

**Process Control-Design, Analysis and Assessment**  
**Doctoral Research Scholar Manikandan S**  
**Department of Chemical Engineering**  
**Indian Institute of Technology, Madras**  
**MATLAB Tutorial-Controller Design - Part 3**

Welcome everyone to MATLAB tutorial for process control analysis, design and assessment. In this tutorial, we look at MATLAB implementation of model predictive controller. Just as an introduction, I will give the formulation of MPC and I will directly go to the code section of MATLAB and then show you how to code implement model predictive controller and how we can during the model predictive controller in order to get the performance we require and so on.

(Refer Slide Time 1:00)

### MPC - Introduction

- MPC solves an optimization problem online
- Computes optimal input moves
- Can handle constraints
- Easily extendable to MIMO systems
- Computationally time consuming



So the model predictive controller is a controller which solves an optimisation problem online. So the optimisation problem is formulated such that the model predictive controller uses model and it can be easily solved. Most of the time, we solve a quadratic program optimisation. It computes optimal input moves which is optimal in the sense of the objective function we have chosen. And since this is an optimisation problem, it can handle constraints easily.

And it is easily extendable to MIMO systems. The example I am going to show here is for SISO system. But the implementation is not restricted to see SISO system. It can be extended to MIMO. Only that we will have to look at how the structure of each term in the objective function

and prediction oriented terms are in MIMO systems. And MPC is also computationally time-consuming but the examples which I am going to show here will be fast enough.

But if we have more inputs and more outputs for example like 25 cross 20 system where we have 25 out and we have 20 inputs then such systems will require more computational time in order to get the optimal in order to calculate the optimal solution for the optimisation.

(Refer Slide Time 2:55)

### MPC - formulation

$$\min_U \sum_{i=1}^P (y^{sp}[k+i] - \hat{y}[k+i])^2 + \sum_{i=1}^M u[k+i-1]^2$$

$$\hat{y}[k+i] = h_1 u[k+i-1] + h_2 u[k+i-2] + \dots + h_\gamma u[k+i-\gamma] \quad \forall i \in (1, P)$$

$$u[k+i] = \psi[k+M-1] \quad \forall i \in (M, P)$$

$$u^{LL} \leq u[k+i] \leq u^{UL} \quad \forall i \in (1, M)$$



Now this is the formulation where we minimise the objective function where  $Y$  set point minus  $\hat{Y}$  of  $K$  plus whole square and there is a minimum move objective added to the deviation from setpoint. Now the  $\hat{Y}$  of  $K$  plus  $I$  is computed as an impulse response that finite table's response model with gamma coefficients. And we have the constraint that  $U$  of  $K$  plus  $I$  equals to  $U$  of  $K$  plus  $M$  minus 1 where  $I$  is  $M$  to  $P$ . So what we are essentially saying is though we have gamma elements of  $U$  to optimise, we choose to optimise only  $M$  elements.

And on top of it, we have limit constraints on  $U$  for all time instants which is from 1 to  $M$ . Notice that I have not specified what  $P$  is and what  $M$  is and neither have I specified the limits. So these will depend on individual applications and we may have to modify them to suit to get a better performance. Look at how we can compute and make the computation easier for us to implement it in MATLAB.

(Refer Slide Time 4:37)

$$\min_u \sum_{i=1}^p (\hat{y}(k+i) - y(k+i))^2$$

$$\hat{y}(k+i) = h_1 u(k) + h_2 u(k-1) + \dots + h_\gamma u(k-i+1)$$

$$\hat{y}(k+2) = h_1 u(k+1) + h_2 u(k) + \dots + h_\gamma u(k-i+2)$$

$$\vdots$$

$$\hat{y}(k+m-1) = h_1 u(k+m-1) + \dots + h_\gamma u(k-i+m-1)$$

$$\hat{y}(k+m) = h_1 u(k+m) + \dots + h_\gamma u(k-i+m)$$

$$\hat{y}(k+p) = (h_1 + h_2) u(k+m-1) + \dots$$

$$\hat{y}(k+p) = (h_1 + \dots + h) u(k+m-1) + \dots$$

$$u \in \begin{pmatrix} u_k \\ u_{k+1} \\ \vdots \\ u_{k+m-1} \end{pmatrix}$$

$$u(k+m) = u(k+m-1)$$

$$\begin{pmatrix} h_1 & & & \\ h_2 & h_1 & & \\ h_3 & h_2 & h_1 & \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

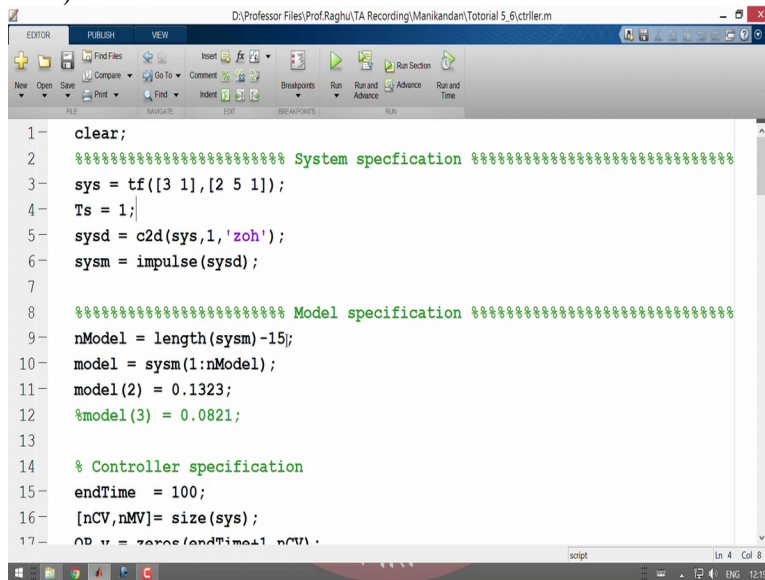


So we are computing summation I equals to 1 to P y of k plus i reference minus y hat of k plus i whole square. So we need to compute y of k plus I y hat of k plus i as a function of h1 to h gamma. So we have h1 u of k plus h2 u of k minus 1 all the way up to h gamma u of k minus gamma plus 1. Now in this optimisation function, we have minimized u. This u is constructed such that it has uk, uk plus 1 all the way up to uk plus m minus 1. So these are the unknowns which we which we have to optimise.

So this uk is unknown, whereas the rest of the terms are known. So this essentially evaluates to a constant. And the next term y of k plus 2 will be h1 times u of k plus 1 plus h2 times u of k plus all the way up to h gamma times u of k minus gamma plus 2. This will continue up to y hat of k plus m minus 1 which is h1 times u of k plus m minus 2 all the way up to h gamma times u of k minus gamma plus and minus 1. And y hat of k plus m is h1 times u of k plus m minus 1 all the way.

Similarly they will have y hat of k plus m plus 1. Here is where because the constraint that u of k plus m equals to u of k plus m minus 1. Because of that we will have h1 plus h2 times u of k plus m minus 1 plus all the way. So finally we will have y hat of k plus p which is h1 to h into u of k plus m minus 1 and etc. So we can notice the general structure where we have h1, h2 h1, h3 h2 h1, like this times u plus some known matrix which we can formulate.

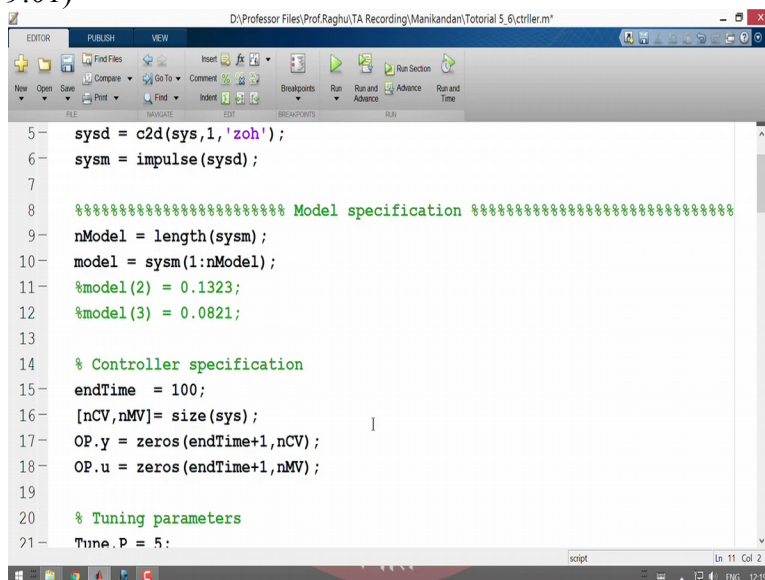
(Refer Slide Time 8:17)



```
1 clear;
2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% System specification %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3 sys = tf([3 1],[2 5 1]);
4 Ts = 1;
5 sysd = c2d(sys,1,'zoh');
6 sysm = impulse(sysd);
7
8 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Model specification %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9 nModel = length(sysm)-15;
10 model = sysm(1:nModel);
11 model(2) = 0.1323;
12 %model(3) = 0.0821;
13
14 % Controller specification
15 endTime = 100;
16 [nCV,nMV]= size(sys);
17 OP.u = zeros(endTime+1,nCV);
```

We will look at MPC implementation in MATLAB. Shown here is the code which implements an MPC for a SISO system. First I have declared the system here as a transfer function with the 3S plus 1 divided by 2S square plus 5S plus 1 as the transfer function. And I have discretised the system using a sampling time of 1 second. So TS equal to 1 implies that I have sampled the system once, for one second and I have computed the impulse coefficient using the inbuilt function, Impulse.

