Uppsala University
Information Technology
Dept. of Systems and Control
Susanne Svedberg 0010
Last rev. October 15, 2002 by ME

Control Design (F, IT)
Computer Controlled Systems (W), 2002

Computer exercise 3

Model Predictive Control

Preparation exercises:
Preparation exercises 1–2.

Reading instructions: Chapter 9 and 16 in Reglerteori, Glad–Ljung –97 and this **entire** manual!

Name		Assistant's comments
Program	Year of reg.	
Date		
Passed prep. ex.	Sign	
Passed comp. ex.	Sign	

Contents

1 Introduction

Model predictive control is a rather new genre within automatic control. It is becoming more and more popular as the computer capacity is increasing and successful implementations are reported from around the world, especially from the chemical process industry. The purpose of this computer exercise is to get acquainted with some of the characteristics of model predictive control, by controlling a tank process. The knowledge about MPC earned here will later be used in a process laboration.

2 Model Predictive Control

Model Predictive Control, MPC, (*prediktionsreglering*) was first developed in the late seventies. MPC quickly became popular in the chemical and petrochemical process industries. It was simple and intuitive to use. The first algorithms used impulse or step response models that needed no or little *a priori* knowledge of the system. Today Model Predictive Control is a broad concept with many different strategies and algorithms that all use models to predict the effect of future control actions and thereafter minimize some criterion or cost function. Chapter 16 in Glad–Ljung describes the basic ideas behind MPC.

The most common algorithms are Model Algorithmic Control (MAC), Dynamic Matrix Control (DMC) and Generalized Predictive Control (GPC). The first two are the "original" algorithms and use an impulse response model and a step response model, respectively. The GPC on the other hand is based on a state space (*tillståndsform*) model. In this exercise the GPC will be used.

2.1 Theory

MPC uses a model to predict the output of the system given a control action. The controller chooses the most suitable control action given the criteria and bounds on the system. MPC will calculate the prediction of the outputs P steps ahead and this is called the prediction horizon. The control signals, or manipulated variables, are computed for M steps into the future, the control horizon. During the remaining prediction steps, $P - M$, the control signal is assumed to be kept constant, see Figure 1. Observe that the notations $P = M$ and $M = N$ are used in Glad–Ljung, but most publications and toolboxes use the notations employed here. Although the horizons, P and M, are tuning/design parameters and are usually determined through simulations and tests of the system and controller, there is a rule of thumb that the prediction horizon should equal the rise time of the system. The size of M determines the size of the optimization problem and in general the controller is faster if M is rather large, but $M \leq P$ must always hold. No matter how many control actions (i.e. how M is chosen) that are calculated, only the first is implemented. When new data is obtained

