1. Consider the 2-D system

\[
\dot{x}_1 = -\frac{6x_1}{(1 + x_1^2)^2} + 2x_2,
\]
\[
\dot{x}_2 = -\frac{2(x_1 + x_2)}{(1 + x_1^2)^2}
\]

and the candidate Lyapunov function

\[
V(x_1, x_2) = \frac{x_1^2}{1 + x_1^2} + x_2^2.
\]

Compute the derivative of this \(V\) along solutions. Can you conclude that all solutions \(x(t)\) are bounded? that all solutions \(x(t)\) with initial conditions \(x(0)\) sufficiently close to 0 are bounded? that all \(x(t)\) converge to 0? that all \(x(t)\) with \(x(0)\) sufficiently close to 0 converge to 0?

For each question, explain which result you’re using or give a reason why you can not.

**Solution.** The derivative of this \(V\) along solutions is

\[
\dot{V} = \frac{\partial V}{\partial x_1} \dot{x}_1 + \frac{\partial V}{\partial x_2} \dot{x}_2
\]

\[
= \frac{2x_1}{(1 + x_1^2)^2} \left( -\frac{6x_1}{(1 + x_1^2)^2} + 2x_2 \right) + 2x_2 \left( -\frac{2(x_1 + x_2)}{(1 + x_1^2)^2} \right)
\]

\[
= -\frac{12x_1^2}{(1 + x_1^2)^4} - \frac{4x_2^2}{(1 + x_1^2)^2},
\]

which is negative definite. However, since

\[
\lim_{x_1 \to \infty} V(x_1, x_2) = 1 + x_2^2 < \infty,
\]

the candidate Lyapunov function is not radially unbounded. Hence

1) we cannot conclude that all solutions \(x(t)\) are bounded;
2) we can conclude that all solutions \(x(t)\) with initial condition \(x(0)\) sufficiently close to 0 are bounded;
3) we cannot conclude that all \(x(t)\) converge to 0; and
4) we can conclude that all solutions \(x(t)\) with initial condition \(x(0)\) sufficiently close to 0 converge to 0.
2. When proving Barbalat’s lemma in class, we assumed for simplicity that $W(x) \geq 0$ (which is one of the hypotheses in Theorem 2). However, Barbalat’s lemma itself is valid even if $W$ is not sign-definite. Refine the proof of Barbalat’s lemma from class so that it works for $W$ possibly taking both positive and negative values.

**Solution.** We prove by contradiction. Suppose $W(x(t))$ doesn’t converge to 0. Then there exist an $\epsilon > 0$ and an increasing sequence $(t_k)_{k \in \mathbb{N}}$ such that $t_k \to \infty$ as $k \to \infty$ and

$$|W(x(t_k))| \geq \epsilon \quad \forall k \in \mathbb{N}.$$ 

As $W$ is a continuous function of $x$ and $x(t)$ is bounded, there exists a $\delta_x > 0$ such that

$$|x(t) - x(t_k)| \leq \delta_x \quad \Rightarrow \quad |W(x(t)) - W(x(t_k))| \leq \epsilon/2$$

for all $k \in \mathbb{N}$. Furthermore, as $\dot{x}(t)$ is also bounded, there exists a $\delta_t > 0$ such that

$$|t - t_k| \leq \delta_t \quad \Rightarrow \quad |x(t) - x(t_k)| \leq \delta_x$$

for all $k \in \mathbb{N}$. Combining the arguments above with the triangle inequality shows that for each $k \in \mathbb{N}$, either $W(x(t)) \geq \epsilon/2$ for all $t \in [t_k, t_k + \delta_t]$, or $W(x(t)) \leq -\epsilon/2$ for all $t \in [t_k, t_k + \delta_t]$. Therefore, the fact that $t_k \nrightarrow \infty$ as $k \to \infty$ implies that for each $T > 0$ there exists a $t_k > T$ such that

$$\left| \int_{t_k}^{t_k + \delta_t} W(x(s))\,ds - \int_{t_k}^{t_k} W(x(s))\,ds \right| = \left| \int_{t_k}^{t_k + \delta_t} W(x(s))\,ds \right| \geq \epsilon \delta_t/2. \quad (1)$$