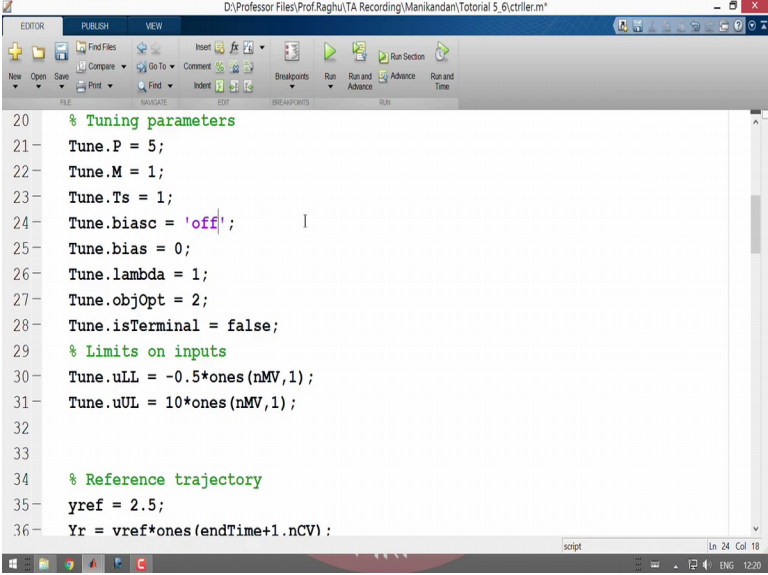
(Refer Slide Time 9:01)



```
5 sysd = c2d(sys,1,'zoh');
6 sysm = impulse(sysd);
7
8 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Model specification %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9 nModel = length(sysm);
10 model = sysm(1:nModel);
11 %model(2) = 0.1323;
12 %model(3) = 0.0821;
13
14 % Controller specification
15 endTime = 100;
16 [nCV,nMV]= size(sys);
17 OP.y = zeros(endTime+1,nCV);
18 OP.u = zeros(endTime+1,nMV);
19
20 % Tuning parameters
21 Tune.P = 5;
```

Now in the next section, I have declared the model. So as a start we will start with the same system as the model and see how the performance is. Then we will look at introducing plant model mismatch and see how the MPC performs. So I will just comment the this line also.

(Refer Slide Time 9:23)

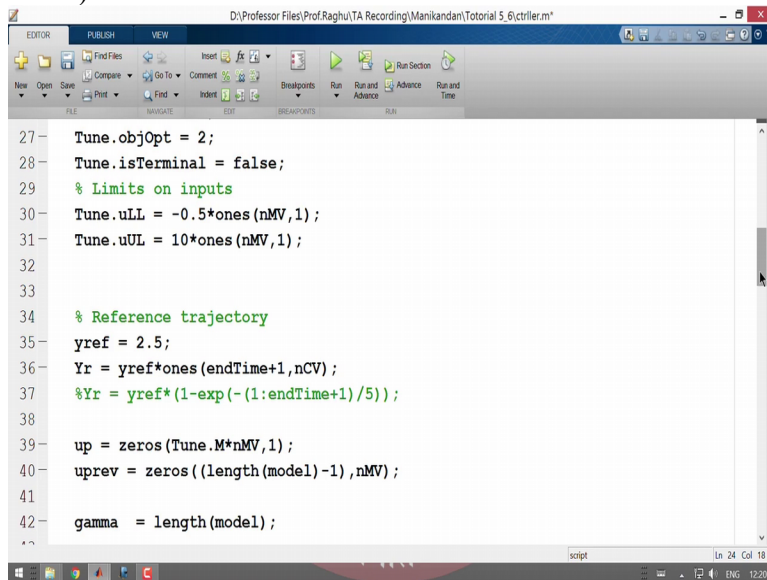


```
20 % Tuning parameters
21 Tune.P = 5;
22 Tune.M = 1;
23 Tune.Ts = 1;
24 Tune.biasc = 'off';
25 Tune.bias = 0;
26 Tune.lambda = 1;
27 Tune.objOpt = 2;
28 Tune.isTerminal = false;
29 % Limits on inputs
30 Tune.uLL = -0.5*ones(nMV,1);
31 Tune.uUL = 10*ones(nMV,1);
32
33
34 % Reference trajectory
35 yref = 2.5;
36 Yr = vref*ones(endTime+1,nCV);
```

Then I move on to control specification where I specify the length of the simulation as 100 seconds and we initialise the output parameters  $y$  and  $u$ . Then I move on to tuning parameters which we can change based on the requirement. So the tuned at  $P$  is the prediction horizon, tuned at  $N$  is the control horizon.  $TS$  is again sampling time. Bias and biasc are bias correction parameters. we will start with dim off now and we will see how they affect the controller performance later.

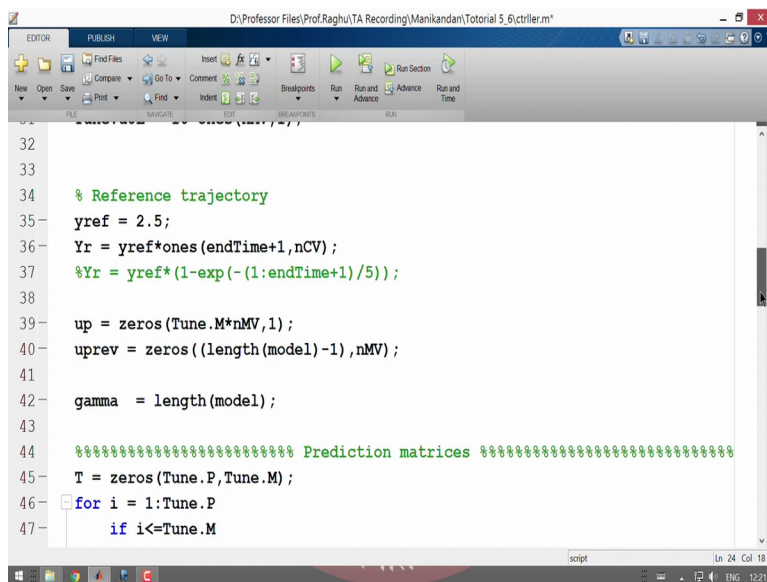
Lambda is the multiplier used for weighting the objectives which is deviation from setpoint square plus minimum move. So the subjective option helps us decide which objective to be optimised in the MPC formulation.

