Homework 6

Nathan Rose

April 22, 2022

1. Consider the LTI SISO system in Problem 2 of homework 5:

$$A = \begin{bmatrix} -2 & -2 & 0 \\ 0 & 0 & 1 \\ 0 & -3 & -4 \end{bmatrix} B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} C = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{1}$$

   Recall that, in that homework, you designed a state feedback control law $\bar{u}(t) = -K\bar{x}(t)$ such that the eigenvalues of the closed-loop system because $\lambda_{d_1} = \lambda_{d_2} = -3$ and $\lambda_{d_3} = -4$. ie you have already obtained the control gain matrix K

   (a) Design a full-order observer with observer poles $\mu_{d_1} = -6, \mu_{d_2} = -7, \mu_{d_3} = -8$ using either methodd 1 in slide 121 or method 2 on slide 122 and verify the eigenvalues of $(A - LC)$.
   The first goal will be to reduce the multi-output model to a single output model

$$v = \begin{bmatrix} 1.647 & 1.299 \end{bmatrix} \tag{2}$$

   the desired eigenvalues for the observer are:

$$\lambda = -6, -7, -8 \tag{3}$$

   Desired Characteristic equation

$$\Delta_d(\lambda) = \lambda^3 + 21\lambda^2 + 146\lambda + 336 \tag{4}$$

   Desired Characteristic equation

$$\bar{\alpha} = \begin{bmatrix} 336 \\ 146 \\ 21 \end{bmatrix} \tag{5}$$

   Desired Characteristic equation

$$\Delta_d(\lambda) = \text{PurePoly}\left(\lambda^3 + 6\lambda^2 + 11\lambda + 6, \lambda, domain = \mathbb{Z}\right) \tag{6}$$

Desired Characteristic equation

$$\alpha = \begin{bmatrix} 6 \\ 11 \\ 6 \end{bmatrix} \tag{7}$$

$\bar{l}$

$$\bar{l} = \bar{\alpha} - \alpha = \tag{8}$$

$$\bar{l} = \begin{bmatrix} 336 \\ 146 \\ 21 \end{bmatrix} - \begin{bmatrix} 6 \\ 11 \\ 6 \end{bmatrix} = \tag{9}$$

$$\bar{l} = \begin{bmatrix} 330 \\ 135 \\ 15 \end{bmatrix} \tag{10}$$

Calculate P

$$Q = P.T = \begin{bmatrix} (A.T)^2C.T & (A.T)C.T & C.T \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ \alpha_1 & 1 & 0 \\ \alpha_2 & \alpha_1 & 1 \end{bmatrix} \tag{11}$$

$$Q = \begin{bmatrix} 6.587 & -3.294 & 1.647 \\ 22.453 & -8.234 & 1.299 \\ 12.921 & -5.289 & 1.647 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 6 & 1 & 0 \\ 11 & 6 & 1 \end{bmatrix} \tag{12}$$

$$Q = \begin{bmatrix} 4.94 & 6.587 & 1.647 \\ -12.667 & -0.443 & 1.299 \\ -0.697 & 4.592 & 1.647 \end{bmatrix} \tag{13}$$

$$P = Q.T = \begin{bmatrix} 4.94 & -12.667 & -0.697 \\ 6.587 & -0.443 & 4.592 \\ 1.647 & 1.299 & 1.647 \end{bmatrix} \tag{14}$$

$$l = P^{-1}\bar{l} = \begin{bmatrix} -187.492 \\ -114.975 \\ 287.261 \end{bmatrix} \tag{15}$$

$$L = lv = \begin{bmatrix} -308.762 & -243.466 \\ -189.34 & -149.299 \\ 473.061 & 373.02 \end{bmatrix} \tag{16}$$

To verify the equations use: eig$(A - LC)$

$$|\lambda - (A - LC)| = \text{PurePoly}\left(1.0\lambda^3 + 21.0\lambda^2 + 146.0\lambda + 336.0, \lambda, domain = \mathbb{R}\right) \tag{17}$$

i. $\lambda = -6.0$

ii. $\lambda = -7.0$

iii. $\lambda = -8.0$

(b) We know that using seperation principle we can design an observer-basesd feedback control. In eqs 69, 70 of the slides, let the control input be $\bar{u}(t) = -K\hat{x}(t)$ with the control gain matrix k obtained obtained in either part a or part b of problem 2 of homework 5, and let the observer gain matrix L be that obtained in part (a) above. Simulate and plot the state response of the closed-loop system assuming that the initial conditions of the states and estimated state are $\bar{x}_0 = \begin{bmatrix} 1 \\ -1 \\ 2.3 \end{bmatrix}$ and $\hat{x}_0 = \begin{bmatrix} 1.7 \\ -0.3 \\ 2.5 \end{bmatrix}$ respectively. In your plots ensure that the transient response and system convergence are depicted clearly.

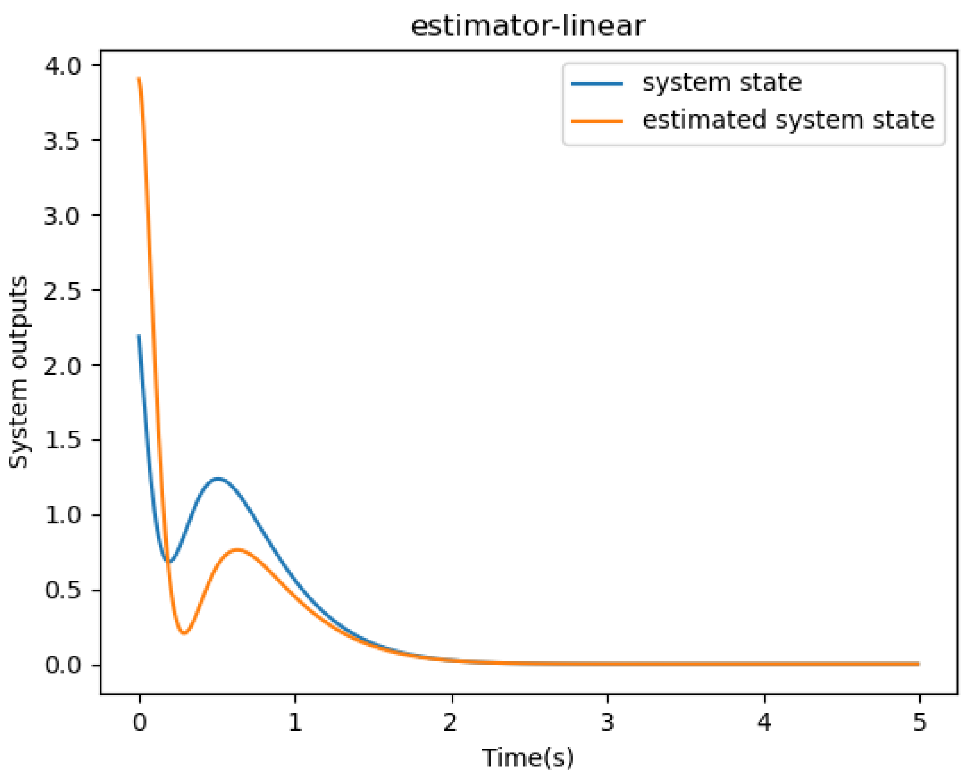

Figure 1: Estimator in NonLinear performance

(c) Design a reduce order observer with observer pole $\mu_{d_1} = -8$. No need for computer simulation

$$F = -8 \tag{18}$$

$$G = 1 \tag{19}$$

$$TA - FT = GC = T\begin{bmatrix} -2 & -2 & 0 \\ 0 & 0 & 1 \\ 0 & -3 & -4 \end{bmatrix} - 8T = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{20}$$

some very manual brute forcing later

$$T = \begin{bmatrix} 0.1667 & -0.5 & -0.14583333 \\ 0 & 0.1143 & -0.0285 \end{bmatrix} \tag{21}$$

2. use Eqs 77-81 to verify equation 82 of the slides Equation 82:

$$\dot{\bar{e}} = \dot{\bar{z}}_2(t) - \dot{\hat{z}}_2(t) = (\tilde{A}_{22} - L\tilde{A}_{12})\bar{e}(t) \tag{22}$$

with initial conditions: $\bar{e}(0) = \bar{z}_2(0) - \hat{z}_2(0)$

Given:

$$\dot{\bar{z}}_2(t) = \tilde{A}_{22}\bar{z}_2(t) + \tilde{A}_{21}(\bar{y}(t) - D\bar{u}(t)) + \tilde{B}_2\bar{u}(t) \tag{23}$$

$$\hat{y}_r(t) = \tilde{A}_{12}\hat{z}_2(t) = \dot{\bar{y}}(t) - \tilde{A}_{11}(\bar{y}(t) - D\bar{u}(t)) - \tilde{B}_1\bar{u}(t) - D\dot{\bar{u}}(t) \tag{24}$$

$$\dot{\hat{z}}_2(t) = \tilde{A}_{22}\hat{z}_2(t) + \tilde{A}_{21}(\bar{y}(t) - D\bar{u}(t)) + \tilde{B}_2\bar{u}(t) + L(\bar{y}_r(t) - \tilde{A}_{12}\bar{z}_2(t)) \tag{25}$$

$$\hat{z}_2(t) = \tilde{A}_{22}\hat{z}_2(t) + \tilde{A}_{21}(\bar{y}(t) - D\bar{u}(t)) + \tilde{B}_2\bar{u}(t) + L(\bar{y}_r(t) - \tilde{A}_{12}\bar{z}_2(t)) \tag{26}$$

$$\dot{\bar{z}}_2(t) - \dot{\hat{z}}_2(t) \tag{27}$$

$$(\tilde{A}_{21}\bar{z}_2(t)+\tilde{A}_{21}(\bar{y}(t)-D\bar{u}(t))+\tilde{B}_2\bar{u}(t))-(\tilde{A}_{22}\hat{z}_2(t)+\tilde{A}_{21}(\bar{y}(t)-D\bar{u}(t))+\tilde{B}_2\bar{u}(t)+L(\bar{y}_r(t)-\tilde{A}_{12}\hat{z}_2(t))) \tag{28}$$

$$\tilde{A}_{21}(\bar{z}_2(t) - \hat{z}_2(t)) - (L(\bar{y}_r(t) - \tilde{A}_{12}\hat{z}_2(t))) \tag{29}$$

$$\tilde{A}_{21}(\bar{z}_2(t) - \hat{z}_2(t)) - (L(\tilde{A}_{12}\bar{z}_2(t) - \tilde{A}_{12}\hat{z}_2(t))) \tag{30}$$

$$\tilde{A}_{21}(\bar{z}_2(t) - \hat{z}_2(t)) - (L\tilde{A}_{12}(\bar{z}_2(t) - \hat{z}_2(t))) \tag{31}$$

$$(\tilde{A}_{21} - L\tilde{A}_{21})(\bar{z}_2(t) - \hat{z}_2(t)) \tag{32}$$

$$(\tilde{A}_{21} - L\tilde{A}_{21})\bar{e}(t) \tag{33}$$

1 Help Recieved

This section is a thank you for people who caught issues or otherwised helped(collaboration is allowed)

1. List people here

A System Modeling Code

```
import sympy
from functools import cache
from sympy import cos, sin
from typing import Callable, Iterable, Tuple
from copy import deepcopy

def model_system(update: Callable[[sympy.Matrix, sympy.Matrix], sympy.Matrix],
                output: Callable[[sympy.Matrix, sympy.Matrix], sympy.Matrix],
                x_0: sympy.Matrix,
                inputs: Iterable[sympy.Matrix],
                *args,
                **dargs) -> Iterable[Tuple]:
    """
    A generic model for any system can work with linear and nonlinear
    systems. update functions returns the next state, and output outputs
    the output matrix. Both of these functions take in state, system input,
    then *args.
    This function will return list of outputs
```

```python
    This is an underlying function see other models for examples
    """
    outputs = []
    x = deepcopy(x_0)
    inputs = deepcopy(inputs)
    for r in inputs:
        x = update(x, r, *args, **dargs)
        outputs.append(output(x, r, *args, **dargs))
    return outputs

def observer_update_wrapper(x: sympy.Matrix,
                            r: sympy.Matrix,
                            system_update: Callable[[sympy.Matrix, sympy.Matrix], sympy.Mat
                            observer_update: Callable[[sympy.Matrix, sympy.Matrix], sympy.M
                            system_output: Callable[[sympy.Matrix, sympy.Matrix], sympy.Mat
                            k: sympy.Matrix=None,
                            *args, **dargs) -> Tuple[sympy.Matrix, sympy.Matrix]:
    system_x, est_x = x
    new_system_x = system_update(system_x, r, k=k, *args, **dargs)
    sys_y = system_output(new_system_x, r, *args, **dargs)
    new_est_x = observer_update(est_x, r, sys_y, k=k, *args, **dargs)
    if k is not None:
        u = (r - k*new_est_x)
        new_system_x = system_update(system_x, u, *args, **dargs)
    return new_system_x, new_est_x

def observer_update(x_est: sympy.Matrix,
                    r: sympy.Matrix,
                    y_sys: sympy.Matrix,
                    L: sympy.Matrix,
                    dt: float,
                    **dargs) -> None:
    # dargs["A"] = dargs["A"] - L*dargs["C"]
    # print(dargs["A"].eigenvals())
    new_est_x = linear_update(x_est, r, dt=dt, **dargs)
    y_est = linear_output(x_est, r, dt=dt, **dargs)
    new_est_x += L*(y_sys - y_est)*dt
    return new_est_x

def observer_output_wrapper(x: sympy.Matrix,
                            r: sympy.Matrix,
```

```python
                              *args, **dargs) -> Tuple[sympy.Matrix, sympy.Matrix]:
    """
    assuming the desired output is not the output itself but a condensed veriosn of the out
    """
    system_x, est_x = x
    condenser = sympy.Matrix([
        [1,1,1]
    ])
    system_out = (condenser * system_x)[0,0]
    observer_out = (condenser * est_x)[0,0]
    return [(condenser * system_x), (condenser * est_x)]


def non_linear_update(x: sympy.Matrix,
                      r: sympy.Matrix,
                      k: sympy.Matrix,
                      f: sympy.Matrix,
                      dt: float,
                      *args, **dargs) -> sympy.Matrix:
    """
    Non linear update equation to handle question 4
    @arg x is the previous state sys
    @arg r input before any feedback
    @arg k feedback matrix
    @arg f output equation given the state variables
    @returns next state
    """
    if k is not None:
        u = r - (k*x)
    else:
        u = r
    y = (r'y', x[0, 0])
    theta_1 = (r'\theta_1', x[1, 0])
    theta_2 = (r'\theta_2', x[2, 0])
    dot_y = (r'\doty', x[3, 0])
    dot_theta_1 = (r'\dot\theta_1', x[4, 0])
    dot_theta_2 = (r'\dot\theta_2', x[5, 0])
    u = ('u', u[0,0])
    dot_x = f.subs([
        theta_1, theta_2, y,
        dot_theta_1, dot_theta_2, dot_y,
```

```
        u
    ])
    new_x = sympy.N(x + (dt*dot_x))
    return new_x

def linear_update(x: sympy.Matrix,
                  r: sympy.Matrix,
                  A: sympy.Matrix,
                  B: sympy.Matrix,
                  dt: float,
                  k: sympy.Matrix=None,
                  *args, **dargs) -> sympy.Matrix:
    """
    Linear output equation
    """
    if k is not None:
        u = r - (k*x)
    else:
        u = r
    dx = A*x + B*u
    return x + (dx*dt)

def linear_output(x: sympy.Matrix,
                  r: sympy.Matrix,
                  C: sympy.Matrix,
                  D: sympy.Matrix,
                  *args, **dargs) -> sympy.Matrix:
    """
    Linear output equation
    """
    return C*x + D*r


def linear_output_with_observer(x: sympy.Matrix,
                                r: sympy.Matrix,
                                A: sympy.Matrix,
                                B: sympy.Matrix,
                                C: sympy.Matrix,
                                D: sympy.Matrix,
                                *args, **dargs) -> sympy.Matrix:
    """
```

```
    Linear output equation with observer,
    performs the observer update as well(linear)
    """
    return C*x + D*r
```

There is also some work on github(link below) in a jupyter notebook

Disclaimer: This is just a few relevant fragments of the source code, as the entire code is a complicated system that takes these fragments and automatically renders them into the final pdf. However all of this is available online on github(its latex + python)