On the other hand, from Cauchy’s convergence test (cf. Cauchy Sequence [Khalil, 2002, p. 654]) it follows that if

$$\int_0^\infty W(x(s))\,ds = \lim_{t \to \infty} \int_0^t W(x(s))\,ds$$

is well-defined and finite then for each $\delta_t > 0$ there exists a $T > 0$ such that

$$\left| \int_{0}^{t_k + \delta_t} W(x(s))\,ds - \int_{0}^{t_k} W(x(s))\,ds \right| < \epsilon \delta_t/2 \quad \forall t_k > T,$$

which is a contradiction. Hence $W(x(t))$ converges to 0.

**Notes:**

(A) You cannot take the sum of (1) for all $k$ and conclude that $\int_0^\infty W(x(s))\,ds$ is infinite, since for different $k$ the signs of $W(x(t))$ on $[t_k, t_k + \delta_t]$ may be different.

(B) It is not true that if $\int_0^\infty W(x(s))\,ds$ converges so does $\int_0^\infty |W(x(s))|\,ds$. Consider $\int_0^\infty \frac{\sin x}{x}\,dx$ for example.
3. Consider the system

\[ \dot{x} = \theta + x + u \]  

(2)

which is similar to Example 1 from the class notes except the unknown parameter \( \theta \) enters additively and not multiplicatively. Suppose we want to make \( x \) converge to 0. Propose an adaptive control law that achieves this, and justify that it works. Base your design and analysis on ideas similar to the ones used to treat Example 1 in class: introduce an estimate \( \hat{\theta} \) and a differential equation for it (tuning law); make the control law depend on \( \hat{\theta} \); analyze the closed-loop system with the help of a Lyapunov function.

When discussing Example 1 in class, we commented that the estimate \( \hat{\theta} \) does not necessarily converge to the true value \( \theta \) (see also Problem 4 below). For the closed-loop system that you obtained from the system (2) with your controller, can you prove that \( \hat{\theta} \) does in fact converge to \( \theta \)?

Solution. Consider the candidate Lyapunov function

\[ V(x, \hat{\theta}) = \frac{x^2}{2} + \frac{(\hat{\theta} - \theta)^2}{2}. \]

It’s derivative along the solution of (2) is given by

\[ \dot{V} = x\dot{x} + (\hat{\theta} - \theta)\dot{\hat{\theta}} = (x - \dot{\theta})\theta + x^2 + xu + \dot{\hat{\theta}}. \]

To eliminate the unknown \( \theta \), we set \( \dot{\hat{\theta}} = x \), and obtain

\[ \dot{V} = (u + x + \hat{\theta})x. \]

Hence we set \( u = -kx - \hat{\theta} \) with a \( k > 1 \), which gives

\[ \dot{V} = -(k - 1)x^2 < 0 \quad \forall x \neq 0. \]

The closed-loop system becomes

\[ \begin{aligned}
\dot{x} &= \theta - \hat{\theta} + (1 - k)x, \\
\dot{\hat{\theta}} &= x.
\end{aligned} \]  

(3)

It follows from Theorem 2 in the lecture notes that \( W(x(t)) = (k - 1)x(t)^2 \to 0 \) as \( t \to \infty \), that is, the state \( x \) converges to 0.

We use LaSalle’s theorem (cf. [Khalil, 2002, Sec. 4.2]) to show that \( \hat{\theta} \) does in fact converge to \( \theta \). Define \( E := \{(x, \hat{\theta})|\dot{V} = 0\} = \{(x, \hat{\theta})|x = 0\} \). As

\[ x \equiv 0 \Rightarrow \dot{x} \equiv 0 \Rightarrow \dot{\hat{\theta}} = \theta, \]

the largest invariant \( M \subset E \) is \( M = \{(0, \theta)\} \). From LaSalle’s theorem it follows that all solutions \((x, \hat{\theta})\) of (3) converge to \( M \), that is, the estimate \( \hat{\theta} \) converges to \( \theta \).

Alternatively, if we let \( y := \hat{\theta} - \theta \) then (3) becomes

\[ \begin{bmatrix}
\dot{x} \\
\dot{y}
\end{bmatrix} = \begin{bmatrix}
1 - k & -1 \\
1 & 0
\end{bmatrix} \begin{bmatrix}
x \\
y
\end{bmatrix}, \]

in which the matrix is Hurwitz as its eigenvalues

\[ \lambda_{1,2} = \frac{1 - k \pm \sqrt{k^2 - 2k - 3}}{2} \]

have negative real parts (for \( k > 1 \)).
4. a) Consider the adaptive control system introduced in class to treat Example 1:

\[
\begin{align*}
\dot{x} &= \theta x + u, \\
\hat{\theta} &= x^2, \\
u &= -(\hat{\theta} + 1)x
\end{align*}
\]

where \(\theta\) is an unknown real parameter. Using MATLAB (or any other software), simulate this system for some arbitrary choices of \(\theta\). Does the state \(x\) converge to 0? Does \(\hat{\theta}\) remain bounded? Does it converge to \(\theta\)? Can you analytically derive the limiting value of \(\hat{\theta}\)?

b) Keep the same tuning/control law (5)–(6) but modify the plant dynamics (4) to

\[
\dot{x} = \theta x + u + d
\]

where \(d\) is a constant nonzero disturbance. Repeat the simulations and answer the same questions as in part a).

c) Keep the disturbance, and modify the tuning law (5) by turning the adaptation off (setting \(\dot{\hat{\theta}} = 0\)) whenever \(x\) is in some small interval around 0. This modification is known as dead zone. Repeat the simulations and answer the same questions as in part a); play with the size of the dead-zone interval and explain how it should be chosen.

d) Keep the disturbance, and modify the tuning law (5) by selecting some value \(\hat{\theta}_{\text{max}}\) and turning the adaptation off (setting \(\dot{\hat{\theta}} = 0\)) if \(\hat{\theta}\) reaches \(\hat{\theta}_{\text{max}}\). This modification is known as projection. Repeat the simulations and answer the same questions as in part a); play with the value of \(\hat{\theta}_{\text{max}}\) and explain how it should be chosen.

Solution. a) The Simulink diagram and the simulation result can be found in Fig. 1. From the simulation result, we see that the state \(x\) converges to 0, and the estimate \(\hat{\theta}\) remains bounded but does not converge to \(\theta\). To derive the limiting value of \(\hat{\theta}\) note that the closed loop can be written as:

\[
x\dot{x} = \left(\theta - \hat{\theta} - 1\right)\dot{\hat{\theta}}
\]

This can be integrated as:

\[
\frac{x^2}{2} - \frac{x_0^2}{2} = \int_{\hat{\theta}_0}^{\hat{\theta}_t} (\theta - z - 1) \, dz = \left[(\theta - 1) z - \frac{z^2}{2}\right]_{\hat{\theta}_0}^{\hat{\theta}_t}
\]

where (1) change of parameter \(z = \hat{\theta}\) was introduced and (2) the argument \(t\) has been reduced to subscripts to avoid clutter. Let \(\hat{\theta}_\infty\) denote the limiting value of \(\hat{\theta}\). As \(x \to 0\) with time we can find the limiting value as the positive solution of the quadratic equation:

\[
-x_0^2 = 2 (\theta - 1) \left(\hat{\theta}_\infty - \hat{\theta}_0\right) + \left(\hat{\theta}_0^2 - \hat{\theta}_\infty^2\right)
\]
b) The Simulink diagram and the simulation result can be found in Fig. 2. From the simulation result, we see that the state $x$ does not converge to 0 (exponentially), and $\hat{\theta}$ goes unbounded and does not converge to $\theta$. The closed-loop system is