(Refer Slide Time 10:33)



```
D:\Professor Files\Prof.Raghu\TA Recording\Manikandan\Tutorial 5.6\ctrlr1.m
EDITOR PUBLISH VIEW
Find Files Compare Go To Comment Indent Breakpoints Run Run and Advance Run and Time
New Open Save Print Find Indent Breakpoints Run Run and Advance Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN

27 Tune.objOpt = 2;
28 Tune.isTerminal = false;
29 % Limits on inputs
30 Tune.uLL = -0.5*ones(nMV,1);
31 Tune.uUL = 10*ones(nMV,1);
32
33
34 % Reference trajectory
35 yref = 2.5;
36 Yr = yref*ones(endTime+1,nCV);
37 %Yr = yref*(1-exp(-(1:endTime+1)/5));
38
39 up = zeros(Tune.M*nMV,1);
40 uprev = zeros((length(model)-1),nMV);
41
42 gamma = length(model);
43
script Ln 24 Col 18
ENG 12:20
```



```
D:\Professor Files\Prof.Raghu\TA Recording\Manikandan\Tutorial 5.6\ctrlr1.m
EDITOR PUBLISH VIEW
Find Files Compare Go To Comment Indent Breakpoints Run Run and Advance Run and Time
New Open Save Print Find Indent Breakpoints Run Run and Advance Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN

32
33
34 % Reference trajectory
35 yref = 2.5;
36 Yr = yref*ones(endTime+1,nCV);
37 %Yr = yref*(1-exp(-(1:endTime+1)/5));
38
39 up = zeros(Tune.M*nMV,1);
40 uprev = zeros((length(model)-1),nMV);
41
42 gamma = length(model);
43
44 % Prediction matrices
45 T = zeros(Tune.P,Tune.M);
46 for i = 1:Tune.P
47     if i<=Tune.M
48
script Ln 24 Col 18
ENG 12:21
```

```

D:\Professor Files\Prof.Raghu\TA Recording\Manikandan\Tutorial 5_6\ctrlr.m*
EDITOR PUBLISH VIEW
New Open Save Print Find Go To Comment Breakpoints Run Run and Advance Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN
38
39 up = zeros(Tune.M*nMV,1);
40 uprev = zeros((length(model)-1),nMV);
41
42 gamma = length(model);
43
44 ***** Prediction matrices *****
45 T = zeros(Tune.P,Tune.M);
46 for i = 1:Tune.P
47     if i<=Tune.M
48         T(i,1:i) = model(i:-1:1);
49     else
50         if i<=length(model)
51             T(i,1:(Tune.M-1)) = model(i:-1:(i-Tune.M+2));
52             T(i,Tune.M) = sum(model((i-Tune.M+1):-1:1));
53         else
54             t = i-length(model);

```

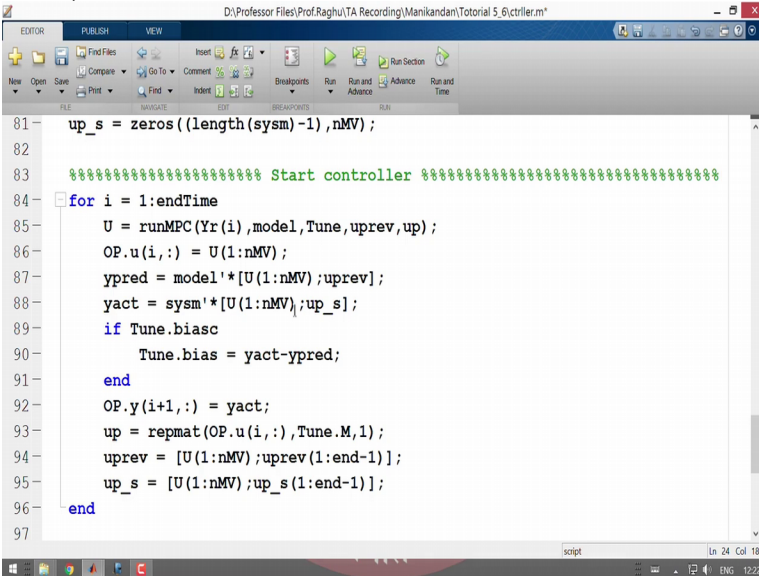
```

D:\Professor Files\Prof.Raghu\TA Recording\Manikandan\Tutorial 5_6\ctrlr.m*
EDITOR PUBLISH VIEW
New Open Save Print Find Go To Comment Breakpoints Run Run and Advance Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN
71     end
72     end
73     k=k+1;
74 end
75
76 Tune.T = T;
77 Tune.S = S;
78
79
80
81 up_s = zeros((length(sysm)-1),nMV);
82
83 ***** Start controller *****
84 for i = 1:endTime
85     U = runMPC(Yr(i),model,Tune,uprev,up);
86     OP.u(i,:) = U(1:nMV);
87     vpred = model'*[U(1:nMV);uprev];

```

Now then I have specified the lower and upper limits for input which are specified as minus 0.5 to 10. Then I declare reference trajectory which is the setpoint as of now as 2.5. And I declare such some internal variables. Now because we are using linear time invariant models, the matrices multiplying the decision variables  $u_k$  to  $u_k$  plus  $m$  minus 1 and the constant matrix are time invariant. Because of that, I precompute those matrices called T and S in this section and I store it in the structure tune.

(Refer Slide Time 11:24)



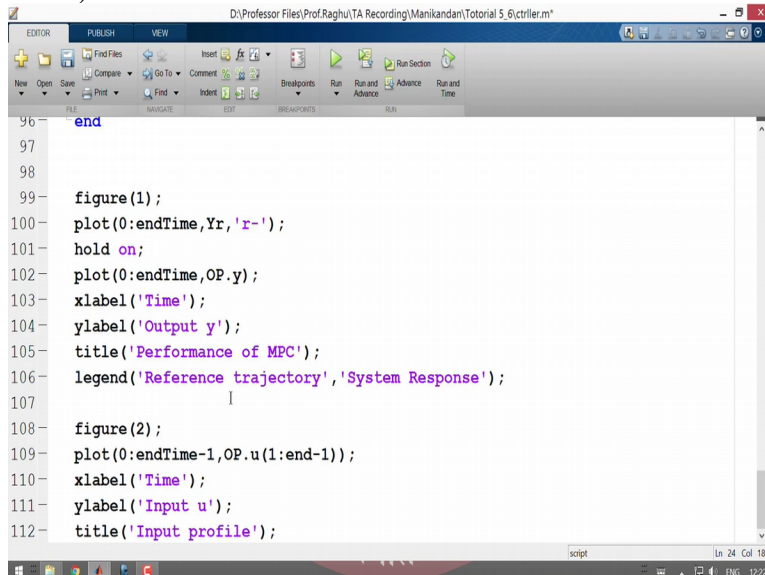
```
81 up_s = zeros((length(sysm)-1),nMV);
82
83 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Start controller %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
84 for i = 1:endTime
85     U = runMPC(Yr(i),model,Tune,uprev,up);
86     OP.u(i,:) = U(1:nMV);
87     ypred = model'*[U(1:nMV);uprev];
88     yact = sysm'*[U(1:nMV);up_s];
89     if Tune.biasc
90         Tune.bias = yact-ypred;
91     end
92     OP.y(i+1,:) = yact;
93     up = repmat(OP.u(i,:),Tune.M,1);
94     uprev = [U(1:nMV);uprev(1:end-1)];
95     up_s = [U(1:nMV);up_s(1:end-1)];
96 end
97
```

These are done apriory so that the computation time is reduced. Now runMPC is the functional which runs the optimisation for one iteration M provides solution from  $u_k$  to  $u_{k+m-1}$  and those values are stored in the variable u of which we will just select the first few first input variable, first-time instant input variables and apply it to the system. Notice that I have computed y predicted as well as y actual using model and system coefficients. This is to generalise the case where we plant model mismatch. So in case of plant model mismatch, predicted y and the actual y will be different.

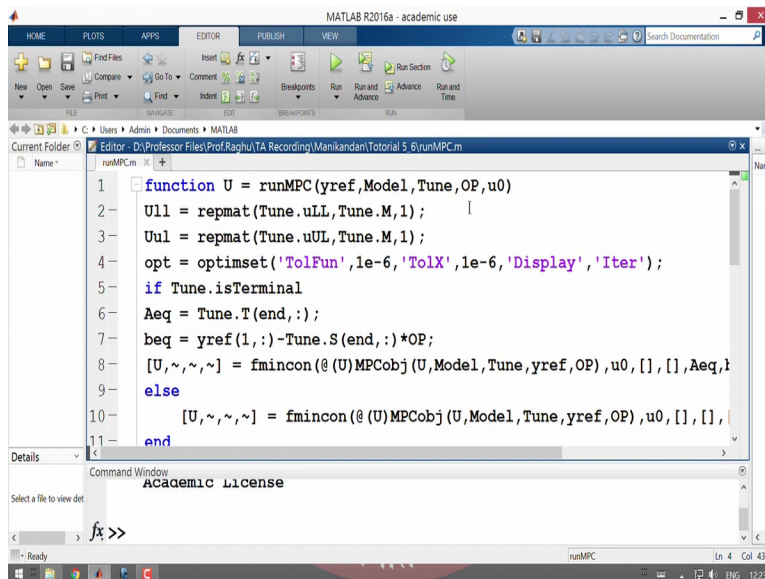
So we will have to correct for that bias in a in real-time fashion. So that has to be updated in the MPC formulation.



