UNIVERSIDADE FEDERAL DE ALAGOAS

Instituto de Computação – IC

Engenharia de Computação

Sistemas de Controle 2

ANÁLISE DE UM SISTEMA ELÉTRICO EM ESPAÇO DE ESTADOS

Aldemir Melo Rocha Filho

Sandoval da Silva Almeida Junior

Tayco Murilo Santos Rodrigues

Maceió – 2022

ÍNDICE

1. Introdução

Este relatório tem como objetivo apresentar os resultados da análise de um sistema elétrico de ordem três modelado em espaço de estados. A análise foi realizada através de implementações computacionais feitas em linguagem *Python*, com auxílio de módulos presentes na biblioteca *control* e *numpy*.

O link para acesso ao notebook com todos os algoritmos utilizados pode ser encontrado nos anexos deste documento.

2. Apresentação do sistema

O sistema elétrico de ordem três escolhido para ser realizada a análise e a modelagem em espaço de estados pode ser visualizado abaixo.

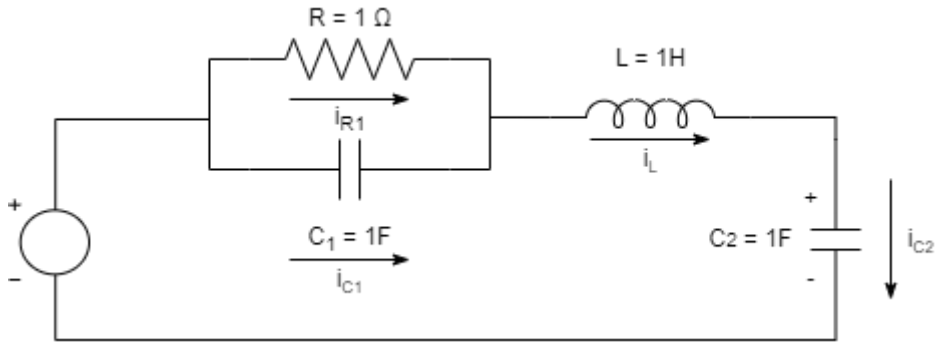

Imagem 2.1: Sistema Elétrico escolhido para análise

O circuito da **imagem 2.1** possui uma fonte de alimentação DC, um indutor com o valor de indutância $L = 1H$, um resistor com valor de resistência $R = 1\Omega$ e dois capacitores com valores de capacitância $C = 1F$.

3. Modelagem em Espaço de Estados

Se tratando de um sistema elétrico, podemos partir dos elementos armazenadores de energia para montar a matriz de estados. Dessa forma, vamos considerar a corrente que passa por L e as tensões em C_1 e C_2. Logo, temos que:

$$x = \begin{bmatrix} v_{C_1} \\ i_L \\ v_{C_2} \end{bmatrix}$$

Agora, precisamos escrever as equações diferencias para cada elemento armazenador de energia, levando em consideração o seguinte:

$$\frac{di_L}{dt} = \frac{1}{L}\, v_L \quad \text{e} \quad \frac{dv_c}{dt} = \frac{1}{C}\, i_c$$

Temos que para o nosso caso onde $L = 1H$ e $C_1 = C_2 = 1F$, as equações diferencias vão assumir o seguinte formato:

$$\frac{di_L}{dt} = v_L \; ; \; \frac{dv_{C_1}}{dt} = i_{C_1} \; ; \; \frac{dv_{C_2}}{dt} = i_{C_2}$$

Portanto temos que:

$$\dot{x} = \begin{bmatrix} i_{C_1} \\ v_L \\ i_{C_2} \end{bmatrix}$$

Agora, aplicando a $Lei\ de\ Kirchhoff$ temos as seguintes relações para as tensões:

$$1: v_{C_1} = v_R \rightarrow v_{C_1} = i_R \cdot R \rightarrow v_{C_1} = i_R;$$

$$2: v_e = i_R + v_L + v_{C_2} \; ;$$

$$3: v_e = v_{C_1} + v_L + v_{C_2} \; ;$$

E para as correntes:

$$1': i_{C_2} = i_L = (i_R + i_{C_1})$$

Assim, partindo de 3:

$$1'': v_L = -v_{C_1} - v_{C_2} + v_e$$

Usando o resultado anterior em conjunto de 2:

$$i_R = v_e - v_{C_2} - v_L = v_e - v_{C_2} - (-v_{C_1} - v_{C_2} + v_e) \rightarrow i_R = v_{C_1}$$

Usando o resultado anterior em conjunto de $1'$:

$$i_L = (i_R + i_{C_1}) \rightarrow i_L = (v_{C_1} + i_{C_1}) \rightarrow i_{C_1} = i_L - v_{C_1}$$

$$2'': i_{C_1} = i_L - v_{C_1}$$

Usando $1'$ temos:

$$3'': i_{C_2} = i_L$$

Dessa forma, considerando a entrada do sistema como v_e e partindo de $1''$, $2''$ e $3''$, podemos escrever as equações diferencias das variáveis de estados como o seguinte sistema:

$$S = \begin{cases} \dot{x}_1 = -1x_1 + 1x_2 + 0x_3 + 0u \\ \dot{x}_2 = -1x_1 + 0x_2 - 1x_3 + 1u \\ \dot{x}_3 = \quad 0x_1 + 1x_2 + 0x_3 + 0u \end{cases}$$

Agora, podemos fazer uso do sistema de equações diferenciais S para descrever o sistema em espaço de estados partindo da seguinte relação:

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

$\dot{x}$ e x já foram definidas anteriormente, as matrizes A e B podem ser escritas partindo dos coeficientes das equações em S e a matriz C é escrita em função do estado que se busca observar:

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

$$y = [\,0 \quad 0 \quad 1\,] \cdot x + [0] \cdot u$$

Por fim, é preciso fazer uso do método *control.matlab.ss()* que recebe quatro argumentos, que são as matrizes que representam o nosso sistema dentro do Espaço de Estados e retorna a modelagem da maneira que os outros métodos possam interpretá-la. O trecho de código utilizado para este fim pode ser observado abaixo.

```
from control import matlab

A = [[-1,1,0],[-1,0,-1],[0,1,0]]
B = [[0],[1],[0]]
C = [0,0,1]
D = [0]

system = matlab.ss(A,B,C,D)
```

Imagem 3.1: Trecho de código utilizado para gerar a modelagem em E.E.

4. Calculando a Função de Transferência Equivalente

Partindo da modelagem em Espaço de Estados realizada no tópico **3**, podemos fazer uso das matrizes *A,B,C e D* para encontrar a Função de Transferência do sistema. Para isso foi utilizado o método *control.matlab.ss2tf()* que recebe quatro argumentos, que são as matrizes que representam o nosso sistema dentro do Espaço de Estados e retorna a Função de Transferência correspondente do sistema. O trecho de código utilizado para este fim e a saída obtida podem ser observados abaixo.

```
system_tf = matlab.ss2tf(A,B,C,D)
print(system_tf)
```

Imagem 4.1: Trecho de código utilizado para gerar a Função de Transferência.

```
-3.331e-16 s^2 + s + 1
----------------------
  s^3 + s^2 + 2 s + 1
```

Imagem 4.2: Saída do método *control.matlab.ss2tf()*.

Levando em conta que o coeficiente $-3.331e-16$ presente no numerador é um valor muito próximo de 0 podemos interpretar a saída do código presente na **imagem 4.2** como a seguinte Função de Transferência:

$$G(s) = \frac{s^2 + 1}{s^3 + s^2 + 2s + 1}$$

5. Simulação do modelo

Inicialmente para que fosse possível realizar a simulação do modelo se fez necessário a montagem de quatro *arrays* que representam nosso vetor de tempo e nossos sinais de entrada:

$$Degrau = \begin{cases} u = k, t \geq 0 \\ 0, t < 0 \end{cases}$$

$$Rampa = r(t) = t$$

$$Parábola = p(t) = t^2$$

O trecho de código utilizado para este fim e as saídas obtidas podem ser observados abaixo:

```
import numpy as np

tempo = np.arange(0,60,0.01) #Vetor de tempo (60 segundos)
u_degrau = np.full(len(tempo),1) #Degrau unitario
u_rampa = np.full(len(tempo),tempo) #Rampa
u_parabola = np.full(len(tempo),tempo**2) #Parabola

print(u_degrau)
print(u_rampa)
print(u_parabola)
```

