Code/Plots

The problems were integrated using `solve_ivp` from the `scipy` library in Python.

12.2.2.1)

```
#!/bin/python

##==============================================================================
# Libraries
import matplotlib.pyplot as plt
import numpy as np

from scipy.integrate import solve_ivp

from plot import plot

##==============================================================================
# Functions

##------------------------------------------------------------------------------
#
def model(t,x):
    """
    input
        t: time step
        x: state
    output
        dx: differential output of ode

    """
    # Extract states
    x1, x2, u = x

    # Calculate z
    z1 = x1
    z2 = x1 + x2
    z  = np.array([[z1], [z2]])

    # Calculate control
    k = np.array([10, 10])
    u = -3*(x1**3)*x2 + 2*x1 + x2 - k@z

    # Calculate ODEs
    xd1 = x1 + x2
    xd2 = 3*(x1**2)*x2 + x1 + u

    # Bundle ODEs
    dx = [xd1, xd2, u]

    return dx

##==============================================================================
# Script

# Plotting/Calculating variables
```

```python

## Time horizon
t0 = 0.0
tf = 15.0
t = [t0, tf]

## Initial conditions
x0 = [1, 0.1, 0]


# Solve ode model
sol = solve_ivp(model, t, x0, method='RK45', dense_output=True)

# Plot solution
plot(sol, tf=tf, title="12.2.2.1", legend=['x1', 'x2', 'u'])
```

12.2.2.2)

```python
#!/bin/python

##==============================================================================
# Libraries
import matplotlib.pyplot as plt
import numpy as np

from scipy.integrate import solve_ivp

from plot import plot

##==============================================================================
# Functions

##------------------------------------------------------------------------------
#
def model(t,x):
    """
    input
        t: time step
        x: state
    output
        dx: differential output of ode

    """
    # Extract state
    x1, x2, x3, u = x
    x            = np.array([x1,x2,x3])

    # Calculate control
    k = np.array([10, 10, 5])
    u = -k@x[0:4]

    # Calculate ODEs
    xd1 = x1 + x2
    xd2 = x1*x2**2 - x1 + x3
    xd3 = u

    # Bundle ODEs
    dx = [xd1, xd2, xd3, u]

    return dx

##==============================================================================
# Script

# Plotting/Calculating variables

## Time horizon
t0 = 0.0
tf = 15.0
t = [t0, tf]

## Initial conditions
```

```
x0 = [0.5, 0.1, 0.1, 0]


# Solve ode model
sol = solve_ivp(model, t, x0, method='RK45', dense_output=True)

# Plot solution
plot(sol, tf=tf, title="12.2.2.2", legend=['x1', 'x2', 'u'])
```

14.31)

```python
#!/bin/python

##==============================================================================
# Libraries
import matplotlib.pyplot as plt
import numpy as np

from scipy.integrate import solve_ivp

from plot import plot

##==============================================================================
# Functions

##------------------------------------------------------------------------------
#
def model(t,x):
    """
    input
        t: time step
        x: state
    output
        dx: differential output of ode

    """
    # Extract states
    x1, x2, u = x
    a     = 0

    # Calculate z
    z = x1 + x2 + a + (x1 - a**1/3)**3 + x1

    # Calculate control
    u = -x1 - x2 - 3*(x1**2)*x2 - z

    # Calculate ODEs

    xd1 = x2 + a + (x1 - a**1/3)**3
    xd2 = x1 + u

    # Bundle ODEs
    dx = [xd1, xd2, u]

    return dx

##==============================================================================
# Script

# Plotting/Calculating variables

## Time horizon
t0 = 0.0
tf = 8.0
t = [t0, tf]
```

```python

## Initial conditions
x0 = [0.5, 0.6, 0]


# Solve ode model
sol = solve_ivp(model, t, x0, method='RK45', dense_output=True)

# Plot solution
plot(sol, tf=tf, title="14.31", legend=['x1', 'x2', 'u'])
```