(Refer Slide Time 12:31)



```
D:\Professor Files\Prof.Raghu\TA Recording\Manikandan\Tutorial 5_6\ctrlr.m
EDITOR PUBLISH VIEW
Find Files Insert
Compare Go To Comment Breakpoints Run Run and Advance Run and Time
New Open Save Print Find Indent Run Section Run and Advance Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN
96 end
97
98
99 figure(1);
100 plot(0:endTime,Yr,'r-');
101 hold on;
102 plot(0:endTime,OP.y);
103 xlabel('Time');
104 ylabel('Output y');
105 title('Performance of MPC');
106 legend('Reference trajectory','System Response');
107
108 figure(2);
109 plot(0:endTime-1,OP.u(1:end-1));
110 xlabel('Time');
111 ylabel('Input u');
112 title('Input profile');
```



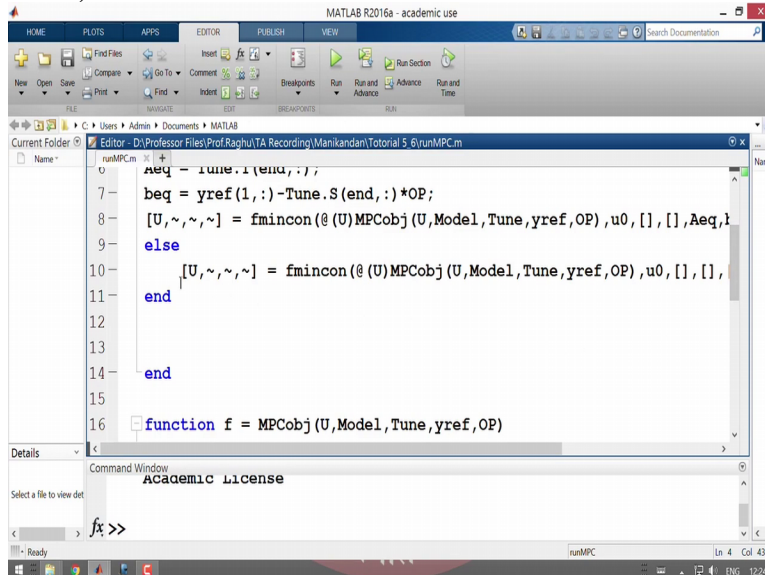
```
MATLAB R2016a - academic use
HOME PLOTS APPS EDITOR PUBLISH VIEW
Find Files Insert
Compare Go To Comment Breakpoints Run Run and Advance Run and Time
New Open Save Print Find Indent Run Section Run and Advance Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN
C:\Users\Admin\Documents\MATLAB
Current Folder Editor - D:\Professor Files\Prof.Raghu\TA Recording\Manikandan\Tutorial 5_6\runMPC.m
Name
runMPC.m
1 function U = runMPC(yref,Model,Tune,OP,u0)
2 U11 = repmat(Tune.uLL,Tune.M,1);
3 U1 = repmat(Tune.uUL,Tune.M,1);
4 opt = optimset('TolFun',1e-6,'TolX',1e-6,'Display','Iter');
5 if Tune.isTerminal
6 Aeq = Tune.T(end,:);
7 beq = yref(1,:)-Tune.S(end,:)*OP;
8 [U,~,~,~] = fmincon(@(U)MPCobj(U,Model,Tune,yref,OP),u0,[],[],Aeq,beq);
9 else
10 [U,~,~,~] = fmincon(@(U)MPCobj(U,Model,Tune,yref,OP),u0,[],[],[]);
11 end
Details
Command Window
Academic License
Select a file to view details
fx >>
runMPC Ln 4 Col 43
Ready ENG 12:23
```

So in order to do that, I have computed both predicted  $y$  and the actual  $y$ . The final section of the code consists of plotting the results and correspondingly I have named the trajectories and given labels for  $y$ ,  $x$  and titles. Let us look at how `runMPC` is configured so that we are sure about the coding. So here, you can see the `runMPC` function. It takes reference  $y$ , the model parameter as model, tuned structure as `tuned`, `OP` is the previous iteration previous time instant input variable.

These are used to compute the constant which is part of MPC formulation and the  $u$  not is the initial condition or initial guess for the optimisation. In the first 2 lines, I have created a matrix

containing the input lower limit and upper limits repeated m times. In order to do that, I have used MATLAB function, repmat.

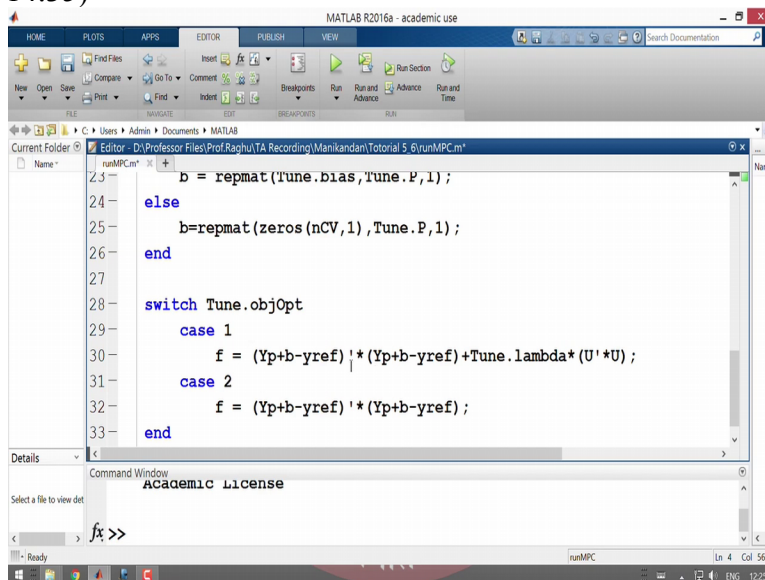
(Refer Slide Time 14:07)



```
runMPC.m
0
7   beq = tune.I(end,:);
8   beq = yref(1,:) - Tune.S(end,:) * OP;
9   [U,~,~] = fmincon(@(U)MPCobj(U,Model,Tune,yref,OP),u0,[],[],Aeq,beq,
10  else
11  [U,~,~] = fmincon(@(U)MPCobj(U,Model,Tune,yref,OP),u0,[],[],
12  end
13
14  end
15
16  function f = MPCobj(U,Model,Tune,yref,OP)
```

And I have also given a provision for specifying terminal constraint which is  $u$  of  $k$  plus  $p$  equals to  $y$  setpoint which we will do in a later case. So now we will be running this case where we have no constraints other than limits.

(Refer Slide Time 14:35)



```
runMPC.m
23   b = repmat(Tune.bias,Tune.P,1);
24  else
25   b=repmat(zeros(nCV,1),Tune.P,1);
26  end
27
28  switch Tune.objOpt
29  case 1
30     f = (Yp+b-yref)' * (Yp+b-yref) + Tune.lambda * (U' * U);
31  case 2
32     f = (Yp+b-yref)' * (Yp+b-yref);
33  end
```

```

24 else
25     b= repmat(zeros(nCV,1),Tune.P,1);
26 end
27
28 switch Tune.objOpt
29     case 1
30         f = (Yp+b-yref)'*(Yp+b-yref)+Tune.lambda*(U'*U);
31     case 2
32         f = (Yp+b-yref)'*(Yp+b-yref);
33 end
34 end

```

Command Window  
Academic License

Now this MPC objective function computes the MPC objective which is  $y$  minus  $y$  setpoint whole square times whole square summation  $y$  minus  $y$  setpoint whole square for  $I$  equals to  $k$  plus 1 to  $k$  plus  $p$ . And similarly we have summation  $u$  of  $k$  plus  $i$  whole square where  $i$  runs from 0 to  $m$  minus 1. So that is calculated here. In order to compute  $yp$ , we use that matrices which we computed as  $T$  and  $S$  here.  $T$  times  $u$ .  $u$  is the decision variables containing  $u$  of  $k$  to  $u$  of  $k$  plus  $m$  minus 1 and  $OP$  is the variable containing previous iteration  $u$  values.