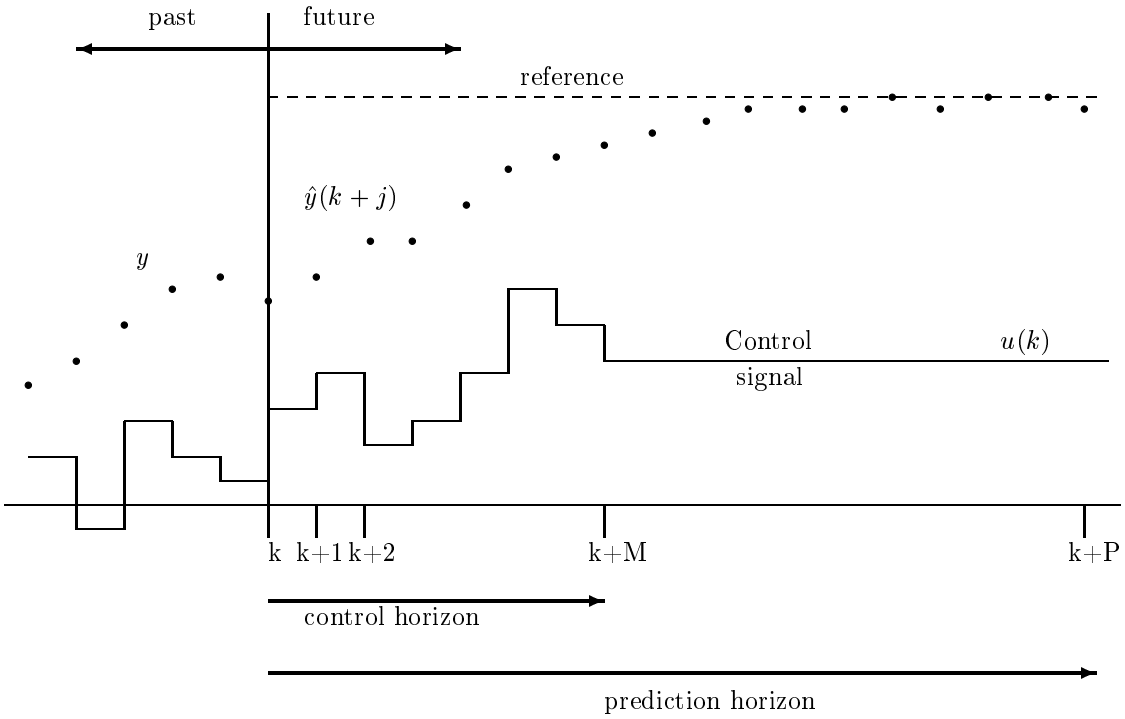


Figure 1: A schematic view of the horizons of Model Predicted Control.

the calculations are repeated, i.e. the problem is minimized at each sampling point.

The criterion or cost function, V, that is minimized during each sampling point is mainly the same as for the LQ-controllers, compare equations (16.5) and (9.5) in Glad–Ljung.

The system is assumed to be given in state space form:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned} \tag{1}$$

where all the states in x are assumed measurable. The basic criterion to be minimized, with respect to the input signal, is given by:

$$\begin{aligned} V(k) = &\sum_{j=1}^{P} \sum_{i=1}^{n_y} (Q_{1_i}(j)(r_i(k+j) - \hat{y}_i(k+j)))^2 \\ &+ \sum_{j=1}^{M} \sum_{i=1}^{n_u} (Q_{2_i}(j)\Delta\hat{u}_i(k+j))^2 \end{aligned} \tag{2}$$

where
n_y = the number of outputs
n_u = the number of inputs
Q_{1_i} = the weights on the deviation of the outputs from the references

Q_{2_i} = the weights on the change in the input signals
r_i = the reference signal (target)
$\hat{y}_i$ = the predicted output, see example 16.1 in Glad–Ljung
$\Delta\hat{u}_i$ = the predicted changes in the input

Here Q_1 and Q_2 are the weights for penalizing the different parts of the criteria. This cost function can be minimized analytically with methods mentioned in chapter 9 in Glad–Ljung. However a main idea in MPC is to extend the criterion with constraints in order to account for practical constraints in the system to be controlled. The way MPC allows the use of constraints is one of its greatest features and has strongly added to its popularity.

The constraints are usually on the form:

$$\begin{aligned} C_{ymin} &\leq y(t) \leq C_{ymax} \\ C_{umin} &\leq u(t) \leq C_{umax} \\ |\Delta u(t)| &\leq C_{urate} \end{aligned}$$

If there are constraints present the control design problem becomes nonlinear even though the model is linear. The minimization problem can no longer be solved analytically but instead a numerical minimization is needed. Linear constraints can be implemented by using QP, Quadratic Programming. There are today many efficient QP tools of different types.

The bounds on the input signal have been present in automatic control for ages but generally the signal has just been "cut off" at its maximum or minimum value. With MPC the controller "knows" it can only reach a particular value and only change in a certain rate. The biggest difference between MPC and traditional controllers are the constraints on the controlled and output variables used in the MPC design. Since a model is used to predict the outcomes, the controller can prevent the system from violating its constraints, given that the constraints are "feasible", i.e. a solution exists.

3 The tank process

The tank system consists of two water tanks that are connected in cascade. The process is pictured schematically in Figure 2.

This system has been thoroughly described in one of the process labs in "Continuous Control Systems" and will be presented again in one of the process laborations in this course. The numerical values and the linearization are all derived from these process laborations. The water level is measured in Volts and is roughly half of the height in cm.