Imagem 5.1: Trecho de código utilizado para gerar os sinais de entrada.

```
Degrau:
 [1 1 1 ... 1 1 1]
Rampa
 [0.000e+00 1.000e-02 2.000e-02 ... 5.997e+01 5.998e+01 5.999e+01]
Parabola:
 [0.0000000e+00 1.0000000e-04 4.0000000e-04 ... 3.5964009e+03
     3.5976004e+03
 3.5988001e+03]
```

Imagem 5.2: Print dos sinais de entrada.

Agora, partindo do vetor de tempo e dos vetores que representam nossos sinais de entrada, podemos fazer uso do método *control.matlab.lsim()* para obtermos a resposta do sistema. O método em questão recebe como parâmetro o sistema, que está representado pela variável chamada *system*, o sinal de entrada e o vetor de tempo. O trecho de código utilizado para este fim pode ser observado abaixo:

```
y_d, t_d, x_d = matlab.lsim(system, u_degrau, tempo) #Resposta do
    sistema ao Degrau
y_r, t_r, x_r = matlab.lsim(system, u_rampa, tempo) #Resposta do
    sistema a Rampa
y_p, t_p, x_p = matlab.lsim(system, u_parabola, tempo) #Resposta do
     sistema a Parabola
```

Imagem 5.3: Trecho de código utilizado para gerar a resposta do sistema.

Uma vez que os passos anteriores foram realizados e os retornos do método *control.matlab.lsim()* foram armazenados em variáveis, podemos utilizar a biblioteca *matplotlib* para exibir os resultados. A função utilizada para este fim pode ser observada abaixo:

```
import matplotlib.pyplot as plt

def Resposta(t, y, sinal, title, xlabel, ylabel):
  plt.plot(t, sinal, 'red',)
  plt.plot(t, y, 'black',)
  plt.title(title)
  plt.xlabel(xlabel)
  plt.ylabel(ylabel)
  plt.grid(0.5)
  plt.show()
  print()

Resposta(tempo, y_d, u_degrau, 'DEGRAU', 'Tempo', 'Amplitude')
Resposta(tempo, y_r, u_rampa, 'RAMPA', 'Tempo', 'Amplitude')
Resposta(tempo, y_p, u_parabola, 'PARBOLA', 'Tempo', 'Amplitude')
```

Imagem 5.4: Trecho de código utilizado para exibir na tela a resposta do sistema.

Os gráficos *tempo* x *amplitude* das respostas do sistema para cada sinal de entrada podem ser observados abaixo:

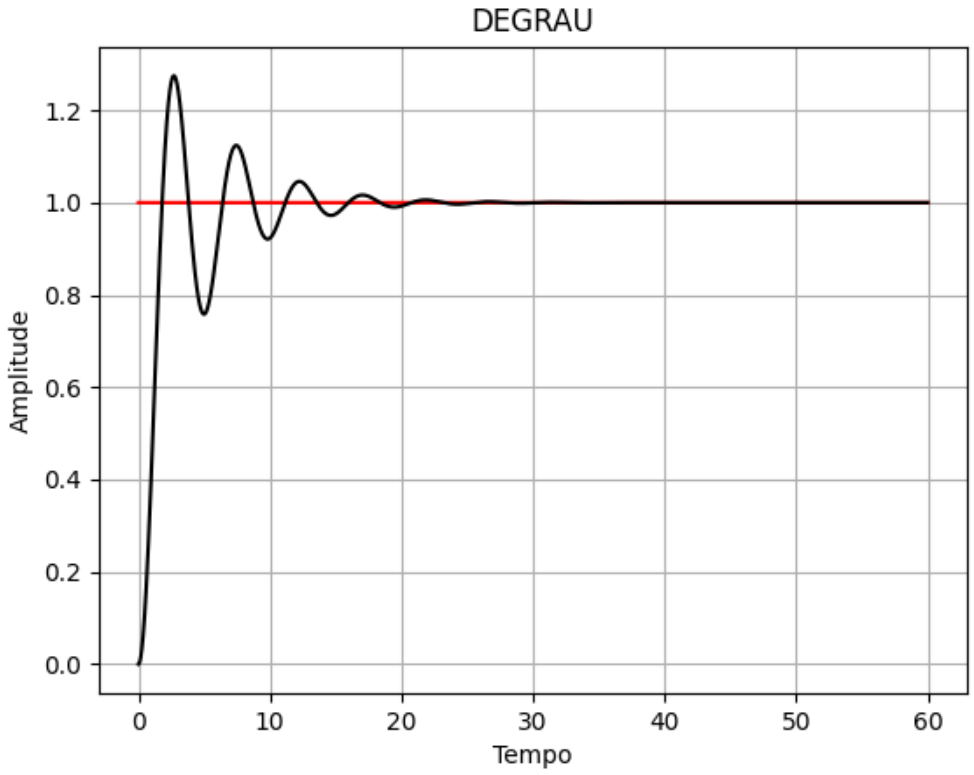

Imagem 5.5: Resposta do sistema ao Degrau

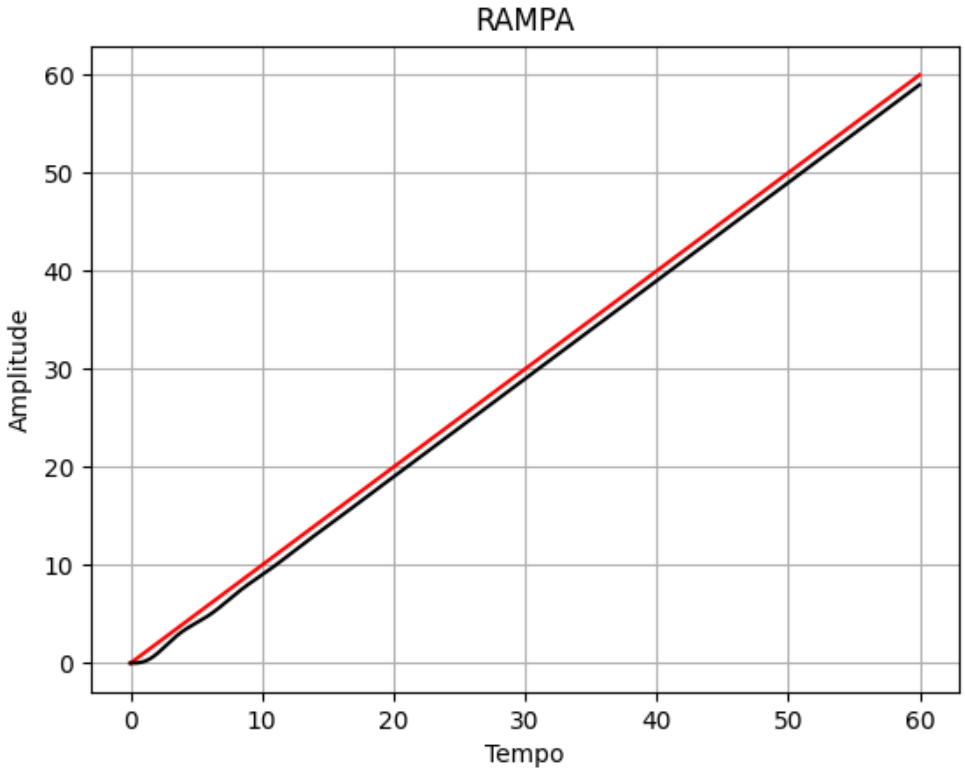

Imagem 5.6: Resposta do sistema à Rampa

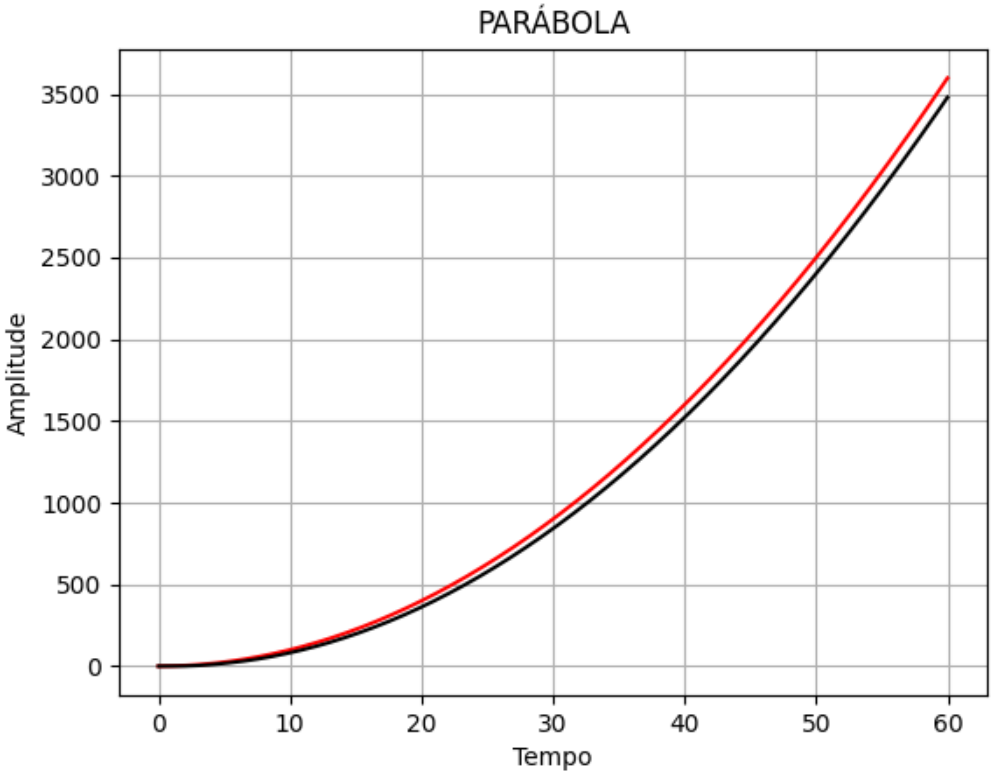

Imagem 5.7: Resposta do sistema à Parábola

As linhas destacadas em vermelho e preto representam respectivamente o sinal de entrada e a resposta do sistema.

6. Análise de comportamento

6.1. Tipo

A análise quanto ao Tipo do sistema pode ser feita partindo dos polos. Esses polos podem ser observados calculando os autovalores da matriz A de nosso sistema. Dessa forma foi feito o uso do método *eigvals()* presente na biblioteca *numpy*. O trecho de código utilizado pode ser observado abaixo:

```
autovalores = eigvals(A)
print('POLOS DO SISTEMA EM  MALHA ABERTA:')
print(autovalores)
```

Imagem 6.1: Trecho de código utilizado para determinar os polos do sistema

```
POLOS DO SISTEMA EM  MALHA ABERTA:
[-0.56984029+0.j , -0.21507985+1.30714128j , -0.21507985-1.30714128j]
```

Imagem 6.2: Saída do método *eigvals*().

Partindo do número de polos localizados na origem, que é 0, e do erro em regime permanente apresentado pelo sistema na **Imagem 5.5,** que aparenta ser um valor constante próximo de 0, podemos concluir que nosso sistema é do *tipo* 0.

