ECEN415
Advanced Control Engineering

Christopher Hollitt
School of Engineering and Computer Science
Victoria University of Wellington

Trimester Two, 2021
Revised October 11, 2021

Contents

1 Controllability

1.1 Operating Points

Part I
Operating Points

In a regulator problem we are interested in stabilising a system about some steady state operating point $\boldsymbol{x}_\text{o}$, which we will assume corresponds to some steady state input $\boldsymbol{u}_\text{o}$.

$$\begin{aligned}\text{Now,}\quad \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \\ \text{So,}\ \frac{\mathrm{d}(\boldsymbol{x}-\boldsymbol{x}_\text{o})}{\mathrm{d}t} &= \boldsymbol{A}(\boldsymbol{x}-\boldsymbol{x}_\text{o}) + \boldsymbol{B}(\boldsymbol{u}-\boldsymbol{u}_\text{o})\end{aligned}$$

Let's define $\boldsymbol{\xi} = \boldsymbol{x} - \boldsymbol{x}_\text{o}$ and $\boldsymbol{v} = \boldsymbol{u} - \boldsymbol{u}_\text{o}$ as deviations from the desired steady state values.

$$\implies \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\xi} = \boldsymbol{A}\boldsymbol{\xi} + \boldsymbol{B}\boldsymbol{v}$$

The deviations are governed by the same state equation as the state. We typically drop the deviation variables and proceed without loss of generality.

In effect we will design a controller to regulate the system around $\boldsymbol{\xi} = 0$ and then use that *same* controller to regulate around some arbitrary $\boldsymbol{x}_\text{o}$. Notice that the fact that the model around $\boldsymbol{x}_\text{o}$ is unchanged from the general model is a consequence of the assumed linearity of the system. Once we have formed a model around the operating point $\boldsymbol{x}_\text{o}$ we will then forget about $\boldsymbol{x}_\text{o}$ while designing the controller. However, when it comes to implementation we *must* remember to include the $\boldsymbol{u}_\text{o}$ signal required to produce the desired operating point. That is, we will develop control techniques to producing an input signal that will actually be $\boldsymbol{v}$. We will need to apply a real signal of $\boldsymbol{u}_\text{o} + \boldsymbol{v}$ to our system.

We will need to know what steady state $\boldsymbol{u}_\text{o}$ we need to apply to hold the system at some operating point $\boldsymbol{x}_\text{o}$. Now,

$$\dot{\boldsymbol{x}}_\text{o} = \boldsymbol{A}\boldsymbol{x}_\text{o} + \boldsymbol{B}\boldsymbol{u}_\text{o}$$

But at steady state $\dot{\boldsymbol{x}} = 0$, so

$$\begin{aligned}-\boldsymbol{A}\boldsymbol{x}_\text{o} &= \boldsymbol{B}\boldsymbol{u}_\text{o} \\ \implies \boldsymbol{u}_\text{o} &= -\boldsymbol{B}^{-1}\boldsymbol{A}\boldsymbol{x}_\text{o}\end{aligned}$$

This last equation makes sense *only* if $\boldsymbol{B}$ is invertible. This requires that there be exactly as many actuators as there are system states ($m = n$) and that those actuators not be arranged in a way that yields a singular $\boldsymbol{B}$. In such a case we can solve uniquely for $\boldsymbol{u}_\text{o}$

We can also identify the $\boldsymbol{x}_\text{o}$ locations for which it is possible find some $\boldsymbol{u}_\text{o}$ that results in the system remaining at $\boldsymbol{x}_\text{o}$.

$$\begin{aligned}\dot{\boldsymbol{x}}_\text{o} &= \boldsymbol{A}\boldsymbol{x}_\text{o} + \boldsymbol{B}\boldsymbol{u}_\text{o} \\ 0 &= \boldsymbol{A}\boldsymbol{x}_\text{o} + \boldsymbol{B}\boldsymbol{u}_\text{o} \quad \text{as } \boldsymbol{x} \text{ is not changing} \\ -\boldsymbol{A}\boldsymbol{x}_\text{o} &= \boldsymbol{B}\boldsymbol{u}_\text{o} \\ \boldsymbol{x}_\text{o} &= -\boldsymbol{A}^{-1}\boldsymbol{B}\boldsymbol{u}_\text{o}\end{aligned}$$

Therefore the set of possible equilibrium points in $\boldsymbol{x}$ is

$$\begin{aligned}\mathcal{E}_{\boldsymbol{x}} &= \{\boldsymbol{x}_\text{o} : \boldsymbol{A}\boldsymbol{x}_\text{o} + \boldsymbol{B}\boldsymbol{u}_\text{o} = 0, \boldsymbol{u}_\text{o} \in \mathbb{R}^m\} \\ &= \{\boldsymbol{x}_\text{o} : \boldsymbol{x}_\text{o} = -\boldsymbol{A}^{-1}\boldsymbol{B}\boldsymbol{u}_\text{o}, \boldsymbol{u}_\text{o} \in \mathbb{R}^m\} \\ \implies \mathcal{E}_{\boldsymbol{x}} &= \mathcal{R}(\boldsymbol{A}^{-1}\boldsymbol{B})\end{aligned}$$

Let's return to our pendulum example, where $\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

$$\begin{aligned} \text{Here, } \boldsymbol{A}^{-1} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \text{ so } \mathcal{E}_{\boldsymbol{x}} &= \mathcal{R}\left(\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) \\ &= \mathcal{R}\begin{bmatrix} -1 \\ 0 \end{bmatrix} \end{aligned}$$

Thus, the space of possible equilibrium points is a line, where the pendulum has zero angular velocity ($x_2 = 0$), but arbitrary angular position (x_1 is some multiple of -1).

You should always check that a proposed control scheme allows you to position the system at the desired operating point(s). Notice that $\dim(\mathcal{E}_{\boldsymbol{x}}) \leq \operatorname{rank}(\boldsymbol{B})$, because $\boldsymbol{A}^{-1}\boldsymbol{B}$ can have no more independent columns that $\operatorname{rank} \boldsymbol{B}$. For example, if a system has one input we can only choose where to put the system operating point along a one dimensional subspace (a line), no matter the dimensionality of the state space. Where that line lies in the state space is determined by the structure of the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices, which is to say by the physics of the system and the actuators. Many real systems have $\operatorname{rank} \boldsymbol{B} < n$, because in practice we don't always need to be able to force the position to arbitrary $\boldsymbol{x}$ (or $\boldsymbol{y}$). As long as $\mathcal{E}_{\boldsymbol{x}}$ includes the desired operating point we can minimise the number of actuators required.

We know that if $\boldsymbol{B}$ is invertible we can find the unique $\boldsymbol{u}_\text{o}$ to produce a desired $\boldsymbol{x}_\text{o}$. However, we also need to consider what happens when we cannot invert $\boldsymbol{B}$, either because it is singular, or because it is not square.

If $m \neq n$ then we cannot invert $\boldsymbol{B}$. There are two cases to consider.

- $m > n$ ($\boldsymbol{B}$ is fat) and $\operatorname{rank} \boldsymbol{B} = n$. We have more actuators than states. In such a case there are typically multiple strategies that would produce a desired $\boldsymbol{x}_\text{o}$. That is, we have a *set* of possible $\boldsymbol{u}$ that would satisfy our requirements. In such a case we might select the smallest $\boldsymbol{u}$ that works. We could find this using the pseudoinverse (`pinv` in Matlab).

$$\boldsymbol{u}_\text{o} = -\boldsymbol{B}^\dagger \boldsymbol{A}\boldsymbol{x}_\text{o}$$

- $m < n$ ($\boldsymbol{B}$ is skinny) or $\operatorname{rank} \boldsymbol{B} < n$. In this case there are fewer actuators than states. There is no guarantee that we will be able to find a $\boldsymbol{u}$ to achieve our desired $\boldsymbol{x}_\text{o}$. Where it is possible to find the desired input, it can again be done using the pseudo-inverse.

The matrix pseudoinverse for skinny $\boldsymbol{B}$

Recall that for our system

$$-\boldsymbol{A}\boldsymbol{x}_\text{o} = \boldsymbol{B}\boldsymbol{u}_\text{o}$$

Now, we would like to isolate $\boldsymbol{u}_\text{o}$, but $\boldsymbol{B}$ is not invertible in this case. However, if $\boldsymbol{B}$ has independent columns (which is normal because we don't typically have redundant actuators), then $\boldsymbol{B}^\mathsf{T}\boldsymbol{B}$ will be both square and invertible. We will premultuply both sides by $\boldsymbol{B}^\mathsf{T}$ and proceed.

$$\begin{aligned} -\boldsymbol{B}^\mathsf{T}\boldsymbol{A}\boldsymbol{x}_\text{o} &= \boldsymbol{B}^\mathsf{T}\boldsymbol{B}\boldsymbol{u}_\text{o} \\ -\left(\boldsymbol{B}^\mathsf{T}\boldsymbol{B}\right)^{-1}\boldsymbol{B}^\mathsf{T}\boldsymbol{A}\boldsymbol{x}_\text{o} &= \boldsymbol{u}_\text{o} \\ -\boldsymbol{B}^\dagger\boldsymbol{A}\boldsymbol{x}_\text{o} &= \boldsymbol{u}_\text{o} \\ \text{where, } \boldsymbol{B}^\dagger &:= \left(\boldsymbol{B}^\mathsf{T}\boldsymbol{B}\right)^{-1}\boldsymbol{B}^\mathsf{T} \end{aligned}$$

The matrix pseudoinverse for fat $\boldsymbol{B}$

In the rare case of a system with fat $\boldsymbol{B}$ we again have

$$-\boldsymbol{A}\boldsymbol{x}_{\text{o}} = \boldsymbol{B}\boldsymbol{u}_{\text{o}}$$

We would again like to "invert" $\boldsymbol{B}$ to isolate $\boldsymbol{u}_{\text{o}}$. However, $\boldsymbol{B}$ no longer has independent columns, and $\boldsymbol{B}^\mathsf{T}\boldsymbol{B}$ would be the wrong size to mutiply with $\boldsymbol{A}$. However, this time $\boldsymbol{B}\boldsymbol{B}^\mathsf{T}$ will be square, invertible and compatible with $\boldsymbol{A}$.

$$\begin{aligned}
-\boldsymbol{A}\boldsymbol{B}^\mathsf{T}\boldsymbol{x}_{\text{o}} &= \boldsymbol{B}\boldsymbol{B}^\mathsf{T}\boldsymbol{u}_{\text{o}} \\
-\boldsymbol{A}\boldsymbol{B}^\mathsf{T}\left(\boldsymbol{B}\boldsymbol{B}^\mathsf{T}\right)^{-1}\boldsymbol{x}_{\text{o}} &= \boldsymbol{u}_{\text{o}} \\
-\boldsymbol{A}\boldsymbol{B}^\dagger\boldsymbol{x}_{\text{o}} &= \boldsymbol{u}_{\text{o}} \\
\text{where, } \boldsymbol{B}^\dagger &:= \boldsymbol{B}^\mathsf{T}\left(\boldsymbol{B}\boldsymbol{B}^\mathsf{T}\right)^{-1}
\end{aligned}$$

This is also implemented as `pinv` in Matlab, so you need not remember either formula for the pseudoinverse.

Let's consider an example with model $\boldsymbol{A} = \begin{bmatrix} 0 & 0.2 \\ -0.2 & 0 \end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. The blue arrows show the

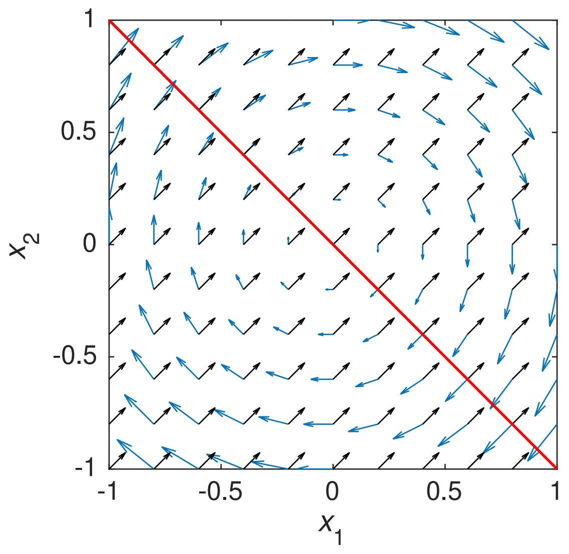


Figure 1.1: The blue vectors show that action of $\boldsymbol{A}\boldsymbol{x}$, and the black vectors indicate the direction of $\boldsymbol{B}\boldsymbol{u}$. The input can only balance the effect of $\boldsymbol{A}\boldsymbol{x}$ for locations along the red line.

contribution to $\dot{\boldsymbol{x}}$ from $\boldsymbol{A}\boldsymbol{x}$. The black arrows show the *direction* in which $\boldsymbol{B}\boldsymbol{u}$ pushes $\boldsymbol{x}$. Remember that it can be scaled. The two can only balance for values of $\boldsymbol{x}$ on the red line.

The subspace of accessible operating points is given by

$$\begin{aligned}
\mathcal{E}_{\boldsymbol{x}} &= \mathcal{R}(\boldsymbol{A}^{-1}\boldsymbol{B}) \\
&= \mathcal{R}\begin{bmatrix} -5 \\ 5 \end{bmatrix}
\end{aligned}$$

That is, the possible operating points are all multiples of the vector $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$. This is consistent with the red line in figure 1.1.

Interestingly, we will see later that we can drive this system to reach an arbitrary value of $\boldsymbol{x}$, but it can't *stay* at that arbitrary point.

Now let's consider $\boldsymbol{A} = \begin{bmatrix} 0 & 0.2 \\ -0.2 & 0 \end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

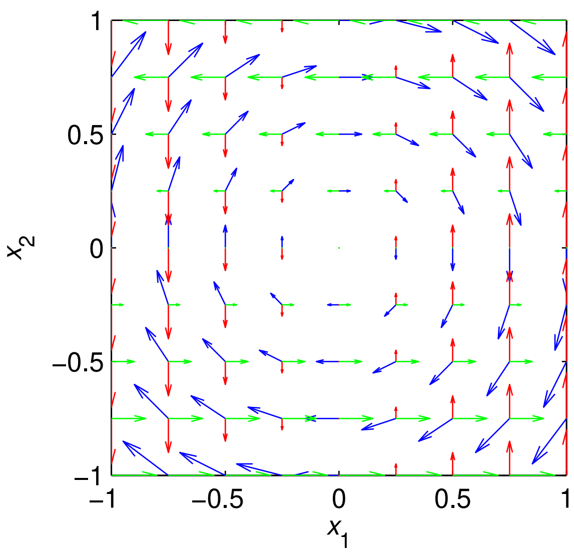


Figure 1.2: Illustration of the $\boldsymbol{u}_o$ vector calculated for an invertible $\boldsymbol{B}$. the blue vectors show that action of $\boldsymbol{Ax}$, and the red and green vectors are actions of the first and second inputs respectively. That is, the red vector is b_1u_1 and the green vector is b_2u_2.

$$\begin{aligned}\dot{\boldsymbol{x}} &= \boldsymbol{Ax} + \boldsymbol{Bu} \\ &= \boldsymbol{Ax} + b_1u_1 + b_2u_2\end{aligned}$$

u_1 acts along $b_1 = \begin{bmatrix}0\\1\end{bmatrix}$. u_2 acts along $b_2 = \begin{bmatrix}1\\0\end{bmatrix}$. The diagram shows the components of $\boldsymbol{u} = -\boldsymbol{B}^{-1}\boldsymbol{Ax}$ at each $\boldsymbol{x}$.

We can again calculate the space of possible operating points.

$$\begin{aligned}\mathcal{E}_{\boldsymbol{x}} &= \mathcal{R}(\boldsymbol{A}^{-1}\boldsymbol{B}) \\ &= \mathcal{R}\begin{bmatrix}0.2 & 0\\0 & -0.2\end{bmatrix} \qquad\qquad = \mathcal{R}\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\end{aligned}$$

We can see that the columns of this matrix span $\mathbb{R}^2$, so we can place the operating point anywhere.

Finally, consider $\boldsymbol{A} = \begin{bmatrix}0 & 0.2\\-0.2 & 0\end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix}0 & 1 & 1\\1 & 0 & 1\end{bmatrix}$.

$$\begin{aligned}\dot{\boldsymbol{x}} &= \boldsymbol{Ax} + \boldsymbol{Bu} \\ &= \boldsymbol{Ax} + b_1u_1 + b_2u_2 + b_3u_3\end{aligned}$$

u_1 acts along $b_1 = \begin{bmatrix}0\\1\end{bmatrix}$. u_2 acts along $b_2 = \begin{bmatrix}1\\0\end{bmatrix}$. u_3 acts along $b_3 = \begin{bmatrix}1\\1\end{bmatrix}$. The diagram shows the elements of $\boldsymbol{u} = -\boldsymbol{B}^{\dagger}\boldsymbol{Ax}$ at each $\boldsymbol{x}$.

Consider $\boldsymbol{A} = \begin{bmatrix}0 & 0.2\\-0.2 & 0\end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix}1\\1\end{bmatrix}$ as before. In this case the pseudoinverse gives us the control signal that does the best it can to counteract the changes due to $\boldsymbol{Ax}$. That is, it yields a $\boldsymbol{u}$ that counteracts $\boldsymbol{Ax}$ along the direction in which $\boldsymbol{u}$ can act ($[1\ 1]^{\mathsf{T}}$ in this case).

For a discrete time system we have

$$\boldsymbol{x}(t+1) = \boldsymbol{Ax}(t) + \boldsymbol{Bu}(t)$$

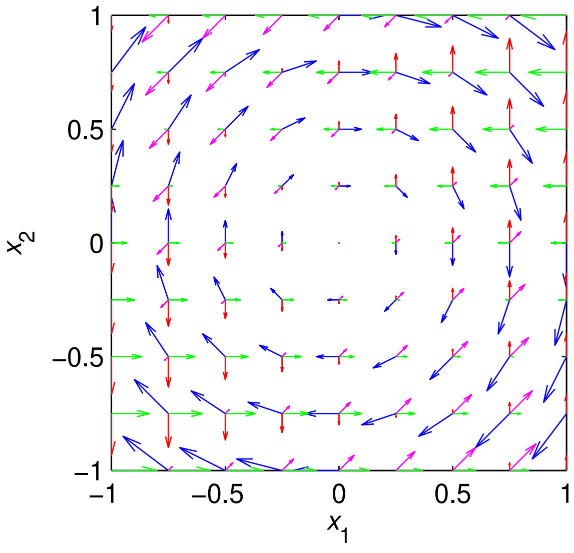


Figure 1.3: An illustration of the balance of changes calculated for a fat $\boldsymbol{B}$ using the pseudoinverse. This calculation yields the least norm $\boldsymbol{u}$ that will balance the motion of the autonomous system. The blue vectors indicate the effects of $\boldsymbol{Ax}$, red is b_1u_1, green is b_2u_2 and magenta is b_3u_3.

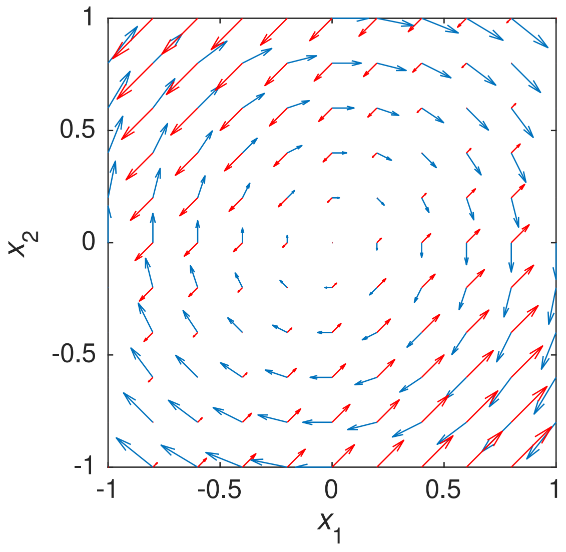


Figure 1.4: Calculation of a $\boldsymbol{u}$ using the pseudoinverse when $\boldsymbol{B}$ is skinny. Blue vectors show the action of $\boldsymbol{Ax}$ while the red vectors are the 'best' possible values for $\boldsymbol{Bu}$.

But at steady state we have $\boldsymbol{x}(t+1) = \boldsymbol{x}(t) = \boldsymbol{x}_\mathrm{o}$. So,

$$\begin{aligned}
\boldsymbol{x}_\mathrm{o} &= \boldsymbol{A}\boldsymbol{x}_\mathrm{o} + \boldsymbol{B}\boldsymbol{u}_\mathrm{o} \\
(\boldsymbol{I}-\boldsymbol{A})\boldsymbol{x}_\mathrm{o} &= \boldsymbol{B}\boldsymbol{u}_\mathrm{o} \\
\text{So,}\quad \mathcal{E}_{\boldsymbol{x}} &= \left\{(\boldsymbol{I}-\boldsymbol{A})^{-1}\boldsymbol{B}\boldsymbol{u}_\mathrm{o},\ \boldsymbol{u}_\mathrm{o} \in \mathbb{R}^m\right\} \\
&= \mathcal{R}\left((\boldsymbol{I}-\boldsymbol{A})^{-1}\boldsymbol{B}\right) \\
\text{and if it exists, } \boldsymbol{u}_\mathrm{o} &= \boldsymbol{B}^{-1}(\boldsymbol{I}-\boldsymbol{A})\boldsymbol{x}_\mathrm{o} \text{ or } \boldsymbol{B}^\dagger(\boldsymbol{I}-\boldsymbol{A})\boldsymbol{x}_\mathrm{o}
\end{aligned}$$

We have the same considerations about invertibility of $\boldsymbol{B}$ as in the continuous time case.

The tests above allow you to determine whether a given system can be placed at equilibrium at some desired setpoint. This calculation should be done as part of designing your suite of actuators, which leads to the $\boldsymbol{B}$ matrix. That is, don't design a system and *then* find whether you can achieve your desired operating points; instead *choose* your actuator configuration (or equivalently your $\boldsymbol{B}$ matrix) to make sure that the system behaves the way that you require.

We have previously discussed controlling nonlinear systems by using a linearising model that holds around the desired operating point. For these systems it is not generally true that the model at $\boldsymbol{x} = 0$ is the same as that at the desired operating point (or anywhere else for that matter). For nonlinear systems you need to use the complete nonlinear system equations to find an appropriate $\boldsymbol{u}_\mathrm{o}$. To complicate matters, depending on the particular nonlinearities that the system exhibits you may not end up at your desired $\boldsymbol{x}_\mathrm{o}$ if you apply $\boldsymbol{u}_\mathrm{o}$. You may need to carry the system along a particular trajectory to reach $\boldsymbol{x}_\mathrm{o}$, after which time your linearised controller can take over. For nonlinear systems the notion of an equilibrium subspace doesn't make any sense, so don't try to find that using the equations given above.

1.2 Controllability

Part II Controllability

Informally, we say that a system is uncontrollable when the inputs have no effect on one or more of the state variables. That is, the system is uncontrollable because the inputs are decoupled from at least one state variables. Equivalently, a system is uncontrollable if we can't independently drive the different state variables to arbitrary locations (more on this shortly). As there is no guarantee that it is possible to control a given system, we would like a mathematical test that tells us when a system is uncontrollable. You should *always* check whether a system is controllable before attempting to design a controller. In some circumstances we might be able to get away with using an uncontrollable system, but often this is a sign that we need to add or move actuators.

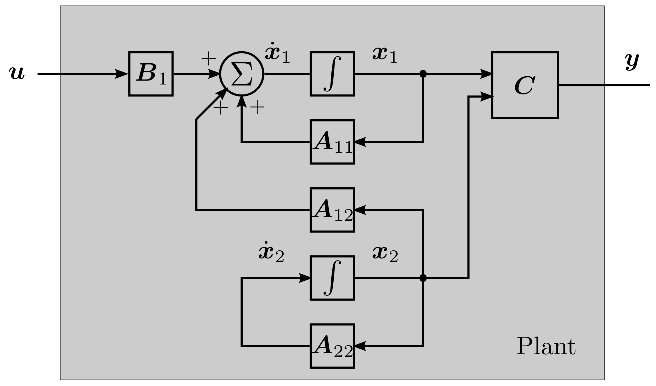


Figure 1.5: Internal structure of an uncontrollable system. The subset of state variables x_2 is not controllable from the inputs.

Here $\boldsymbol{A} := \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ 0 & \boldsymbol{A}_{22} \end{bmatrix}$, $\boldsymbol{B} := \begin{bmatrix} \boldsymbol{B}_1 \\ 0 \end{bmatrix}$, $\boldsymbol{x} := \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}$. Notice that $\boldsymbol{u}$ cannot affect $\boldsymbol{x}_2$, so the plant is uncontrollable.

We might expect that uncontrollable states should play no role in the transfer matrix of a system. Because no input can influence them, the application of an input should have no effect on the output that is mediated by those states. That is, $\boldsymbol{u}$ cannot change $\boldsymbol{x}_2$ in the diagram, so there is no way that a change at $\boldsymbol{u}$ can propagate through to $\boldsymbol{y}$ through the uncontrollable states.

There are several ways to consider system controllability, but for continuous-time linear systems (and most practical discrete time systems) they are happily equivalent. We will consider a system controllable if we can drive the system from an initial state $\boldsymbol{x}(0) = 0$ to an arbitrary state $\boldsymbol{x}(\tau)$. Strictly speaking this is known as *reachability* whereas controllability is the ability to drive an arbitrary $\boldsymbol{x}(0)$ to $\boldsymbol{x}(\tau) = 0$. All reachable systems are controllable (and vice versa) for CT systems and also for DT systems (excepting pathological cases like $\boldsymbol{x}(t+1) = 0$). Note that controllability doesn't place any restrictions on the behaviour of the system outputs. It is purely about driving the system's internal state to an arbitrary point.

1.2.1 Controllability of discrete-time systems

Consider a discrete time SISO system described by $\boldsymbol{x}(t+1) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}u(t)$ with $\boldsymbol{x}(0) = 0$. We can write expressions for the state of the system at various times.

$$\begin{aligned}
\boldsymbol{x}(1) &= \boldsymbol{B}u(0) \\
\boldsymbol{x}(2) &= \boldsymbol{A}\boldsymbol{x}(1) + \boldsymbol{B}u(1) \\
&= \boldsymbol{A}\big(\boldsymbol{B}u(0)\big) + \boldsymbol{B}u(1) \\
&= \boldsymbol{A}\boldsymbol{B}u(0) + \boldsymbol{B}u(1) \\
\boldsymbol{x}(3) &= \boldsymbol{A}\boldsymbol{x}(2) + \boldsymbol{B}u(2) \\
&= \boldsymbol{A}\big(\boldsymbol{A}\boldsymbol{B}u(0)\big) + \boldsymbol{A}\boldsymbol{B}u(1) + \boldsymbol{B}u(2) \\
&= \boldsymbol{A}^2\boldsymbol{B}u(0) + \boldsymbol{A}\boldsymbol{B}u(1) + \boldsymbol{B}u(2) \\
&\vdots \\
\boldsymbol{x}(k) &= \boldsymbol{A}^{k-1}\boldsymbol{B}u(0) + \boldsymbol{A}^{k-2}\boldsymbol{B}u(1) + \boldsymbol{A}^{k-3}\boldsymbol{B}u(2) + \cdots + \boldsymbol{B}u(k-1)
\end{aligned}$$

$$\begin{bmatrix} \boldsymbol{x}(1) \\ \boldsymbol{x}(2) \\ \boldsymbol{x}(3) \\ \vdots \\ \boldsymbol{x}(k) \end{bmatrix} = \begin{bmatrix} & & & & \boldsymbol{B} \\ & & & \boldsymbol{B} & \boldsymbol{A}\boldsymbol{B} \\ & & \boldsymbol{B} & \boldsymbol{A}\boldsymbol{B} & \boldsymbol{A}^2\boldsymbol{B} \\ & \iddots & \iddots & \iddots & \vdots \\ \boldsymbol{B} & \boldsymbol{A}\boldsymbol{B} & \boldsymbol{A}^2\boldsymbol{B} & \cdots & \boldsymbol{A}^{k-1}\boldsymbol{B} \end{bmatrix} \begin{bmatrix} u(k-1) \\ u(k-2) \\ u(k-3) \\ \vdots \\ u(0) \end{bmatrix}$$

If we concentrate on the expression for $\boldsymbol{x}(k)$ we see

$$\boldsymbol{x}(k) = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{A}\boldsymbol{B} & \boldsymbol{A}^2\boldsymbol{B} & \cdots & \boldsymbol{A}^{k-1}\boldsymbol{B} \end{bmatrix} \begin{bmatrix} u(k-1) \\ u(k-2) \\ u(k-3) \\ \vdots \\ u(0) \end{bmatrix} \equiv \boldsymbol{\mathcal{M}}_{\mathrm{c}}(k) \begin{bmatrix} u(k-1) \\ u(k-2) \\ u(k-3) \\ \vdots \\ u(0) \end{bmatrix}$$

where $\boldsymbol{\mathcal{M}}_{\mathrm{c}}(k)$, closely related to the *controllability matrix*, is defined as $\begin{bmatrix} \boldsymbol{B} & \boldsymbol{A}\boldsymbol{B} & \boldsymbol{A}^2\boldsymbol{B} & \cdots & \boldsymbol{A}^{k-1}\boldsymbol{B} \end{bmatrix}$.

If we want to choose a sequence of u values to achieve a certain $\boldsymbol{x}(k)$, then we could invert this expression. Note that $\boldsymbol{\mathcal{M}}_{\mathrm{c}}(k)$ in a SISO system can only be square, and (possibly) invertible after $n-1$

time steps, so (in general) we need to wait that long for this to work.

$$\begin{bmatrix} u(n-1) \\ u(n-2) \\ \vdots \\ u(1) \\ u(0) \end{bmatrix} = \boldsymbol{\mathcal{M}}_\mathrm{c}^{-1}\boldsymbol{x}(n)$$

That is, iff $\boldsymbol{\mathcal{M}}_\mathrm{c}$ is invertible then we can find $u(t)$ to bring about our desired $\boldsymbol{x}(n)$.

Getting there faster

Sometimes it is possible to reach the desired state faster than the $n-1$ time steps mentioned above. This depends on whether the target location lies within the range of $\begin{bmatrix}\boldsymbol{B} & \boldsymbol{AB} & \cdots & \boldsymbol{A}^{k-1}\boldsymbol{B}\end{bmatrix}$ for $k < n$. In such cases we can use the pseudoinverse to find the required control signal as long as the columns of the controllability matrix are linearly independent.

$$\boldsymbol{\mathcal{M}}_\mathrm{c}(k)^\dagger \boldsymbol{x}(k) = \begin{bmatrix} u(k-1) \\ u(k-2) \\ u(k-3) \\ \vdots \\ u(0) \end{bmatrix}$$

To use this approach you would need to have an understanding of how the range of the controllability matrix grows in dimension as time passes. The time step at which it encompasses the desired target would be the k used in the pseudo-inverse calculation.

We can extend this treatment for the case where the system does not start at $\boldsymbol{x}(0) = 0$, but at some general other state $\boldsymbol{x}_\mathrm{o}$. In this case we need to account for the fact that the system will carry the initial $\boldsymbol{x}_\mathrm{o}$ through to some other state $\boldsymbol{\Phi}(k)\boldsymbol{x}_\mathrm{o}$ after k timesteps. So, combining these two, we have

$$\boldsymbol{x}(k) = \boldsymbol{\Phi}(k)\boldsymbol{x}_\mathrm{o} + \boldsymbol{\mathcal{M}}_\mathrm{c}(k)\begin{bmatrix} u(k-1) \\ u(k-2) \\ u(k-3) \\ \vdots \\ u(0) \end{bmatrix}$$

Consider a MIMO discrete-time LTI system described by

$$\begin{aligned} \boldsymbol{x}(t+1) &= \boldsymbol{Ax}(t) + \boldsymbol{Bu}(t) \\ \boldsymbol{x}(0) &= 0 \\ \boldsymbol{x} \in \mathbb{R}^n&, \, t \in \mathbb{Z}_+ \end{aligned}$$

We want to determine the complete set of possible values for $\boldsymbol{x}(t)$, as a function of t. If $\boldsymbol{x}(t)$ can (eventually) take any value in $\mathbb{R}^n$ then the system is controllable. We can write an expression for $\boldsymbol{x}(1)$:

$$\begin{aligned} \boldsymbol{x}(1) &= \boldsymbol{Ax}(0) + \boldsymbol{Bu}(0) \\ &= \boldsymbol{Bu}(0) \end{aligned}$$

At $t = 1$ the system can have any value that is in the range of $\boldsymbol{B}$.

$\boldsymbol{x}(1) \in \mathcal{R}\boldsymbol{B}$, so if there are fewer inputs than system states then we will not be able to reach every possible state in $\mathbb{R}^n$ in one time step. If we do have enough inputs then we can reach every state only if $\boldsymbol{B}$ is full rank. Recall the example for $\boldsymbol{x} \in \mathbb{R}^2$ with $\boldsymbol{B} = \begin{bmatrix} 1 & -2 \\ 2 & -4 \end{bmatrix}$. If we generate random samples for $\boldsymbol{u}(0)$ we find that the effect on $\boldsymbol{x}(1)$ is one dimensional, namely $\mathcal{R}\boldsymbol{B}$. This ought not surprise us, as

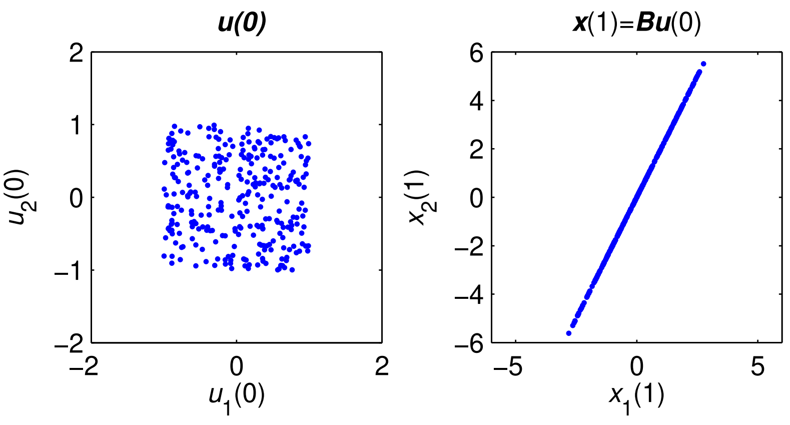


Figure 1.6: Effect of control with a rank deficient $\boldsymbol{B}$ after one time step. The left subfigure shows a random sample of 300 possible $\boldsymbol{u}(0)$ drawn from a uniform distribution between -1 and 1. The right subfigure shows how the various $\boldsymbol{u}$ values are mapped into $\boldsymbol{x}(1)$ by the action of $\boldsymbol{B}$. In particular notice that the result spans a one dimensional subspace of $\mathbb{R}^2$.

$\boldsymbol{B}$ is clearly not full rank because its columns are not independent. Figure 1.6 shows how the random samples for $\boldsymbol{u}(0)$ map to $\boldsymbol{x}(1)$.