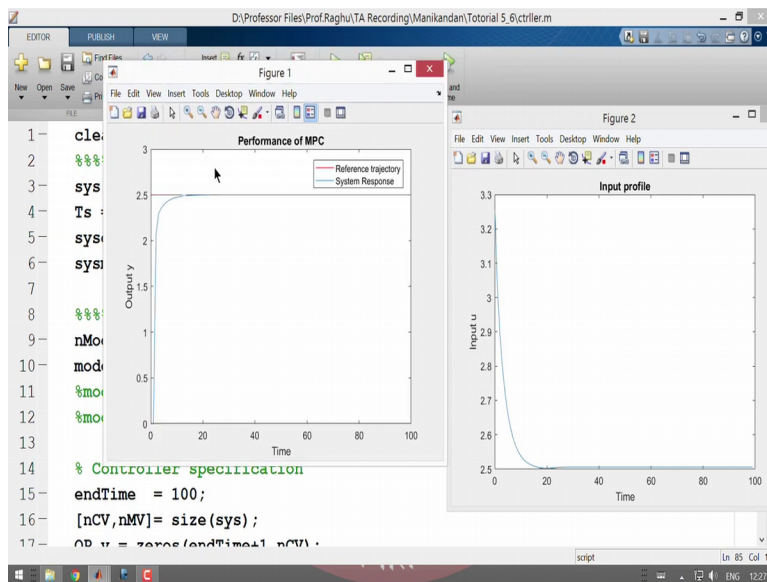
So we will have  $YP$  as a vector of  $P$  output variables and then we update the bias here by constructing a vector of  $P$  values repeated either as 0s or using the constant which is computed in the previous iteration. Then we have 2 cases where in one case, we do not have this summation  $u$  of  $k$  plus  $i$  whole square and in the other case where we have summation  $u$  of  $k$  plus  $i$  whole square, the effect of this will be easily seen later.

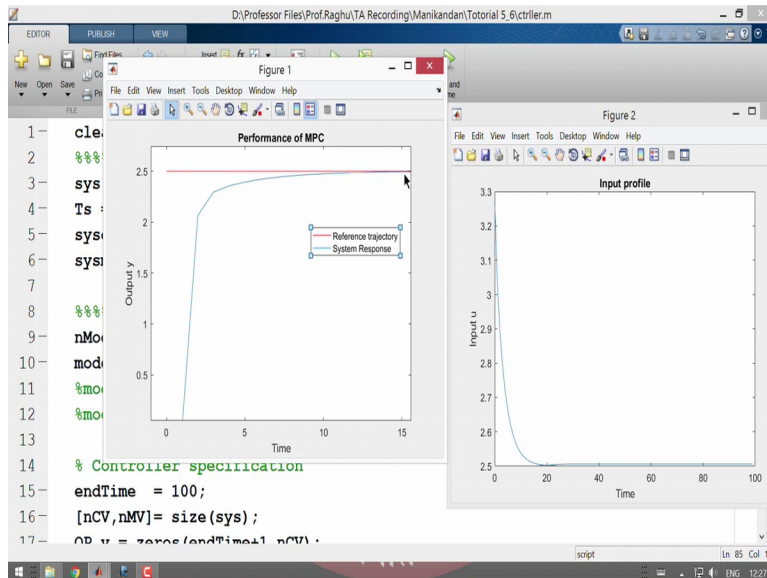
(Refer Slide Time 16:28)

```
clear;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% System specification %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sys = tf([3 1],[2 5 1]);
Ts = 1;
sysd = c2d(sys,1,'zoh');
sysm = impulse(sysd);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Model specification %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nModel = length(sysm);
model = sysm(1:nModel);
%model(2) = 0.1323;
%model(3) = 0.0821;

% Controller specification
endTime = 100;
[nCV,nMV] = size(sys);
OD u = zeros(endTime+1,nCV);
```





Let us move on and run the code. So this is the function that has to be studied. This is the script we have to run in order to see the performance of MPC. Let us run it. So now you can see that the system reaches the setpoint fairly quickly, within see 10 to 15 seconds. We can zoom in and see. So around 15 second it reaches to 0.5 and stays there. Notice the input variation. It starts from 3.3, goes down to 2.5. So it has 0.8 as the variation.

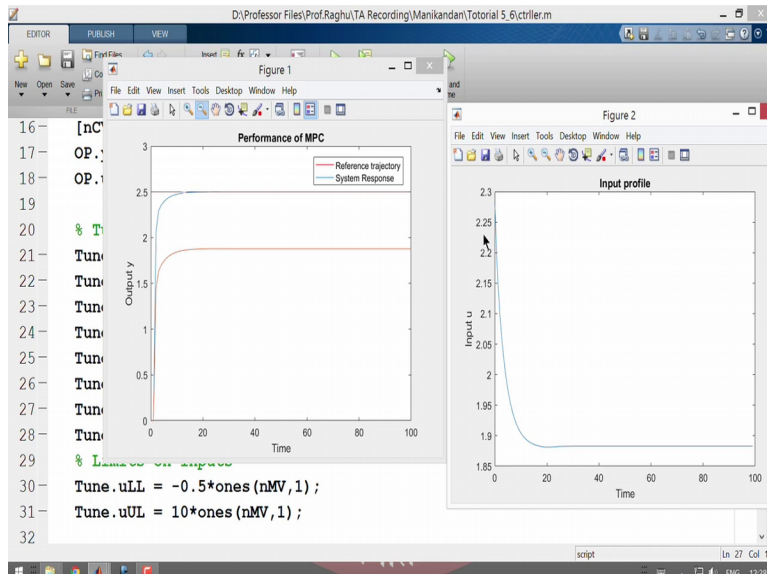
(Refer Slide Time 17:28)

The figure shows a MATLAB editor window with a script containing tuning parameters and limits on inputs. The script includes the following code:

```

16 [nCV,nMV]= size(sys);
17 OP.y = zeros(endTime+1,nCV);
18 OP.u = zeros(endTime+1,nMV);
19
20 % Tuning parameters
21 Tune.P = 5;
22 Tune.M = 1;
23 Tune.Ts = 1;
24 Tune.biasc = 'off';
25 Tune.bias = 0;
26 Tune.lambda = 1;
27 Tune.objOpt = 2;
28 Tune.isTerminal = false;
29 % Limits on inputs
30 Tune.uLL = -0.5*ones(nMV,1);
31 Tune.uUL = 10*ones(nMV,1);
32

```



Now without closing this window, I am going to just change the type of objective which we have used. So this, the objective we have used did not have the summation of  $u$  square. So if we just change the objective, how will the system perform is what we are going to look at now. Now you can see that initially the whole variation was around 3.3 to 2.8 which was 0.5 and here also it is similar but the output since the initial input was 3.3 at that time, the output was able to reach 2.5. But since we have minimal variation in the input as a objective, there is a bias in the performance.

(Refer Slide Time 18:36)

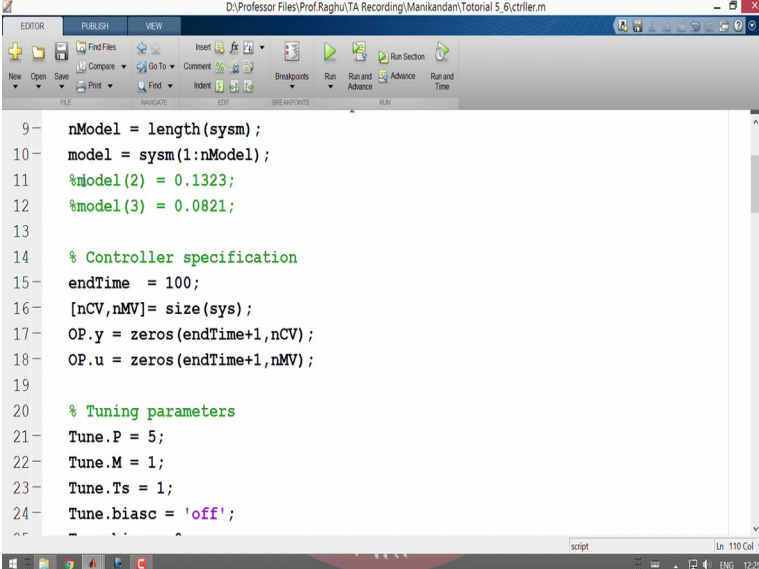
```

96 end
97
98
99 figure(1);
100 plot(0:endTime,Yr,'r-');
101 hold on;
102 plot(0:endTime,OP.y);
103 xlabel('Time');
104 ylabel('Output y');
105 title('Performance of MPC');
106 legend('Reference trajectory','System Response');
107
108 figure(2);
109 plot(0:endTime-1,OP.u(1:end-1));
110
111 xlabel('Time');
112 ylabel('Input u');

```

So we can tune that by tuning this lambda. We will put minimal weight to the 2<sup>nd</sup> term which is summation u square. Let us see how that works. Now you can see the response actually reach the setpoint and the variation is like 3.3 to 2.5 which was the previous variation also.

(Refer Slide Time 19:21)



```
9- nModel = length(sysm) ;
10- model = sysm(1:nModel);
11- %model(2) = 0.1323;
12- %model(3) = 0.0821;
13-
14- % Controller specification
15- endTime = 100;
16- [nCV,nMV]= size(sys);
17- OP.y = zeros(endTime+1,nCV);
18- OP.u = zeros(endTime+1,nMV);
19-
20- % Tuning parameters
21- Tune.P = 5;
22- Tune.M = 1;
23- Tune.Ts = 1;
24- Tune.biasc = 'off';
```

Now let us look at introducing plant model mismatch and how the system performs. I will just change the 2<sup>nd</sup> parameter to say 0.1323. So what we are saying is all the parameters other than 2<sup>nd</sup> term in the model is estimated properly. This particular coefficient is estimated in a wrong way. So if we do that without the bias correction, how will the performance be? Now you can see, we have oscillatory performance. This oscillation is because of the constraint which is being forced in the object optimisation function. You can see that it becomes minus 0.5 for 2 seconds and then it goes back. So this is the reason why the response is oscillatory.

(Refer Slide Time 20:33)

The image displays two screenshots of the MATLAB R2016a - academic use interface. The top screenshot shows the Editor window with a script named 'runMPC.m'. The code defines a control law based on the 'Tune' object's 'objOpt' property. It uses a switch statement to handle two cases for calculating the control signal 'f'. The workspace window shows various variables, including 'endTime', 'gamma', 'i', 'j', 'k', 'model', 'nCV', 'nModel', 'nNf', 'OP', 'S', 'sys', 'sysd', 'system', 'T', 'Ts', 'Tune', 'U', 'up', 'up.s', 'sprw', 'yact', 'yprd', 'Yr', and 'yref'.

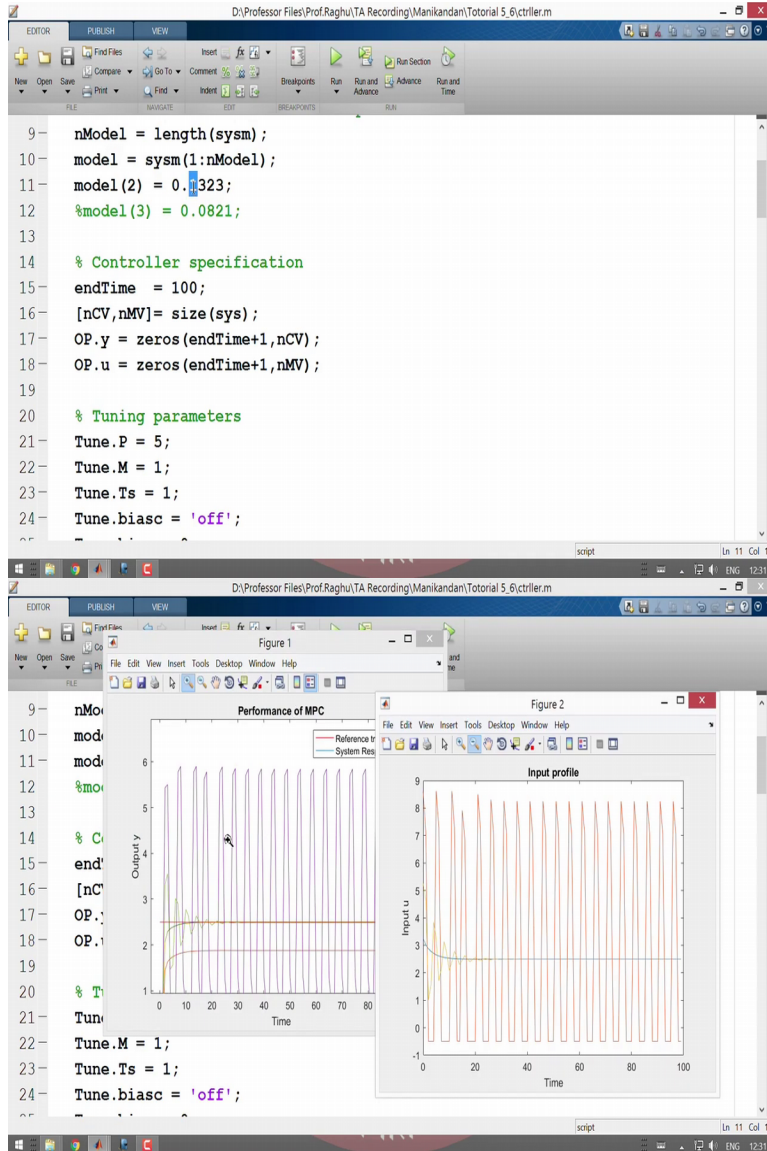
```
24 else
25     b=repmat(zeros(nCV,1),Tune.P,1);
26 end
27
28 switch Tune.objOpt
29     case 1
30         f = (Yp+b-yref)'*(Yp+b-yref)+Tune.lambda*(U'*U);
31     case 2
32         f = (Yp+b-yref)'*(Yp+b-yref);
33 end
34 end
```

Command Window:  
>> ctrlrler  
>> ctrlrler  
fx >>

The bottom screenshot shows the same MATLAB interface with the 'Variables - sysm' window open. This window displays a table of variables and their values. The 'model' variable is highlighted, showing a 24x1 double array. The workspace window shows the same variables as in the top screenshot.

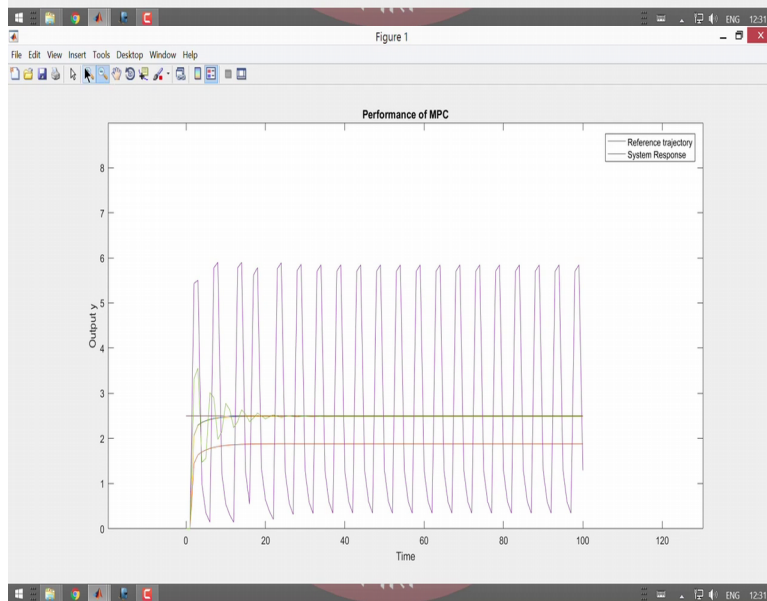
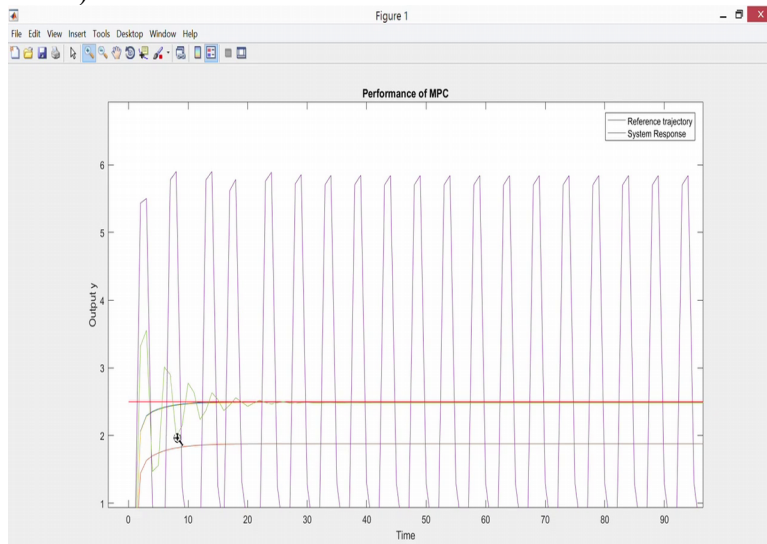
Variable	Value	Size
endTime	100	1x1
gamma	24	1x1
i	100	1x1
j	23	1x1
k	6	1x1
model	24x1 double	24x1
nCV	1	1x1
nModel	24	1x1
nNf	1	1x1
OP	1x1 struct	1x1
S	5x23 double	5x23
sys	1x1 tf	1x1
sysd	1x1 tf	1x1
system	24x1 double	24x1
T	800x132 double	8x1
Ts	1	1x1
Tune	1x1 struct	1x1
U	-0.5000	1x1
up	-0.5000	1x1
up.s	23x1 double	23x1
sprw	23x1 double	23x1
yact	1.2896	1x1
yprd	1.5396	1x1
Yr	101x1 double	101x1
yref	2.5000	1x1

