

LQR Tracking Example

Richard M. Murray, 25 Jan 2022

This example uses a linear system to show how to implement LQR based tracking and some of the tradeoffs between feedforward and feedback. Integral action is also implemented.

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
import control as ct
```

System definition

We use a simple linear system to illustrate the concepts. This system corresponds to the linearized lateral dynamics of a vehicle driving down a road at 10 m/s.

```
In [2]: # Define a simple linear system that we want to control
sys = ct.ss([[0, 10], [-1, 0]], [[0], [1]], np.eye(2), 0, name='sys')
print(sys)
```

```
<LinearIOSystem>: sys
Inputs (1): ['u[0]']
Outputs (2): ['y[0]', 'y[1]']
States (2): ['x[0]', 'x[1]']
```

```
A = [[ 0. 10.]
      [-1.  0.]
```

```
B = [[0.]
      [1.]
```

```
C = [[1. 0.]
      [0. 1.]
```

```
D = [[0.]
      [0.]
```

Controller design

We start by defining the equilibrium point that we plan to stabilize.

```
In [3]: # Define the desired equilibrium point for the system
x0 = np.array([2, 0])
u0 = np.array([2])
Tf = 4
```

Then construct a simple LQR controller (gain matrix) and create the controller + closed loop system models:

```
In [4]: # Construct an LQR controller for the system
K, _, _ = ct.lqr(sys, np.eye(sys.nstates), np.eye(sys.ninputs))
ctrl, clsys = ct.create_statefbk_iosystem(sys, K)
print(ctrl)
print(clsys)

<LinearIOSystem>: sys [2]
Inputs (5): ['xd[0]', 'xd[1]', 'ud[0]', 'y[0]', 'y[1]']
Outputs (1): ['u[0]']
States (0): []

A = []

B = []

C = []

D = [[ 0.41421356  3.04701021  1.          -0.41421356 -3.04701021]]

<LinearICSystem>: u[0]
Inputs (3): ['xd[0]', 'xd[1]', 'ud[0]']
Outputs (3): ['y[0]', 'y[1]', 'u[0]']
States (2): ['sys_x[0]', 'sys_x[1]']

A = [[ 0.          10.         ]
      [-1.41421356 -3.04701021]]

B = [[0.          0.          0.         ]
      [0.41421356  3.04701021  1.         ]]

C = [[ 1.          0.         ]
      [ 0.          1.         ]
      [-0.41421356 -3.04701021]]