After an additional time step we have the following:

$$\begin{aligned} \boldsymbol{x}(2) &= \boldsymbol{A}\boldsymbol{x}(1) + \boldsymbol{B}\boldsymbol{u}(1) \\ &= \boldsymbol{A}\boldsymbol{B}\boldsymbol{u}(0) + \boldsymbol{B}\boldsymbol{u}(1) \end{aligned}$$

We chose a $\boldsymbol{u}(0)$, which resulted in $\boldsymbol{x}(1) \in \mathcal{R}\boldsymbol{B}$. The effect of $\boldsymbol{u}(0)$ is now in $\mathcal{R}(\boldsymbol{AB})$. In addition we have a $\boldsymbol{u}(1)$, the effect of which is in $\mathcal{R}\boldsymbol{B}$. That is,

$$\begin{aligned} \boldsymbol{x}(2) &\in \mathcal{R}(\boldsymbol{B}) \oplus \mathcal{R}(\boldsymbol{AB}) \\ &\in \mathcal{R}\begin{bmatrix}\boldsymbol{B} & \boldsymbol{AB}\end{bmatrix} \end{aligned}$$

That is, $\boldsymbol{x}(2)$ is in a space spanned by the columns of $\boldsymbol{B}$ plus the columns of $\boldsymbol{AB}$. This is equivalent to saying that it is spanned by a matrix made by concatenating $\boldsymbol{B}$ and $\boldsymbol{AB}$.

Returning to our example in $\mathbb{R}^2$, at $t = 2$ we can see in figure 1.7 that the effect of $u(0)$ has now been swept along by the action of $\boldsymbol{A} = \begin{bmatrix} 0.6\cos(\pi/3) & -0.6\sin(\pi/3) \\ 0.6\sin(\pi/3) & 0.6\cos(\pi/3) \end{bmatrix}$. We can see that this is still a one dimensional subspace, $\mathcal{R}(\boldsymbol{AB})$. The effect of the random samples for $\boldsymbol{u}(1)$ spans the same subspace $\mathcal{R}\boldsymbol{B}$ as before. However, the sum of a point from each now spans $\mathbb{R}^2$.

$$\begin{aligned} \boldsymbol{x}(3) &= \boldsymbol{A}\boldsymbol{x}(2) + \boldsymbol{B}\boldsymbol{u}(2) \\ &= \boldsymbol{A}\big(\boldsymbol{AB}\boldsymbol{u}(0) + \boldsymbol{B}\boldsymbol{u}(1)\big) + \boldsymbol{B}\boldsymbol{u}(2) \\ &= \boldsymbol{A}^2\boldsymbol{B}\boldsymbol{u}(0) + \boldsymbol{AB}\boldsymbol{u}(1) + \boldsymbol{B}\boldsymbol{u}(2) \\ \text{So,} \quad \boldsymbol{x}(3) &\in \mathcal{R}\begin{bmatrix}\boldsymbol{B} & \boldsymbol{AB} & \boldsymbol{A}^2\boldsymbol{B}\end{bmatrix} \end{aligned}$$

We can write a general equation for the state at time t

$$\begin{aligned} \boldsymbol{x}(t) = \boldsymbol{A}^{t-1}\boldsymbol{B}\boldsymbol{u}(0) + \boldsymbol{A}^{t-2}\boldsymbol{B}\boldsymbol{u}(1) + \boldsymbol{A}^{t-3}\boldsymbol{B}\boldsymbol{u}(2) \\ + \cdots + \boldsymbol{AB}\boldsymbol{u}(t-2) + \boldsymbol{B}\boldsymbol{u}(t-1)\,, \end{aligned}$$

for which the reachable subspace is

$$\mathcal{R}\begin{bmatrix}\boldsymbol{B} & \boldsymbol{AB} & \boldsymbol{A}^2\boldsymbol{B} & \cdots & \boldsymbol{A}^{t-1}\boldsymbol{B}\end{bmatrix}\,.$$

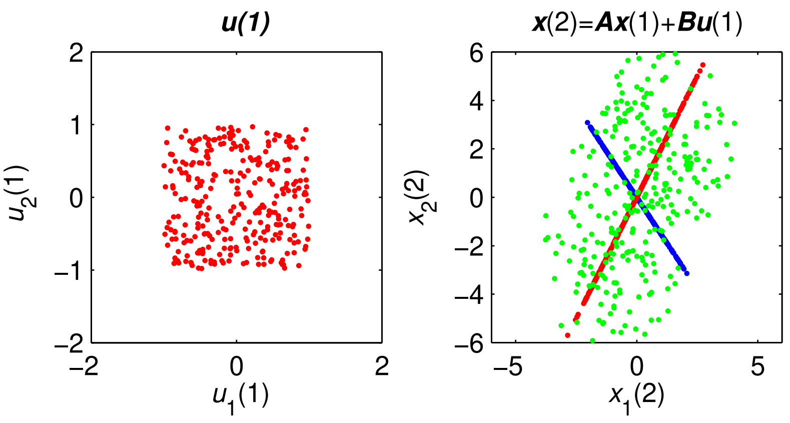


Figure 1.7: The left subfigure shows 300 random samples for $\boldsymbol{u}(1) \sim \mathcal{U}(-1,1)$. The right subfigure shows the effect of $\boldsymbol{u}(1)$ on $\boldsymbol{x}(2)$ in red and the effect of $\boldsymbol{u}(0)$ and blue. The green points are produced by summing one of the red points with one of the blue. That is, the green points show the $\boldsymbol{x}(2)$ values reached by using a random $\boldsymbol{u}(0)$ followed by another random $\boldsymbol{u}(1)$.

It would appear that the range of the matrix

$$\mathcal{R}\begin{bmatrix}\boldsymbol{B} & \boldsymbol{AB} & \boldsymbol{A}^2\boldsymbol{B} & \cdots & \boldsymbol{A}^{t-1}\boldsymbol{B}\end{bmatrix}$$

continues to grow as t grows. However, the Cayley-Hamilton theorem allows us to write powers of a matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ at or beyond the n-th as a linear combination of lower powers of $\boldsymbol{M}$. That is, from $t = n$ the $\boldsymbol{A}^t$ matrices are linearly dependent on lower powers of $\boldsymbol{A}$. That means that the columns of these higher powers of the $\boldsymbol{A}$ matrix *cannot* be independent of the columns of the lower powers. Thus, the range of the matrix cannot increase after $t = n - 1$.

We define the controllability matrix as

$$\boldsymbol{\mathcal{M}}_\mathrm{c} := \begin{bmatrix}\boldsymbol{B} & \boldsymbol{AB} & \boldsymbol{A}^2\boldsymbol{B}\cdots & \boldsymbol{A}^{n-1}\boldsymbol{B}\end{bmatrix}.$$

$\mathcal{R}(\boldsymbol{\mathcal{M}}_\mathrm{c})$ is the entire set of possible $\boldsymbol{x}$ that the system can eventually be driven to with *some* choice of $\boldsymbol{u}$. If the system is controllable then we can drive $\boldsymbol{x}$ to an arbitrary point in $\mathbb{R}^n$. That is, we require $\mathcal{R}(\boldsymbol{\mathcal{M}}_\mathrm{c}) = \mathbb{R}^n$. As $\boldsymbol{\mathcal{M}}_\mathrm{c}$ has n rows, a system is controllable iff $\boldsymbol{\mathcal{M}}_\mathrm{c}$ is full rank. Recall for completeness, that we also define

$$\boldsymbol{\mathcal{M}}_\mathrm{c}(t) := \begin{bmatrix}\boldsymbol{B} & \boldsymbol{AB} & \boldsymbol{A}^2\boldsymbol{B}\cdots & \boldsymbol{A}^{t-1}\boldsymbol{B}\end{bmatrix} \text{ for } t < n.$$

One test for the rank of a matrix is to find the largest block inside the matrix that has non-zero determinant. The size of that submatrix is the rank of the matrix. A SISO system will always have a square controllability matrix, so one way to check its controllability is to find the determinant of the controllability matrix. Any square matrix that is not full rank will have a determinant equal to zero. That is, for controllability of a system with square $\boldsymbol{\mathcal{M}}_\mathrm{c}$, check that

$$\boxed{|\boldsymbol{\mathcal{M}}_\mathrm{c}| \neq 0}$$

In the case of a non-square controllability matrix you could check the determinant of $\boldsymbol{\mathcal{M}}_\mathrm{c}\boldsymbol{\mathcal{M}}_\mathrm{c}^\mathsf{T}$, since the rank of a matrix $\boldsymbol{A}$ is the same as $\boldsymbol{AA}^\mathsf{T}$. An even easier test is to use Matlab's `rank(ctrb(A, B))`.

1.2.2 Reachability in time t

It is sometimes useful to consider the subspace that can be reached (or controlled) in some given t. That is, we can define the reachable subspace in time t as

$$\begin{aligned}\mathcal{R}_t &:= \mathcal{R}\left(\begin{bmatrix}\boldsymbol{B} & \boldsymbol{AB} & \cdots & \boldsymbol{A}^{t-1}\boldsymbol{B}\end{bmatrix}\right)\\ &= \mathcal{R}\boldsymbol{\mathcal{M}}_\mathrm{c}(t)\end{aligned}$$

Notice that the reachable/controllable subspace can never shrink as time increases, though eventually it stops growing.

$$\mathcal{R}_1 \subseteq \mathcal{R}_2 \subseteq \cdots \subseteq \mathcal{R}_{n-1} = \mathcal{R}\boldsymbol{\mathcal{M}}_{\mathrm{c}}$$

1.2.3 Controllable and Equilibrium Subspaces

The subspace of equilibrium points can be smaller than the reachable (or controllable) subspace. That is, it might be possible to reach parts of the state space, but not stay there. However, all parts of the subspace of possible equilibria must be contained within the controllable subspace.

$$\mathcal{E}_{\boldsymbol{x}} \subseteq \mathcal{R}(\boldsymbol{\mathcal{M}}_{\mathrm{c}})$$

This is just stating that if we are able to stay at some point in state space, then we must be able to get there in the first place.

1.2.4 Controllability in practice

The controllability test tells us that every state variable can be influenced by the inputs to *some* extent, even if the effect is tiny. It does not guarantee that it is practical to control a system in that way. For example, if the coupling between the inputs and a state variable is weak then it may require an unrealistically large control effort to have the system show any discernable response. You should regard a controllability test as a necessary, but not sufficient test of whether you can practically control a system. When you design a real system make sure that you check the results in simulation to see whether the control signals are sensible. That is, think of the rank of the controllability matrix as revealing when a system is *uncontrollable*, but not so much about whether it is controllable.

1.2.5 Stabilisability

A system is said to be *stabilisable* iff all its uncontrollable modes are stable. In a stabilisable system, any modes that we cannot control will naturally decay to zero anyway. That means that eventually the system state will be driven to zero, so we can stabilise the system at an operating point. This may or may not imply that the system is practical, depending on whether the natural behaviour of the uncontrolled modes is acceptable. As a result, all is not lost if a system is uncontrollable. Examine the system modes carefully to see whether the system will perform adequately anyway. Stabilisability is a weaker test than controllability; all controllable systems are stabilisable by definition, but the converse is not true.

Stabilisable systems are quite common in practical situations. For example, mechanical systems often have high frequency vibrational modes that we choose to ignore. We might design a controller for a robot arm that is slightly flexible. If the motion of the arm (ie the joints) is controllable then we can neglect the flexibility, as long as the resulting induces high frequency vibration dies away rapidly enough. Using simplified models that ignore unimportant (usually high frequency) modes in this way is very common. This is only a legitimate approach if the system is stabilisable, because otherwise the apparently unimportant modes can grow to dominate the system response.

There are several ways to identify uncontrollable modes. The easiest is perhaps to put the system into modal canonical form. Inspection of the $\boldsymbol{B}$ matrix will then make it clear when the inputs cannot affect the states. That is, the presence of a 0 row in $\boldsymbol{B}$ suggests that there is an uncontrollable mode (though be careful if there are non-trivial Jordan blocks or second order systems). If there are uncontrollable state variables, their associated eigenvalues will reveal whether the system is stabilisable.

1.2.6 Controllability of continuous-time systems

For a continuous time system with $\boldsymbol{x}(0) = 0$ we have

$$\begin{aligned}\boldsymbol{x}(t) &= \int_0^t \boldsymbol{\Phi}(\tau)\boldsymbol{B}\boldsymbol{u}(t-\tau)\,\mathrm{d}\tau \\ &= \int_0^t \mathrm{e}^{\tau\boldsymbol{A}}\boldsymbol{B}\boldsymbol{u}(t-\tau)\,\mathrm{d}\tau\end{aligned}$$

Again, we would like to find the subset of possible $\boldsymbol{x}(t)$ values for an arbitrary choice of $\boldsymbol{u}(t) \in [0,t] \to \mathbb{R}^m$. That is $\boldsymbol{u}$ is an arbitrary function between times 0 and t. Let's expand the matrix exponential into its series form

$$\begin{aligned}\boldsymbol{x}(t) &= \int_0^t \left(\boldsymbol{I} + \tau\boldsymbol{A} + \frac{1}{2}\tau^2\boldsymbol{A}^2 + \cdots\right)\boldsymbol{B}\boldsymbol{u}(t-\tau)\,\mathrm{d}\tau \\ &= \int_0^t \left(\boldsymbol{B} + \tau\boldsymbol{A}\boldsymbol{B} + \frac{1}{2}\tau^2\boldsymbol{A}^2\boldsymbol{B} + \cdots\right)\boldsymbol{u}(t-\tau)\,\mathrm{d}\tau\end{aligned}$$

$$\begin{aligned}\boldsymbol{x}(t) &= \int_0^t \left(\boldsymbol{B} + \tau\boldsymbol{A}\boldsymbol{B} + \frac{1}{2}\tau^2\boldsymbol{A}^2\boldsymbol{B} + \cdots\right)\boldsymbol{u}(t-\tau)\,\mathrm{d}\tau \\ &= \boldsymbol{B}\int_0^t \boldsymbol{u}(t-\tau)\,\mathrm{d}\tau + \boldsymbol{A}\boldsymbol{B}\int_0^t \tau\boldsymbol{u}(t-\tau)\,\mathrm{d}\tau \\ &\quad + \boldsymbol{A}^2\boldsymbol{B}\int_0^t \frac{\tau^2}{2}\boldsymbol{u}(t-\tau)\,\mathrm{d}\tau + \cdots\end{aligned}$$

We don't need to worry too much about the details of this integral. Each of the integrals will evaluate to a vector of constants, so the important point is that $\boldsymbol{x}(t)$ will be a linear combination of the columns of $\boldsymbol{A}$, $\boldsymbol{A}\boldsymbol{B}$, $\boldsymbol{A}^2\boldsymbol{B}$ etc up to $\boldsymbol{A}^{n-1}\boldsymbol{B}$. Again the reachable values of $\boldsymbol{x}(t)$ will be given by the range of the controllability matrix $\begin{bmatrix}\boldsymbol{B} & \boldsymbol{A}\boldsymbol{B} & \boldsymbol{A}^2\boldsymbol{B} & \cdots & \boldsymbol{A}^{n-1}\boldsymbol{B}\end{bmatrix}$.

There is an interesting difference between discrete and continuous time systems. Notice that a discrete time system potentially has to "wait" for its reachable subspace to grow before it can get to some desired point in state space. In the case of a continuous time system, $\boldsymbol{x}(t)$ is a linear combination of the rows of $\boldsymbol{A}^k\boldsymbol{B}$ for all k *immediately.* No matter how small we make t, we can reach any point in state space that will ever be reachable. However, in practice this may require very large input signals.

1.3 Popov-Belevitch-Hautus Controllability Test

The PBH test provides an alternate method for assessing the controllability (or stabilizability) of a system. We will just state the result without proof.

Theorem 1 (The PBH Controllability Test)**.** *A system is controllable if and only if*

$$\operatorname{rank}\begin{bmatrix}\boldsymbol{A} - \lambda\boldsymbol{I} & \boldsymbol{B}\end{bmatrix} = n \qquad \forall\lambda \in \mathbb{C}$$

In practice we rarely use the PBH test directly to assess controllability. However, the form of the test does provide us with additional insight into the ways in which a system might be uncontrollable.

We know that $\boldsymbol{A} - \lambda\boldsymbol{I}$ has rank n unless λ is an eigenvalue of $\boldsymbol{A}$. That is, for all non-eigenvalues of $\boldsymbol{A}$, the PBH test is satisfied. When λ is a unique eigenvalue of $\boldsymbol{A}$, then $\boldsymbol{A} - \lambda\boldsymbol{I}$ will drop rank and its independent columns will be the eigenvectors of $\boldsymbol{A}$ *without* the eigenvector corresponding to λ. It will also have a linearly dependent column or a column of zeros. As a result, to satisfy the PBH controllability test we require the columns of $\boldsymbol{B}$ to make up for the independent column that we have lost. That is, the columns of $\boldsymbol{B}$ must span (at least) the same space as the eigenvectors of $\boldsymbol{A}$. If an eigenvalue has geometric multiplicity k then $\boldsymbol{A} - \lambda\boldsymbol{I}$ will drop rank by k at the corresponding eigenvalue. That, is we immediately know that $\boldsymbol{B}$ must have (at least) as many independent columns as the eigenvalue of $\boldsymbol{A}$ having the highest multiplicity.

We can use the PBH test to identify which modes of a system are uncontrollable. If $\boldsymbol{A}$ has unique eigenvectors $\boldsymbol{v}_i$, then the mode corresponding to eigenvector $\boldsymbol{v}_i$ will be uncontrollable if $\boldsymbol{v}_i^\top\boldsymbol{B} = 0$. While this analysis can be extended to repeated eigenvalues, it gets a little messy, so we will omit it here.

The PBH test is extremely useful when designing the actuators for a system. It tells us how many actuators we need (as a minimum), as well as the directions in which the actuators need to push. This provides a more sensible route to actuator placement than trying different $\boldsymbol{B}$ matrices and checking the resulting controllability matrix. Note that the PBH test is still a binary test; systems are either controllable or they are not.

1.4 The Controllability Gramian

The *Controllabity Gramian* provides yet another method of assessing the controllability of a system. Unlike checking of the rank of the controllability matrix or the Popov-Belevitch-Hautus test, it allows us to quantitatively characterise the controllability of a system. That is, the controllability Gramian allows us to talk about how *difficult* it is to control a system in a certain direction. Note that we will again be a little loose in the discussion to follow. Technically the Gramian that we will discuss is the *Reachability Gramian*, but this is equivalent to the controllability Gramian for all continuous time and most discrete time systems that we will encounter.

Consider a continuous time system starting at state $\boldsymbol{x} = 0$. We wish to drive the system to state $\boldsymbol{x}_\mathrm{f}$ in time t_f by choosing some control signal $\boldsymbol{u}(t)$. We know that the final state of the system can be described by

$$\boldsymbol{x}_\mathrm{f} = \boldsymbol{x}(t_\mathrm{f}) = \int_0^{t_\mathrm{f}} \mathrm{e}^{(t_\mathrm{f}-t)\boldsymbol{A}} \boldsymbol{B}\boldsymbol{u}(t)\,\mathrm{d}t$$

This is awkward, because we would really like to somehow invert the equation to find $\boldsymbol{u}$ from $\boldsymbol{x}_\mathrm{f}$. The problem that we have is that $\mathrm{e}^{(t_\mathrm{f}-t)\boldsymbol{A}}\boldsymbol{B}$ is not generally square, so inversion seems impossible. However, we can be clever and choose $\boldsymbol{u}$ to be of the form $\boldsymbol{u}(t) = \boldsymbol{B}^\mathsf{T}\mathrm{e}^{(t_\mathrm{f}-t)\boldsymbol{A}^\mathsf{T}}\boldsymbol{z}$, where $\boldsymbol{z}$ is some unknown vector. Then,

$$\boldsymbol{x}(t_\mathrm{f}) = \int_0^{t_\mathrm{f}} \mathrm{e}^{(t_\mathrm{f}-t)\boldsymbol{A}} \boldsymbol{B}\boldsymbol{B}^\mathsf{T}\mathrm{e}^{(t_\mathrm{f}-t)\boldsymbol{A}^\mathsf{T}}\boldsymbol{z}\,\mathrm{d}t$$

The term inside the integral (excluding $\boldsymbol{z}$) is now guaranteed to be square, so we are on track to be able to invert the expression.

$$\begin{aligned}
\boldsymbol{x}(t_\mathrm{f}) &= -\left(\int_{-t_\mathrm{f}}^{0} \mathrm{e}^{\tau\boldsymbol{A}}\boldsymbol{B}\boldsymbol{B}^\mathsf{T}\mathrm{e}^{\tau\boldsymbol{A}^\mathsf{T}}\,\mathrm{d}\tau\right)\boldsymbol{z} \quad \text{where } \tau := t_\mathrm{f} - t\\
\implies \boldsymbol{z} &= \left(\int_0^{t_\mathrm{f}} \mathrm{e}^{\tau\boldsymbol{A}}\boldsymbol{B}\boldsymbol{B}^\mathsf{T}\mathrm{e}^{\tau\boldsymbol{A}^\mathsf{T}}\,\mathrm{d}\tau\right)^{-1}\boldsymbol{x}_\mathrm{f}\\
\text{So, } \boldsymbol{u} &= \boldsymbol{B}^\mathsf{T}\mathrm{e}^{(t_\mathrm{f}-t)\boldsymbol{A}^\mathsf{T}}\left(\int_0^{t_\mathrm{f}} \mathrm{e}^{\tau\boldsymbol{A}}\boldsymbol{B}\boldsymbol{B}^\mathsf{T}\mathrm{e}^{\tau\boldsymbol{A}^\mathsf{T}}\,\mathrm{d}\tau\right)^{-1}\boldsymbol{x}_\mathrm{f}
\end{aligned}$$

This will always have a solution (ie we *can* get to $\boldsymbol{x}_\mathrm{f}$ from 0), if the bracketed portion is invertible. The bracketed expression is the so-called *Controllability Gramian*, denoted $\boldsymbol{\mathcal{W}}_\mathrm{c}$.

$$\boldsymbol{\mathcal{W}}_\mathrm{c}(t) := \int_0^{t_\mathrm{f}} \mathrm{e}^{\tau\boldsymbol{A}}\boldsymbol{B}\boldsymbol{B}^\mathsf{T}\mathrm{e}^{\tau\boldsymbol{A}^\mathsf{T}}\,\mathrm{d}\tau$$

Minimum energy state transfer

It can be shown that the above solution for $\boldsymbol{u}$ is in fact, the minimum-energy input (in the sense of the input with the smallest $\int_0^{t_\mathrm{f}} \|\boldsymbol{u}(\tau)\|^2\,\mathrm{d}\tau$) that moves the system from $\boldsymbol{x} = 0$ to $\boldsymbol{x} = \boldsymbol{x}_\mathrm{f}$. We can generalise this result to find the minimum energy solution from moving from some initial state $\boldsymbol{x} = \boldsymbol{x}_\mathrm{i}$ to final state $\boldsymbol{x} = \boldsymbol{x}_\mathrm{f}$ in time t_f as

$$\boldsymbol{u}(t) = \boldsymbol{B}^\mathsf{T}\mathrm{e}^{(t_\mathrm{f}-t)\boldsymbol{A}^\mathsf{T}}\left(\int_0^{t_\mathrm{f}} \mathrm{e}^{\tau\boldsymbol{A}}\boldsymbol{B}\boldsymbol{B}^\mathsf{T}\mathrm{e}^{\tau\boldsymbol{A}^\mathsf{T}}\,\mathrm{d}\tau\right)^{-1}\left[\boldsymbol{x}_\mathrm{f} - \mathrm{e}^{t_\mathrm{f}\boldsymbol{A}}\boldsymbol{x}_\mathrm{i}\right]$$

Notice that the only difference here is that we have added the extra term $-\mathrm{e}^{t_\mathrm{f}\boldsymbol{A}}\boldsymbol{x}_\mathrm{i}$, which describes the state change that the system produces naturally by evolving the initial state forward by a time t_f.

If the controllability Gramian is not invertible, then an infinite amount of energy would be required to affect state transfer (at least to some states). Therefore a system is controllable if and only if its controllability Gramian is invertible (full rank). The controllability Gramian must also be positive definite for the system to be controllable (more on this later.) Normally in assessing controllabiilty we care about the case where $t \to \infty$, implying that can we drive the system to $\boldsymbol{x} = 0$ eventually. In this case the particular form of the Gramian that we care about is

$$\boldsymbol{\mathcal{W}}_\mathrm{c} := \int_0^\infty \mathrm{e}^{\tau \boldsymbol{A}} \boldsymbol{B}\boldsymbol{B}^\mathsf{T} \mathrm{e}^{\tau \boldsymbol{A}^\mathsf{T}} \,\mathrm{d}\tau$$

We don't normally solve this integral explicitly, but *if $\boldsymbol{A}$ has only negative eigenvalues* (the system is open loop stable), instead find the Gramian as the unique solution to the continuous time Lyapunov equation;

$$\boldsymbol{A}\boldsymbol{\mathcal{W}}_\mathrm{c} + \boldsymbol{\mathcal{W}}_\mathrm{c}\boldsymbol{A}^\mathsf{T} = -\boldsymbol{B}\boldsymbol{B}^\mathsf{T}$$

Matlab can be used to do this calculation using `Wc = gram(<sys>,'c')`

Let's check that the controllability Gramian is indeed a solution to the continuous time Lyapunov equation.

$$\begin{aligned}
\boldsymbol{A}\boldsymbol{\mathcal{W}}_\mathrm{c} + \boldsymbol{\mathcal{W}}_\mathrm{c}\boldsymbol{A}^\mathsf{T} &= \boldsymbol{A}\int_0^\infty \mathrm{e}^{\tau \boldsymbol{A}} \boldsymbol{B}\boldsymbol{B}^\mathsf{T} \mathrm{e}^{\tau \boldsymbol{A}^\mathsf{T}} \,\mathrm{d}\tau + \int_0^\infty \mathrm{e}^{\tau \boldsymbol{A}} \boldsymbol{B}\boldsymbol{B}^\mathsf{T} \mathrm{e}^{\tau \boldsymbol{A}^\mathsf{T}} \,\mathrm{d}\tau \boldsymbol{A}^\mathsf{T} \\
&= \int_0^\infty \boldsymbol{A}\mathrm{e}^{\tau \boldsymbol{A}} \boldsymbol{B}\boldsymbol{B}^\mathsf{T} \mathrm{e}^{\tau \boldsymbol{A}^\mathsf{T}} \,\mathrm{d}\tau + \int_0^\infty \mathrm{e}^{\tau \boldsymbol{A}} \boldsymbol{B}\boldsymbol{B}^\mathsf{T} \mathrm{e}^{\tau \boldsymbol{A}^\mathsf{T}} \boldsymbol{A}^\mathsf{T} \,\mathrm{d}\tau \\
&= \int_0^\infty \frac{\mathrm{d}}{\mathrm{d}\tau}\left(\mathrm{e}^{\tau \boldsymbol{A}} \boldsymbol{B}\boldsymbol{B}^\mathsf{T} \mathrm{e}^{\tau \boldsymbol{A}^\mathsf{T}}\right) \mathrm{d}\tau \\
&= \mathrm{e}^{\tau \boldsymbol{A}} \boldsymbol{B}\boldsymbol{B}^\mathsf{T} \mathrm{e}^{\tau \boldsymbol{A}^\mathsf{T}} \Big|_{\tau=0}^{\infty} \\
&= 0 - \boldsymbol{B}\boldsymbol{B}^\mathsf{T} \\
&= -\boldsymbol{B}\boldsymbol{B}^\mathsf{T}
\end{aligned}$$

Notice that this requires that $\mathrm{e}^{\tau \boldsymbol{A}} = 0$ for $\tau \to \infty$. This is what requires $\boldsymbol{A}$ to have only negative eigenvalues for Lyapunov's equation to be useful in finding the Gramian.

It can also easily be shown that the Gramian is a unique solution to Lyapunov's equation, but we won't do so here.

The controllability Gramian gives us more than a bare binary determination of whether a system is controllable. Let's look again at the form of $\boldsymbol{u}(t)$ required to push the system from $\boldsymbol{x} = 0$ to $\boldsymbol{x} = \boldsymbol{x}_\mathrm{f}$;

$$\begin{aligned}
\boldsymbol{u} &= \boldsymbol{B}^\mathsf{T}\mathrm{e}^{(t_\mathrm{f}-t)\boldsymbol{A}^\mathsf{T}} \left(\int_0^{t_\mathrm{f}} \mathrm{e}^{\tau \boldsymbol{A}} \boldsymbol{B}\boldsymbol{B}^\mathsf{T} \mathrm{e}^{\tau \boldsymbol{A}^\mathsf{T}} \,\mathrm{d}\tau\right)^{-1} \boldsymbol{x}_\mathrm{f} \\
&= \boldsymbol{B}^\mathsf{T}\mathrm{e}^{(t_\mathrm{f}-t)\boldsymbol{A}^\mathsf{T}} \boldsymbol{\mathcal{W}}_\mathrm{c}^{-1}\boldsymbol{x}_\mathrm{f}
\end{aligned}$$

This equation tells us that $\boldsymbol{\mathcal{W}}_\mathrm{c}^{-1}$ is acting simply as a scaling, that tells us which directions require large (or small) $\boldsymbol{u}$ to bring about a required change in $\boldsymbol{x}$. That is, in directions where $\boldsymbol{\mathcal{W}}_\mathrm{c}$ is large, it is "easy" to move the system. In directions where $\boldsymbol{\mathcal{W}}_\mathrm{c}$ is small, it is more difficult to move the system

We can use eigenanalysis to find the spatial properties of the controllability Gramian, as well as to do our assessment of overall controllability.

- If $\boldsymbol{\mathcal{W}}_\mathrm{c}$ has only positive eigenvalues, then it is positive definite, and the system is controllable.
- Conversely if there are non-positive eigenvalues the system (including zero eigenvalues) is *not* controllable.
- Eigenvectors corresponding to non-positive eigenvalues of $\boldsymbol{\mathcal{W}}_\mathrm{c}$ are the directions in which the system is not controllable.

As you might expect, the Gramian is also defined for discrete time systems. The argument is much the same as that presented above, and we arrive at

$$\boldsymbol{\mathcal{W}}_\mathrm{c}(t) := \sum_{t=0}^{t_\mathrm{f}} \boldsymbol{A}^t\boldsymbol{B}\boldsymbol{B}^\mathsf{T}\left(\boldsymbol{A}^\mathsf{T}\right)^t$$

$$\text{and,}\qquad \boldsymbol{\mathcal{W}}_\mathrm{c} := \sum_{t=0}^{\infty} \boldsymbol{A}^t\boldsymbol{B}\boldsymbol{B}^\mathsf{T}\left(\boldsymbol{A}^\mathsf{T}\right)^t$$

The discrete time controllability Gramian is the unique solution to the discrete time Lyapunov equation, this time assuming that $\boldsymbol{A}$ has only eigenvalues with magnitude less than one.

$$\boldsymbol{A}\boldsymbol{\mathcal{W}}_{\mathrm{c}}\boldsymbol{A}^{\mathsf{T}} - \boldsymbol{\mathcal{W}}_{\mathrm{c}} = -\boldsymbol{B}\boldsymbol{B}^{\mathsf{T}}$$

The discrete time Gramian can be calculated using the same Matlab command as in the continuous time case; `Wc = gram(<sys>,'c')`

2 Regulator Design

We have now developed our state space theory sufficiently to examine the design of compensators (aka controllers). There are two main classes of control problems:

1. Regulator problems, where we seek to stabilise a system at some set point.

2. Servo (or tracking) problems, where we want our system to follow some command input(s).

Regulator problems are simpler, as we do not have command (or reference) inputs entering the system. When designing regulators we concentrate on modification of the system dynamics. When we later look at servo systems we will concentrate on reducing errors.

2.1 Compensator Design

The basic structure of our system with an added compensator is as shown in figure 2.1.

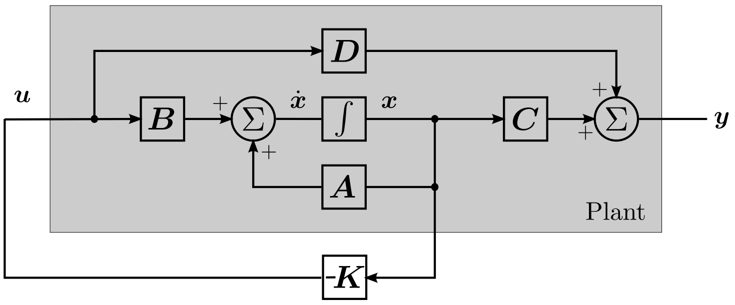


Figure 2.1: A regulated system using state feedback.

- The basic premise is that we use the current state vector to provide feedback to the system.

- The feedback is a linear function of the state and the matrix $\boldsymbol{K}$ describes *how* the state variables are fed back to the various inputs.

The first of these properties should make intuitive sense. The state of the system encapsulates everything there is to know about the system. What other information could possibly be valuable in a feedback signal?

The second property is less obvious. In particular, it is not obvious that a linear combination of states will provide the best feedback signal. In fact, although one could certainly use nonlinear feedback laws, the linear option can be shown to be an optimal choice presuming that the system being controlled is also linear.

- $\boldsymbol{K} \in \mathbb{R}^{m \times n}$, where n is the number of states and m the number of system inputs.

- For now we will consider $\boldsymbol{K}$ as time-invariant, but later we will allow it to change in optimal control systems.

- In a classical control problem we apply feedback from only the single output. In modern control we (might) have more information available, and more places to "push" on the system, so it is not unreasonable to hope that we will have more flexibility with the state space approach.

- We will assume for now that we have perfect knowledge of the current internal state of the system. This is often an unrealistic assumption that we will address later.

- It is simply a convention that the feedback matrix is $-\boldsymbol{K}$ rather than $\boldsymbol{K}$.

Recall that our system is described by $\dot{\boldsymbol{x}} = \boldsymbol{Ax} + \boldsymbol{Bu}$, $\boldsymbol{y} = \boldsymbol{Cx} + \boldsymbol{Du}$. We seek to design an appropriate feedback matrix $\boldsymbol{K}$ so that we can set $\boldsymbol{u} = -\boldsymbol{Kx}$. We will now substitute for $\boldsymbol{u}$ in the $\dot{\boldsymbol{x}}$ equation and investigate the behaviour of the new system.

$$\begin{aligned}\dot{\boldsymbol{x}} &= \boldsymbol{Ax} + \boldsymbol{Bu}\\ \dot{\boldsymbol{x}} &= \boldsymbol{Ax} - \boldsymbol{BKx}\\ \dot{\boldsymbol{x}} &= (\boldsymbol{A} - \boldsymbol{BK})\,\boldsymbol{x}\end{aligned}$$

Notice that this new system looks like an autonomous state space system with the $\boldsymbol{A}$ matrix replaced by $\boldsymbol{A} - \boldsymbol{BK}$. That is, the evolution of the state is now determined by the characteristics of $\boldsymbol{A} - \boldsymbol{BK}$ rather than $\boldsymbol{A}$.

Recall that the open loop system poles were the eigenvalues of $\boldsymbol{A}$. Similarly we can find the closed loop poles of the system as the eigenvalues of $(\boldsymbol{A} - \boldsymbol{BK})$. That is, the closed loop system poles will be the roots of the closed loop characteristic equation;

$$\boxed{\det\big(s\boldsymbol{I} - (\boldsymbol{A} - \boldsymbol{BK})\big) = 0}$$

As before, substituting the appropriate matrices will result in a polynomial in s. However, the elements of the $\boldsymbol{K}$ matrix will now appear in the various coefficients of the powers of s. If we know what we would like the characteristic polynomial to be, we can equate coefficients of the powers of s to determine $\boldsymbol{K}$.

