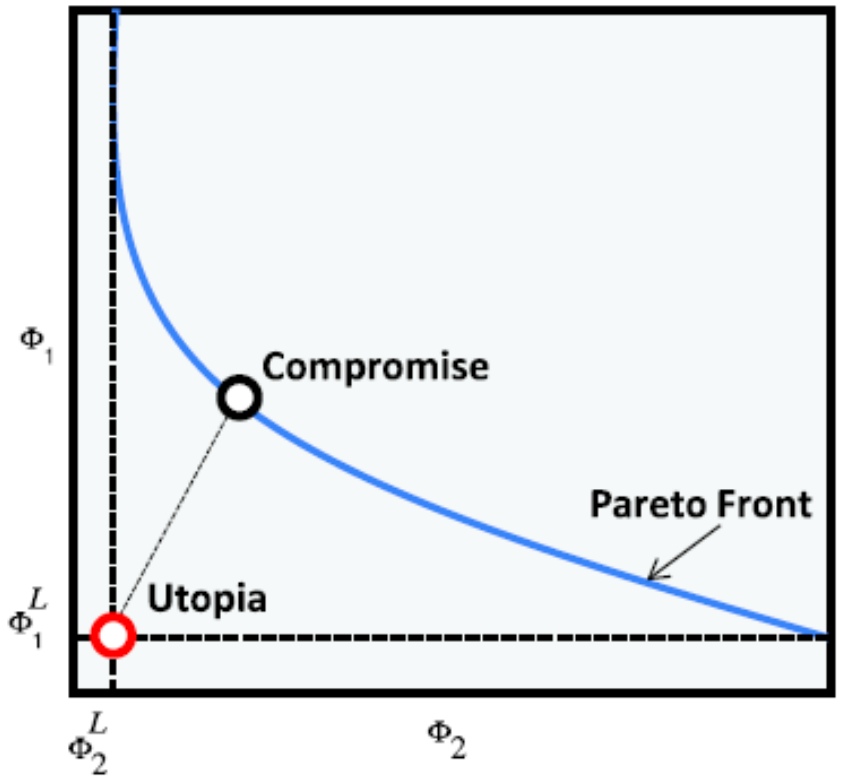
$$s.t. \quad x = f(x,u) \quad \text{Steady state process model}$$
$$g(x,u) \leq 0$$
$$x \,\mathcal{E}\, X, u \,\mathcal{E}\, U$$

$$i = 1,2,\dots,M$$

$$\boldsymbol{\phi}^{L,s} = [\phi_1^L, \phi_2^L, \dots, \phi_M^L]$$

- Utopia point: The point where each objective function $\phi_i(x,u)$ is minimum individually.
- Utopia point is an ideal point. It cannot be reached when you consider all the objectives of the given problem at the same time. This is because objectives functions are conflicting each other.

Refs: Zavala and Tlacuahuac (2012), Tlacuahuac et al. (2011)



Application: Utopia Tracking MPC

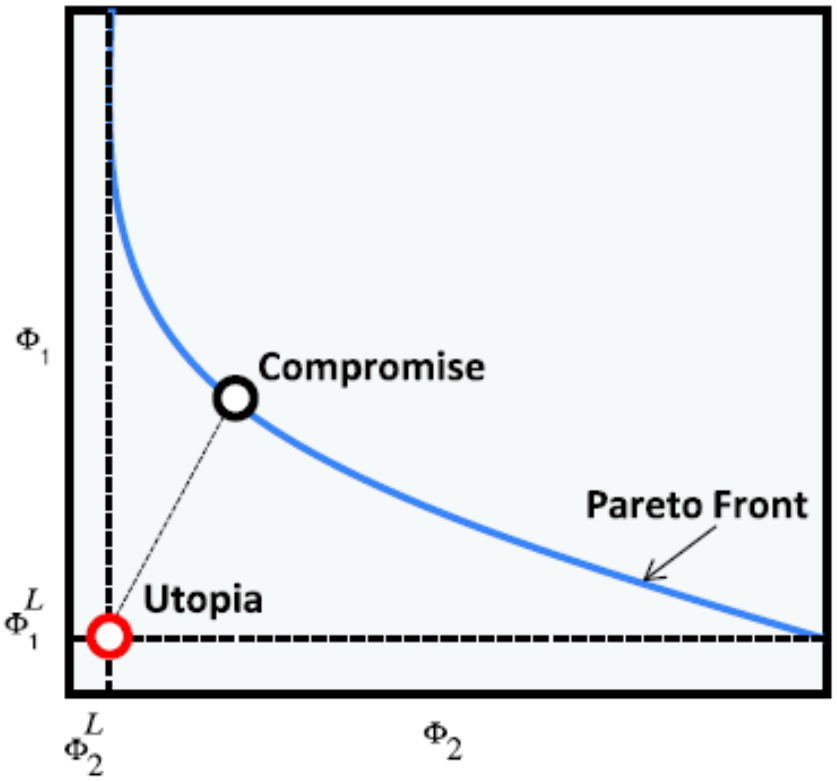
MAIN IDEA:

- If you cannot reach to utopia point, then try to be as close as possible.
- Minimize directly the distance to the steady state utopia point from the pareto front.

$$J_s(x,u) = \sum_{k=0}^{N-1} \left\| \boldsymbol{\phi}(x_k,u_k) - \boldsymbol{\phi}^{L,s} \right\|_p$$

N = prediction horizon

Normally norm 2 is used



- MPC optimal control problem

$$\begin{aligned} \min_{x,u} \quad & J_s(x,u) \\ s.t. \quad & x^+ = f(x,u) \\ & g(x,u) \le 0 \\ & x \in X, u \in U \end{aligned}$$

Refs: Zavala and Tlacuahuac (2012)
Tlacuahuac et al. (2011)

Application: Utopia Tracking MPC

Advantage:

- No need to generate pareto front
  - Trade off: handled dynamically

Some Comments:

For non steady operation (e.g. Cyclic, periodic)

- Sub-optimal

- Average cyclic (dynamic) performance is better
- Steady state Utopia-point impose limitation

Refs: Maree and Imsland (2013)

Application: Dynamic Utopia Tracking MPC

- Utopia Point (Dynamic)

Find the dynamic mean utopia point

$$\phi_{d,i}^{L} = \min_{x,u} \frac{1}{N^L} \sum_{i=0}^{N^L-1} \phi_i(x,u)$$

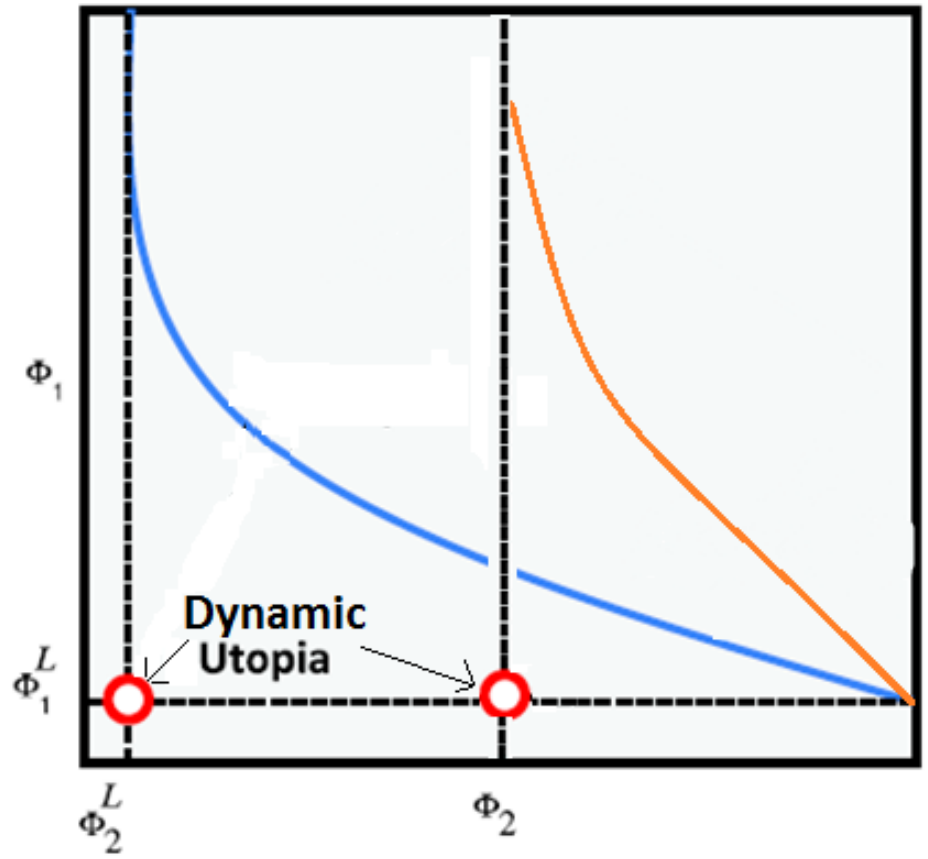
N^L is optimal period

$$s.t.$$

$$x^+ = f(x,u)$$
$$g(x,u) \leq 0$$
$$x \in X, u \in U$$

$$i = 1,2,\dots,M$$

$$\boldsymbol{\phi_d^L} = [\phi_{d,1}^L, \phi_{d,2}^L, \dots, \phi_{d,M}^L]$$



Refs: Maree and Imsland (2013, 2014)

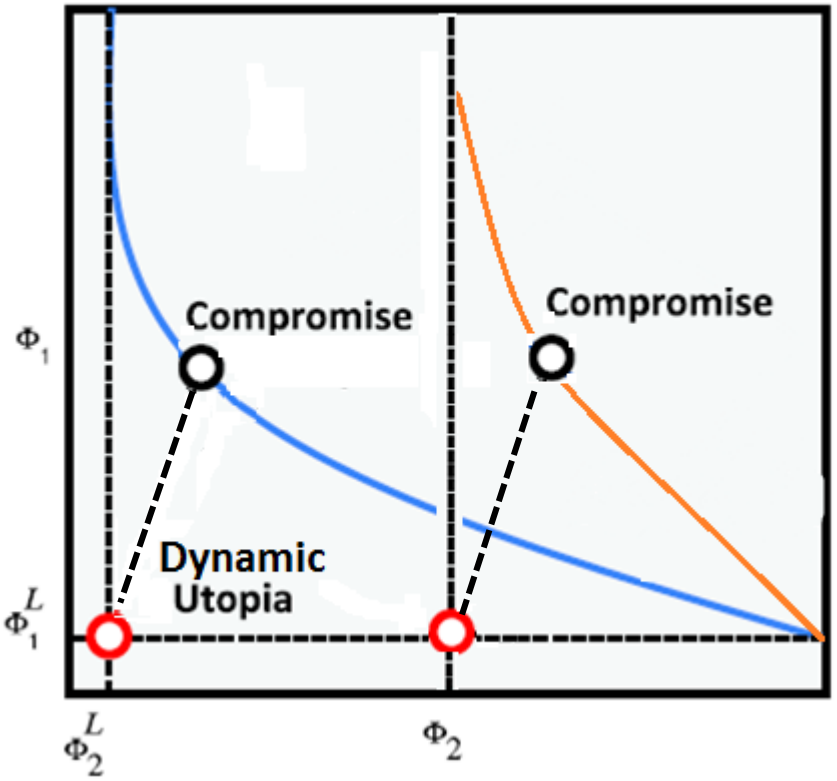
Application: Dynamic Utopia Tracking MPC

- Minimize directly the distance to the dynamic utopia point

$$J_d(x,u) = \left\| \frac{1}{N} \sum_{k=0}^{N} \boldsymbol{\phi}(x_k, u_k) - \boldsymbol{\phi}_{\boldsymbol{d}}^{\boldsymbol{L}} \right\|_p$$

