1. The purpose of this exercise is to extend the backstepping procedure beyond pure integrator backstepping. Consider the system

\[ \dot{x} = f(x) + g(x)\xi \]
\[ \dot{\xi} = f_1(x, \xi) + g_1(x, \xi)u \]

where \( x, \xi, u \) are all scalar variables and \( g_1(x, \xi) \neq 0 \) for all \( x, \xi \). Assume that a Lyapunov function \( V_0(x) \) and a feedback law \( k_0(x) \) (with \( k_0(0) = 0 \)) are known such that

\[ \frac{\partial V_0}{\partial x} (f(x) + g(x)k_0(x)) \leq -W(x) < 0 \quad \forall x \neq 0 \]

Construct a control Lyapunov function \( V_1(x, \xi) \) and a stabilizing feedback law \( k_1(x, \xi) \) for the 2-D system.

**Solution.** Consider the candidate Lyapunov function

\[ V_1(x, \xi) := V_0(x) + \frac{(\xi - k_0(x))^2}{2}. \]

Its derivative along the state trajectory is

\[ \dot{V}_1(x, \xi) = \frac{\partial V_0}{\partial x} (f(x) + g(x)\xi) + (\xi - k_0(x)) \left( f_1(x, \xi) + g_1(x, \xi)u - \frac{\partial k_0}{\partial x} (f(x) + g(x)\xi) \right) \]
\[ = \frac{\partial V_0}{\partial x} (f(x) + g(x)k_0(x)) + (\xi - k_0(x)) \left( \frac{\partial V_0}{\partial x} g(x) + f_1(x, \xi) + g_1(x, \xi)u - \frac{\partial k_0}{\partial x} (f(x) + g(x)\xi) \right) \]
\[ \leq -W(x) + (\xi - k_0(x)) \left( \frac{\partial V_0}{\partial x} g(x) + f_1(x, \xi) + g_1(x, \xi)u - \frac{\partial k_0}{\partial x} (f(x) + g(x)\xi) \right). \]

Note the \( g_1(x, \xi) \neq 0 \). Letting

\[ u := k_1(x, \xi) := \frac{1}{g_1(x, \xi)} \left( -\frac{\partial V_0}{\partial x} g(x) - f_1(x, \xi) + \frac{\partial k_0}{\partial x} (f(x) + g(x)\xi) - (\xi - k_0(x)) \right) \]

gives that

\[ \dot{V}_1(x, \xi) = -W(x) - (\xi - k_0(x))^2 \leq 0, \]

and \( \dot{V}_1(x, \xi) < 0 \) for all \( (x, \xi) \neq (0, k_0(0)) = (0, 0) \). Hence \( V_1(x, \xi) \) is a control Lyapunov function, and \( k_1(x, \xi) \) is a stabilizing feedback law.

2. Consider the system

\[ \dot{x} = \theta x + \xi_1 \]
\[ \dot{\xi}_1 = \xi_2 \]
\[ \dot{\xi}_2 = u \]

where \( x, \xi_1, \xi_2, u \) are all scalar variables and \( \theta \) is an unknown parameter. Continue the adaptive backstepping procedure given in Section 5.2 in the lecture notes and design an adaptive control law that asymptotically stabilizes this system.

**Solution.** Consider the candidate Lyapunov function

\[ V_2(x, \xi_1, \xi_2, \dot{\theta}) := V_1(x, \dot{\theta}, \xi_1) + \frac{1}{2}(\xi_2 - k_1)^2 \]

where

\[ V_1(x, \dot{\theta}, \xi_1) = \frac{1}{2}(x^2 + (\dot{\theta} - \theta)^2) + \frac{1}{2}(\xi_1 - k_0)^2, \]
and

\[ k_0(x, \dot{\theta}) := - (\dot{\theta} + 1) x, \quad (1) \]

\[ k_1(x, \dot{\theta}, \xi_1) := - x + \frac{\partial k_0}{\partial x} (\dot{\theta} x + \xi_1) + \frac{\partial k_0}{\partial \theta} (\tau_0 + \tau_1) - (\xi_1 - k_0), \quad (2) \]

\[ \tau_0(x) := x^2, \quad (3) \]

\[ \tau_1(x, \dot{\theta}, \xi_1) := - \frac{\partial k_0}{\partial x} (\xi_1 - k_0) x \quad (4) \]

as defined in the lecture notes. Calculating the derivative of \( V_1 \) along solutions gives that

\[
\dot{V}_1(x, \dot{\theta}, \xi_1) = x(\theta x + \xi_1) + (\dot{\theta} - \theta) \dot{\theta} + (\xi_1 - k_0) \left( \xi_2 - \frac{\partial k_0}{\partial x} (\theta x + \xi_1) - \frac{\partial k_0}{\partial \theta} \dot{\theta} \right) \\
= x(\theta x + k_0) + (\dot{\theta} - \theta) \dot{\theta} + (\xi_1 - k_0) \left( \xi_2 + x - \frac{\partial k_0}{\partial x} (\theta x + \xi_1) - \frac{\partial k_0}{\partial \theta} \dot{\theta} \right) \\
\overset{(1)}{=} - x^2 + (\dot{\theta} - \theta) (\dot{\theta} - x^2) + (\xi_1 - k_0) \left( \xi_2 + x - \frac{\partial k_0}{\partial x} (\theta x + \xi_1) - \frac{\partial k_0}{\partial \theta} \dot{\theta} \right) \\
= - x^2 + (\dot{\theta} - \theta) \left( \dot{\theta} - x^2 + \frac{\partial k_0}{\partial x} (\xi_1 - k_0) x \right) + (\xi_1 - k_0) \left( \xi_2 + x - \frac{\partial k_0}{\partial x} (\theta x + \xi_1) - \frac{\partial k_0}{\partial \theta} \dot{\theta} \right) \\
\overset{(2)(3)(4)}{=} - x^2 - (\xi_1 - k_0)^2 + (\dot{\theta} - \theta) (\dot{\theta} - \tau_0 - \tau_1) + (\xi_1 - k_0) \left( \xi_2 - k_1 - \frac{\partial k_0}{\partial \theta} (\dot{\theta} - \tau_0 - \tau_1) \right).
\]

Then calculating the derivative of \( V_2 \) along solutions gives that

\[
\dot{V}_2(x, \dot{\theta}, \xi_1, \xi_2) = \dot{V}_1(x, \dot{\theta}, \xi_1) + (\xi_2 - k_1) \left( u - \frac{\partial k_1}{\partial x} (\theta x + \xi_1) - \frac{\partial k_1}{\partial \theta} \dot{\theta} - \frac{\partial k_1}{\partial \xi_1} \xi_2 \right) \\
= - x^2 - (\xi_1 - k_0)^2 + (\dot{\theta} - \theta) (\dot{\theta} - \tau_0 - \tau_1) + (\xi_1 - k_0) \left( \xi_2 - k_1 - \frac{\partial k_0}{\partial \theta} (\dot{\theta} - \tau_0 - \tau_1) \right) \\
+ (\xi_2 - k_1) \left( u - \frac{\partial k_1}{\partial x} (\theta x + \xi_1) - \frac{\partial k_1}{\partial \theta} \dot{\theta} - \frac{\partial k_1}{\partial \xi_1} \xi_2 \right) \\
= - x^2 - (\xi_1 - k_0)^2 + (\dot{\theta} - \theta) \left( \dot{\theta} - \tau_0 - \tau_1 + \frac{\partial k_1}{\partial x} (\xi_2 - k_1) x \right) \\
+ (\xi_1 - k_0) \left( \xi_2 - k_1 - \frac{\partial k_0}{\partial \theta} (\dot{\theta} - \tau_0 - \tau_1) \right) \\
+ (\xi_2 - k_1) \left( u - \frac{\partial k_1}{\partial x} (\theta x + \xi_1) - \frac{\partial k_1}{\partial \theta} \dot{\theta} - \frac{\partial k_1}{\partial \xi_1} \xi_2 \right) \\
= - x^2 - (\xi_1 - k_0)^2 + (\dot{\theta} - \theta) \left( \dot{\theta} - \tau_0 - \tau_1 + \frac{\partial k_1}{\partial x} (\xi_2 - k_1) x \right) \\
+ (\xi_1 - k_0) \left( - \frac{\partial k_0}{\partial \theta} \left( \dot{\theta} - \tau_0 - \tau_1 + \frac{\partial k_1}{\partial x} (\xi_2 - k_1) x \right) \right) \\
+ (\xi_2 - k_1) \left( u - \frac{\partial k_1}{\partial x} (\theta x + \xi_1) - \frac{\partial k_1}{\partial \theta} \dot{\theta} - \frac{\partial k_1}{\partial \xi_1} \xi_2 + (\xi_1 - k_0) \left( 1 + \frac{\partial k_0}{\partial \theta} \frac{\partial k_1}{\partial x} x \right) \right).
\]
Hence setting
\[ \dot{\tau} := \tau_0 + \tau_1 + \tau_2, \]
\[ \tau_2 := -\frac{\partial k_1}{\partial x} (\xi_2 - k_1)x, \]
\[ u := \frac{\partial k_1}{\partial x} (\dot{x} + \xi_1) + \frac{\partial k_1}{\partial \theta} \dot{\theta} + \frac{\partial k_1}{\partial \xi_1} \xi_2 - (\xi_1 - k_0) \left( 1 + \frac{\partial k_0}{\partial \theta} \frac{\partial k_1}{\partial x} x \right) - (\xi_2 - k_1) \]
gives
\[ \dot{V}_2(x, \theta, \xi_1, \xi_2) = -x^2 - (\xi_1 - k_0)^2 - (\xi_2 - k_1)^2 < 0 \quad \forall (x, \xi_1, \xi_2) \neq (0, 0, 0). \]

Then Theorem 2 implies that \( x, \dot{\theta}, \xi_1, \xi_2 \) are bounded and \( x, \xi_1, \xi_2 \to 0. \)

3. Prove Lemma 4 from the lecture notes. Be sure to give explicit expressions for \( \lambda, c \) in terms of \( \alpha_0, T_0 \) and vice versa.

**Solution.** Recall that
\[ \tilde{\theta}(t) = e^{-\gamma \int_{t_0}^t u(s)^2 ds} \tilde{\theta}(t_0) \]

For all \( s \in \mathbb{R} \), let \( \lfloor s \rfloor \) denote the largest integer less or equal to \( s \), i.e., \( \lfloor s \rfloor := \max\{ z \in \mathbb{Z} | z \leq s \} \).

First, suppose there exist \( \alpha_0, T_0 > 0 \) such that \( u \) is PE. Then
\[
|\tilde{\theta}(t)| = e^{-\gamma \int_{t_0}^t u(s)^2 ds} |\tilde{\theta}(t_0)| \leq e^{-\gamma \int_{t_0}^{t_0 + \lfloor \frac{t - t_0}{T_0} \rfloor T_0} u(s)^2 ds} |\tilde{\theta}(t_0)| \leq e^{-\gamma \int_{t_0}^{t_0 + \lfloor \frac{t - t_0}{T_0} \rfloor T_0} u(s)^2 ds} |\tilde{\theta}(t_0)| \leq e^{-\gamma \int_{t_0}^{t_0 + \lfloor \frac{t - t_0}{T_0} \rfloor T_0} u(s)^2 ds} |\tilde{\theta}(t_0)| \]
for all \( t \geq t_0 \geq 0 \). Hence \( \tilde{\theta} \) is UEC with \( c := e^{\gamma \alpha_0 T_0} > 0 \) and \( \lambda := \gamma \alpha_0 > 0 \).

Second, suppose there exist \( c, \lambda > 0 \) such that \( \tilde{\theta} \) is UEC. Then
\[ e^{-\gamma \int_{t_0}^t u(s)^2 ds} |\tilde{\theta}(t)| \leq ce^{-\lambda (t - t_0)} |\tilde{\theta}(t)| \quad \forall \tau \geq t \geq 0. \]

Taking logarithms on both sides gives
\[ \int_{t}^{\tau} u(s)^2 ds \geq \frac{\lambda}{\gamma} (t - \tau) - \frac{\log c}{\gamma} \quad \forall \tau \geq t \geq 0. \]

By selecting \( \alpha_0 < \lambda/\gamma \) and \( T_0 \geq (\log c)/(\lambda - \gamma \alpha_0) \) and letting \( \tau = t + T_0 \), we get
\[ \int_{t}^{t+T_0} u(s)^2 ds \geq \frac{\lambda}{\gamma} T_0 - \frac{\log c}{\gamma} \geq \frac{\lambda}{\gamma} T_0 - \frac{\lambda - \gamma \alpha_0}{\gamma} T_0 = \alpha_0 T_0 \quad \forall t \geq 0, \]
i.e., \( u \) is PE.

4. As defined in class, a vector-valued signal \( \phi \) is persistently exciting (PE) if for some \( \alpha_0, T_0 > 0 \) we have \( \int_{t}^{t+T_0} \phi(s) \phi^T(s) ds \geq \alpha_0 T_0 I \) for all \( t \), where the inequality is to be understood in the matrix sense. Prove that the vector signal
\[ \phi(t) := \begin{pmatrix} A \sin(\omega t + \alpha) \\ \sin \omega t \end{pmatrix} \]
is PE under suitable constraints on the numbers \( A, \omega, \alpha \). Be sure to specify these constraints.
Solution. First we notice that if \( \omega = 0 \) then \( \phi \) cannot be PE. Hence integrating the matrix gives

\[
\int_{t}^{t+T_{0}} \phi(s)\phi(s)^{\top} \, ds = \int_{t}^{t+T_{0}} \begin{bmatrix}
A^{2}\sin^{2}(\omega s + \alpha) & A\sin(\omega s + \alpha)\sin\omega s \\
A\sin(\omega s + \alpha)\sin\omega s & \sin^{2}\omega s
\end{bmatrix} \, ds
\]

\[
= \int_{t}^{t+T_{0}} \frac{1}{2} \begin{bmatrix}
A^{2} - A^{2}\cos(2(\omega s + \alpha)) & A\cos(2s\omega s + \alpha) - A\cos(2\omega s + \alpha) \\
A\cos(2\omega s + \alpha) - A\cos(2s\omega s + \alpha) & 1 - \cos(2\omega s)
\end{bmatrix} \, ds
\]

\[
= \frac{T_{0}}{2} \begin{bmatrix}
A^{2} & A\cos\alpha \\
A\cos\alpha & 1
\end{bmatrix} + \frac{1}{4\omega} \begin{bmatrix}
A^{2}\sin(2(\omega s + \alpha)) & A\sin(2\omega s + \alpha) \\
A\sin(2\omega s + \alpha) & \sin(2\omega s)
\end{bmatrix}^{t+T_{0}}_{s=t}
\]

\[
= \frac{T_{0}}{2} \begin{bmatrix}
A^{2} & A\cos\alpha \\
A\cos\alpha & 1
\end{bmatrix} + \frac{\sin(\omega T_{0})}{2\omega} \begin{bmatrix}
A^{2}\cos(2\omega t + 2\alpha + \omega T_{0}) & A\cos(2\omega t + 2\alpha + \omega T_{0}) \\
A\cos(2\omega t + 2\alpha + \omega T_{0}) & \cos(2\omega t + 2\omega T_{0})
\end{bmatrix}
\]

Letting \( T_{0} = \pi/|\omega| \) gives

\[
H(t) := \int_{t}^{t+T_{0}} \phi(s)\phi(s)^{\top} \, ds = \frac{\pi}{2|\omega|} \begin{bmatrix}
A^{2} & A\cos\alpha \\
A\cos\alpha & 1
\end{bmatrix} \quad \forall \, t.
\]

Note that we can always select a sufficiently small \( \alpha_{0} > 0 \) so that \( H(t) \geq \alpha_{0}T_{0}I \) for all \( t \), provided that \( H(t) > 0 \) for all \( t \), or equivalently, that both of the principal minors of \( H \) are always positive, that is,

\[
m_{1} = \frac{\pi}{2|\omega|} A^{2} > 0,
\]

and

\[
m_{2} = \frac{\pi}{2|\omega|} \begin{bmatrix}
A^{2} & A\cos\alpha \\
A\cos\alpha & 1
\end{bmatrix} = \frac{\pi}{2|\omega|} A^{2}(1 - \cos^{2}\alpha) > 0.
\]

Therefore, \( \phi \) is PE if \( \omega \neq 0 \), \( A \neq 0 \) and \( \alpha \neq n\pi, n \in \mathbb{Z} \).

5. Exercise 4.2 on page 246 in [Ioannou-Sun].

Consider the second order stable system

\[
\dot{x} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & 0 \end{bmatrix} x + \begin{bmatrix} b_{1} \\ b_{2} \end{bmatrix} u
\]

where \( x, u \) are available for measurement, \( u \in \mathcal{L}_{\infty} \) and \( a_{11}, a_{12}, a_{21}, b_{1}, b_{2} \) are unknown parameters. Design an on-line estimator to estimate the unknown parameters. Simulate your scheme using \( u = 10\sin 2t \) and

\[
A = \begin{bmatrix} -0.25 & 3 \\ -5 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 2.2 \end{bmatrix}
\]

Repeat the simulation when \( u = 10\sin 2t + 7\cos 3.6t \).

Solution. Since the plant is stable, consider the estimator

\[
\dot{x} = A_{m}(\dot{x} - x) + \dot{A}x + \dot{B}u,
\]

where \( A_{m} \) is Hurwitz, and \( \dot{x}, \dot{A}, \dot{B} \) are the estimations of \( x, A, B \) respectively. The update laws are given by

\[
\dot{A} = -P(\dot{x} - x)x^{\top},
\]

\[
\dot{B} = -P(\dot{x} - x)u,
\]

where \( P \) is the solution to the Lyapunov function \( A_{m}^{\top}P + PA_{m} = -I \). In the simulation, we select \( A_{m} = -5I \), which gives \( P = 0.1I \).

\[1\] Use the link on the class homepage to access the electronic version of [Ioannou-Sun].
1) The simulation using $u = 10 \sin(2t)$ can be found in Fig. 1. We observe that $e \to 0$, but $\hat{A} \not\to A$ and $\hat{B} \not\to B$. This is due to the input $u$ not being sufficiently rich (in term of frequency components).

2) The simulation result using $u = 10 \sin(2t) + 7 \cos(3.6t)$ can be found in Fig. 2. We observe that $e \to 0$, $\hat{A} \to A$ and $\hat{B} \to B$ as the input has one more frequency component.

Fig. 1: Simulation result using $u = 10 \sin(2t)$

Fig. 2: Simulation result using $u = 10 \sin(2t) + 7 \cos(3.6t)$
Problem 5 (optional)

First lets make the necessary imports

```
In [1]: from scipy.integrate import odeint
import numpy as np
from numpy import sin, cos
import matplotlib.pyplot as plt
from matplotlib import rcParams
from scipy.linalg import solve_lyapunov
```

Recall the system and its estimates’ dynamics are:

\[ \dot{x} = Ax + Bu \]
\[ \dot{x} = A_m (\hat{x} - x) + \hat{A}x + \hat{B}u \]

while the parameter estimates are given by the equations:

\[ \dot{\hat{A}} = -P (\hat{x} - x)x^T \]
\[ \dot{\hat{B}} = -P (\hat{x} - x)u \]

Now we can first define all the dynamics involved as Python functions.

```
In [2]: def ahat_dynamics(x, x_hat, P):
    \n    val = - np.matmul(np.matmul(P, x_hat - x)[[:, None], x[:, None].T)
    return np.squeeze(np.reshape(val, (4,1)))

def bhat_dynamics(x, t, x_hat, u, P):
    return np.multiply(-np.matmul(P, x_hat - x), u(t))

def xhat_dynamics(x_hat, t, u, x, ahat, bhat, Am):
    ahat_x = np.matmul(ahat, x)
    bhat_u = np.squeeze(bhat*u(t))
    return np.matmul(Am, x_hat - x) + ahat_x + bhat_u

def x_dynamics(x, t, A, B, u):
    return np.matmul(A, x) + np.squeeze(B*u(t))
```

I snuck in some numpy tricks here. The `squeeze` function eliminates singleton dimensions. That is, if your array has size \( 3 \times 1 \times 3 \times 4 \), then after squeezing it will be of size \( 3 \times 3 \times 4 \). The other item is the use of `None` in indexing. This is the opposite of squeezing. It adds back a dimension. For example:

```
In [3]: print(np.random.rand(10).shape)
print(np.random.rand(10)[[:, None].shape)
```
The rest of the functions are standard. Now we define the two different excitation signals we will use and write a function that will be passed into odeint to integrate all the state equations.

In [10]: def u1(t):
    return 10.0 * sin(2.0*t)

def u2(t):
    return 10.0 * sin(2.0*t) + 7.0 * cos(3.6*t)

def all_states(y, t, u, A, B, Am, P):
    x, xhat, ahat, bhat = extract(y)
    d_x = x_dynamics(x, t, A, B, u)
    d_xhat = xhat_dynamics(xhat, t, u, x, ahat, bhat, Am)
    d_ahat = ahat_dynamics(x, xhat, P)
    d_bhat = bhat_dynamics(x, t, xhat, u, P)
    d_y = np.concatenate((d_x, d_xhat, d_ahat, d_bhat))
    return d_y

The function np.concatenate does what you would expect it to. But another function sneaked in: extract. That is a function we will write as follows to facilitate dealing with all the different states. We need it because odeint only works for vector differential equation. But for our convenience we wrote the dynamics on paper as matrix equations and that is what we implemented above.

In [5]: def extract(y):
    x, xhat = y[:2], y[2:4]
    ahat, bhat = y[4:8], y[8:10]
    ahat = np.apply_along_axis(np.reshape, 0, arr=ahat, newshape=(2, 2))
    bhat = np.apply_along_axis(np.reshape, 0, arr=bhat, newshape=(2, 1))
    return x, xhat, ahat, bhat

The function np.apply_along_axis does what it name says. It takes a function that works on 1-D arrays and applies it to 1-D slices along the axis you specify. Why do we need it? Because we want the same function extract() to work in all_states() as well as in the plotting routine written below. That is the last function we write.

In [6]: def make_plot(solution, u):
    x, xhat, ahat, bhat = extract(solution)
    plt.figure(figsize=(12,4))
    rcParams['axes.grid'] = True
    ax1, ax2, ax3 = plt.subplot(1,3,1), plt.subplot(1,3,2), plt.subplot(1,3,3)
    ax1.plot(t,(x-xhat).T)
    ax1.set_title('x - \hat{x}')
    ax2.plot(t,np.reshape(A[...,None] - ahat, newshape=(4, 50)).T)
    ax2.set_title('A - \hat{A}')
    ax3.plot(t,(B-np.squeeze(bhat)).T)
ax3.set_title('B - $\hat{B}$')
plt.suptitle(u)
plt.show()

Couple of things going on here:

1. To save vertical space I stacked multiple calls to sub_plot into a single line. But because of that I had to use ax.set_title() instead of plt.title().
2. We are plotting the difference between the real parameters and our estimates.

This means $\hat{b}$ (once the system is solved) is going to be a tensor of shape $2 \times 1 \times 50$. The 50 comes from np.linspace(0,100) below because it returns 50 points by default. Similarly $\hat{a}$ will be a tensor of shape $2 \times 2 \times 50$. We squeeze out the singleton dimension from $\hat{b}$ but promote $A$ to size $2 \times 2 \times 1$ so that it can be subtracted from $\hat{a}$.

Now we set up the system parameters. Note that we choose $A_m = -5 I$ and solve the Lyapunov equation

$$A_m^T P + PA_m = -I$$

In [11]: A = np.asarray([[-0.25, 3], [-5,0]])
B = np.asarray([[1],[2.2]])
Q = -np.eye(2)
Am = -5 * np.eye(2)
P = solve_lyapunov(Am, Q)
init_vals = np.random.randint(1,10,10)
t = np.linspace(0,100)

Finally we integrate and plot the solutions.

1 Part a

In [12]: sols = odeint(all_states, init_vals, t, args=(u1, A, B, Am, P))
make_plot(sols.T, '$u = 10 \sin 2t$')
2 Part b

In [13]: sols = odeint(all_states, init_vals, t, args=(u2, A, B, Am, P))
make_plot(sols.T, '$u = 10 \sin 2t + 7 \cos 3.6t$')