2.1.1 Compensator design example

Let's look at an example drawn from Franklin, Powell and Emami-Naeini 'Feedback Control of Dynamic Systems' 3rd ed, p 495.
Consider an undamped oscillator with frequency ω_0. This has a state space description of

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -\omega_0^2 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ 1\end{bmatrix} u\ , \quad y = x_1 \quad .$$

We wish to apply feedback to the oscillator so that its closed loop poles move to $-2\omega_0$. That is, we wish to double the natural frequency of the oscillator and increase ζ to 1. First let's check the pole locations of the open loop system. We solve the characteristic equation of this system to find the pole locations.

$$\begin{aligned}\left|s\boldsymbol{I} - \boldsymbol{A}\right| = \begin{vmatrix}s & -1\\ \omega_0^2 & s\end{vmatrix} &= 0\\ \implies s^2 + \omega_0^2 &= 0\\ s &= \pm \mathrm{j}\omega_0\end{aligned}$$

As we would expect.
Figure 2.2 shows the open loop response of the oscillator. We can see that the two state variables are two sinusoidal signals $\pi/2$ out of phase. The output signal is equal to x_1.

We would like both poles to move to $s = -2\omega_0 + \mathrm{j}0$. That is, we would like the characteristic polynomial of the closed loop system to be $\chi_c(s) = (s+2\omega_0)(s+2\omega_0) = s^2+4\omega_0 s+4\omega_0^2$. Let us compare the desired characteristic equation with that of the closed loop system:

$$\begin{aligned}\left|s\boldsymbol{I} - (\boldsymbol{A} - \boldsymbol{BK})\right| &= \det\left(\begin{bmatrix}s & 0\\ 0 & s\end{bmatrix} - \left(\begin{bmatrix}0 & 1\\ -\omega_0^2 & 0\end{bmatrix} - \begin{bmatrix}0\\ 1\end{bmatrix}\begin{bmatrix}k_1 & k_2\end{bmatrix}\right)\right)\\ &= \det\left(\begin{bmatrix}s & 0\\ 0 & s\end{bmatrix} - \begin{bmatrix}0 & 1\\ -\omega_0^2 & 0\end{bmatrix} + \begin{bmatrix}0 & 0\\ k_1 & k_2\end{bmatrix}\right)\\ &= \det\begin{bmatrix}s & -1\\ \omega_0^2 + k_1 & s + k_2\end{bmatrix}\\ &= s(s + k_2) + \omega_0^2 + k_1\\ &= s^2 + k_2 s + \omega_0^2 + k_1\end{aligned}$$

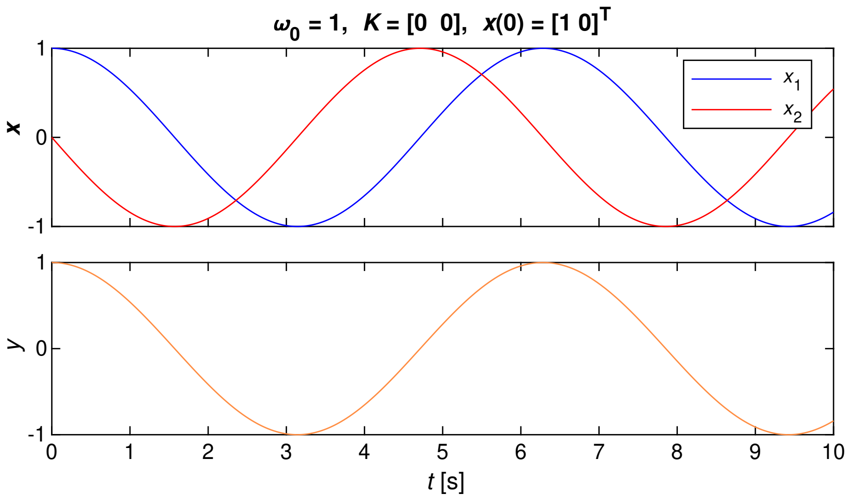


Figure 2.2: State and Output trajectories for an oscillator.

To find k_1 and k_2 we can equate the coefficients of the various powers of s in these two equations.

$$\begin{cases} s^1: & k_2 = 4\omega_0 \\ s^0: & \omega_0^2 + k_1 = 4\omega_0^2 \end{cases}$$

$$\begin{aligned} \text{So,} \quad & k_1 = 3\omega_0^2 \\ & k_2 = 4\omega_0 \\ \implies & \boldsymbol{K} = \begin{bmatrix} 3\omega_0^2 & 4\omega_0 \end{bmatrix}. \end{aligned}$$

Now $u = -\boldsymbol{Kx}$, so our compensator is realized by building a circuit (or writing a piece of software) to implement $u = 3\omega_0^2 x_1 + 4\omega_0 x_2$. Remember that ω_0 would be a constant in a real problem, so u is simply a linear function of $\boldsymbol{x}$.

The operation of our regulated system is shown in figure 2.3, which shows that the response of the state variables is now consistent with a pole pair at $s = -2\omega_0$. We can see that we have successfully achieved regulation of the output signal.

2.1.2 Compensator Discussion

State space compensators such as this provide feedback from only the state variables and their derivatives. For example, the oscillator damping exercise above implemented a feedback rule of

$$u = 3\omega_0^2 x_1 + 4\omega_0 x_2 \equiv 3\omega_0^2 x_1 + 4\omega_0 \dot{x}_1$$

Such controllers are therefore simply a bank of PD compensators (or the degenerate possibility of P or D controllers). A PD controller does not increase the system type, so this simple state space controller may not be adequate for removing steady state errors (or errors in the presence of a ramp etc.). We will talk later about extending our regulator to add integral action.

2.1.3 Compensator design in control canonical form

It turns out that designing a compensator is trivial if the system is expressed in control canonical form.

In control canonical form we have matrices $\boldsymbol{A}_c = \begin{bmatrix} -d_{n-1} & -d_{n-2} & \cdots & -d_1 & -d_0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$ and $\boldsymbol{B}_c = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$.

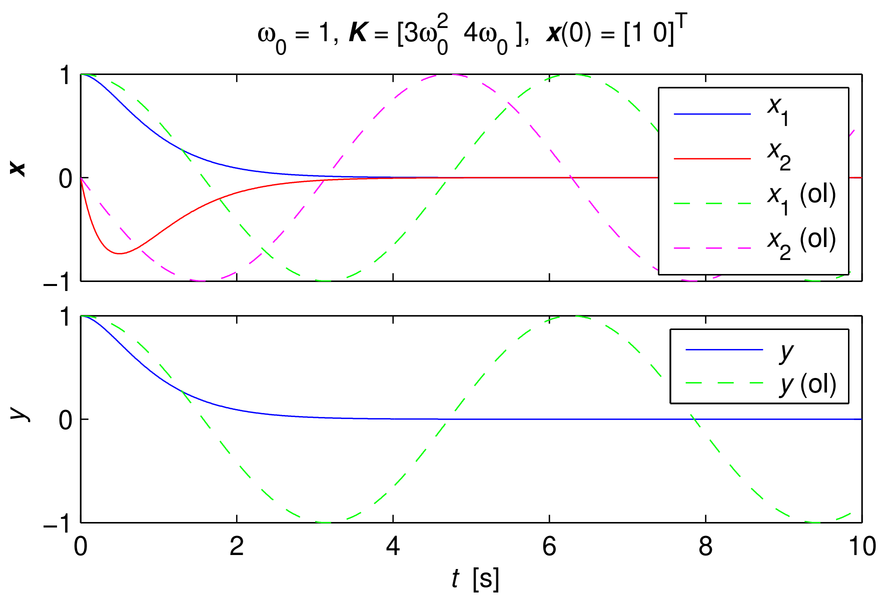


Figure 2.3: Performance of the damped oscillator system. The operation of the original system is shown in dashed lines for comparison.

In this case the closed loop state matrix is

$$\boldsymbol{A}_c - \boldsymbol{B}_c\boldsymbol{K}_c = \begin{bmatrix} -d_{n-1}-k_1 & \cdots & -d_1-k_{n-1} & -d_0-k_n \\ 1 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix}$$

The characteristic equation of this matrix will be

$$\det\left(s\boldsymbol{I} - (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K})\right) = 0$$
$$s^n + (d_{n-1}+k_1)s^{n-1} + \cdots + (d_0+k_n) = 0$$

It is now easy to compare this to our desired characteristic equation,

$$\chi_{\mathrm{c}} = s^n + \alpha_1 s^{n-1} + \cdots + \alpha_n$$

Equating the two we get a system of n simultaneous equations to find the various k_i:

$$\begin{aligned} \alpha_1 &= d_{n-1}+k_1 \implies k_1 = \alpha_1 - d_{n-1} \\ \alpha_2 &= d_{n-2}+k_2 \implies k_2 = \alpha_2 - d_{n-2} \\ &\vdots \qquad\qquad\qquad \vdots \\ \alpha_n &= d_0 + k_n \quad\implies k_n = \alpha_n - d_0 \end{aligned}$$

This gives us an equation for $\boldsymbol{K}_c$ in the control canonical form.

In general, when we transform a system we must transform $\boldsymbol{K}$ as well. $\boldsymbol{K}$ provides feedback from the state vector, so the feedback will cease working if we change the definition of the state vector without changing $\boldsymbol{K}$. As before, consider a change of state vector from $\boldsymbol{x}$ to $\boldsymbol{z}$ according to $\boldsymbol{x} = \boldsymbol{T}\boldsymbol{z}$. We wish to find $\bar{\boldsymbol{K}}$, the realisation of $\boldsymbol{K}$ appropriate to the new state vector. The input signal $\boldsymbol{u}$ produced in the two realisations must be the same. Thus

$$\begin{aligned}\bar{\boldsymbol{K}}\boldsymbol{z} &= \boldsymbol{K}\boldsymbol{x}\\ &= \boldsymbol{K}\boldsymbol{T}\boldsymbol{z}\\ \implies \bar{\boldsymbol{K}} &= \boldsymbol{K}\boldsymbol{T}\end{aligned}$$

In the current situation we have designed $\boldsymbol{K}$ in the control canonical form. If we were given the system in that form then using $\boldsymbol{K}$ is fine, but if we began by transforming the state space transformation then we need to move $\boldsymbol{K}$ back to the initial representation with the formula above.

We now have a second technique for finding $\boldsymbol{K}$ which is useful when the order of the system is greater than three. For small systems it is normally easier to simply equate the coefficients directly.

1. Put the system into control canonical form using a transformation matrix $\boldsymbol{T}^{-1}$.
2. Determine the required $\boldsymbol{K}$ values by inspection.
3. Convert $\boldsymbol{K}$ back to the initial realisation using $\boldsymbol{T}$.

2.1.4 Ackermann's formula

Direct comparison of the closed loop poles and the desired pole locations allows us to solve for $\boldsymbol{K}$ directly, however that method becomes messy as the order of the system increases. Often we would like to avoid converting the system to control canonical form also. *If we have a SISO system*, an alternative is to use Ackermann's formula. We will skip derivation of Ackermann's formula and simply state the result. Ackermann's formula provides us with a general process that can be used to choose $\boldsymbol{K}$. It basically automates shifting the system into control canonical form, finding $\boldsymbol{K}_c$ and then shifting back to the original realization.

Ackermann's formula states that

$$\boxed{\boldsymbol{K} = \begin{bmatrix}0 & 0 & \cdots & 0 & 1\end{bmatrix}\boldsymbol{\mathcal{M}}_{\mathrm{c}}^{-1}\chi_{\mathrm{c}}(\boldsymbol{A})}$$

where $\boldsymbol{\mathcal{M}}_{\mathrm{c}}$ is the controllability matrix and $\chi_{\mathrm{c}}(\boldsymbol{A})$ is a the desired characteristic equation with $\boldsymbol{A}$ substituted throughout for s. That is, if we want a characteristic equation

$$\begin{aligned}\chi_{\mathrm{c}}(s) &= s^n + \alpha_1 s^{n-1} + \alpha_2 s^{n-2} + \cdots + \alpha_n\\ \text{then, } \chi_{\mathrm{c}}(\boldsymbol{A}) &= \boldsymbol{A}^n + \alpha_1\boldsymbol{A}^{n-1} + \alpha_2\boldsymbol{A}^{n-2} + \cdots + \alpha_n\boldsymbol{I}\end{aligned}$$

Note that $\boldsymbol{\mathcal{M}}_{\mathrm{c}}$ will be invertible in a controllable, SISO, system. The fact that it is SISO will make $\boldsymbol{\mathcal{M}}_{\mathrm{c}}$ square, and the fact that is controllable means that the columns of $\boldsymbol{\mathcal{M}}_{\mathrm{c}}$ are independent, and hence $\boldsymbol{\mathcal{M}}_{\mathrm{c}}^{-1}$ exists. Also notice that is the controllability matrix is "small" then $\boldsymbol{\mathcal{M}}_{\mathrm{c}}^{-1}$ will be large. As a consequence, a system that is only weakly controllable will require large gain.

As an example, we will redo the oscillator example that we encountered above using Ackermann's formula. If you look back you will see that our desired polynomial is $s^2 + 4\omega s + 4\omega_0^2$. Thus we have $\alpha_1 = 4\omega_0$ and $\alpha_2 = 4\omega_0^2$. Let's find α_{c} using the equation above:

$$\begin{aligned}\chi_{\mathrm{c}}(\boldsymbol{A}) &= \boldsymbol{A}^2 + 4\omega_0\boldsymbol{A} + 4\omega_0^2\boldsymbol{I}\\ &= \begin{bmatrix}0 & 1\\ -\omega_0^2 & 0\end{bmatrix}\begin{bmatrix}0 & 1\\ -\omega_0^2 & 0\end{bmatrix} + 4\omega_0\begin{bmatrix}0 & 1\\ -\omega_0^2 & 0\end{bmatrix} + 4\omega_0^2\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\\ &= \begin{bmatrix}-\omega_0^2 & 0\\ 0 & -\omega_0^2\end{bmatrix} + 4\omega_0\begin{bmatrix}0 & 1\\ -\omega_0^2 & 0\end{bmatrix} + 4\omega_0^2\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\\ &= \begin{bmatrix}3\omega_0^2 & 4\omega_0\\ -4\omega_0^3 & 3\omega_0^2\end{bmatrix}\end{aligned}$$

Now $\boldsymbol{\mathcal{M}}_{\mathrm{c}} = \begin{bmatrix}\boldsymbol{B} & \boldsymbol{A}\boldsymbol{B}\end{bmatrix} = \left[\begin{bmatrix}0\\1\end{bmatrix} \quad \begin{bmatrix}0 & 1\\ -\omega_0^2 & 0\end{bmatrix}\times\begin{bmatrix}0\\1\end{bmatrix}\right] = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \implies \boldsymbol{\mathcal{M}}_{\mathrm{c}}^{-1} = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$ Substituting into

Ackermann's formula, we get

$$\begin{aligned}
\boldsymbol{K} &= \begin{bmatrix}0 & 1\end{bmatrix} \boldsymbol{\mathcal{M}}_{\mathrm{c}}^{-1} \chi_{\mathrm{c}} \\
&= \begin{bmatrix}0 & 1\end{bmatrix} \begin{bmatrix}0 & 1 \\ 1 & 0\end{bmatrix} \begin{bmatrix}3\omega_0^2 & 4\omega_0 \\ -4\omega_0^3 & 3\omega_0^2\end{bmatrix} \\
\boldsymbol{K} &= \begin{bmatrix}3\omega_0^2 & 4\omega_0\end{bmatrix}
\end{aligned}$$

As before.

2.1.5 Regulator Design for Stabilisable Systems

There are a number of methods that can be used to design regulators for systems that are uncontrollable, but stabilisable. The simplest is to simply ignore the uncontrollable states by removing them from the model while completing your regulator design. The states can then be added back to the system to assess overall performance.

1. Put the system into a form where uncontrollable modes can be identified (typically modal or Jordan canonical form or similar). Alternatively you might use the PBH test, or the Controllability Grammian to assess controllability.
2. Remove the uncontrollable state variables from the model to make a simplified model that neglects the uncontrollable dynamics. This results in all of the system matrices "shrinking" as there are not as many state variables to deal with.
3. Design the feedback matrix $\boldsymbol{K}$ using any appropriate method (using the simplified model).
4. Refine $\boldsymbol{K}$ as necessary, perhaps using simulation with the simplified model.
5. Add the uncontrolled states back into the model and assess overall performance.

2.2 Choice of Pole Locations

Until now we have assumed that we knew where we wanted the closed loop poles to be. In reality we will rarely be told where the system poles should be. There are a number of strategies that we could adopt to locate the poles.

1. Choose poles manually to place a dominant pole pair so that the system satisfy some damping or settling time requirement.
2. Use a prototype set of responses, such as ITAE, Butterworth or Bessel.
3. Use dead-beat control (discrete time only)
4. Use LQR techniques.

2.2.1 Manual selection of pole locations

The typical method of selecting poles is to arrange them such that there is a dominant pair of complex poles. These poles can be located to satisfy the system performance requirements. All other poles in the system should be placed considerably further into the left half of the s-plane, or centre of the z-plane, so that they have negligible effect on the overall response. Make their model response at least 3–5 times faster than the dominant mode. However, it is not that simple, as moving poles requires control effort. Moving poles a long way might require a larger servo for example. Therefore, try not to move poles further than necessary or to poles that are hard to move (such as those close to zeros). If there are modes that are not troublesome (such as fast decaying modes in mechanical systems), then don't move the corresponding poles. Certainly avoid making them worse by slowing them down.

2.2.2 Butterworth Configuration

A disadvantage of moving all but a dominant pair of poles towards the left of the s-plane (or centre of the z-plane) is that it takes a large control effort (large $\boldsymbol{K}$ and therefore $\boldsymbol{u}$). A system will tend to require less control effort if the closed loop poles are located at about the same distance from the origin, though this does make the response more sluggish. If we took this approach to the extreme we could place the closed loop poles in a Butterworth configuration, which in continuous time spaces the poles equally along a circle centered at the origin. You might consider using a Butterworth arrangement when you have a system with a group of similarly slow open poles. Rather than choosing a single dominant pair, a Butterworth arrangement might prove more useful. You should certainly never slow down fast modes just to put them in a Butterworth arrangement.

The following equations are the denominators of the transfer functions that generate the Butterworth configurations when $\omega_0 = 1$ (the numerator is one in each case).

1	$s+1$
2	$s^2+\sqrt{2}s+1$
3	$(s+1)(s^2+s+1)$
4	$(s^2+0.7654s+1)(s^2+1.8478s+1)$
5	$(s+1)(s^2+0.6180s+1)(s^2+1.6180s+1)$
6	$(s^2+0.5176s+1)(s^2+1.4142s+1)(s^2+1.9319s+1)$

Matlab has good support for Butterworth filter design, so don't enter the pole locations manually. We choose the cutoff frequency to be the desired magnitude of the pole location, though some fine tuning is likely to be necessary. Set the filter's order to the number of poles that you wish to place in the Butterworth configuration. For continuous time use

```
[z, p, k] = butter(<order>, <cutoff freq>, 's')
```

For discrete time express the cutoff frequency as a fraction of the Nyquist frequency and use

```
[z, p, k] = butter(<order>, <cutoff freq>)
```

In each case the resulting poles can be found in `p`. Alternatively use `[n, d] = butter(...)` to get the characteristic polynomial in `d`.

2.2.3 ITAE responses for pole locations

An alternate is to use the ITAE prototypes, which are pole locations calculated to minimise the **i**ntegral of the **t**ime multiplied by the **a**bsolute value of the **e**rror. That is, we use pole locations precalculated to minimize

$$J\big(u(t)\big) = \int_0^\infty t|e|\,\mathrm{d}t$$

with $e = y - y_{\text{desired}}$. This cost functional J will be small if the controller drives y towards y_{desired}, as the integrand then goes to zero quickly. Controllers that do so quickly will be preferred as the presence of the t term in J means that deviations from the desired performance cost more if they happen later. We have two sets of these pole locations to choose from. The "normal" ITAE locations simply minimise J. The Bessel family of transfer functions also minimize the integral, but with the extra constraint that they exhibit no overshoot to a step input. The Bessel prototypes are slower than the ITAE responses, so use them only when necessary.

The following equations are the characteristic equations for the ITAE responses when $\omega_0 = 1$. For other ω_0, substitute s/ω_0 for s throughout.

1	$s + 1$
2	$s + 0.7071 \pm 0.7071\mathrm{j}$
3	$(s + 0.7081)(s + 0.5210 \pm 1.068\mathrm{j})$
4	$(s + 0.4240 \pm 1.2630\mathrm{j})(s + 0.6260 \pm 0.4141\mathrm{j})$
5	$(s + 0.8955)(s + 0.3764 \pm 1.2920\mathrm{j})(s + 0.5758 \pm 0.5339\mathrm{j})$
6	$(s + 0.3099 \pm 1.2634\mathrm{j})(s + 0.5808 \pm 0.7828\mathrm{j})$
	$\times\, (s + 0.7346 \pm 0.2873\mathrm{j})$

$s + \alpha \pm \beta\mathrm{j}$ indicates that there is a pole pair at $-\alpha + \beta\mathrm{j}$ and $-\alpha - \beta\mathrm{j}$. Sadly, there appears to be no in-built Matlab function to produce the ITAE poles.

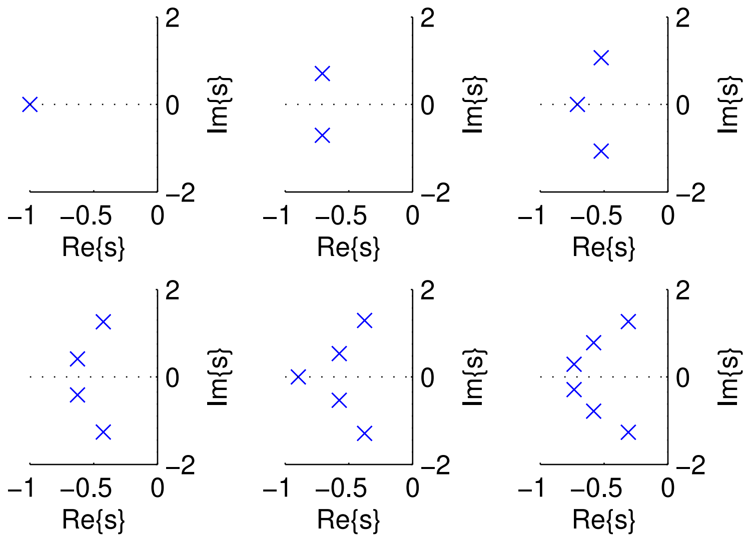


Figure 2.4: Pole locations for first to sixth order ITAE responses.

2.2.4 Bessel transfer functions

The following are the characteristic polynomials corresponding to the Bessel responses when $\omega_0 = 1$. For other ω_0, substitute s/ω_0 throughout.

1	$s + 1$
2	$s + 0.8660 \pm 0.5\mathrm{j}$
3	$(s + 0.9420)(s + 0.7455 \pm 0.7112\mathrm{j})$
4	$(s + 0.6573 \pm 0.8302\mathrm{j})(s + 0.9047 \pm 0.2711\mathrm{j})$
5	$(s + 0.9264)(s + 0.5906 \pm 0.9072\mathrm{j})(s + 0.8516 \pm 0.4427\mathrm{j})$
6	$(s + 0.5385 \pm 0.9617\mathrm{j})(s + 0.7998 \pm 0.5622\mathrm{j})$
	$\times\, (s + 0.9093 \pm 0.1856\mathrm{j})$

$s + \alpha \pm \beta\mathrm{j}$ indicates that there is a pole pair at $-\alpha + \beta\mathrm{j}$ and $-\alpha - \beta\mathrm{j}$. Use Matlab to get these: `[z,p,k] = besself(<order>,<cutoff frequency>)` will put the poles into `p`.

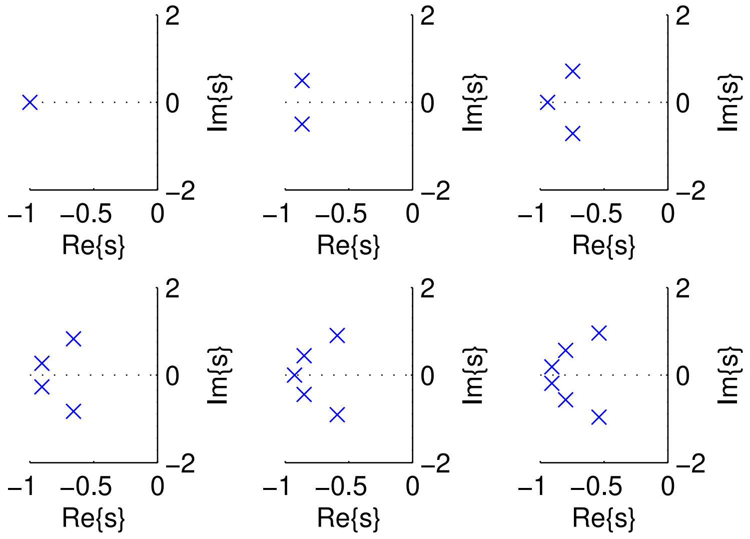


Figure 2.5: Pole-zero map for Bessel arrangements of first through sixth order.

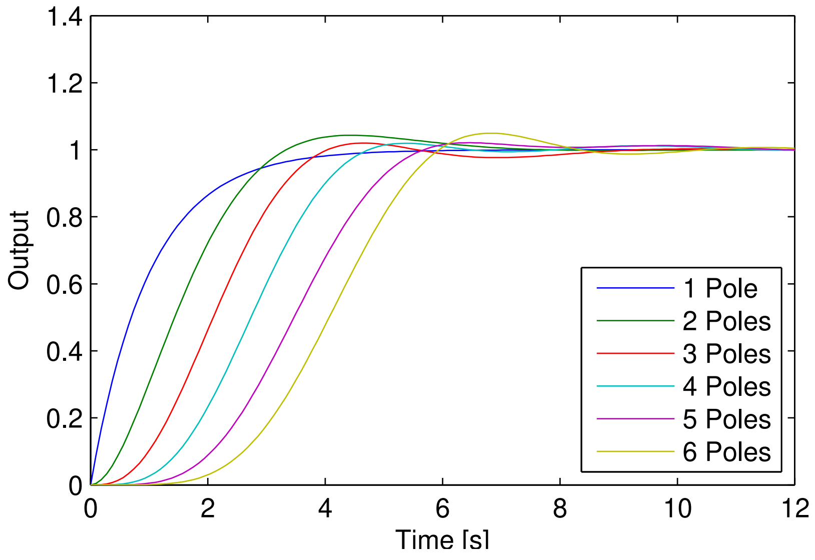


Figure 2.6: Step responses of different orders of ITAE pole arrangements.

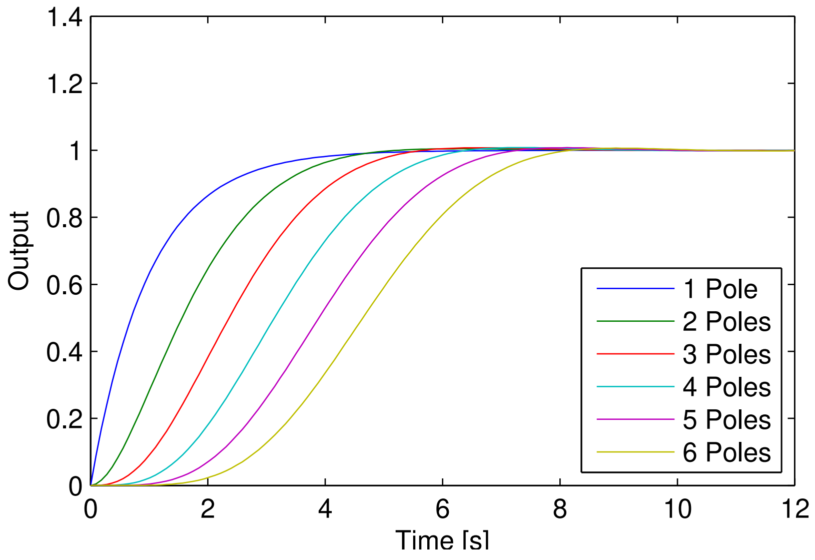


Figure 2.7: Step responses for different order Bessel pole configurations.

Figures 2.6 and 2.7 show the step responses of systems with ITAE and Bessel pole arrangements respectively.

In each case the order of the curves increases from left to right. Notice that the responses become more sluggish as we increase the number of poles in the same region.

2.2.5 Dead-beat Control

For a discrete time problem, a so called *dead-beat* controller can be used to drive the state to zero in the minimum time possible. That is, we can drive the state from some initial x_0 to zero in the smallest t for which $x_0 \in \mathcal{R}_t$. To build a dead-beat controller we simply place *all* of the closed loop poles at $z = 0$. Dead-beat controllers have the best possible speed and also zero steady state error, but normally produce very large control signals. There is no analogue of dead-beat control for CT systems, as they can be driven to any point instantly, though this may require an *extremely* violent control effort.

2.2.6 Linear Quadratic Regulators

Notice that the magnitude of the required control effort was ignored in the previous approaches. They can therefore lead to controller designs that require very large control signals. A more complete approach is to include the magnitude of the control effort into our performance metric. Thus, instead of minimising $J\big(u(t)\big) = \int_0^\infty t|e|\,\mathrm{d}t$, we will add a term that penalises large values of u.
That is, we seek to minimize a function like

$$J\big(u(t)\big) = \int_0^\infty x^2(t) + \rho u^2(t)\,\mathrm{d}t$$

for a designer selected weighting ρ which indicates how "worried" the designer is about deviations in x and u. We will return to this idea when we look at *optimal control*.

2.3 Compensator Design Example

Consider a continuous time system described in state space by

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -18 & -15 & -2 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, \boldsymbol{D} = \begin{bmatrix} 0 \end{bmatrix} \quad .$$

Design a compensator such that the closed loop system has an overshoot of 6% or better and a settling time of 3 seconds. We begin by checking the controllability of the system. The controllability matix is

$$\boldsymbol{\mathcal{M}}_\mathrm{c} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{AB} & \boldsymbol{A}^2\boldsymbol{B} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \\ -2 \end{bmatrix} & \begin{bmatrix} 1 \\ -2 \\ -11 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & -2 \\ 1 & -2 & -11 \end{bmatrix}$$

We can immediately see that $\boldsymbol{\mathcal{M}}_\mathrm{c}$ is full rank becuase it it triangular.

Recall from previous studies of second order systems that $\zeta = \dfrac{-\ln\left(\%OS/100\right)}{\sqrt{\pi^2 + \ln^2(\%OS/100)}}$ and $t_\mathrm{s} = \dfrac{4}{\zeta\omega_\mathrm{n}}$.
To achieve the specification we must have $\zeta = 0.67$ and $\omega_\mathrm{n} = 2$ rad/s, which corresponds to a dominant pole pair at $s = -1.33 \pm \mathrm{j}1.49$. Our system is third order, so we must choose the location of a third closed loop pole. Let's place it a factor of ten further into the left half of the s-plane than the dominant poles (ie, at $s = -13.3$). Thus our desired characteristic polynomial is

$$(s + 1.33 + \mathrm{j}1.49)(s + 1.33 - \mathrm{j}1.49)(s + 13.3) = s^3 + 16s^2 + 39.55s + 53.26$$

We need to equate this to the characteristic equation of the closed loop system, $\det\left(s\boldsymbol{I} - (\boldsymbol{A} - \boldsymbol{BK})\right)$.

The expression for the closed loop characteristic polynomial is

$$\begin{aligned} &\det(s\boldsymbol{I} - \boldsymbol{A} + \boldsymbol{BK}) \\ &= \det\left(\begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -18 & -15 & -2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} k_1 & k_2 & k_3 \end{bmatrix} \right) \\ &= \det\left(\begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -18 & -15 & -2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ k_1 & k_2 & k_3 \end{bmatrix} \right) \\ &= \begin{vmatrix} s & -1 & 0 \\ 0 & s & -1 \\ 18 + k_1 & 15 + k_2 & s + 2 + k_3 \end{vmatrix} \\ &= s^3 + (k_3 + 2)s^2 + (k_2 + 15)s + k_1 + 18 \end{aligned}$$

Equating the various coefficients of this characteristic equation with our desired characteristic equation yields

$$\begin{cases} s^2: & k_3 + 2 &= 16 \\ s^1: & k_2 + 15 &= 39.55 \\ s^0: & k_1 + 18 &= 53.26 \end{cases}$$

Solving for the various k values we obtain,

$$\boldsymbol{K} = \begin{bmatrix} 35.26 & 24.55 & 14 \end{bmatrix}$$

Let's compare the compensated closed loop and open loop responses to an impulse. The two responses are shown in in figure 2.8, which was produced with the matlab code

```
impulse(ss(A,B,C,D))
hold on
impulse(ss(A-B*K,B,C,D))
hold off
```

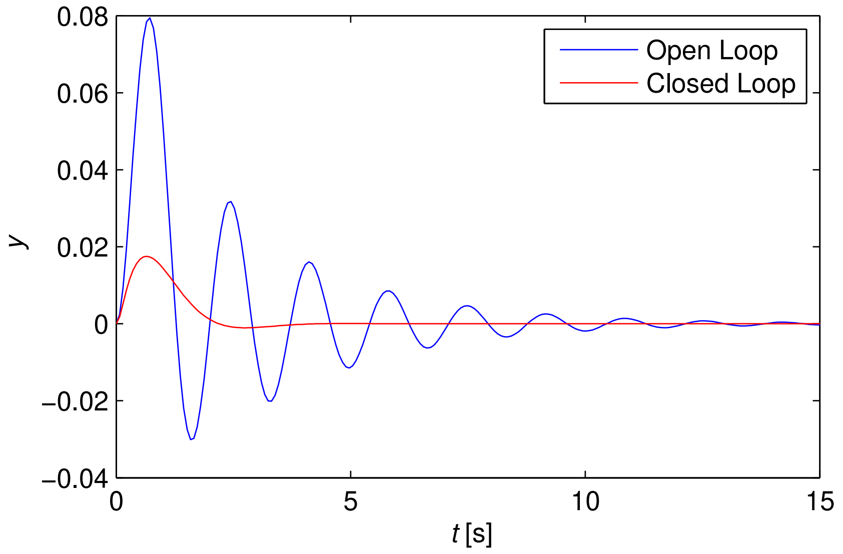


Figure 2.8: Comparison of the impulse responses of the open and closed loop (compensated) systems.

There is some subtlety in the Matlab construction of the closed loop system. As we are now using $\boldsymbol{u}$ to provide feedback, it is not immediately obvious what the input to our closed loop system should be. We typically rearrange the block diagram to pull the $\boldsymbol{B}$ matrix before the summation. This leads to the arrangement in figure 2.9, where $\boldsymbol{B}$ appears both in the the feedback path and in the path of the new signal $\boldsymbol{r}$, which is the input to the *closed loop system*.

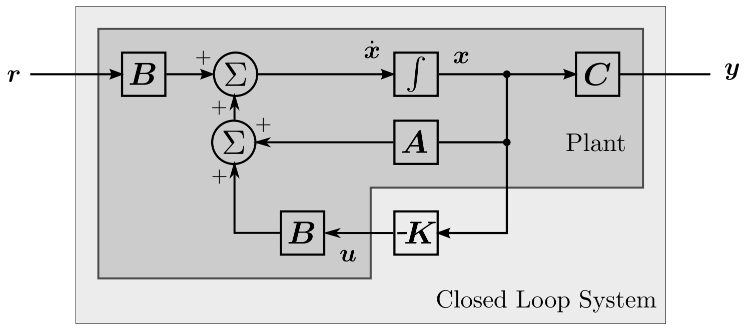


Figure 2.9: Construction of a closed loop system.

