lqr

October 4, 2019

```
In [62]: from IPython import display
         import matplotlib.pyplot as plt
         from matplotlib.lines import Line2D
         from gym.envs.mujoco import *
         from envs.hopper_env import HopperModEnv
         from envs.cheetah_env import CheetahModEnv
         import numpy as np
         import copy
         import gym
         from scipy.io import loadmat
         from scipy.io import savemat
         import moviepy.editor as mpy
         from simulators import *
         from rot_utils import *
         import seaborn as sns
         sns.set_style('darkgrid')
         import warnings
         warnings.filterwarnings('ignore')
```

(a) LQR for Linear Systems Let's start with a linear system:

```
In [63]: A = np.array([[0.0481, -0.5049, 0.0299, 2.6544, 1.0608],
                       [2.3846, -0.2312, -0.1260, -0.7945, 0.5279],
                       [1.4019, -0.6394, -0.1401, 0.5484, 0.1624],
                       [-0.0254, 0.4595, -0.0862, 2.1750, 1.1012],
                       [0.5172, 0.5060, 1.6579, -0.9407, -1.4441]])
         B = np.array([[-0.7789, -1.2076],
                       [0.4299, -1.6041],
                       [0.2006, -1.7395],
                       [0.8302, 0.2295],
                       [-1.8465, 1.2780]])
         dx = A.shape[0]
         du = B.shape[1]
```

Now verify the system is controllable.

To do so we need to check if rank($[B \;\; AB \;\; A^2B \;\; A^3B \;\; A^4B]$) == 5. (generally we'd go up to $A^{(n-1)}B$, and verify rank == n)

In [64]:
```
# verify the above statement
lst = [B]
"""YOUR CODE HERE"""
An = np.identity(dx)
for i in range(1, dx):
    An = An.dot(A)
    AnB = An.dot(B)
    lst.append(AnB)

"""YOUR CODE ENDS HERE"""
np.linalg.matrix_rank(np.hstack(lst))
```

Out[64]: 5

Recall the following optimal control problem:

$$\min_{x,u} \sum_{t=0}^{T-1} (x_t' Q x_t + u_t' R u_t) + x_T' Q_{final} x_T \text{s.t. } x_{t+1} = A x_t + B u_t$$

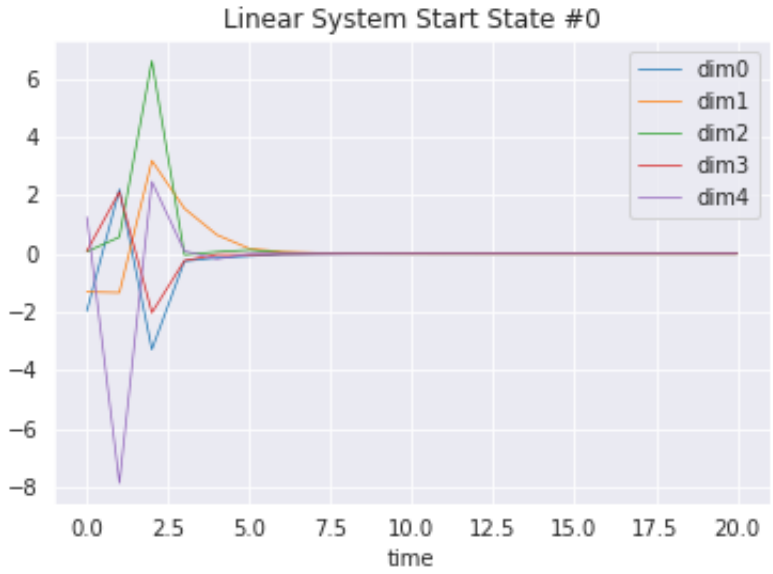
For T going to infinity, we'll have that the Value Function and the Feedback Controller at time $t = 0$ reach a steady-state — the optimal value function and feedback controller for infinitely many time-steps to-go

So let's run the Value Iteration Solution to the LQR control problem until it has converged, and then use that infinite horizon optimal feedback controller to stabilize our system at 0.

In [65]:
```
# implement the infinite horizon optimal feedback controller
def lqr_infinite_horizon(A, B, Q, R):
    """
    find the infinite horizon K and P through running LQR back-ups
    until l2-norm(K_new - K_curr, 2) <= 1e-4
    return: K, P
    """

    dx, du = A.shape[0], B.shape[1]
    P, K_current = np.eye(dx), np.zeros((du, dx))

    """YOUR CODE HERE"""
    while True:
        K_new = - np.linalg.inv(R + B.T @ P @ B) @ B.T @ P @ A
        P = Q + K_new.T @ R @ K_new + (A + B @ K_new).T @ P @ (A + B @ K_new)
        if np.linalg.norm(K_new - K_current, 2) <= 1e-4:
            break
        else:
            K_current = K_new

    """YOUR CODE ENDS HERE"""
    return K_new, P
```

In [66]:
```
# problem has been defined, let's solve it:
Q, R = np.eye(dx), np.eye(du)
K_inf, P_inf = lqr_infinite_horizon(A, B, Q, R)
```

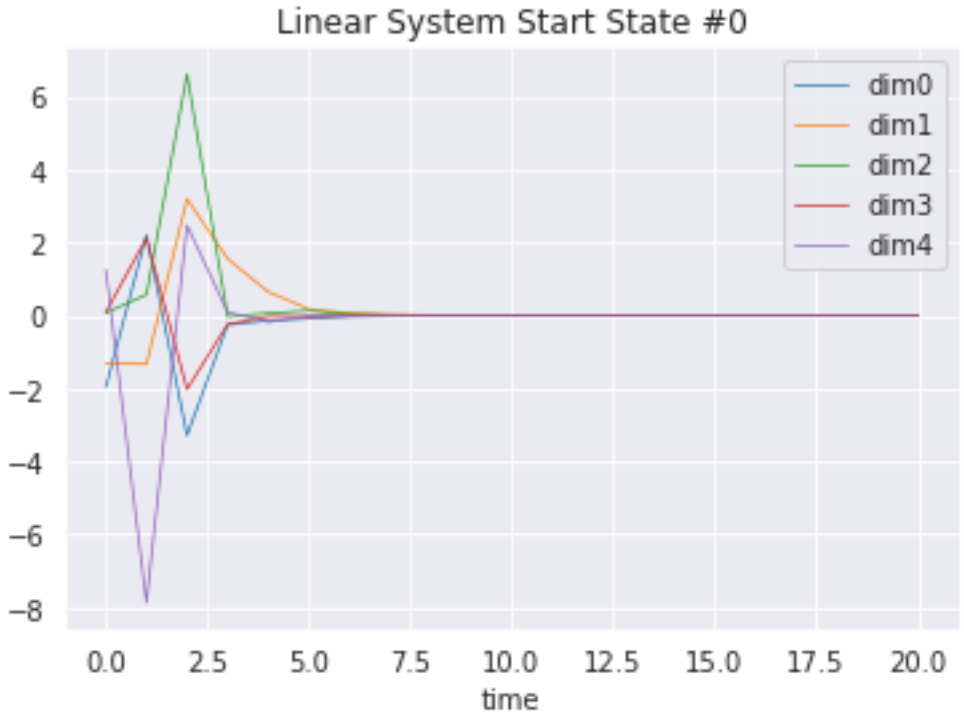
Now let's simulate and see what happens for a few different starting states.
Here's what a reference plot looks like for the *first starting state with no noise* in state dynamics

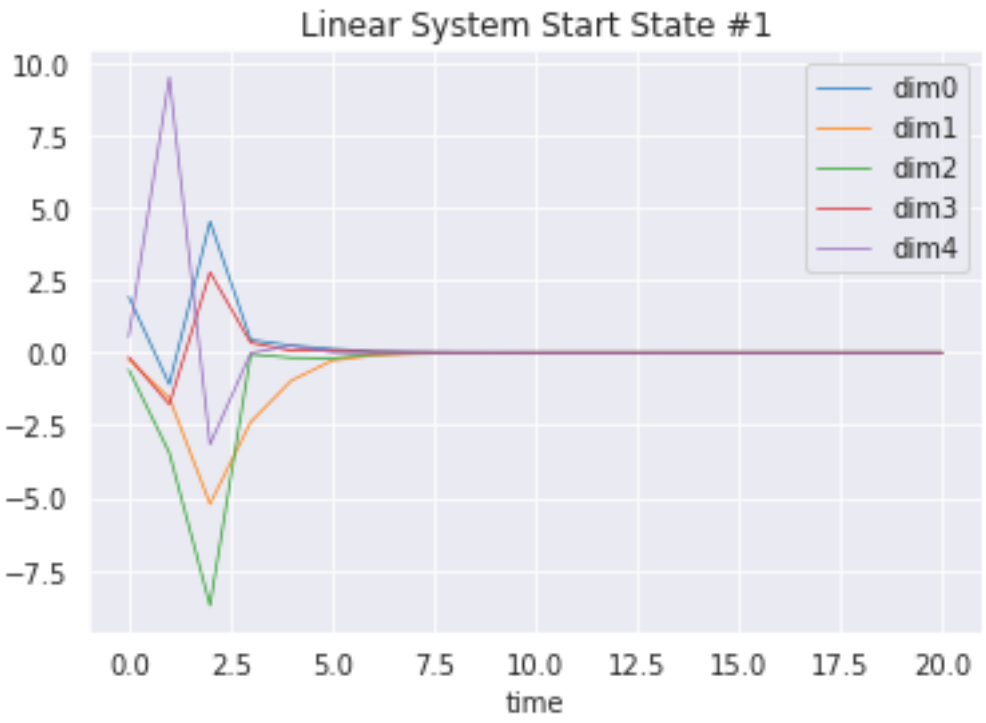


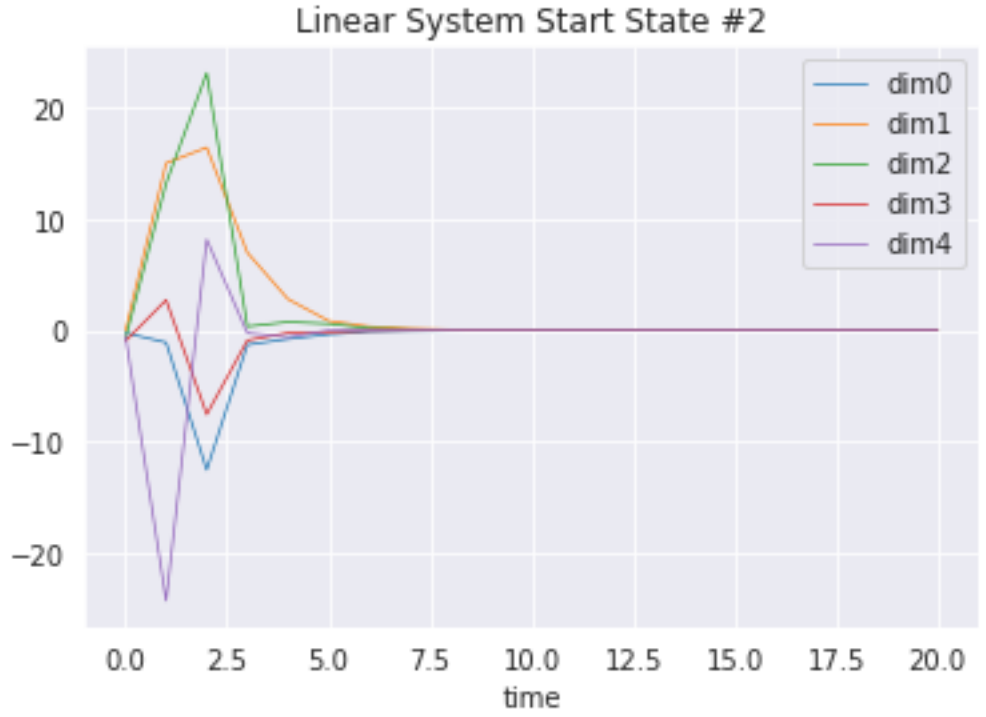
so you may compare:

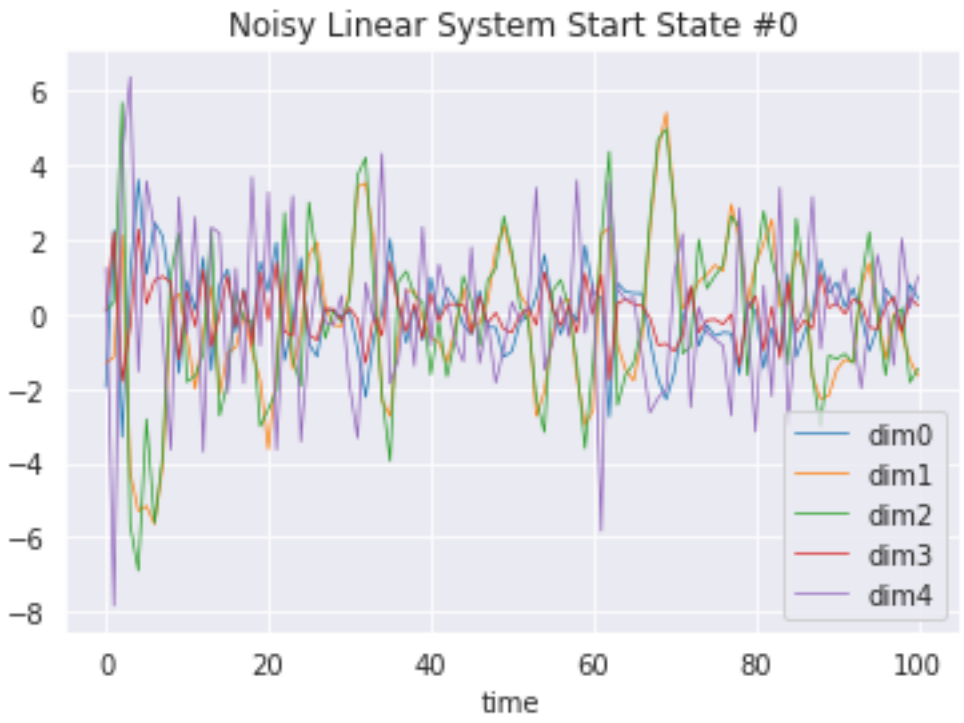
```
In [67]: # fill in the simulation to use your controller, K_inf, at each timestep then run the c
         def simulate(A, B, K_inf, n_starting_states, T, noise=None):
             for s in np.arange(n_starting_states):
                 x, u = np.zeros((K_inf.shape[1], T+1)), np.zeros((K_inf.shape[0], T+1))
                 x[:,0] = starting_states[:,s]
                 for t in np.arange(T):
                     """YOUR CODE HERE"""
                     u[:,t] = K_inf @ x[:,t]
                     """YOUR CODE ENDS HERE"""
                     x[:,t+1] = A @ x[:,t] + B @ u[:,t]
                     if noise is not None:
                         x[:,t+1] += noise[:,t]
                 plt.plot(x.T, linewidth=.7)
                 plt.xlabel('time')
                 plt.title("Noisy Linear System Start State #{}".format(s)) if noise is not None
                 plt.legend(["dim"+str(i) for i in range(len(x))])
                 plt.show()


         starting_states = np.array([[-1.9613, 1.9277, -0.2442],
                                     [-1.3127, -0.2406, -0.0260],
                                     [0.0698, -0.5860, -0.7522],
                                     [0.0935, -0.1524, -0.9680],
                                     [1.2494, 0.5397, -0.5146]])
         n_starting_states = starting_states.shape[1]
         T = 20 # simulating for 20 steps
         simulate(A, B, K_inf, n_starting_states, T)
```

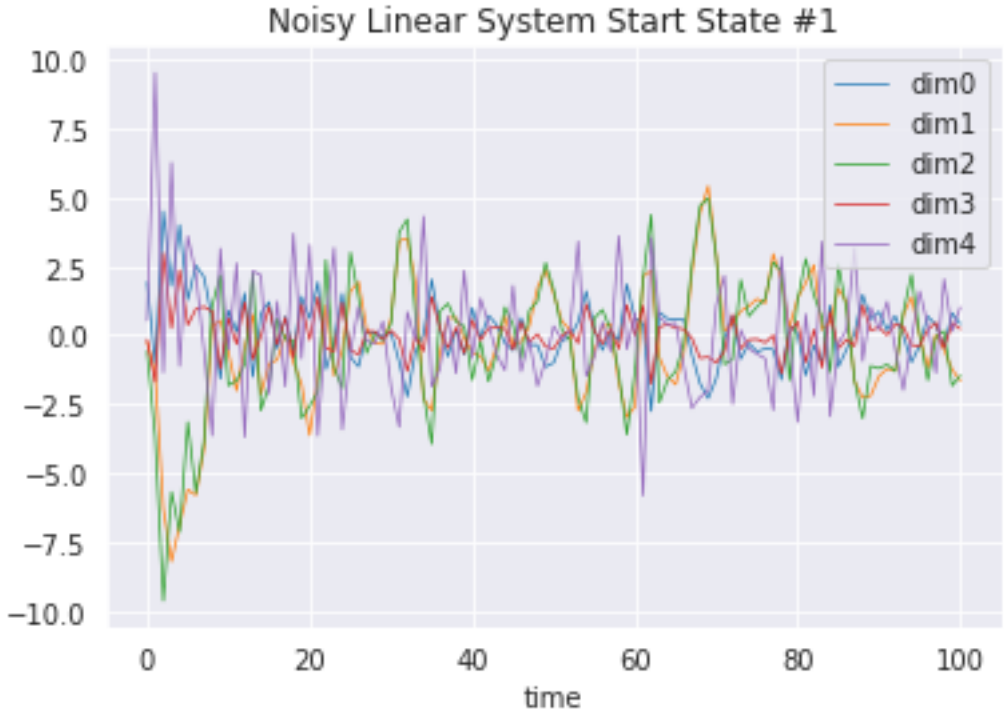
```
# and in the presence of noise:
noise_id = "p_a_w"
T = 100 # simulating for 100 steps
simulate(A, B, K_inf, n_starting_states, T, noise=loadmat("mats/"+noise_id+".mat")[nois
```

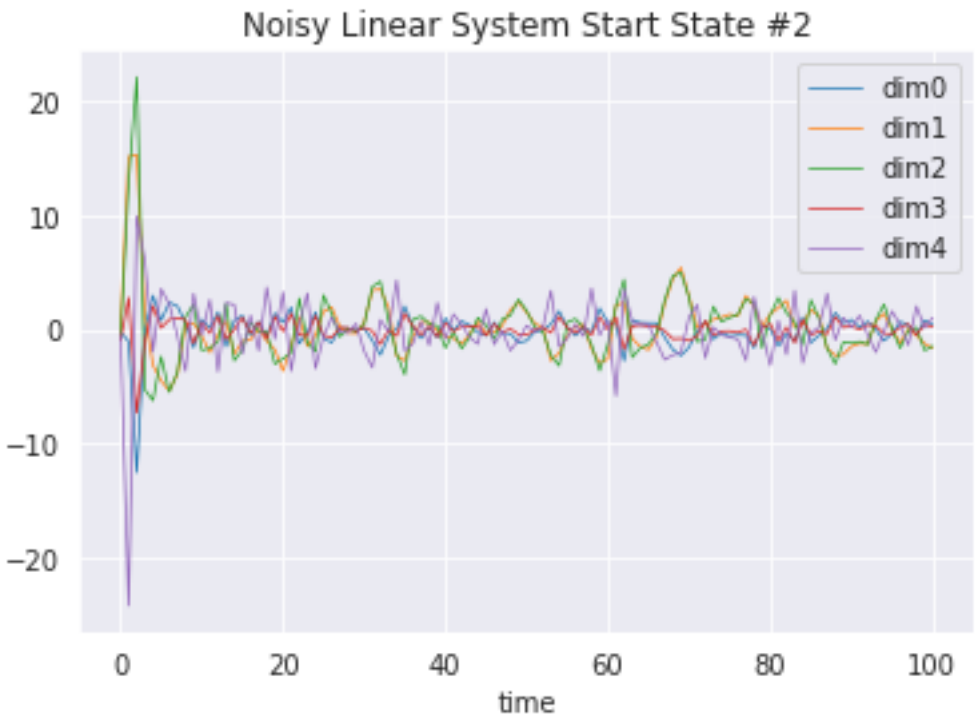



Linear System Start State #1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
time
dim0
dim1
dim2
dim3
dim4


Linear System Start State #2
20
10
0
−10
−20
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
time
dim0
dim1
dim2
dim3
dim4


Noisy Linear System Start State #0
6
4
2
0
−2
−4
−6
−8
0
20
40
60
80
100
time
dim0
dim1
dim2
dim3
dim4


Noisy Linear System Start State #1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
20
40
60
80
100
time
dim0
dim1
dim2
dim3
dim4



(b) LQR-based Stabilization for Nonlinear Systems Now let's consider nonlinear systems. Linearize around one point and design an infinite horizon controller for the resulting system.

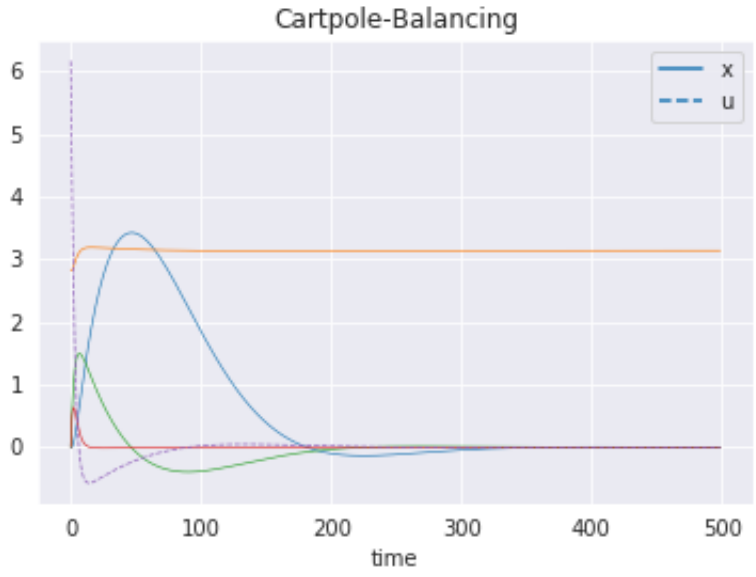
```
In [72]: # implement linearization about a point
         def linearize_dynamics(f, x_ref, u_ref, dt, my_eps, x_ref_tplus1=None):
             """
             f : dynamics simulator
             my_eps : delta for forward and backward differences you'll need
             NOTE: please use centered finite differences!

             x(:,t+1) - x_ref  approximately = A*( x(:,t)-x_ref ) + B* ( u(:,t) - u_ref ) + c
             If we pick x_ref and u_ref to constitute a fixed point, then c == 0

             For part (b), you do not need to use the optional argument (nor c).
             For part (d), you'll have to revisit and modify this function
                 --at this point, you'll want to use the optional argument and the resulting c.

             return: A, B, c
             """

             if x_ref_tplus1 is not None:
                 x_ref_next = x_ref_tplus1
             else:
```

```
        x_ref_next = x_ref

    dx, du = x_ref.shape[0], u_ref.shape[0]
    A, B = np.zeros((dx, dx)), np.zeros((dx, du))

    """YOUR CODE HERE"""
    for j in range(dx):
        xj_forward = x_ref.copy()
        xj_forward[j] = xj_forward[j] + my_eps / 2
        xj_backward = x_ref.copy()
        xj_backward[j] = xj_backward[j] - my_eps / 2
        A[:, j] = (f(xj_forward, u_ref, dt) - f(xj_backward, u_ref, dt)) / (my_eps)

    for j in range(du):
        uj_forward = u_ref.copy().astype(float)
        uj_forward[j] = uj_forward[j] + my_eps / 2
        uj_backward = u_ref.copy().astype(float)
        uj_backward[j] = uj_backward[j] - my_eps / 2
        B[:, j] = (f(x_ref, uj_forward, dt) - f(x_ref, uj_backward, dt)) / (my_eps)
    """YOUR CODE ENDS HERE"""

    c = f(x_ref, u_ref, dt) - x_ref_next
    if len(B.shape) == 1:
        return A, B.reshape(-1, 1), c
    return A, B, c
```

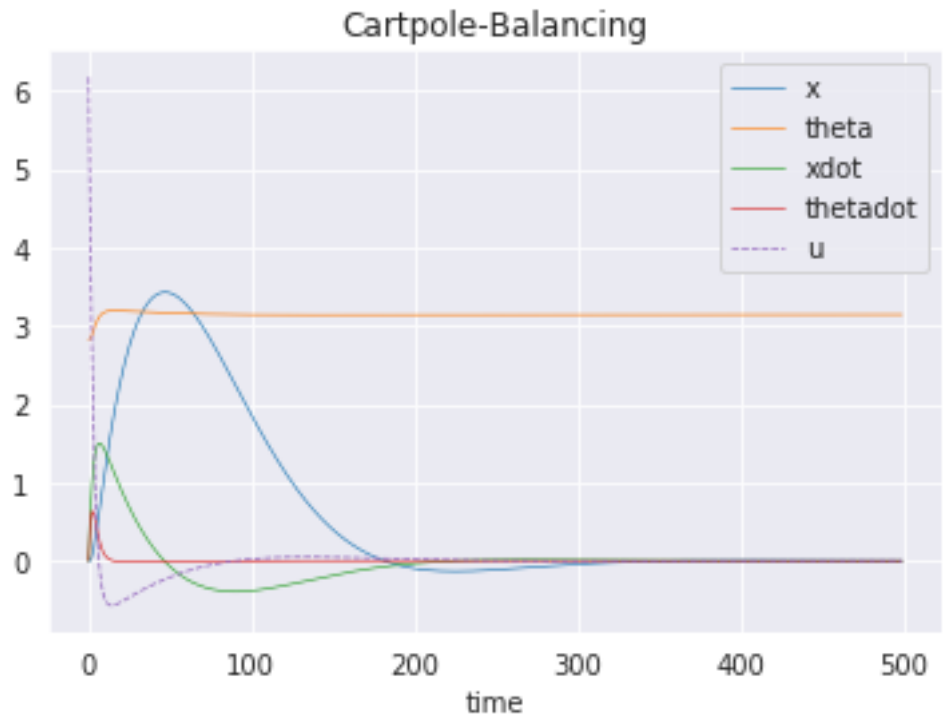
In [73]:
```
# take an environment and find the infinite horizon controller for the linearized syste
def lqr_nonlinear(config):
    env = config['env']
    f = config['f']
    dt = 0.1 # we work with discrete time
    my_eps = 0.01 # finite difference for numerical differentiation

    # load in our reference points
    x_ref, u_ref = config['x_ref'], config['u_ref']

    # linearize
    A, B, c = linearize_dynamics(f, x_ref, u_ref, dt, my_eps)
    dx, du = A.shape[0], B.shape[1]
    Q, R = np.eye(dx), np.eye(du)*2

    # solve for the linearized system
    K_inf, P_inf = lqr_infinite_horizon(A, B, Q, R) # you implemented in part (a)

    # recognize the simulation code from part (a)? modify it to use your controller at
    def simulate(K_inf, f, x_ref, u_ref, dt, n_starting_states, T, noise=None):
        for s in np.arange(n_starting_states):
            x, u = np.zeros((K_inf.shape[1], T+1)), np.zeros((K_inf.shape[0], T+1))
```

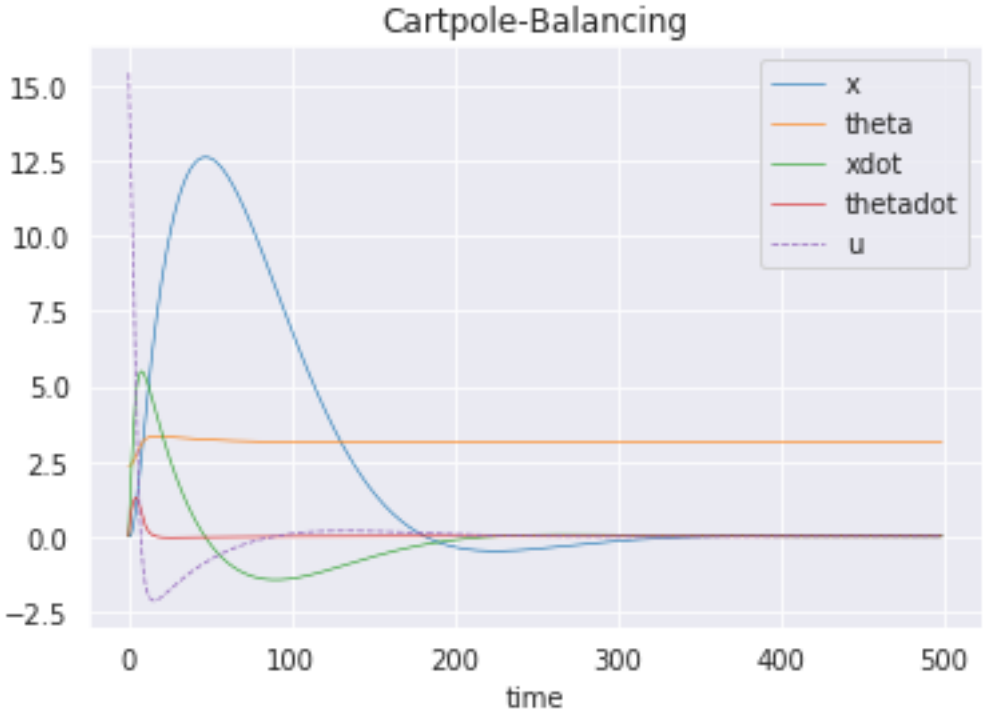
```
            x[:,0] = starting_states[:,s]
            for t in np.arange(T):
                """YOUR CODE HERE"""
                u[:,t] = u_ref + K_inf @ (x[:, t] - x_ref)
                """YOUR CODE ENDS HERE"""
                x[:,t+1] = f(x[:,t], u[:,t], dt)
                if "p_val" in config.keys():
                    perturbation_values = config["p_val"]
                    perturb = perturbation_values[t//(T//len(perturbation_values))]
                    x[:,t+1] = f(x[:,t], u[:,t], dt, rollout=True,perturb=perturb)
                if env is not None:
                    if t % 5 == 0:
                        plt.clf()
                        plt.axis('off')
                        plt.grid(b=None)
                        plt.imshow(env.render(mode='rgb_array', width=256, height=256))
                        plt.title("Perturbation Magnitude {}".format(perturb))
                        display.clear_output(wait=True)
                        display.display(plt.gcf())

                if noise is not None:
                    x[:,t+1] += noise[:,t]
            if env is not None:
                plt.clf()

            plt.plot(x.T[:-1], linewidth=.6)
            plt.plot(np.squeeze(u.T[:-1])/10.0, linewidth=.7, linestyle='--') # scaling
            if 'legend' in config.keys():
                config['legend'].append('u')
                plt.legend(config['legend'])
            else:
                legend_elements = [Line2D([0], [0], label='x'),Line2D([0], [0], linesty
                plt.legend(handles=legend_elements)
            plt.xlabel('time')
            plt.title(config["exp_name"])
            plt.show()

    # now let's simulate and see what happens for a few different starting states
    starting_states = config['ss']
    n_starting_states = starting_states.shape[1]
    T = config['steps'] # simulating for T steps
    simulate(K_inf, f, x_ref, u_ref, dt, n_starting_states, T)
    if 'noise' in config.keys():
        # and now in the presence of noise
        noise_id = config['noise']
        noise_loaded = loadmat("mats/"+noise_id+".mat")[noise_id]
        simulate(K_inf, f, x_ref, u_ref, dt, n_starting_states, noise_loaded.shape[1],
```

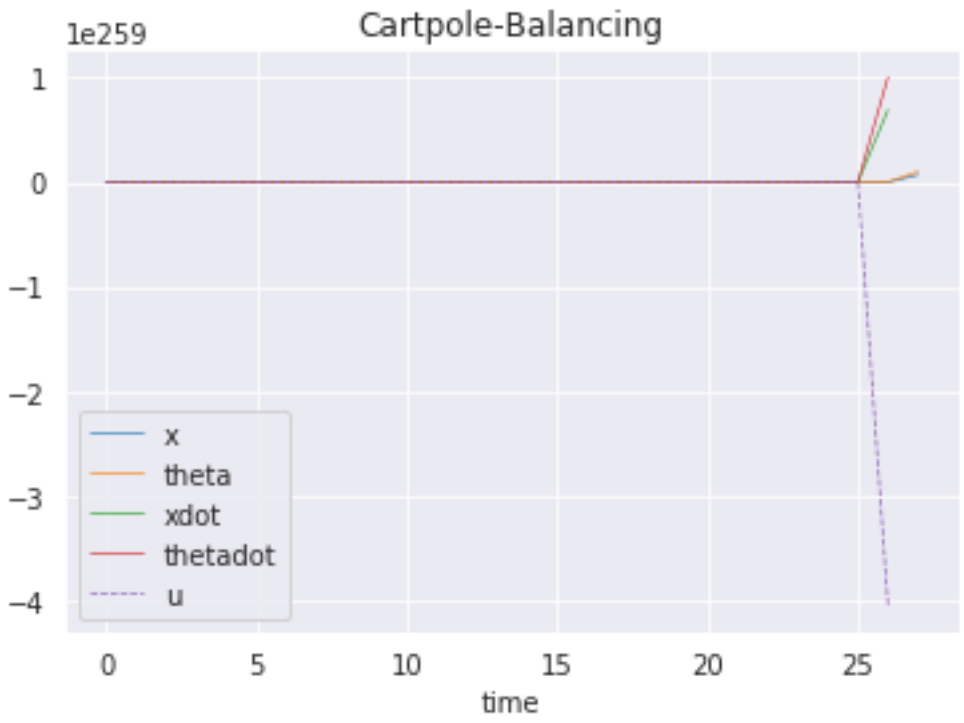
Cartpole-Balancing Here's what a reference plot looks like for the *first starting state with no noise* in state dynamics so you may compare:

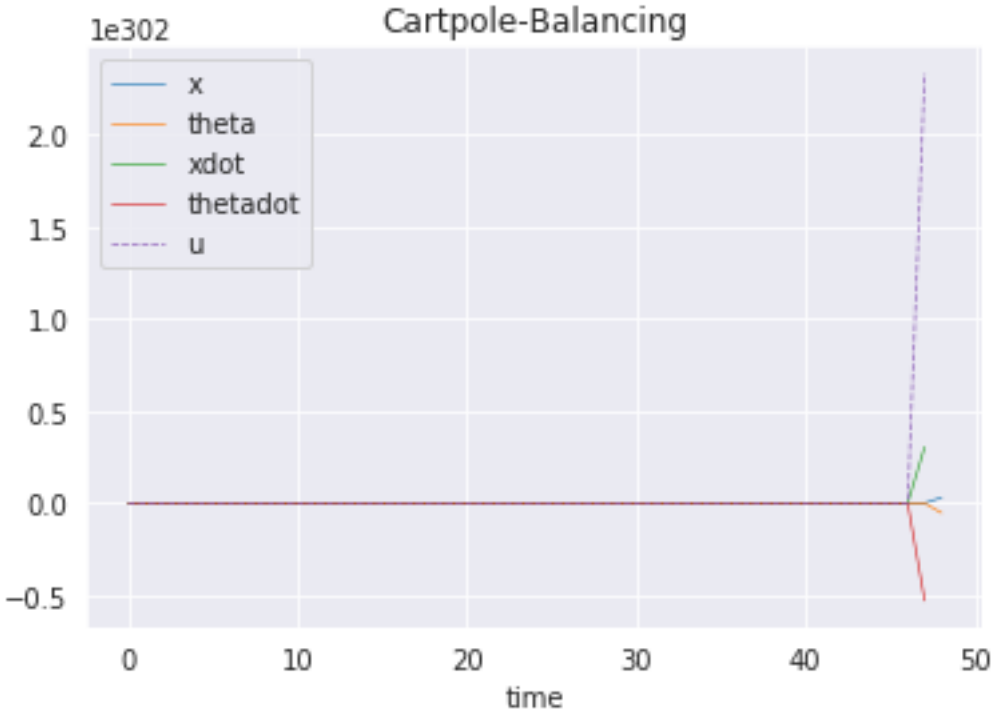


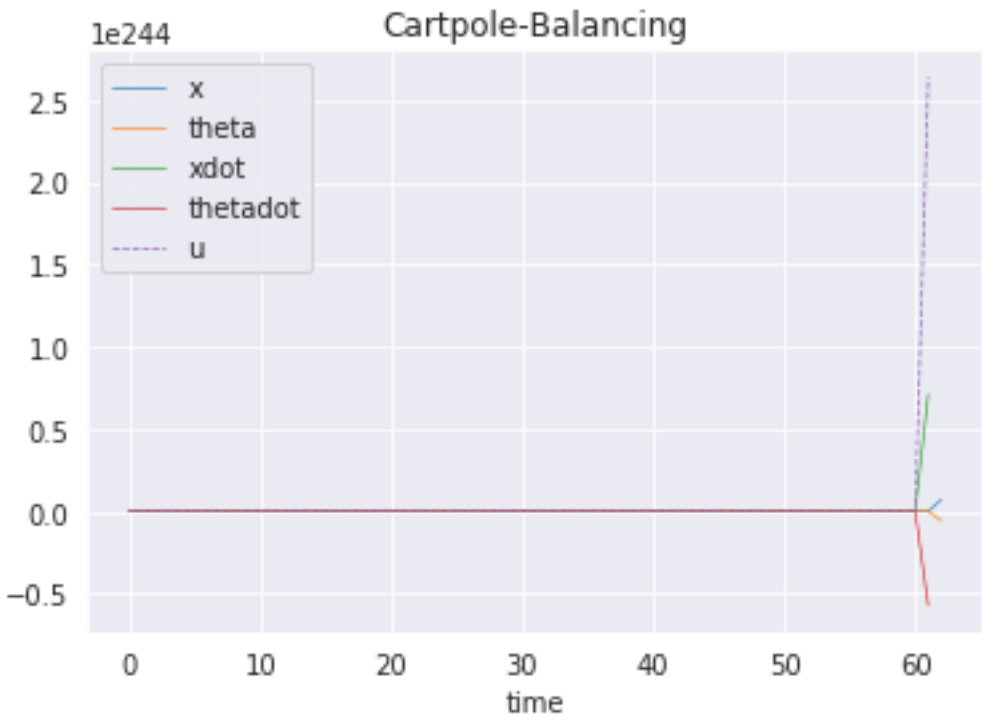
In [74]:
```
# Find the infinite horizon controller for the linearized version of the cartpole balan
cartpole_config = {
    'f': sim_cartpole,
    'exp_name': "Cartpole-Balancing",
    'env': None,
    'steps': 500,
    'x_ref': np.array([0, np.pi, 0, 0]),
    'u_ref': np.array([0]),
    'legend':['x', 'theta', 'xdot', 'thetadot'],
    'ss': np.array([[0, 0, 0, 10, 50],
                    [9*np.pi/10, 3*np.pi/4, np.pi/2, 0, 0],
                    [0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0]]), #ss = starting states
    'noise': 'p_b_w',
}
lqr_nonlinear(cartpole_config)
```

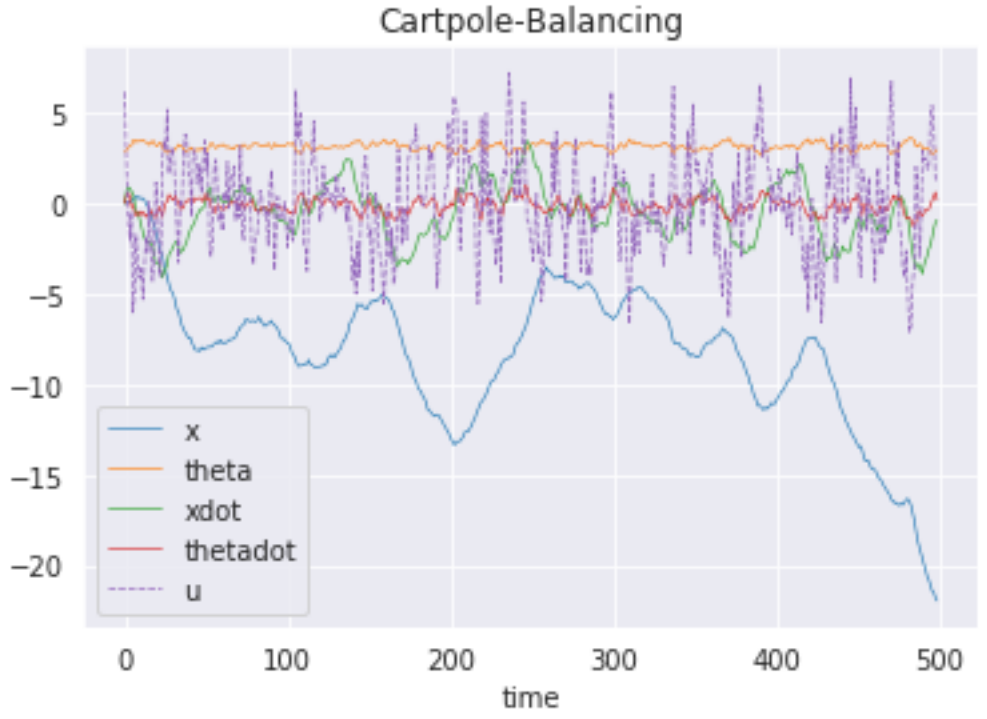

Cartpole-Balancing
x
theta
xdot
thetadot
u
6
5
4
3
2
1
0
0
100
200
300
400
500
time

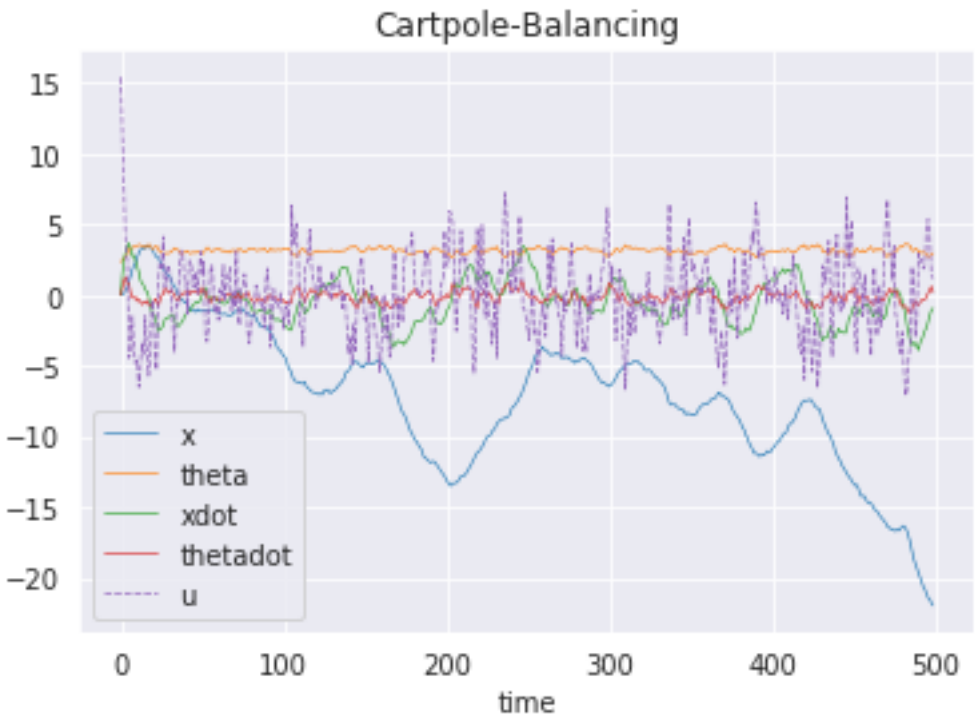

Cartpole-Balancing
x
theta
xdot
thetadot
u
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
0
100
200
300
400
500
time

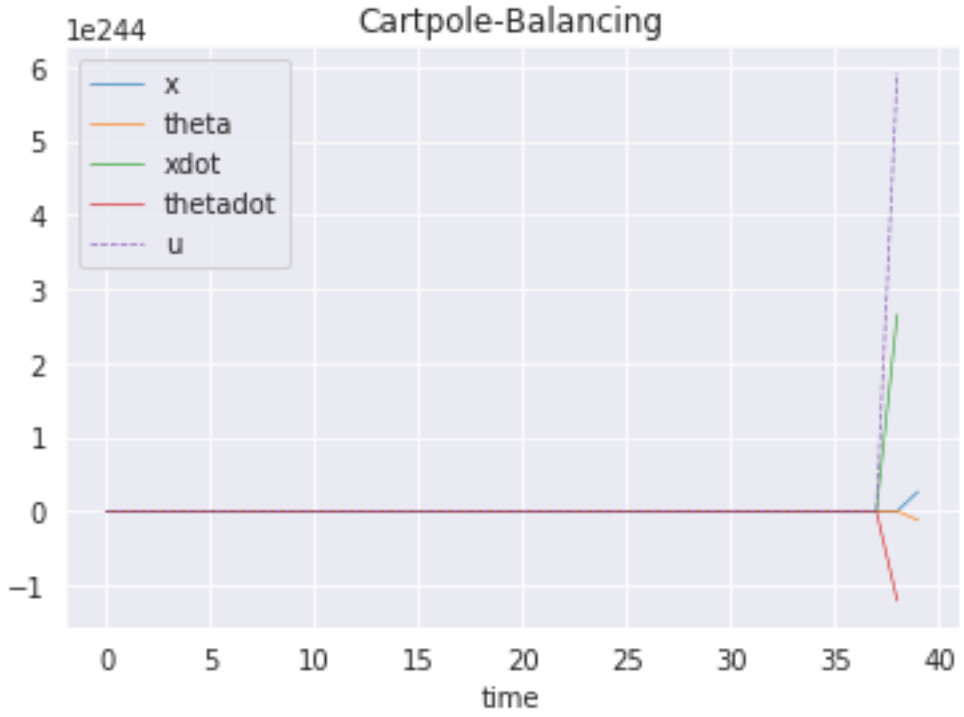

Cartpole-Balancing
1e259
1
0
−1
−2
−3
−4
x
theta
xdot
thetadot
u
0
5
10
15
20
25
time

