**MARINHA DO BRASIL**

**DIRETORIA DE ENSINO DA MARINHA**

**CENTRO DE INSTRUÇÃO ALMIRANTE ALEXANDRINO**

CURSO DE APERFEIÇOAMENTO AVANÇADO EM

PROPULSÃO NAVAL

**TRABALHO DE SISTEMAS DE CONTROLE E AUTOMAÇÃO II**



1ºTen VINICIUS DE LEMOS CARDOSO RONDON

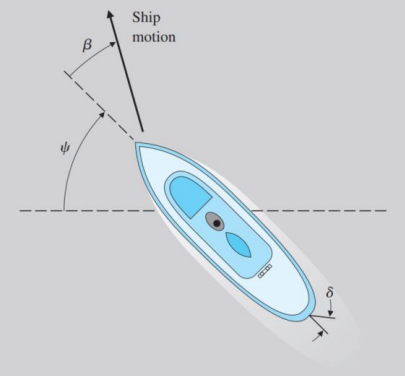
1ºTen GILBERTO AZEVEDO CHAGAS JUNIOR

CIAA

Rio de Janeiro

2023

**Questão 01:**

****

**Item a)**

Afirmativo, o sistema é manipulável, pois se trata de uma matriz 3x3 na qual as linhas e colunas são linearmente independentes, permitindo o controle total dos estados do sistema a partir de um conjunto específico de entradas. Isso, aliado ao fato de que o determinante da matriz é .

**Item b)**

Utilizando o código abaixo no *Python* obteve-se a seguinte função de transferência:

|  |
| --- |
| import numpy as np  *# Definindo as matrizes A e B do sistema* A = np.array([[-0.0895, -0.286, 0],  [-0.0439, -0.272, 0],  [0, 1, 0]]) B = np.array([[0.0145],  [-0.0122],  [0]])  *# Calculando a matriz de controlabilidade manualmente* def calc\_controlability\_matrix(A, B):  control\_matrix = B  for i in range(1, A.shape[0]):  control\_matrix = np.hstack((control\_matrix, np.linalg.matrix\_power(A, i) @ B))  return control\_matrix  *# Calculando a matriz de controlabilidade* C = calc\_controlability\_matrix(A, B)  *# Verificando se o sistema é controlável* *# O sistema é controlável se o rank da matriz de controlabilidade é igual ao número de estados* is\_controlable = np.linalg.matrix\_rank(C) == A.shape[0] controlability\_rank = np.linalg.matrix\_rank(C)  C, is\_controlable, controlability\_rank |

|  |
| --- |
| from scipy.signal import ss2tf  *# Definindo a matriz C para a saída de interesse (ψ - cabeceio)* C\_psi = np.array([[0, 0, 1]]) *# A matriz D é dada como zero* D = np.array([[0]])  *# Convertendo sistema do espaço de estados para função de transferência* numerator, denominator = ss2tf(A, B, C\_psi, D)  *# Calculando as raízes da equação característica do sistema* *# que são os polos da função de transferência (ou seja, as raízes do denominador)* characteristic\_eq\_roots = np.roots(denominator)  numerator, denominator, characteristic\_eq\_roots |

Resultando em:

|  |
| --- |
| (array([[ 0.00000000e+00, 2.77555756e-16, -1.22000000e-02,  -1.72845000e-03]]),  array([1. , 0.3615 , 0.0117886, 0. ]),  array([-0.32525593, -0.03624407, 0. ])) |

A função de transferência representa o sistema sem alocação de polos, na qual o zero se encontra em **0.141676229508197** e os polos se encontram em **0**, **-0.325255925484044** e **-0.0362440745159563.** Como um dos polos se encontra em 0, conclui-se que o sistema é marginalmente estável.

**Item c)**

Os valores de , , para que as raízes se localizassem em 𝑠 = −0.2, −0.2 ± 0.2𝑗 foram obtidos utilizando programação em Python, detalhada abaixo. Na qual, , , resultaram em 0.2766, -19.2203 e -9.2568 respectivamente.

