

Code/Plots

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

12.2.2.1)

```
1  #!/bin/python
2
3  ##=====
4  # Libraries
5  import matplotlib.pyplot as plt
6  import numpy as np
7
8  from scipy.integrate import solve_ivp
9
10 from plot import plot
11
12 ##=====
13 # Functions
14
15 ##-----
16 #
17 def model(t,x):
18     """
19     input
20         t: time step
21         x: state
22     output
23         dx: differential output of ode
24
25     """
26     # Extract states
27     x1, x2, u = x
28
29     # Calculate z
30     z1 = x1
31     z2 = x1 + x2
32     z = np.array([[z1], [z2]])
33
34     # Calculate control
35     k = np.array([10, 10])
36     u = -3*(x1**3)*x2 + 2*x1 + x2 - k@z
37
38     # Calculate ODEs
39     xd1 = x1 + x2
40     xd2 = 3*(x1**2)*x2 + x1 + u
41
42     # Bundle ODEs
43     dx = [xd1, xd2, u]
44
45     return dx
46
47 ##=====
48 # Script
49
50 # Plotting/Calculating variables
```

```
51
52 ## Time horizon
53 t0 = 0.0
54 tf = 15.0
55 t = [t0, tf]
56
57 ## Initial conditions
58 x0 = [1, 0.1, 0]
59
60
61 # Solve ode model
62 sol = solve_ivp(model, t, x0, method='RK45', dense_output=True)
63
64 # Plot solution
65 plot(sol, tf=tf, title="12.2.2.1", legend=['x1', 'x2', 'u'])
```

12.2.2.2)

```
1  #!/bin/python
2
3  ##=====
4  # Libraries
5  import matplotlib.pyplot as plt
6  import numpy as np
7
8  from scipy.integrate import solve_ivp
9
10 from plot import plot
11
12 ##=====
13 # Functions
14
15 ##-----
16 #
17 def model(t,x):
18     """
19     input
20         t: time step
21         x: state
22     output
23         dx: differential output of ode
24
25     """
26     # Extract state
27     x1, x2, x3, u = x
28     x = np.array([x1,x2,x3])
29
30     # Calculate control
31     k = np.array([10, 10, 5])
32     u = -k*x[0:4]
33
34     # Calculate ODEs
35     xd1 = x1 + x2
36     xd2 = x1*x2**2 - x1 + x3
37     xd3 = u
38
39     # Bundle ODEs
40     dx = [xd1, xd2, xd3, u]
41
42     return dx
43
44 ##=====
45 # Script
46
47 # Plotting/Calculating variables
48
49 ## Time horizon
50 t0 = 0.0
51 tf = 15.0
52 t = [t0, tf]
53
54 ## Initial conditions
```

```
55 x0 = [0.5, 0.1, 0.1, 0]
56
57
58 # Solve ode model
59 sol = solve_ivp(model, t, x0, method='RK45', dense_output=True)
60
61 # Plot solution
62 plot(sol, tf=tf, title="12.2.2.2", legend=['x1', 'x2', 'u'])
```

14.31)

```
1  #!/bin/python
2
3  ##=====
4  # Libraries
5  import matplotlib.pyplot as plt
6  import numpy as np
7
8  from scipy.integrate import solve_ivp
9
10 from plot import plot
11
12 ##=====
13 # Functions
14
15 ##-----
16 #
17 def model(t,x):
18     """
19     input
20         t: time step
21         x: state
22     output
23         dx: differential output of ode
24
25     """
26     # Extract states
27     x1, x2, u = x
28     a = 0
29
30     # Calculate z
31     z = x1 + x2 + a + (x1 - a**1/3)**3 + x1
32
33     # Calculate control
34     u = -x1 - x2 - 3*(x1**2)*x2 - z
35
36     # Calculate ODEs
37
38     xd1 = x2 + a + (x1 - a**1/3)**3
39     xd2 = x1 + u
40
41     # Bundle ODEs
42     dx = [xd1, xd2, u]
43
44     return dx
45
46 ##=====
47 # Script
48
49 # Plotting/Calculating variables
50
51 ## Time horizon
52 t0 = 0.0
53 tf = 8.0
54 t = [t0, tf]
```

```
55
56 ## Initial conditions
57 x0 = [0.5, 0.6, 0]
58
59
60 # Solve ode model
61 sol = solve_ivp(model, t, x0, method='RK45', dense_output=True)
62
63 # Plot solution
64 plot(sol, tf=tf, title="14.31", legend=['x1', 'x2', 'u'])
```

14.34)

```
1  ##=====
2  # Libraries
3  import matplotlib.pyplot as plt
4  import numpy as np
5
6  from scipy.integrate import solve_ivp
7
8  from plot import plot
9
10 ##=====
11 # Functions
12
13 ##-----
14 #
15 def model(t,x):
16     """
17     input
18         t: time step
19         x: state
20     output
21         dx: differential output of ode
22
23     """
24     # Extract states
25     x1, x2, x3, u = x
26
27     # Calculate z
28     z = x2
29
30     # Calculate control
31     u = -x1 + x2 + x1*x3 - z
32
33     # Calculate ODEs
34     xd1 = -x1 + x2
35     xd2 = x1 - x2 - x1*x3 + u
36     xd3 = x1 + x1*x2 - 2*x3
37
38     # Bundle ODEs
39     dx = [xd1, xd2, xd3, u]
40
41     return dx
42
43 ##=====
44 # Script
45
46 # Plotting/Calculating variables
47
48 ## Time horizon
49 t0 = 0.0
50 tf = 8.0
51 t = [t0, tf]
52
53 ## Initial conditions
54 x0 = [0.1, 0.2, 0.3, 0]
```

```
55
56
57 # Solve ode model
58 sol = solve_ivp(model, t, x0, method='RK45', dense_output=True)
59
60 # Plot solution
61 plot(sol, tf=tf, title="14.34", legend=['x1', 'x2', 'x3', 'u'])
```

14.15 a)

```
1  #!/bin/python
2
3  ##=====
4  # Libraries
5  import matplotlib.pyplot as plt
6  import numpy as np
7
8  from scipy.integrate import solve_ivp
9
10 from plot import plot
11
12 ##=====
13 # Functions
14
15 ##-----
16 #
17 def model(t,x):
18     """
19     input
20         t: time step
21         x: state
22     output
23         dx: differential output of ode
24
25     """
26     # Extract states
27     x1, x2, u = x
28
29     # Calculate control
30     k = 0.25
31     u = -k*x2
32
33     # Calculate ODEs
34
35     xd1 = x2
36     xd2 = -x1**3 + u
37
38     # Bundle ODEs
39     dx = [xd1, xd2, u]
40
41     return dx
42
43 ##=====
44 # Script
45
46 # Plotting/Calculating variables
47
48 ## Time horizon
49 t0 = 0.0
50 tf = 100
51 t = [t0, tf]
52
53 ## Initial conditions
54 x0 = [0.5, 0.6, 0]
```

```
55
56
57 # Solve ode model
58 sol = solve_ivp(model, t, x0, method='RK45', dense_output=True)
59
60 # Plot solution
61 plot(sol, tf=tf, title="14.15_a", legend=['x1', 'x2', 'u'])
```

14.15 b)

```
1  #!/bin/python
2
3  ##=====
4  # Libraries
5  import matplotlib.pyplot as plt
6  import numpy as np
7
8  from scipy.integrate import solve_ivp
9
10 from plot import plot
11
12 ##=====
13 # Functions
14
15 ##-----
16 #
17 def model(t,x):
18     """
19     input
20         t: time step
21         x: state
22     output
23         dx: differential output of ode
24
25     """
26     # Extract states
27     x1, x2, u = x
28
29     # Calculate control
30     k = 0.25
31     u = -(2*k/np.pi)*np.arctan(x2)
32
33     # Calculate ODEs
34
35     xd1 = x2
36     xd2 = -x1**3 + u
37
38     # Bundle ODEs
39     dx = [xd1, xd2, u]
40
41     return dx
42
43 ##=====
44 # Script
45
46 # Plotting/Calculating variables
47
48 ## Time horizon
49 t0 = 0.0
50 tf = 100
51 t = [t0, tf]
52
53 ## Initial conditions
54 x0 = [0.1, 0.1, 0.0]
```

```
55
56
57 # Solve ode model
58 sol = solve_ivp(model, t, x0, method='RK45', dense_output=True)
59
60 # Plot solution
61 plot(sol, tf=tf, title="14.15_b", legend=['x1', 'x2', 'u'])
```