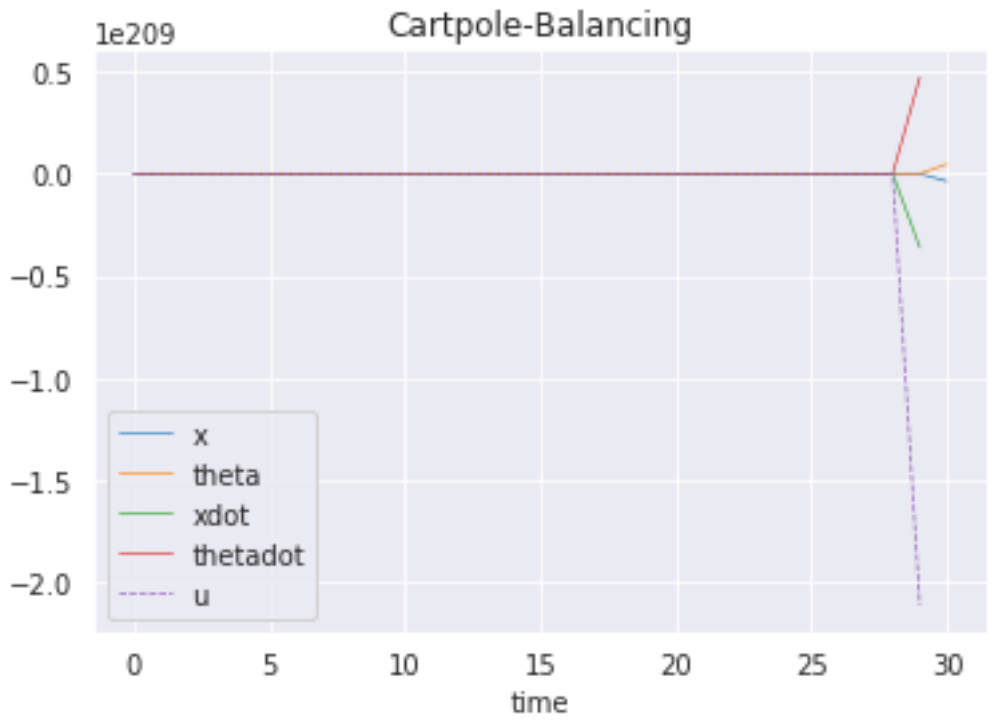

Cartpole-Balancing
1e302
2.0
1.5
1.0
0.5
0.0
−0.5
x
theta
xdot
thetadot
u
0
10
20
30
40
50
time

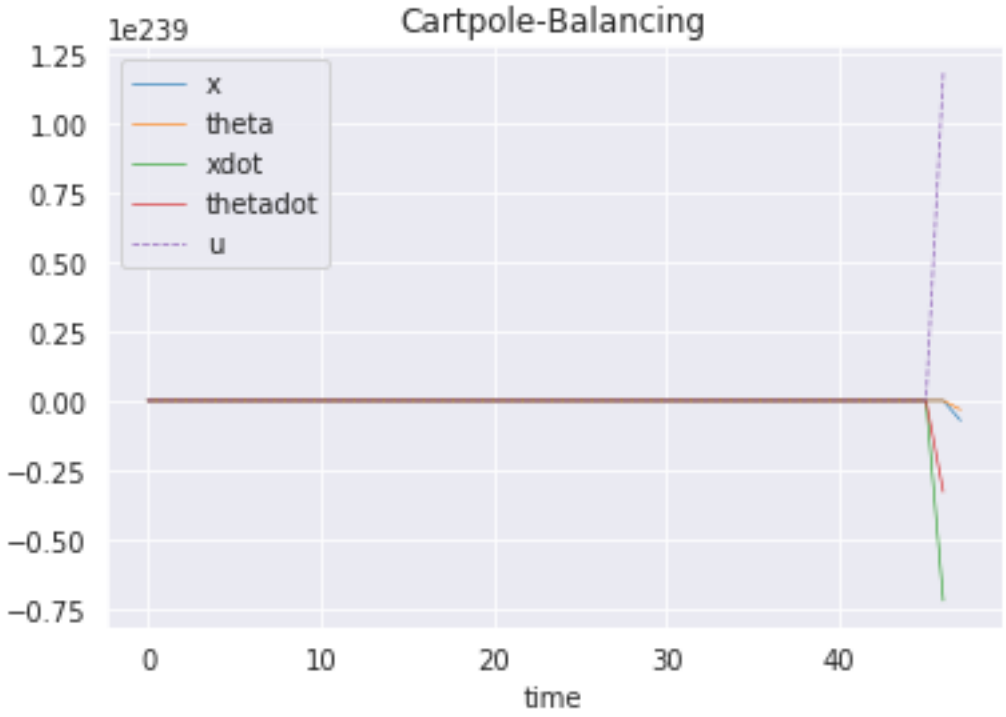

Cartpole-Balancing
1e244
x
theta
xdot
thetadot
u
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0
10
20
30
40
50
60
time


Cartpole-Balancing
x
theta
xdot
thetadot
u
5
0
−5
−10
−15
−20
0
100
200
300
400
500
time


Cartpole-Balancing
15
10
5
0
−5
−10
−15
−20
x
theta
xdot
thetadot
u
0
100
200
300
400
500
time


Cartpole-Balancing
1e244
6
5
4
3
2
1
0
−1
x
theta
xdot
thetadot
u
0
5
10
15
20
25
30
35
40
time

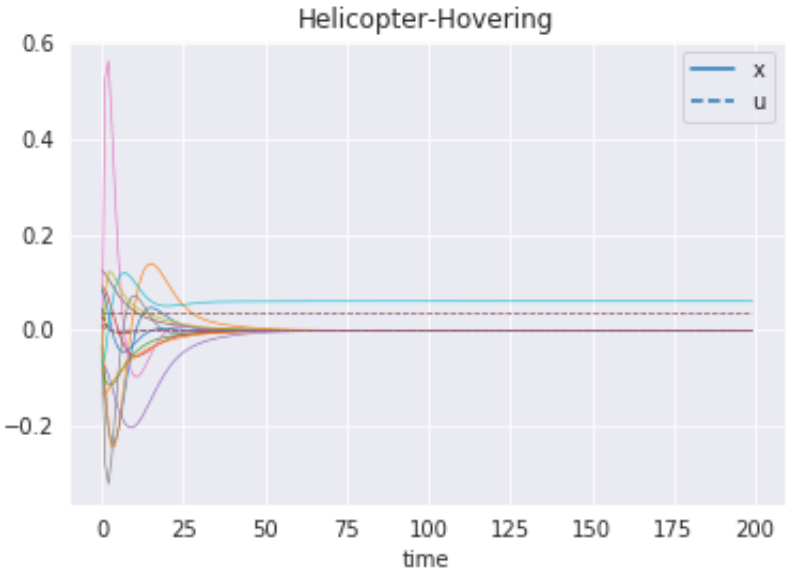

Cartpole-Balancing
1e209
0.5
0.0
−0.5
−1.0
−1.5
−2.0
x
theta
xdot
thetadot
u
0
5
10
15
20
25
30
time


Cartpole-Balancing
1e239
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
x
theta
xdot
thetadot
u
0
10
20
30
40
time

Question: Inspecting the generated plots for the system without noise, explain in 2-3 sentences how the different starting states affect the problem differently. In your answer, include an analysis of one starting state which causes failure and explain why this might happen.

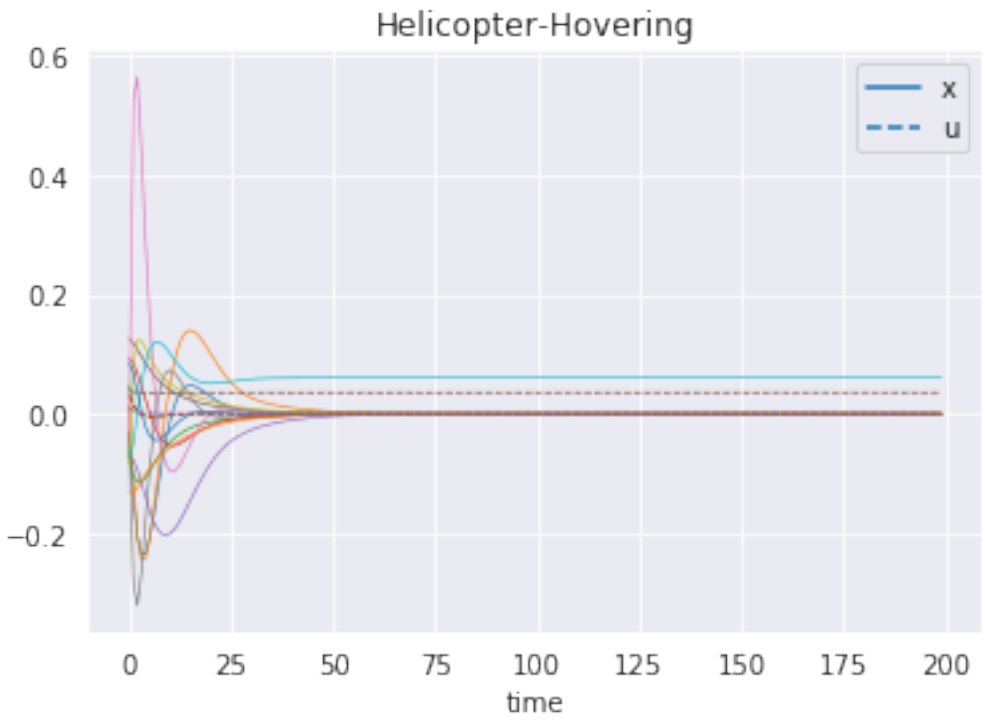
Response: The starting states may be close to or far from the reference state. In the latter case, the linearization using Taylor's expansion drives the system far away from its original nonlinearity, leading to the inaccurate approximation. For instance, the last starting state is [50, 0, 0, 0] which is very far from the reference state [0, pi, 0, 0], resulting in the diverging result.

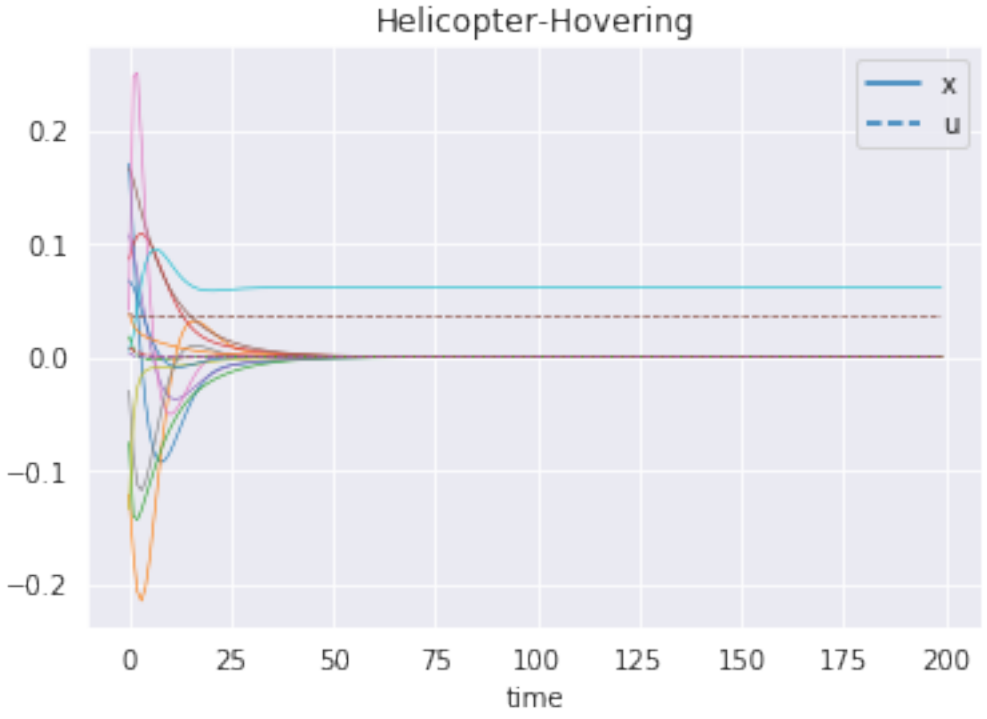
Helicopter in Hover Now let's stabilize a helicopter in hover. We can use the *same* code written for the cartpole system to do so; simply run the cell below which defines the fixed point. Here's what a reference plot looks like for the *first starting state with no noise* in state dynamics so

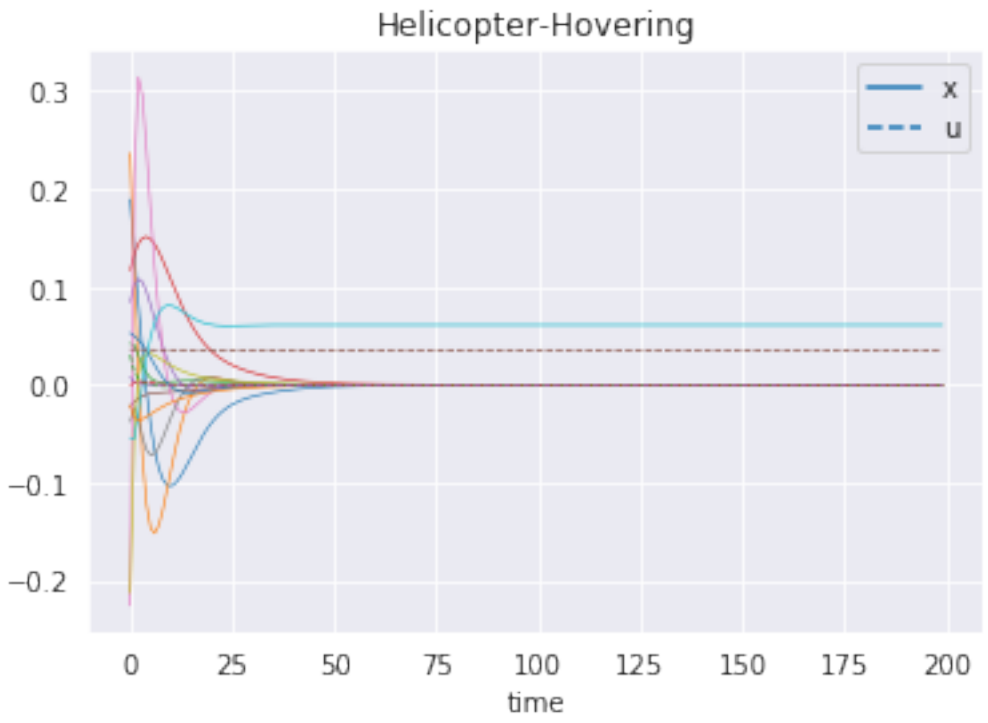


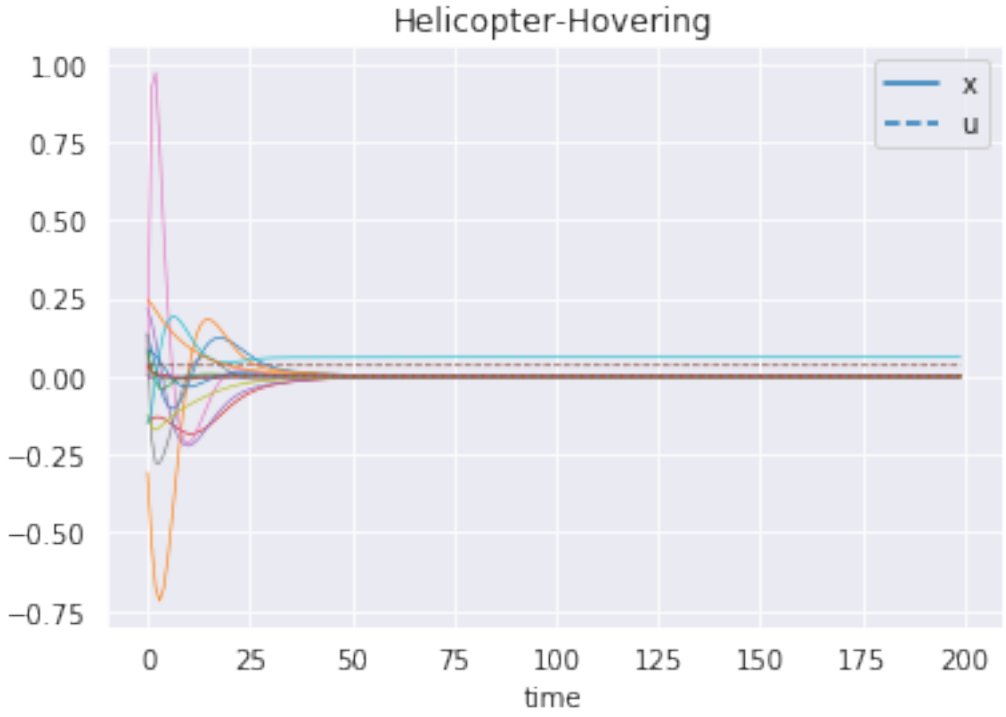
you may compare:

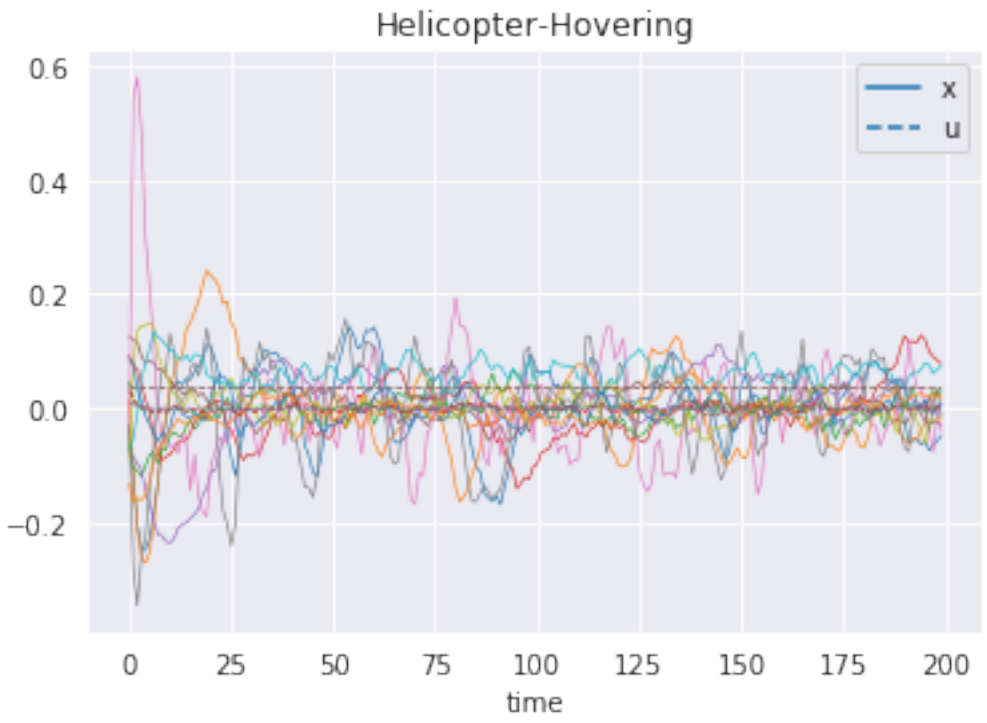
```
In [10]: # Find the infinite horizon controller for the linearized version of the hovering copte
         # Just run the cell below to generate plots using the code you wrote for cartpole!
         x_ref, u_ref = np.zeros(12), np.zeros(4)
         x_ref[9] = np.arcsin(3.0/(5*9.81))
         u_ref[3] = 9.81*5*np.cos(x_ref[9])/137.5
         heli_config = {
             'f': sim_heli,
             'env': None,
             'exp_name': "Helicopter-Hovering",
             'steps': 200,
             'x_ref': x_ref,
             'u_ref': u_ref,
             'ss': loadmat("mats/p_c_heli_starting_states.mat")["heli_starting_states"], #ss = s
             'noise': 'p_c_w',
         }
         lqr_nonlinear(heli_config)
```