$$\dot{x} = (\theta - \hat{\theta} - 1)x + d,$$

$$\dot{\hat{\theta}} = x^2.$$

From the closed-loop dynamics we are able to conclude that $\hat{\theta}$ cannot be bounded. Indeed, if it is bounded then $\dot{x} > 0$ for all $|x| < d/|\hat{\theta} + 1 - \theta|$, meaning that the state $x$ is bounded away from 0. But then from the dynamics of $\hat{\theta}$ we see that $\theta \to \infty$ as $t \to \infty$, which is a contradiction. Hence $\hat{\theta}$ goes unbounded and does not converge to $\theta$. However, we cannot definitively conclude that $x$ converges to 0 from the simulation.

![Simulink diagram](image1)

![Simulation result](image2)

**Fig. 2: Problem 4.b)**

c) The Simulink diagram and the simulation result can be found in Fig. 3. From the simulation result, we see that the state $x$ does not converge to 0, and $\hat{\theta}$ remains bounded but does not converge to $\theta$. In this case, instead of $\dot{\hat{\theta}} = x^2$, we use

$$\dot{\hat{\theta}} = \begin{cases} 
  x^2, & \text{if } |x| > \delta, \\
  0, & \text{if } |x| \leq \delta
\end{cases}$$

for some $\delta > 0$. Thus, knowledge of the disturbance $d$ would help to choose the size of dead-zone in this case. Switching the adaptation off after $|x| \leq \delta$ then guarantees that $\hat{\theta}$ remains bounded but doesn’t necessarily converge to $\theta$.

![Simulink diagram](image3)

![Simulation result](image4)

**Fig. 3: Problem 4.c)**
d) The Simulink diagram and the simulation result can be found in Fig. 4. From the simulation result, we see that the state $x$ does not converge to 0, and $\hat{\theta}$ remains bounded and does not converge to $\theta$. In this case, as $\dot{x} = (\theta - \hat{\theta} - 1)x + d$, the value of $\hat{\theta}_{\text{max}}$ should be chosen so that $\theta - \hat{\theta}_{\text{max}} - 1 < 0$. Thus, knowledge on the unknown parameter $\theta$ would help to choose the value of $\hat{\theta}_{\text{max}}$ in this case.

 Except for part (a) is not required to derive any closed-form formulae, but the answer and explanation needs to be consistent with the plots.
1 Introduction

MATLAB solutions are accepted and posted; but since MATLAB licenses are no longer provided to students and toolbox licenses are available in limited quantities, this additional part is provided in a tutorial format to show how the question can be attempted with open source software, namely the Python programming language. The author could also have done this on Scilab\(^1\) or Octave\(^2\) which are probably closest in spirit to MATLAB but . . .

1. Python recently\(^3\) became the third most popular language.
2. Python is widely being used in data science, machine learning, etc and is an extremely capable general purpose programming language and so . . .
3. . . . a lot of you are likely to be already familiar with Python and finally . . .
4. He wanted to have some fun with Python.

If you are not familiar with Python, IPython and Jupyter see footnote\(^4\). If you have programming experience in C/C++/Java etc. you can probably pick up Python over a weekend (to your own great benefit).

This PDF was generated using a Jupyter notebook (also posted). If you have familiarized yourself with Python a little bit, you will know that the Python paradigm is to create environments\(^5\) for each project you start. For getting started with Python’s environment/package business using Anaconda\(^6\) is recommended.

To get this notebook to run, I ran the following commands at the terminal (i.e. on Linux, MacOS . . . apologies Windows) after installing Anaconda to set-up my environment.

```bash
conda create --name tastuff python=3.7
source activate tastuff
conda install scipy matplotlib jupyter nb_conda
```

1.1 Problem 4

Here on, the author assumes familiarity with Python and/or Jupyter notebooks and we use Python Version ≥ 3.6. You can also easily copy paste and minimally modify the code to make it run as a script (*.py) file, but notebooks are handy. First make the necessary imports.