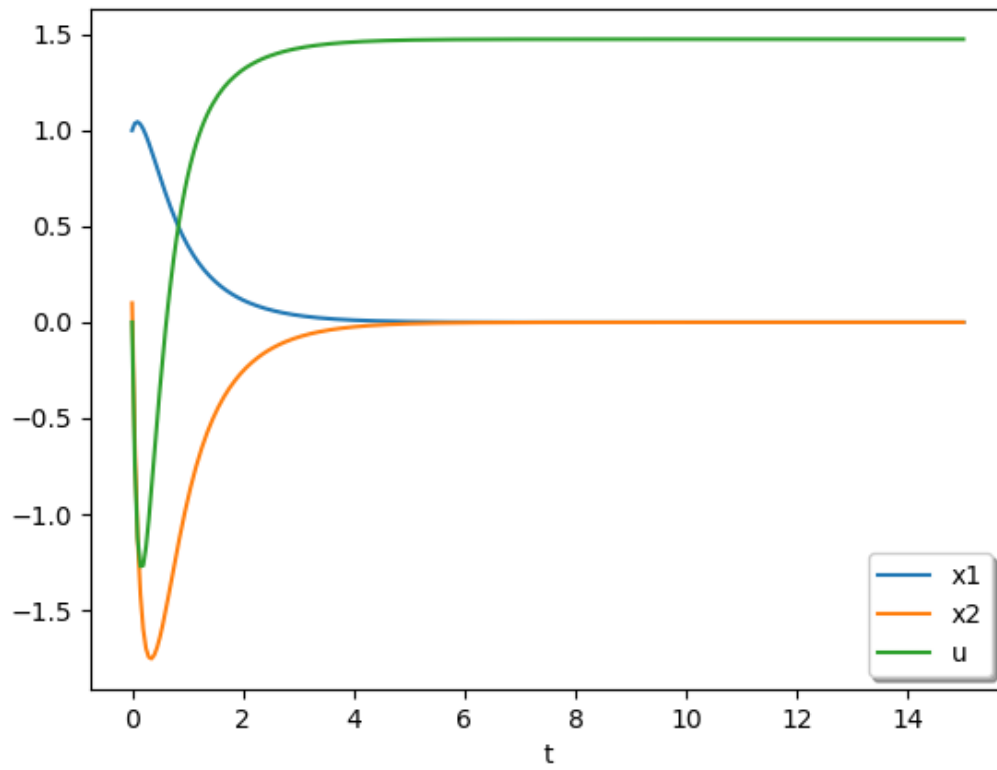
Plotter

```
1  #!/bin/python
2
3  ##=====
4  # Libraries
5  import matplotlib.pyplot as plt
6  import numpy as np
7
8
9  ##=====
10 #
11 def plot(solution,
12         tf      : float  = 15,
13         x_lbl    : str    = "t",
14         legend   : np.array = ['x','y'],
15         title    : str    = "Model",
16         show_plot : bool   = False):
17     """
18     Plotting module for ODE's.
19
20     input
21         solution: Solution from ode
22
23     output
24         NONE
25     """
26
27     # Extract
28     t = np.linspace(0,tf,300)
29     z = solution.sol(t)
30
31     # Plot
32     plt.plot(t, z.T)
33     plt.xlabel(x_lbl)
34     plt.legend(legend, shadow=True)
35     plt.title(title)
36
37     # Determine whether to display or save plot
38     if show_plot:
39         plt.show()
40     else:
41         plt.savefig("img/"+title+".png")
42
43     return
```