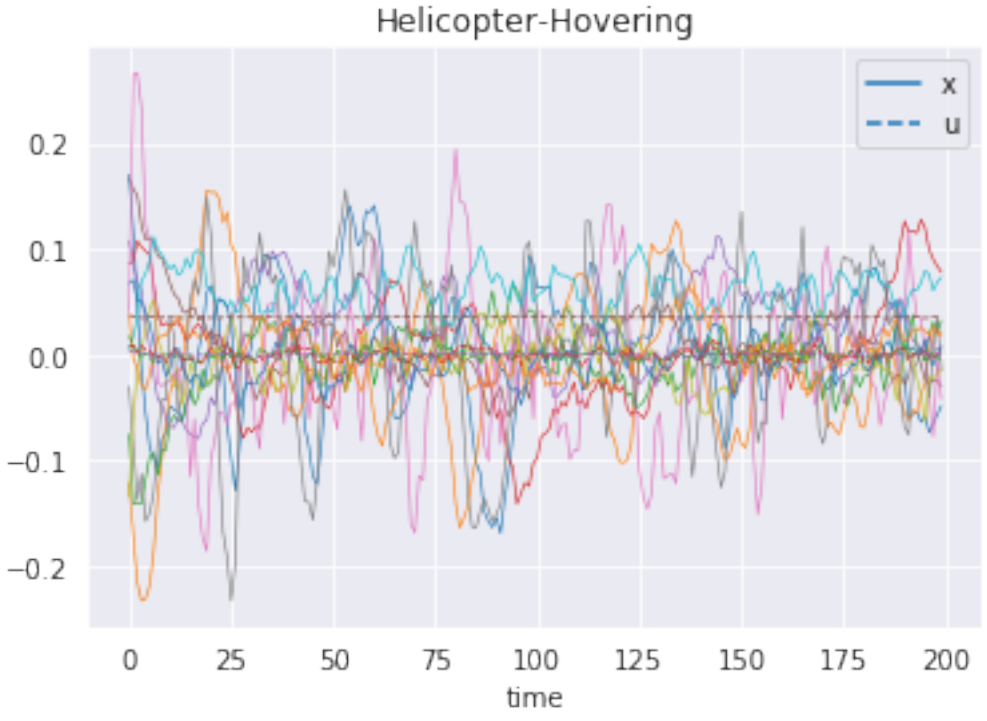

Helicopter-Hovering
0.6
0.4
0.2
0.0
−0.2
x
u
0
25
50
75
100
125
150
175
200
time

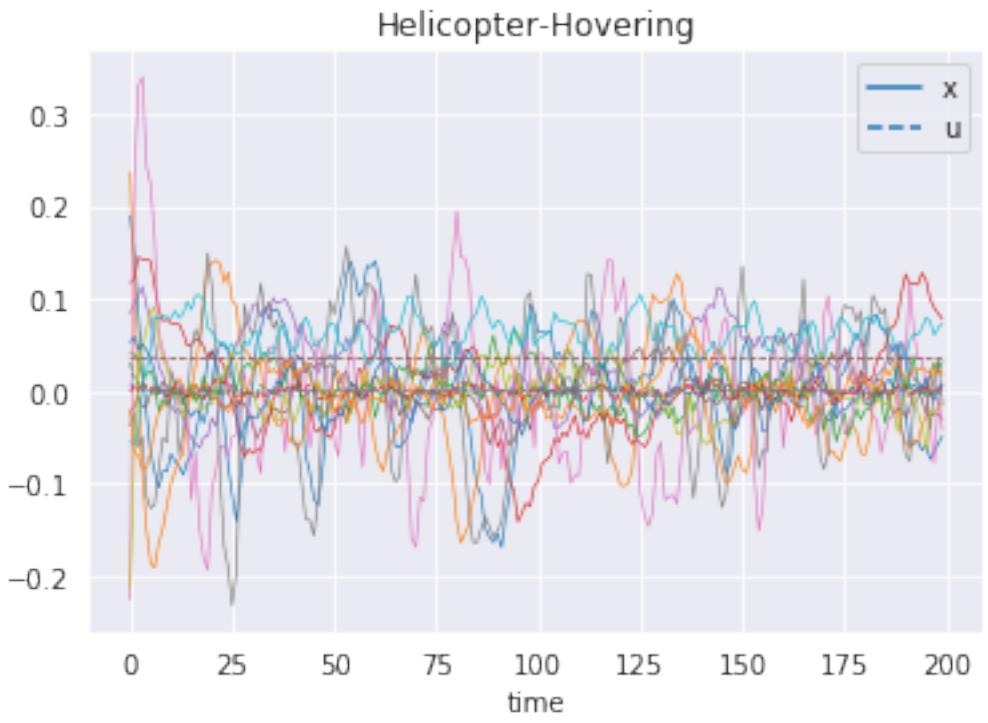

Helicopter-Hovering
0.2
0.1
0.0
−0.1
−0.2
x
u
0
25
50
75
100
125
150
175
200
time

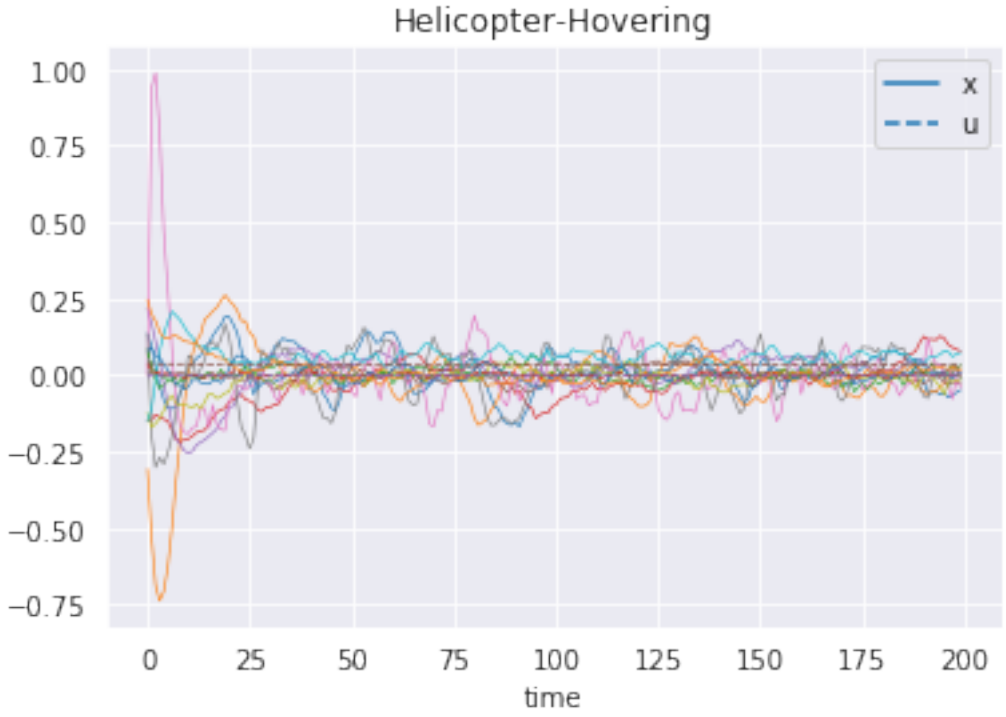

Helicopter-Hovering
x
u
0.3
0.2
0.1
0.0
−0.1
−0.2
0
25
50
75
100
125
150
175
200
time


Helicopter-Hovering
x
u
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
25
50
75
100
125
150
175
200
time


Helicopter-Hovering
x
u
0.6
0.4
0.2
0.0
−0.2
0
25
50
75
100
125
150
175
200
time


Helicopter-Hovering
x
u
0.2
0.1
0.0
−0.1
−0.2
0
25
50
75
100
125
150
175
200
time


Helicopter-Hovering
x
u
0.3
0.2
0.1
0.0
−0.1
−0.2
0
25
50
75
100
125
150
175
200
time


Helicopter-Hovering
x
u
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
25
50
75
100
125
150
175
200
time