N = prediction horizon

Normally norm 2 is used



- MPC optimal control problem

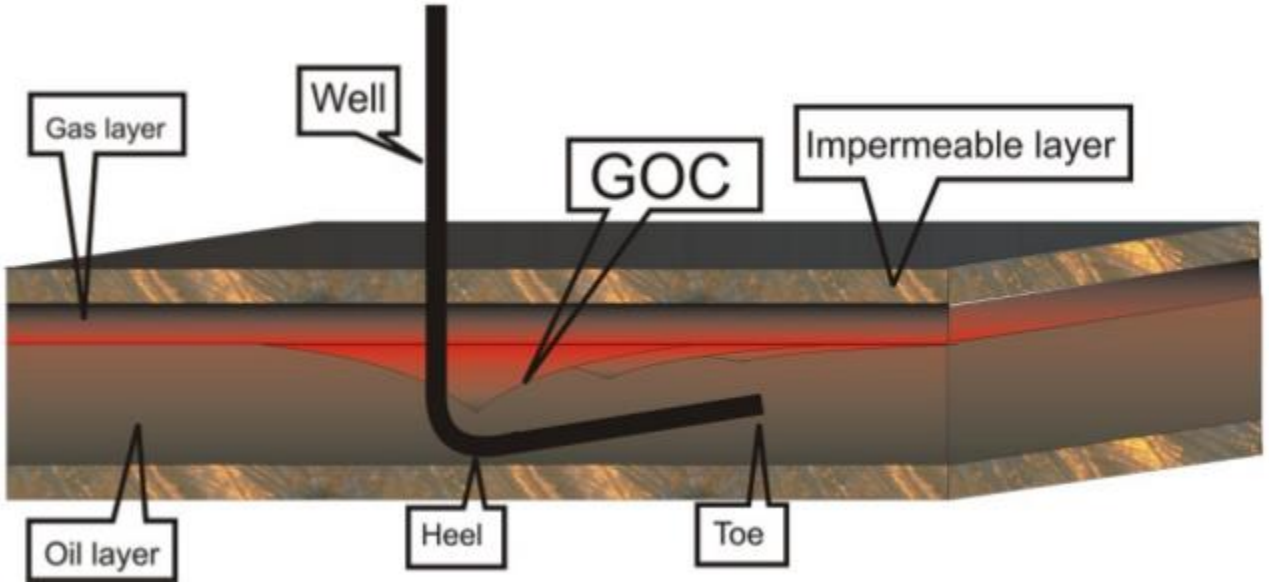
$$\min_{x,u} \quad J_d(x,u)$$

$$s.t. \quad x^+ = f(x,u)$$
$$g(x,u) \leq 0$$
$$x(0) = free, \qquad x(0) = x(N)$$
$$x \in X, u \in U$$

Refs: Maree and Imsland (2013, 2014)

Application: Utopia Tracking MPC

Thin oil-rim reservoir



Courtesy: IO-Center (www.iocenter.no)

GOC = Gas Oil Contact

- Objectives
  - Maximize oil production rate
  - Minimize Gas Oil Ratio (GOR)

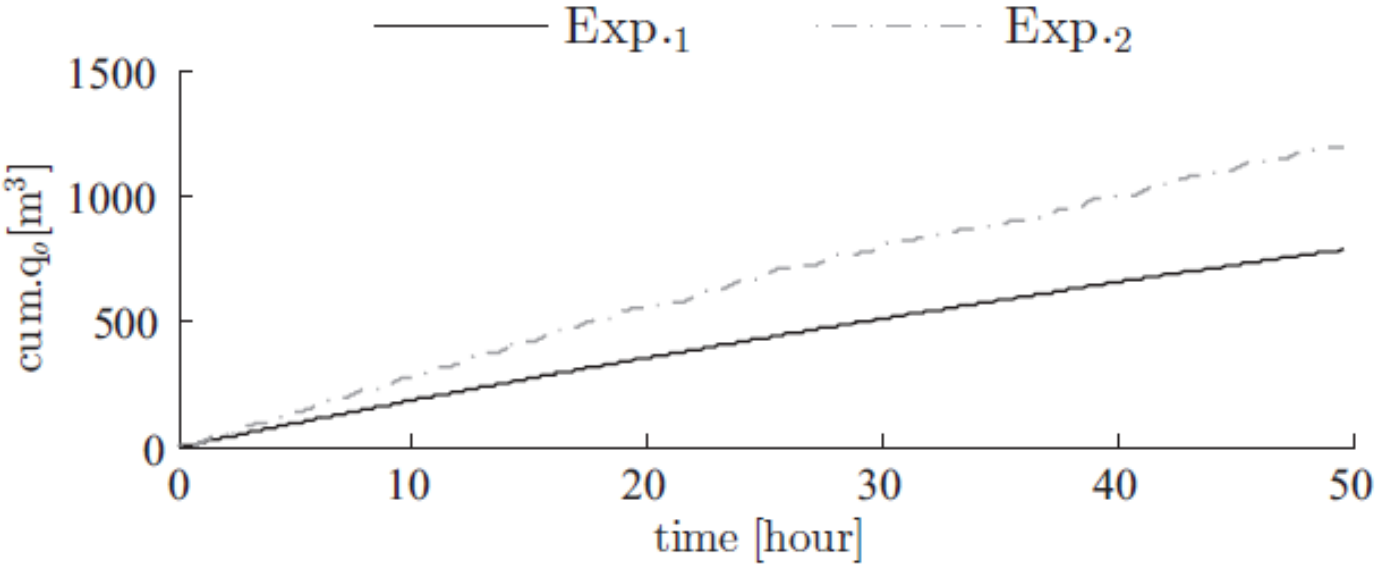
- Gas Coning after prolonged period of time
- Gas breakthrough