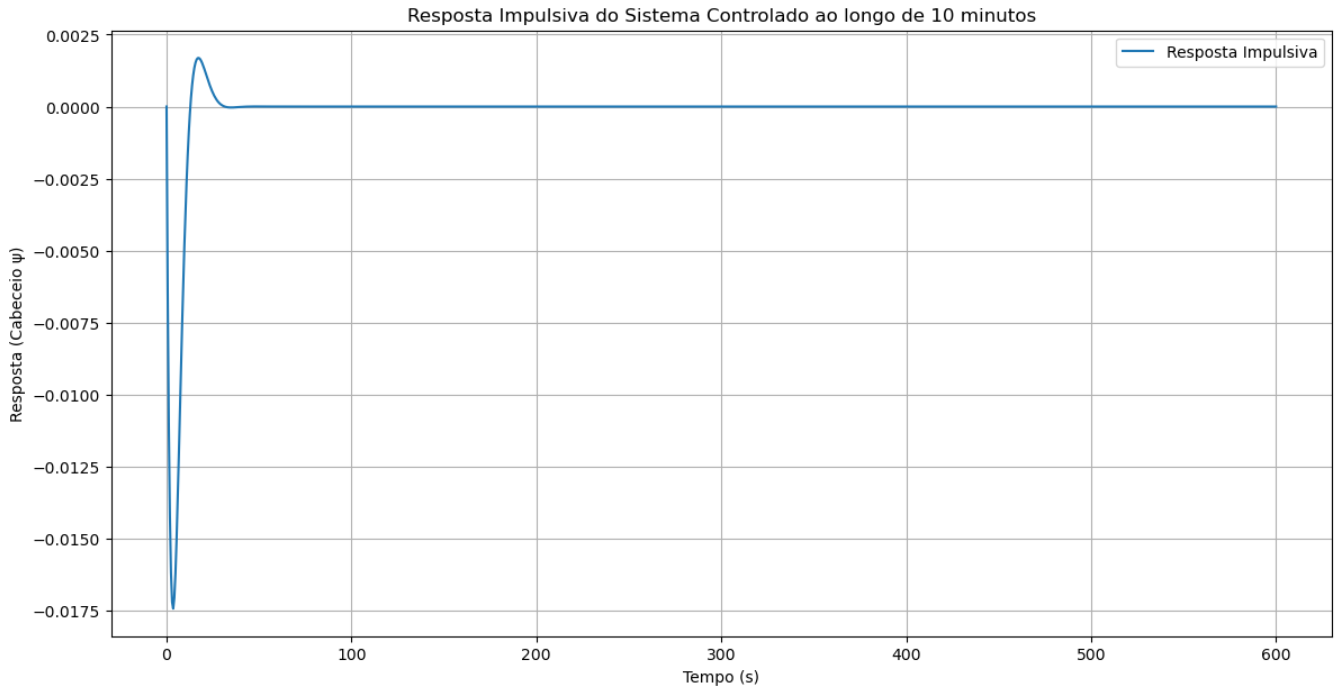
|  |
| --- |
| from scipy.signal import place\_poles  *# Definindo as raízes desejadas para o sistema controlado* desired\_poles = np.array([-0.2, -0.2 + 0.2j, -0.2 - 0.2j])  *# Utilizando a função place\_poles para calcular o vetor de ganhos K* placed\_poles = place\_poles(A, B, desired\_poles)  *# O vetor de ganhos K* K = placed\_poles.gain\_matrix |