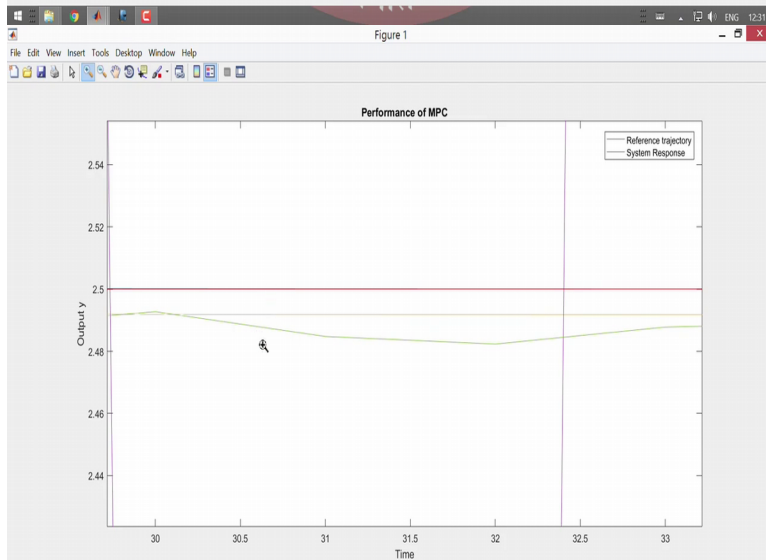
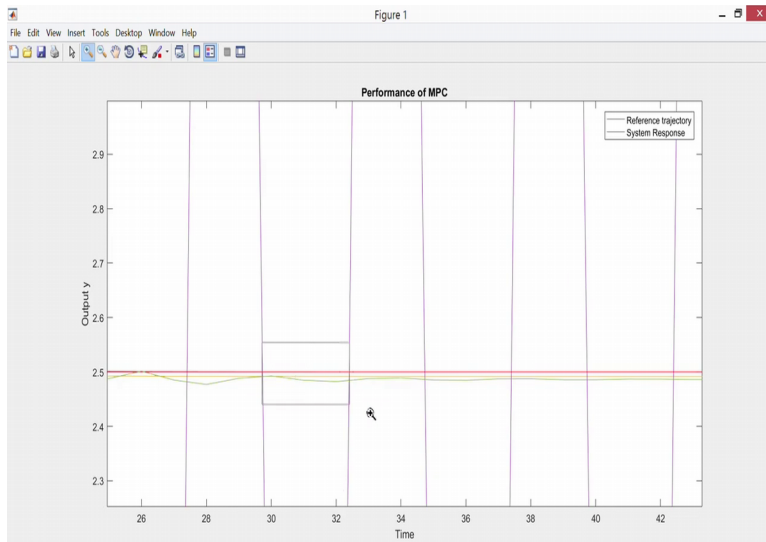




The actual model function of the 2<sup>nd</sup> constant has the value of 0.6 sorry, 0.6323. So when the model is changed, the 2<sup>nd</sup> coefficient is changed by 0.5, we have oscillatory response.

(Refer Slide Time 21:03)

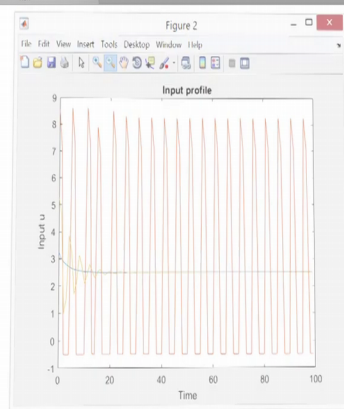




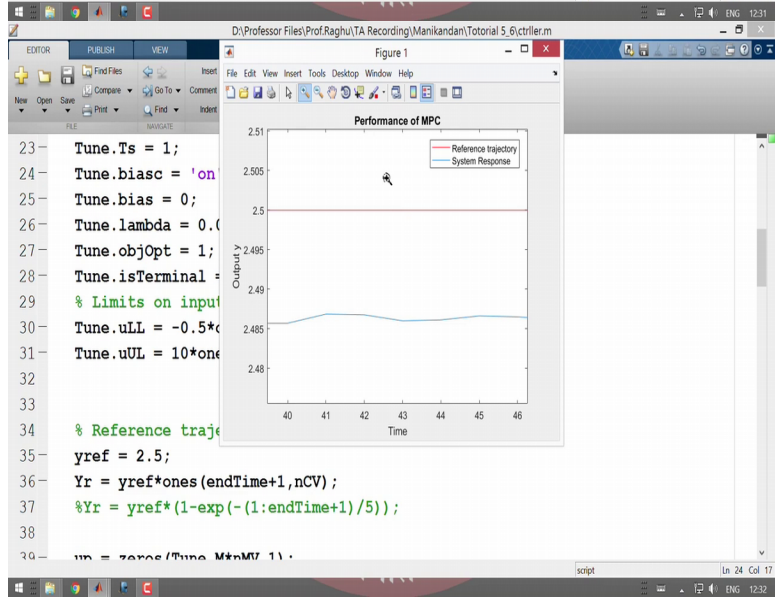
```

9      nModel = length(sys);
10     model = sysm(1:nModel);
11     model(2) = 0.3323;
12     %model(3) = 0.0821;
13
14     % Controller specification
15     endTime = 100;
16     [nCV,nMV] = size(sys);
17     OP.y = zeros(endTime+1,nCV);
18     OP.u = zeros(endTime+1,nMV);
19
20     % Tuning parameters
21     Tune.P = 5;
22     Tune.M = 1;
23     Tune.Ts = 1;
24     Tune.biasc = 'off';

```



```
D:\Professor Files\Prof.Raghu\TA Recording\Manikandan\Tutorial 5_6\ctrlr.m
EDITOR PUBLISH VIEW
New Open Save Print Find Indent Breakpoints Run Run and Advance Run and Time
23 Tune.Ts = 1;
24 Tune.biasc = 'off';
25 Tune.bias = 0;
26 Tune.lambda = 0.01;
27 Tune.objOpt = 1;
28 Tune.isTerminal = false;
29 % Limits on inputs
30 Tune.uLL = -0.5*ones(nMV,1);
31 Tune.uUL = 10*ones(nMV,1);
32
33
34 % Reference trajectory
35 yref = 2.5;
36 Yr = yref*ones(endTime+1,nCV);
37 %Yr = yref*(1-exp(-(1:endTime+1)/5));
38
39 u = zeros(Tune.M, nMV, 1);
```



```

D:\Professor Files\Prof.Raghu\TA Recording\Manikandan\Tutorial 5_6\ctrlr.m
EDITOR PUBLISH VIEW
File Edit Breakpoints Run Run and Advance Run and Time
New Open Save Print Compare Go To Comment Find Indent Breakpoints Run Run and Advance Run and Time
30 Tune.uLL = -0.5*ones(nMV,1);
31 Tune.uUL = 10*ones(nMV,1);
32
33
34 % Reference trajectory
35 yref = 2.5;
36 Yr = yref*ones(endTime+1,nCV);
37 %Yr = yref*(1-exp(-(1:endTime+1)/5));
38
39 up = zeros(Tune.M*nMV,1);
40 uprev = zeros((length(model)-1),nMV);
41
42 gamma = length(model);
43
44 %***** Prediction matrices %*****
45 T = zeros(Tune.P,Tune.M);
15 usages of "model" found script Ln 11 Col 1

```

We can change this to say 0.3323 where with we will have smaller variation than the previous case. We will have fairly stable response. You can see that this green curve goes to stable and it reaches the setpoint but if we zoom in, we can see. If we zoom in, so you can see that the green curve has a deviation from 2.5. Let us close this and we will turn on the bias correction and see how this works. So we have like 0.1 sorry 0.015 as the bias. It is still not reaching the setpoint. So we will have to tune depending on how different the model is.