D = [[0.          0.          0.         ]
      [0.          0.          0.         ]
      [0.41421356  3.04701021  1.         ]]
```

Note that the name of the second system is `u[0]`. This is a bug in control-0.9.3 that will be fixed in a [future release](#).

System simulations

Baseline controller

To see how the baseline controller performs, we ask it to track a step change in (xd, ud):

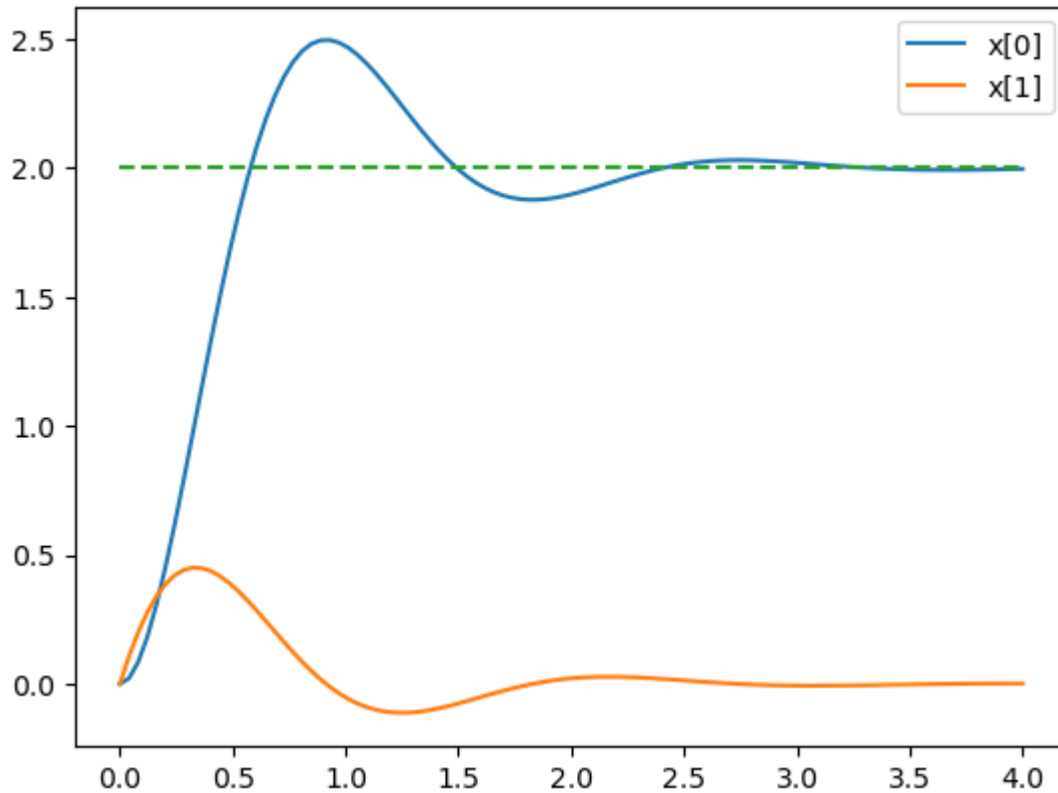
```
In [5]: # Plot the step response with respect to the reference input
tvec = np.linspace(0, Tf, 100)
```

```

xd = x0
ud = u0

# U = np.hstack([xd, ud])
U = np.outer(np.hstack([xd, ud]), np.ones_like(tvec))
time, output = ct.input_output_response(clsys, tvec, U)
plt.plot(time, output[0], time, output[1])
plt.plot([time[0], time[-1]], [xd[0], xd[0]], '--');
plt.legend(['x[0]', 'x[1]']);

```



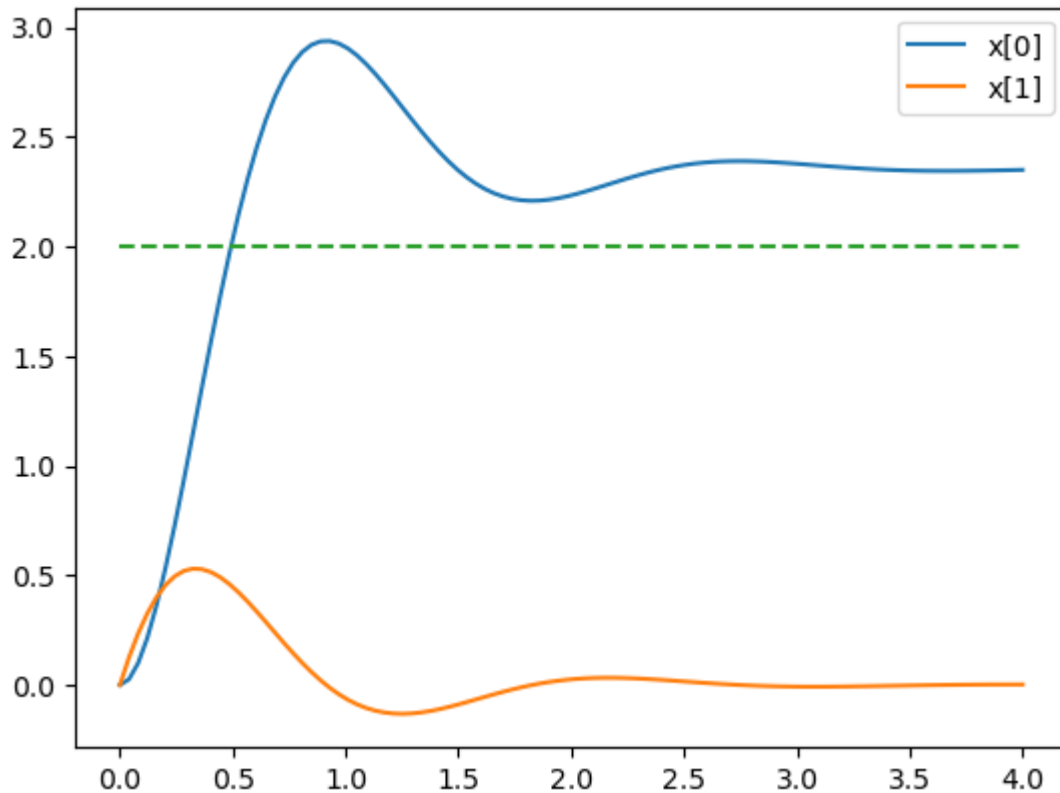
Disturbance rejection

We add a disturbance to the system by modifying `ud` (since this enters directly at the system input `u`).