By making a linearization around a working point, the system can be described as :

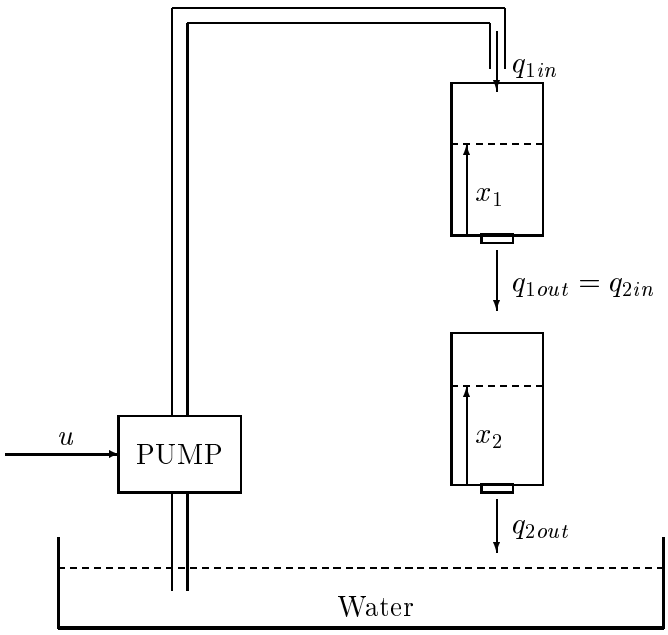


Figure 2: The tank process

$$X_1(s) = \frac{k_o}{1+sT}U(s) \qquad\qquad X_2(s) = \frac{1}{1+sT}X_1(s)$$

where the input signal, u, is the voltage to the electrical pump and x_1 and x_2, are the water levels in the upper and lower tank, respectively. The water level in the upper tank is controlled by the pump and the water level in the lower tank depends on the water level in the upper tank.

Although only the water level in the lower tank is to be controlled, both levels are assumed to be measurable. This is important since both the tanks have physical limits which should not be violated and therefore constraints on both the tanks will be included in the MPC criterion.

3.1 The system in state space form

Preparation exercise 1:
Write the system on state space form in continuous time. Assume that the parameters are $k_o = 1.66$, $T = 94\ s$ at the working point ($u \approx 2\ Volt$, $x_1 \approx x_2 \approx 5\ Volt$).

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned}$$

Choose the water levels as the state vectors and the water level in the lower tank to be the output signal.

ANSWER:

4 LQ control of the tank process

LQ and LQG design methods are described in chapter 9 in Glad–Ljung.

In the first simulation task, you will study LQ control (linear quadratic state-feedback control) of the tank process. We will first give a short overview of the design methodology.

A state space model of the SISO tank process in discrete time is:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned} \tag{3}$$

If all states can be measured, then eq.(3), where y is a scalar, can be controlled with the state-feedback:

$$u(k) = -Lx(k) + mr(k) \tag{4}$$

A linear quadratic state-feedback controller can be designed such that the gain matrix L in the state-feedback law (4) minimizes a quadratic cost function:

$$V = ||e||_Q^2 + ||u||_R^2 = \min \sum_{k=1}^{\infty} [r(k) - y(k]^2 Q_1 + [u(k)]^2 Q_2 \tag{5}$$

The closed loop system can then be described as

$$\begin{aligned} x(k+1) &= (A - BL)x(k) + Bmr(k) \\ y(k) &= Cx(k) \end{aligned} \tag{6}$$

Preparation exercise 2:
Suggest how to write a Matlab m-file given the continuous system parameters $\texttt{A}_c, \texttt{B}_c, \texttt{C}_c$ and $\texttt{D}_c\texttt{(=0)}$ (for example, obtained from the earlier preparation exercise):

$$\begin{aligned} \dot{x} &= A_c x + B_c u \\ y &= C_c x + D_c u \end{aligned}$$

that:

- converts the model to discrete time assuming a sampling interval Ts.
- calculates L from the LQ criteria.
  Hint: In order to punish the output and input use following weights in the Matlab function **dlqr**:

  $$Q = Q_1 C^T C \qquad\qquad R = Q_2$$

- calculates m so that the static gain ($r \rightarrow y = x_2$) equals 1.
- simulates the closed loop system given a reference signal, $r(t)$.

Use the Matlab commands described in the Appendix. Observe that you do not have to implement this but give a suggestion on how to do it.

ANSWER:

4.1 Simulations

The design of an LQ controller and simulation of the tank system controlled by the LQ controller are built into some Matlab functions:

lq_sim0.m where the system and controller are simulated without any limitations. This is one solution to preparation exercise 2.

lq_sim1.m where the system is simulated with its physical limitations:
pump: $0 \leq u \leq 9$ Volts tanks: $0 \leq y_i \leq 10$ Volts
This means that the pump interprets all input signals above 9 as 9 Volts and all signals less than 0 as 0 Volts. The tanks are simulated so that the water levels can not be less than zero and the tanks overflow at 10 Volts, i.e. the heights can not be more than 10 Volts.

All simulations are performed around the linearization point, i.e. the simulations of the tank start at 4 Volts.
The simulation functions can be called with:

lq_sim0($\mathbf{Q}_1$, $\mathbf{Q}_2$, r0, Tscale, tsamp, n_std)
lq_sim1($\mathbf{Q}_1$, $\mathbf{Q}_2$, r0, Tscale, tsamp, n_std)

where

$\mathbf{Q}_1$ is the weight on the output signal and must be given. Note that only the ratio Q_1/Q_2 determines the characteristics of the controller.

$\mathbf{Q}_2$ is the weight on the manipulated signal and must be given.

r0 is the size of the reference step for the water level in the lower tank given in Volts. At t=10 s, the reference signal is changed from 4 to $4 + r0$ Volts. The default value is $r0 = 2$ Volts.

Tscale is the duration of the simulation. Default value is [360] seconds.

tsamp is the sampling time. Default is [1] second.

n_std is the standard deviation of the measurement noise. Default is [0].

Computer exercise 1
a, Run your program from preparation exercise 2 (alternatively run **lq_sim0($\mathbf{Q}_1$, $\mathbf{Q}_2$)**). Try some different values on Q_1 and Q_2 and verify that it is only the ratio Q_1/Q_2 that affects the control performance. How does Q_1/Q_2 affect the control performance? Finally tune Q_1/Q_2 so that the rise time is approximately 40s. What are your values on Q_1 and Q_2?

ANSWER:

b, Add the physical constraints on the simulated system by running **lq_sim1(Q_1, Q_2)**. Try the same values on Q_1 and Q_2 as in the previous exercise. Does anything change? Tune the controller so that it works as good as possible without violating the constraints. What values on Q_1 and Q_2 do you choose?

ANSWER:

c, Now try the "good" controller with another reference value. Try both $r0 = 1$ and 5 while keeping the "good" Q_1 and Q_2. Use the function: **lq_sim1(Q_1, Q_2, r0)**
Does the controller still work well with respect to speed and violation of constraints? Comment on the result!

ANSWER:

For a given set point change it is in this case possible to design an LQ controller that gives a nice step-response for the system. However, for a larger set point change this design will cause problem by violating the physical constraints (i.e. tank overflow). It might therefore be a good idea to use the limitations on the system when designing the controller. That is what the MPC does. MPC uses a linear quadratic criteria and is similar to the LQ controller in many ways. Comparing equation (2) with equation (5) you can see that the prediction horizon, P, in MPC corresponds to ∞ in LQ.

5 Model Predictive Control of the tank process

The great advantage of MPC is the way the controller implements the known constraints of the system. Here the effect of different design parameters will be evaluated.
The system to be controlled is the tank system described earlier, which will also be used in the process laboration.

5.1 Simulations

With the following Matlab functions, model predictive control of the tanks can be simulated:

mpc_sim0.m where the system is simulated without any limitations and no constraints are taken into account in the controller design.

mpc_sim1.m where the system is simulated with its physical limitations.

mpc_sim2.m where the constraints are taken into account in the control design.

There are many parameters that can be manipulated. The design and simulation functions can be called with:
mpc_sim0($\mathbf{Q}_1$, $\mathbf{Q}_2$, r0, Tscale, tsamp, n_std)
mpc_sim1($\mathbf{Q}_1$, $\mathbf{Q}_2$, r0, Tscale, tsamp, n_std)
mpc_sim2(ylim, ulim, $\mathbf{Q}_1$, $\mathbf{Q}_2$, M, P, r0, n_std, Tscale, tsamp)
where

$\mathbf{Q}_1$ is the weight on the output signal, the water level in the lower tank. Default value for Q_1 is [1].

$\mathbf{Q}_2$ is the weight on the manipulated signal. Default value for Q_2 is [0.01].

r0 is the size of the reference step for the water level in the lower tank given in Volts. At t=10 s, the reference signal is changed from 4 to $4 + r0$ Volts. Default value is [2] Volts.

ylim are the constraints on the two output signals, the tank levels: [$C_{y_1 min}$ C_{y2min} $C_{y_1 max}$ C_{y2max}]. The default values are [0 0 10 10] in mpc_sim2.

ulim are the constraints on the manipulated signal, the pump voltage: [C_{umin} C_{umax} C_{urate}]. The default values are [0 9 9] in mpc_sim2.

M is the control horizon (called N in Glad–Ljung). It is usually much smaller than P, it can be from 1 up to P. The size of the optimization problem is determined by the size of M. Default value is [3].

P is the prediction horizon (called M in Glad-Ljung) and should cover the rise time of the system. Default value is [40].

n_std is the standard deviation of the measurement noise. Default is [0].

Tscale is the duration of the simulation. Default value is [360] seconds.

tsamp is the sampling time. Default is [1] second.

By only typing mpc_sim# (where # = 0, 1 or 2) in the Matlab window, the program will design a controller and simulate the closed loop system using default values. To change a default value, proceed as follows: for example to change M from 3(default) to 2 in mpc_sim2 type: **mpc_sim2([ ], [ ], [ ], [ ], [2])**. Note that the parameters before M must be either set or left empty, (otherwise the function will not know which parameter to change!)

5.1.1 Constraints

The tank system has a few natural constraints. The most obvious one is that the tanks have a limited height. The water level can vary between 0 Volt (= 0 cm), empty tank, and 10 Volts (≈ 20 cm), full tank. Notice that although we do not want to control the upper tank level, it is still subject to the constraints. The input signal also has a few constraints. The pump is a one way pump, it will not suck up water from the upper tank, so $u \geq 0$! It does not have infinite capacity in pumping either, its maximum is 9 Volts. The pump dynamics is negligible (in comparison with the tank dynamics). Therefore $C_{urate} = C_{umax} - C_{umin} = 9$ Volts/sample.

$$\begin{aligned} 0 \leq y_1(t) &\leq 10 \\ 0 \leq y_2(t) &\leq 10 \\ 0 \leq u(t) &\leq 9 \\ |\Delta u(t)| &\leq 9 \end{aligned}$$

Computer exercise 2:
a, Start by running **mpc_sim0** with the default values. This equals lq_sim0 where there are no limitations in the system or in the control design. Are any signals violating the real physical constraints on the system? Comment!

ANSWER:

b, Run **mpc_sim1**. This shows the response of the system with physical constraints on the system but with the same controller as in `mpc_sim0`.Compare with **2.a**. Comment the result?

ANSWER:

c, Run **mpc_sim2** with default values. This gives a controller with the physical constraints added to the criteria, ylim and ulim are given!. Any difference from above or compared to lq_sim1? Comment!

ANSWER:

d, Make a change in the reference signal to $r0 = 5$. Run **mpc_sim2([0 0 10 10],[0 9 9],[ ],[ ],[ ],[ ],[5])**. Compare this to the LQ controller in Computer exercise 1.c. Which controller works best. Explain why?

ANSWER:

5.1.2 Weights

The MPC uses quadratic programming and iterates until it finds an optimal control signal that also satisfies the constraints. The weights Q_1 and Q_2 corresponds to the "importance" of the output to follow the reference value and small changes in the control signal, respectively. The weights are correlated and the quotient Q_1/Q_2 is of importance! See the weights in the LQ chapter.

Computer exercise 3:
In the exercise above, constraints were added and the controller kept the system within its physical limits. You could see that the changes in the input signal were very quick. Try to make the control signal change in a more smooth way by changing the weights. How does Q_1/Q_2 affect the control performance?

ANSWER:

How does Q_1/Q_2 affect the system when there is measurement noise with the standard deviation $n\_std = [0.01]$
The exercise can be solved by using the following syntax:
mpc_sim2([ylim],[ulim],[$\mathbf{Q}_1$],[$\mathbf{Q}_2$],[ ],[ ],[ ],[n_std]) Comment!

ANSWER:

5.1.3 Horizons

Typical design variables for all MPC algorithms are the prediction horizon, P, and most often the control horizon, M. The prediction horizon describes how many steps ahead the controller evaluates to complete its task. The control horizon is how many control actions that are to be calculated at each step, but only the first one is implemented. The horizon lengths in real time depend on the sampling time, *tsamp*. Usually *tsamp* also affects the change rate, Δu (change/second $\Rightarrow$ change/sample), but since the pump dynamics is negligible at the sampling times used here, Δu will equal $u_{max} - u_{min}$.

Computer exercise 4:
a, With a rise time of 40 seconds for the tank system, give reasonable values for P for the sampling intervals $tsamp = 1s,\ 2.5s,\ 5s$? (Rule of thumb!)

ANSWER:

b, Try different values on M, P and *tsamp*. For example try $M = 1, 5, 10$ and $P = 10, 20, 100$ and *tsamp* from above. How does *tsamp* affect the impact of the horizons? How does M and P affect the computing time and rise time?
The exercise can be solved by using the following syntax:
mpc_sim2([ylim],[ulim],[Q_1],[Q_2],[M],[P],[ ],[ ],[ ], [tsamp])
If the simulation takes too long time, >2min, press **ctrl-c**.

ANSWER:

Computer exercise 5:
a, Run mpc_sim2 with its default values except M. Compare the execution times and the rise times for $M = 1$ and 5. Why is the execution time longer when M is larger? Why is the rise time shorter when M is longer?

ANSWER:

b, Compare the execution times when $r0 = 2V$ (default) and $6V$. The other parameters should have their default values. Is there a difference? If there is, explain why.

ANSWER:

c, Looking at the last plot (r0=6), how does the MPC handle situations when the reference signal is outside the constraints of the controller? (*If a longer simulation is necessary run with* **M=1**).

ANSWER:

Computer exercise 6:
Design a model predictive controller that satisfactory controls the tank process given a reference step of 3 Volts and a measurement noise with a standard deviation of 0.005. What are your values on M, P, $tsamp$ and Q_1/Q_2? Motivate your choices.
mpc_sim2([ylim],[ulim],[Q_1],[Q_2],[M],[P],[r0],[n_std],[ ],[tsamp])

ANSWER:

A Matlab functions

» help c2d

C2D Conversion of continuous-time models to discrete time.

SYSD = C2D(SYSC,TS,METHOD) converts the continuous-time LTI model SYSC to a discrete-time model SYSD with sample time TS. The string METHOD selects the discretization method among the following: 'zoh' Zero-order hold on the inputs. 'foh' Linear interpolation of inputs (triangle appx.) 'tustin' Bilinear (Tustin) approximation. 'prewarp' Tustin approximation with frequency prewarping. The critical frequency Wc is specified as fourth input by C2D(SYSC,TS,'prewarp',Wc). 'matched' Matched pole-zero method (for SISO systems only). The default is 'zoh' when METHOD is omitted.

For state-space models SYS and the 'zoh' or 'foh' methods,

[SYSD,G] = C2D(SYSC,TS,METHOD) also returns a matrix G that maps continuous initial conditions into discrete initial conditions. Specifically, if x0,u0 are initial states and inputs for SYSC, then equivalent initial conditions for SYSD are given by xd0 = G * [x0;u0], ud0 = u0 .

See also D2C, D2D, LTIMODELS.

Overloaded methods help zpk/c2d.m help tf/c2d.m help ss/c2d.m help lti/c2d.m help frd/c2d.m

» help dcgain

DCGAIN DC gain of LTI models.

K = DCGAIN(SYS) computes the steady-state (D.C. or low frequency) gain of the LTI model SYS.

If SYS is an array of LTI models with dimensions [NY NU S1 ... Sp], DCGAIN returns an array K with the same dimensions such that K(:,:,j1,...,jp) = DCGAIN(SYS(:,:,j1,...,jp)) .

See also NORM, EVALFR, FREQRESP, LTIMODELS.

Overloaded methods help zpk/dcgain.m help tf/dcgain.m help ss/dcgain.m help frd/dcgain.m

» help dlqr

DLQR Linear-quadratic regulator design for discrete-time systems.

[K,S,E] = DLQR(A,B,Q,R,N) calculates the optimal gain matrix K such that the state-feedback law u[n] = -Kx[n] minimizes the cost function

J = Sum x'Qx + u'Ru + 2*x'Nu

subject to the state dynamics x[n+1] = Ax[n] + Bu[n].

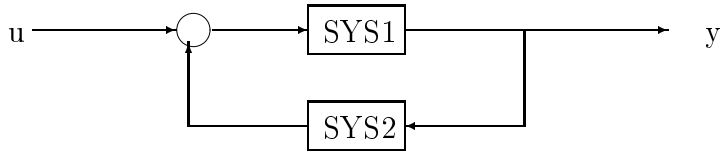
The matrix N is set to zero when omitted. Also returned are the Riccati equation solution S and the closed-loop eigenvalues E: -1 A'SA - S - (A'SB+N)(R+B'SB) (B'SA+N') + Q = 0, E = EIG(A-B*K).

See also DLQRY, LQRD, LQGREG, and DARE.

» help feedback

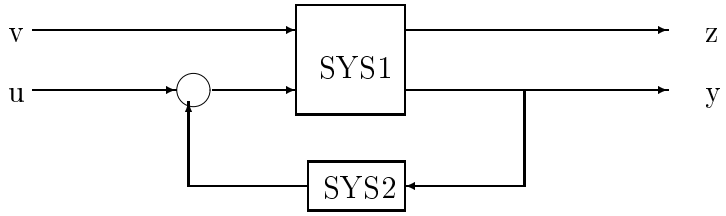
FEEDBACK Feedback connection of two LTI models.

SYS = FEEDBACK(SYS1,SYS2) computes an LTI model SYS for the closed-loop feedback system



Negative feedback is assumed and the resulting system SYS maps u to y. To apply positive feedback, use the syntax SYS = FEEDBACK(SYS1,SYS2,+1).

SYS = FEEDBACK(SYS1,SYS2,FEEDIN,FEEDOUT,SIGN) builds the more general feedback interconnection:



The vector FEEDIN contains indices into the input vector of SYS1 and specifies which inputs u are involved in the feedback loop. Similarly, FEEDOUT specifies which outputs y of SYS1 are used for feedback. If SIGN=1 then positive feedback is used. If SIGN=-1 or SIGN is omitted, then negative feedback is used. In all cases, the resulting LTI model SYS has the same inputs and outputs as SYS1 (with their order preserved).

If SYS1 and SYS2 are arrays of LTI models, FEEDBACK returns an LTI array SYS of the same dimensions where SYS(:,:,k) = FEEDBACK(SYS1(:,:,k),SYS2(:,:,k)) .

See also LFT, PARALLEL, SERIES, CONNECT, LTIMODELS.

Overloaded methods help zpk/feedback.m help tf/feedback.m help ss/feedback.m help lti/feedback.m help frd/feedback.m

» help lsim

LSIM Simulate time response of LTI models to arbitrary inputs.

LSIM(SYS,U,T) plots the time response of the LTI model SYS to the input signal described by U and T. The time vector T consists of regularly spaced time samples and U is a matrix with as many columns as inputs and whose i-th

row specifies the input value at time T(i). For example, t = 0:0.01:5; u = sin(t); lsim(sys,u,t) simulates the response of a single-input model SYS to the input u(t)=sin(t) during 5 seconds.

For discrete-time models, U should be sampled at the same rate as SYS (T is then redundant and can be omitted or set to the empty matrix). For continuous-time models, choose the sampling period T(2)-T(1) small enough to accurately describe the input U. LSIM checks for intersample oscillations and resamples U if necessary.

LSIM(SYS,U,T,X0) specifies an additional nonzero initial state X0 (for state-space models only).

LSIM(SYS1,SYS2,...,U,T,X0) simulates the response of multiple LTI models SYS1,SYS2,... on a single plot. The initial condition X0 is optional. You can also specify a color, line style, and marker for each system, as in lsim(sys1,'r',sys2,'y–','sys3,'gx',u,t).

When invoked with left-hand arguments, [YS,TS] = LSIM(SYS,U,T) returns the output history YS and time vector TS used for simulation. No plot is drawn on the screen. The matrix YS has LENGTH(TS) rows and as many columns as outputs in SYS. WARNING: TS contains more points than T when U is resampled to reveal intersample oscillations. To get the response at the samples T only, extract YS(1:d:end,:) where d=round(length(TS)/length(T)).

For state-space models, [YS,TS,XS] = LSIM(SYS,U,T,X0) also returns the state trajectory XS, a matrix with LENGTH(TS) rows and as many columns as states.

See also GENSIG, STEP, IMPULSE, INITIAL, LTIMODELS.

Overloaded methods help lti/lsim.m help frd/lsim.m

» **help ss**

SS Create state-space model or convert LTI model to state space.

Creation: SYS = SS(A,B,C,D) creates a continuous-time state-space (SS) model SYS with matrices A,B,C,D. The output SYS is a SS object. You can set D=0 to mean the zero matrix of appropriate dimensions.

SYS = SS(A,B,C,D,Ts) creates a discrete-time SS model with sample time Ts (set Ts=-1 if the sample time is undetermined).

SYS = SS creates an empty SS object. SYS = SS(D) specifies a static gain matrix D.

In all syntax above, the input list can be followed by pairs 'PropertyName1', PropertyValue1, ... that set the various properties of SS models (type LTIPROPS for details). To make SYS inherit all its LTI properties from an existing LTI model REFSYS, use the syntax SYS = SS(A,B,C,D,REFSYS).

Arrays of state-space models: You can create arrays of state-space models by using ND arrays for A,B,C,D above. The first two dimensions of A,B,C,D determine the number of states, inputs, and outputs, while the remaining dimensions

specify the array sizes. For example, if A,B,C,D are 4D arrays and their last two dimensions have lengths 2 and 5, then SYS = SS(A,B,C,D) creates the 2-by-5 array of SS models SYS(:,:,k,m) = SS(A(:,:,k,m),...,D(:,:,k,m)), k=1:2, m=1:5. All models in the resulting SS array share the same number of outputs, inputs, and states.

SYS = SS(ZEROS([NY NU S1...Sk])) pre-allocates space for an SS array with NY outputs, NU inputs, and array sizes [S1...Sk].

Conversion: SYS = SS(SYS) converts an arbitrary LTI model SYS to state space, i.e., computes a state-space realization of SYS.

SYS = SS(SYS,'min') computes a minimal realization of SYS.

See also LTIMODELS, DSS, RSS, DRSS, SSDATA, LTIPROPS, TF, ZPK, FRD.

» **help ssdata**

— help for ss/ssdata.m —

SSDATA Quick access to state-space data.

[A,B,C,D] = SSDATA(SYS) retrieves the matrix data A,B,C,D for the state-space model SYS. If SYS is not a state-space model, it is first converted to the state-space representation.

[A,B,C,D,TS] = SSDATA(SYS) also returns the sample time TS. Other properties of SYS can be accessed with GET or by direct structure-like referencing (e.g., SYS.Ts).

For arrays of LTI models with the same order (number of states), A,B,C,D are multi-dimensional arrays where A(:,:,k), B(:,:,k), C(:,:,k), D(:,:,k) give the state-space matrices of the k-th model SYS(:,:,k).

For arrays of LTI models with variable order, use the syntax [A,B,C,D] = SSDATA(SYS,'cell') to return the variable-size A,B,C matrices into cell arrays.

See also SS, GET, DSSDATA, TFDATA, ZPKDATA, LTIMODELS, LTIPROPS.

There is more than one ssdata available. See also help lti/ssdata.m help frd/ssdata.m