The other way of eliminating this plant model mismatch is to add filters like Kalman filters which estimates the states online using measurements. We can tune the filter to have optimal performance and then we can couple the filter implementation along with MPC so that we will have better control.

(Refer Slide Time 23:02)

```
D:\Professor Files\Prof.Raghu\TA Recording\Manikandan\Tutorial 5_6\ctrlr.m
EDITOR PUBLISH VIEW
Find Files Compare Go To Comment Insert Find Indent Breakpoints Run Run and Advance Run and Time
New Open Save Print Find Indent Run Run and Advance Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN
30 Tune.uLL = -0.5*ones(nMV,1);
31 Tune.uUL = 10*ones(nMV,1);
32
33
34 % Reference trajectory
35 yref = 2.5;
36 %Yr = yref*ones(endTime+1,nCV);
37 Yr = yref*(1-exp(-(1:endTime+1)/5));
38
39 up = zeros(Tune.M*nMV,1);
40 uprev = zeros((length(model)-1),nMV);
41
42 gamma = length(model);
43
44 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Prediction matrices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
45 T = zeros(Tune.P,Tune.M);
iscpt Ln 37 Col 1
ENG 12:33
```

Next next let us at when the reference trajectory is changing how the system will respond? So we will change the reference trajectory as a first order response, first order response with sometime constant as 5 seconds and we will use the same model as the system and see how the system responds. So the idea is we are not restricted to using a single set point. We can give a reference trajectory which is a function of time and we can still control the system fairly, easily. So you can see that the reference trajectory is a first order response and the system was able to reach the trajectory.

(Refer Slide Time 24:08)

```
D:\Professor Files\Prof.Raghu\TA Recording\Manikandan\Tutorial 5_6\ctrlr.m
EDITOR PUBLISH VIEW
Find Files Compare Go To Comment Insert Find Indent Breakpoints Run Run and Advance Run and Time
New Open Save Print Find Indent Run Run and Advance Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN
12 %model(3) = 0.08;
13
14 % Controller spec
15 endTime = 100;
16 [nCV,nMV] = size(s);
17 OP.y = zeros(endTime,nCV);
18 OP.u = zeros(endTime,nMV);
19
20 % Tuning parameters
21 Tune.P = 5;
22 Tune.M = 1;
23 Tune.Ts = 1;
24 Tune.biasc = 'on';
25 Tune.bias = 0;
26 Tune.lambda = 0.01;
27 Tune.objOpt = 1;
28 Tune.isTerminal = false;
iscpt Ln 22 Col 11
ENG 12:35
```

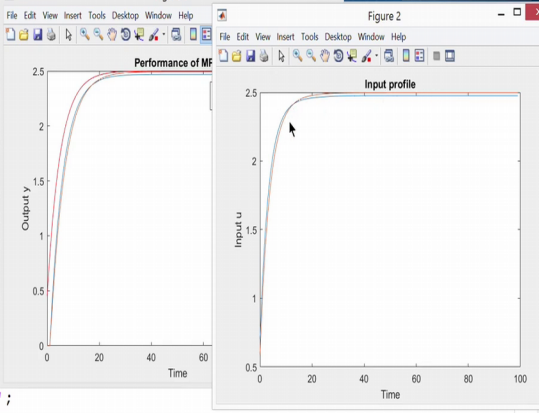


Figure 1: Performance of MF. The plot shows Output y on the y-axis (ranging from 0 to 2.5) versus Time on the x-axis (ranging from 0 to 60). The output starts at 0 and rises to a steady-state value of 2.5. The plot is titled 'Performance of MF'.

Figure 2: Input profile. The plot shows Input u on the y-axis (ranging from 0.5 to 2.5) versus Time on the x-axis (ranging from 0 to 100). The input starts at 0.5 and rises to a steady-state value of 2.5. The plot is titled 'Input profile'.

We can try changing  $m$  which I have put it as 1. Let us put it as 3 so that we will have more room for optimisation. So in that case, the variation will become smoother and easier compared to the case where it is 1. So the orange curve is the new cover which uses one setpoint and the corresponding curve is here. So you can see that it is kind of smooth. We can increase the puff prediction horizon together with control horizon to get a better performance. So in that case, we will sorry let us see here. So in that case we will get easier better approximations and hence we will get better control. Now we have looked at MPC implementation in MATLAB.

(Refer Slide Time 25:45)

**MPC - formulation**

$$\min_y \sum_{i=1}^P (y^{sp}[k+i] - \hat{y}[k+i])^2 + \lambda \sum_{i=1}^M u[k+i-1]^2$$

*Handwritten:*  $h = \begin{pmatrix} h_{11} & h_{12} & \dots & h_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ h_{m1} & h_{m2} & \dots & h_{mm} \end{pmatrix}$

$$\hat{y}[k+i] = h_1 u[k+i-1] + h_2 u[k+i-2] + \dots + h_\gamma u[k+i-\gamma] \quad \forall i \in (1, P)$$

$$u[k+i] = u[k+M-1] \quad \forall i \in (M, P)$$


$$u^{LL} \leq u[k+i] \leq u^{UL} \quad \forall i \in (1, M)$$

*Handwritten:*  $y^c = \hat{y}(k+i) \leq y^{UL}$

*Handwritten:*  $u^{LL} \leq \Delta u \leq \Delta u^{UL}$

*Handwritten:*  $\hat{y}(k+i) = y^{ref}(k+i)$

*Handwritten:*  $\rightarrow \hat{y}(k+P) = y^{ref}(k+P)$



We can do other customisations like we can add weights to different elements present in this as well as this. So that will advise the optimiser to put more weightage towards that prediction or less weightage towards that prediction. So we already have a tuning parameter  $\lambda$  multiplying this. So based on which we advise the optimiser to weight this the 2<sup>nd</sup> term such that the first term is either more weight, given more weightage given less weightage.

We can also add other constraints like  $y$  of  $k$  plus  $i$  lying between  $y$  of upper limit to  $y$  of lower limit, sorry  $y$  of lower limit to upper limit. Now this constraint maybe these constraints may not be achievable because we assume the system to be linear and finite impulse response system. So there are plant model mismatches. Because of that, we may not exactly match this constraint but we can give some room for the constraint to be relaxed so that we can still get a feasible solution. So adding these constraint which is on output limits, we will may lead to visibility problems.

And we can add other constraints like  $\Delta u$  being within some  $\Delta u$  limits and other constraints called coincidence point which will help us reach the set point faster by means of specifying  $\hat{y}$  of  $k + i$  to be equal to some  $y$  reference of  $k + i$ . But remember, this is also an output constraint. Because of that we may have problem of feasibility. The terminal constraint which is  $\hat{y}$  of  $k + p$  equal to  $y_{ref}$  of  $k + p$  is easy to implement and it also stabilises the controller. So we may, we will not have oscillatory response with this constraint present.

So MPC formulation is not just restricted to SISO systems, we can always use or extend this to MIMO system. In that case the  $h_1$  will become a matrix. So we will have  $h_{11}$   $h_{12}$  all the way up to  $h_{1m}$  and  $h_{n1}$  all the way up to  $h_{nn}$ . So similarly we will have for each coefficient like this. So it can be easily extended for MIMO systems. We can as well use nonlinear model to predict  $\hat{y}$  of  $k + i$ . But the problem is if we are to use on nonlinear model, the optimisation becomes nonlinear optimisation. So we may end up with local optimum rather than global optimum which is what we get in by solving this optimisation function.

So like the MPC eformulation by itself does not specify which type of model to use. Other than that, if we were to if we have to solve optimisation problem in accorded a programming framework or using linear programming principles, the model has to be linear. Other than that, there is no specification of how, which type of model to use for prediction. So we can use any type of model for prediction. We can add constraint on both  $u$ ,  $y$ ,  $\Delta u$  and other such constraints which may occur and we can as well customised the objective function based on minimizing  $u$  square or minimising  $\Delta u$  square, minimising time, et cetera.

So the MPC framework by itself is flexible enough but it has to be customised and each constraint we add may each equality constraint we we add and its output constraint we add may make the optimisation infeasible. So the stability and feasibility of this MPC framework has to be theoretically proved. So like this, we can implement MPC in MATLAB. There is also a toolbox called MPC Toolbox in MATLAB which is very easy to learn. At the same time, it will be it will have more diagnostic facilities and other options.

So I have shown here the traditional way of coding the MPC by hand and implementing it for any system. So with this, I finish MPC tutorial.