```

In [6]: # Resimulate with a disturbance input
delta = 0.5
U = np.outer(np.hstack([xd, ud + delta]), np.ones_like(tvec))
time, output = ct.input_output_response(clsys, tvec, U)
plt.plot(time, output[0], time, output[1])
plt.plot([time[0], time[-1]], [xd[0], xd[0]], '--')
plt.legend(['x[0]', 'x[1]']);

```



We see that this leads to steady state error, since some amount of system error is required to generate the force to offset the disturbance.

Integral feedback

A standard approach to compensate for constant disturbances is to use integral feedback. To do this, we have to decide what output we want to track and create a new controller with integral feedback.

We do this by creating an "augmented" system that includes the dynamics of the process along with the dynamics of the controller (= integrators for the errors that we choose):

```
In [7]: # Create a controller with integral feedback
C = np.array([[1, 0]])

# Define an augmented state space for use with LQR
A_aug = np.block([
    [sys.A, np.zeros((sys.nstates, 1))],
    [C, 0]
])
B_aug = np.vstack([sys.B, 0])
print("A =", A_aug, "\nB =", B_aug)
```

```
A = [[ 0. 10.  0.]
      [-1.  0.  0.]
      [ 1.  0.  0.]]
B = [[0.]
      [1.]
      [0.]]
```

Now generate an LQR controller for the the augmented system:

```
In [8]: # Create an LQR controller for the augmented system
K_aug, _, _ = ct.lqr(
    A_aug, B_aug, np.diag([1, 1, 1]), np.eye(sys.ninputs))
print(K_aug)

[[0.65930346  3.76643986  1.          ]]
```

We can think about this gain as $K_{\text{aug}} = [K, k_i]$ and the resulting controller becomes

$$u = u_d - K(x - x_d) - k_i \int_0^t (y - y_d) d\tau.$$

```
In [9]: # Construct an LQR controller for the system
integral_ctrl, sys_integral = ct.create_statefbk_iosystem(sys, K_aug, integrator)
print(integral_ctrl)
print(sys_integral)

# Resimulate with a disturbance input
delta = 0.5
U = np.outer(np.hstack([xd, ud + delta]), np.ones_like(tvec))
time, output = ct.input_output_response(sys_integral, tvec, U)
plt.plot(time, output[0], time, output[1])
plt.plot([time[0], time[-1]], [xd[0], xd[0]], '--')
plt.legend(['x[0]', 'x[1]']);
```

```
<LinearIOSystem>: sys[4]
Inputs (5): ['xd[0]', 'xd[1]', 'ud[0]', 'y[0]', 'y[1]']
Outputs (1): ['u[0]']
States (1): ['x[0]']
```