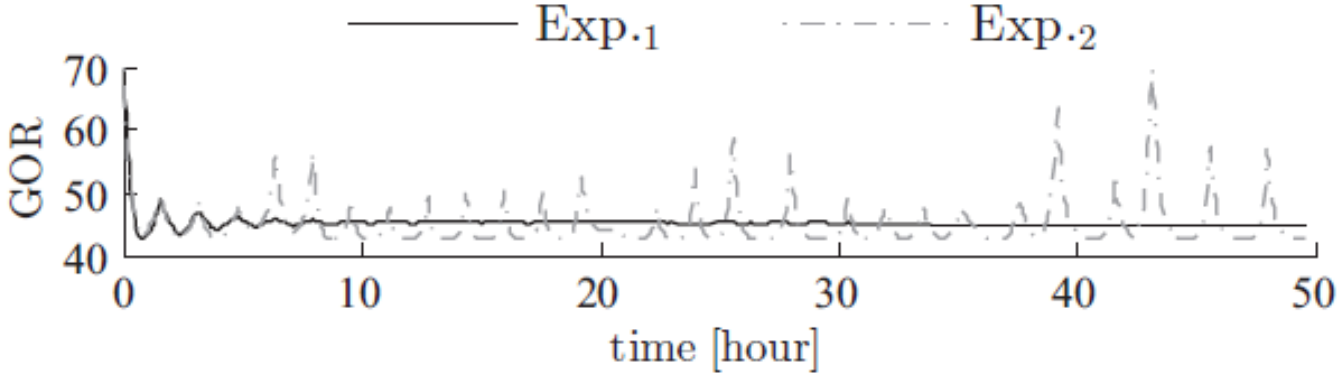
- Temporary shutdown
- Cyclic operation

Refs: Maree and Imsland (2014)

Application: Utopia Tracking MPC

Thin oil-rim reservoir





Exp.1 = Steady UT MPC

Exp.2 = Dynamic UT MPC

Maree and Imsland (2014)

References:

Adam (2009). World Nearing Peak Oil – Eventually peak Water. *The Adam Lee Commentary,* Information and Opinion about current events. Online: www. damrlee.org, Accessed: 24th Sep. 2014.

Andersson, J. (2000). A survey of multiobjective optimization in engineering design, *Technical Report LiTH-IKP-R-1097,* Department of Mechanical Engineering, Linköping University, Linköping, Sweden.

Essen, Van G. M., Hof, Den J. V. and Jansen, J. D. (2009). Hierarchical Long-Term and Short-Term Production Optimization, *SPE 124332, Annual Technical Conference and Exhibition,* New Orleans, Lousisiana, USA.

Fronseca, R. M., Leeuwenburgh O. and Jansen, J.D. (2012). Ensemble basd multi-objective production optimization of smart wells, *13th European Conference on the Mathematics of Oil Recovery,* Biarritz, France, 10-13 September 2012.

Gambier, A. (2008). MPC and PID Control Based On Multi-Objective Optimization, *2008 American Control Conference,* Westlin Seattle, Washington, USA, June 11-13,pp. 4727-4732.

Hof, P. V. D. (2013). Model based optimization and control of subsurface flow in oil reservoirs. *Plenary Lecture, 32nd Chinese Control Conference,* 26-28 July, Xian, China.

Isebor. O.J. and Durlofsky L.J. (2014). Biobjective optimization for general oil field development. *Journal of Petroleum Science and Engineering,* Vol. 119, pp. 123-138.

Kim, N. H. (2014). Approximation and Optimization in Engineering, *Lecture notes,* University of Florida, Gainesville, Florida.

Maree, J.P. and Imsland, L. (2013). Multi-objective predictive control for non steady state operation, *2013 European Control Conference,* Zurich, Switzerland, 17-19 July.

Marler, R. T. and Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering, *Struct Multidisc Optim,* Vol. 26, pp. 369-395.

Reyes, A.N., Dutra, C.B.S. and Bordons, C. (2002). Comparison of different predictive controllers with multi-objective optimization. Application to an Olive oil mill, *Proceedings of the 2002 IEEE International Conference on Control Applications,* Glasgow, Scotland, UK, 18-20 September.

Tlacuahuac. A.F., Morales, P. and Toledo, M.R. (2011). Multiobjective nonlinear model predictive control of a class of chemical reactors, *I&EC research, Special Issue: AMIDIQ 2011,* pp. 5891-5899.

Tyler, M.L. and Morari, M. (1998). Propositional logic in control and monitoring problems, *Automatica,* 35:565-582.

Wang, P. (2003). Development and applications of production optimization techniques for petroleum fields. *PhD Thesis,* Stanford University.

Zavla, V. M. and Tlacuahuac, A.F. (2012). Stability of multiobjective predictive control: A utopia-tracking approach, *Automatica,* 48(10):2627-2632.