1 Task 1

My name is *Anton Brisilin*, and my email is *a.brisilin@innopolis.university*, so my generated variant is **F**.

2 Task 2

First, lets obtain our new linearized system (with new coefficients).
From the previous assignment I obtained such nonlinear dynamics equation for the pendulum on a cart system:

$$\begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \frac{-mlsin(\theta)\dot{\theta}^2+mgsin(\theta)cos(\theta)}{(M+m)-mcos^2(\theta)} \\ \frac{(M+m)gsin(\theta)-mlcos(\theta)sin(\theta)\dot{\theta}^2}{l((M+m)-mcos^2(\theta))} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{(M+m)-mcos^2(\theta)} \\ \frac{cos(\theta)}{(M+m)-mcos^2(\theta)} \end{bmatrix} u$$

Just like in the previous assignment, I should linearize it by computing derivatives. I will not pay too much attention to it, the script for computing derivatives is located as `Task2/diff.m`

$$\begin{bmatrix} \delta\dot{x} \\ \delta\dot{\theta} \\ \delta\ddot{x} \\ \delta\ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0.7796 & 0 & 0 \\ 0 & 30.256 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta x \\ \delta\theta \\ \delta\dot{x} \\ \delta\dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0.066 \\ 0.066 \end{bmatrix} u \tag{1}$$

2.1 Prove that it is possible to design state observer of the linearized system

In order to make possible existence of state observer, the observed system should be fully observable.
To prove observability of the obtained system, I will use Principle of Duality. It says, that if we have system

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$$

then we can determine whether it is observable or not by considering controllability of its dual system, that is:

$$\begin{cases} \dot{z} = A^*z + C^*k \\ t = B^*z \end{cases}$$

where A^*, B^*, C^*, D^* are conjugate transposes of matrices A, B, C and D respectively. In our case A and B are same as in (1). Matrix C looks like this:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Thus, our conjugate transpose matrices are

$$\begin{cases} A^* = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7796 & 30.256 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \\ B^* = \begin{bmatrix} 0 & 0 & 0.066 & 0.066 \end{bmatrix} \\ C^* = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \end{cases}$$

According to Ogata, "necessary and sufficient condition for complete observability [of my first system] is that the rank of the $n \times nm$ matrix

$$[C^* | A^* C^* | \dots | (A^*)^{(n-1)} C^*]$$

be n". Fortunately, Matlab has built-in function to check for controllability building controllability matrix. By checking its rank (that is 4 - the number of dimensions of A) with script in `Task2/contr.m`, we cane make conclusion, that it is observable.

2.2 For open-loop observation: is the error dynamics stable?

Open-loop observation: the idea is that if we know exact model, and initial conditions, we can estimate state of the system even not considering input and output.
Let's look at our linearized system, and check it for stability by looking at eigenvalues of A.

$$eig(A) = \begin{bmatrix} 0. & 0. & 5.50054543 & -5.50054543 \end{bmatrix}$$

As we can see, system is not stable, hence error dynamics (2) is not stable,

$$\dot{\epsilon} = A\epsilon, \tag{2}$$

too. We can not say that error eventually will go to zero, because it will not, if observer used in the model considers initial conditions different from actual ones.

2.3 Design Luenberger observer

Luenberger observer is also called the Predictor-Corrector observer. Its inputs are system input u and $\tilde{y} = y - \hat{y}$. It is described by the following formula: $\dot{\hat{x}} = A\hat{x} + Bu + L(y - \hat{y})$.
To design a Luenberger observer means to find such matrix L that will minimize the output error and make it stable.

$$\dot{\epsilon} = (A - LC)\epsilon$$

Hence, we need to find such L (that is called, according to [1] "user selectable gain matrix") that will make $eig(A - LC) < 0$.
Our A is 4x4, C is 2x4. Then, the L should be 4x2.
First, I will use pole-placement for dual system to design the observer gain matrix. I have chosen eigenvalues with negative real parts, just to make sure that my error dynamics will become stable eventually. My desired poles are:

, which leads to

$$L = \begin{bmatrix} 5 & 1 \\ 1 & 5 \\ 5.5 & 3.2796 \\ 2.5 & 35.756 \end{bmatrix}$$

Computations of L are located at the `Task2/poleplace.py`.
As second approach i will use LQR method. Just like in previous case, I will consider dual system. I used `control.lqr()` python function, because I did not succeed in implementing LQR for previous homework.
I set

$$Q = 100 * I_4, R = I_2$$

because I actually don't care about the 'actuator effort' in this case, because it is just simulation of a real system. Thus I found new matrix L, that is

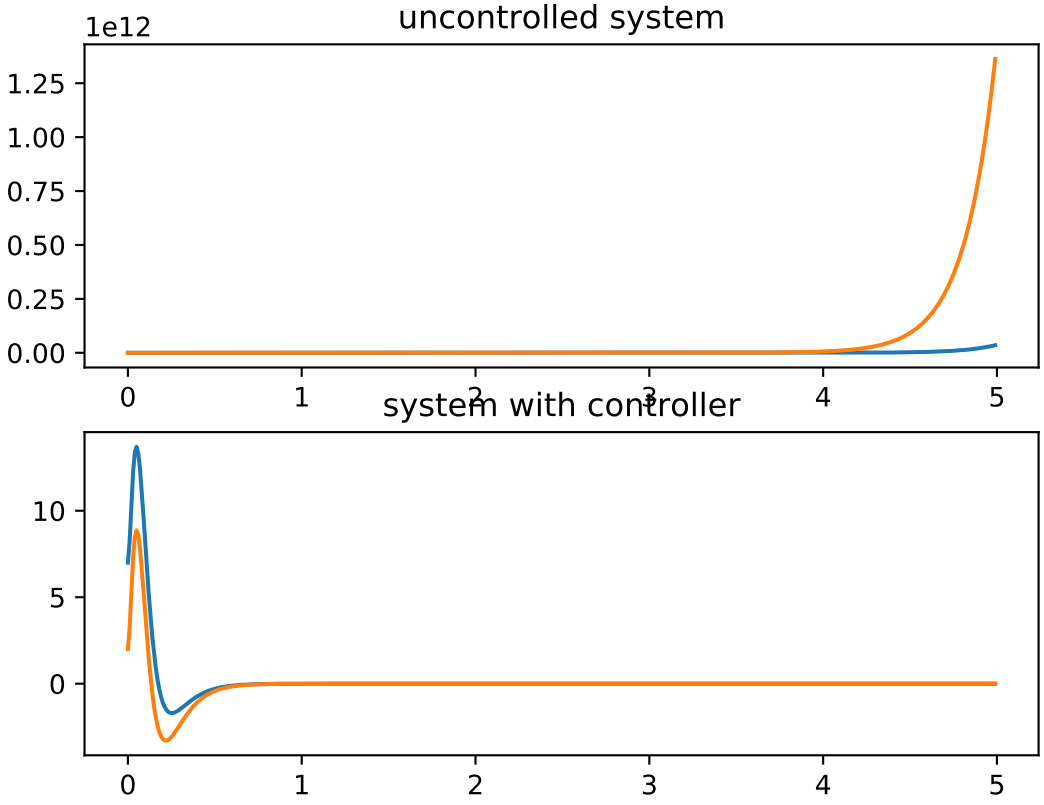
$$L = \begin{bmatrix} 10.95558675 & 0.07458349 \\ 0.07458349 & 14.97419382 \\ 10.01522191 & 1.3301508 \\ 0.60378264 & 62.1160217 \end{bmatrix}$$

Calculations of L with LQR can be found in `Task2/lqr.py`
The estimator model can be found in `Task2/estimator_model.py`.

It is needed to notice, that designing the estimator with pole-placement and LQR differs only in way how we find L, so the model is actually the same. On each step I integrating model of the actual system, to find the output that it produces, and feed this output to the estimator. The estimator adjusts its estimation to be closer to actual state of the system.

2.4 Design state-feedback controller

For simplicity I decided to design state feedback controller just like in the previous homework - with pole-placement method. The controller code can be found at `Task2/state_feedback_controller.py`. Here is its output of the system with controller:



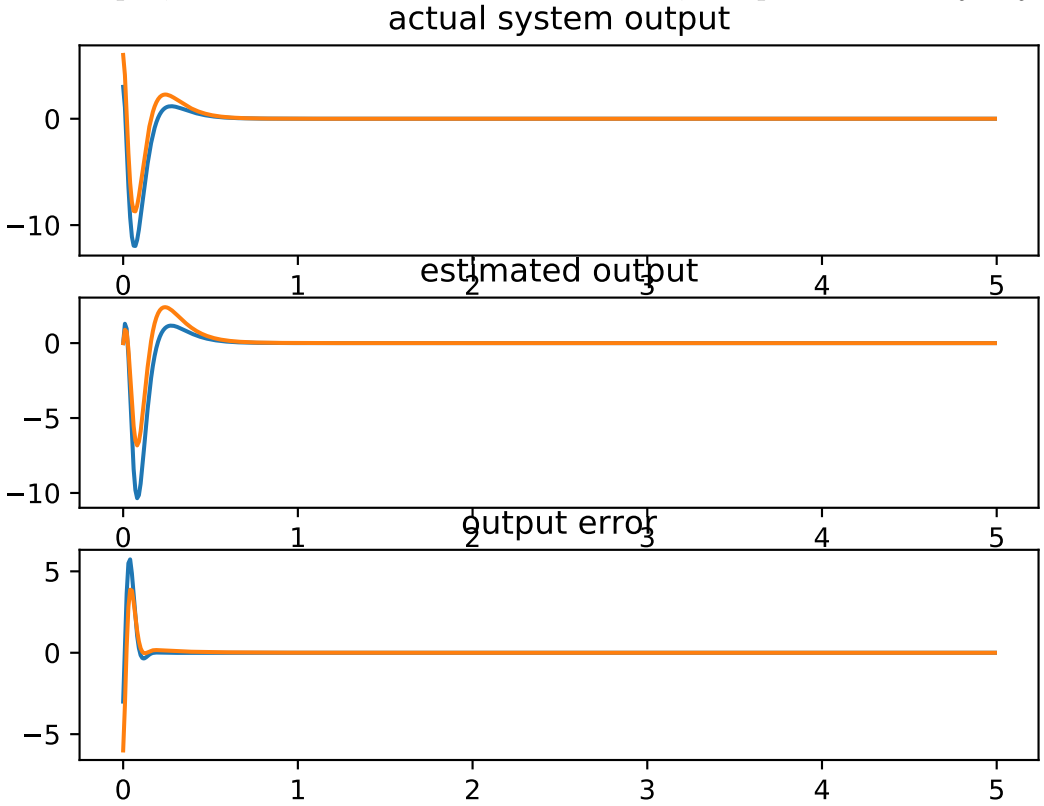
2.5 Simulate the system with observer and controller

Designing an observer to estimate system with controller is pretty the same as doing it without one. The main difference is that our dynamics matrix that we feed to LQR is not A, but $A - BK$. Everything other is pretty the same: we finding control gain K, creating controller based on it just like in

the previous section, then designing observer with new dynamics matrix, just like in section 2.3. So our algorithm is

1. Find gain K
2. Find gain L for $A = A - BK$ and $C = C$
3. Substitute K into the controller
4. Substitute L and K into the estimator
5. Simulate work of the systems

I used Python to accomplish the task, just like before. The code of solution is located at `Task2/obs_cont.py` Here is the plot of stabilized system output, estimated one and estimation error, computed as $\epsilon = y - \hat{y}$.



As it is seen from the plot, the system estimation error converges to zero eventually, and both estimation and actual system simulation are stable.

2.6 Add Additive White Gaussian Noise

My first attempt of adding White Gaussian noise was unsuccessful, because i tried to keep using `scipy.integrate.odeint()`, that does not support

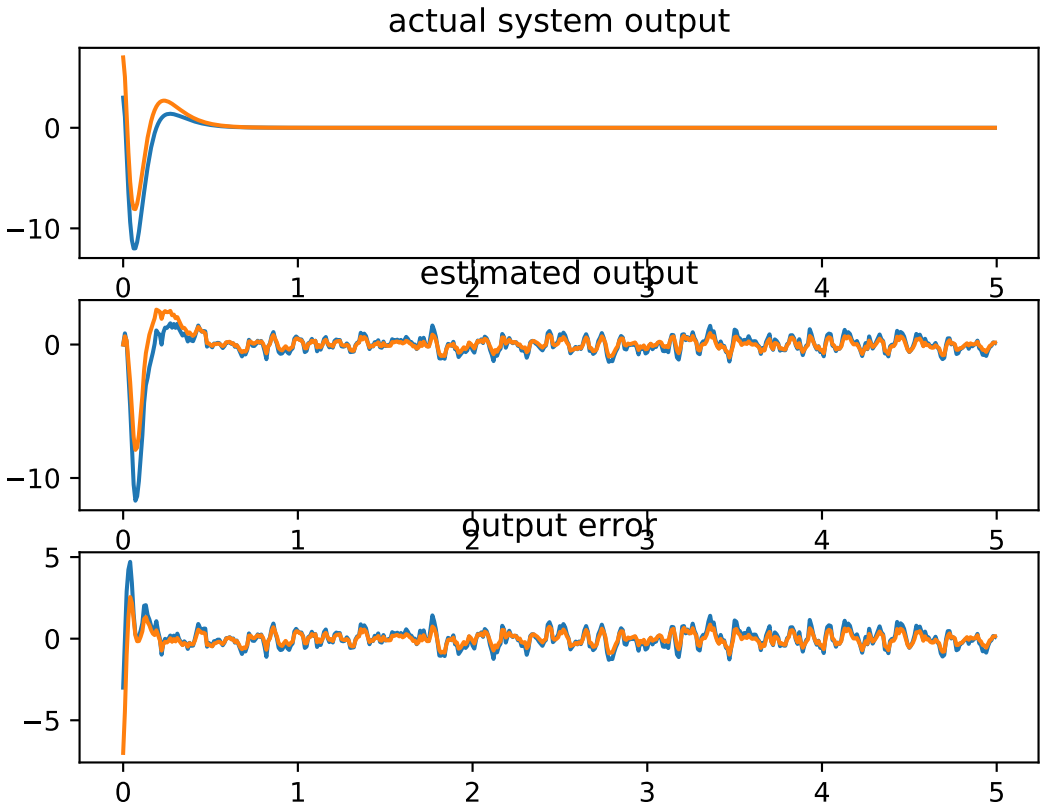
integration of stochastic differential equations, i.e. differential equation with stochastic term - that is in my case the noise. I decided to use simple Euler numerical solver to integrate the system estimation:

```
def euler(f,y0,a=START,b=STOP,h=STEP,order=4):
    count = int((b-a)/h)
    ts=np.zeros(count)
    ys=np.zeros((count,order))
    t,y = a,y0
    ind = 0
    while t < b and ind < int((b-a)/h):
        ts[ind] = t
        ys[ind] = y
        t += h
        y += h * f(y,t, count=ind)
        ind += 1
    return (ts,ys)
```

