homework 8

January 3, 2025

1 Homework 8 (DL Friday, November 15 at 12:00 PM)

ELEC-E8740 - Basics of sensor fusion - Autumn 2024

1.0.1 Question: Consider the scalar differential equation

$$\dot{x} = ax + u, \qquad x(0) = x_0, \tag{1}$$

where $u = u(t)$ is some given input function.

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
```

1.0.2 Part a (2 points): With discretization step dt, form discretization of the model with zeroth-order-hold (ZOH) approximation in form

$$x_n = f_n x_{n-1} + l_n u_{n-1}. \tag{2}$$

1.0.3 In the section below, implement the solution of f_n and l_n:

[9]:
```python
def discretized_model_parameters_zoh(a, dt):
    """ Implement parameters f_n and l_n of discretized dynamic model such that␣
 ↪x_n = f_n x_{n-1} + l_n u_{n-1}
    Input:
        a: parameter of the given continous time model in the question
        dt: discretization step
    Output:
        f_n and l_n
    """
    # f_n = np.exp(a * dt)
    # l_n = ?
    # YOUR CODE HERE
    # Zeroth-Order Hold discretization
    f_n = np.exp(a * dt)

    l_n = dt

    return f_n, l_n # do not change this line, do not change the order of output
```

[10]:
```
"""Check the result for several inputs"""
assert np.allclose(discretized_model_parameters_zoh(-0.3, 0.01)[0], 0.997,␣
↪rtol=1e-03, atol=1e-04)
```

[11]:
```
"""Check the result for several inputs"""
assert np.allclose(discretized_model_parameters_zoh(-0.3, 0.01)[1], 0.01,␣
↪rtol=1e-03, atol=1e-04)
```

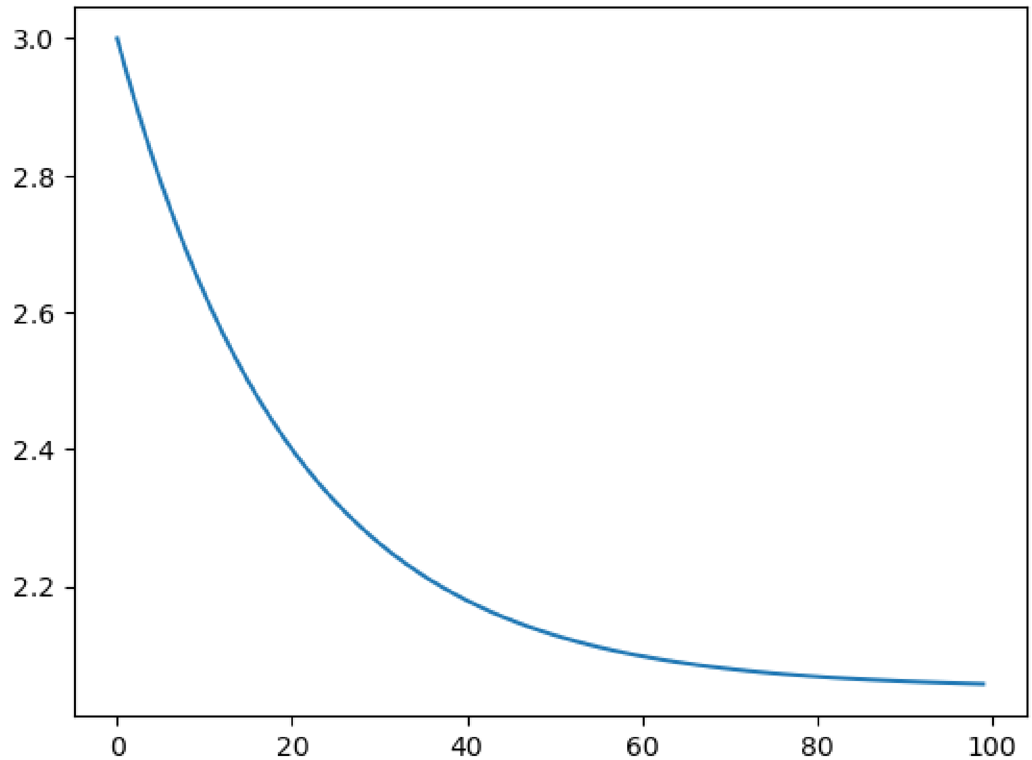
1.0.4 Part b (1 point): Implement the trajectory of the discretized model

[12]:
```
def x_trajectory(x0, steps, a , dt, ut):
    """ Implement trajectory of discretized dynamic model --> x_n = f_n x_{n-1}␣
↪+ l_n u_{n-1}
    Input:
        x0: initial point
        steps: total time steps
        a: parameter of the given continuous time model in the question
        dt: discretization step
        ut: input of the continuous time model
    Output:
        x: trajectory of x
    """
    x = np.zeros((steps,))
    x[0] = x0
    # implement the trajectory
    # you could use discretized_model_parameters_zoh(a, dt) - note that it has␣
↪two outputs
    # YOUR CODE HERE
    f_n, l_n = discretized_model_parameters_zoh(a, dt)