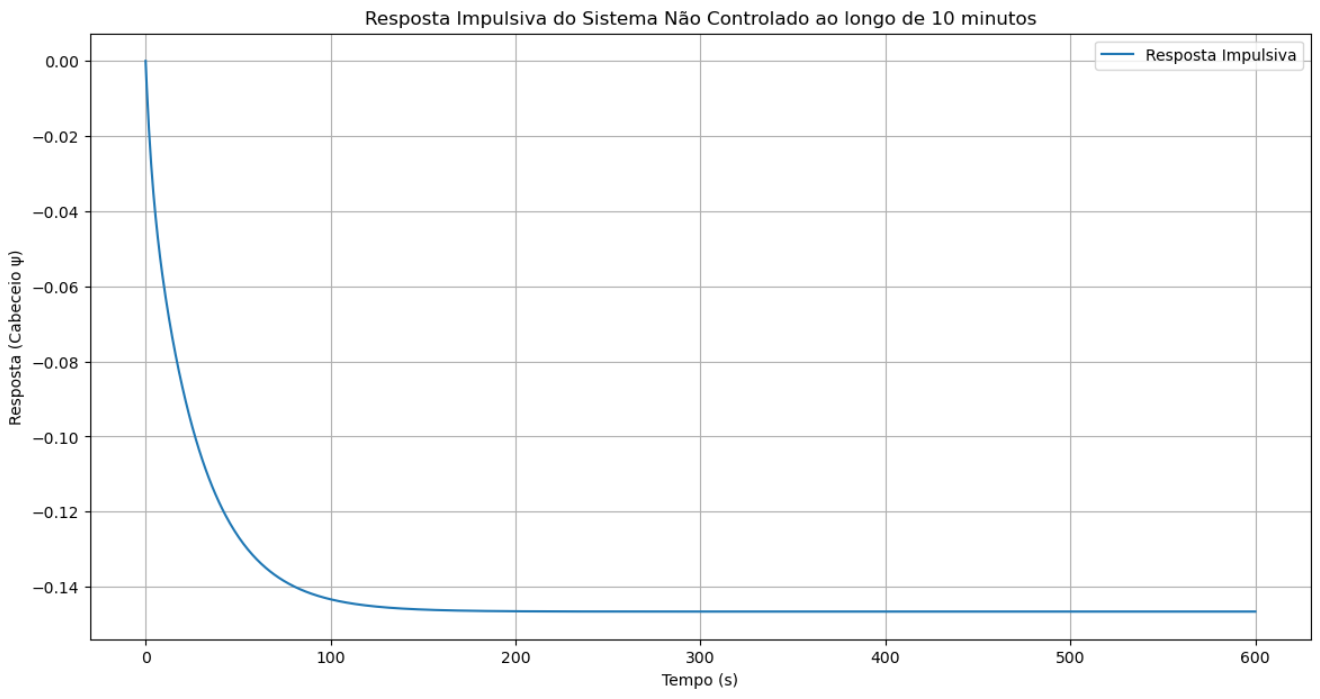
array([[ 0.27664905, -19.22037613, -9.25684862]])

**Item d)**

Segue abaixo, a programação em *Python* simulando a resposta impulsiva por 10 minutos e o resultado da resposta para o sistema controlado e não controlado.

|  |
| --- |
| from scipy.signal import StateSpace, impulse  *# Criando o sistema de espaço de estados com feedback usando o vetor de ganhos K* A\_feedback = A - B @ K system = StateSpace(A\_feedback, B, C\_psi, D)  *# Simulando a resposta impulsiva por 600 segundos (10 minutos)* t\_impulse = np.linspace(0, 600, 1000) *# Gerando um vetor de tempo com 1000 pontos* t\_impulse, response\_impulse = impulse(system, T=t\_impulse)  t\_impulse, response\_impulse |

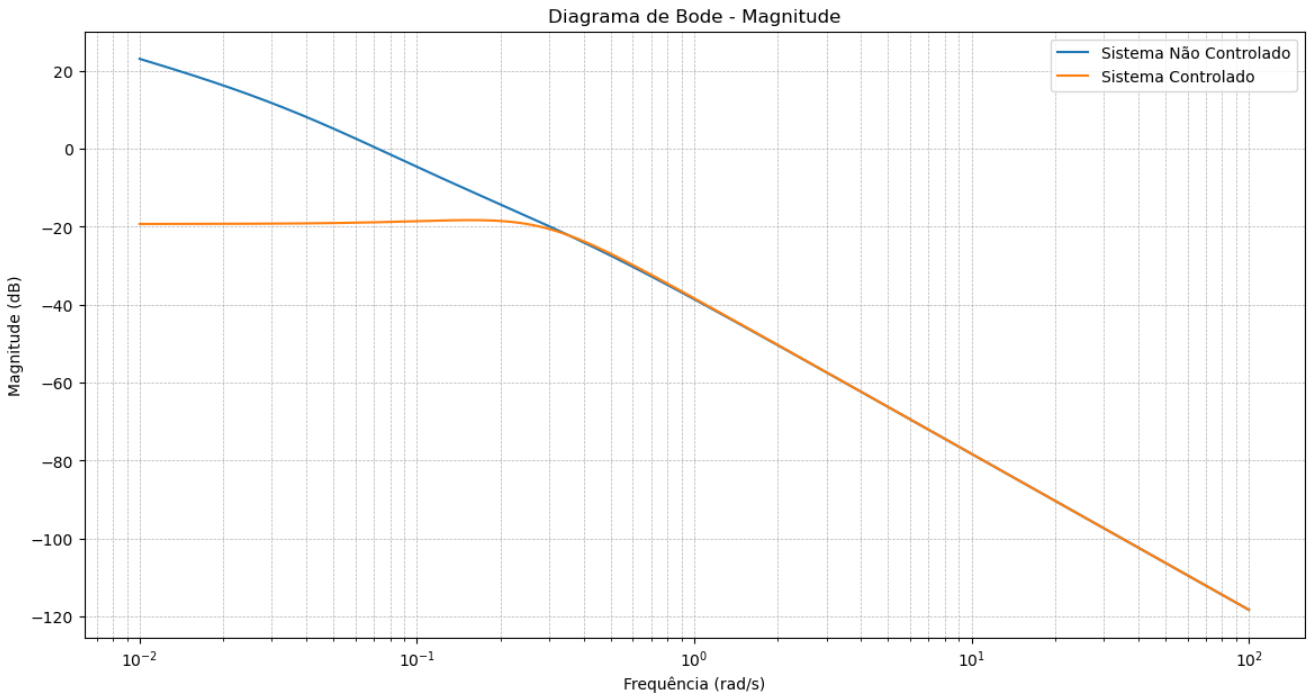
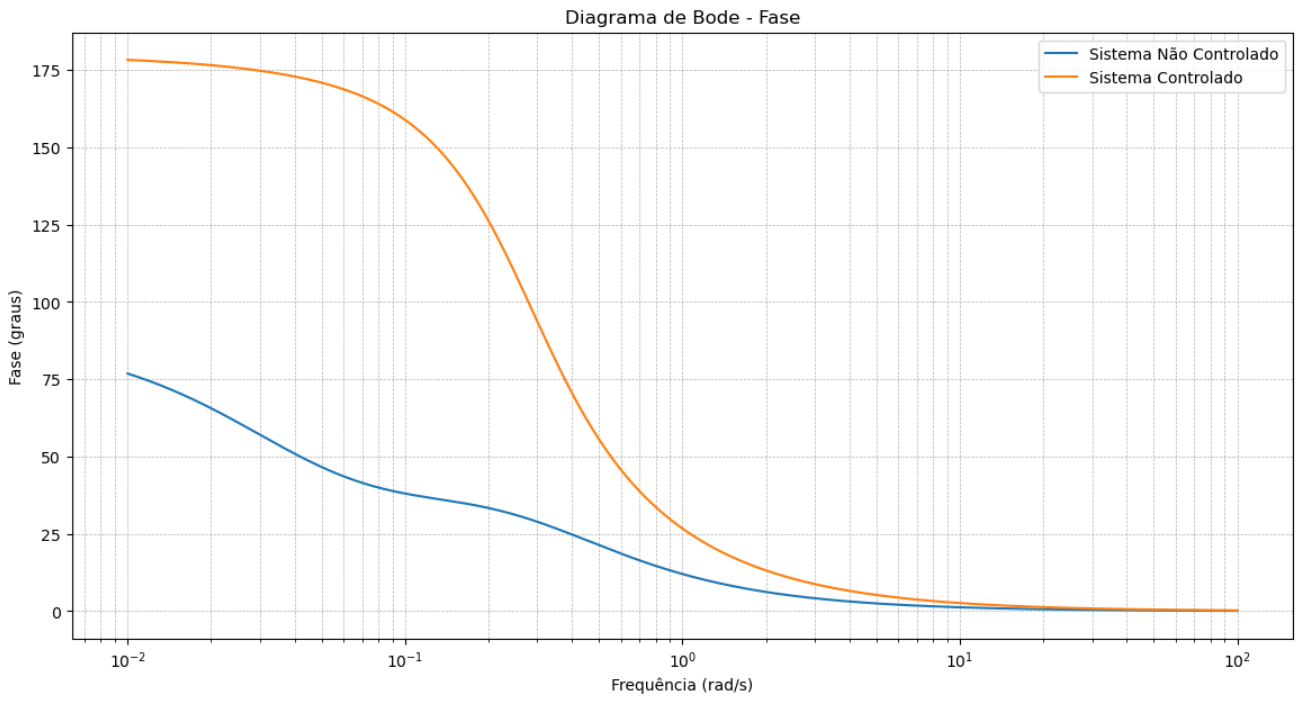




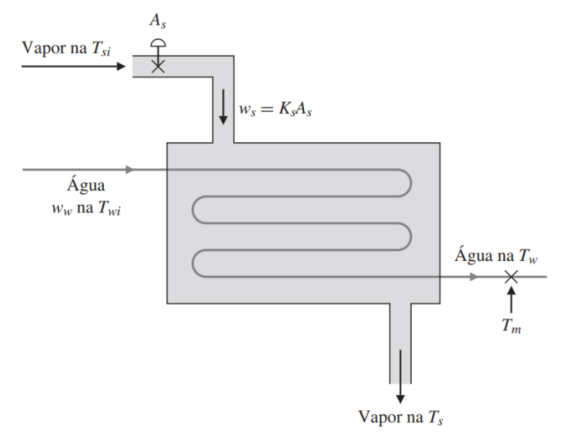
**Item e)**

A seguir constam os diagramas de Bode para resposta em frequência para o sistema não controlado e para o sistema controlado, respectivamente.

|  |
| --- |
| from scipy.signal import bode  *# Sistema não controlado* system\_uncontrolled = StateSpace(A, B, C\_psi, D)  *# Frequências para o diagrama de Bode* frequencies = np.logspace(-2, 2, 500) *# De 0.01 a 100 rad/s*  *# Calculando a resposta à frequência para ambos os sistemas* w\_uncontrolled, mag\_uncontrolled, phase\_uncontrolled = bode(system\_uncontrolled, w=frequencies) w\_controlled, mag\_controlled, phase\_controlled = bode(system, w=frequencies)  *# Plotando os diagramas de Bode para a magnitude* plt.figure(figsize=(14, 7)) plt.semilogx(w\_uncontrolled, mag\_uncontrolled, label='Sistema Não Controlado') plt.semilogx(w\_controlled, mag\_controlled, label='Sistema Controlado') plt.title('Diagrama de Bode - Magnitude') plt.xlabel('Frequência (rad/s)') plt.ylabel('Magnitude (dB)') plt.grid(which='both', linestyle='--', linewidth=0.5) plt.legend() plt.show()  *# Plotando os diagramas de Bode para a fase* plt.figure(figsize=(14, 7)) plt.semilogx(w\_uncontrolled, phase\_uncontrolled, label='Sistema Não Controlado') plt.semilogx(w\_controlled, phase\_controlled, label='Sistema Controlado') plt.title('Diagrama de Bode - Fase') plt.xlabel('Frequência (rad/s)') plt.ylabel('Fase (graus)') plt.grid(which='both', linestyle='--', linewidth=0.5) plt.legend() plt.show() |

****

**Questão 02:**



**Item a)**

Passando para o domínio de Laplace, resulta em:

**(1)**

**(2)**

Substituindo **(1)** em **(2)**:

Considerando p1 e p2 as raízes do polinômio:

Resulta em:

Considerando:

Resulta em:

Sendo:

O atraso no tempo é representado por uma exponencial:

**Item b)**

**Item c)**

Para posicionar os polos em −0.15 ± 0.26𝑖 podem ser utilizados 2 ganhos, sendo eles **K1= 0.1833** e **K2= 0.088433**. O cálculo dos ganhos foi realizado a partir do código de programação abaixo:

|  |
| --- |
| from scipy.signal import tf2ss  *# Definindo os polinômios do numerador e do denominador da função de transferência* num = [0, 0, 1] den = [600, 70, 1]  *# Convertendo a função de transferência para espaço de estados* A, B, C, D = tf2ss(num, den)  *# Definindo os polos desejados para o sistema em malha fechada* desired\_poles = np.array([-0.15 + 0.26j, -0.15 - 0.26j])  *# Calculando o vetor de ganho K usando a fórmula de Ackermann* result\_ackermann = place\_poles(A, B, desired\_poles) K\_ackermann = result\_ackermann.gain\_matrix  A, B, C, D, K\_ackermann |

|  |
| --- |
| (array([[-0.11666667, -0.00166667],  [ 1. , 0. ]]),  array([[1.],  [0.]]),  array([[0. , 0.00166667]]),  array([[0.]]),  array([[0.18333333, 0.08843333]])) |

**Item d)**

A conversão do sistema foi realizada a partir do código de programação abaixo:

|  |
| --- |
| from scipy.signal import cont2discrete  *# Definindo a taxa de amostragem* sampling\_period = 10 *# segundos*  *# Sistema contínuo a partir da função de transferência* sys\_continuous = lti(num, den)  *# Obtendo as matrizes de espaço de estados do sistema contínuo* (A\_cont, B\_cont, C\_cont, D\_cont) = tf2ss(num, den)  *# Convertendo o sistema de espaço de estados contínuo para discreto usando ZOH* system\_discrete = cont2discrete((A\_cont, B\_cont, C\_cont, D\_cont), sampling\_period, method='zoh')  *# Extraindo as matrizes discretas do resultado* (A\_discrete, B\_discrete, C\_discrete, D\_discrete, dt) = system\_discrete  (A\_discrete, B\_discrete, C\_discrete, D\_discrete) |

|  |
| --- |
| (array([[ 0.27215898, -0.00957205],  [ 5.7432274 , 0.94220218]]),  array([[ 5.7432274 ],  [34.67869102]]),  array([[0. , 0.00166667]]),  array([[0.]])) |

|  |
| --- |
| import numpy as np from scipy.signal import cont2discrete, dlti  def encontrar\_polos\_discretos(system\_discrete):  *# Extrair as matrizes de espaço de estados discretas do sistema*  A\_discrete, B\_discrete, C\_discrete, D\_discrete, dt = system\_discrete    *# Criar um sistema discreto a partir das matrizes de espaço de estados discretas*  system\_discrete = dlti(A\_discrete, B\_discrete, C\_discrete, D\_discrete, dt=dt)    *# Encontrar os polos discretos do sistema*  poles\_discrete = system\_discrete.poles    return poles\_discrete  *# Definindo a taxa de amostragem* sampling\_period = 10 *# segundos*  *# Sistema contínuo a partir da função de transferência* sys\_continuous = lti(num, den)  *# Obtendo as matrizes de espaço de estados do sistema contínuo* (A\_cont, B\_cont, C\_cont, D\_cont) = tf2ss(num, den)  *# Convertendo o sistema de espaço de estados contínuo para discreto usando ZOH* system\_discrete = cont2discrete((A\_cont, B\_cont, C\_cont, D\_cont), sampling\_period, method='zoh')  *# Encontrar os polos discretos do sistema* polos\_discretos = encontrar\_polos\_discretos(system\_discrete)  print("Polos discretos do sistema:") print(polos\_discretos) |

Através do código de programação mostrado acima, foram encontrados **polos** em **0.367879441171442** e **0.846481724890614** para um sistema discreto.

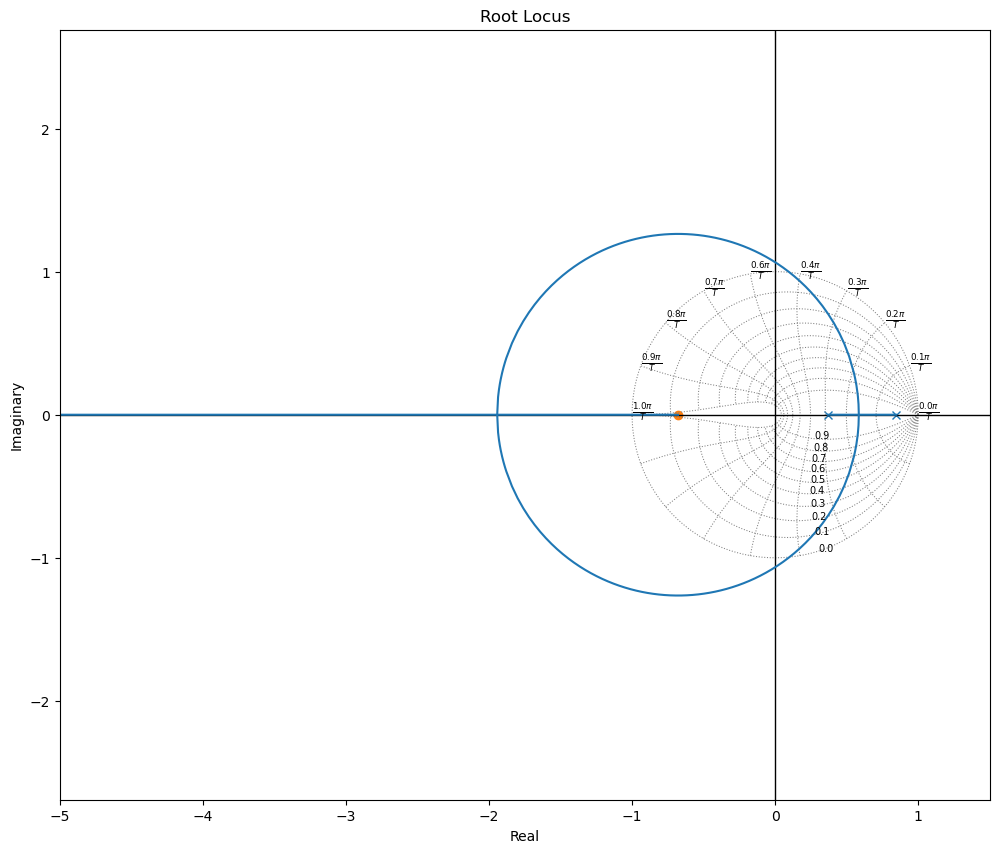
|  |
| --- |
| Polos discretos do sistema: [0.84648172 0.36787944] |

**Item e)**

Para determinar o lugar das raízes foi utilizado o código abaixo em *Python*:

|  |
| --- |
| import control as ctrl import matplotlib.pyplot as plt  *# Definindo a função de transferência contínua* num = [0, 0, 1] den = [600, 70, 1] sys\_continuous = ctrl.TransferFunction(num, den)  *# Definindo a taxa de amostragem* sampling\_period = 10 *# segundos*  *# Convertendo a função de transferência contínua para o domínio discreto usando ZOH* sys\_discrete = ctrl.sample\_system(sys\_continuous, sampling\_period, method='zoh')  *# Definindo as dimensões da figura* plt.figure(figsize=(12, 10))  *# Plotando o lugar das raízes para o sistema discreto* rlocus\_data = ctrl.root\_locus(sys\_discrete, Plot=True)  *# Ajustando os limites dos eixos* plt.axis([-5, 1.5, -1.5, 1.5])  plt.show() |

Obtendo o seguinte resultado:



Para determinar o ganho máximo foi utilizado o código abaixo em *Python*, no qual o **ganho máximo** é no valor de **17.5**

|  |
| --- |
| import numpy as np import control as ctrl  *# Definindo a função de transferência contínua e convertendo para o domínio discreto* num = [0, 0, 1] den = [600, 70, 1] sys\_continuous = ctrl.TransferFunction(num, den) sampling\_period = 10 *# segundos* sys\_discrete = ctrl.sample\_system(sys\_continuous, sampling\_period, method='zoh')  *# Plotando o lugar das raízes e obtendo os dados* rlocus\_data, gains = ctrl.root\_locus(sys\_discrete, Plot=False, kvect=np.linspace(0, 17.5, 1000))  *# Encontrando o ganho máximo para estabilidade* def find\_max\_gain\_for\_stability(rlocus\_data, gains):  for i, poles in enumerate(rlocus\_data.T):  if np.any(np.abs(poles) >= 1):   return gains[i-1] if i > 0 else 0 *# Retorna o ganho anterior se existir, senão retorna 0*  return gains[-1] *# Retorna o último ganho se todos forem estáveis*  max\_gain = find\_max\_gain\_for\_stability(rlocus\_data, gains) print("Ganho máximo para estabilidade:", max\_gain) |

|  |
| --- |
| Ganho máximo para estabilidade: 17.5 |