6.2. Critérios de desempenho

As análises a seguir foram realizadas partindo de uma aproximação da resposta temporal de nosso sistema de terceira ordem para um de segunda ordem. Também foi feito uso do método *step_info()* presente na biblioteca *control* para determina os valores apresentados. O trecho de código utilizado e sua saída podem ser observados abaixo.

```
import control as ctrl

ctrl.step_info(system)
```

Imagem 6.3: Trecho de código utilizado para determinar os critérios de desempenho

```
{'Overshoot': 27.412024245469247,
 'Peak': 1.2741202424546925,
 'PeakTime': 2.6449430397438554,
 'RiseTime': 1.133547017033081,
 'SettlingMax': 1.2741202424546925,
 'SettlingMin': 0.7591090829563235,
 'SettlingTime': 15.302884729946593,
 'SteadyStateValue': 1.0,
 'Undershoot': 0}
```

Imagem 6.4: Saída do método *control.step_info*().

6.2.1. Estabilidade

Usando os resultados descritos na **Imagem 6.2** podemos admitir que nosso sistema é *estável* uma vez que todos os seus polos estão contidos no semiplano esquerdo.

6.2.2. Amortecimento

Usando os resultados descritos na imagem **6.4** e observando a resposta do sistema ao degrau na **Imagem 5.5** podemos admitir que nosso sistema é *subamortecido* uma vez que seus polos estão contidos no semiplano esquerdo e sua resposta é próxima à resposta típica de um sistema de segunda ordem subamortecido.

6.2.3. Tempo de pico (t_p)

O tempo de pico é definido como o instante que ocorre o primeiro pico de $y(t)$. Para o nosso sistema temos que $t_p \cong 2.64s$. Partindo do último resultado podemos calcular $y(t_p)$, que é o valor da resposta do sistema no tempo de pico e também o valor de regime do sistema. Para nosso sistema temos que $y(t_p) \cong 1.27$.

6.2.4. Sobressinal Máximo (M_p)

O valor de sobressinal (*overshoot*) é realizado levando em consideração o resultado de $y(t_p)$ obtido no item **6.2.3** da seguinte forma:

$$M_p = \frac{y(t_p) - y_{ss}}{y_{ss}}$$

Onde:

$$y_{ss} = \lim_{t \to \infty} y(t)$$

Dessa forma, para o nosso sistema temos que $M_p = 27.41$.

6.2.5. Tempo de subida (t_r)

Observando a **Imagem 6.4** podemos admitir que para o nosso sistema temos $t_r \cong 1.13s$.

6.2.6. Tempo de acomodação (t_s)

O tempo de acomodação é o tempo que $y(t)$ leva para atingir e permanecer em uma faixa de $\pm 2\%$ (ou $\pm 5\%$) do valor final. Temos que para o nosso sistema $t_s \cong 15.3s$.

6.2.7. Representação gráfica dos critérios de desempenho

Os valores apontados nos itens anteriores podem ser observados graficamente abaixo:

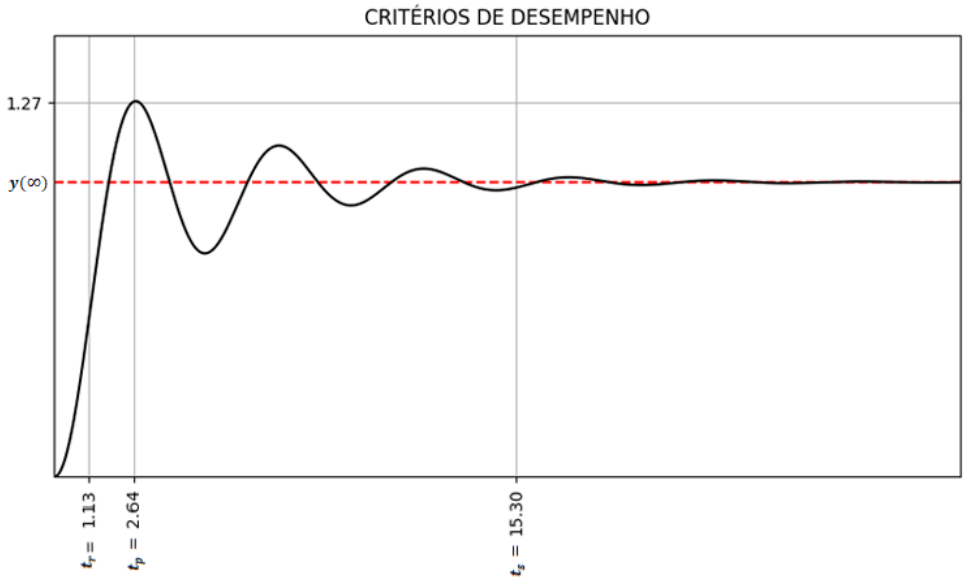

Imagem 6.5: Representação gráfica dos critérios de desempenho

7. Controlabilidade e Observabilidade

Do livro: *Engenharia de Sistemas de Controle - 6ª edição*, temos as seguintes definições para um Sistema controlável e um Sistema observável:

- Controlabilidade: Se para um sistema for possível obter uma entrada capaz de transferir todas as variáveis de estado de um estado inicial desejado para um estado final desejado, o sistema é dito controlável; caso contrário, o sistema é não controlável.
- Observabilidade: Se o vetor de estado inicial, $x(t_0)$, puder ser obtido a partir de $u(t)$ e $y(t)$ medidos durante um intervalo de tempo finito a partir de t_0, o sistema é dito observável; caso contrário o sistema é dito não observável.

Uma das formas de se realizar a análise de um sistema quanto a esses critérios é construir suas matrizes, tanto de controlabilidade quanto de observabilidade.

7.1. Análise quanto à Matriz de Controlabilidade

Temos que um sistema de ordem n definido em Espaço de Estados pelas matrizes A, B, C e D é dito completamente controlável se sua matriz de controlabilidade C_M possui posto n. Onde C_M é definida da seguinte forma:

$$C_M = [B \;\; A^1B \;\; A^2B \;\; \ldots \;\; A^{n-1}B]$$

Temos que para o nosso sistema, uma vez que as matrizes A e B já estão definidas, podemos fazer uso do método $ctrb()$ presente no módulo $matlab$ da biblioteca $control$

para gerar a matriz $\boldsymbol{C_M}$. O trecho de código utilizado para este fim e a saída obtida podem ser observados abaixo.

```
from control import matlab

#Matriz de Controlabilidade do sistema
CTR = matlab.ctrb(A,B)
print(CTR)
```

Imagem 7.1: Trecho de código utilizado para gerar $\boldsymbol{C_M}$.

```
MATRIZ DE CONTROLABILIDADE DO SISTEMA:
[[ 0.  1. -1.]
 [ 1.  0. -2.]
 [ 0.  1.  0.]]
```

Imagem 7.2: Saída do método *matlab.ctrb*().

Dessa forma, temos que a saída mostrada na **Imagem 7.2** que representa nossa matriz $\boldsymbol{C_M}$ assume a seguinte forma:

$$C_M = \begin{pmatrix} 0 & 1 & -1 \\ 1 & 0 & -2 \\ 0 & 1 & 0 \end{pmatrix}$$

Agora, para critério de análise quanto a controlabilidade do sistema é preciso calcular o posto da mesma, para isso foi utilizado o método *matrix_rank*() presente na biblioteca *numpy*. O trecho de código utilizado para este fim e a saída obtida podem ser observados abaixo.

```
from numpy.linalg import matrix_rank

print(matrix_rank(CTR))
```

Imagem 7.3: Trecho de código utilizado para calcular o posto de *CTR*.

```
POSTO DA MATRIZ DE CONTROLABILIDADE:
3
```

Imagem 7.4: Saída do método *matriz_rank*().

Portanto, uma vez que o sistema possui ordem $n = 3$ e o método que calcula o posto retornou 3, podemos concluir que o sistema é *completamente controlável*.

7.2. Análise quanto à Matriz de Observabilidade

Temos que um sistema de ordem n definido em Espaço de Estados pelas matrizes A, B, C e D é dito completamente observável se sua matriz de observabilidade $\boldsymbol{O_M}$ possui posto n. Onde $\boldsymbol{O_M}$ é definida da seguinte forma:

$$\boldsymbol{O_M} = \begin{bmatrix} C \\ CA^1 \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

Temos que para o nosso sistema, uma vez que as matrizes A e C já estão definidas, podemos fazer uso do método $obsv()$ presente no módulo $matlab$ da biblioteca $control$ para gerar a matriz $\boldsymbol{O_M}$. O trecho de código utilizado para este fim e a saída obtida podem ser observados abaixo.

```
from control import matlab

#Matriz de Observabilidade do sistema
OBS = matlab.obsv(A,C)
print(OBS)
```

Imagem 7.5: Trecho de código utilizado para gerar $\boldsymbol{O_M}$

```
MATRIZ DE OBSERVABILIDADE DO SISTEMA:
[[ 0.  0.  1.]
 [ 0.  1.  0.]
 [-1.  0. -1.]]
```

Imagem 7.6: Saída do método *matlab.obsv*().

Dessa forma, temos que a saída mostrada na **Imagem 7.6** que representa nossa matriz $\boldsymbol{O_M}$ assume a seguinte forma:

$$O_M = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & -1 \end{pmatrix}$$

Agora, para critério de análise quanto a observabilidade do sistema é preciso calcular o posto da mesma, para isso foi utilizado o método $matrix\_rank()$ presente na biblioteca $numpy$. O trecho de código utilizado para este fim e a saída obtida podem ser observados abaixo.

```
from numpy.linalg import matrix_rank

print(matrix_rank(OBS))
```

Imagem 7.7: Trecho de código utilizado para calcular o posto de *OBS*.

```
POSTO DA MATRIZ DE OBSERVABILIDADE:
3
```

Imagem 7.8: Saída do método *matriz_rank*().

Portanto, uma vez que o sistema possui ordem $n = 3$ e o método que calcula o posto retornou 3, podemos concluir que o sistema é *completamente observável*.

8. Conclusões

A partir das atividades elaboradas para este relatório podemos concluir que ao realizarmos a modelagem em Espaço de Estados de um sistema qualquer, a priori com suas características desconhecidas, e em seguida partindo desta modelagem, é possível determinar o comportamento desse sistema quando alimentamos sua entrada com um sinal referência, por exemplo um degrau. Levando em conta esse comportamento podemos determinar os seus critérios de desempenho e partindo de seus critérios de desempenho podemos classificar este sistema.

As características que dizem respeito a controlabilidade e a observabilidade também podem ser obtidas partindo da modelagem em E.E. Por fim, a função de transferência do sistema também pode ser descrita, junto de seus *polos* e *zeros*.

9. Anexos

ANEXO A – Notebook com as implementações realizadas para a análise deste documento:

< Sistemas de Controle 2 - Parte 1 - AB1 - Colaboratory (google.com) >

10. Bibliografia

NISE, Norman. Engenharia de Sistemas de Controle. 6ª ED. Rio de Janeiro: LTC - Livros Técnicos e Científicos Editora Ltda, 2012.

OGATA, Katsushiro. Engenharia de controle Moderno. 5ª ED. São Paulo: Pearson Education do Brasil, 2011.

KLUEVER, Craig. Sistemas Dinâmicos – Modelagem, Simulação e Controle. 1ª ED. Rio de Janeiro: LTC - Livros Técnicos e Científicos Editora Ltda, 2017.

PYTHON SOFTWARE FOUNDATION. Python Language Site: Python Control Systems Library, 2022. Página de documentação. Disponível em: <https://python-control.readthedocs.io/en/0.9.1/>. Acesso em: 06 de Maio. de 2022.

PYTHON SOFTWARE FOUNDATION. Python Language Site: Signal processing, 2022. Página de documentação. Disponível em: < https://docs.scipy.org/doc/scipy/reference/signal.html>. Acesso em: 06 de Maio. de 2022.