As a noise model I took random variables generated by numpy with normal distribution with $\mu = 0$ and $\sigma^2 = 1$. Estimator model with noise:

```
noise = np.random.normal(0,1,(COUNT,2))

def estimated_sys(x_hat, t, count=0):
    x_hat_r = x_hat.reshape(4, 1)

    temp_space = [0, t]

    y = C.dot(odeint(actual_sys, initial, temp_space)[-1, :])
    y += noise[count]
    y = y.reshape(2, 1)

    y_hat = C.dot(x_hat)
    y_hat = y_hat.reshape(2, 1)

    return ((A-B.dot(K)).dot(x_hat).reshape((4,1)) + L.dot(y
   - y_hat)).reshape((4,))
```

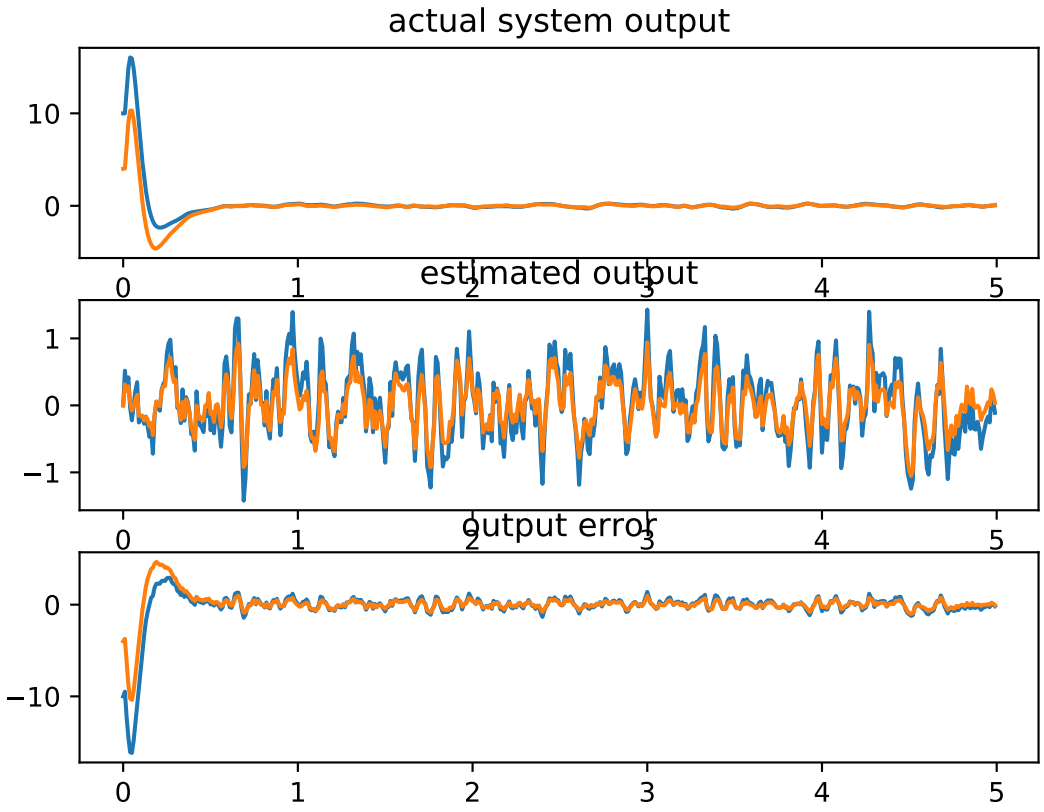
And here is the plot of estimator output with noise.



From this it is visible, that output is highly affects estimation accuracy. Full code of the solution can be found in `Task2/obs_cont_noise.py`.

2.7 Add noise to dynamics

Addition of noise to the dynamics of the system leads to very wrong estimation results. It can easily be seen from the plot, the dynamics of the estimator does not reflect the actual system dynamics. However, both of them are seems to be stable.



3 Used software

- Python 3.8.1
- Matlab R2018b 9.5.0
- draw.io

All software was run under Manjaro Linux with 5.4.18-rt kernel

4 References

- MIT OpenCourseWare