It doesn't really make sense to talk about an input to a closed loop regulator, so for now $\boldsymbol{r}$ is just a convenient way to excite the closed loop system so that you can see its response. We will talk more about $\boldsymbol{r}$ when we cover servo systems. The closed loop input is typically assumed to be coupled into the system in exactly the same way as $\boldsymbol{u}$ was originally, so this input to state coupling is described by the same $\boldsymbol{B}$ as before. This is simply saying that we have a set of actuators that can be used to apply feedback ($-\boldsymbol{BKx}$), but also to apply an external reference signal $\boldsymbol{Br}$. In fact, these two $\boldsymbol{B}$ matrices need not be the same, so we can use the two appearances of $\boldsymbol{B}$ to gain some design flexibility. For example we might decide to exclude one of more of the inputs from our feedback scheme, so remove the appropriate column(s) from $\boldsymbol{B}$ in the feedback loop. Similarly if we want $\boldsymbol{r}$ to excite the state in some restricted way then we could change the $\boldsymbol{B}$ matrix seen by $\boldsymbol{r}$.

2.4 Matlab for Compensator Design

Matlab has two functions that can be used to find the required $\boldsymbol{K}$ vector to build a state-feedback controller; `acker` and `place`.

```
K = acker(A, B, p)
K = place(A, B, p)
```

Here `p` is a vector containing the list of desired pole locations. Note that there is a limit on the number of repeated roots allowed when you use `place` (see Matlab's help). However, `place` has better numerical behaviour than `acker`, so use `place` whenever you can. Note: `acker` is only suitable for SISO systems.

So, to repeat our previous example using matlab, we write:

```
K= place(A, B, [-1.33+1.49j, -1.33-1.49j, -13.3] )
>> K = 35.0537 24.3670 13.9600
```

This results in the feedback gain matrix, $\boldsymbol{K} = \begin{bmatrix}35 & 24.4 & 14\end{bmatrix}$, which is in good agreement with our manual design.

Consider the temperature control apparatus shown in figure 2.10.

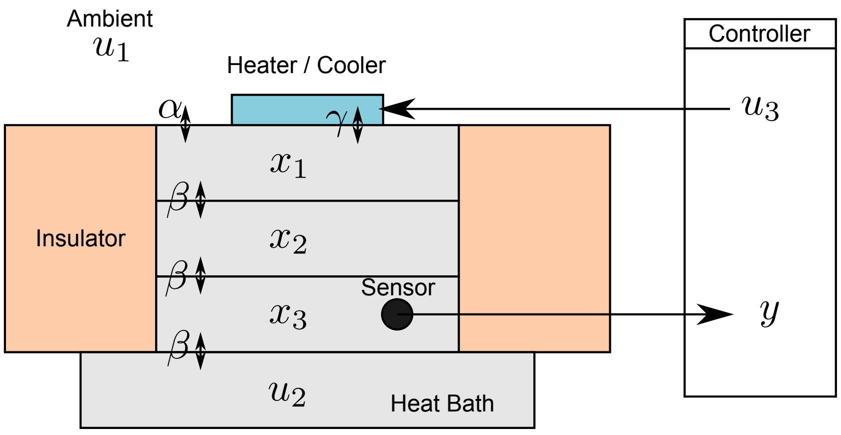


Figure 2.10: A schematic of the temperature control apparatus

The state variables $x_{1..3}$ represent the temperatures in regions one to three. We aim to design a temperature controller for region three of the apparatus using the heater/cooler. That is, we want to be able to regulate x_3 to be any desired value using only u_3. We will use $t_s = 0.1$ s. Transients in temperature should decay in 20 seconds (or less).

The temperature at several locations within the apparatus can be approximated by the discrete time state space model

$$\boldsymbol{x}(t+1) = \begin{bmatrix} 1-\alpha t_s-\beta t_s & \beta t_s & 0 & 1 \\ \beta t_s & 1-2\beta t_s & \beta t_s & 0 \\ 0 & \beta t_s & 1-\beta t_s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} \alpha t_s & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \beta t_s & 0 \\ 0 & 0 & \gamma \end{bmatrix} \boldsymbol{u}$$

$$y = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{x}$$

$$\text{where, } \begin{cases} \alpha = 0.01 & \text{is the rate of heat flow between metal and air,} \\ \beta = 0.1 & \text{is the rate of heat flow between metal volumes,} \\ \gamma = 0.01 & \text{is coupling for the heater/cooler actuator.} \end{cases}$$

Note that u_1 represents ambient temperature and u_2 is the temperature of the heat-bath. These represent real inputs to our systems, but we cannot use them to apply feedback. When designing your controller, we can only vary the heater/cooler power (u_3), so we will need to use a modified $\boldsymbol{B}$ during controller design.

$$\boldsymbol{x}(t+1) = \begin{bmatrix} 0.989 & 0.01 & 0 & 1 \\ 0.01 & 0.98 & 0.01 & 0 \\ 0 & 0.01 & 0.98 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.01 \end{bmatrix} u_3 + \begin{bmatrix} 0.001 & 0 \\ 0 & 0 \\ 0 & 0.01 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

For convenience, let's define $\boldsymbol{B}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.01 \end{bmatrix}$ and $\boldsymbol{B}_2 = \begin{bmatrix} 0.001 & 0 \\ 0 & 0 \\ 0 & 0.01 \\ 0 & 0 \end{bmatrix}$.

We will simply ignore the existence of $\boldsymbol{B}_2$ during design, but use it again when building the closed loop system.

We know that we must achieve a closed loop settling time of 20 s. Recall from previous studies that the settling time is approximately four time constants for a first order system, so a continuous-time, dominant pole at $s = \frac{4}{20} = -0.2$ would achieve the required settling. We would need the other three poles to be at least a factor of three faster, say at $s = -0.6, -0.6 \pm \mathrm{j}0.6$. We can find the z-plane equivalents using the transformation $z = \mathrm{e}^{st_s}$ (or any other appropriate discretization method). In this case we obtain a desired dominant pole location of $\lambda_\mathrm{d} = \mathrm{e}^{-0.2\times 0.1} = 0.98$ and other pole locations of $\lambda = \{0.94, 0.94 \pm \mathrm{j}0.06\}$. We should check the controllability of the system using $\boldsymbol{A}$ and $\boldsymbol{B}_1$. In this case the controllability matrix is rank four, so we can proceed with our design.

```
K = place(A, B_1, [0.98 0.94 0.94+j0.06 0.94-j0.06])
S_cl = ss(A - B_1 * K, B, C, D, ts)
```

Notice that we have coupled all three inputs to the system, so we will be able to simulate the effects of environmental changes. The dynamics of the closed loop system are only modified by applying feedback to u_3, so only $\boldsymbol{B}_1\boldsymbol{K}$ alters the open loop $\boldsymbol{A}$ matrix. The Matlab command `eig(A-B_1*K)` will quickly confirm that the `place` command has correctly placed the closed loop poles in the desired location. The closed loop response of the system is shown in figure 2.11. In this case the system is driven by impulses in each of the inputs to the closed loop system ($\boldsymbol{r}$) successively. The impulses have amplitudes of 10, 1 and 0.2, and each lasts for one sampling interval.

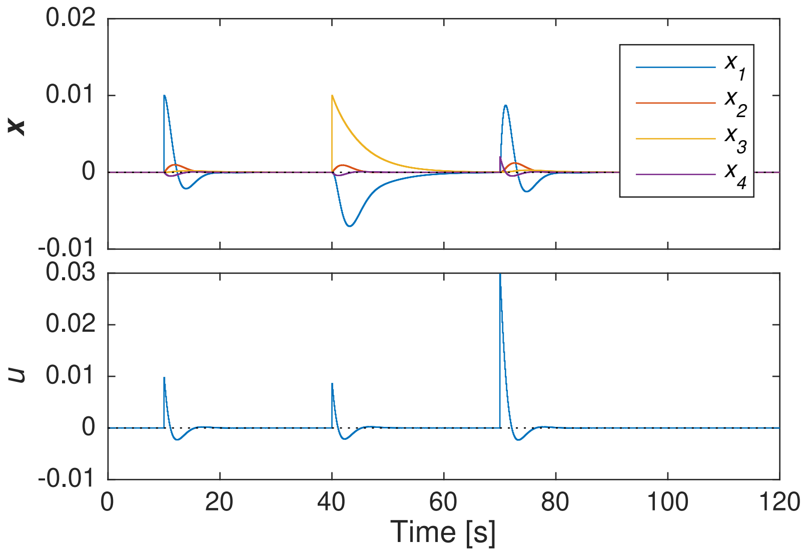


Figure 2.11: Impulse response of the closed loop system when driven by all three inputs successively. The lower panel shows the feedback signal $u(t)$.

We could repeat the design using a deadbeat controller. In this case we wish to place all four closed loop poles at $z = 0$. We will not be able to use Matlab's `place`, because it can't deal with that many colocated poles. However, as this is a SISO problem we can use `acker`.

```
K_db = acker(A, B_1, [0 0 0 0]);
S_db = ss(A - B_1 * K_db, B, C, D, ts);
```

Repeating the impulse response experiment from above yields the response shown in figure 2.12. Notice that each of the transients is complete within four sampling intervals. Though the amplitudes of the inputs in the two simulations were not identical, you can nevertheless get a sense that the control in the dead beat example is quite violent by comparison with the previous design.

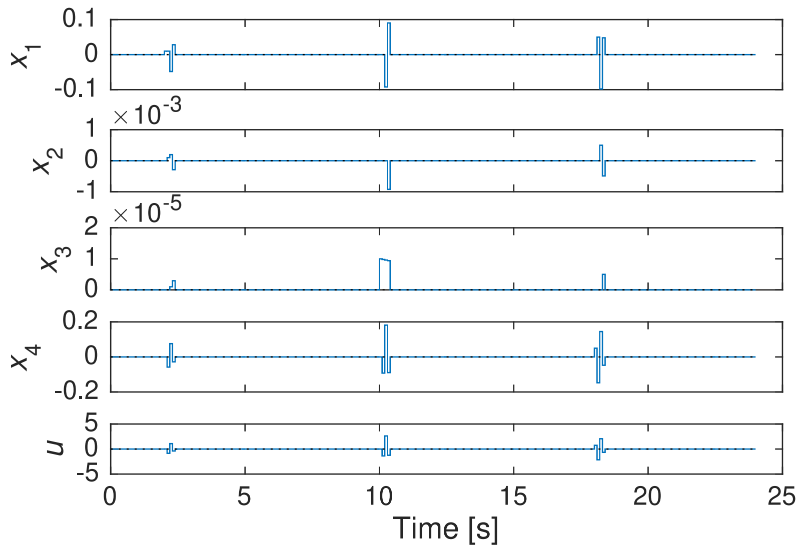


Figure 2.12: Impulse response of the deadbeat system when driven by all three inputs successively. The

2.5 Pole placement in MIMO systems

It is worth noting that `place` can find $\boldsymbol{K}$ for systems with multiple inputs and outputs. In these systems there can be more than one possible $\boldsymbol{K}$ matrix that will produce the desired pole placement. Matlab chooses a $\boldsymbol{K}$ which has low sensitivity to inaccurate implementation. So, if you use a controller generated by `place` you can be confident that it will work reasonably well in a real application.

However, there are circumstances where you might want to choose some other criteria on which to select from the possible $\boldsymbol{K}$ solutions. Common extra constraints that might be imposed on the structure of $\boldsymbol{K}$ include not applying feedback from a particular state variable (so that it need not be measured or estimated), not applying feedback to a particular actuator, minimising the number of non-zero elements of $\boldsymbol{K}$, or minimising the magnitude of the elements of $\boldsymbol{K}$. We will not cover these options in this course, in part because we will find that the LQR technique that we will look at later will allow us to tackle many of the same issues more elegantly. Nevertheless, you should be aware that there are techniques that allow those sorts of design flexibility.

As an example, let's reexamine our first example compensator design, but now add a second actuator that acts on x_2, so

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -18 & -15 & -2 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, \boldsymbol{D} = \begin{bmatrix} 0 \end{bmatrix} \quad .$$

We will design a controller that can drive the two inputs via some appropriate feedback matrix $\boldsymbol{K} = \begin{bmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \end{bmatrix}$.

We will again use the manual method of calculating the eigenvalues of the closed loop system and equating coefficients with the desired characteristic equation.

$$\begin{aligned} s^3 + 16s^2 + 39.55s + 53.26 &= \left| s\boldsymbol{I} - \boldsymbol{A} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \end{bmatrix} \right| \\ &= s^3 \\ &\quad + (k_3 + k_5 + 2)s^2 \\ &\qquad + (k_2 + k_4 + 2k_5 - 15k_6 - k_2k_6 + k_3k_5 + 15)s \\ &\qquad\quad + (k_1 + 2k_4 - 18k_6 - k_1k_6 + k_3k_4 + 18) \end{aligned}$$

We can now equate coefficients

$$\begin{cases} s^3 : 1 = 1 \\ s^2 : k_3 + k_5 + 2 = 16 \\ s^1 : k_2 + k_4 + 2k_5 - 15k_6 - k_2k_6 + k_3k_5 + 15 = 39.55 \\ s^0 : k_1 + 2k_4 - 18k_6 - k_1k_6 + k_3k_4 + 18 = 53.26 \end{cases}$$

$$\implies \begin{cases} s^3 : 1 = 1 \\ s^2 : k_3 + k_5 = 14 \\ s^1 : k_2 + k_4 + 2k_5 - 15k_6 - k_2k_6 + k_3k_5 = 24.55 \\ s^0 : k_1 + 2k_4 - 18k_6 - k_1k_6 + k_3k_4 + 18 = 35.26 \end{cases}$$

As we now have three equations in six unknowns there will be many possible solutions to this set of equations. We could make many choices for values of k_i depending on what we want to achieve.

For our example, let's imagine that we do not want to measure x_2, so we will not provide any feedback from that state variable. Therefore, let's set $k_2 = k_5 = 0$ as a starting point for $\boldsymbol{K}$.

$$\rightsquigarrow \begin{cases} k_3 = 14 \\ k_4 - 15k_6 = 24.55 \implies k_4 = 24.55 + 15k_6 \\ k_1 + 2k_4 - 18k_6 - k_1k_6 + k_3k_4 = 35.26 \end{cases}$$

Substituting the first two of these equations into the last, after a few lines of algebra we get

$$k_1 + (222 - k_1)k_6 = -357.54$$

Again we have a family of viable solutions. Let's arbitrarily choose $k_1 = 0$. We can now calculate the values for the remaining elements of $\boldsymbol{K}$.

$$\begin{aligned} k_6 &= -1.6105 \\ \implies k_4 &= 0.39189 \\ \text{and, } \boldsymbol{K} &= \begin{bmatrix} 0 & 0 & 14 \\ 0.39189 & 0 & -1.6105 \end{bmatrix} \end{aligned}$$

A check of the closed loop eigenvalues using this feedback matrix reveals that we have again achieved the required close loop pole positions, but as required we do not use feedback from x_2.

2.5.1 Design constraints using Matlab

By default Matlab will solve a MIMO state feedback controller problem to minimise sensitivity. The situation of wanting to force Matlab *not* to use some particular actuator(s) for feedback occurs reasonably often, and there is a reasonably straightforward method to use Matlab in the design of such systems.

We should recognise that the $\boldsymbol{B}$ matrix fed into Matlab's `place` command tells the algorithm what actuators are available for feedback. The key is that we need not tell Matlab about the full set of available actuators. If we do not want to apply feedback via a particular actuator, then we can simply omit the corresponding column from the $\boldsymbol{B}$ matrix when designing $\boldsymbol{K}$.

Of course when doing this we need to be careful that the system remains controllable without the actuator(s) in question.

2.6 Sampled Data Systems

In this chapter we have looked at how to design controllers for both continuous time and discrete time systems. In the case of discrete time systems the assumption is that the action of $\boldsymbol{K}$ on the state generates a feedback signal at each time instant. However, we often use a discrete time model for a system that is actually continuous time. That is, when we are dealing with sampled data systems we know that things are happening (essentially open loop) between sampling intervals. In certain circumstances we might choose to simulate the operation of the "real world" using a continuous time model, while still operating our control loop in discrete time.

We could simulate such a system as follows

$$\begin{aligned}
\boldsymbol{x}(t+1) &= \mathrm{e}^{t_\mathrm{s}\boldsymbol{A}_\mathrm{c}}\boldsymbol{x}(t) + \int_0^{t_\mathrm{s}} \mathrm{e}^{\tau\boldsymbol{A}_\mathrm{c}}\,\mathrm{d}\tau\,\boldsymbol{B}\boldsymbol{u}(t)\\
\boldsymbol{y}(t+1) &= \boldsymbol{C}\boldsymbol{x}(t+1)\\
\boldsymbol{u}(t+1) &= \boldsymbol{K}\boldsymbol{x}(t)
\end{aligned}$$

This gets more interesting if we ask what happens at non integer values of t. For example, you could choose to update what is happening with $\boldsymbol{x}$ and $\boldsymbol{y}$ more frequently than you calculate a new $\boldsymbol{u}$.

$$\begin{aligned}
\boldsymbol{x}(t+\alpha) &= \mathrm{e}^{\alpha t_\mathrm{s}\boldsymbol{A}_\mathrm{c}}\boldsymbol{x}(t) + \int_0^{\alpha t_\mathrm{s}} \mathrm{e}^{\tau\boldsymbol{A}_\mathrm{c}}\,\mathrm{d}\tau\,\boldsymbol{B}\boldsymbol{u}(t)\\
\boldsymbol{y}(t+\alpha) &= \boldsymbol{C}\boldsymbol{x}(t+\alpha)\\
&\vdots \qquad \text{repeat at time increments of } \alpha \text{ until we reach } t+1\\
\boldsymbol{u}(t+1) &= \boldsymbol{K}\boldsymbol{x}(t)
\end{aligned}$$

Doing this sort of simulation is useful if you think that your discrete time model might not capture some aspect of the continuous time system. For example, you might choose to ignore some high frequency dynamics of a system to make a simplified model for control design. In such a case performing this richer simulation will show you whether those ignored modes are likely to cause any difficulties, such as excessive ringing between samples (which would be invisible to your sampled system). If you have an adequate sampling rate it is not generally necessary to perform these mixed domain simulations.

3 Servo Problems

We have previously discussed how to choose a feedback matrix $\boldsymbol{K}$ to stabilise a system in the presence of noise or system disturbance. That problem is known as a regulator problem, where the task is to regulate the system behaviour so that it remains constant. In this section we will consider the other main class of control problem, the servo or tracking problem. The requirement here is that the system output be made to follow an external command signal (known as the *reference* signal, $\boldsymbol{r}$). Typically in a MIMO system this means that we would like $y_i = r_i$, so we would expect to have as many reference signals as there are outputs. We will see that this task can be accomplished without changing any of our previous work. We will instead add an extra element to our overall compensator to produce the required tracking behaviour.

3.1 Introducing a Reference Input

There are a variety of ways we can introduce the reference into our system. Intuitively we might consider adding the reference signal to the feedback signal; that is we might try $\boldsymbol{u} = -\boldsymbol{K}\boldsymbol{x} + \boldsymbol{r}$. This works poorly

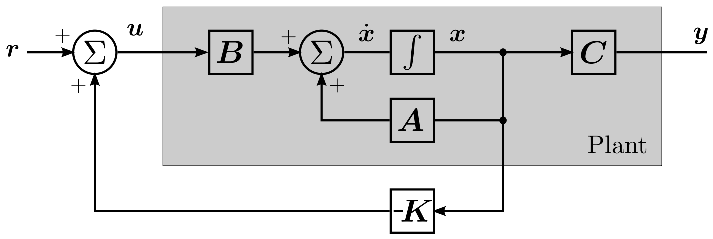


Figure 3.1: Naïve addition of a reference signal $\boldsymbol{r}$ to a state feedback controller.

in general, as it almost certainly results in a finite steady state output error. That is, if we apply a step change at $\boldsymbol{r}$, then $\boldsymbol{y}$ will change by a different amount.

3.1.1 Reference Input – example

Let's plot the step response of one of our previous examples. $\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -18 & -15 & -2 \end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$, $\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$, $\boldsymbol{D} = \begin{bmatrix} 0 \end{bmatrix}$, $\boldsymbol{K} = \begin{bmatrix} 35.26 & 24.55 & 14 \end{bmatrix}$.

```
step(ss(A,B,C,D))
hold on
step(ss(A-B*K,B,C,D))
hold off
```

The open loop and closed loop responses of this system are shown in figure 3.2 It is apparent from this figure adding feedback has altered the steady state gain of the system.

We scale the input by including an extra block between the input and the real plant. This block, $\overline{\boldsymbol{N}}$, can be used to modify the dc gain of the system as desired. In a MIMO system we will see that $\overline{\boldsymbol{N}}$ can play the additional role of untangling the geometrical relationship between the inputs and outputs. That is, because it can perform an arbitrary linear mapping it can align $\boldsymbol{r}$ and $\boldsymbol{y}$.

The block described by $\overline{\boldsymbol{N}}$ is often called a prefilter. This is a somewhat unfortunate description, as it implies that the block exhibits frequency dependent behaviour. However, in its basic form this block simply implements frequency independent gains. The language here is inherited from classical control, where a (frequency dependent) prefilter is placed in this position to provide extra design flexibility.

To find $\overline{\boldsymbol{N}}$ in the SISO case we could set $s = 0$ in the closed loop transfer function $\boldsymbol{C}\left(s\boldsymbol{I} - \boldsymbol{A} + \boldsymbol{B}\boldsymbol{K}\right)^{-1}\boldsymbol{B}$ to find the dc gain of the system. We would then set $\overline{\boldsymbol{N}}$ to add (or remove) any additional gain necessary. For this example if we wanted to match the dc gain of the open loop system we would need to add an additional gain of $\overline{\boldsymbol{N}} = 2.96$.

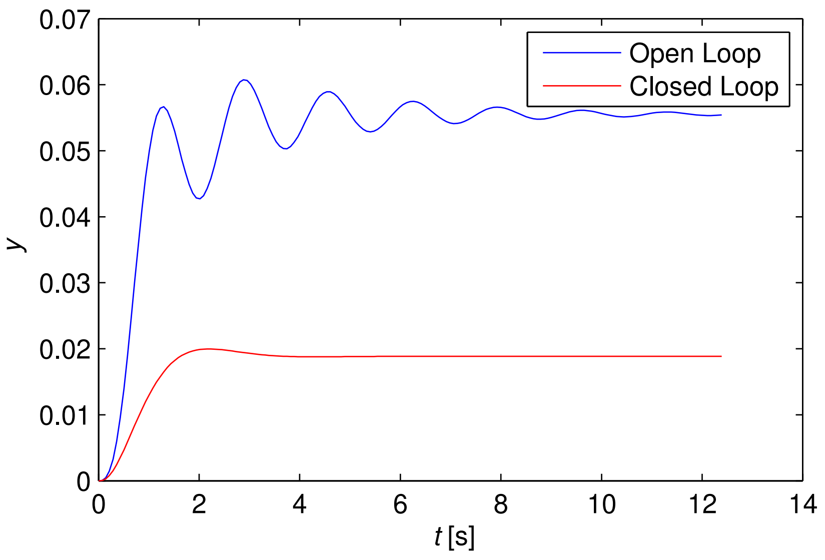


Figure 3.2: A compensated system with incorrect dc gain.

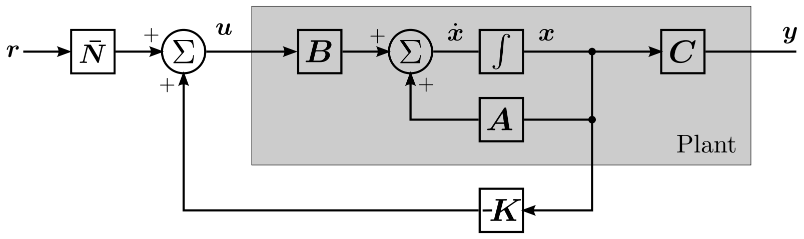


Figure 3.3: Addition of a reference signal $\boldsymbol{r}$ to a state feedback controller via a prefilter block $\overline{\boldsymbol{N}}$.

```
step(ss(A,B,C,D))
hold on
step(ss(A-B*K,B*Nbar,C,D))
hold off
```

Notice that changing the feedback gain of a system will require a compensating change on the prefilter gain. That is, changing $\boldsymbol{K}$ will force a change in $\overline{\boldsymbol{N}}$. For this reason we start a control system design by considering stability and time response (which determines $\boldsymbol{K}$), after which we turn attention to dc gain (which determines $\overline{\boldsymbol{N}}$).

3.1.2 Introducing the reference input the right way

A better way to find the required prefilter gain is to calculate the required steady state $\boldsymbol{x}$ vector that will result in an output that matches the reference. Once we know our steady state $\boldsymbol{x}$ vector ($\boldsymbol{x}_\text{o}$) we will also know the required steady state input signal u_o. (Notice that we are considering a SISO system first for simplicity, though this approach can readily be extended.) The new input signal using this scheme will be

$$u = u_\text{o} - \boldsymbol{K}(\boldsymbol{x} - \boldsymbol{x}_\text{o})$$

Now in the steady state, we should have $\dot{\boldsymbol{x}} = 0$, so our system equations can be simplified to

$$\begin{aligned} \boldsymbol{0} &= \boldsymbol{A}\boldsymbol{x}_\text{o} + \boldsymbol{B}u_\text{o} \\ y_\text{o} &= \boldsymbol{C}\boldsymbol{x}_\text{o} + Du_\text{o} \end{aligned}$$

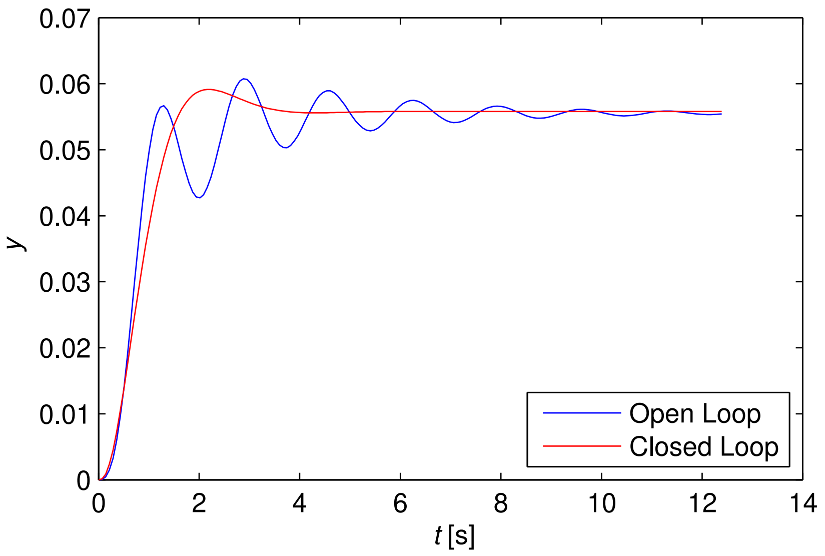


Figure 3.4: A compensated system where a prefilter has been used to correct the dc gain.

We wish to arrange a system so that $y_\mathrm{o} = r_\mathrm{o}$. We have previously examined whether we can achieve a desired operating point, so for now we will simply assume that we can find some linear combination of the state vector and inputs that will result in this condition. That is, we presume that if we choose some appropriate $\boldsymbol{N}_x$ and N_u with $\boldsymbol{x}_\mathrm{o} = \boldsymbol{N}_x r_\mathrm{o}$ and $u_\mathrm{o} = N_u r_\mathrm{o}$ then we will achieve $y_\mathrm{o} = r_\mathrm{o}$. We can now substitute these expressions into our steady state system equations.

$$\begin{aligned}
\mathbf{0} &= (\boldsymbol{A}\boldsymbol{N}_x + \boldsymbol{B}N_u)\, r_\mathrm{o} \\
y_\mathrm{o} &= (\boldsymbol{C}\boldsymbol{N}_x + \boldsymbol{D}N_u)\, r_\mathrm{o} \\
\begin{bmatrix} \mathbf{0} \\ y_\mathrm{o} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{N}_x \\ N_u \end{bmatrix} r_\mathrm{o}
\end{aligned}$$

However, at steady state we want $y_\mathrm{o} = r_\mathrm{o}$, so we can write

$$\begin{aligned}
\begin{bmatrix} \mathbf{0} \\ r_o \end{bmatrix} &= \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{N}_x \\ N_u \end{bmatrix} r_\mathrm{o} \\
\begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} &= \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{N}_x \\ N_u \end{bmatrix} \\
\implies \begin{bmatrix} \boldsymbol{N}_x \\ N_u \end{bmatrix} &= \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}
\end{aligned}$$

We can calculate $\boldsymbol{N}_x$ and N_u using this equation and use them to produce the new input for our system;

$$u_o = N_u r_\mathrm{o} - \boldsymbol{K}(\boldsymbol{x}_o - \boldsymbol{N}_x r_\mathrm{o}).$$

This equation is satisfied everyhere (not just for one particular $\boldsymbol{r}_\mathrm{o}$), so

$$\begin{aligned}
u &= N_u r - \boldsymbol{K}(\boldsymbol{x} - \boldsymbol{N}_x r) \\
&= -\boldsymbol{K}\boldsymbol{x} + (N_u + \boldsymbol{K}\boldsymbol{N}_x) r
\end{aligned}$$

We can draw the total system in the following equivalent ways. For convenience we denote $N_u + \boldsymbol{K}\boldsymbol{N}_x$ by the symbol $\overline{\boldsymbol{N}}$. While in theory these two representations are equivalent, in practice the left system proves more robust. It is fine to use the version on the right when doing design work. Use the form on the left to implement a real system.

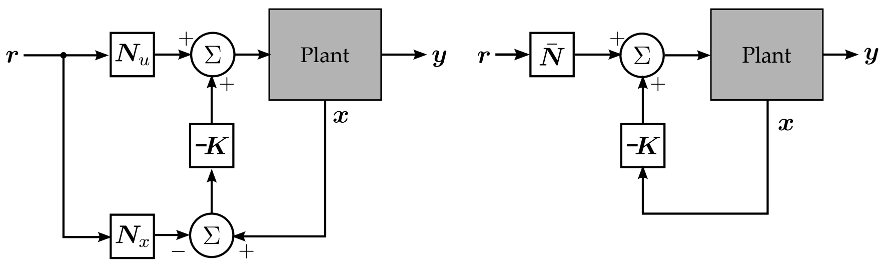


Figure 3.5: Two equivalent configurations for the prefilter. The first is preferred for implementation, though the second simpler form is fine for design and analysis.

We can generalise the above treatment to the MIMO case and also allow for non-unity gain. The derivation is almost identical, but one need to take care of the size of the zero matrices in the equations. Consider the general situation where we want a system to have a particular dc gain described by $\boldsymbol{y}_\text{o} = \boldsymbol{M}\boldsymbol{r}_\text{o}$, where $\boldsymbol{M}$ is full rank and $\boldsymbol{M} \in \mathbb{R}^{p\times p}$ for p the number of outputs (and reference inputs). We will extend the previous treatment to develop a procedure for finding $\overline{\boldsymbol{N}}$ in this case. We now begin with $\boldsymbol{y}_\text{o} = \boldsymbol{M}\boldsymbol{r}_\text{o}$ and presume there exists $\boldsymbol{N}_x$ and $\boldsymbol{N}_u$ such that $\boldsymbol{x}_\text{o} = N_x\boldsymbol{r}_\text{o}$ and $\boldsymbol{u}_\text{o} = \boldsymbol{N}_u\boldsymbol{r}_\text{o}$. Note that $\boldsymbol{N}_u \in \mathbb{R}^{p\times p}$.

$$\begin{cases} \boldsymbol{0}_{n\times 1} &= (\boldsymbol{A}\boldsymbol{N}_x + \boldsymbol{B}\boldsymbol{N}_u)\boldsymbol{r}_\text{o} \\ \boldsymbol{y}_0 &= (\boldsymbol{C}\boldsymbol{N}_x + \boldsymbol{D}\boldsymbol{N}_u)\boldsymbol{r}_\text{o} \end{cases}$$

$$\begin{bmatrix} \boldsymbol{0}_{n\times 1} \\ \boldsymbol{y}_\text{o} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{N}_x \\ \boldsymbol{N}_u \end{bmatrix} \boldsymbol{r}_\text{o}$$

$$\begin{bmatrix} \boldsymbol{0}_{n\times 1} \\ \boldsymbol{M}\boldsymbol{r}_\text{o} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{N}_x \\ \boldsymbol{N}_u \end{bmatrix} \boldsymbol{r}_\text{o}$$

$$\begin{bmatrix} \boldsymbol{0}_{n\times p} \\ \boldsymbol{M} \end{bmatrix} \boldsymbol{r}_\text{o} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{N}_x \\ \boldsymbol{N}_u \end{bmatrix} \boldsymbol{r}_\text{o}$$

$$\begin{bmatrix} \boldsymbol{0}_{n\times p} \\ \boldsymbol{M} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{N}_x \\ \boldsymbol{N}_u \end{bmatrix}$$

$$\begin{bmatrix} \boldsymbol{N}_x \\ \boldsymbol{N}_u \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{0}_{n\times p} \\ \boldsymbol{M} \end{bmatrix}$$

The same equation for finding u_o in the SISO case then applies.

$$\begin{aligned} \boldsymbol{u}_\text{o} &= -\boldsymbol{K}\boldsymbol{x} + (\boldsymbol{N}_u + \boldsymbol{K}\boldsymbol{N}_x)\boldsymbol{r} \\ &= -\boldsymbol{K}\boldsymbol{x} + \overline{\boldsymbol{N}}\boldsymbol{r} \end{aligned}$$

$\overline{\boldsymbol{N}}$ maps the input vector $\boldsymbol{r}$ to be aligned with $\boldsymbol{y}$ and then scales it as necessary. This method is useful because it allows us to choose the dc effect that each input variable has on the various outputs.

In practice the design of a servo controller can be broken into two parts:

1. Design the feedback gain $\boldsymbol{K}$ using any desired method. The aim of this part of the design is to achieve the desired dynamic performance for the system.
2. Choose $\overline{\boldsymbol{N}}$ so that the dc gain of the system is correct. In the SISO case you could use a calculation of the dc gain, however, it is simpler (and necessary for a MIMO system) to use $\begin{bmatrix} \boldsymbol{N}_x \\ \boldsymbol{N}_u \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{M} \end{bmatrix} \implies \overline{\boldsymbol{N}} = \boldsymbol{N}_u + \boldsymbol{K}\boldsymbol{N}_x$

Note that in the somewhat typical case where we want $\boldsymbol{y} = \boldsymbol{r}$ we set $\boldsymbol{M} = \boldsymbol{I}_m$. The size of the identity matrix used must match the number of inputs (and outputs) to the system.

We previously saw a method for finding a $\boldsymbol{u}_\mathrm{o}$ required to produce a desired $\boldsymbol{y}_\mathrm{o}$. The method developed here can be considered complimentary to that used before. The previous method left the system unaltered and found a $\boldsymbol{u}_\mathrm{o}$ signal to achieve $\boldsymbol{x}_\mathrm{o}$ (and hence $\boldsymbol{y}_\mathrm{o}$). In this method we instead specify a desired $\boldsymbol{r}_\mathrm{o}$ that we want to produce $\boldsymbol{x}_\mathrm{o}$ and then alter the system by adding $\overline{\boldsymbol{N}}$.

3.2 Integral Control

The system we have described above will ideally produce zero steady state error when presented with a fixed input. However, that method requires careful tuning to work perfectly;

- Our model of the plant must be correct ($\boldsymbol{A}$,$\boldsymbol{B}$,$\boldsymbol{C}$,$\boldsymbol{D}$ must be accurate),
- The various gains that we apply must be implemented quite accurately, so we need to worry about things like numerical precision.

The deficiencies of the method should not be surprising given that it is an open loop method, and as such is sensitive to modelling errors. As an alternative to introducing $\overline{\boldsymbol{N}}$, we can build a more robust system by introducing integral control. Recall that we used integral control to drive steady state errors to zero in classical control systems. (We could also extend our treatment to add additional integrator stages to allow perfect tracking of input ramps, parabaloids or higher order polynomials).