14.34)

```
##==============================================================================
# Libraries
import matplotlib.pyplot as plt
import numpy as np

from scipy.integrate import solve_ivp

from plot import plot

##==============================================================================
# Functions

##------------------------------------------------------------------------------
#
def model(t,x):
    """
    input
        t: time step
        x: state
    output
        dx: differential output of ode

    """
    # Extract states
    x1, x2, x3, u = x

    # Calculate z
    z = x2

    # Calculate control
    u = -x1 + x2 + x1*x3 - z

    # Calculate ODEs
    xd1 = -x1 + x2
    xd2 = x1 - x2 - x1*x3 + u
    xd3 = x1 + x1*x2 - 2*x3

    # Bundle ODEs
    dx = [xd1, xd2, xd3, u]

    return dx

##==============================================================================
# Script

# Plotting/Calculating variables

## Time horizon
t0 = 0.0
tf = 8.0
t = [t0, tf]

## Initial conditions
x0 = [0.1, 0.2, 0.3, 0]
```

```python


# Solve ode model
sol = solve_ivp(model, t, x0, method='RK45', dense_output=True)

# Plot solution
plot(sol, tf=tf, title="14.34", legend=['x1', 'x2', 'x3', 'u'])
```

14.15 a)

```python
#!/bin/python

##==============================================================================
# Libraries
import matplotlib.pyplot as plt
import numpy as np

from scipy.integrate import solve_ivp

from plot import plot

##==============================================================================
# Functions

##------------------------------------------------------------------------------
#
def model(t,x):
    """
    input
        t: time step
        x: state
    output
        dx: differential output of ode

    """
    # Extract states
    x1, x2, u = x

    # Calculate control
    k = 0.25
    u = -k*x2

    # Calculate ODEs

    xd1 = x2
    xd2 = -x1**3 + u

    # Bundle ODEs
    dx = [xd1, xd2, u]

    return dx

##==============================================================================
# Script

# Plotting/Calculating variables

## Time horizon
t0 = 0.0
tf = 100
t = [t0, tf]

## Initial conditions
x0 = [0.5, 0.6, 0]
```

```python


# Solve ode model
sol = solve_ivp(model, t, x0, method='RK45', dense_output=True)

# Plot solution
plot(sol, tf=tf, title="14.15_a", legend=['x1', 'x2', 'u'])
```

14.15 b)

```python
#!/bin/python

##==============================================================================
# Libraries
import matplotlib.pyplot as plt
import numpy as np

from scipy.integrate import solve_ivp

from plot import plot

##==============================================================================
# Functions

##------------------------------------------------------------------------------
#
def model(t,x):
    """
    input
        t: time step
        x: state
    output
        dx: differential output of ode

    """
    # Extract states
    x1, x2, u = x

    # Calculate control
    k = 0.25
    u = -(2*k/np.pi)*np.arctan(x2)

    # Calculate ODEs

    xd1 = x2
    xd2 = -x1**3 + u

    # Bundle ODEs
    dx = [xd1, xd2, u]

    return dx

##==============================================================================
# Script

# Plotting/Calculating variables

## Time horizon
t0 = 0.0
tf = 100
t = [t0, tf]

## Initial conditions
x0 = [0.1, 0.1, 0.0]
```

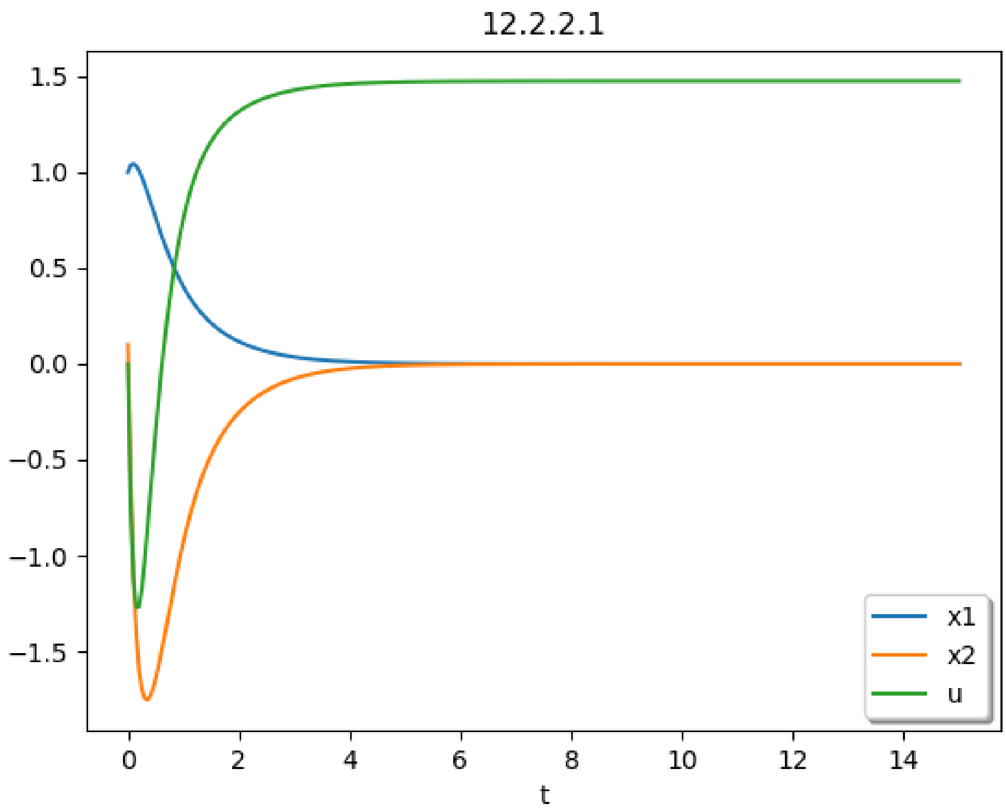
```


# Solve ode model
sol = solve_ivp(model, t, x0, method='RK45', dense_output=True)

# Plot solution
plot(sol, tf=tf, title="14.15_b", legend=['x1', 'x2', 'u'])
```

Plotter

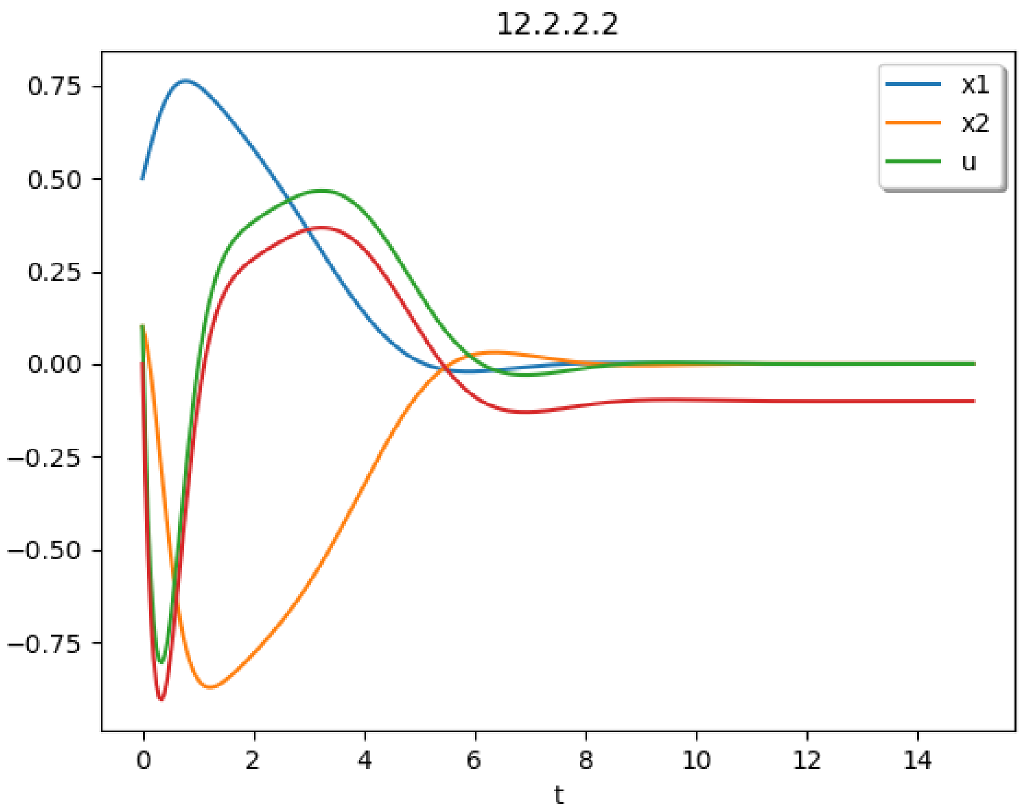
```python
#!/bin/python

##==============================================================================
# Libraries
import matplotlib.pyplot as plt
import numpy as np


##==============================================================================
#
def plot(solution,
        tf        : float   = 15,
        x_lbl     : str     = "t",
        legend    : np.array = ['x','y'],
        title     : str     = "Model",
        show_plot : bool    = False):
    """
    Plotting module for ODE's.

    input
        solution: Solution from ode

    output
        NONE
    """

    # Extract
    t = np.linspace(0,tf,300)
    z = solution.sol(t)

    # Plot
    plt.plot(t, z.T)
    plt.xlabel(x_lbl)
    plt.legend(legend, shadow=True)
    plt.title(title)

    # Determine whether to display or save plot
    if show_plot:
        plt.show()
    else:
        plt.savefig("img/"+title+".png")

    return
```

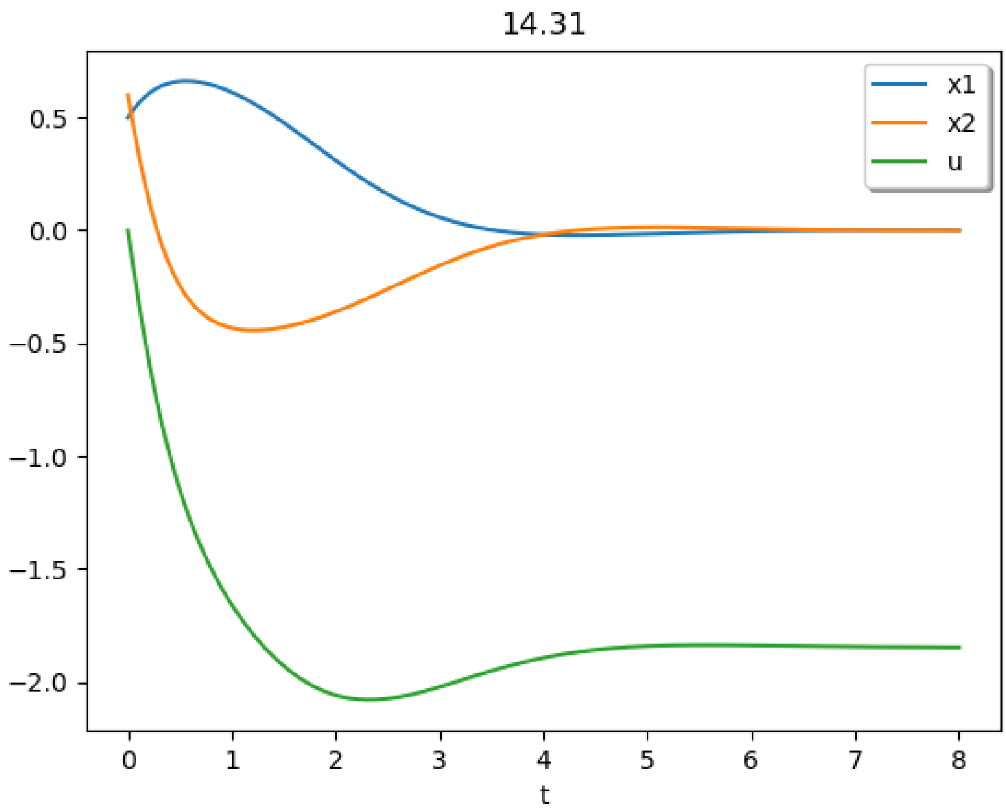
Script Runner

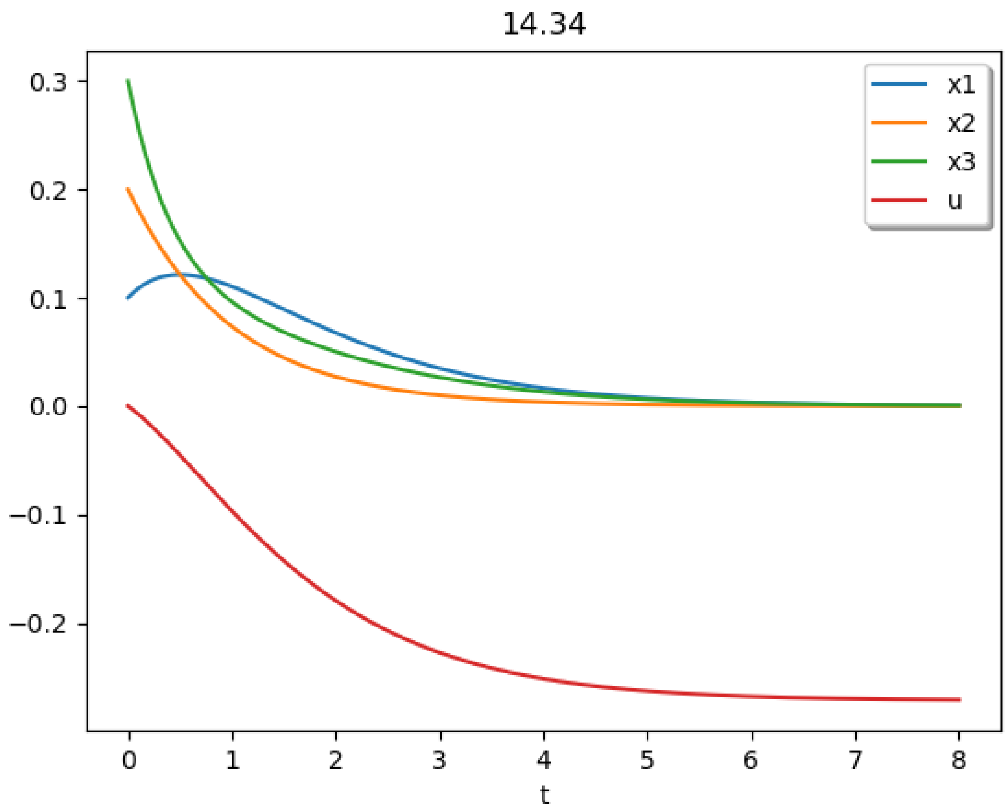
```
#!/bin/sh

# Create a fancy little progress bar
# https://stackoverflow.com/questions/238073/how-to-add-a-progress-bar-to-a-shell-script
function ProgressBar
{
    # Process data
    let _progress=(${1}*100/${2}*100)/100
    let _done=(${_progress}*4)/10
    let _left=40-$_done

    # Build progressbar string lengths
    _fill=$(printf "%${_done}s")
    _empty=$(printf "%${_left}s")

    # 1.2 Build progressbar strings and print the ProgressBar line
    # 1.2.1 Output example:
    # 1.2.1.1 Progress : [########################################] 100%
    printf "\rProcessing $s: [${_fill// /#}${_empty// /-}] ${_progress}%% \r"
}
# Variables
i=1

# Create list of scripts to be ran
declare -a scripts=(`find . -type f -name "*.py" ! -name "plot.py"`)

# Loop through each script
for s in "${scripts[@]}"
do
    ## Update progress bar
    ProgressBar $i ${#scripts[@]}

    ## Process script
    python $s

    ## Update Index
    i=$((i+1))
done

printf '\nFinished!\n'
```

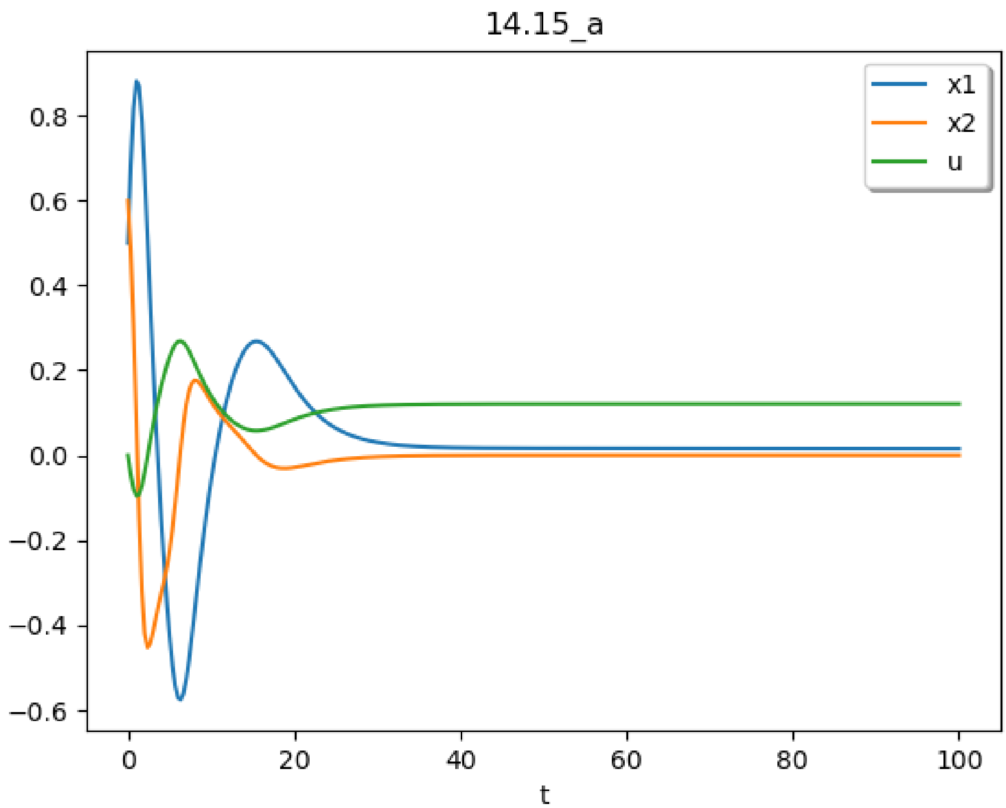


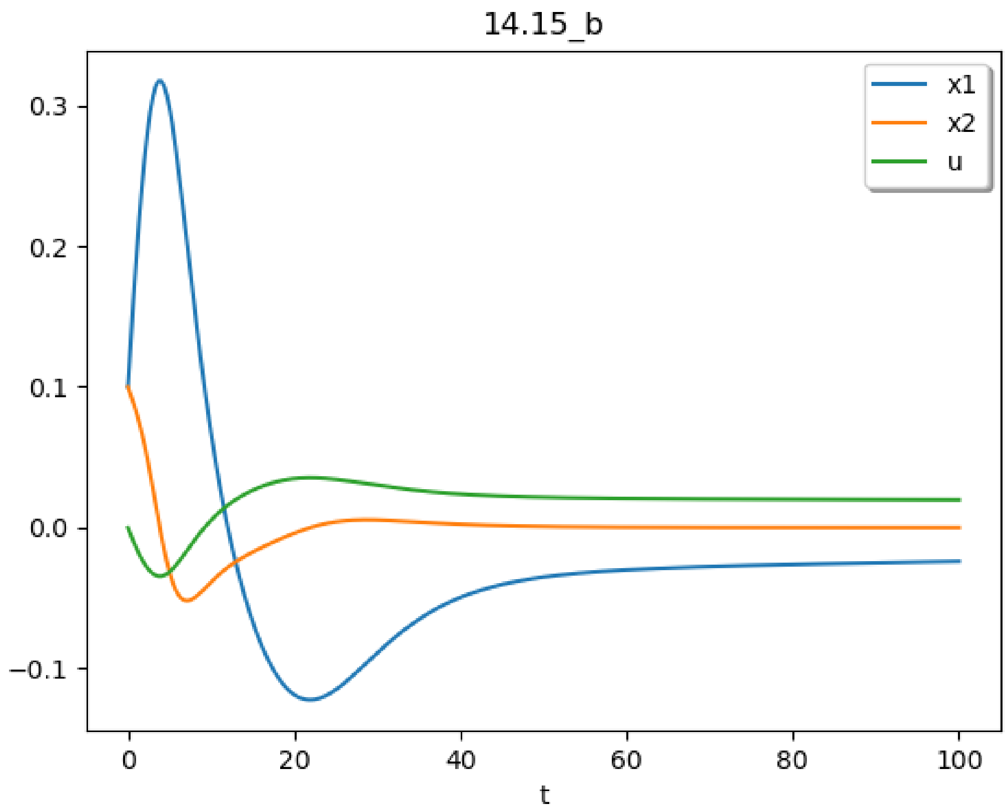
12.2.2.1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
2
4
6
8
10
12
14
t
x1
x2
u


12.2.2.2
x1
x2
u
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2
4
6
8
10
12
14
t


14.31
x1
x2
u
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
1
2
3
4
5
6
7
8
t


14.34
x1
x2
x3
u
0.3
0.2
0.1
0.0
−0.1
−0.2
0
1
2
3
4
5
6
7
8
t


14.15_a
x1
x2
u
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
20
40
60
80
100
t



A note on `14-15-a` and `14-15-b`

Comparing the two plots, it can be noted that `a` has a better state response with more control applied. `b` converges slower, but has less overshoot with and less control applied.