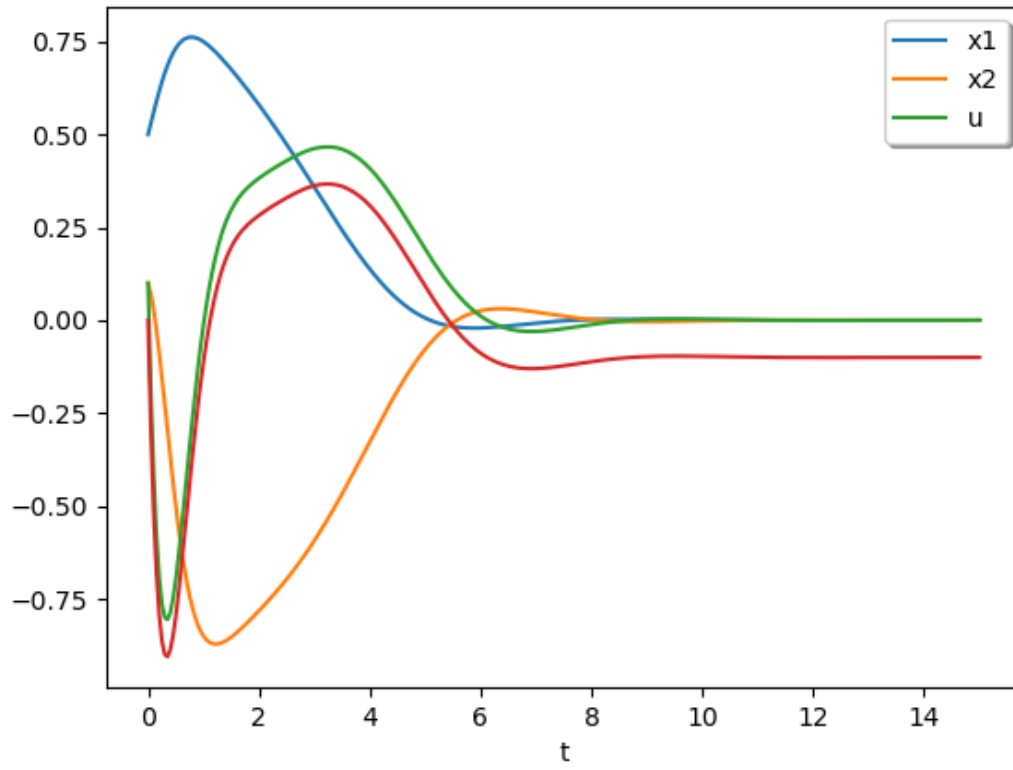
Script Runner

```
1 #!/bin/sh
2
3 # Create a fancy little progress bar
4 # https://stackoverflow.com/questions/238073/how-to-add-a-progress-bar-to-a-shell-script
5 function ProgressBar
6 {
7     # Process data
8     let _progress=(${1}*100/${2}*100)/100
9     let _done=(${_progress}*4)/10
10    let _left=40-${_done}
11
12    # Build progressbar string lengths
13    _fill=$(printf "%${_done}s")
14    _empty=$(printf "%${_left}s")
15
16    # 1.2 Build progressbar strings and print the ProgressBar line
17    # 1.2.1 Output example:
18    # 1.2.1.1 Progress : [#####] 100%
19    printf "\rProcessing $s: [${_fill} // #]${_empty} // -}] ${_progress}% \r"
20 }
21 # Variables
22 i=1
23
24 # Create list of scripts to be ran
25 declare -a scripts=(`find . -type f -name "*.py" ! -name "plot.py"`)
26
27 # Loop through each script
28 for s in "${scripts[@]}"
29 do
30     ## Update progress bar
31     ProgressBar $i ${#scripts[@]}
32
33     ## Process script
34     python $s
35
36     ## Update Index
37     i=$((i+1))
38 done
39
40 printf '\nFinished!\n'
```