We know that the action of a regulator is to drive all of the state variables to zero. It seems reasonable then to add a new state variable that will represent the integral of the difference between the output and our reference signal (ie, the integral of the output error). The compensator will then drive the integral term to zero by ensuring that the output is the same as the reference. For simplicity we will assume that we wish to introduce integral control to a single output variable, though the treatment extends naturally to multiple outputs. We will denote the new state variable as x_i, which is defined by $x_\mathrm{i}(t) = \int_0^t \big(r(\tau) - y(\tau)\big)\,\mathrm{d}\tau$. We would like to augment our normal state space model with x_i, so we need to find a way to express its rate of change as a function of present state and the system input.

$$\begin{aligned} \dot{x}_\mathrm{i} &= r - y \\ &= r - \boldsymbol{C}\boldsymbol{x} \end{aligned}$$

We thus have an augmented system described by the equations

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \\ \dot{x}_\mathrm{i} &= -\boldsymbol{C}\boldsymbol{x} + r \\ y &= \boldsymbol{C}\boldsymbol{x} \end{aligned}$$

We can write this more succinctly as

$$\begin{aligned} \begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{x}_\mathrm{i} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ -\boldsymbol{C} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ x_\mathrm{i} \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} \\ 0 \end{bmatrix} u + \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix} r \\ y &= \begin{bmatrix} \boldsymbol{C} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ x_\mathrm{i} \end{bmatrix} \end{aligned}$$

We now also need a larger feedback gain matrix so we can feed back from the additional Integral) state,

$$\begin{aligned} u &= -\boldsymbol{K}\boldsymbol{x} - k_\mathrm{i} x_\mathrm{i} \\ &= -\begin{bmatrix} \boldsymbol{K} & k_\mathrm{i} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ x_\mathrm{i} \end{bmatrix} \end{aligned}$$

We can now solve for $\boldsymbol{K}$ and k_i simultaneously by any of the methods that we have previously employed. The matrices used for this are the *augmented* $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices, $\left[\begin{smallmatrix} \boldsymbol{A} & 0 \\ -\boldsymbol{C} & 0 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} \boldsymbol{B} \\ 0 \end{smallmatrix}\right]$. We do not use the $\boldsymbol{r}$ input to apply feedback, so it does not appear in these modified system matrices. Remember that the system now has an extra pole to place. Substituting our equation for u into the system equation we find the new closed loop system.

$$
\begin{aligned}
\begin{bmatrix}\dot{\boldsymbol{x}}\\ \dot{x}_\mathrm{i}\end{bmatrix} &= \left(\begin{bmatrix}\boldsymbol{A} & \boldsymbol{0}\\ -\boldsymbol{C} & 0\end{bmatrix} - \begin{bmatrix}\boldsymbol{B}\\ 0\end{bmatrix}\begin{bmatrix}\boldsymbol{K} & k_\mathrm{i}\end{bmatrix}\right)\begin{bmatrix}\boldsymbol{x}\\ x_\mathrm{i}\end{bmatrix} + \begin{bmatrix}\boldsymbol{0}\\ 1\end{bmatrix} r\\
\begin{bmatrix}\dot{\boldsymbol{x}}\\ \dot{x}_\mathrm{i}\end{bmatrix} &= \left(\begin{bmatrix}\boldsymbol{A} & \boldsymbol{0}\\ -\boldsymbol{C} & 0\end{bmatrix} - \begin{bmatrix}\boldsymbol{BK} & \boldsymbol{B}k_\mathrm{i}\\ 0 & 0\end{bmatrix}\right)\begin{bmatrix}\boldsymbol{x}\\ x_\mathrm{i}\end{bmatrix} + \begin{bmatrix}\boldsymbol{0}\\ 1\end{bmatrix} r\\
\begin{bmatrix}\dot{\boldsymbol{x}}\\ \dot{x}_\mathrm{i}\end{bmatrix} &= \begin{bmatrix}\boldsymbol{A}-\boldsymbol{BK} & -\boldsymbol{B}k_\mathrm{i}\\ -\boldsymbol{C} & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{x}\\ x_\mathrm{i}\end{bmatrix} + \begin{bmatrix}\boldsymbol{0}\\ 1\end{bmatrix} r\\
y &= \begin{bmatrix}\boldsymbol{C} & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{x}\\ x_\mathrm{i}\end{bmatrix}
\end{aligned}
$$

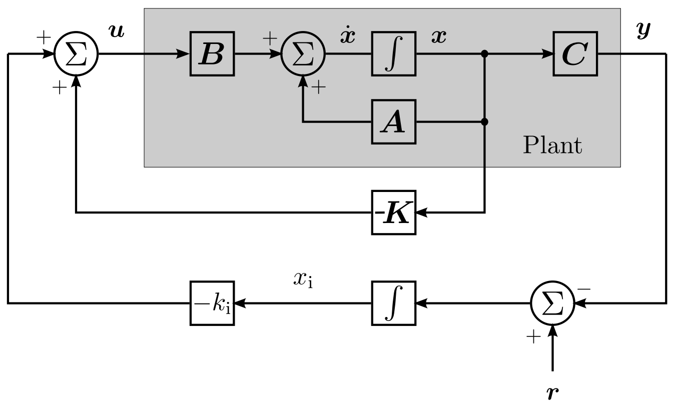


Figure 3.6: Block diagram of a state feedback system including integral control.

3.2.1 Integral control – example

Consider the example that we have previously seen. For comparison we begin by calculating a nominal value of $\overline{\boldsymbol{N}}$ for our example.

```
N = inv([A B; C D])*[zeros(length(A),1); 1];
Nbar = N(end) + K*N(1:end-1);
```

This yields $\overline{\boldsymbol{N}} = 53.05$. If we use this as a prefilter we achieve unity gain as expected. However, if we have the wrong model, then we find that the steady state gain is incorrect. For comparison, let's see the effect of using our nominal $\overline{\boldsymbol{N}}$ with $\boldsymbol{A} = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ -21 & -14 & -2.2\end{bmatrix}$ instead of $\boldsymbol{A}_\mathrm{nominal} = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ -18 & -15 & -2\end{bmatrix}$.

The resulting curve can be seen in figure 3.7. Notice that output does not reach the correct final value of $y = 1$.

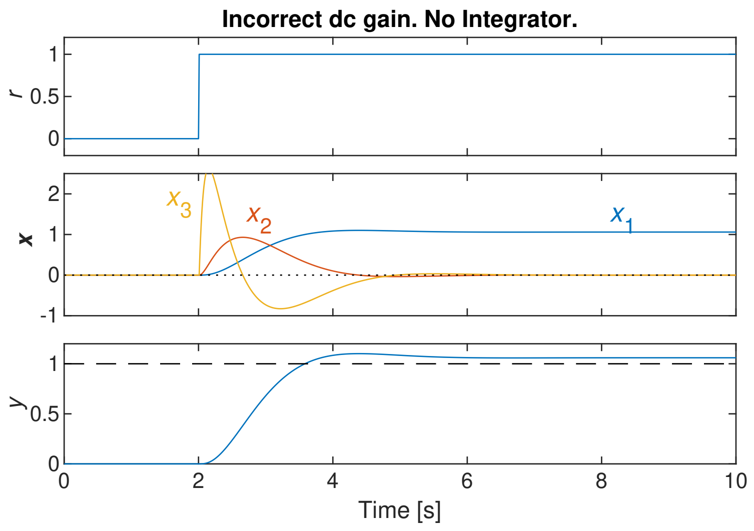


Figure 3.7: Step response of the system when the plant has been perturbed away from the nominal model (that used to design $\overline{\boldsymbol{N}}$).

We would like to add an integrator to remove the error. Begin by adding the extra state.

```
A_i = [A zeros(length(A),1) ; -C 0];
B_i = [B ; 0];
```

We next add an extra pole at $s = -10$ and calculate a new feedback gain matrix, $\boldsymbol{K}_\mathrm{i} = \begin{bmatrix}\boldsymbol{K} & k_\mathrm{i}\end{bmatrix}$.

```
alpha_c= [-1.33+1.49j, -1.33-1.49j, -13.3, -10];
K_i= place(A_i, B_i, alpha_c);
```

Finally, form the new closed loop system.

```
S_i = ss(A_i - B_i * K_i, [zeros(size(B)); 1], [C 0], D);
```

The dc gain is now correct ($y = r$ at steady state) as shown in figure 3.8.

The integrator works well, but it does require that the integrator state build up some error before it can influence the state of the system. We can often do a little better by combining the prefilter approach with an integrator. The prefilter attempts to set the dc gain, and then any remaining errors are removed by the integrator. This can be implemented by simply including the $\boldsymbol{B}\overline{\boldsymbol{N}}$ term into the input of the new closed loop system.

```
S_i = ss(A_i - B_i * K_i, [B*Nbar; 1], [C 0], D);
```

In figure 3.9 we can see that the new system responds slightly quicker. In addition x_4 (the integrator state) does not need to work quite as hard at steady state.

It may appear that it is always a good idea to use this prefilter feedforward. However, one advantage of leaving it out is that the reference signal is effectively filtered as it passes through the integrator. The advantage of this can be seen in the comparison figure 3.9, where the state variables deviate much more dramatically when the prefilter block is not included. If you have a reference signal that is noisy, or changes very rapidly, then it may be preferable not to use the prefilter (or to use a prefilter that does not apply the full gain, in a scheme called *setpoint weighting*). Simulation of the situation in this type of scenario is a good idea.

We can extend our treatment to place integrators on multiple input/output pairs. The system shown in figure 3.11 shows how this can be done in practice (with the assumption that we want to make $\boldsymbol{y} = \boldsymbol{r}$). In this diagram $-\boldsymbol{K}$ describes the total feedback from the augmented state $\begin{bmatrix}\boldsymbol{x} & \boldsymbol{x}_\mathrm{i}\end{bmatrix}^\mathsf{T}$.

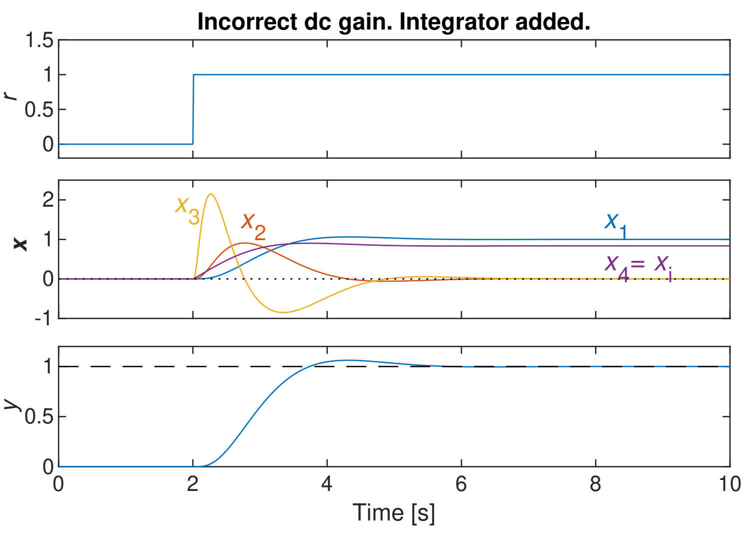


Figure 3.8: Step response of the system when integral control is included. The system was perturbed away from the nominal model.

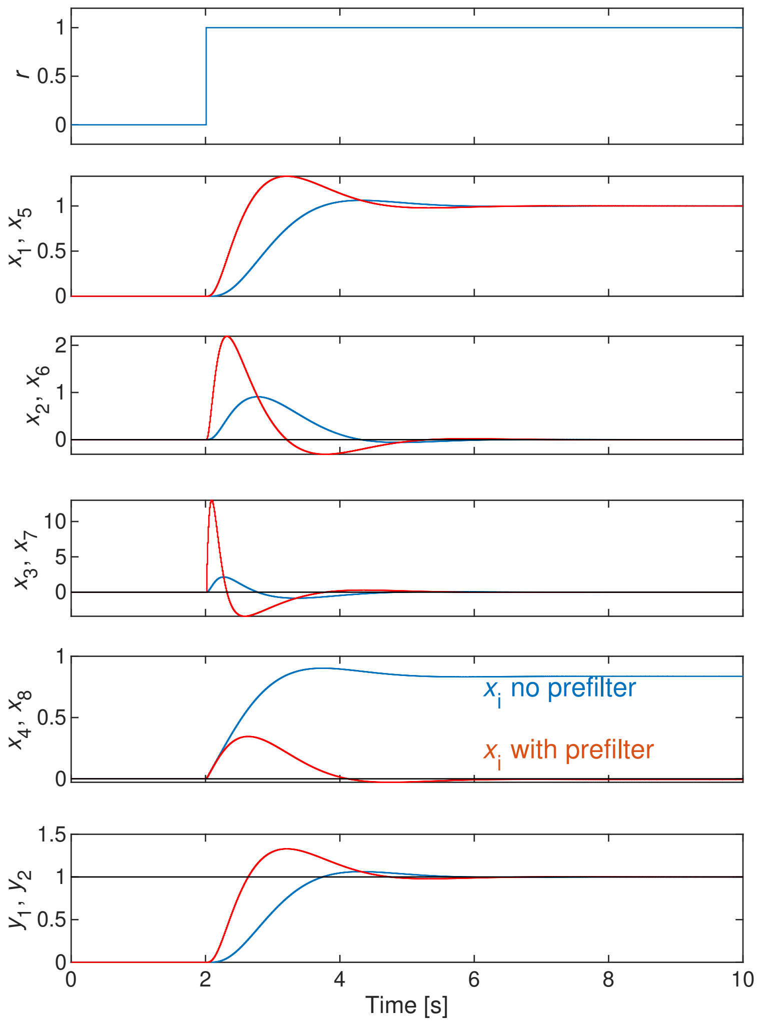


Figure 3.9: Comparison of the integrator system performance with and without the feedforward block $\overline{\boldsymbol{N}}$ in use. The solid blue lines show the performance of the integrator system, while the red lines show the change when the feedforward prefilter is added.

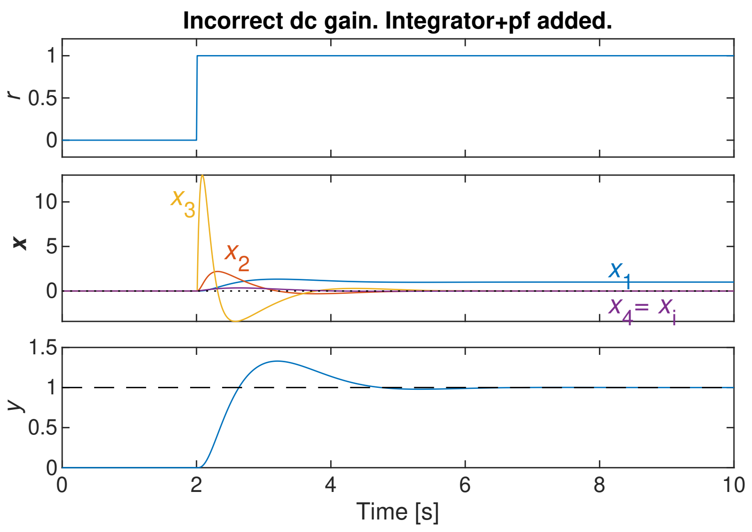


Figure 3.10: Step response of the system when integral control is included, along with the appropriate prefilter block $\overline{\boldsymbol{N}}$.

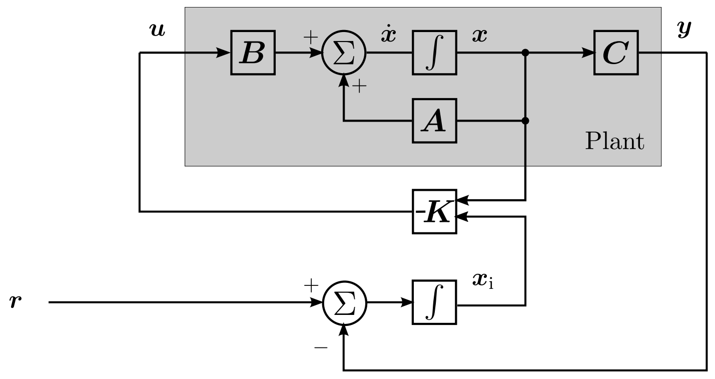


Figure 3.11: Block diagram of a state feedback system including integral control for a MIMO system.

3.3 Integral Control in Discrete Time

We can also perform integral control for a discrete time system. Consider the case where again we augment a SISO system with a new state $\boldsymbol{x}_\mathrm{i}$ which is intended to drive the system output to match its input. The evolution of the integrator state is governed by

$$\begin{aligned} x_\mathrm{i}(t+1) &= x_\mathrm{i}(t) + r - y \\ &= x_\mathrm{i}(t) + r - C\boldsymbol{x} \end{aligned}$$

So, we can form an augmented system of

$$\begin{aligned} \begin{bmatrix} \boldsymbol{x}(t+1) \\ x_\mathrm{i}(t+1) \end{bmatrix} &= \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ -\boldsymbol{C} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(t) \\ x_\mathrm{i}(t) \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} \\ 0 \end{bmatrix} u(t) + \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix} r(t) \\ y &= \begin{bmatrix} \boldsymbol{C} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ x_\mathrm{i} \end{bmatrix} \end{aligned}$$

Note the "1" in the augmented $\boldsymbol{A}$ matrix, where there was a "0" in the continuous time case. With this exception of this entry, the design process is identical for the discrete and continuous time cases.

4 State Observers

When we developed our compensator design techniques we assumed that we had full information to the state vector. That is, we assumed that $\boldsymbol{x}$ was available so that we could use it to generate feedback.

In practice there are number of reasons that we might not have measurements of the entire state vector.

- It may be impossible to measure some state variables.
- A fully instrumented system may be very expensive.
- A measurement might be too noisy.

In this section we consider how to overcome this problem by using an estimate of the state vector in place of the real data when we build our control system. The subsystem that produces the estimate is known as a *Luenberger observer*, or more generally a *state estimator*.

Observer design requires that we have a good model of the system. In other words, we must know a reasonably accurate set of system matrices $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$. However, we will see that we don't need to have a perfect model as we can use feedback to make the observer somewhat insensitive to modelling errors. For the purposes of this treatment we will assume $\boldsymbol{D} = 0$. Should you encounter a situation where it is not, the extension can be found in numerous texts. $\boldsymbol{D} \neq 0$ doesn't change things much, but it cumbersome to carry around in the mathematical treatment.

We have a number of questions to consider.

- Is it always possible to estimate the entire state of a system?
- How might we best estimate the state?
- How can we design an observer?

4.1 Observability

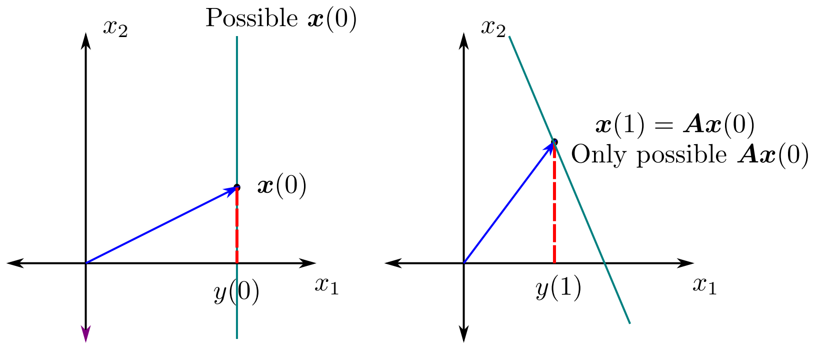
The basic premise of an observer is that we use the system output $\boldsymbol{y}$ to infer the state $\boldsymbol{x}$. We should first check whether it is possible to build an observer for our system before we attempt it. That is we need to determine whether our system structure allows enough information about $\boldsymbol{x}$ to reach $\boldsymbol{y}$. At first glance we might suspect that we will only be able to determine the state $\boldsymbol{x}(t)$ from $\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t)$ if we could solve $\boldsymbol{x} = \boldsymbol{C}^{-1}\boldsymbol{y}$. This would require as many outputs as there are states, as $\boldsymbol{C}$ would need to be square and full rank. We could also go further and take multiple measurements of the system, so that $\boldsymbol{C}$ has more rows than there are states. We could then use the pseudoinverse to infer $\boldsymbol{x}$ while reducing the effect of measurement noise (this is just least square fitting). That is, we (correctly) determine that $\boldsymbol{y}(t_1)$ is observable from $\boldsymbol{x}(t_1) \in \mathbb{R}^n$ if $\boldsymbol{C} \in \{\mathbb{R}^{n\times n} : \mathcal{R}\boldsymbol{C} = \mathbb{R}^n\}$. This is not very profound. It simply tells us that we can determine all of the state variables if we measure them.

However, we are *not* required to infer $\boldsymbol{x}$ from a single measurement of $\boldsymbol{y}$. We can watch the trajectory of $\boldsymbol{y}$ and then use *multiple* $\boldsymbol{y}$ values to untangle the behaviour of $\boldsymbol{x}$. Consider the trajectory of $\boldsymbol{y}$ produced by a non-zero initial state $\boldsymbol{x}(0)$ in an autonomous, discrete time system.

$$\begin{aligned}
\boldsymbol{y}(0) &= \boldsymbol{C}\boldsymbol{x}(0) \\
\boldsymbol{y}(1) &= \boldsymbol{C}\boldsymbol{x}(1) = \boldsymbol{C}\boldsymbol{A}\boldsymbol{x}(0) \\
\boldsymbol{y}(2) &= \boldsymbol{C}\boldsymbol{x}(2) = \boldsymbol{C}\boldsymbol{A}^2\boldsymbol{x}(0) \\
&\vdots \\
\boldsymbol{y}(k) &= \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{x}(0)
\end{aligned}$$

Each successive sample *could* give us more information about $\boldsymbol{x}(0)$ until we reach the $(n-1)$-th state. In other words, the effect of $\boldsymbol{x}(0)$ is "smeared" over $n-1$ time steps as a consequence of the convolution property of LTI systems.

The schematic in figure 4.1 shows an example operating in $\mathbb{R}^2$ with $\boldsymbol{C} = \begin{bmatrix}1 & 0\end{bmatrix}$. At each time step we can see only the $x_1(t)$ component directly, but if we know $\boldsymbol{A}$ then knowing $y(1)$ allows us to work backwards to infer $x_2(0)$.



x_2
Possible $\boldsymbol{x}(0)$
$\boldsymbol{x}(0)$
$y(0)$
x_1
x_2
$\boldsymbol{x}(1) = \boldsymbol{A}\boldsymbol{x}(0)$
Only possible $\boldsymbol{A}\boldsymbol{x}(0)$
$y(1)$
x_1

Figure 4.1: Evolution of the output as time passes. The value of $y(1)$ in this case reveals the value of $x_2(0)$. The blue vector points to the true value of $\boldsymbol{x}(0)$, while the teal line shows the set of possible $\boldsymbol{x}(0)$ values that are consistent with $y(0)$.

We can write the part of the output trajectory that contains information about $\boldsymbol{x}(0) \in \mathbb{R}^n$ as

$$\begin{bmatrix} \boldsymbol{y}(0) \\ \boldsymbol{y}(1) \\ \boldsymbol{y}(2) \\ \vdots \\ \boldsymbol{y}(n-1) \end{bmatrix} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{C}\boldsymbol{A} \\ \boldsymbol{C}\boldsymbol{A}^2 \\ \vdots \\ \boldsymbol{C}\boldsymbol{A}^{n-1} \end{bmatrix} \boldsymbol{x}(0)$$
$$:= \boldsymbol{\mathcal{M}}_{\mathrm{o}} \boldsymbol{x}(0)$$

where $\boldsymbol{\mathcal{M}}_{\mathrm{o}}$ is the *observability matrix*, which is defined by the above expression for a linear time invariant system.

The observability matrix is dual to the controllability matrix we encountered earlier. The observability matrix $\boldsymbol{\mathcal{M}}_{\mathrm{o}}$ must be full rank for our system to be *observable*. That is, if the observability matrix is full rank then if we watch the syste for long enough we can infer the state fully. In the case of a single output system, $\boldsymbol{\mathcal{M}}_{\mathrm{o}}$ will be square, which leads to a requirement for $\det \boldsymbol{\mathcal{M}}_{\mathrm{o}} \neq 0$ for full observability. You should always test for observability before designing an observer. Matlab's `obsv` command may prove useful. Sometimes you can see that a system is unobservable by inspection. For example, if a system is in modal form, then a zero column in the $\boldsymbol{C}$ matrix corresponds with a state variable that is not directly observable. That is, that state variable does not couple to any of the output variables. (However, be careful as it may couple to some other state which *is* observable).

4.1.1 Unobservable systems

A system is unobservable with one or more of its state variables are not coupled to the output variables. As they do not affect the output we cannot infer their behaviour by looking solely at the output.

This structure is illustrated in figure 4.2, in which we have divided the state vector into two parts $\boldsymbol{x} := \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 \end{bmatrix}^{\mathsf{T}}$. The various system matrices have also been partitioned to make clear the portions that act on $\boldsymbol{x}_1$ and those that act on $\boldsymbol{x}_2$. So, $\boldsymbol{A} := \begin{bmatrix} \boldsymbol{A}_{11} & 0 \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix}$, $\boldsymbol{B} := \begin{bmatrix} \boldsymbol{B}_1 \\ \boldsymbol{B}_2 \end{bmatrix}$ and $\boldsymbol{C} := \begin{bmatrix} \boldsymbol{C}_1 & 0 \end{bmatrix}$. Notice that $\boldsymbol{x}_2$ cannot affect $\boldsymbol{y}$, so the plant is unobservable.

As was the case for controllability, a mathematical test for observability does not mean that all state variables can be inferred in reality. If the coupling of a state variable to the system output is very weak it can prove unrealistic to try to extract its value. As was the case for controllability, we could use PBH or the (observability) Gramian to see how close to unobservable a system might be. This lets us determine which state variables we might have trouble inferring.

If a system is unobservable you can sometimes redesign the system itself. For example, it is often possible to add extra sensors to gather more information about the state.

An unobservable system may still be acceptable for use if the unobservable modes are sufficiently well behaved. If all of the unobservable modes of a system decay to zero asymptotically (are stable),

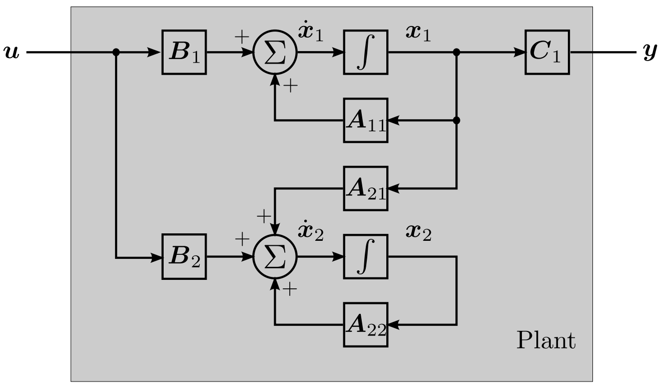


Figure 4.2: Graphical representation of an unobservable system. In this system the subset of state variables represented by $\boldsymbol{x}_2$ are not connected to the output, either directly or indirectly (though $\boldsymbol{x}_1$). As a result looking at $\boldsymbol{y}$ does not allow inference of the behaviour of $\boldsymbol{x}_2$.

then a system is said to be *detectable.* If those modes are sufficiently fast then we could just assume the corresponding state variables to be zero and proceed with our design.

4.2 Observer Design

We know that the evolution of our plant is described by the state equation $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$. We now seek to construct a model of the system $\dot{\hat{\boldsymbol{x}}} = \widehat{\boldsymbol{A}}\hat{\boldsymbol{x}} + \widehat{\boldsymbol{B}}\boldsymbol{u}$, where we have used $\hat{\boldsymbol{x}}$ to denote our *estimate* of the state vector. When necessary we will use $\widehat{\boldsymbol{A}}$, $\widehat{\boldsymbol{B}}$, $\widehat{\boldsymbol{C}}$ to clarify when we are referring to the observer's model of our system rather than the true system. However, for simplicity we will assume a perfect model ($\widehat{\boldsymbol{A}} = \boldsymbol{A}$ and so forth) for much of the time. Note that we assume that we have access to the true $\boldsymbol{u}$ signal. We define $\tilde{\boldsymbol{x}}$ as the error in our estimate, that is the error is the difference between the real state vector and our estimate of it; $\tilde{\boldsymbol{x}} := \boldsymbol{x} - \hat{\boldsymbol{x}}$. Our task in designing an observer is to make $\tilde{\boldsymbol{x}}$ as small as possible. We also extend this notation to $\hat{\boldsymbol{y}}$ and $\tilde{\boldsymbol{y}}$ in the natural way.

4.2.1 Open loop observer

One possible way to build an observer is to run a modelled version of the system in parallel with the real system. If the system is stable (and we have a sensible model) then this observer will converge to the real state vector. There are however, a couple of problems.

- The model and the observer are likely to have different initial conditions. There can be large discrepancies between the estimate and reality while the transient decays.
- We have no way of tailoring the response of the observer to suit our application.

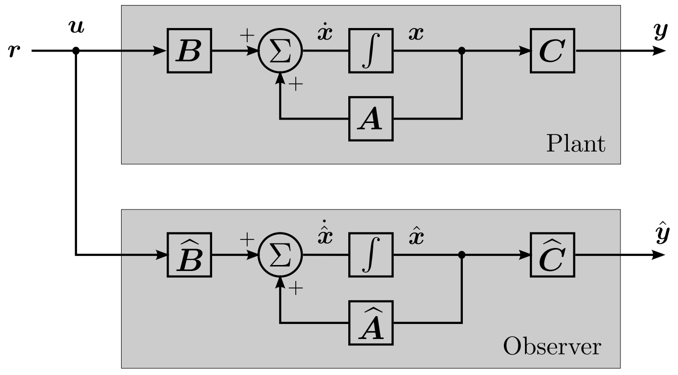


Figure 4.3: Plant with an open loop observer.

4.2.2 Closed loop observer

The open loop observer provides an estimate of the output $\hat{\boldsymbol{y}}$. We can improve performance if we find the difference between this estimate and the real output and feed this back to the observer. That is, we measure $\tilde{\boldsymbol{y}} = \boldsymbol{y} - \hat{\boldsymbol{y}}$ and feed it back to the input of the observer via gain $\boldsymbol{L}$.

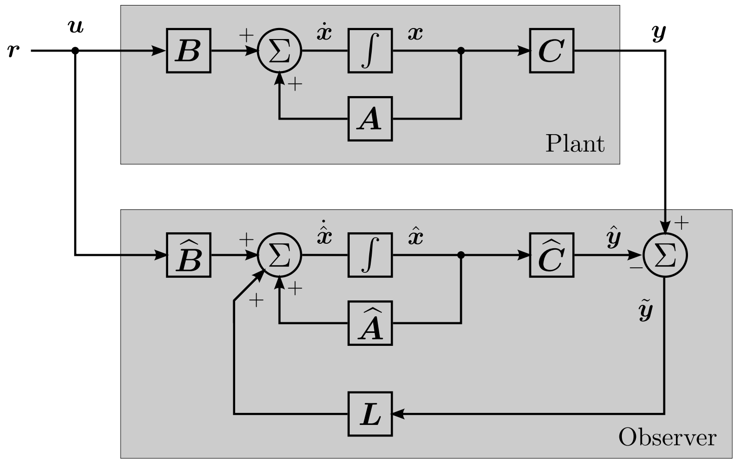


Figure 4.4: Plant with a closed loop observer.

As with all negative feedback systems, the effect of this is that the system will attempt to drive the error signal to zero. That is the feedback will try to make $\tilde{\boldsymbol{y}}$ zero. The only way this can be done is to make $\hat{\boldsymbol{y}} = \boldsymbol{y}$ (and presumably $\hat{\boldsymbol{x}} = \boldsymbol{x}$), so the presence of feedback makes our estimate better.

In the open loop case our observer dynamics are governed by $\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{u}$. In the closed loop case we must include the effects of feedback, so we replace $\boldsymbol{B}\boldsymbol{u}$ with $\boldsymbol{B}\boldsymbol{u} + \boldsymbol{L}\tilde{\boldsymbol{y}}$. Thus we have

$$\begin{aligned}
\dot{\hat{\boldsymbol{x}}} &= \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{L}\tilde{\boldsymbol{y}} \\
&= \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{L}\boldsymbol{y} - \boldsymbol{L}\hat{\boldsymbol{y}} \\
&= \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{L}\boldsymbol{C}\boldsymbol{x} - \boldsymbol{L}\boldsymbol{C}\hat{\boldsymbol{x}} \\
&= \boldsymbol{L}\boldsymbol{C}\boldsymbol{x} + (\boldsymbol{A} - \boldsymbol{L}\boldsymbol{C})\hat{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{u}
\end{aligned}$$

We are also interested in the behaviour of $\tilde{\boldsymbol{x}}$, as want to drive this to zero. Let's subtract the equation for $\hat{\boldsymbol{x}}$ from that for $\boldsymbol{x}$.

$$\begin{aligned}
\dot{\tilde{\boldsymbol{x}}} := \dot{\boldsymbol{x}} - \dot{\hat{\boldsymbol{x}}} &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} - (\boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{L}\boldsymbol{C}\boldsymbol{x} - \boldsymbol{L}\boldsymbol{C}\hat{\boldsymbol{x}}) \\
&= \boldsymbol{A}\tilde{\boldsymbol{x}} - \boldsymbol{L}\boldsymbol{C}\tilde{\boldsymbol{x}} \\
\dot{\tilde{\boldsymbol{x}}} &= (\boldsymbol{A} - \boldsymbol{L}\boldsymbol{C})\,\tilde{\boldsymbol{x}}
\end{aligned}$$

As the behaviour of our state error is described by $\dot{\tilde{\boldsymbol{x}}} = (\boldsymbol{A} - \boldsymbol{L}\boldsymbol{C})\tilde{\boldsymbol{x}}$, we can find the modes of the estimate error signal by solving the characteristic equation,

$$\det(s\boldsymbol{I} - (\boldsymbol{A} - \boldsymbol{L}\boldsymbol{C})) = \det(s\boldsymbol{I} - \boldsymbol{A} + \boldsymbol{L}\boldsymbol{C}).$$

This equation is very similar to that which we encountered while designing compensators. This time we can control the location of the observer poles by choosing $\boldsymbol{L}$ appropriately. This allows us to set how quickly the estimator will converge to the true state. We choose the appropriate value for $\boldsymbol{L}$ using the same range of techniques we used for $\boldsymbol{K}$ earlier.

We can use any of the methods described in the section on compensator design to determine $\boldsymbol{L}$; we can solve directly, convert to observer canonical form and solve by inspection, or alternatively we can use Ackermann's formula.

In this case we express Ackermann's equation as

$$\boldsymbol{L} = \alpha_{\mathrm{e}}(\boldsymbol{A})\boldsymbol{\mathcal{M}}_{\mathrm{o}}^{-1}\begin{bmatrix}0\\0\\\vdots\\1\end{bmatrix}$$

where χ_{e} is the desired closed loop polynomial. The roots of α_{e} are the poles that govern the modes of the estimator error.

The compensator and observer design problems are obviously very similar, but not identical. In one we seek to position the poles of $\boldsymbol{A}-\boldsymbol{BK}$, in the other $\boldsymbol{A}-\boldsymbol{LC}$. In particular notice the position of $\boldsymbol{B}/\boldsymbol{C}$ is different in the two.