    for n in range(1, steps):
        x[n] = f_n * x[n-1] + l_n * ut
    return x
```

[13]:
```
"""Check the result for the given inputs"""
assert np.allclose(x_trajectory(3., 100, -0.5 , 0.1, 1.)[0], 3.0, rtol=1e-03,␣
↪atol=1e-04)
assert (x_trajectory(3., 100, -0.5 , 0.1, 1.)[2] - x_trajectory(3., 100, -0.5 ,␣
↪0.1, 1.)[0] < 0)
assert (x_trajectory(3., 100, -0.5 , 0.1, 1.)[-1] - x_trajectory(3., 100, -0.5␣
↪, 0.1, 1.)[-5] < 0)
```

Uncomment and then Run the code below to see the trajectory By assuming that $u(t) = 1$, and $dt = 0.1$ and with $a = -1/2$ and $x_0 = 3$, and steps= 100

[14]:
```
x0_ = 3.
a_ = -0.5
dt_ = 0.1
ut_ = 1
steps_ = 100
plt.plot(x_trajectory(x0_, steps_, a_ , dt_, ut_));
```



1.0.5 Part c (1 point): Solve the equation using builtin ODE solver (Python's odeint) and check that the solution matches the above at the discretization points.

[20]:
```
# do not change this function
def odefun(t, x, a, u):
    return a * x + u
```

1.0.6 In the section below, use odeint to find the solution of the ode.

Hint: in this part, your main task is to revisit 'odeint' function in 'https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html',

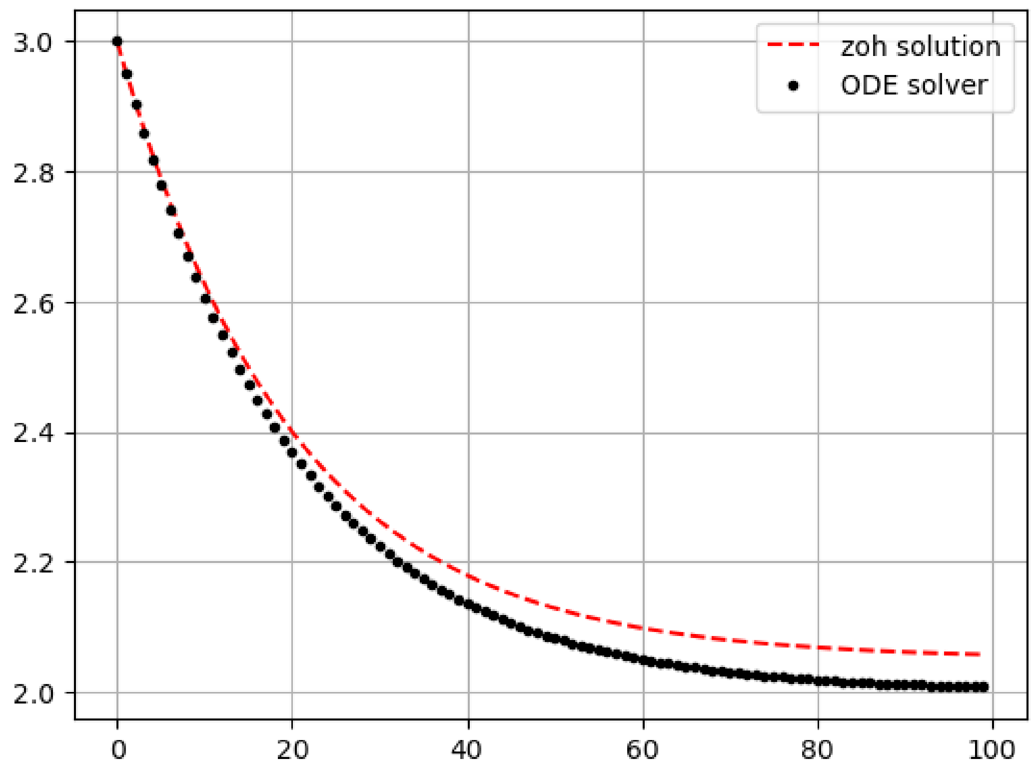
and realize what you need to consider for 'tfirst' and 'args' input parameters of 'odeint' based on function 'odefun' presented above.

```python
[23]: def builtin_ODE_solver(x0, steps, dt, ode_function, a, u):
          T = np.arange(0, steps*dt, dt)
          # x_builtin_ODE_solver = ?
          # YOUR CODE HERE
          args = (a, u)

          x_builtin_ODE_solver = odeint(ode_function, x0, T, args=args, tfirst=True).
       ↪flatten()

          return x_builtin_ODE_solver
```

```python
[24]: """Check the result for the given inputs"""
      assert np.allclose(builtin_ODE_solver(3., 100, 0.1, odefun, -0.5 , 1.)[0], 3.0,␣
       ↪rtol=1e-03, atol=1e-04)
      assert (builtin_ODE_solver(3., 100, 0.1, odefun, -0.5 , 1.)[2] -␣
       ↪builtin_ODE_solver(3., 100, 0.1, odefun, -0.5 , 1.)[0] < 0)
      assert (builtin_ODE_solver(3., 100, 0.1, odefun, -0.5 , 1.)[-1] -␣
       ↪builtin_ODE_solver(3., 100, 0.1, odefun, -0.5 , 1.)[-5] < 0)
```

Run the code below to see that the solution matches the above at the discretization points.

```python
[25]: plt.plot(x_trajectory(x0_, steps_, a_ , dt_, ut_), 'r--',  label = 'zoh␣
       ↪solution')
      plt.plot(builtin_ODE_solver(x0_, steps_, dt_, odefun, a_, ut_), 'k.',  label =␣
       ↪'ODE solver')
      plt.grid()
      plt.legend();
```



zoh solution
ODE solver
3.0
2.8
2.6
2.4
2.2
2.0
0
20
40
60
80
100

[ ]: