Homework 3

Nathan Rose

February 23, 2022

1. In the LTI system described for $\dot{\bar{x}}(t) = A\bar{x}$ with $A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -3 & -3 \end{bmatrix}$

   (a) Obtain all eigenvalues and eigenvectors of A
   Note I cannot seem to get the right answer for this question, so I show my work for how I would calculate the answer which produces the wrong answer and then use jordan-form function to get V. This can be shown in A

$$|\lambda I - A| = 0 \tag{1}$$

$$det(\begin{bmatrix} \lambda & -1 & 0 \\ 0 & \lambda & -1 \\ 1 & 3 & \lambda + 3 \end{bmatrix}) = \lambda^3 + 3\lambda^2 + 3\lambda + 1 = 0 \tag{2}$$

$$\lambda = -1, -1, -1 \tag{3}$$

   (b) Use the eigenvectors in part 1a to obtain the m odal matrix V and Jordan Form J
   Solving for eigenvalue(s) of -1, with a multiplicity of 3

$$(A - \lambda i)^3 x_0 = 0 \tag{4}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = 0 \tag{5}$$

$$x_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{6}$$

$$(A - \lambda i)^2 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = x_1 \tag{7}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ -1 & -2 & -1 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = x_1 \tag{8}$$

$$x_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \tag{9}$$

$$(A - \lambda i)^1 \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = x_2 \tag{10}$$

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ -1 & -3 & -2 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = x_2 \tag{11}$$

$$x_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{12}$$

$$V = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 0 \\ 1 & -1 & 0 \end{bmatrix} \tag{13}$$

$$J = V^{-1}AV = \begin{bmatrix} 0 & -1 & 0 \\ 0 & -1 & -1 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -3 & -3 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 0 \\ 1 & -1 & 0 \end{bmatrix} \tag{14}$$

$$J = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix} \tag{15}$$

2. In each case below discuss BIBS stability of the LTI system $\dot{\bar{x}}(t) = A\bar{x}(t)$:

   (a) $A = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$

   This system is stable as both of its eigenvalues are less 0

(b) $A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -3 \end{bmatrix}$

This system is semi-stable as two of the three are less than 0, and one is 0

(c) $A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -3 \end{bmatrix}$

This system is not stable an eigenvalue is greater than 0

(d) $A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -3 \end{bmatrix}$

This system has two semi-simple eigenvalues that are 0, meaning this is not stable

(e) $A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -3 \end{bmatrix}$

this system has three eigenvaleus that are all less than or equal to 0

3. The linearized equatinos of motion of a pendulum can be written in the form of

$$\dot{x}_1 = x_2 \tag{16}$$

$$\dot{x}_2 = -ax_1 - cx_2 \tag{17}$$

where $a > 0$ is a constant parameter of the system and $c > 0$ is the torsional friction coefficient

(a) Study BIBS stability of the system

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -a & -c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{18}$$

This allows you to calculate the eigenvalues with:

$$\Delta = |\lambda I - A| = 0 \tag{19}$$

$$0 = \begin{vmatrix} \lambda & -1 \\ a & \lambda + c \end{vmatrix} = \lambda^2 + c\lambda + a \tag{20}$$

$$\lambda = \frac{-c \pm \sqrt{c^2 - 4a}}{2} \tag{21}$$

the system will be stable if the eigenvalues described by this equation are less then 0(or equal to 0 if the eigenvalues produce linearly independent eigenvectors)

(b) Consider the quadratic Lyapunov functino $V = \bar{x}^T P \bar{x}$ with $P = \begin{bmatrix} \frac{a}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$. What can be said about the stability of the system based on this choice of the Lyapunov function?

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} \frac{a}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} \frac{ax_1}{2} \\ \frac{x_2}{2} \end{bmatrix} = \begin{bmatrix} \frac{a(x_1)^2}{2} \\ \frac{(x_2)^2}{2} \end{bmatrix} \tag{22}$$

V is positive definite if a is positive? TODO: check

(c) What can be said about the stability of the system based on the analysis of b) and c)?
The system will be stable if a is positive

(d) Study BIBS stability of the system when $c = 0$

$$\lambda = \pm\sqrt{a}j \tag{23}$$

if the eigenvectors are linearly independent, then the system is lyapunov stable

4. For the transfer function matrix

$$H(s) = \begin{bmatrix} \frac{s}{s-2} & 0 \\ \frac{2}{s-2} & 1 \end{bmatrix} \tag{24}$$

(a) Obtain the controllable canoncical form

$$\frac{1}{s+2} \begin{bmatrix} s & 0 \\ 2 & s+2 \end{bmatrix} \tag{25}$$

$$\dot{x} = \begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u \tag{26}$$

$$N_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{27}$$

$$N_1 = \begin{bmatrix} 0 & 0 \\ 2 & 2 \end{bmatrix} \tag{28}$$

$$y = (\begin{bmatrix} 0 & 0 \\ 2 & 2 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} 2)x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u \tag{29}$$

$$y = \begin{bmatrix} -2 & 0 \\ 2 & 0 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u \tag{30}$$

(b) obtain the observable canonical form

$$\frac{1}{s+2}\begin{bmatrix} s & 0 \\ 2 & s+2 \end{bmatrix} \tag{31}$$

$$N_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{32}$$

$$N_1 = \begin{bmatrix} 0 & 0 \\ 2 & 2 \end{bmatrix} \tag{33}$$

$$\dot{x} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} x + \left(\begin{bmatrix} 0 & 0 \\ 2 & 2 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \right) u \tag{34}$$

$$\dot{x} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} x + \begin{bmatrix} -2 & 0 \\ 2 & 0 \end{bmatrix} u \tag{35}$$

$$y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{36}$$

(c) show that the realization in (a) and (b) are dual as per

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^T = \begin{bmatrix} A^T & B^T \\ C^T & D^T \end{bmatrix} \tag{37}$$

it can be seen that this is not the case for this example as $A_c \neq {A_o}^T$

1 Help Recieved

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

1. Andrew Stradz general resource and checking my work

A Eigen code

```
import numpy as np
import sympy
from LatexTemplater.TemplateCore import TemplateCore
from sympy import Matrix, eye, latex, simplify, roots, zeros
from typing import Iterable, Tuple

def eigenvalues(A: Matrix, show_work: bool=True) -> Tuple[str, Iterable[complex]]:
    """
    This function provides a method for calculating the eigenvalues of a matrix A
    it return a tuple of the string with work and a list of all the eigenvalues
    """
    #cannot use lambda as variable name as that is a key word for an anonymous function
    work = ""
    if show_work:
        work += r"\begin{equation}" + '\n'
        work += r"  \vert \lambda I - A \vert = 0" + '\n'
        work += r"\end{equation}" + '\n'
    l = sympy.symbols(r'\lambda')
    delta_l = simplify((eye(A.shape[0])*l) - A)
    det = simplify(delta_l.det())
    if show_work:
        work += r"\begin{equation}" + '\n'
        work += f"  det({latex(delta_l)}) = {latex(det)} = 0" + '\n'
        work += r"\end{equation}" + '\n'
    root_vals = roots(det)
    zeros = []
    for root, multiplicity in root_vals.items():
        zeros += [root for i in range(multiplicity)]
    inst = TemplateCore.instance() # this is just used for simple formatting
    if show_work:
        work += r"\begin{equation}" + '\n'
        work += r"  \lambda = " + f"{inst.filter('complexCsv', zeros)}\n"
        work += r"\end{equation}" + '\n'
    return work, zeros

def eigenvectors(A: Matrix, show_eigval_work: bool=True) -> Tuple[str, Iterable[complex]]:
    """
    This function provides a method for calculating the eigenvectors of a matrix,
    if show_eigval_work is set will include the work for finding the eigenvalues
```

```
    """
    work = ""
    eig_work, eig_vals = eigenvalues(A, show_eigval_work)
    if show_eigval_work:
        work += eig_work
    eig_multiplicity = {}
    for eig in eig_vals:
        if eig in eig_multiplicity:
            eig_multiplicity[eig] += 1
        else:
            eig_multiplicity[eig] = 1
    eig_vectors = []
    for eig, mult in eig_multiplicity.items():
        vector_work, vectors = _handle_multiplicity_eig(A, eig, mult)
        eig_vectors = list(vectors) + eig_vectors
        work += vector_work
    P, J = A.jordan_form()
    # cheating code
    eig_vectors = [P.col(i) for i in range(A.shape[0])]
    return work, eig_vectors

def _handle_multiplicity_eig(A: Matrix, eig_val: complex, multiplicity: int) -> Tuple[str,
    """
    This function handles one group of eigenvalues with the same values
    returns the work and the
    """
    vector_symbols = ", ".join([f'x_{i}' for i in range(A.shape[0])])
    symbols = list(sympy.symbols(vector_symbols))
    eig_vectors = []
    a_minus_lambda_i = simplify(A - (eye(A.shape[0])*eig_val))
    work = f"Solving for eigenvalue(s) of {eig_val}, with a multiplicity of {multiplicity}"
    print(work)
    print(f"A = {A}")
    print(f"A-li = {a_minus_lambda_i}")
    for i in range(multiplicity):
        power = multiplicity-i
        left = a_minus_lambda_i ** power
        print(f"{power} - {left}")
        eig_vec = Matrix([[x_i] for x_i in symbols])
        if np.all(np.matrix(left) == np.matrix(zeros(*left.shape))):
            work += r"\begin{equation}"
```

```
        work += r"  (A - \lambda i)^" + f"{power}x_{i} = 0"
        work += r"\end{equation}"
        work += r"\begin{equation}"
        work += f"  {latex(left)}{latex(eig_vec)} = 0"
        work += r"\end{equation}"
        result_vec = Matrix([[0] for _ in range(left.shape[0])])
        result_vec[0,0] = 1
        eig_vectors.append(result_vec)
        work += r"\begin{equation}"
        work += f"  x_{i} = {latex(result_vec)}"
        work += r"\end{equation}"
    elif i == 0:
        work += r"\begin{equation}"
        work += f"  {latex(left)}{latex(eig_vec)} = 0"
        work += r"\end{equation}"
        work += r"\begin{equation}"
        work += f"  {latex(left)}{latex(eig_vec)} = 0"
        work += r"\end{equation}"
        work += r"\begin{equation}"
        work += f"  {latex(left)}{latex(eig_vec)} = 0"
        work += r"\end{equation}"
        result = sympy.solve(left*eig_vec, symbols, particular=True)
        result_vec = sympy.zeros(len(symbols), 1)
        print(left*eig_vec)
        print(symbols)
        print(result)
        for j, x_i in enumerate(symbols):
            if x_i in result:
                result_vec[j, 0] = result[x_i]
        eig_vectors.append(result_vec)
        work += r"\begin{equation}"
        work += f"  x_{i} = {latex(result_vec)}"
        work += r"\end{equation}"
    else:
        previous_eig_vector = eig_vectors[-1]
        work += r"\begin{equation}"
        work += r"  (A - \lambda i)^" + f"{power}{latex(previous_eig_vector)} = x_{i}"
        work += r"\end{equation}"
        work += r"\begin{equation}"
        work += f"{latex(left)}{latex(previous_eig_vector)} = x_{i}"
        work += r"\end{equation}"
```

```
            result = sympy.solve(left*previous_eig_vector - eig_vec, symbols, particular=Tr
            result_vec = sympy.zeros(len(symbols), 1)
            for j, x_i in enumerate(symbols):
                result_vec[j, 0] = result[x_i]
            eig_vectors.append(result_vec)
            work += r"\begin{equation}"
            work += f"  x_{i} = {latex(result_vec)}"
            work += r"\end{equation}"
    inst = TemplateCore.instance()
    return (work, reversed(eig_vectors))
def _handle_multiplicity_eig_two(A: Matrix, eig_val: complex, multiplicity: int) -> Tuple[s
    """
    This function handles one group of eigenvalues with the same values
    returns the work and the
    """
    vector_symbols = ", ".join([f'x_{i}' for i in range(A.shape[0])])
    symbols = list(sympy.symbols(vector_symbols))
    eig_vectors = []
    a_minus_lambda_i = simplify(A - (eye(A.shape[0])*eig_val))
    work = f"Solving for eigenvalue(s) of {eig_val}, with a multiplicity of {multiplicity}"
    print(work)
    print(f"A = {A}")
    print(f"A-li = {a_minus_lambda_i}")
    for i in range(multiplicity):
        power = i+1
        left = a_minus_lambda_i ** power
        print(f"{power} - {left}")
        eig_vec = Matrix([[x_i] for x_i in symbols])
        if np.all(np.matrix(left) == np.matrix(zeros(*left.shape))):
            work += r"\begin{equation}"
            work += r"  (A - \lambda i)^" + f"{power}x_{i} = 0"
            work += r"\end{equation}"
            work += r"\begin{equation}"
            work += f"  {latex(left)}{latex(eig_vec)} = 0"
            work += r"\end{equation}"
            result_vec = Matrix([[0] for _ in range(left.shape[0])])
            result_vec[0,0] = 1
            eig_vectors.append(result_vec)
            work += r"\begin{equation}"
            work += f"  x_{i} = {latex(result_vec)}"
            work += r"\end{equation}"
```

```
            raise ValueError("uhhh")
        elif i == 0:
            work += r"\begin{equation}"
            work += f"  {latex(left)}{latex(eig_vec)} = 0"
            work += r"\end{equation}"
            work += r"\begin{equation}"
            work += f"  {latex(left)}{latex(eig_vec)} = 0"
            work += r"\end{equation}"
            work += r"\begin{equation}"
            work += f"  {latex(left)}{latex(eig_vec)} = 0"
            work += r"\end{equation}"
            result = sympy.solve(left*eig_vec, symbols, particular=True)
            result_vec = sympy.zeros(len(symbols), 1)
            print(left*eig_vec)
            print(symbols)
            print(result)
            for j, x_i in enumerate(symbols):
                if x_i in result:
                    result_vec[j, 0] = result[x_i]
            eig_vectors.append(result_vec)
            work += r"\begin{equation}"
            work += f"  x_{i} = {latex(result_vec)}"
            work += r"\end{equation}"
        else:
            previous_eig_vector = eig_vectors[-1]
            work += r"\begin{equation}"
            work += r"  (A - \lambda i)^" + f"{power}{latex(previous_eig_vector)} = x_{i}"
            work += r"\end{equation}"
            work += r"\begin{equation}"
            work += f"{latex(left)}{latex(previous_eig_vector)} = x_{i}"
            work += r"\end{equation}"
            result = sympy.solve(left*previous_eig_vector - eig_vec, symbols, particular=Tr
            result_vec = sympy.zeros(len(symbols), 1)
            for j, x_i in enumerate(symbols):
                result_vec[j, 0] = result[x_i]
            eig_vectors.append(result_vec)
            work += r"\begin{equation}"
            work += f"  x_{i} = {latex(result_vec)}"
            work += r"\end{equation}"
    inst = TemplateCore.instance()
    return (work, reversed(eig_vectors))
```

```
def modal_matrix(A: Matrix, show_eigen_work: True) -> Matrix:
    work, vectors = eigenvectors(A, show_eigen_work)
    modal = Matrix([[vectors[row][column,0] for row in range(len(vectors))]
              for column in range(len(vectors))])
    work += "\n"
    work += r"\begin{equation}" + "\n"
    work += f"  V = {latex(modal)}"
    work += r"\end{equation}" + "\n"
    return work, modal
```

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)