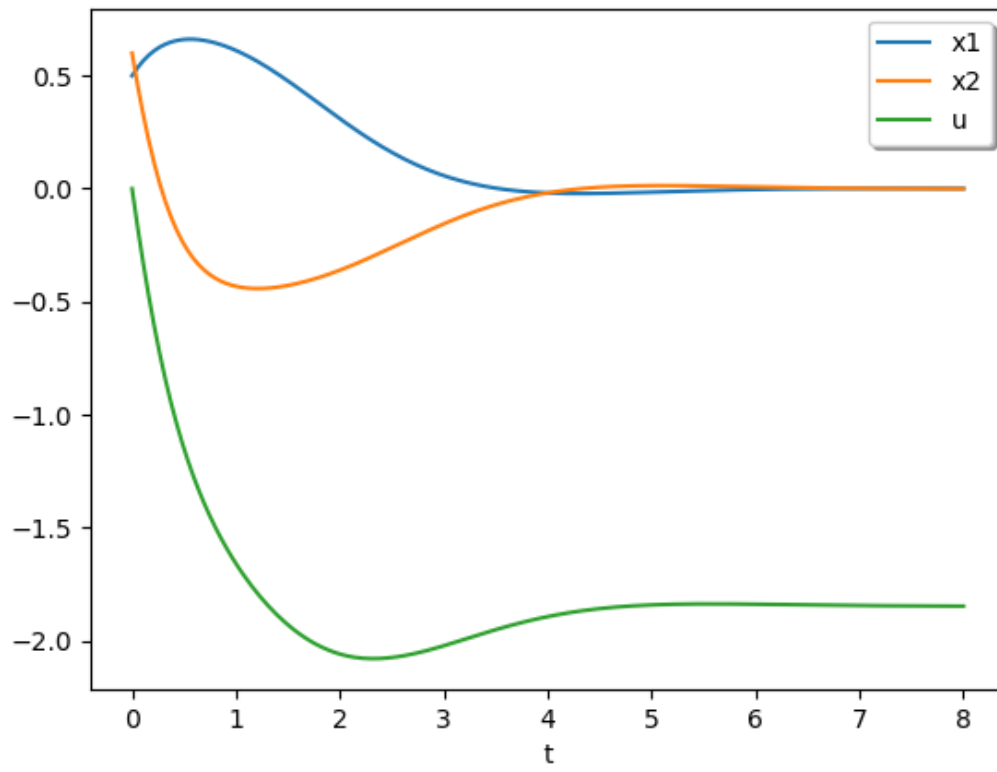
12.2.2.1



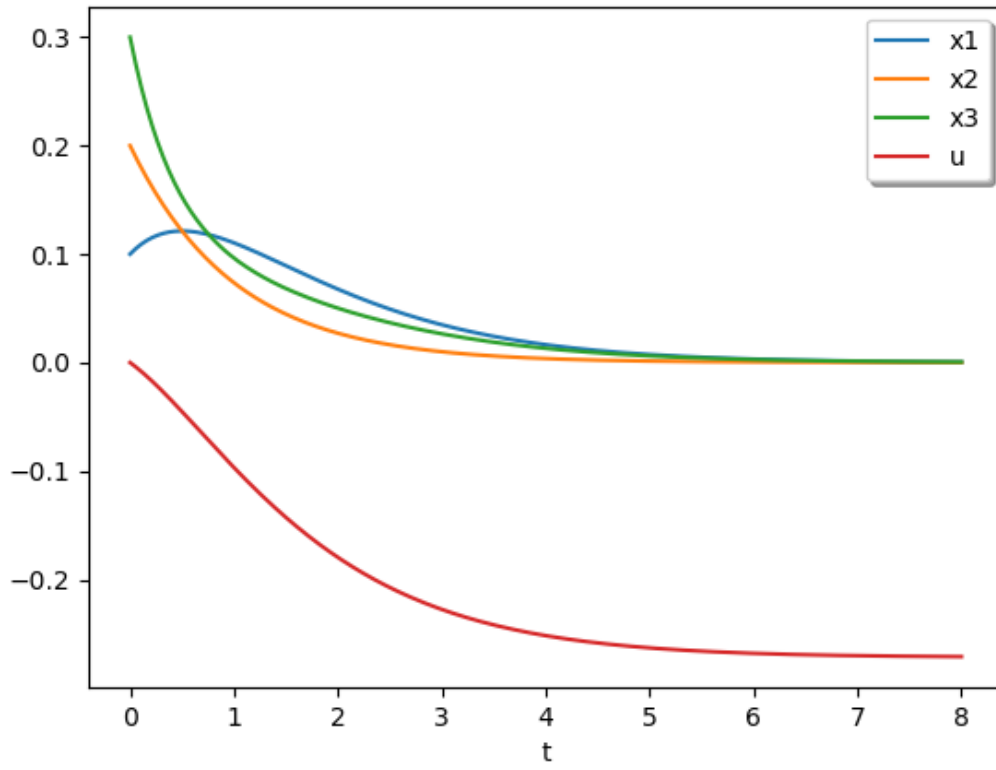
12.2.2.2

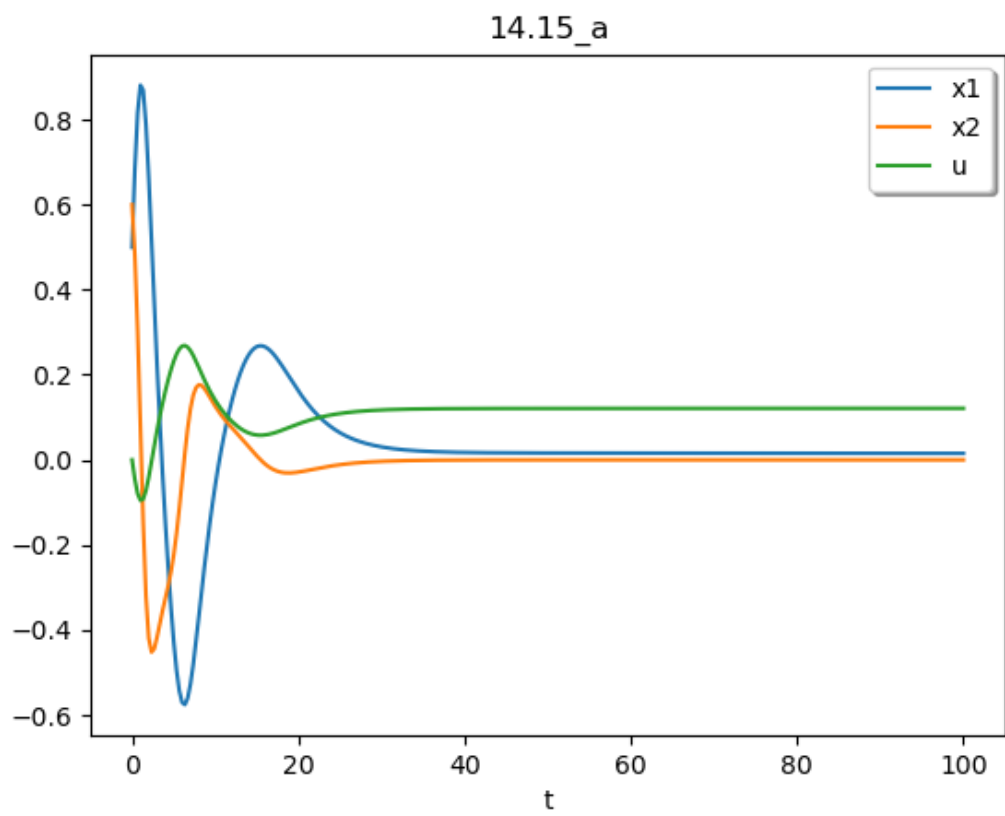