We can make the equations look more similar if we recognise that the poles of $\boldsymbol{A}-\boldsymbol{LC}$ will be the same as those of $(\boldsymbol{A}-\boldsymbol{LC})^{\mathsf{T}}$. Now $(\boldsymbol{A}-\boldsymbol{LC})^{\mathsf{T}} = \boldsymbol{A}^{\mathsf{T}} - \boldsymbol{C}^{\mathsf{T}}\boldsymbol{L}^{\mathsf{T}}$.

We now have the observer equation in exactly the same form as the compensator equation (so long as we transpose the matrices before and after). In matlab for example, we could find $\boldsymbol{L}$ with

```
L = place(A', C', p_o)'
```

Here `p_o` is a matrix containing the desired closed loop observer poles.

We considered a number of approaches to placement of the compensator poles. We have a similar set of options now, though there are a couple of extra considerations.

- We don't need to worry as much about control effort as we don't need to control a real plant, just the model in the observer. Large signals in an observer are just large numbers in a computer.
- The observer poles must be faster than the compensator poles, so that the estimate becomes meaningful before the compensator acts on it.

The requirement that the observer be faster than the compensator translates to a requirement to place the poles further to the left in the s-plane (or closer to the origin in the z-plane). However, you should be wary of placing them too far to the left (or centre), as this increases the bandwidth of the system and makes it more susceptible to noise. A factor of 3-5 faster is generally sufficient.

We should not get too carried away with making the observer poles fast. One problem that can arise is that of peaking; where a very high observer gain produces a (short) transient where the state estimate is wildly wrong. This can be a problem in practical systems if it causes saturation of an actuator for example. Often the dynamics of real actuators will be sufficiently slow that this is not an issue, but this is something you should check in simulation. In general you might want to avoid relying on actuator (or sensor) dynamics to make your system work. It is undesirable for your system to break if a component is upgraded.

4.3 Observer Design Example

Consider the system described in the regulator lecture. We now wish to design an appropriate state observer and use its state estimate to stabilise the system. Recall that the regulator poles were at $s \in \{-1.33 \pm \mathrm{j}1.49, -13.3\}$. We need to place three observer poles at least three times further into the s-plane than the regulator poles. Let's place the poles just a factor of three further to the left. That is, we will put the observer poles at $s \in \{-4 \pm \mathrm{j}4.5, -39\}$

First let's check the system's observability. `obsv(A,C)` yields the identity matrix (which is full rank). Therefore the system is observable. Let's find $\boldsymbol{L}$ to place the observer poles:

```
L= place(A', C', [-39,-4+4.5j,-4-4.5j])'
```

which yields $\boldsymbol{L} = \begin{bmatrix}45.9\\248\\231\end{bmatrix}$.

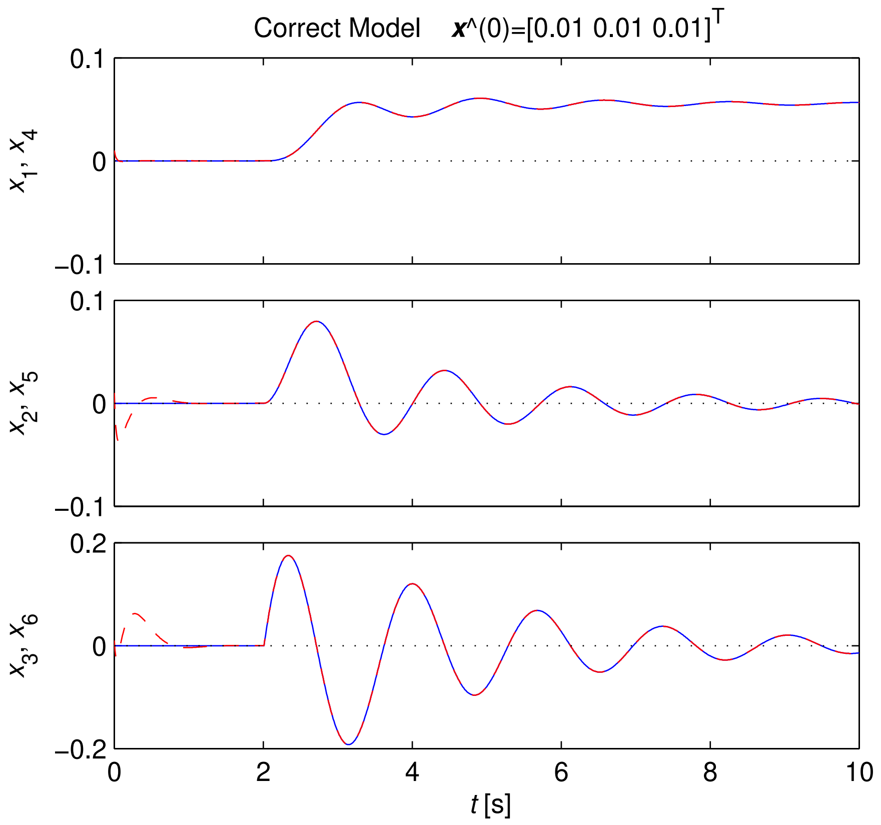


Figure 4.5: Comparison of a plant and an observer than is given an incorrect initial guess at the state. The plant responses are the solid blue curves, while the red dashed responses are those of the estimate. In this case, $\boldsymbol{x} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$, but $\hat{\boldsymbol{x}} = \begin{bmatrix} 0.01 & 0.01 & 0.01 \end{bmatrix}^T$

.

Let's run a simulation of the system open loop and see how the observer performs in predicting the system state. We will give the observer an incorrect initial guess at the state to see the observer transient. The result can be seen in figure 4.5.

4.3.1 Incorrect Model

Consider the case where we have in incorrect model of the system. We will again use the system that we have previously discussed, which had $\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -18 & -15 & -2 \end{bmatrix}$. However, we will use $\widehat{\boldsymbol{A}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -19 & -14 & -2.2 \end{bmatrix}$ for the model inside the observer. This has the effects of shifting the open loop eigenvalues from $\{-1.29, -0.36 \pm \mathrm{j}3.73\}$ to $\{-1.47, -0.37 \pm \mathrm{j}3.58\}$, which makes the exponential mode faster and decreases the frequency of the oscilliatory mode slightly while increasing its damping. In the following figures we will see the step response of an open loop observer (4.6) , a closed loop observer with relatively slow response (4.7) and a closed loop observer with fast observer poles (4.8) . In all cases there is no controller used ($\boldsymbol{K} = 0$), so we are observing the natural open loop behaviour of the plant. On the following graphs, $x_4 = \hat{x}_1$, $x_5 = \hat{x}_2$ and $x_6 = \hat{x}_3$.

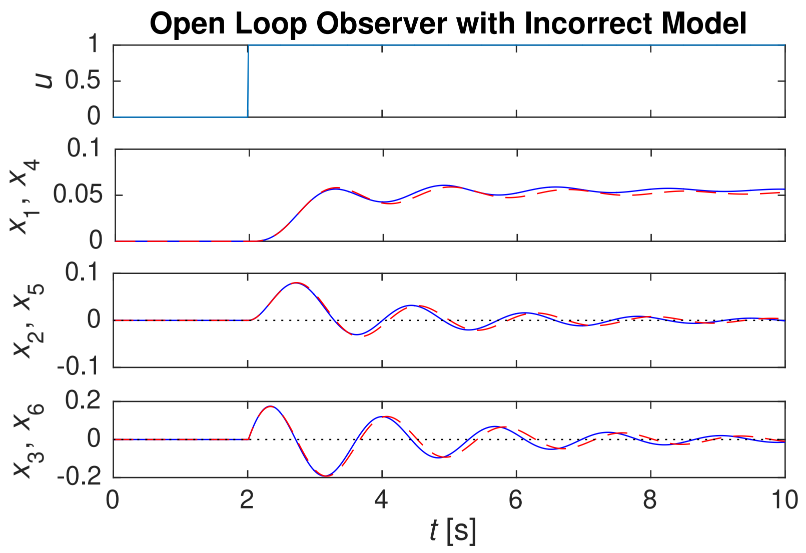


Figure 4.6: Operation of an observer that has an incorrect model of the plant.

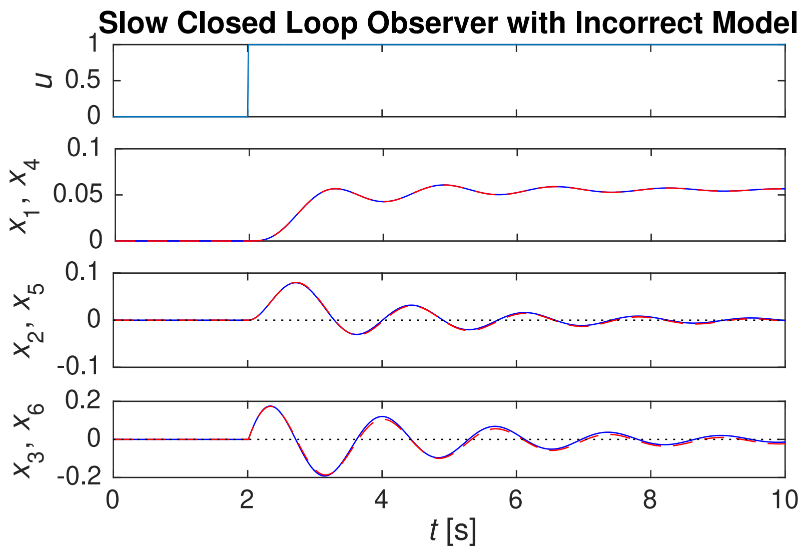


Figure 4.7: Operation of an observer having an incorrect model and slow observer poles.

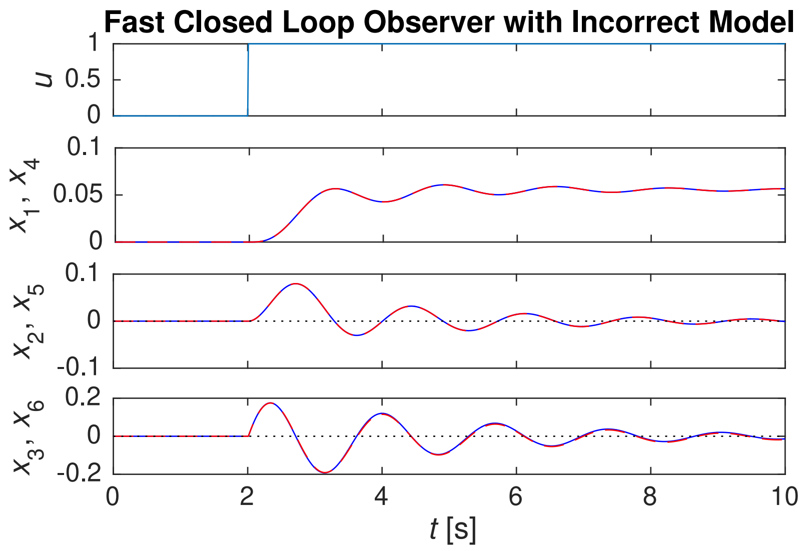


Figure 4.8: Operation of an observer having an incorrect model and slow observer poles.

Disturbance Rejection with an Incorrect Model

We can also examine the response of the three observers when the plant is excited with an impulse, but the observer is not. This is a model of a simple disturbance to the plant that we do not know about, so therefore cannot feed into the observer directly. The respones of the open loop observer, slow closed loop observer and fsat closed loop observer are shown in figures 4.9, 4.10 and 4.11 respectively. We see that the open loop observer does not respond to the disturbance at all. The closed loop observers do respond, with the observer having fast poles acting to reduce the state error more effectively. However, we should not get too excited about making our observer fast, because this also increases suscceptibiliy to high frequency noise in the measurements.

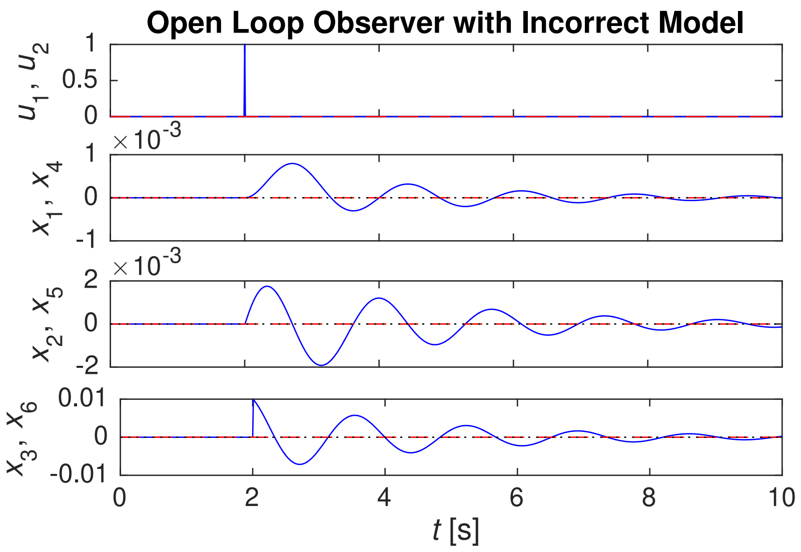


Figure 4.9: Operation of an observer that has an incorrect model of the plant.

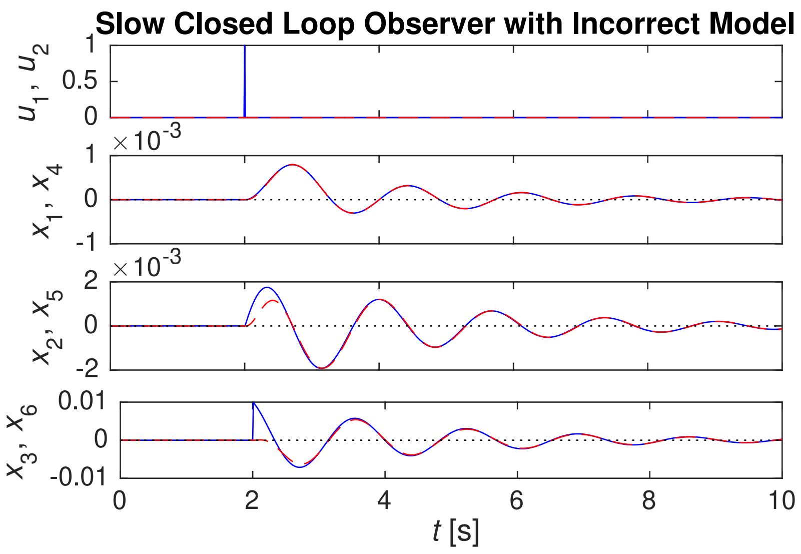


Figure 4.10: Operation of an observer having an incorrect model and slow observer poles.

One problem of making the observer poles too fast is that the effects of measurement noise is more significant. Sensor noise is modelled as a signal $\boldsymbol{v}$, such that $\boldsymbol{y} = \boldsymbol{Cx} + \boldsymbol{v}$. This noise is impressed upon $\tilde{\boldsymbol{y}}$ and then fed back to the estimator state via $\boldsymbol{L}$, which degrades the quality of the estimate. In this

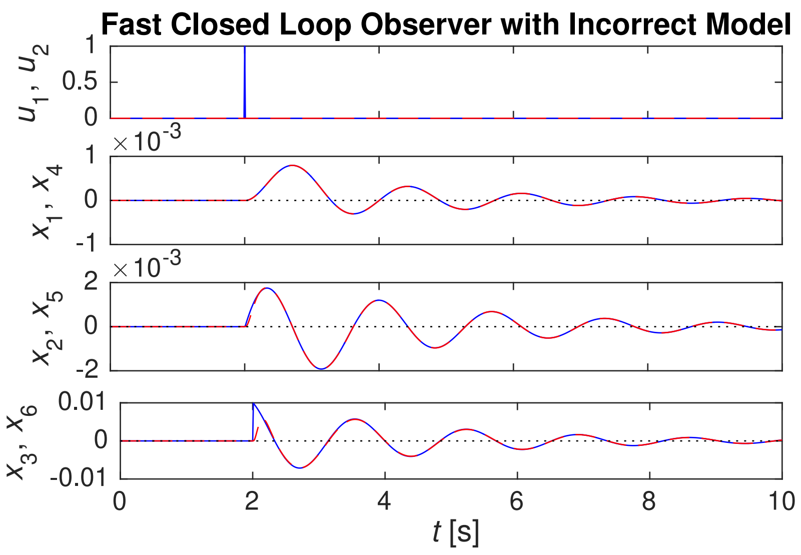


Figure 4.11: Operation of an observer having an incorrect model and slow observer poles.

case we have

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{u} - \boldsymbol{L}\boldsymbol{C}\hat{\boldsymbol{x}} + \boldsymbol{L}\boldsymbol{C}\boldsymbol{x} + \boldsymbol{L}v$$

$$\Longrightarrow \begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\hat{\boldsymbol{x}}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & 0 \\ \boldsymbol{L}\boldsymbol{C} & \boldsymbol{A} - \boldsymbol{L}\boldsymbol{C} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \hat{\boldsymbol{x}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} & 0 \\ \boldsymbol{B} & \boldsymbol{L} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

In the examples to follow we have contaminated the measurement with white noise having variance 100×10^{-6}. That is,

$$y = \boldsymbol{C}\boldsymbol{x} + v, \text{ where } v \sim \mathcal{N}\left(0, 100 \times 10^{-6}\right)$$

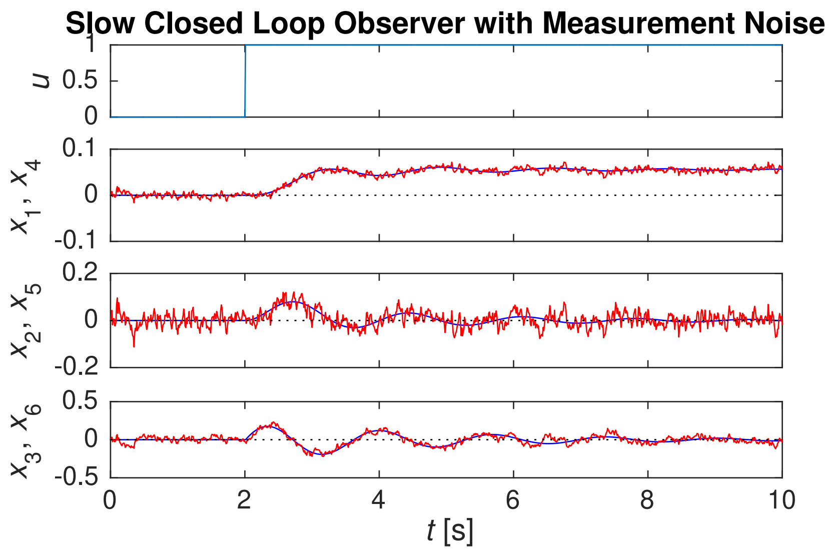


Figure 4.12: Operation of an observer having finite measurement noise and slow observer poles.

4.4 Using observers for regulator problems

We now have an observer that generates an estimate of the state vector $\hat{\boldsymbol{x}}$. We can now modify our compensator design to use the estimate of the state vector rather than the real vector.

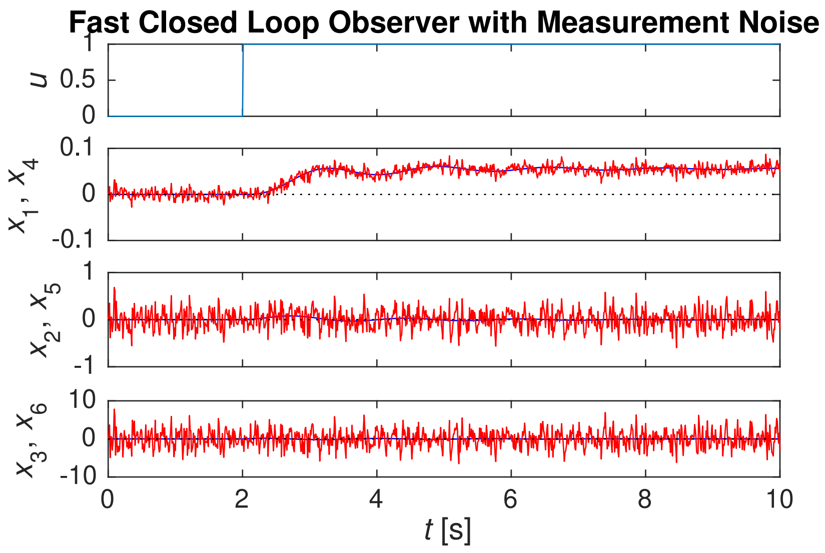


Figure 4.13: Operation of an observer having finite measurement noise and slow observer poles.

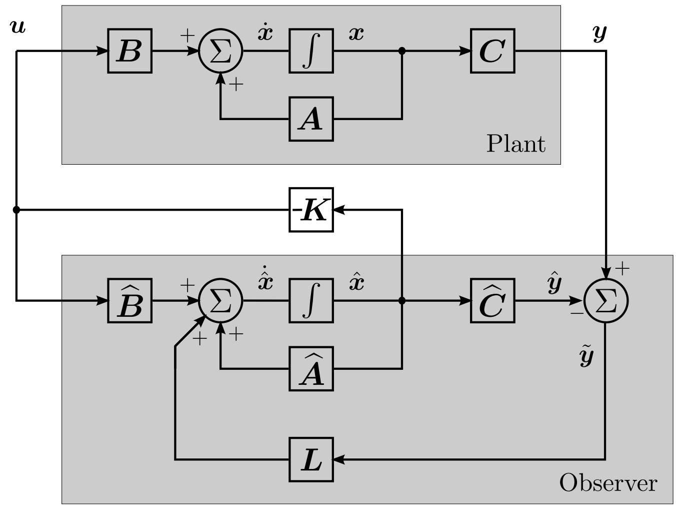


Figure 4.14: A regulator where the feedback is derived from the state estimate produced by a closed loop observer.

Recall that when we developed the compensator, we arrived at a state transition equation that incorporated the effect of the feedback,

$$\dot{\boldsymbol{x}} = (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K})\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{K}\boldsymbol{x}$$

We now modify this to use $\hat{\boldsymbol{x}}$ rather than $\boldsymbol{x}$ for the feedback source.

$$\begin{aligned}\dot{\boldsymbol{x}} &= \boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{K}\hat{\boldsymbol{x}}\\ &= \boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{K}(\boldsymbol{x} - \tilde{\boldsymbol{x}})\\ \dot{\boldsymbol{x}} &= (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K})\boldsymbol{x} + \boldsymbol{B}\boldsymbol{K}\tilde{\boldsymbol{x}}\end{aligned}$$

Thus the dynamics of the system depend on $\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}$ as before, but now there is an extra term arising from the estimate error $\tilde{\boldsymbol{x}}$. However, we know how the error behaves, so we will be able to incorporate this into our model.

We can build a new state space representation of the entire system. It will contain twice as many states as the original system as we now must model the behaviour of both $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$. Recall that $\dot{\tilde{\boldsymbol{x}}} = (\boldsymbol{A} - \boldsymbol{L}\boldsymbol{C})\tilde{\boldsymbol{x}}$. Combining this with the previous equation yields our new state equation:

$$\begin{bmatrix}\dot{\boldsymbol{x}}\\ \dot{\tilde{\boldsymbol{x}}}\end{bmatrix} = \begin{bmatrix}\boldsymbol{A}-\boldsymbol{BK} & \boldsymbol{BK}\\ 0 & \boldsymbol{A}-\boldsymbol{LC}\end{bmatrix}\begin{bmatrix}\boldsymbol{x}\\ \tilde{\boldsymbol{x}}\end{bmatrix}$$

We can now determine the dynamics of the new super-system by solving the characteristic equation:

$$\det\begin{bmatrix}s\boldsymbol{I}-\boldsymbol{A}+\boldsymbol{BK} & -\boldsymbol{BK}\\ 0 & s\boldsymbol{I}-\boldsymbol{A}+\boldsymbol{LC}\end{bmatrix} = 0$$

This matrix is in block triangular form which makes it easy to simplify the determinant calculation.

$$\begin{aligned}\det\begin{bmatrix}s\boldsymbol{I}-\boldsymbol{A}+\boldsymbol{BK} & -\boldsymbol{BK}\\ 0 & s\boldsymbol{I}-\boldsymbol{A}+\boldsymbol{LC}\end{bmatrix} &= 0\\ \implies \det[s\boldsymbol{I}-\boldsymbol{A}+\boldsymbol{BK}].\det[s\boldsymbol{I}-\boldsymbol{A}+\boldsymbol{LC}] &= 0\\ \chi_{\mathrm{c}}(s)\chi_{\mathrm{e}}(s) &= 0\end{aligned}$$

Thus the total system poles are a combination of the poles of the compensated system (χ_{c}) and the observer (χ_{e}). If we place our observer poles sufficiently far to the left (or centre) the compensator poles should be dominant and the switch to using $\hat{\boldsymbol{x}}$ should have a negligible effect on the final result. This is known as the separation principle; we can design our compensator and observer separately and combine them without concern.

4.5 Observers in servo systems

We need to consider how the addition of the reference signal will change the observer. We can follow two approaches here:

1. Design the system so that changes in the reference do not excite the observer error. That is, we drive the observer in the same way as the plant.

2. Allow reference changes to excite the observer in such a way that we can tailor the transient response.

The second approach is very powerful, as it actually allows us to place the zeros of the system in an arbitrary location. This give us almost complete control over the system's transient response. However, we will use the simpler approach as it is usually sufficient.

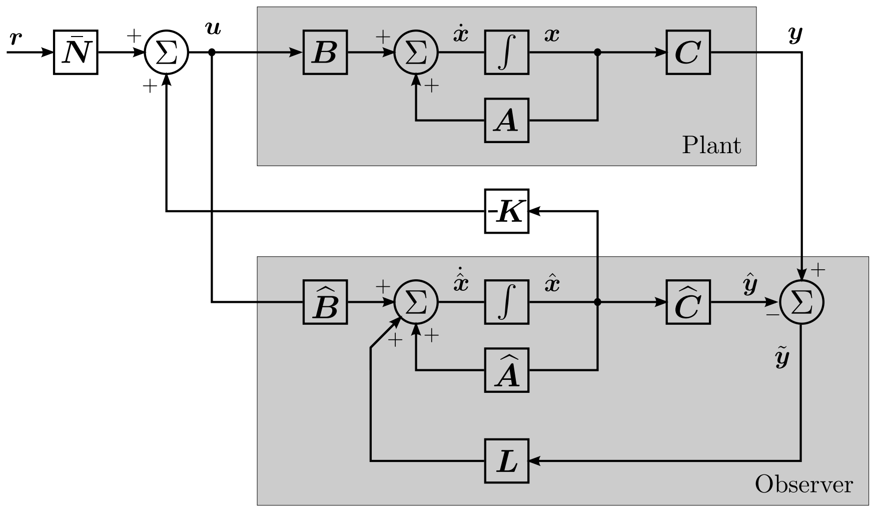


Figure 4.15: Block diagram for a system employing estimated state feedback and using a prefilter $\overline{\boldsymbol{N}}$ to adjust the dc gain for the reference signal $\boldsymbol{r}$.

This is a trivial change to the system diagram. We have just added the prefilter matrix $\overline{\boldsymbol{N}}$ to adjust the gain of the system.

4.6 Implementation of an Observer

The formulation above describes the evolution of $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$, which is useful for control system analysis and design. However, for implementation we need to find the evolution of $\hat{\boldsymbol{x}}$ so that we can use the estimate.

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{Ax} - \boldsymbol{BK}\hat{\boldsymbol{x}} \\ \text{and, } \dot{\hat{\boldsymbol{x}}} &= \boldsymbol{LCx} + (\boldsymbol{A} - \boldsymbol{BK} - \boldsymbol{LC})\hat{\boldsymbol{x}} \end{aligned}$$

We can see that the matrices in the bracketed term refer to the estimates of the system matrices, so if we want to generalise to the case where our model differs from the real system then we have

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{LCx} + (\widehat{\boldsymbol{A}} - \widehat{\boldsymbol{B}}\boldsymbol{K} - \boldsymbol{L}\widehat{\boldsymbol{C}})\hat{\boldsymbol{x}}$$

So, for simulation work we form an augmented state matrix that is governed by

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\hat{\boldsymbol{x}}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & -\boldsymbol{BK} \\ \boldsymbol{LC} & \widehat{\boldsymbol{A}} - \widehat{\boldsymbol{B}}\boldsymbol{K} - \boldsymbol{L}\widehat{\boldsymbol{C}} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \hat{\boldsymbol{x}} \end{bmatrix}$$

Of course, for implementation we do not know $\boldsymbol{x}$, nor do we need to calculate it. We would simply use

$$\begin{aligned} \dot{\hat{\boldsymbol{x}}} &= \boldsymbol{Ly} + (\widehat{\boldsymbol{A}} - \widehat{\boldsymbol{B}}\boldsymbol{K} - \boldsymbol{L}\widehat{\boldsymbol{C}})\hat{\boldsymbol{x}} \\ \boldsymbol{u} &= -\boldsymbol{K}\hat{\boldsymbol{x}} \end{aligned}$$

Figure 4.16 shows a schematic for such a system.

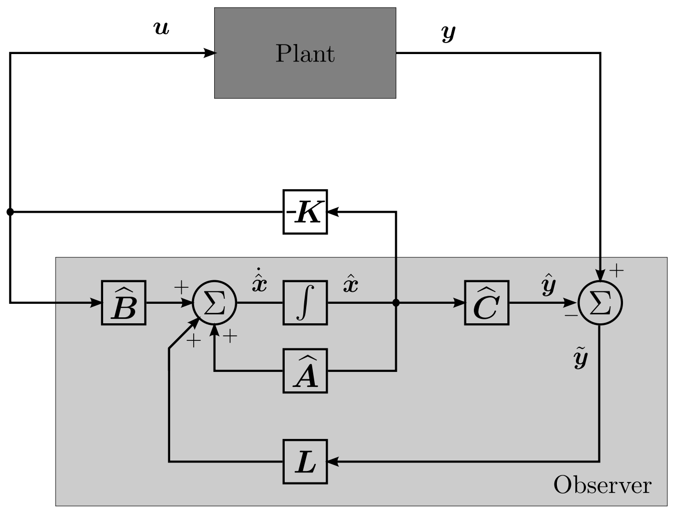


Figure 4.16: Implementation of an observer.

There is some subtlety in writing simulations of observer performance. The examples in these notes all "cheat" because they manipulate the entire state and estimate time histories at once. A simulation that is closer to reality would process only one time step at once. This is not yet a problem, but will become one shortly when we generalise the Luenberger observer. The only problem is keeping information that the universe knows and that the observer knows separate. The input and output signals $\boldsymbol{u}$ and $\boldsymbol{y}$ should be the only things that appear in the two blocks. It also helps keep things clear if you use estimated system matrices ($\widehat{\boldsymbol{A}}$ etc.) in the observer section, even if you know $\widehat{\boldsymbol{A}} = \boldsymbol{A}$ (etc) for your problem. The following code is not complete (or even good), but attempts to show the sort of structure that you should use. It is missing initialisation, and storage of signals for later analysis.

```
for now = 0:max_time
  % The universe
```

```
    x_next = A * x + B * u;
    y = C * x

    % The observer and compensator
    yest = Cest * xest;
    yerr = y - yest;
    xest_next = Aest * xest + Best * u + L * yerr;
    u = r - K * xerr;

    % Updates
    x = x_next;
    xest = xest_next;
  end
```

One of the problems with simulation of observers is that they tend to work too well, particularly if you make them fast. That is, they will do so well in estimating the state that you won't know whether your code is actually doing an estimate or not (it is easy to let information that the observer isn't supposed to know sneak into your observer code).

- Give your observers the wrong initial condition so that you can see the transient.
- Give the plant a kick (but not the observer).
- Give your observer the wrong model to see how it responds. The "best" way of doing this is to perturb the model's eigenvalues a little, and/or change the scaling of the $\boldsymbol{C}$ matrix.
- Try slowing down the observer to make the transients more visible.

If you want to perturb the plant's state (but not its estimate) then we need to introduce some extra inputs. These should be coupled into the true state only. Let's denote these extra inputs $\boldsymbol{d}$ (for disturbance) and introduce them via

$$\begin{cases} \dot{\boldsymbol{x}} &= \boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{K}\hat{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{r} + \boldsymbol{d} \\ \dot{\hat{\boldsymbol{x}}} &= \boldsymbol{L}\boldsymbol{C}\boldsymbol{x} + \left(\widehat{\boldsymbol{A}} - \widehat{\boldsymbol{B}}\boldsymbol{K} - \boldsymbol{L}\widehat{\boldsymbol{C}}\right)\hat{\boldsymbol{x}} + \widehat{\boldsymbol{B}}\boldsymbol{r} \end{cases}$$

$$\rightsquigarrow \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \boldsymbol{x} \\ \hat{\boldsymbol{x}} \end{bmatrix} = \cdots + \begin{bmatrix} \boldsymbol{B} & \boldsymbol{I}_n \\ \widehat{\boldsymbol{B}} & 0 \end{bmatrix}\begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{d} \end{bmatrix}$$

Here we have chosen to inject our disturbance directly into the state. There are some circumstances where we might want to inject our disturbances via some other coupling mechanism, which we can do by replacing $\boldsymbol{I}_n$ with some other suitable coupling matrix.

5 Linear Quadratic Regulators

So far we have designed control systems by choosing desired closed loop poles locations manually. This often works well, but there is a measure of engineering judgement involved. For example, how do we know that we can't do 'better'? Optimal control takes a different approach:

- We define a metric that tells us how well the system is performing.
- We design the controller that is *optimal*, in the sense that it minimises the metric.

An optimal controller is only optimal with respect to a specific metric. In general a different metric will result in a different optimal controller.

There are a wide variety of possible metrics.

- 'I want the output deviations to be as small as possible.'
- 'I want the worst case error to be small.'
- 'I want the control effort to be small.'
- 'I want to minimise the total cost ($) of running the plant.'
- 'I want to minimise $\int |e|\, t \,\mathrm{d}t$'

We will focus on metrics that allow us to make both the state variables $\boldsymbol{x}$ and the control effort $\boldsymbol{u}$ small. We will penalise large $\boldsymbol{x}$ and $\boldsymbol{u}$ quadratically. That is we will have a cost function (metric) that involves *squares* of the state and input variables. This can be regarded as attempting to minimise the 'energy' of the system, but in reality we choose squares because it makes the maths tractable. More complex metrics are possible, but more troublesome.

5.1 The SISO case

Consider a single output, single input system where we want to keep the output close to zero, but we also don't want the input to be too large. We posit a cost functional J for our system which we will try to minimise. The cost has contributions from both y and u.

$$J\big(u(t)\big) := \int_0^\infty y^2 + \rho u^2 \,\mathrm{d}t \quad \rho \in \mathbb{R}$$

If our control system keeps y small then the first term in the integral will be small. However, if we use too large an input signal to achieve this then the second term will be large. The constant ρ allows us to specify how much we care about the relative sizes of y and u. For example, if ρ is large we heavily penalise large control signals. ρ can be thought of as the relative costs of u and y.

5.1.1 Oscillator Control Optimization Example

Consider a lightly damped oscillator which is described by $\boldsymbol{A} = \begin{bmatrix} -0.2 & 10 \\ -10 & -0.2 \end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix} 30 \\ 0 \end{bmatrix}$, $\boldsymbol{C} = \begin{bmatrix} 1 & 0 \end{bmatrix}$, $\boldsymbol{D} = 0$. The system has open loop eigenvalues at $-0.2 \pm \mathrm{j}10$. The open loop response of the oscillator is shown in figure 5.1.

We will use state feedback to examine the family of controllers that place the closed loop poles at $\sigma \pm \mathrm{j}\omega$. We will consider controllers which have their closed-loop pole locations randomly drawn from uniform distributions over the following intervals.

$$\sigma \in [-20, -4] \qquad \text{and} \qquad \omega \in [0, 10]$$

That is, we randomly select σ and ω and find K that places the poles in the selected locations. For the examples to follow we use a set of 10 000 different controllers. Effectively we are testing the system with 10 000 different inputs ($\boldsymbol{u} = -\boldsymbol{K}\boldsymbol{x}$) to see how the output behaves. These inputs are all sensible inputs, but are only a partial representation of all of the signals that could be applied to the system.

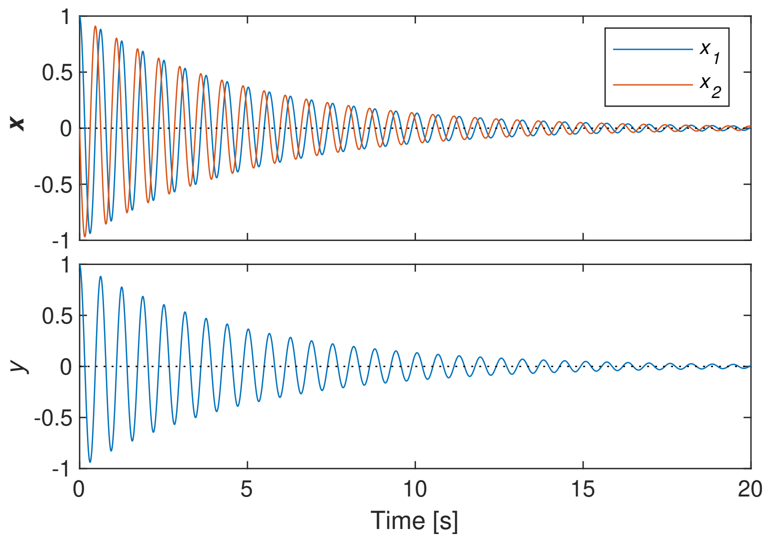


Figure 5.1: Open loop impulse response of the oscillator.

Each controller will have a different performance in terms of overshoot, settling time etc, so each will have different costs for output deviation and control effort.

For each controller we calculate $J_u = \int_0^{20} u(t)^2\,\mathrm{d}t$ and $J_y = \int_0^{20} y(t)^2\,\mathrm{d}t$. In this case the integrals terminate at $t = 20$ because each closed loop system has decayed close to zero by then, so further integration is unnecessary. (The slowest systems have closed loop poles with $\sigma = -4$, so 20 s is five time constants.) A graphical representation of these costs is shown in figure 5.3.

Moving the poles further to the left of the s-plane produces better performance (lower J_y) but requires more effort (higher J_u). The lowest J_u values are produced when the the closed loop poles are near the open loop poles. This is for σ near 0, ω near 10.

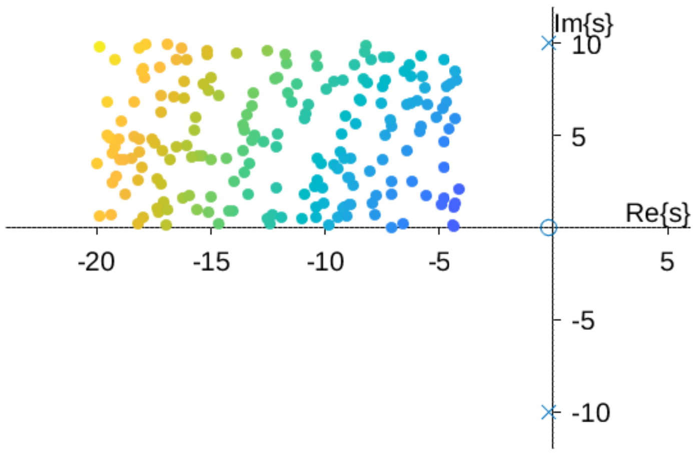


Figure 5.2: Illustration of the closed loop pole locations for 200 of the 10 000 controllers. The open loop poles are also shown in blue.

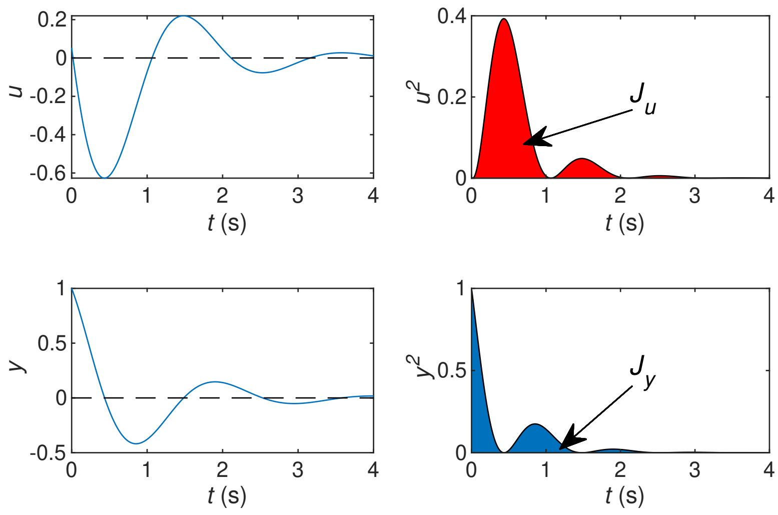


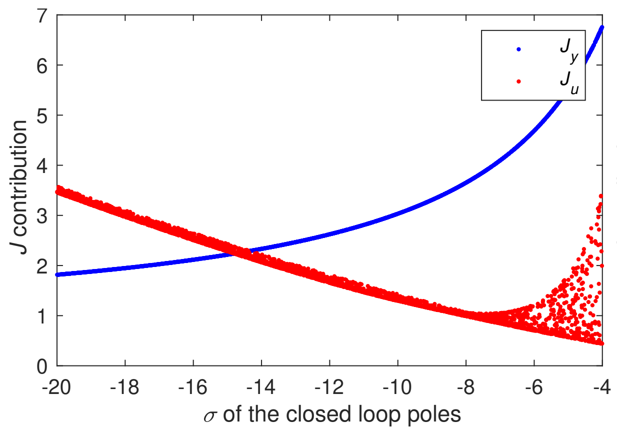
Figure 5.3: Formation of the two elements of the cost metric J. J_u is derived by squaring and then integrating u and similarly J_y is derived by squaring and then integrating y.

We can plot the two costs J_u and J_y as points in a two dimensional space, as shown in 5.5.

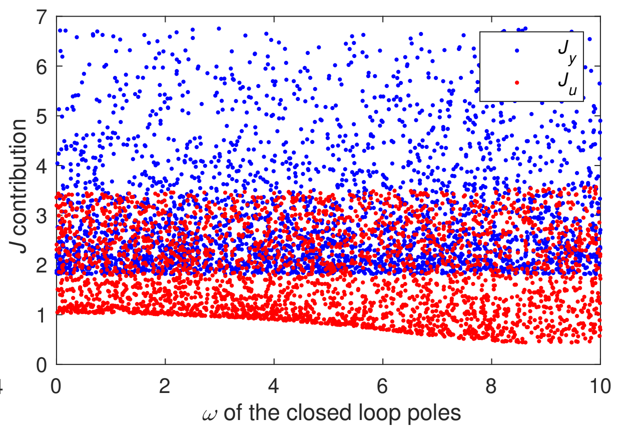
Each point on the graph represents the performance of an individual controller. For comparison, the open loop system has $J_u = 0$, $J_y \approx 125$.

Which controller is best depends on how much we care about J_u and J_y, but we know that the optimal point lies on the Pareto front, as shown in figure 5.6

If a controller produces a response that does not lie on the Pareto front, that means that a different controller can produce the same performance (as measured by output cost) for less effort. Equivalently, a different controller could produce better performance for the same cost. That is, controllers with a response not on the Pareto front are said to be *dominated*, and need not be considered for adoption.



(a) Variation of costs with Re$\{s\}$.



(b) Variation of costs with Im$\{s\}$.

Figure 5.4: Output and control costs for a set of 10,000 controllers with randomly selected closed loop poles.

5.1.2 Choosing ρ

Recall that the overall metric is

$$J := \int_0^{t_f} y^2 + \rho u^2 \, \mathrm{d}t = J_y + \rho J_u$$

By changing ρ we can trade off better performance against control effort. If we make ρ large then we penalise u heavily, so we are asking for a controller that is relatively gentle. If we make ρ small then we are saying that we don't mind a large u, so we would expect to get a much more aggressive controller. We can regard ρ as a design parameter, which allows the designer to choose which aspect of the system performance is more important.

If $\rho = 1$ then $J = J_y + J_u$. This leads to lines of constant J that are straight lines having slope -1. The optimal point is where the line with smallest J is tangent to the Pareto front.

If we change ρ we change the slope of the constant J lines. Consider $J = J_y + 2J_u$, which gives constant J lines with slope -2. The use of a different metric leads to a different optimal point.

The effect of choosing a different ρ is illustrated in figures 5.7 and 5.8. Be careful not to think that the $\rho = 1$ controller is 'better' because it has a lower optimal value for J. It is a different metric, so we can't compare the two examples.

The particular pattern of points in figures 5.5 and 5.6 are a result of the choice of state feedback control for this example. It is possible to have additional controllers that fill the apparent void on the right side of the diagram. For example, one could take one of the controllers on the Pareto front and vary the u faster than the system could respond. This would not change the output performance appreciably, but could generate an arbitrary control cost. One would not do this in practice of course.

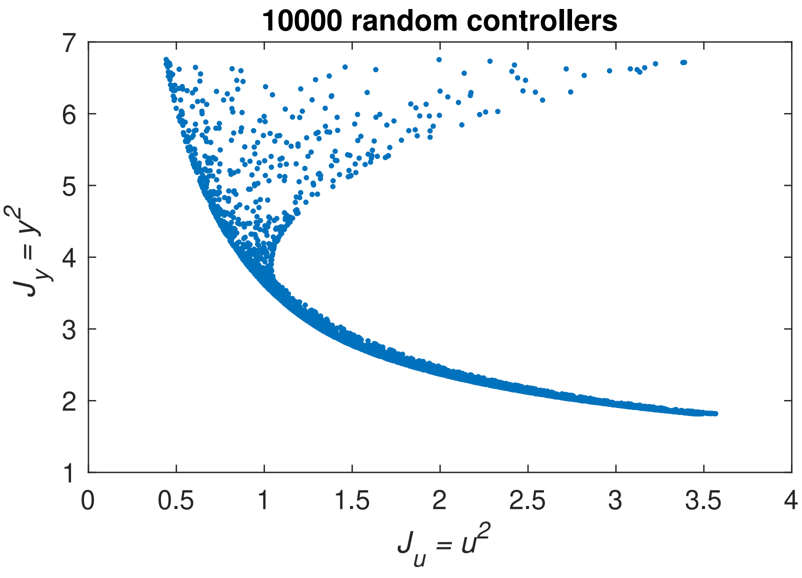


Figure 5.5: Relationship between output and control costs for 10,000 controllers with randomly chosen closed loop pole locations.

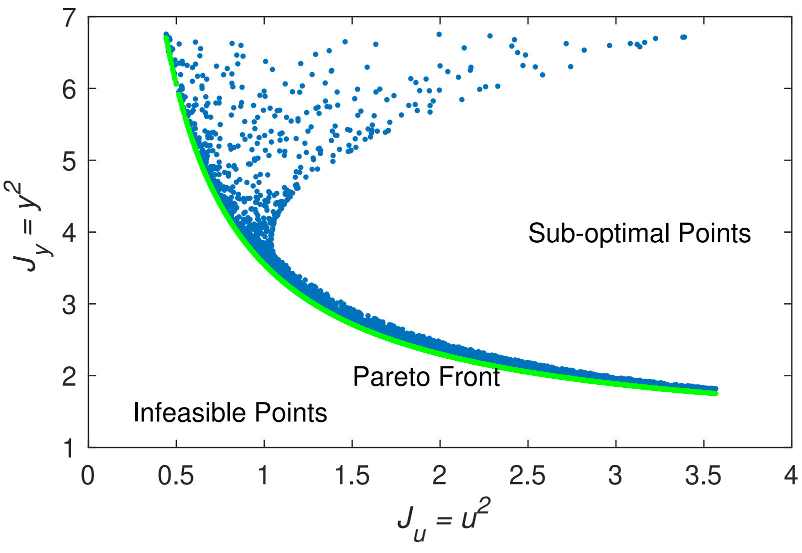


Figure 5.6: Regions of sub-optimal and infeasible controllers, along with an illustration of the Pareto front.

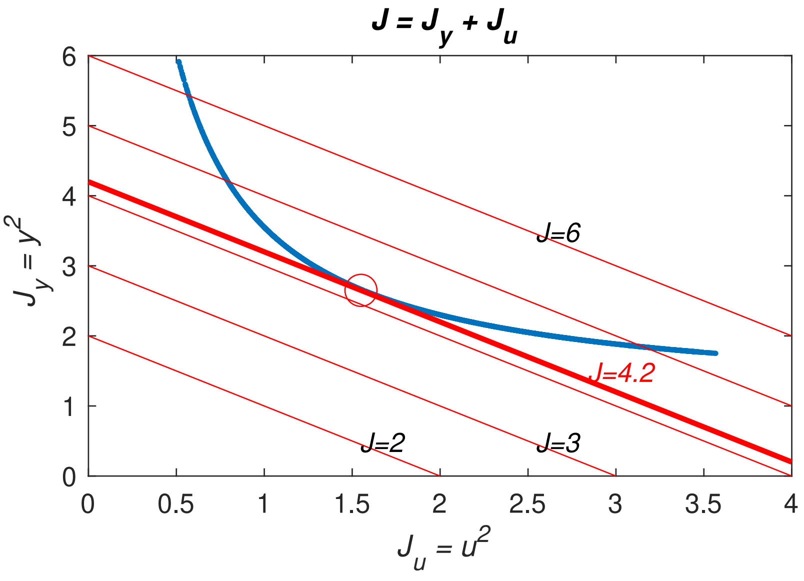


Figure 5.7: Determination of the optimal controller and minimum cost for a metric having $\rho = 1$.

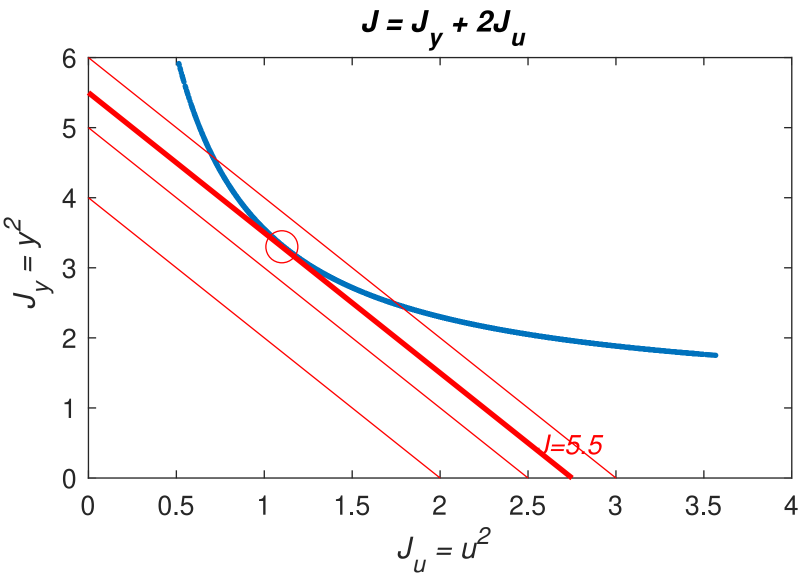


Figure 5.8: Determination of the optimal controller and minimum cost for a metric having $\rho = 2$.

5.2 The MIMO LQR metric

We need to generalise the procedure to higher dimensional systems. Perhaps the most obvious approach would be to penalise vector quantities according to their Euclidean length. For $\boldsymbol{x} \in \mathbb{R}^n$

$$\|\boldsymbol{x}\|^2 = \boldsymbol{x}^\mathsf{T}\boldsymbol{x} = \sum_{i=1..n} x_i^2$$

However, we might want to penalise certain elements of a vector more heavily than others, so we introduce a scaling by a symmetric matrix $\boldsymbol{Q}$.

$$\boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} = \sum_{i,j=1..n} q_{ij}x_i x_j$$

Similarly we use a cost $\boldsymbol{u}^\mathsf{T}\boldsymbol{R}\boldsymbol{u}$ to penalise excursions in the control effort $\boldsymbol{u}$.

$$\boldsymbol{u}^\mathsf{T}\boldsymbol{R}\boldsymbol{u} = \sum_{i,j=1..m} r_{ij}u_i u_j$$

In many cases it suffices to make $\boldsymbol{Q}$ and $\boldsymbol{R}$ diagonal. We then only need to consider q_{ii} which tells us how much we care about x_i in our metric and r_{ii} which tells us how much we care about u_i. Thus, we see that

$$\boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} = \sum_{i=1..n} q_{ii}x_i^2$$
$$\boldsymbol{u}^\mathsf{T}\boldsymbol{R}\boldsymbol{u} = \sum_{i=1..m} r_{ii}u_i^2$$

For the special case of $\boldsymbol{Q}, \boldsymbol{R}$ both diagonal.

If we aim to drive the system to state $\boldsymbol{x} = 0$ in time t_f, we can generalise our previous expression for J to obtain

$$J\big(t_\mathrm{f}, \boldsymbol{u}(t)\big) := \boldsymbol{x}^\mathsf{T}(t_\mathrm{f})\boldsymbol{Q}_\mathrm{f}\boldsymbol{x}(t_\mathrm{f}) + \int_0^{t_\mathrm{f}} \boldsymbol{x}(t)^\mathsf{T}\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}(t)^\mathsf{T}\boldsymbol{R}\boldsymbol{u}(t)\,\mathrm{d}t$$

where $\boldsymbol{Q}$ is a symmetric positive definite matrix and $\boldsymbol{R}$ and $\boldsymbol{Q}_\mathrm{f}$ are positive semidefinite matrices.

- Positive definite $\boldsymbol{Q}$ means that $\boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} > 0 \quad \forall \boldsymbol{x} \neq 0$.
- Positive semidefinite $\boldsymbol{R}$ means that $\boldsymbol{u}^\mathsf{T}\boldsymbol{R}\boldsymbol{u} \geq 0 \quad \forall \boldsymbol{u} \neq 0$.

$\boldsymbol{Q}$ penalises deviations in the state variables, $\boldsymbol{R}$ penalises control effort, and $\boldsymbol{Q}_\mathrm{f}$ penalises non-zero state variables at some final time t_f.

In many cases we do not have a final state $\boldsymbol{x}(t_\mathrm{f})$ to which we are attempting to drive the system. Rather we wish to regulate the system's performance for all time. Such problems are known as infinite-horizon problems and in such cases we can simplify the cost functional by removing the term penalizing $\boldsymbol{x}(t_\mathrm{f})$.

$$J := \int_0^\infty \boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^\mathsf{T}\boldsymbol{R}\boldsymbol{u}\,\mathrm{d}t$$

Notice that we have moved the upper limit of integration to ∞, so we are now attempting to find a controller that minimises the integral over all time. We have also dropped the time arguments on $\boldsymbol{x}$ and $\boldsymbol{u}$ for brevity.

5.3 Choosing Q and R

We modify $\boldsymbol{Q}$ and $\boldsymbol{R}$ to tune the performance of a control system.

- Multiplying one of the two by a scalar changes the *relative* weighting of the state variables and the inputs. Multiplying $\boldsymbol{Q}$ by a positive scalar places more emphasis on keeping $\boldsymbol{x}$ small. Scaling both $\boldsymbol{Q}$ and $\boldsymbol{R}$ by the same factor has no effect.

- The absolute numbers in $\boldsymbol{Q}$ and $\boldsymbol{R}$ rarely matter, as it is their relative sizes that determine the LQR compensators action. If you care a lot about keeping x_i low then q_{ii} should be high relative to other elements of $\boldsymbol{Q}$ and $\boldsymbol{R}$. Similarly make r_{ii} high to keep tight control of u_i.
- Reducing deviation in one variable by increasing the corresponding q_{ii} or r_{ii} typically increases the deviation of other variables.

One possible initial guess for $\boldsymbol{Q}$ and $\boldsymbol{R}$ is to set them to the identity matrix to equally penalise state and input deviations. Or you could set $\boldsymbol{Q} = \boldsymbol{C}^{\mathsf{T}}\boldsymbol{C}$ to equally penalise output and inputs instead. Simulation will then show which variables are larger than you would like. You can then modify the cost matrices to achieve more desirable performance. You would usually do this in simulation because large u_i or x_i might cause damage, or move a system away from the applicable range of a linearised model. However, for sufficiently benign systems you could do this experimentally. Then adjust to $\boldsymbol{Q} = \operatorname{diag}\left\{\dfrac{\max(x_i^2)}{x_{i,\mathrm{b}}^2}\right\}$, $\boldsymbol{R} = \operatorname{diag}\left\{\dfrac{\max(u_i^2)}{u_{i,\mathrm{b}}^2}\right\}$ where $x_{i,\mathrm{b}}$ is the desired bounding value for x_i. While this is likely to get you close to the desired performance (if it is possible) you may need to fine tune $\boldsymbol{Q}$ and $\boldsymbol{R}$ manually.

An even simpler approach is to set first guesses at $\boldsymbol{Q}$ and $\boldsymbol{R}$ without measuring or simulating the system performance. We simply assume that all state variables and input signals have similar magnitudes under typical operating conditions. Sometimes this is not a bad assumption; in complex problems we sometimes deliberately scale states to be similar in magnitude for numerical reasons. In this case we use *Bryson's* rule:

$$\boldsymbol{Q} = \operatorname{diag}\left\{\frac{1}{x_{i,\mathrm{b}}^2}\right\}$$
$$\boldsymbol{R} = \operatorname{diag}\left\{\frac{1}{u_{i,\mathrm{b}}^2}\right\}$$

where $x_{i,\mathrm{b}}$ is the desired bounding value for x_i. Manual tuning of $\boldsymbol{Q}$ and $\boldsymbol{R}$ will most likely then be required.

The final state deviation cost matrix $\boldsymbol{Q}_\mathrm{f}$ only contributes to J at one time instant. By comparison $\boldsymbol{Q}$ and $\boldsymbol{R}$ act to run up cost throughout the running time of the problem. Consequently $\boldsymbol{Q}_\mathrm{f}$ will need to be bigger than $\boldsymbol{Q}$ by a factor of t_f to make it as important in the optimization. This approach is particularly useful in problems where you are changing t_f, as it ensures that the relative importance of the final state is preserved through the tuning. Similarly, if we want to make deviations in x_i and u_j about the same then we need to compensate for any imbalance in the number of states and inputs. ie. we may sum over more states in penalising $\boldsymbol{x}^{\mathsf{T}}\boldsymbol{Q}\boldsymbol{x}$ than over inputs in penalising $\boldsymbol{u}^{\mathsf{T}}\boldsymbol{R}\boldsymbol{u}$. For example we could modify Bryson's rule to

$$\boldsymbol{Q} = \frac{1}{n}\operatorname{diag}\left\{\frac{1}{x_{i,\mathrm{b}}^2}\right\}$$
$$\boldsymbol{R} = \frac{1}{m}\operatorname{diag}\left\{\frac{1}{u_{i,\mathrm{b}}^2}\right\}$$

When $\boldsymbol{Q}$ or $\boldsymbol{R}$ are not diagonal we penalise having two state be variables high (or low) simultaneously. For example a nonzero q_{34} is a cost applied to x_3x_4. There are times when we would like to penalise various *combinations* of state variables, or *combinations* of inputs. For example,

- If x_1 is a voltage and x_2 is an associated current, then keeping x_1x_2 low (by making q_{12} large) minimises power dissipation.
- If u_1, u_2 and u_3 are dc currents drawn from the same source, then we could make r_{12}, r_{13} and r_{23} large to keep the source current low.

Typically the off diagonal elements of $\boldsymbol{R}$ and $\boldsymbol{Q}$ are adjusted only once a first pass design with the diagonal elements has been completed.

5.3.1 Intuition about $\boldsymbol{Q}$ and $\boldsymbol{R}$

$\boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} = 1$ describes an ellipsoid that has its axis aligned with the eigenvectors of $\boldsymbol{Q}^{-1}$ and equatorial radii equal to the square root of the corresponding eigenvalues. For example consider $\boldsymbol{Q} = \begin{bmatrix} q_1 & 0 \\ 0 & q_2 \end{bmatrix} \implies \boldsymbol{Q}^{-1} = \begin{bmatrix} 1/q_1 & 0 \\ 0 & 1/q_2 \end{bmatrix}$. We can find the cost arising from use of this matrix;

$$\begin{aligned} \boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} = 1 &\implies q_1 x_1^2 + q_2 x_2^2 = 1 \\ &\implies \frac{x_1^2}{(1/\sqrt{q_1})^2} + \frac{x_2^2}{(1/\sqrt{q_2})^2} = 1 \end{aligned}$$

This is the equation for an ellipse (in $\mathbb{R}^2$) with its axes along the directions x_1 and x_2 and having equatorial radii $1/\sqrt{q_1}$ and $1/\sqrt{q_2}$.

An example of the cost landscape (ie, a plot of J) along with ellipses of constant cost can be seen in figure 5.9. In this case the ellipsoids and the cost surface can be simultaneously plotted as they are only two dimensional. In higher dimensions it is (arguably) easier to just think of the constant cost ellipsoids, as illustrated in figure 5.10 for three dimensional systems.

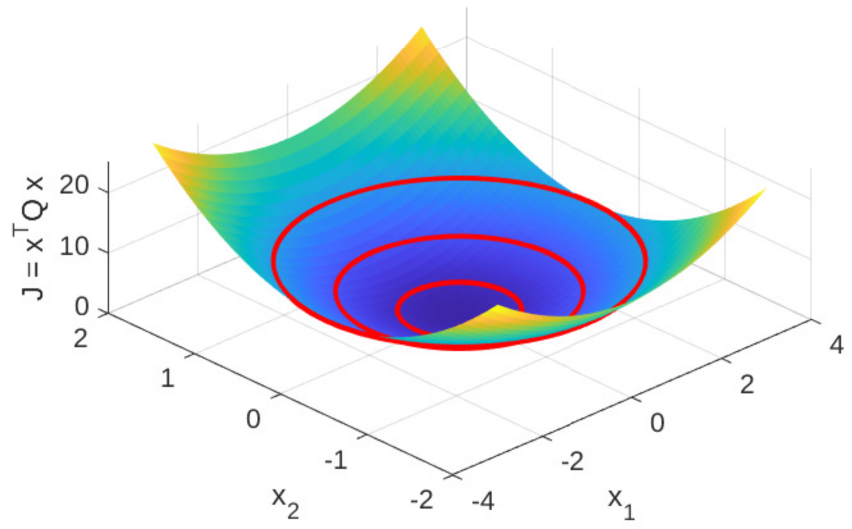


Figure 5.9: Cost landscape $J = \boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x}$ for the case $\boldsymbol{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}$. Ellipsoids of constant cost $J = 1, 2, 3$ are also shown. In this case they are two dimensional ellipsoids, which is to say that they are simple ellipses.

The ellipse $\boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} = 1$ describes the 'shape' of the deviations in $\boldsymbol{x}$ that cost one unit in J. If the ellipse is large in some direction then the state variables are allowed to vary a lot in that direction without costing too much. Directions for which the ellipse is small are the directions in which deviation is expensive.

5.3.2 Constraints and $\boldsymbol{Q}$

We can imagine that a control system has certain constraints on the allowable deviations in the state variables or control signals. The constraints here could be of two types:

1. Hard constraints, where there is a physical limit on a variable, due to issues such as physical properties of the plant, actuator size or safety.

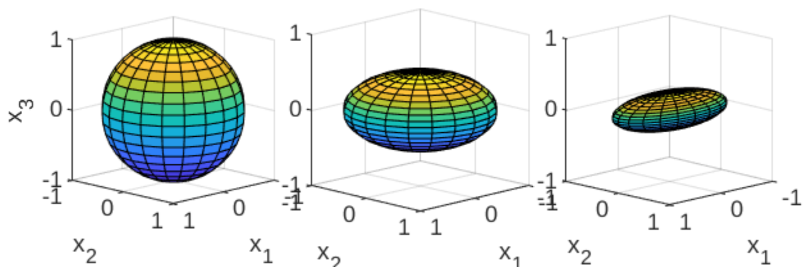


Figure 5.10: Ellipsoids of constant cost for three cost matrices in $\mathbb{R}^3$. The three ellipsoids correspond to three $\boldsymbol{Q}$ matrices, $\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 4 \end{bmatrix}$ and $\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 16 \end{bmatrix}$ respectively.

2. Soft constraint, where we don't *want* a variable to be large, perhaps due to cost, or a desire to ensure performance.

These can be considered as collectively defining a region within which we would like the state to be confined. By choosing the entries of $\boldsymbol{Q}$ and $\boldsymbol{R}$ we can alter the shape of the associated ellipsoid so that it fits inside the system constraints.

Consider $\boldsymbol{x} \in \mathbb{R}^2$, with a constraint that $\|x_1\| < x_{1,\mathrm{b}}$ and $\|x_2\| < x_{2,\mathrm{b}}$. As an example we will consider $x_{1,\mathrm{b}} = 2$, $x_{2,\mathrm{b}} = 1$. We choose $\boldsymbol{Q}$ so that the ellipsoid is the largest that can fit within the constraint

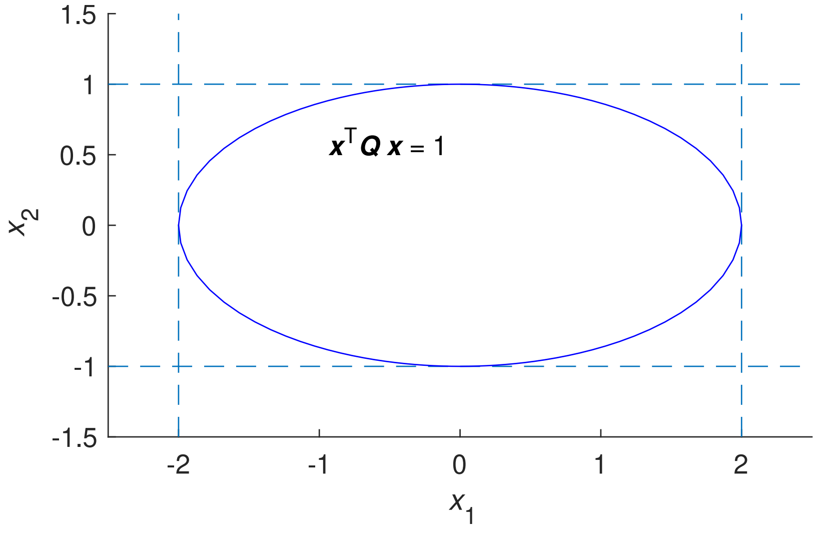


Figure 5.11: The maximum inscribed ellipse given the constraints $\|x_1\| < x_{1,\mathrm{b}}$ and $\|x_2\| < x_{2,\mathrm{b}}$

region. In this case $q_{11} = \sqrt{\frac{1}{2}}$, $q_{22} = 1 \implies \boldsymbol{Q} = \begin{bmatrix} \frac{1}{4} & 0 \\ 0 & 1 \end{bmatrix}$. The arrangement can be seen in figure 5.11

If we have $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{u} \in \mathbb{R}^m$, choosing $\boldsymbol{Q}$ and $\boldsymbol{R}$ can be seen as finding the largest $n+m$-dimensional ellipsoid that can fit inside the constraint area defined by inequalities on elements of $\boldsymbol{x}$ and $\boldsymbol{u}$. The procedure gets slightly more complicated if we have constraints on $x_i x_j$, $i \neq j$ or $u_i x_j$, but the general picture stays the same. We will not use this procedure in calculations, but it provides some intuition into what is happening. However, this method *could* be used in practice (even for very large problems) as there are optimization tools available to find maximum inscribed ellipses given a set of constraints.

5.4 Determining the LQR Feedback Gain

5.4.1 Discrete Time LQR

Let us consider the discrete time version of the LQR problem. This requires conversion of the integral in the expression for J to a summation.

$$J = \boldsymbol{x}(t_\text{f})^\mathsf{T}\boldsymbol{Q}_\text{f}\boldsymbol{x}(t_\text{f}) + \sum_{t=0}^{t_\text{f}-1}\left(\boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^\mathsf{T}\boldsymbol{R}\boldsymbol{u}\right)$$

As in the continuous case we often don't have a specific end state, so we can simplify our cost functional.

$$J = \sum_{t=0}^{\infty}\left(\boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^\mathsf{T}\boldsymbol{R}\boldsymbol{u}\right)$$

We wish to find a feedback controller that has a $\boldsymbol{K}$ matrix that is determined by $\boldsymbol{Q}$ and $\boldsymbol{R}$. We will not examine the derivation of the optimal gain as gaining insight would require more background work than we have time for. In the general case where we have a final target state $\boldsymbol{x}(t_\text{f})$ the optimal gain for the LQR problem is given by

$$\begin{gathered}\boldsymbol{u}(t) = -\boldsymbol{K}(t)\boldsymbol{x}(t)\\ \text{with } \boldsymbol{K}(t) = (\boldsymbol{R} + \boldsymbol{B}^\mathsf{T}\boldsymbol{P}\boldsymbol{B})^{-1}\boldsymbol{B}^\mathsf{T}\boldsymbol{P}\boldsymbol{A}\end{gathered}$$

with $\boldsymbol{P}$ being the positive definite solution to the Riccati equation

$$\boldsymbol{P}(t-1) = \boldsymbol{A}^\mathsf{T}\boldsymbol{P}(t)\boldsymbol{A} - \boldsymbol{A}^\mathsf{T}\boldsymbol{P}(t)\boldsymbol{B}\left(\boldsymbol{B}^\mathsf{T}\boldsymbol{P}(t)\boldsymbol{B} + \boldsymbol{R}\right)^{-1}\boldsymbol{B}^\mathsf{T}\boldsymbol{P}(t)\boldsymbol{A} + \boldsymbol{Q}$$

Notice that the equation is solved *backwards in time* and that it has no dependence on any data. At $t = t_\text{f}$ there is no remaining control cost, so we know that $\boldsymbol{P}(t_\text{f}) = \boldsymbol{Q}$ (or $\boldsymbol{Q}_\text{f}$ if it has been specified separately). Knowing this final value then allows us to find $\boldsymbol{P}$ and hence $\boldsymbol{K}$ for all t. This means that the optimal gain trajectory must be precalculated. If we know how that the system will change at some time then we can calculate a gain profile incorporating these change. However, if the system changes unexpectedly then our optimal $\boldsymbol{K}(t)$ will become invalid and need to be recomputed. It can be shown that $\boldsymbol{x}^\mathsf{T}\boldsymbol{P}(t)\boldsymbol{x}$ is the expected *cost-to-go* for state deviations at time t. It is simply how much cost (in terms of J) must be spent to reach the end of the problem, whether that be at t_f, or the point where $\boldsymbol{x}$ settles to 0.

Our method has given us an optimal input signal $\boldsymbol{u}^\star(t)$ that is calculated at each time step using a feedback law of the form that is by now familiar, although with a time varying $\boldsymbol{K}$.

$$\begin{aligned}\boldsymbol{u}^\star(t) &= -\boldsymbol{K}(t)\boldsymbol{x}(t)\\ &= -\left(\boldsymbol{R} + \boldsymbol{B}^\mathsf{T}\boldsymbol{P}(t)\boldsymbol{B}\right)^{-1}\boldsymbol{B}^\mathsf{T}\boldsymbol{P}(t)\boldsymbol{A}\boldsymbol{x}(t)\end{aligned}$$

In principle we know $\boldsymbol{x}(t)$, so we could precalculate all of $\boldsymbol{u}^\star(t)$. We would then simply apply $\boldsymbol{u}^\star(t)$ in an open loop control scheme to bring the system to a desired state ($\boldsymbol{x} = 0$ for the regulator problem). However, this is rarely done. In practice it is better to use a closed loop system where we allow our optimal $\boldsymbol{K}$ to generate the appropriate $\boldsymbol{u}^\star$ at each time step. This allows the system to respond to unexpected disturbances as they occur.

Let us first examine the behaviour of LQR derived $\boldsymbol{K}$ for a time invariant system. The response is shown in figure 5.12. We will look at this example in more detail shortly.

5.4.2 Steady State LQR

Notice from the previous example that the gain determined by the LQR equation is constant until time approaches the end point. This turns out to be a general property for time invariant problems. For times well away from t_f we can therefore approximate the gain as being constant. In the case of an

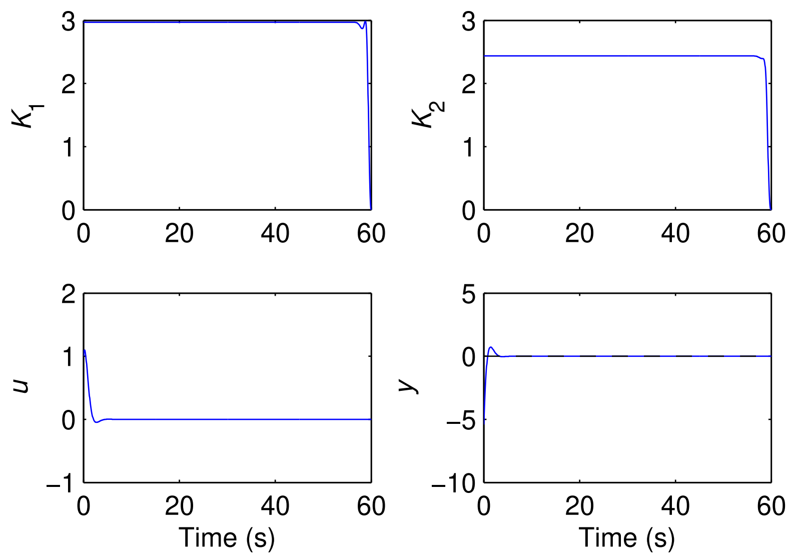


Figure 5.12: Performance of the system described in Burl 'Linear Optimal Control' p196.

infinite-horizon problem (no end point) we *never* approach $t_{\mathrm{f}} = \infty$, so we do not need to incur the overhead of calculating $\boldsymbol{P}(t)$. We therefore seek a steady state approximation to the optimal LQR gain.

In the steady state $\boldsymbol{P}$ does not change, so we can set $\boldsymbol{P}(t-1) = \boldsymbol{P}(t)$ in the Riccati equation. This leads to the the discrete time algebraic Riccati equation. Again, this is called an *algebraic* Riccati equation because it has no time dependence so we can solve for $\boldsymbol{P}$ using algebraic techniques.

$$\boldsymbol{P} = \boldsymbol{A}^{\mathsf{T}}\boldsymbol{P}\boldsymbol{A} - \boldsymbol{A}^{\mathsf{T}}\boldsymbol{P}\boldsymbol{B}(\boldsymbol{B}^{\mathsf{T}}\boldsymbol{P}\boldsymbol{B} + \boldsymbol{R})^{-1}\boldsymbol{B}^{\mathsf{T}}\boldsymbol{P}\boldsymbol{A} + \boldsymbol{Q}$$

As for the finite horizon case, once we know $\boldsymbol{P}$ we calculate the feedback gain using

$$\boldsymbol{K} = (\boldsymbol{R} + \boldsymbol{B}^{\mathsf{T}}\boldsymbol{P}\boldsymbol{B})^{-1}\boldsymbol{B}^{\mathsf{T}}\boldsymbol{P}\boldsymbol{A} \qquad \text{for all } t.$$

5.5 Continuous Time LQR

We can find analogous equations for continuous time systems. If we have a final target we find

$$\boldsymbol{K}(t) = \boldsymbol{R}^{-1}\boldsymbol{B}^{\mathsf{T}}\boldsymbol{P}(t)$$

where $\boldsymbol{P}$ is the solution to the Riccati equation

$$-\dot{\boldsymbol{P}}(t) = \boldsymbol{A}^{\mathsf{T}}\boldsymbol{P}(t) + \boldsymbol{P}(t)\boldsymbol{A} - \boldsymbol{P}(t)\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^{\mathsf{T}}\boldsymbol{P}(t) + \boldsymbol{Q}$$

If we have no final time then the equation for the gain remains unchanged, but $\boldsymbol{P}$ is now found from the continuous time algebraic Riccati equation (CARE);

$$0 = \boldsymbol{A}^{\mathsf{T}}\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^{\mathsf{T}}\boldsymbol{P} + \boldsymbol{Q}$$

5.6 Matlab for steady state LQR design

Matlab is capable of solving the CARE and DARE, so we can use it to find LQR controllers. Matlab uses a slightly more general model than that we have seen, by allowing penalisation of cross terms between state and inputs. This can be formulated as below for discrete and continuous time systems respectively.

$$\text{DT:} \quad J\big(u(t)\big) = \sum_0^{\infty} \left(\boldsymbol{x}^{\mathsf{T}}\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^{\mathsf{T}}\boldsymbol{R}\boldsymbol{u} + 2\boldsymbol{x}^{\mathsf{T}}\boldsymbol{N}\boldsymbol{u}\right)$$

$$\text{CT:} \quad J\big(u(t)\big) = \int_0^{\infty} \left(\boldsymbol{x}^{\mathsf{T}}\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^{\mathsf{T}}\boldsymbol{R}\boldsymbol{u} + 2\boldsymbol{x}^{\mathsf{T}}\boldsymbol{N}\boldsymbol{u}\right)\,\mathrm{d}t$$

- `[K,P,e]=lqr(A,B,Q,R,N)` for continuous time LQR.
- `[K,P,e]=dlqr(A,B,Q,R,N)` for discrete time LQR.
- `[K,P,e]=lqr(S,Q,R,N)` for either if S is an appropriate model.

The functions return the optimal steady state LQR gain $\boldsymbol{K}$, the solution to the Riccati equation $\boldsymbol{P}$ and the closed loop eigenvalues for the system e. In either case the system must be stabilisable for this procedure to work.

5.7 LQR Example

Inspired by Burl 'Linear Optimal Control' p196. Consider the dynamics of a three axis stabilised satellite. If we are near the operating point then the pitch, roll and yaw dynamics can be separated. The dynamics for each axis are assumed not to interact and are given by

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}\boldsymbol{u}$$

with $\boldsymbol{x} = \begin{bmatrix} \theta & \dot{\theta} \end{bmatrix}^{\mathsf{T}}$. We have normalized everything so that $\boldsymbol{u}$ is the angular acceleration produced by the thrusters. We wish to control the (angular) position of the satellite but we don't want to use too much fuel.

Let's use a $\boldsymbol{Q}$ that penalises only the first state variable. We also give some weight to the control effort to prevent excessive fuel use.

$$\begin{aligned} J &= \int_0^{\infty} x^{\mathsf{T}}\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^{\mathsf{T}}\boldsymbol{R}\boldsymbol{u}\,\mathrm{d}t \\ &= \int_0^{\infty} \boldsymbol{x}^{\mathsf{T}}\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\boldsymbol{x} + \boldsymbol{u}^{\mathsf{T}}\frac{1}{10}\boldsymbol{u}\,\mathrm{d}t \end{aligned}$$

`[K,P,e] = lqr(S,Q,R);` results in $\boldsymbol{K} = \begin{bmatrix} 3.16 & 2.51 \end{bmatrix}$. The resulting closed loop eigenvalues are at $-0.75 \pm \mathrm{j}0.75$. In the following simulations we examine the satellite's response to an initial perturbation $\boldsymbol{x}(0) = \begin{bmatrix} 1 & 1 \end{bmatrix}^{\mathsf{T}}$ using the steady state LQR gain. The resulting response can be seen in figure 5.13.

We can vary $\boldsymbol{R}$ to see what effect changing the balance of $\boldsymbol{R}$ and $\boldsymbol{Q}$ has on the response of the system. The resulting set of responses can be seen in figure 5.14. Increasing $\boldsymbol{R}$ increases the cost of $\boldsymbol{u}$, so as expected larger values of $\boldsymbol{R}$ result in smaller deviations in u. The downside of this is that it leads to a more sluggish response, as can be seen in the increasing settling times for y.

We now impose the additional requirement that both the angle and the angular velocity be driven to zero within a specified time. That is, we require $\theta(t_{\mathrm{f}}) = 0$ and $\dot{\theta}(t_{\mathrm{f}}) = 0$. To do this we set $\boldsymbol{P}(t_{\mathrm{f}}) = \boldsymbol{Q}_{\mathrm{f}} = \begin{bmatrix} 10 & 0 \\ 0 & 1000 \end{bmatrix}$. This choice places a relatively large cost on the position of the satellite after the maneuver and an even larger cost on its final velocity. We then use the $\boldsymbol{Q}$ discussed above in determining the remainder of the trajectory. In the following graphs (figures 5.15-5.18) you should notice that the feedback gains remain close to the steady state values until near the end of the responses. As t_{f} decreases we see that the time varying controllers need to work harder (use larger $\|\boldsymbol{u}\|$) to reach the goal. The simulations use $\boldsymbol{R} = 0.1$. 'T.V.' is used to denote the time varying compensators and 'ss' the steady state version.

Notice that the version of the controller with $t_{\mathrm{f}} = 1$ s does not do a good job of driving the satellite position to zero. This is because the greater weight attached to the final velocity forces it to slow the satellite before it can reach the desired position. We could try changing $\boldsymbol{Q}_{\mathrm{f}}$ to weight errors in

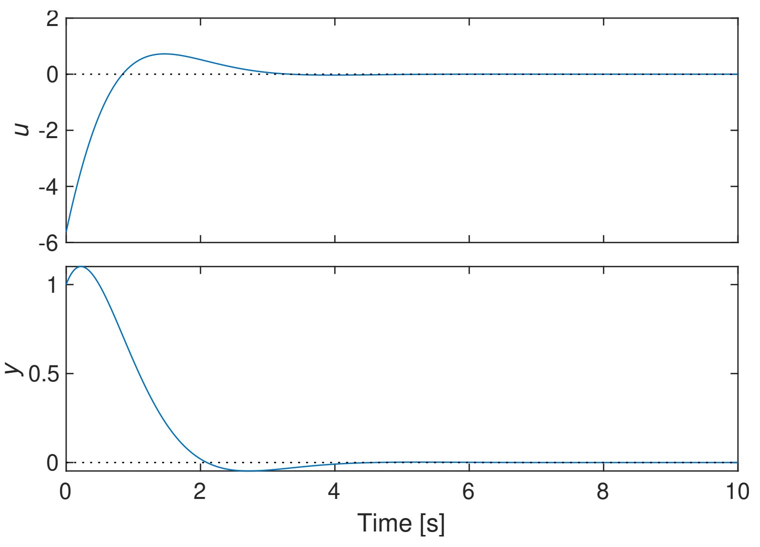


Figure 5.13: Response of the satellite to an initial condition of $\boldsymbol{x}(0) = \begin{bmatrix}1 & 1\end{bmatrix}^{\mathsf{T}}$ using the steady state LQR gain.

final pointing more heavily that errors in final velocity. However, that leads to a solution that is more accurately located at $\theta = 0$, but is still spinning after a second. The reality of this problem is that we have not allowed enough fuel use to complete the manoeuvre successfully within only a second. In figure 5.19 we see the effects of allowing more fuel use by decreasing $\boldsymbol{R}$ to 0.01 from its previous value of 0.1. Notice that the performance is now much better (but not yet perfect). The cost of this rises in fuel use, as $\sum_{t=0}^{1} u(t)^2$ rises from 437 to 2865.

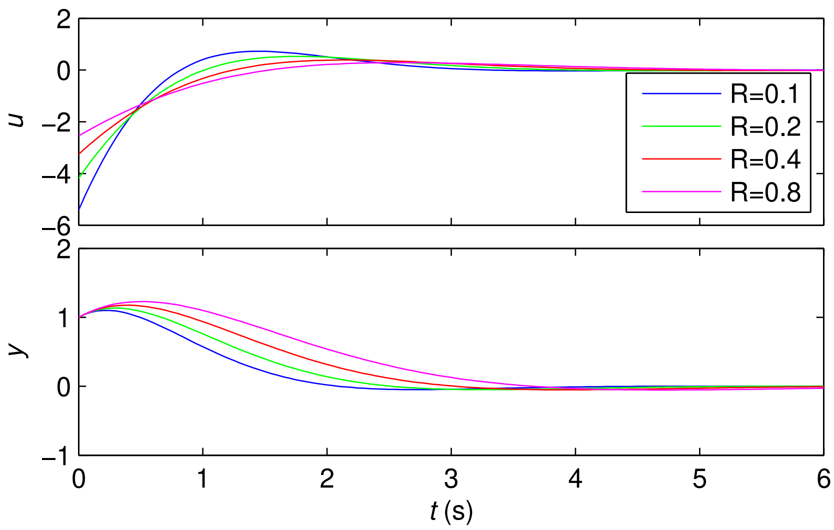


Figure 5.14: Response of the satellite for different values of $\boldsymbol{R}$. The cost associated with state errors ($\boldsymbol{Q}$) is constant for each curve.

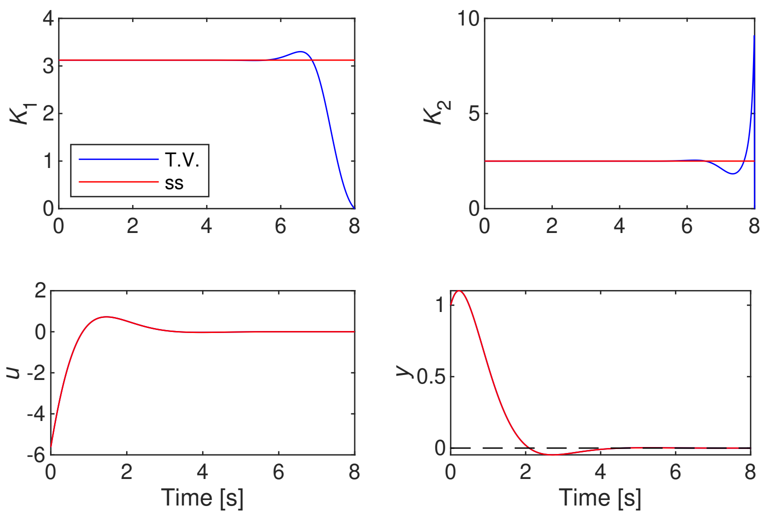


Figure 5.15: Response of the satellite from the initial condition $\boldsymbol{x}(0) = \begin{bmatrix} 1 & 1 \end{bmatrix}^\mathsf{T}$ with $t_\mathrm{f} = 8$ s. The blue curve shows the response of a time varying LQR controller, while the red curve corresponding to the steady state LQR controller is shown for comparison.

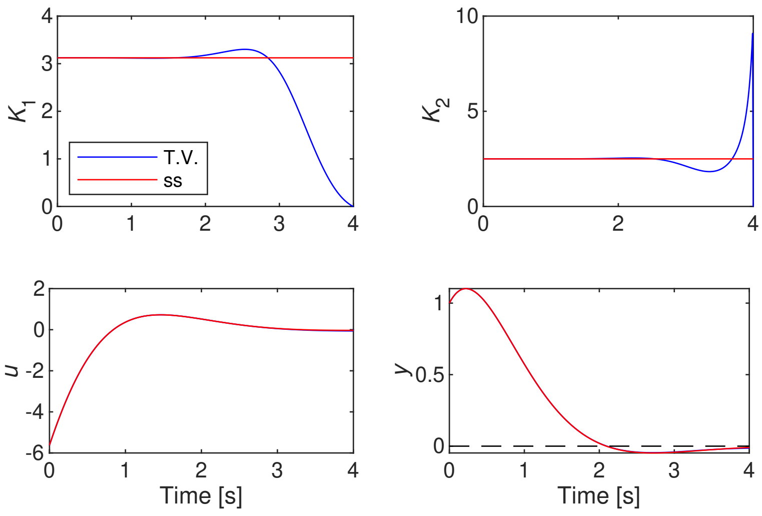


Figure 5.16: Response of the satellite from the initial condition $\boldsymbol{x}(0) = \begin{bmatrix} 1 & 1 \end{bmatrix}^{\mathsf{T}}$ with $t_{\mathrm{f}} = 4$ s. The blue curve shows the response of a time varying LQR controller, while the red curve corresponding to the steady state LQR controller is shown for comparison.

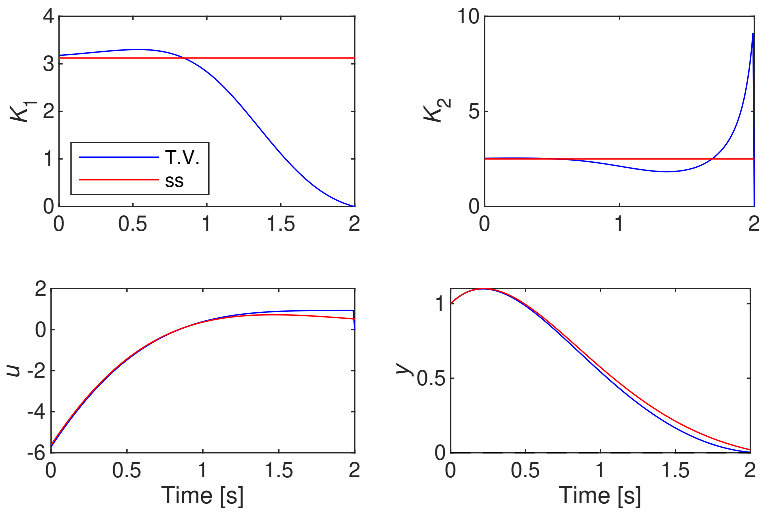


Figure 5.17: Response of the satellite from the initial condition $\boldsymbol{x}(0) = \begin{bmatrix} 1 & 1 \end{bmatrix}^{\mathsf{T}}$ with $t_{\mathrm{f}} = 2$ s. The blue curve shows the response of a time varying LQR controller, while the red curve corresponding to the steady state LQR controller is shown for comparison.

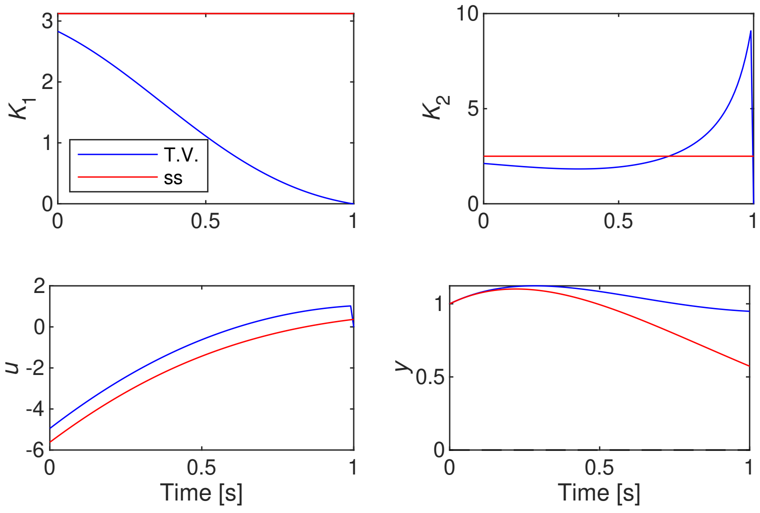


Figure 5.18: Response of the satellite from the initial condition $\boldsymbol{x}(0) = \begin{bmatrix}1 & 1\end{bmatrix}^{\mathsf{T}}$ with $t_{\mathrm{f}} = 1$ s. The blue curve shows the response of a time varying LQR controller, while the red curve corresponding to the steady state LQR controller is shown for comparison.

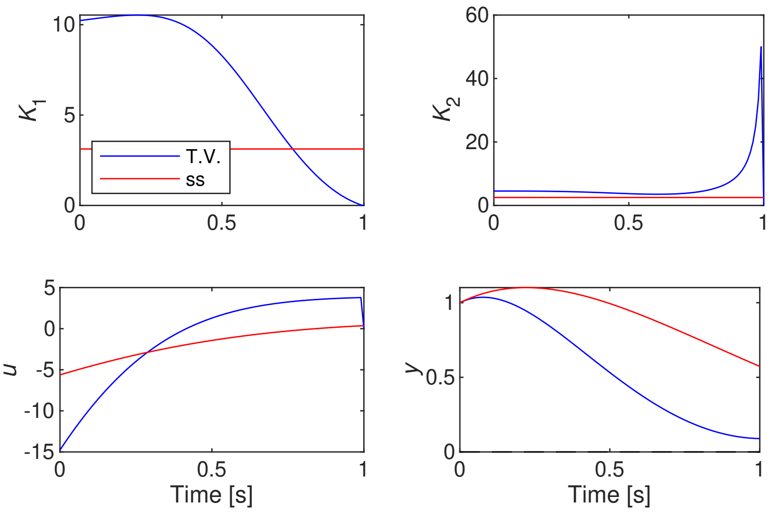


Figure 5.19: Response of the satellite from the initial condition $\boldsymbol{x}(0) = \begin{bmatrix}1 & 1\end{bmatrix}^{\mathsf{T}}$ with $t_{\mathrm{f}} = 1$ s with R reduced so that more fuel use is allowed. The blue curve shows the response of a time varying LQR controller, while the red curve corresponding to the steady state LQR controller is shown for comparison.

5.8 LQR Extensions

The basic LQR structure can be modified in numerous ways to solve a wide variety of different problems. The basic premise of all of these extensions is that we characterise some undesirable property of a system and then penalise it in the LQR metric. The optimal controller will then act to reduce the undesirable behaviour. Some examples include

- Penalising the output ($\boldsymbol{y}$);
- Penalising correlations between $\boldsymbol{x}$ and $\boldsymbol{u}$;
- Ensuring a set rate of decay for transients;
- Penalising changes in input.

5.8.1 Using $\boldsymbol{y}$ in the performance metric

It is sometimes more natural to think about the system trying to constrain deviations in $\boldsymbol{y}$ rather than $\boldsymbol{x}$. We are free to formulate our performance metric as

$$J = \int_0^\infty \boldsymbol{y}^\mathsf{T} \boldsymbol{Q}_y \boldsymbol{y} + \boldsymbol{u}^\mathsf{T} \boldsymbol{R} \boldsymbol{u} \,\mathrm{d}t$$

However, as $\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x}$ we can rewrite this as

$$\begin{aligned} J &= \int_0^t (\boldsymbol{C}\boldsymbol{x})^\mathsf{T} \boldsymbol{Q}_y \boldsymbol{C}\boldsymbol{x} + \boldsymbol{u}^\mathsf{T} \boldsymbol{R} \boldsymbol{u} \,\mathrm{d}t \\ &= \int_0^t \boldsymbol{x}^\mathsf{T} \boldsymbol{C}^\mathsf{T} \boldsymbol{Q}_y \boldsymbol{C}\boldsymbol{x} + \boldsymbol{u}^\mathsf{T} \boldsymbol{R} \boldsymbol{u} \,\mathrm{d}t \\ &= \int_0^t \boldsymbol{x}^\mathsf{T} \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{u}^\mathsf{T} \boldsymbol{R} \boldsymbol{u} \,\mathrm{d}t \end{aligned}$$

with $\boldsymbol{Q} = \boldsymbol{C}^\mathsf{T} \boldsymbol{Q}_y \boldsymbol{C}$.

5.8.2 Introducing $\boldsymbol{z}$

Sometimes you will see a state space system used for LQ described by

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \\ \boldsymbol{y} &= \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u} \\ \boldsymbol{z} &= \boldsymbol{C}_z\boldsymbol{x} + \boldsymbol{D}_z\boldsymbol{u} \end{aligned}$$

Here we make a distinction between the outputs that are measured ($\boldsymbol{y}$) and the 'outputs' that are to be controlled ($\boldsymbol{z}$). The controlled outputs can be arbitrary linear functions of the state that you care about for the particular problem. We would then like $\boldsymbol{z}$ to appears in the LQ metric rather than $\boldsymbol{x}$ or $\boldsymbol{y}$. We can achieve this by using $\boldsymbol{Q} = \boldsymbol{C}_z^\mathsf{T} \boldsymbol{Q}_z \boldsymbol{C}_z$. One reason for doing this is that your choice of basis for $\boldsymbol{x}$ may not make it easy to express the constraints on your problem. You could then choose $\boldsymbol{C}_z$ to map $\boldsymbol{x}$ to a form in which you have physical understanding.

5.8.3 Penalising correlation between $\boldsymbol{u}$ and $\boldsymbol{x}$

Sometimes we do not want a state variable and an input variable to be large at the same time. For example, we might not want current in a heater element to be high when it is already too hot. In such cases we can introduce an extra element into the J functional to penalise this. The extra term is of form $2\boldsymbol{x}^\mathsf{T}\boldsymbol{N}\boldsymbol{u}$, where $\boldsymbol{N}$ is a symmetric positive semidefinite matrix. Large n_{ij} would penalise having x_i and u_j be large simultaneously. This type of penalty term is relatively rare, so can often be neglected. However, one situation in which the cross term does arise in where we are penalising $\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u}$ for $\boldsymbol{D} \neq 0$.

$$\begin{aligned}
J &= \int_0^\infty \boldsymbol{y}^\mathsf{T}\boldsymbol{Q}_y\boldsymbol{y} + \boldsymbol{u}^\mathsf{T}\boldsymbol{R}'\boldsymbol{u}\,\mathrm{d}t \\
&= \int_0^\infty (\boldsymbol{Cx}+\boldsymbol{Du})^\mathsf{T}\boldsymbol{Q}_y(\boldsymbol{Cx}+\boldsymbol{Du}) + \boldsymbol{u}^\mathsf{T}\boldsymbol{R}'\boldsymbol{u}\,\mathrm{d}t \\
&= \int_0^\infty (\boldsymbol{x}^\mathsf{T}\boldsymbol{C}^\mathsf{T}+\boldsymbol{u}^\mathsf{T}\boldsymbol{D}^\mathsf{T})\boldsymbol{Q}_y(\boldsymbol{Cx}+\boldsymbol{Du}) + \boldsymbol{u}^\mathsf{T}\boldsymbol{R}'\boldsymbol{u}\,\mathrm{d}t \\
&= \int_0^\infty \boldsymbol{x}^\mathsf{T}\boldsymbol{C}^\mathsf{T}\boldsymbol{Q}_y\boldsymbol{Cx} + \boldsymbol{x}^\mathsf{T}\boldsymbol{C}^\mathsf{T}\boldsymbol{Q}_y\boldsymbol{Du} + \boldsymbol{u}^\mathsf{T}\boldsymbol{D}^\mathsf{T}\boldsymbol{Q}_y\boldsymbol{Cx} + \boldsymbol{u}^\mathsf{T}(\boldsymbol{D}^\mathsf{T}\boldsymbol{Q}_y\boldsymbol{D}+\boldsymbol{R}')\boldsymbol{u}
\end{aligned}$$

$$\begin{aligned}
\text{Now, } \boldsymbol{x}^\mathsf{T}\boldsymbol{C}^\mathsf{T}\boldsymbol{Q}_y\boldsymbol{Du} + \boldsymbol{u}^\mathsf{T}\boldsymbol{D}^\mathsf{T}\boldsymbol{Q}_y\boldsymbol{Cx} &= \boldsymbol{x}^\mathsf{T}\boldsymbol{C}^\mathsf{T}\boldsymbol{Q}_y\boldsymbol{Du} + (\boldsymbol{x}^\mathsf{T}\boldsymbol{C}^\mathsf{T}\boldsymbol{Q}_y\boldsymbol{Du})^\mathsf{T} \\
&= 2\boldsymbol{x}^\mathsf{T}\boldsymbol{Nu}
\end{aligned}$$

That is, to use a penalty of this form we do normal LQR design after making the replacements:

$$\boldsymbol{C}^\mathsf{T}\boldsymbol{Q}_y\boldsymbol{C} \mapsto \boldsymbol{Q} \qquad\qquad \boldsymbol{D}^\mathsf{T}\boldsymbol{Q}_y\boldsymbol{D}+\boldsymbol{R}' \mapsto \boldsymbol{R} \qquad\qquad \boldsymbol{C}^\mathsf{T}\boldsymbol{Q}_y\boldsymbol{D} \mapsto \boldsymbol{N}$$

Sometimes we want to make sure that the closed loop poles of a CT system are such that the system response decays at least as fast as $\mathrm{e}^{\alpha t}$. This is easy when using pole placement, as we just ensure that the poles are sufficiently far to the left of the s-plane. We can force an LQR controller to do this by weighting $\boldsymbol{x}$ and $\boldsymbol{u}$ with exponential functions of time. That is, we penalise $\boldsymbol{x}$ and $\boldsymbol{u}$ exponentially more as time passes.

$$\begin{aligned}
J &= \int_0^\infty (\mathrm{e}^{\alpha t}\boldsymbol{x})^\mathsf{T}\boldsymbol{Q}(\mathrm{e}^{\alpha t}\boldsymbol{x}) + (\mathrm{e}^{\alpha t}\boldsymbol{u})^\mathsf{T}\boldsymbol{R}(\mathrm{e}^{\alpha t}\boldsymbol{u})\,\mathrm{d}t \\
&= \int_0^\infty \mathrm{e}^{2\alpha t}\left(\boldsymbol{x}^\mathsf{T}\boldsymbol{Qx} + \boldsymbol{u}^\mathsf{T}\boldsymbol{Ru}\right)\,\mathrm{d}t
\end{aligned}$$

The fact that this functional is time dependent makes it awkward to work with. We would like to make this look like a normal LQR problem, so that we can design a controller with the same procedure as before.

Let's define a modified new state representation $\boldsymbol{x}' = \mathrm{e}^{\alpha t}\boldsymbol{x}$ and $\boldsymbol{u}' = \mathrm{e}^{\alpha t}\boldsymbol{u}$ and find a modified state space model that describes the dynamics of $\boldsymbol{x}'$.

$$\begin{aligned}
\frac{\mathrm{d}\boldsymbol{x}'}{\mathrm{d}t} &= \frac{\mathrm{d}\mathrm{e}^{\alpha t}\boldsymbol{x}}{\mathrm{d}t} \\
&= \alpha\mathrm{e}^{\alpha t}\boldsymbol{x} + \mathrm{e}^{\alpha t}\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} \qquad \text{(Product rule)} \\
&= \alpha\mathrm{e}^{\alpha t}\boldsymbol{x} + \mathrm{e}^{\alpha t}\left(\boldsymbol{Ax}+\boldsymbol{Bu}\right) \\
&= \alpha\boldsymbol{x}' + \boldsymbol{Ax}' + \boldsymbol{Bu}' \\
&= (\boldsymbol{A}+\alpha\boldsymbol{I})\boldsymbol{x}' + \boldsymbol{Bu}' \\
&= \boldsymbol{A}'\boldsymbol{x}' + \boldsymbol{Bu}'
\end{aligned}$$

That is, we have found a state space model for our new exponentially weighted state by changing $\boldsymbol{A}$ to $(\boldsymbol{A}+\alpha\boldsymbol{I})$. We can use this new state space model to design a controller in the new state.

Solving the appropriate Riccati equation using $(\boldsymbol{A}+\alpha\boldsymbol{I})$ will generate an optimal controller to form the feedback signal $\boldsymbol{u}' = -\boldsymbol{K}'\boldsymbol{x}'$. Of course, we don't want to estimate $\boldsymbol{x}'$, nor apply $\boldsymbol{u}'$ to our system; we want to use $\boldsymbol{x}$ and $\boldsymbol{u}$.

$$\begin{aligned}
\boldsymbol{u}' &= -\boldsymbol{K}'\boldsymbol{x}' \\
\mathrm{e}^{\alpha t}\boldsymbol{u} &= -\mathrm{e}^{\alpha t}\boldsymbol{Kx} \\
\boldsymbol{u} &= -\boldsymbol{K}'\boldsymbol{x}
\end{aligned}$$

That is, we just directly use the $\boldsymbol{K}$ found with our modified system. Remember though that $\boldsymbol{P}$ calculated in solving the Riccati equation will be associated with the cost to go in the modified state variables.

Notice that applying α shift will not alter the controllability of a system. Let's examine the controllability matrix for the α shifted system.

$$\begin{aligned} \boldsymbol{\mathcal{M}}_{\mathrm{c}} &= \begin{bmatrix} \boldsymbol{B}(\boldsymbol{A}')^n & \boldsymbol{B}(\boldsymbol{A}')^{n-1} & \dots & \boldsymbol{B} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{B}(\boldsymbol{A}+\alpha\boldsymbol{I})^n & \boldsymbol{B}(\boldsymbol{A}+\alpha\boldsymbol{I})^{n-1} & \dots & \boldsymbol{B} \end{bmatrix} \end{aligned}$$

Let's look at the k-th term in this matrix and use the binomial expansion.

$$\boldsymbol{B}(\boldsymbol{A}+\alpha\boldsymbol{I})^k = \boldsymbol{B}\left[\boldsymbol{A}^k + \alpha\binom{k}{1}\boldsymbol{A}^{k-1} + \alpha^2\binom{k}{2}\boldsymbol{A}^{k-2} + \ldots + \boldsymbol{I}\right]$$

We can see that k-th term of the controllability contributes a potentially new term $\boldsymbol{B}\boldsymbol{A}^k$ to the controllability matrix, but that all of the other terms would already be included in $\boldsymbol{\mathcal{M}}_{\mathrm{c}}$ by the lower powers of $\boldsymbol{A}$. Thus the α shift does not affect the controllability of the system.

We could similarly show that the observability of a system is not altered by an α shift. However, in an α shifted system we are forcing the closed loop poles to lie in a given region. We may therefore alter the detectability or stabilizability of a system by forcing the closed loop poles into the left of the s-plane, or the interior of the z-plane.

An alternate method for reducing the high frequency content in the control signal is to directly penalise those changes. That is, rather than penalising $\boldsymbol{u}$ we would like to penalise $\Delta\boldsymbol{u}$. That is, we can take a system described by $\boldsymbol{x}(t+1) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)$ and rewrite it as $\boldsymbol{x}'(t+1) = \boldsymbol{A}'\boldsymbol{x}'(t) + \boldsymbol{B}'(\Delta\boldsymbol{u})(t)$. (The notation $(\Delta\boldsymbol{u})(t)$ here indicates the change in $\boldsymbol{u}$ between times $t-1$ and t.) We will do this by augmenting the model so that it generates $\boldsymbol{u}$ from $\Delta\boldsymbol{u}$.

$$\begin{aligned} \begin{bmatrix} \boldsymbol{x}(t+1) \\ \boldsymbol{u}(t) \end{bmatrix} &= \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ 0 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(t) \\ \boldsymbol{u}(t-1) \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{I} \end{bmatrix} (\Delta\boldsymbol{u})(t) \\ \implies \boldsymbol{x}'(t+1) &= \boldsymbol{A}'\boldsymbol{x}'(t) + \boldsymbol{B}'\boldsymbol{u}'(t) \end{aligned}$$

We can use conventional LQR techniques to design a controller for this, using $\boldsymbol{Q}' = \begin{bmatrix} \boldsymbol{Q} & 0 \\ 0 & \boldsymbol{R} \end{bmatrix}$ and $\boldsymbol{R}'$ penalises changes in the input.