\(^1\)https://www.scilab.org/en/scilab/features/scilab/control
\(^2\)https://www.gnu.org/software/octave/
\(^3\)https://www.zdnet.com/article/python-now-a-top-3-programming-language-as-julias-rise-speeds-up/
\(^4\)https://plot.ly/python/ipython-vs-python/
\(^5\)https://realpython.com/python-virtual-environments-a-primer/
\(^6\)https://www.anaconda.com/download/
1.2 Part (a)

Next define the plant for the first part of the question. We subsume \((x, \dot{\theta})\) into a single augmented state vector \(y\). In the function definition \(y, t, \theta, \text{control and tuning are arguments to be provided.}

```
In [2]: def plant_a(y, t, theta, control, tuning):
    '''y is state. t is time.
    theta, control and tuning laws are to be supplied'''
    x, hat_theta = y # y is a list/tuple, we unpack it
    dx = theta * x + control(x, hat_theta)
    dhat_theta = tuning(x, hat_theta)
    return [dx, dhat_theta]
```

Now define the tuning law and control law. If we read ahead into parts (b), (c) and (d); we see that we will modify the plant once and the tuning law many times. So when we write the tuning law function we incorporate those additional laws as well. Otherwise we have to change the plant function (specifically the call to the tuning law inside the plant function) each time we change the law. Depending on whether arguments \(dzone\) or \(thetamax\) are provided to \(tuning\_law()\), the function determines which law to use. These are called optional/keyword arguments since we provide them with default values ... contrasted with positional/required arguments \(x\) and \(hat\_theta\).

```
In [3]: def control_law(x, hat_theta):
    return -(hat_theta + 1)*x

def tuning_law(x, hat_theta, dzone=None, thetamax=None):
    '''In anticipation of Part (c)-(d), as long as argument
    dzone and thetamax are not provided it always defaults to None'''
    if dzone is None and thetamax is None:
        return x**2
    elif dzone:
        if np.abs(x) > dzone:
            return x**2
        else:
            return 0
    elif thetamax:
        if hat_theta <= thetamax:
            return x**2
        else:
            return 0
```

return x**2
else:
  return 0
else:  # Can't have both dzone and thetamax
  print('Error!')

In the above None is a falsy (see Truth Value Testing\cite{7} in Python) object of type NoneType. It is frequently used to indicate the absence of values. Since it is an object and objects can not be usually compared, the special comparison operator\cite{8} is must be used. Now we define the initial conditions and solve for the state of the system using odeint()\cite{9} function.

In [4]: t = np.arange(0,5,0.05)

    # To replicate solution plots use values below, change them to play around
    init_state = [5.0, 0.0]
    theta = 3.7843

    state = odeint(plant_a, init_state, t, args=(theta, control_law, tuning_law))

Finally we make the plots, and since we will make the plots many times, we turn this into a function too. The package matplotlib.pypolt\cite{10} provides pretty much MATLAB like functionality in plot generation.

In [5]: def make_plots(state, title):
   
   rcParams["font.size"] = "14"
   plt.figure(figsize=(12,6))
   plt.subplot(1,2,1)
   plt.title('State value $x$')
   plt.plot(t,state[:,0])
   plt.subplot(1,2,2)
   plt.plot(t,state[:,1], label='$\hat{\theta}$')
   plt.plot(t,len(t)*[theta], label='$\theta$')
   plt.title('The estimation $\hat{\theta}$')
   plt.subplots_adjust(top=0.75)
   plt.legend()
   plt.suptitle(title)
   plt.show()

   # Note that \ is required to escape \ which is usual escape
   title = 'Solution plots for Problem 4 - Part (a) \n $\theta =$ ' + str(theta)
   make_plots(state, title)

\cite{7}https://docs.python.org/3.8/library/stdtypes.html
\cite{8}https://docs.python.org/3.7/library/stdtypes.html#comparisons
\cite{9}https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html
\cite{10}https://matplotlib.org/api/pyplot_api.html
1.3 Part (b)

Now the plant dynamics have changed so we define a new plant function for parts (b), (c), (d)

In [6]: def plant_bcd(y, t, theta, control, tuning, disturbance):
   x, hat_theta = y
   dx = theta * x + control(x, hat_theta) + disturbance
   dhat_theta = tuning(x, hat_theta)
   return [dx, dhat_theta]

Now we solve for it as before and make the plots. In the below a lone backslash `\` is a line continuation character, just like MATLABs .... Since Python plays by the offside rule\(^\text{11}\) note that you cannot have even an empty space after `\`.

In [7]: # To replicate solution plots use
   t = np.arange(0,5,0.05)
   init_state = [-6.3, 0.0]
   theta, disturbance = 3.3306, 3.9621
   state = odeint(plant_bcd, init_state, t,
                  args=(theta, control_law, tuning_law, disturbance))

   title = 'Solution plots for Problem 4 - Part (b) \n $\theta = $'
\(^\text{11}\)https://en.wikipedia.org/wiki/Off-side_rule
1.4 Part (c)

Now we use the same plant_bcd for the parts (c) and (d). We modify the tuning law in these two parts. The first one implements a deadzone; we activate this deadzone by providing a value for dzone to the tuning_law function (see its definition above), which means it will implement the second part of its body. But the function tuning_law accepts two required arguments and two optional arguments. How can we specify that we know what the optional arguments should be before we know the required ones are? This is where we use partial\textsuperscript{12} functions. In Python everything is an object, including functions. For example if you define

\begin{verbatim}
In [8]: def foo(x=5):
    print(int(x)+2)

foo
\end{verbatim}

and type \texttt{foo} at the REPL/interpreter (as above) it will return a reference to the function. If you type \texttt{foo(4)} it will evaluate the function and print out 6. You can loosely think of partial functions as an abstraction that stores away \texttt{function_reference + list_of_arguments} until its evaluation time.

\textsuperscript{12}\url{https://www.pydanny.com/python-partial-are-fun.html}
In [9]: $t = \text{np.arange}(0,5,0.01)$

```python
# To replicate solution plots use
init_state = [-3.0, 0.0]
theta, disturbance, deadzone = 8.5853, -5.1295, 1

# Key difference; we make it a partial function
new_tuning_law = partial(tuning_law, dzone=deadzone)

state = odeint(plant_bcd, init_state, t,
               args=(theta, control_law, new_tuning_law, disturbance))

title = 'Solution plots for Problem 4 - Part (c) 
$\theta =$ ' + str(theta) + ' $d=$' + str(disturbance) + ' deadzone: $|x| < 1$

make_plots(state, title)
```

**Solution plots for Problem 4 - Part (c)**

$\theta = 8.5853$  $d = -5.1295$  deadzone: $|x| < 1$

1.5 Part (d)

Finally we implement the projection part of the tuning law by providing an argument for \( \text{thetamax} \).

In [10]: $t = \text{np.arange}(0,5,0.01)$

```python
# To replicate solution plots use
init_state = [4, 0.0]
```
theta, disturbance, theta_max = 0.11914, -4.8981, 4
new_tuning_law = partial(tuning_law, thetamax=theta_max)

state = odeint(plant_bcd, init_state, t,
              args=(theta, control_law, new_tuning_law, disturbance))

title = 'Solution plots for Problem 4 - Part (d) 
$\theta =$ ' \
+ str(theta) + ' $d=$ ' + str(disturbance) \
+ ' $\theta_{\text{max}} = 4$'

make_plots(state, title)

Solution plots for Problem 4 - Part (d)
$\theta = 0.11914$ $d = -4.8981$ $\theta_{\text{max}} = 4$