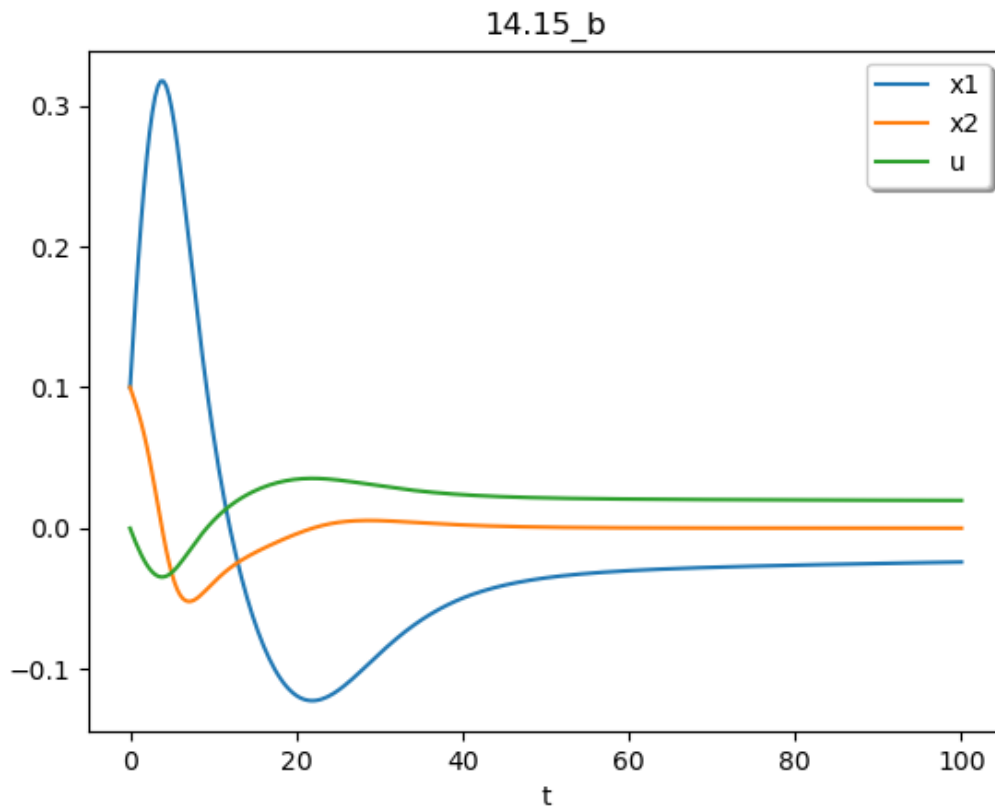
14.31



14.34







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.