```
A = [[0.]]
```

```
B = [[-1.  0.  0.  1.  0.]]
```

```
C = [[-1.]]
```

```
D = [[ 0.65930346  3.76643986  1.          -0.65930346 -3.76643986]]
```

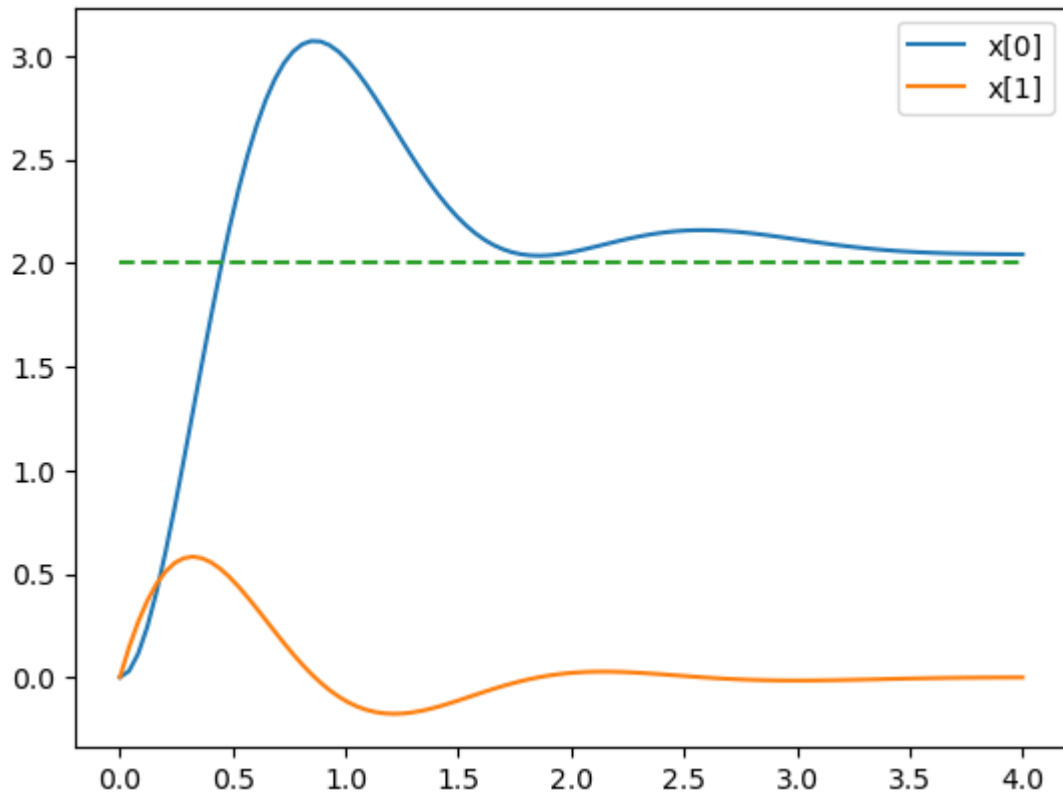
```
<LinearICSystem>: u[0]
Inputs (3): ['xd[0]', 'xd[1]', 'ud[0]']
Outputs (3): ['y[0]', 'y[1]', 'u[0]']
States (3): ['sys_x[0]', 'sys_x[1]', 'sys[4]_x[0]']
```

```
A = [[ 0.          10.          0.          ]
      [-1.65930346 -3.76643986 -1.          ]
      [ 1.          0.          0.          ]]
```

```
B = [[ 0.          0.          0.          ]
      [ 0.65930346  3.76643986  1.          ]
      [-1.          0.          0.          ]]
```

```
C = [[ 1.          0.          0.          ]
      [ 0.          1.          0.          ]
      [-0.65930346 -3.76643986 -1.          ]]
```

```
D = [[0.          0.          0.          ]
      [0.          0.          0.          ]
      [0.65930346  3.76643986  1.          ]]
```



Things to try

- Play around with the gains and see whether you can reduce the overshoot (50%!)
- Try following more complicated trajectories (hint: linear systems are differentially flat...)

In []: