UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Universidade Federal do Ceará - Campus Quixadá

Simulações - Transformada Z
QXD0143 - Processamento Digital de Sinais

Prof. Carlos Igor Bandeira

Aluno: Hugo Bessa - **Matrícula:** 496870

Aluno: Isaac Vinícius - **Matrícula:** 500935

Aluna: Kassia Lopes- **Matrícula:** 49367

Quixadá-CE

Sumário

Simulações Transformada Z

1 Exemplos

1.1 Exemplo 4.4

1.1.1 Código Python

```
import numpy as np

#definindo os coeficientes das sequ ncias X1 e X2

x1 = [2,3,4]
x2 = [3,4,5,6]

#fun   o que realiza a convolu   o

x3 = np.convolve(x1,x2)

print(x3)

#sa da
x3 = [ 6 17 34 43 38 24]
```

1.2 Exemplo 4.5

1.2.1 Código Python

```
#definindo a fun   o conv_m(do MatLab) em Python
def conv_m(x1, n1, x2, n2):
    # Convolu   o dos sinais
    x3 = np.convolve(x1, x2)

    # C lculo dos novos   ndices
    n3_start = n1[0] + n2[0]   #   ndice   inicial
    n3_end = n1[-1] + n2[-1]   #   ndice   final
```

```
    n3 = np.arange(n3_start, n3_end + 1)  # Cria os novos
     ndices

    return x3, n3

#calculando a convolu  o entre as sequ ncias dadas. a
    fun  o conv_m retorna o sinal resultante e seus ind ces
    .

x1 = [1, 2, 3]
n1 = np.arange(-1, 2)  #cria o array n1 que representa os
    ind ces de X1 [-1, 0, 1]
x2 = [2, 4, 3, 5]
n2 = np.arange(-2, 2)  #cria o array n1 que representa os
    ind ces de X2 [-2, -1, 0, 1]

x3, n3 = conv_m(x1, n1, x2, n2)

print("x3 =", x3)
print("n3 =", n3)

#sa da

x3 = [ 2  8 17 23 19 15]
n3 = [-3 -2 -1  0  1  2]
```

1.3 Exemplo 4.6

1.3.1 Código Python

```
import numpy as np
from scipy.signal import lfilter

# Parametros
b = [0, 0, 0, 0.25, -0.5, 0.0625]
a = [1, -1, 0.75, -0.25, 0.0625]

```

```
# Criacao de delta
n = np.arange(0, 8)  # Vetor de  ndices  de 0 a 7
delta = np.zeros_like(n)
delta[n == 0] = 1  # Impulso unit rio em n = 0

# Filtragem do impulso
x = lfilter(b, a, delta)

# Funcao stepseq
def stepseq(n0, n1, n2):
    return np.where((n >= n1) & (n <= n2), 1, 0)

# Cria  o da sequencia original
step = stepseq(2, 0, 7)
x_orig = (n - 2) * (1/2)**(n - 2) * np.cos(np.pi * (n - 2) /
    3) * step

print("Delta:")
print(delta)
print("x (filtragem):")
print(x)
print("x (sequencia original):")
print(x_orig)

#saida

Delta:
[1 0 0 0 0 0 0 0]
x (filtragem):
[ 0.         0.         0.         0.25      -0.25      -0.375
   -0.125
  0.078125]
x (sequencia original):
[ 4.        -1.         0.         0.25      -0.25      -0.375
   -0.125
  0.078125]
```

1.4 Exemplo 4.8

Para encontrar a trasnformada Z inversa da equação $X(z) = \frac{z}{3z^2-4z+1}$, foram necessárias utilizar várias propriedades da transformada Z para manipular e reescrever a expressão. As principais propriedades envolvidas são:

- Fatoração Polinomial: Inicialmente, o denominador da equação $3z^2 - 4z + 1$ foi fatorado para facilitar a decomposição em frações parciais. Esta fatoração resulta em dois fatores lineares que são essenciais para aplicar as propriedades seguintes.

- Decomposição em Frações Parciais: A equação foi decomposta em frações parciais para expressar $X(z)$ como uma soma de termos mais simples que podem ser invertidos diretamente. Esta técnica é comum quando se deseja encontrar a transformada inversa de Z, pois permite o uso de tabelas de transformadas Z inversas para cada termo individual.

- Deslocamento de Amostra: A expressão $\frac{1}{1-az^{-1}}$ é a forma padrão de uma função de transferência que corresponde a uma série geométrica no domínio do tempo. O uso desta forma sugere a aplicação da propriedade de deslocamento de amostra para identificar termos na série geométrica.

- Propriedade da Convolução: A equação resultante envolve produtos de termos do tipo $\frac{1}{1-az^{-1}}$, que, ao serem invertidos, correspondem a convoluções de sequências no domínio do tempo. A propriedade da convolução pode ser usada para obter a resposta devida a cada termo.

- Deslocamento de Frequência: A presença de termos como z^{-1} dentro das frações parciais indica que a propriedade de deslocamento de frequência está sendo aplicada. Esta propriedade é essencial para interpretar corretamente os termos na forma da série geométrica invertida.

Para fazer a implementação de forma computacional, foi utilizado a ferramenta GNU Octave para realizar todas as manipulações, a fim de encontrar a transformada Z inversa da equação $X(z) = \frac{z}{3z^2-4z+1}$.

1.4.1 GNU Octave

```
clear all
close all

b = [0, 1]; % Definindo numerador da funcao X(z)

% O polinomio no denominador: 3z^2 -4z + 1
% Os coeficientes (az^2 + bz + c) sao: a = 3, b = -4, c = 1
% As raizes sao z1 = 1 e z2 = 1/3
a = [3, -4, 1];

% Realizando o calculo da decomposicao em fracoes parciais
    com a funcao residue
[r, p, c] = residue(b, a)

% Reconstruindo a funcao original a partir das componentes em
     partes
[b, a] = residue(r, p, c)
```

Primeiramente, foi realizada a limpeza de todas as variáveis do ambiente de trabalho do Octave, e de todas as janelas de figuras que possam estar abertas. Em sequência, utilizando as propriedades descritas anteriormente, houve a definição dos coeficientes do Numerador e Denominador da transformada Z onde:

- A variável b = [0, 1] Define o numerador da função $X(z)$ que, no seu caso, é simplesmente z, ou seja, $1 \cdot z^1$ sem termos constantes. Isso é representado como o vetor [0, 1].

- A variável a = [3, -4, 1] Define o denominador da função $X(z) = \frac{z}{3z^2-4z+1}$. Aqui, $a(z) = 3z^2 - 4z + 1$, então o vetor correspondente é [3,

-4, 1].

Para a realização da Expansão em Frações Parciais, foi utilizada o comando Residue, que calcula os resíduos "r", os polos "p", e os termos diretos "c", conforme é mostrado abaixo:

- linha 12: [r, p, c] = residue(b, a).

A função residue no Octave é uma ferramenta utilizada para realizar a expansão em frações parciais de quocientes de polinômios e para reconstituir esses quocientes a partir de sua expansão em frações parciais. A função pode ser usada tanto para decompor uma função racional em suas partes componentes quanto para reconstruir a função original a partir dessas partes. Na figura 1 é apresentado a saída, no qual podemos analisar da seguinte forma:

- r (Resíduos): Os coeficientes de cada termo fracionário simples na expansão de $X(z)$.
- p (Polos): As raízes do polinômio denominador $3z^2 - 4z + 1$, que são os valores de z onde o denominador se anula.
- c (Termos Diretos): Como estamos trabalhando com uma função que não possui termos de ordem superior ao denominador, esse vetor deve ser vazio (‘[]‘).

Além disso, a função residue permite que seja feita a reconstituição do Quociente dos Polinômios b(s)/a(s) a partir dos resíduos "r", polos "p", e o polinômio direto "c", conforme mostrado no item abaixo:

- linha 15: [b, a] = residue(r, p, c).

Portanto, os resultados obtidos podem ser usados diretamente para reescrever $X(z)$ na forma expandida, facilitando a aplicação da transformada Z inversa para encontrar $x(n)$ no domínio do tempo.

```
[b, a] = residue(r, p, c)
r =

   0.5000
  -0.5000

p =

   1.0000
   0.3333

c = [](0x0)
b = 0.3333
a =

   1.0000  -1.3333   0.3333

>>
```

Figura 1: Resultado da operação da questão 4.8

1.5 Exemplo 4.9

1.5.1 GNU Octave

```
1 clear all
2 close all
3
4 %* ===  FUNCTION
    ======================================================================

5 %         Name: impseq
6 %  Description: Funcao para gerar sequencia impulso.
7 %   Parametros: n0 = posicao do impulso; n1 = inicio da
    sequencia; n2 = fim da sequencia
8 %                x = sequencia de impulso; n = vetor de
    indices de tempo
```

```
%*
    ================================================================================

function [x, n] = impseq(n0, n1, n2)
  n = [n1:n2];
  x = (n-n0) == 0;
end

b = 1; % Definindo coeficiente do numerador

% Calculando os coeficientes do polinomio do denominador com
    a funcao poly
a = poly([0.9, 0.9, -0.9]);

% Realizando o calculo da decomposicao em fracoes parcias com
     a funcao residue
[r, p, c] = residue(b, a)

% ----------- Processamento para calcular a transformada Z
    inversa -----------

[delta, n] = impseq(0, 0, 7); % Gerando sequencia pulso [n =
    0 ate n = 7]
x = filter(b, a, delta)  % Aplicando filtro digital, e
    obtendo resposta do sistema

% Realizando o calculo da sequencia x(n) manualmente para
    verificar a sequencia
x_m = (0.75)*(0.9).^n + (0.5)*n.*(0.9).^n + (0.25)*(-0.9).^n;

% ----------- Plotagem para comparar as sequencias
    -----------

% Plotando a resposta do sistema obtida com o filtro digital
figure;
stem(n, x, 'b', 'filled');
hold on;
```

```
37
38 % Plotando a sequencia calculada manualmente
39 stem(n, x_m, 'r--');
40 hold off;
41
42 % Configuracoes do grafico
43 title('Compara  o entre a Resposta do Sistema e a
       Sequ ncia Calculada Manualmente');
44 xlabel('n');
45 ylabel('x(n)');
46 legend('Resposta do Sistema', 'Sequ ncia Manual');
47 grid on;
```

1.6 Exemplo 4.10

1.6.1 GNU Octave

```
1 clear all
2 close all
3
4 %* ===  FUNCTION
      ======================================================================

5 %         Name: impseq
6 %  Description: Funcao para gerar sequencia impulso.
7 %   Parametros: n0 = posicao do impulso; n1 = inicio da
      sequencia; n2 = fim da sequencia
8 %                x = sequencia de impulso; n = vetor de
      indices de tempo
9 %*
      ======================================================================

10 function [x, n] = impseq(n0, n1, n2)
11   n = [n1:n2];
12   x = (n-n0) == 0;
13 end
14
```

```
% Definindo numerador da funcao X(z)
b = [1, 0.4*sqrt(2)];

% Para remover os termos negativos de Z, multiplica-se o
    denominador por Z^2
a = [1, -0.8*sqrt(2), 0.64]; % Definindo os coeficientes do
    denominador

% Realizando o calculo da decomposicao em fracoes parcias com
     a funcao residue
[r, p, c] = residue(b, a)

% Realizando a conversao dos polos para a Forma Polar: Z = r.
    e^(j.theta)
magnitude = abs(p);   % Modulo r (magnitude polar)
angle_rad = angle(p); % Angulo theta (em radianos)

% ----------- Processamento para calcular a transformada Z
    inversa -----------

[delta, n] = impseq(0, 0, 7); % Gerando sequencia pulso [n =
    0 ate n = 7]
x = filter(b, a, delta)  % Aplicando filtro digital, e
    obtendo resposta do sistema

% Realizando o calculo da sequencia x(n) manualmente para
    verificar a sequencia
x_m = ((0.8).^n) .* (cos(pi*n/4) + 2*sin(pi*n/4));


% ----------- Plotagem grafica para comparar as sequencias
    -----------

% Plotando a resposta do sistema obtida com o filtro digital
figure;
stem(n, x, 'b', 'filled');
hold on;
```

```
43
44 % Plotando a sequencia calculada manualmente
45 stem(n, x_m, 'r--');
46 hold off;
47
48 % Configura  es do gr fico
49 title('Compara  o entre a Resposta do Sistema e a
       Sequ ncia Calculada Manualmente');
50 xlabel('n');
51 ylabel('x(n)');
52 legend('Resposta do Sistema', 'Sequ ncia Manual');
53 grid on;
```

1.7 Exemplo 4.11

1.7.1 Código Python

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy import signal
4
5 # Coeficientes do numerador e denominador de H(z)
6 b = [1, 0]  # Numerador
7 a = [1, -0.9]  # Denominador
8
9 # a) Diagrama de Polos e Zeros
10 zeros, poles, _ = signal.tf2zpk(b, a)
11
12 # b) Resposta em Frequencia |H(e^jw)| e H(e^jw)
13 w, h = signal.freqz(b, a, worN=200, whole=True)
14 magnitude = np.abs(h)
15 phase = np.angle(h)
16
17 # c) Resposta ao Impulso
18 impulse_response = signal.dimpulse((b, a, 1), n=20)
19
20 # Plotando tudo em um unico grafico com subplots
```

```
plt.figure(figsize=(18, 5))

# Subplot 1: Diagrama de Polos e Zeros
plt.subplot(1, 3, 1)
plt.plot(np.real(poles), np.imag(poles), 'x', markersize=10,
    label='Polos')
plt.plot(np.real(zeros), np.imag(zeros), 'o', markersize=10,
    label='Zeros')
plt.title('Pole-Zero Plot')
plt.xlabel('Real Part')
plt.ylabel('Imaginary Part')
plt.grid()
plt.axhline(0, color='black')
plt.axvline(0, color='black')
plt.legend()

# Subplot 2: Resposta de Magnitude
plt.subplot(1, 3, 2)
plt.plot(w/np.pi, magnitude)
plt.title('Magnitude Response')
plt.xlabel('Frequency (in pi units)')
plt.ylabel('Magnitude')
plt.grid()

# Subplot 3: Resposta de Fase
plt.subplot(1, 3, 3)
plt.plot(w/np.pi, phase)
plt.title('Phase Response')
plt.xlabel('Frequency (in pi units)')
plt.ylabel('Phase (radians)')
plt.grid()

# Exibindo os graficos
plt.tight_layout()
plt.show()

# Novo grafico para a Resposta ao Impulso
```

```
plt.figure(figsize=(6, 4))
plt.stem(np.arange(0, 20), np.squeeze(impulse_response[1]),
    use_line_collection=True)
plt.title('Impulse Response')
plt.xlabel('n')
plt.ylabel('h(n)')
plt.grid()
plt.show()
```

1.8 Exemplo 4.12

1.8.1 Código Python

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

# Coeficientes do numerador (z + 1) e denominador (z^2 - 0.9z
    + 0.81)
b = [1, 1]
a = [1, -0.9, 0.81]

# a) Diagrama de Polos e Zeros
zeros, poles, _ = signal.tf2zpk(b, a)

plt.figure(figsize=(18, 5))

plt.subplot(1, 3, 1)
plt.plot(np.real(poles), np.imag(poles), 'x', markersize=10,
    label='Polos')
plt.plot(np.real(zeros), np.imag(zeros), 'o', markersize=10,
    label='Zeros')
plt.title('Pole-Zero Plot')
plt.xlabel('Real Part')
plt.ylabel('Imaginary Part')
plt.grid()
plt.axhline(0, color='black')
```

```
plt.axvline(0, color='black')
plt.legend()

# b) Resposta em Frequencia |H(e^jw)| e H(e^jw)
w, h = signal.freqz(b, a)
magnitude = np.abs(h)
phase = np.angle(h)

plt.subplot(1, 3, 2)
plt.plot(w/np.pi, magnitude)
plt.title('Magnitude Response')
plt.xlabel('Frequency (in pi units)')
plt.ylabel('Magnitude')
plt.grid()

plt.subplot(1, 3, 3)
plt.plot(w/np.pi, phase)
plt.title('Phase Response')
plt.xlabel('Frequency (in pi units)')
plt.ylabel('Phase (radians)')
plt.grid()

plt.tight_layout()
plt.show()

# c) Resposta ao Impulso
t, h_impulse = signal.dimpulse((b, a, 1), n=20)

plt.figure(figsize=(6, 4))
plt.stem(t, np.squeeze(h_impulse), use_line_collection=True)
plt.title('Impulse Response')
plt.xlabel('n')
plt.ylabel('h(n)')
plt.grid()
plt.show()
```

1.9 Exemplo 4.13

1.9.1 Código Python

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

# Coeficientes do numerador e denominador da funcao de transferencia H(z)
b = [1, 0, -1]  # Coeficientes do numerador corretos
a = [1, 0, -0.81]  # Coeficientes do denominador

# a) Diagrama de Polos e Zeros
zeros, poles, _ = signal.tf2zpk(b, a)

plt.figure(figsize=(18, 5))

plt.subplot(1, 3, 1)
plt.plot(np.real(poles), np.imag(poles), 'x', markersize=10, label='Polos')
plt.plot(np.real(zeros), np.imag(zeros), 'o', markersize=10, label='Zeros')
plt.title('Pole-Zero Plot')
plt.xlabel('Real Part')
plt.ylabel('Imaginary Part')
plt.grid()
plt.axhline(0, color='black')
plt.axvline(0, color='black')
plt.legend()

# b) Resposta em Frequencia |H(e^jw)| e H(e^jw)
w, h = signal.freqz(b, a, worN=500)
magnitude = np.abs(h)
phase = np.angle(h)

plt.subplot(1, 3, 2)
```

```
plt.plot(w/np.pi, magnitude)
plt.title('Magnitude Response')
plt.xlabel('Frequency (in pi units)')
plt.ylabel('Magnitude')
plt.grid()

plt.subplot(1, 3, 3)
plt.plot(w/np.pi, phase)
plt.title('Phase Response')
plt.xlabel('Frequency (in pi units)')
plt.ylabel('Phase (radians)')
plt.grid()

plt.tight_layout()
plt.show()

# c) Resposta ao Impulso
t, h_impulse = signal.dimpulse((b, a, 1), n=20)

plt.figure(figsize=(6, 4))
plt.stem(t, np.squeeze(h_impulse), use_line_collection=True)
plt.title('Impulse Response')
plt.xlabel('n')
plt.ylabel('h(n)')
plt.grid()
plt.show()

# d) Resposta ao Degrau
t, h_step = signal.dstep((b, a, 1), n=20)

plt.figure(figsize=(6, 4))
plt.stem(t, np.squeeze(h_step), use_line_collection=True)
plt.title('Step Response')
plt.xlabel('n')
plt.ylabel('v(n)')
plt.grid()
plt.show()
```

1.10 Exemplo 4.15

1.10.1 Código Python

```
import numpy as np
from scipy import signal

# Coeficientes do numerador e denominador da equacao de
    diferenca
b = [1/3, 1/3, 1/3]  # Coeficientes de x(n)
a = [1, -0.95, 0.9025]  # Coeficientes de y(n)

# Condicoes iniciais
xic = [-2, -3]  # y(-1) e y(-2)

# Definindo a entrada x(n) = cos(pi*n/3) para n = 0 a 7
n = np.arange(0, 8)
x = np.cos(np.pi * n / 3)

# Calculando y(n) usando scipy.signal.lfilter
y = signal.lfilter(b, a, x, zi=signal.lfiltic(b, a, y=xic))

print("y(n) =", y[0])

# Verificacao Manual da Solucao
A = np.real(2 * x[0]) + np.imag(2 * x[1])
B = np.real(2 * x[1]) + np.imag(2 * x[0])
C = np.real(2 * x[2]) + np.imag(2 * x[3])
D = np.imag(2 * x[3])

# Calculando y usando a formula dada
y_manual = A * np.cos(np.pi * n / 3) + B * np.sin(np.pi * n /
    3) * ((0.95) ** -n) + \
           C * (np.cos(np.pi / 3) + D * np.sin(np.pi / 3))

print("y(n) verificacao manual =", y_manual)
```

2 Simulações Propostas

P4.4

A questão pede para que seja $x(n)$ uma sequência de valores complexos com a parte real $x_R(n)$ e a parte imaginária $x_I(n)$.

1. Prove as seguintes relações da transformada Z:

$$X_R(z) = \mathcal{Z}\{x_R(n)\} = \frac{X(z) + X^*(z^*)}{2}$$

$$X_I(z) = \mathcal{Z}\{x_I(n)\} = \frac{X(z) - X^*(z^*)}{2j}$$

2. Verifique essas relações para $x(n) = \exp\{(-1 + j0,2\pi)n\}u(n)$.

Resolução

Seja $x(n) = x_R(n) + jx_I(n)$, onde $x_R(n)$ é a parte real e $x_I(n)$ é a parte imaginária de $x(n)$. A Transformada Z de $x(n)$ é dada por $X(z)$, ou seja:

$$X(z) = \mathcal{Z}\{x(n)\} = \mathcal{Z}\{x_R(n) + jx_I(n)\} = X_R(z) + jX_I(z)$$

O conjugado complexo de $X(z)$, denotado por $X^*(z^*)$, é obtido trocando z pelo seu conjugado z^* e aplicando o conjugado complexo à função:

$$X^*(z^*) = X_R(z^*) - jX_I(z^*)$$

Provando as relações:

Prova da Relação para $X_R(z)$

Sabemos que a Transformada Z da parte real de $x(n)$, ou seja, $x_R(n)$, é $X_R(z)$. Podemos encontrar $X_R(z)$ somando $X(z)$ e $X^*(z^*)$:

$$X(z)+X^*(z^*) = (X_R(z)+jX_I(z))+(X_R(z^*)-jX_I(z^*)) = X_R(z)+X_R(z^*)+jX_I(z)-jX_I(z^*) = 2X_R($$

Dividindo ambos os lados por 2, obtemos a Transformada Z da parte real $x_R(n)$:

$$X_R(z) = \frac{X(z) + X^*(z^*)}{2}$$

Prova da Relação para $X_I(z)$

A Transformada Z da parte imaginária de $x(n)$, ou seja, $x_I(n)$, pode ser obtida subtraindo $X^*(z^*)$ de $X(z)$:

$$X(z)-X^*(z^*) = (X_R(z)+jX_I(z))-(X_R(z^*)-jX_I(z^*)) = X_R(z)-X_R(z^*)+jX_I(z)+jX_I(z^*) = 2jX_I$$

Dividindo ambos os lados por $2j$, obtemos a Transformada Z da parte imaginária $x_I(n)$:

$$X_I(z) = \frac{X(z) - X^*(z^*)}{2j}$$

Portanto, provamos que:

$$X_R(z) = \frac{X(z) + X^*(z^*)}{2} \quad \text{e} \quad X_I(z) = \frac{X(z) - X^*(z^*)}{2j}$$

Para verificar a relação para a sequência,

$$x(n) = \exp\left(\left(-1 + j0.2\pi\right) n\right) u(n)$$

foi utilizado Python.

Primeiramente, definimos a sequência exponencial $x(n) = \exp\left(\left(-1 + j0.2\pi\right) n\right) u(n)$. Em seguida, foi utilizada a função **summation** para calcular a transformada de x(n).

Nas linha 14 é calculados o conjugado de X(z) e nas linhas 17 e 20 o conjugado é usado para encontrar a parte real e imaginária, respectivamente.

Como saída, obtemos a representação das partes real e imaginária da transformada Z.

```python
import sympy as sp
from sympy import I, exp, pi, symbols, conjugate

# Definindo as vari veis simb licas
n, z = symbols('n z', real=True)

# Definindo x(n) = exp{((-1 + j0.2  )n)} u(n)
x_n = exp((-1 + I * 0.2 * pi) * n)

# Definindo a transformada z de x(n)
X_z = sp.summation(x_n * z**-n, (n, 0, sp.oo))

# Calculando o conjugado complexo de X(z)
X_conj_z_conj = conjugate(X_z.subs(z, conjugate(z)))

# Parte real X_R(z)
X_R_z = (X_z + X_conj_z_conj) / 2

# Parte imagin ria X_I(z)
X_I_z = (X_z - X_conj_z_conj) / (2 * I)

# Exibindo as express es simplificadas
X_R_z_simplified = sp.simplify(X_R_z)
X_I_z_simplified = sp.simplify(X_I_z)

X_R_z_simplified, X_I_z_simplified
```

P4.5

O problema 4.5 pede para determinar a transformada Z das sequências dadas e a região de convergência, dado que a transformada de x(n) é:

$$X(z) = \frac{1}{(1 + 0.5z^{-1})}$$

Para encontrar a transformada Z de cada uma das sequências dadas, foram usadas algumas propriedades para manipulação e reformulação da expressão. As simulações foram realizadas em linguagem Python utilizando o JupyterLab.

Para as simulações usando Python foram utilizadas algumas funções da biblioteca **Symply**.

- **symbols:** Função utilizada para criar variáveis simbólicas. Utilizamos ela para representar a variável z da transformada.
- **simplify:** Utilizada para simplificar expressões, combinando termos semelhantes, manipulando expoentes e simplificando frações. Realiza as manipulações que fazemos ao determinar as transformadas de forma manual.
- **solve:** Função utilizada para resolver equações, encontrando os valores que a satisfazem.

A Região de Convergência(ROC) é a região onde a transformada Z converge, ou seja, onde a série geométrica que representa o sinal é convergente e depende das propriedades do sinal e da função da transformada.

Para encontrar a ROC usando Python, primeiramente foi calculado os pólos do denominador usando a função **solve**, citada anteriormente, para encontrar os valores de z para o qual a equação resulta em zero. A partir do valor encontrado para os pólos, determinando a ROC.

A determinação das transformadas para cada uma das sequências são apresentadas a seguir.

1. $x_1(n) = x(3-n) + x(n-3)$

   Primeiramente foi definida a variável z, que será utilizada para representar a função $X(z)$ e da definição da mesma. Em seguida, foi feita

a substituição de z por 1/z, para calcular a transformada inversa da função. Após isso somamos X(z) com a inversa e multiplicamos por z^{-3} para representar o deslocamento de três unidades de tempo ocorrida na sequência(linha 13). Por fim, expadimos a funçao e simplificamos para encontrar o resultado final.

O resultado obtido para $X_1(z)$ foi: $\frac{0.5z^2+2z+0.5}{z^3(0.5z+1)(z+0.5)}$

```
1  import sympy as sp
2
3  # Definir a variavel z
4  z = sp.symbols('z')
5
6  # Definir a transformada Z original X(z)
7  X_z = 1 / (1 + 0.5 * z**-1)
8
9  # Calcular X(1/z)
10 X_inv_z = X_z.subs(z, 1/z)
11
12 # Definir a transformada Z de x1(n) = x(3-n) + x(n-3)
13 X1_z = z**-3 * (X_inv_z + X_z)
14
15 # Expandir e simplificar a expressao completamente
16 X1_z_simplified = sp.simplify(sp.expand(X1_z))
17
18 # Exibir o resultado
19 sp.pretty_print(X1_z_simplified)
20
21 #saida
22
23 0.5z^2 + 2z + 0.5/z^3(0.5z+1)(z + 0.5)
```

2. $x_2(n) = (1 + n + n^2)x(n)$

Para determinar a transformada da sequncia dada neste item, primeiramente foi realizado o passo padrão para definição da variável z e função

$X(z)$.

Em seguida, foi calculado a derivada primeira da função $X(z)$ usando a função **diff**. Isso é feito para calcular a taxa de variação de X(z) em relação a z. Em seguida foi calculada a derivada segunda da função resultante da primeira derivada.

Por último, calculamos a trasnformada Z(linha 14) como sendo a soma da função original X(z) com a derivada segunda e subtraída pela derivada primeira. Após isso, simplificamos a expressão para encontrar o resultado final.

O resultado obtido para $X_2(z)$ foi: $\frac{z^2(z-0.5)}{(z+0.5)^3}$

```
1  from sympy import symbols, diff
2
3  # Define the symbols
4  z = symbols('z')
5  X_z = 1 / (1 + 0.5 * z**-1)
6
7  # Calculo da primeira derivada em relacao a z
8  dX_dz = diff(X_z, z)
9
10 # Calculo da segunda derivada em relacao a z
11 d2X_dz2 = diff(dX_dz, z)
12
13 # Expressao da funcao X2(z)
14 X2_z = X_z - z * dX_dz + z**2 * d2X_dz2
15
16 # Simplificacao
17 X2_z_simplified = X2_z.simplify()
18
19 # Resultado
20 X2_z_simplified
```

3. $x_3(n) = \left(\frac{1}{2}\right)^n x(n-2)$

Para essa sequência primeiramente foi definida a variável z, que será utilizada para representar a função $X(z)$.

Em seguida definimos a função $X(z)$ como dado no enunciado da questão(linha 7). z^{-1} representa o deslocamento no tempo e nesse caso, o -1 representa o atraso em uma unidade de tempo.

Na linha 10, substituimos o z por $\frac{z}{2}$, que corresponde a multiplicação da sequncia original x(n) por $(\frac{1}{2})^2$. Em seguida multiplicamos a sequência por z^{-2}, representado o deslocamento de (n-2).

Por fim, simplificamos a expressão para encontrar $X_3(z)$.

O resultado obtido para $X_3(z)$ foi: $\frac{1}{z\cdot(z+1)}$

```
1  from sympy import symbols, simplify
2
3  # Definir a vari vel z
4  z = symbols('z')
5
6  # Definir a fun   o X(z) = 1 / (1 + 0.5 * z**(-1))
7  X_z = 1 / (1 + 0.5 * z**(-1))
8
9  #Substituir z por z/2 para o termo (1/2)^n
10 X3_z_substituted = X_z.subs(z, z/2)
11
12
13 # Multiplica por z^(-2) para o deslocamento n-2
14 X3_z_shifted = z**(-2) * X3_z_substituted
15
16
17 #  Simplificacao
18 X3_z_simplified = simplify(X3_z_shifted)
```

4. $x_4(n) = x(n+2) * x(n-2)$

Primeiramente, assim como feito nos itens anteriores, definimos z e

X(z). Em seguida modificado X(z). Na sequência dada $x_4(n) = x(n+2) * x(n-2)$
, temos que $(n+2)$ corresponde a uma muliplicação por z^2 e $(n-2)$ corresponde a uma multiplicação por z^{-2}, representando o atraso em duas unidades de tempo. Para fazer essa representação no Python, multiplicamos a função original X(z) por z^2, representando o deslocamento (n+2) e elevamos o resultado da multiplicação ao quadrado, que representação o deslocamento (n-2). Por último, simplificamos a expressão para encontrar o resultado final.

Para determinar a região de convergência, a função X(z) foi reescrita de uma forma mais simples para em seguida encontrar os pólos, ou seja, os valores para z que zeram o denominador, usando a função **solve**.

O resultado obtido para $X_4(z)$ foi: $\frac{z^6}{(z+0.5)^2}$

Os polos obtidos foram [-0.5], logo, a ROC será $|z| > 0.5$

```
import sympy as sp
from sympy import symbols, solve, Eq

# Definindo a vari vel z
z = sp.symbols('z')

# Fun   o X(z)
X_z = 1 / (1 + 0.5 * z**-1)

# Modificando X(z)
X4_z = (z**2 * X_z)**2

# Exibindo a express o final
X4_z_simplified = sp.simplify(X4_z)

X4_z_simplified

# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
# Calculo dos pelos e regiao de convergencia
z = symbols('z')

# Fun   o original
X_z = 1 / (1 + 0.5 * z**-1)

# Reescrevendo X(z)
X_z_rewritten = z / (z + 0.5)

# Calculando os polos (denominador = 0)
denominator = z + 0.5
poles = solve(Eq(denominator, 0), z)

# Regi o de converg ncia (fora do m dulo do polo
    principal)
roc = f'|z| > {abs(poles[0])}'

poles, roc
```

5. $x_5(n) = \cos(\frac{\pi n}{2}) * x(n)$

   Para determinar a transformada Z da sequência, primeiramente definimos a variável z. Em seguida representamos o termo $\cos(\frac{\pi n}{2}$ na forma de exponenciais, representando tanto a parte negativa quanto a positiva. O termo $\frac{314159}{2}$ representa o valor aproximado para $\frac{\cos}{2}$.

   Nas linhas 12 e 13 é representado a transformada Z da função z[n], sendo uma para a exponencial positiva e outra para a exponencial negativa. O termo z^{-1} representa o atraso em uma unidade de tempo.

   Em seguida, combinamos as expressões para formar a função da trasnformada Z para a sequência. Por fim, simplificamos as expressões e calculamos a região de convergência seguindo os passos já descritos nos itens anteriores.

   O resultado obtido para $X_5(z)$ foi: $\frac{z^2}{1+\frac{0.25}{z^2}}$

Os polos obtidos foram [-0.5i, 0.5i], logo, a ROC será $|z| > 0.5$

```
from sympy import symbols, exp, I, simplify, solve, Eq


# Definindo z como vari vel simb lica
z = symbols('z')

# Definindo as express es exponenciais para cos(pi*n/2)
exp_pos = exp(I * z * (3.14159 / 2))
exp_neg = exp(-I * z * (3.14159 / 2))

# Definindo X(z) como a fun  o transformada
X_z_pos = 1 / (1 + 0.5 * exp_pos * z**-1)
X_z_neg = 1 / (1 + 0.5 * exp_neg * z**-1)

# Combina  o linear das duas express es
X_z = (X_z_pos + X_z_neg) / 2

# Simplificando a express o final
X_z_simplified = simplify(X_z)


# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

z = symbols('z')

# Fun  o transformada Z j  simplificada (express o
    obtida da transformada Z)
X_z = 1 / (1 + 0.25 * z**-2)

# Multiplicar ambos os lados por z^2 para remover as
    pot ncias negativas
X_z_simplified = X_z * z**2

# Determinar os polos resolvendo a equa  o 1 + 0.25z
    ^(-2) = 0
```

```
denominator = 1 + 0.25 * z**-2

# Encontrar os polos resolvendo a equa  o denominador =
    0
poles = solve(Eq(denominator, 0), z)

roc = f'|z| > {abs(poles[0])}'
X_z_simplified, roc
```

P4.6

O problema 4.6 pede para repetir o que foi feito na questão agora, calcular a transformada z para as sequências dadas, porém agora com X(z) sendo:

$$X(z) = \frac{1 + z^{-1}}{1 + \frac{5}{6}z^{-1} + \frac{1}{6}z^{-2}}$$

Para cada uma das sequências dadas foram utilizadas as mesmas funções e passos realizados no problema P4.5.

1. $x_1(n) = x(3 - n) + x(n - 3)$

```
import sympy as sp


# Definindo a transformada original
X_z = (1 + z**-1) / (1 + 0.5 * z**-1)

# Expressoes de Z[x(3-n)] e Z[x(n-3)]
X1_z = z**-3 * X_z + z**3 * X_z

# Simplificacao
X1_z_simplified = sp.simplify(X1_z)
X1_z_simplified
```

O valor encontrada para $X_1(z)$ foi: $\frac{(z+1)(z^6+1)}{z^3(z+0.5)}$

2. $x_2(n) = (1 + n + n^2)x(n)$

```
# Definir a express o de X(z)
X_z = (1 + z**-1) / (1 + 0.5 * z**-1)

# Primeira e segunda derivadas em rela   o a z
dX_dz = sp.diff(X_z, z)
d2X_dz2 = sp.diff(dX_dz, z)

# Resultado final da transformada
X2_z = z * dX_dz + z**2 * d2X_dz2
X2_z
```

O valor encontrada para $X_2(z)$ foi:

]:
$$z^2 \cdot \left(\frac{2}{z^3 \cdot \left(1 + \frac{0.5}{z}\right)} - \frac{1.0 \cdot \left(1 + \frac{1}{z}\right)}{z^3 \left(1 + \frac{0.5}{z}\right)^2} - \frac{1.0}{z^4 \left(1 + \frac{0.5}{z}\right)^2} + \frac{0.5 \cdot \left(1 + \frac{1}{z}\right)}{z^4 \left(1 + \frac{0.5}{z}\right)^3} \right) + z \left(-\frac{1}{z^2 \cdot \left(1 + \frac{0.5}{z}\right)} + \frac{0.5 \cdot \left(1 + \frac{1}{z}\right)}{z^2 \left(1 + \frac{0.5}{z}\right)^2} \right)$$

3. $x_3(n) = \left(\frac{1}{2}\right)^n x(n-2)$

```
# Definir o deslocamento temporal (2 unidades)
X3_z = (1/2)**z * X_z.subs(z, z**-2)
X3_z_simplified = sp.simplify(X3_z)
X3_z_simplified
```

O valor encontrada para $X_3(z)$ foi: $\frac{0.5^z(z^2+1)}{0.5z^2+1}$

4. $x_4(n) = x(n+2) * x(n-2)$

```
# Multiplica   o das express es deslocadas
X4_z = z**2 * X_z * z**-2 * X_z
X4_z_simplified = sp.simplify(X4_z)
X4_z_simplified
```

O valor encontrada para $X_4(z)$ foi: $\frac{(z+1)^2}{(z+0.5)^2}$

5. $x_5(n) = \cos(\frac{\pi n}{2}) * x(n)$

```
# Usar a propriedade de cosseno e a express o X(z)
X5_z = (1/2) * (X_z.subs(z, sp.exp(sp.I * sp.pi / 2) * z
    **-1) + X_z.subs(z, sp.exp(-sp.I * sp.pi / 2) * z**-1)
    )
X5_z_simplified = sp.simplify(X5_z)
X5_z_simplified
```

O valor encontrada para $X_5(z)$ foi: $\frac{0.5z^2+1}{0.25z^2+1}$

P4.8

O problema 4.8 pede para provar as sequênciasx$_1(n)$, $x_2(n)$ e $x_3(n)$ estão relacionadas por $x_3(n) = x_1(n)\ x_2(n)$, então é fornecida a seguinte proprieade:

$$\sum_{n=-\infty}^{\infty} x_3(n) = \left(\sum_{n=-\infty}^{\infty} x_1(n)\right)\left(\sum_{n=-\infty}^{\infty} x_2(n)\right),$$

Espera-se que faça a prova utilizando a definição de covolução no lado esquedo, e a propriedade de covolução da transformada Z, para validar a equação e, em seguida, simular utilizando o Matlab ou Octave.

A convolução $x_3(n) = x_1(n) * x_2(n)$ é definida como a soma dos produtos das sequências $x_1(n)$ e $x_2(n)$ com um deslocamento k. Esta abordagem confirma que, pela definição de convolução, a soma infinita da sequência convoluída é igual ao produto das somas infinitas das sequências originais. Na figura 2 é apresentado a definição de convolução no lado esquerdo.

Ao aplicar a propriedade de convolução da Transformada Z, a equação dada é confirmada ao mostrar que o somatório infinito de $x_3(n)$ corres-

$$\sum_{n=-\infty}^{\infty} x_3(n) = \sum_{n=-\infty}^{\infty} x_1(n) * x_2(n) = \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} x_1(k)x_2(n-k) = \left(\sum_{k=-\infty}^{\infty} x_1(k)\right) \sum_{n=-\infty}^{\infty} x_2(n-k)$$
$$= \left(\sum_{n=-\infty}^{\infty} x_1(n)\right)\left(\sum_{n=-\infty}^{\infty} x_2(n)\right)$$

Figura 2: Definição de convolução no lado esquerdo

ponde ao produto dos somatórios infinitos de $x_1(n)$ e $x_2(n)$. Isso prova que a equação é consistente com a propriedade de convolução da Transformada Z. Na figura 3 é apresentado a propriedade de convolução da Transformada Z.

$$\mathcal{Z}[x_3(n)] = \mathcal{Z}[x_1(n) * x_2(n)] = \mathcal{Z}[x_1(n)]\,\mathcal{Z}[x_2(n)]$$
$$\left(\sum_{n=-\infty}^{\infty} x_3(n)z^{-n}\right)\Bigg|_{z=1} = \left(\sum_{n=-\infty}^{\infty} x_1(n)z^{-n}\right)\Bigg|_{z=1} \left(\sum_{n=-\infty}^{\infty} x_2(n)z^{-n}\right)\Bigg|_{z=1}$$
$$\sum_{n=-\infty}^{\infty} x_3(n) = \left(\sum_{n=-\infty}^{\infty} x_1(n)\right)\left(\sum_{n=-\infty}^{\infty} x_2(n)\right)$$

Figura 3: Propriedade de convolução da Transformada Z

Ou seja, Ambas as provas, usando a definição de convolução e a propriedade de convolução da Transformada Z, validam a equação proposta. Elas mostram que o resultado é uma consequência natural das propriedades matemáticas associadas à convolução de sequências e suas representações no domínio da Transformada Z.

Para implementar essa verificação no Octave, seguimos os seguintes passos:

- **Definir o Comprimento das Sequências**: Determinamos um tamanho suficientemente grande para as sequências, de modo a aproximar uma soma infinita. Para isso, definimos o comprimento

$N = 1000$, resultando em sequências que possuem 1001 elementos, variando de 0 a 1000. Essa escolha busca minimizar erros de truncamento e garantir uma precisão adequada na simulação da convolução infinita.

- **Gerar Sequências Aleatórias**: Utilizamos a função `rand` do Octave para criar duas sequências de números aleatórios, $x_1(n)$ e $x_2(n)$, ambas de comprimento $N + 1$. Essa geração de sequências aleatórias permite testar a propriedade de convolução de maneira geral, sem depender de valores específicos.
- **Calcular a Convolução das Sequências**: A função customizada `conv_m` é utilizada para calcular a convolução entre $x_1(n)$ e $x_2(n)$. Essa função não apenas executa a convolução das duas sequências, mas também calcula o suporte temporal da sequência resultante $x_3(n)$, facilitando a análise subsequente.
- **Verificar a Propriedade da Convolução**: Calculamos a soma de todas as amostras das sequências $x_1(n)$, $x_2(n)$ e $x_3(n)$. Comparamos então a soma da sequência convoluída $x_3(n)$ com o produto das somas das sequências individuais $x_1(n)$ e $x_2(n)$. A propriedade da convolução é verificada se a diferença entre essas quantidades for mínima, demonstrando que o produto das somas das sequências originais é igual à soma da convolução das mesmas.

A verificação é considerada bem-sucedida quando o erro (a diferença entre o produto das somas individuais e a soma da convolução) é próximo de zero, indicando que a propriedade é mantida dentro da precisão numérica do Octave.

2.0.1 GNU Octave

```
1 clear all
```

```
close all

%
    ================================================================================

% conv_m - Funcao de covolucao modificada para
    processamento de sinais.
%
% Sintaxe:
%   [y, ny] = conv_m(x, nx, h, nh)
%
% Entradas:
%   x  - Primeira sequencia (sinal) a ser convoluida.
%   nx - Suporte da primeira sequencia (vetor de indices
    de tempo).
%   h  - Segunda sequencia (sinal) a ser convoluida.
%   nh - Suporte da segunda sequencia (vetor de indices
    de tempo).
%
% Saidas:
%   y  - Resultado da covolucao das duas sequencias.
%   ny - Suporte da sequencia convoluida (vetor de
    indices de tempo para 'y').
%
% Descricao:
%   A funcao `conv_m` realiza a covolucao de duas
    sequencias discretas `x` e `h`,
%   e calcula o suporte da sequencia resultante. Esta
    funcao se faz util em
%   aplicacoes de processamento de sinais onde o
    alinhamento temporal das
%   sequencias sao necessarias.
%
% Exemplos:
%   x = [1, 2, 3]; nx = [0, 1, 2];
%   h = [4, 5, 6]; nh = [0, 1, 2];
%   [y, ny] = conv_m(x, nx, h, nh);
```

```
30 %
    ================================================================================

31 function [y, ny] = conv_m(x, nx, h, nh)
32   % Calcula o inicio e o fim do suporte da sequencia
      resultante da convolucao
33   nyb = nx(1) + nh(1);  % Inicio do suporte de y
34   nye = nx(end) + nh(end); % Fim do suporte de y
35
36   % Cria o vetor de indices de tempo para a sequencia
      covoluida
37   ny = nyb:nye;
38
39   % Calcula a convolucao das duas sequencias usando a
      funcao 'conv'
40   y = conv(x, h);
41 end
42
43 N = 1000; % Definindo o tamanho das sequencias
44
45 % Criando o vetor de indices para as sequencias de 0 a N
46 n1 = [0:N];
47 n2 = [0:N];
48
49 % Gerando duas sequencias aleatorias x1 e x2 de
     comprimento N+1
50 x1 = rand(1, length(n1));
51 x2 = rand(1, length(n2));
52
53 % Realizando o calculo da covolucao das sequencias
54 [x3, n3] = conv_m(x1, n1, x2, n2);
55
56 % Calculando a soma de todas as amostras das sequencias
     x1, x2 e x3
57 sumx1 = sum(x1); % Soma da sequencia x1
58 sumx2 = sum(x2); % Soma da sequencia x2
59 sumx3 = sum(x3); % Soma da sequencia resultante x3 (
```

```
    convolucao)
60
61 % Calculando o erro como a diferenca maxima absoluta
    entre a soma da Convolucao
62 % e o produto das somas das sequencias individuais x1 e
    x2
63 error = max(abs(sumx3-sumx1*sumx2))
```

Primeiramente, foi realizada a limpeza de todas as variáveis do ambiente de trabalho do Octave, e de todas as janelas de figuras que possam estar abertas. Para realizar o cálculo da convolução, é utilizada a função customizada `conv_m`. Esta função calcula a convolução de duas sequências e determina o suporte (ou índice de tempo) da sequência resultante. A seguir, detalhamos a função `conv_m` e seus parâmetros:

Entradas e Saídas da Função `conv_m`

A função recebe os seguintes parâmetros e retorna os seguintes valores:

- **Entradas:**
  - `x`: Primeira sequência (sinal) a ser convoluída.
  - `nx`: Suporte da primeira sequência (vetor de índices de tempo).
  - `h`: Segunda sequência (sinal) a ser convoluída.
  - `nh`: Suporte da segunda sequência (vetor de índices de tempo).
- **Saídas:**
  - `y`: Resultado da convolução das duas sequências.
  - `ny`: Suporte da sequência convoluída (vetor de índices de tempo para `y`).

Cálculo do Suporte Resultante

A função `conv_m` calcula o suporte da sequência resultante da convolução, executando os seguintes passos:

- `nyb = nx(1) + nh(1)`: Define o início do suporte da sequência resultante. Este é calculado como a soma do início dos suportes das sequências `x` e `h`.
- `nye = nx(length(x)) + nh(length(h))`: Define o fim do suporte da sequência resultante. Este é calculado como a soma dos últimos índices dos suportes das sequências `x` e `h`.
- `ny = [nyb:nye]`: Cria o vetor de índices de tempo para a sequência convoluída, abrangendo do início `nyb` até o fim `nye`.

Cálculo da Convolução

Para calcular a convolução completa das sequências, ou seja, a multiplicação polinomial das duas sequências, a função `conv_m` utiliza a função `conv` do Octave.

- `y = conv(x, h)`: Calcula o resultado da convolução das duas sequências `x` e `h`.

A função `conv` do Octave é usada para calcular a convolução de dois vetores. Esta operação pode representar tanto a multiplicação de polinômios quanto a operação de convolução em processamento de sinais. Quando `a` e `b` são vetores de coeficientes de dois polinômios, a convolução representa o vetor de coeficientes do polinômio produto.

- **Definição do comprimento das sequências**: Na linha 39, o comprimento das sequências a serem geradas é definido. Aqui, `N` é definido como 1000, o que significa que as sequências terão 1001 elementos (de 0 a 1000). Este valor é escolhido para garantir uma sequência suficientemente longa para o processamento de sinais.
- **Criação dos vetores de índices**: Nas linhas 48 e 49, são criados dois vetores de índices que variam de 0 a `N` para as duas sequências

`x1` e `x2`. Esses vetores de índices são necessários para definir o domínio de tempo discreto em que as sequências são definidas.

Para gerar duas sequências de números aleatórios, a função `rand` é utilizada, gerando cada sequência com comprimento `N+1`:

- **Linha 52**: `x1 = rand(1, length(n1));` - Gera a primeira sequência aleatória `x1`.
- **Linha 53**: `x2 = rand(1, length(n2));` - Gera a segunda sequência aleatória `x2`.

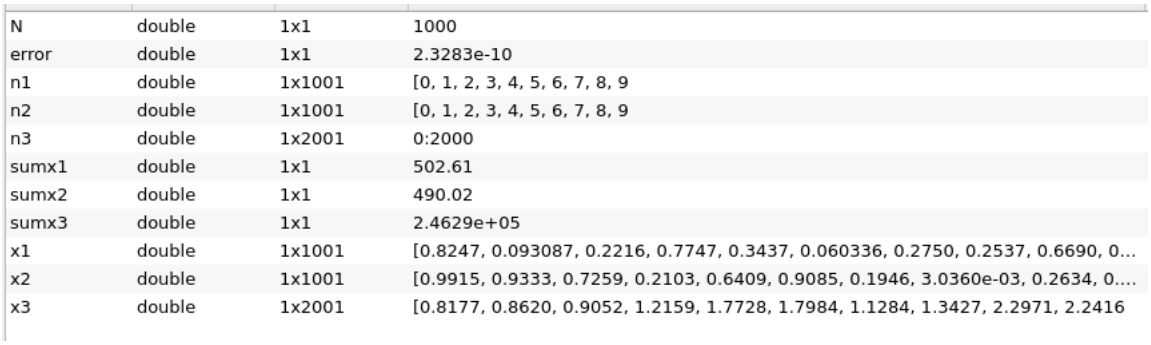
Para calcular a convolução das sequências, é utilizada a função `conv_m`, que é uma versão modificada da função de convolução para sinais discretos:

- **Linha 56**: `[x3, n3] = conv_m(x1, n1, x2, n2);` - Executa a convolução das sequências `x1` e `x2` utilizando seus respectivos vetores de índice. A função retorna a sequência convoluída `x3` e seu suporte `n3`.

Para calcular o erro e verificar a propriedade da convolução, a diferença absoluta máxima entre `sumx3` e o produto `sumx1 * sumx2` é computada. Isso é feito para garantir que o resultado da convolução satisfaça a propriedade esperada da transformada Z:

- **Linha 63**: `error = max(abs(sumx3 - sumx1 * sumx2));` - Calcula o erro como a diferença absoluta máxima entre a soma da sequência convoluída e o produto das somas das sequências individuais. A função `max(abs(...))` é utilizada para capturar o valor máximo da diferença, levando em consideração possíveis variações devido a erros de precisão numérica.

Na Figura 4, observamos que o erro calculado é extremamente pequeno (2.3283×10^{-10}), indicando que a propriedade está validada dentro da precisão numérica do Octave.



N	double	1x1	1000
error	double	1x1	2.3283e-10
n1	double	1x1001	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9
n2	double	1x1001	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9
n3	double	1x2001	0:2000
sumx1	double	1x1	502.61
sumx2	double	1x1	490.02
sumx3	double	1x1	2.4629e+05
x1	double	1x1001	[0.8247, 0.093087, 0.2216, 0.7747, 0.3437, 0.060336, 0.2750, 0.2537, 0.6690, 0...
x2	double	1x1001	[0.9915, 0.9333, 0.7259, 0.2103, 0.6409, 0.9085, 0.1946, 3.0360e-03, 0.2634, 0....
x3	double	1x2001	[0.8177, 0.8620, 0.9052, 1.2159, 1.7728, 1.7984, 1.1284, 1.3427, 2.2971, 2.2416

Figura 4: Resultado da operação do problema 4.8

Portanto, a verificação confirma que a soma da convolução é aproximadamente igual ao produto das somas das sequências individuais, conforme previsto pela teoria da convolução e pela propriedade da transformada Z. Esse resultado valida a precisão da implementação computacional e a conformidade com os princípios teóricos.

P4.9

O Problema 4.9 solicita a simulação de cada item utilizando operações polinomiais no Matlab ou GNU Octave. Para isso, utilizamos a função `conv`, que realiza a convolução dos coeficientes dos polinômios, o que corresponde à multiplicação dos polinômios.

2.0.2 GNU Octave

Antes de começar, é necessário identificar os coeficientes de cada polinômio para os itens do Problema 4.9.

1.

$$X_1(z) = (1 - 2z^{-1} + 3z^{-2} - 4z^{-3})(4 + 3z^{-1} - 2z^{-2} + z^{-3})$$

Definindo os coeficientes:

- O polinômio $1-2z^{-1}+3z^{-2}-4z^{-3}$ possui os coeficientes $[1, -2, 3, -4]$.
- O polinômio $4+3z^{-1}-2z^{-2}+z^{-3}$ possui os coeficientes $[4, 3, -2, 1]$.

Para realizar a multiplicação dos polinômios, utilizamos o código a seguir:

```
% Definindo os coeficientes dos polin mios
P1 = [1, -2, 3, -4]; % Coeficientes de 1 - 2z^-1 + 3z^-2
    - 4z^-3
```

```
P2 = [4, 3, -2, 1];  % Coeficientes de 4 + 3z^-1 - 2z^-2
    + z^-3

% Multiplicando os polin mios
X1 = conv(P1, P2);

% Exibindo o resultado
disp('Resultado de X1(z):');
disp(X1);
```

Após a execução no Octave, o vetor `X1` contém os coeficientes do polinômio resultante da multiplicação. O resultado é:

$$X_1(z) = 4 - 5z^{-1} + 12z^{-2} - 14z^{-3} - z^{-4} + 10z^{-5} - 4z^{-6}$$

O vetor de coeficientes para $X_1(z)$ é $[4, -5, 12, -14, -1, 10, -4]$, representando o polinômio final.

2. Para o segundo item, multiplicamos os polinômios:

$$X_2(z) = (z^2 - 2z + 3 + 2z^{-1} + z^{-2})(z^3 - z^{-3})$$

Definindo os coeficientes:

- O polinômio $z^2 - 2z + 3 + 2z^{-1} + z^{-2}$ possui os coeficientes $[1, -2, 3, 2, 1]$.
- O polinômio $z^3 - z^{-3}$ possui os coeficientes $[1, 0, 0, 0, 0, -1]$.

Utilizando o seguinte código:

```
% Definindo os coeficientes dos polin mios
P3 = [1, -2, 3, 2, 1];  % Coeficientes de z^2 - 2z + 3 +
    2z^-1 + z^-2
P4 = [1, 0, 0, 0, 0, -1]; % Coeficientes de z^3 - z^-3

% Multiplicando os polin mios
```

```
X2 = conv(P3, P4);

% Exibindo o resultado
disp('Resultado de X2(z):');
disp(X2);
```

O resultado da multiplicação é:

$$X_2(z) = z^5 - 2z^4 + 3z^3 + 2z^2 - z^{-1} + 2z^{-2} - z^{-3}$$

O vetor `X2` resultante possui 8 coeficientes $[1, -2, 3, 2, 0, -1, 2, -1]$, representando o polinômio final.

3. Para o terceiro item, elevamos o polinômio $1 + z^{-1} + z^{-2}$ ao cubo:

$$X_3(z) = (1 + z^{-1} + z^{-2})^3$$

Os coeficientes do polinômio $1 + z^{-1} + z^{-2}$ são $[1, 1, 1]$. Para elevar o polinômio ao cubo, usamos a função `conv` para multiplicá-lo por si mesmo três vezes.

```
% Definindo os coeficientes do polin mio
P5 = [1, 1, 1];  % Coeficientes de 1 + z^-1 + z^-2

% Elevando o polin mio ao cubo
X3 = conv(conv(P5, P5), P5);

% Exibindo o resultado
disp('Resultado de X3(z):');
disp(X3);
```

Após a execução, obtemos o polinômio:

$$X_3(z) = 1 + 3z^{-1} + 6z^{-2} + 7z^{-3} + 6z^{-4} + 3z^{-5} + z^{-6}$$

O vetor resultante de coeficientes é $[1, 3, 6, 7, 6, 3, 1]$.

4. Para o quarto item, a expressão a ser calculada é:

$$X_4(z) = X_1(z)X_2(z) + X_3(z)$$

Definimos os coeficientes previamente calculados:

- $X_1(z)$: $[4, -5, 12, -14, -1, 10, -4]$
- $X_2(z)$: $[1, -2, 3, 2, 0, -1, 2, -1]$
- $X_3(z)$: $[1, 3, 6, 7, 6, 3, 1]$

Usamos o código a seguir para realizar a multiplicação e soma:

```
% Coeficientes de X1(z), X2(z), e X3(z)
X1 = [4, -5, 12, -14, -1, 10, -4];
X2 = [1, -2, 3, 2, 1, 0, -1, 2, -3, -2, -1];
X3 = [1, 3, 6, 7, 6, 3, 1];

% Multiplicando X1(z) e X2(z)
X1_X2 = conv(X1, X2);

% Somando o resultado com X3(z)
% Precisamos alinhar os tamanhos dos vetores
X3_aligned = [X3, zeros(1, length(X1_X2) - length(X3))];
X4 = X1_X2 + X3_aligned;

% Exibindo o resultado
disp('Resultado de X4(z):');
disp(X4);
```

O vetor `X4` contém os coeficientes resultantes da multiplicação e soma dos polinômios.

5. No quinto item, multiplicamos os polinômios:

$$X_5(z) = (z^{-1} - 3z^{-3} + 2z^{-5} + 5z^{-7} - z^{-9})(z + 3z^2 + 2z^3 + 4z^4)$$

Definimos os coeficientes:

- O polinômio $z^{-1} - 3z^{-3} + 2z^{-5} + 5z^{-7} - z^{-9}$ possui os coeficientes $[0, 1, 0, -3, 0, 2, 0, 5, 0, -1]$.
- O polinômio $z + 3z^2 + 2z^3 + 4z^4$ possui os coeficientes $[0, 1, 3, 2, 4]$.

Usamos a função `conv` para multiplicá-los:

```
% Definindo os coeficientes dos polin mios
P6 = [0, 1, 0, -3, 0, 2, 0, 5, 0, -1]; % Coeficientes de
    z^-1 - 3z^-3 + 2z^-5 + 5z^-7 - z^-9
P7 = [0, 1, 3, 2, 4]; % Coeficientes de z + 3z^2 + 2z^3 +
     4z^4

% Multiplicando os polin mios
X5 = conv(P6, P7);

% Exibindo o resultado
disp('Resultado de X5(z):');
disp(X5);
```

O vetor `X5` contém os coeficientes do polinômio resultante após a multiplicação.

P4.10

A função `deconv_m` foi criada como uma variante da função `deconv`. A função `deconv_m` é projetada para realizar a deconvolução (divisão de polinômios) de sequências não causais ($n < 0$) e dá suporte aos seus limites de variância.

A função `deconv_m` realiza a deconvolução de duas sequências não causais, retornando o quociente e o resto, juntamente com os suportes correspondentes.

A função `deconv_m` segue os seguintes passos principais:

- **Entrada:** A função recebe os coeficientes do numerador b e denominador a das sequências, bem como seus respectivos suportes nb e na.
- **Saída:** A função retorna o quociente p e o resto r_full, além dos suportes np e nr associados.
- **Lógica Principal:**
  - O início do suporte do quociente é calculado como $np1 = nb(1) - na(1)$.
  - A deconvolução é realizada usando a função interna `deconv`, que retorna o quociente e o resto.
  - O suporte do quociente np é ajustado com base em $np1$.
  - O vetor completo do resto r_full é criado para incluir todos os coeficientes, incluindo zeros, dentro do suporte nr.

Após executar o código no Octave, o vetor `X5` fornecerá os coeficientes do polinômio resultante da multiplicação:

$$X_5(z) = [\text{coeficientes resultantes}]$$

Os coeficientes resultantes do vetor `X5` representam o polinômio final, que combina todas as potências de z resultantes da multiplicação.

2.0.3 GNU Octave

```
function [p, np, r_full, nr] = deconv_m(b, nb, a, na)
    % Modified deconvolution routine for noncausal
    sequences

    % Adjust the support of the quotient
    np1 = nb(1) - na(1);

    % Perform deconvolution using the built-in function
    [p, r] = deconv(b, a);

    % Adjust the support of the quotient
    np = [np1, np1 + length(p) - 1];

    % Determine the complete support of the remainder
    nr = [nb(1), nb(1) + length(r) - 1];

    % Create the complete remainder vector (including
    zeros)
    r_full = zeros(1, diff(nr) + 1);

    r_full(end - length(r) + 1 : end) = r;
end

% Determinar os coeficientes do numerador da equa  o
    dada na quest o
% z^2 + z + 1 + z^(-1) + z^(-2) + z^(-3)
b = [1, 1, 1, 1, 1, 1];
nb = [-3, 2];  % -3 <= z <= 2

% Determinar os coeficientes do denominador da equa  o
    dada na quest o
% Coeficientes de z + 2 + z^(-1)
a = [1, 2, 1];
na = [-1, 1];  % -1 <= z <= 1

% Para a equa  o dada a fun  o dever  retornar os
    coeficientes do quociente
```

```
% e os coeficientes do numerador do resto
[p, np, r_full, nr] = deconv_m(b, nb, a, na);

% Quociente
% (z - 1 + 2z^(-1) - 2z^(-2))
% p = [1, -1, 2, -2];
% np = [-2, 1];

% Resto
% 0 + 0 + 0 + 0 + 3z^(-2) + 3z^(-3)
% r_full = [0, 0, 0, 0, 3, 3];
% nr = [-3, 2];

% Exibir os resultados
disp('Quociente p(z):');
disp(p);
disp('Intervalo de exist ncia p(z):');
disp(np);

disp('Resto completo r(z):');
disp(r_full);
disp('Intervalo de exist ncia r(z):');
disp(nr);
```

Após a execução do código de teste, os seguintes resultados foram obtidos:

- Quociente $p(z)$: $[1, -1, 2, -2]$
- Suporte de $p(z)$: $[-2, 1]$
- Resto $r(z)$: $[0, 0, 0, 0, 3, 3]$
- Suporte de $r(z)$: $[-3, 2]$

Na figura 5 é apresentado os resultados no Octave.

Esses resultados confirmam que a função `deconv_m` está operando corretamente, retornando tanto o quociente quanto o resto com os coefi-

Name	Class	Dimension	Value
a	double	1x3	[1, 2, 1]
b	double	1x6	[1, 1, 1, 1, 1, 1]
na	double	1x2	[-1, 1]
nb	double	1x2	[-3, 2]
np	double	1x2	[-2, 1]
nr	double	1x2	[-3, 2]
p	double	1x4	[1, -1, 2, -2]
r_full	double	1x6	[0, 0, 0, 0, 3, 3]

Figura 5: Resultado da operação do problema 4.10

cientes e limites esperados.

P4.11

A questão 4.11 pede para determinar as transformadas Z inversas das funções $X(z)$ fornecidas utilizando o método de expansão em frações parciais. A decomposição em frações parciais permite reescrever uma função racional em termos de frações simples, cujas inversas são conhecidas e facilmente identificadas através de uma tabela de transformadas Z inversas.

O enunciado apresenta cinco funções $X(z)$ e especifica para cada uma delas o comportamento da sequência no domínio do tempo, ou seja, se a sequência é "right-sided"(definida para $n \geq 0$) ou "left-sided"(definida para $n < 0$).

Para resolver essa questão, os seguintes passos foram seguidos:

(a) Para cada $X(z)$, foi realizada a decomposição em frações parciais.

(b) Uma vez obtidas as frações parciais, cada termo foi associado a uma forma conhecida de transformada Z inversa, utilizando as tabelas de transformadas Z inversas.

(c) As propriedades de ROC (região de convergência) foram consideradas para determinar a causalidade da sequência (se é uma sequência à direita ou à esquerda).

Método Utilizado:

O método de frações parciais consiste em escrever a função racional $X(z)$ como a soma de termos do tipo:

$$X(z) = \frac{A}{z - p}$$

onde A é um coeficiente e p é um polo. Esses termos são facilmente invertidos utilizando tabelas de transformada Z inversa. A função 'residue' do GNU Octave ou Python ('sympy.apart') pode ser utilizada para automatizar o cálculo das frações parciais.

Código Python

O código a seguir implementa a solução para a questão 4.11 utilizando Python e a biblioteca Sympy para a decomposição em frações parciais:

```
import sympy as sp

# Definindo as variaveis simbolicas
z = sp.symbols('z')

# Funcoes dadas no problema
X1 = (1 - z**-1 - 4*z**-2 + 4*z**-3) / (1 - (11/4)*z**-1
    + (13/8)*z**-2 - (1/4)*z**-3)
X2 = (1 + z**-1 - 4*z**-2 + 4*z**-3) / (1 - (11/4)*z**-1
    + (13/8)*z**-2 - (1/4)*z**-3)
X3 = (z**3 - 3*z**2 + 4*z + 1) / (z**3 - 4*z**2 + z -
    0.16)
```

```
X4 = z / (z**3 + 2*z**2 + 1.25*z + 0.25)
X5 = z / (z**2 - 0.25)**2

# Decomposicao em fracoes parciais
partial_X1 = sp.apart(X1, z)
partial_X2 = sp.apart(X2, z)
partial_X3 = sp.apart(X3, z)
partial_X4 = sp.apart(X4, z)
partial_X5 = sp.apart(X5, z)

# Exibindo os resultados das fracoes parciais
print("Fracoes Parciais de X1(z):")
sp.pprint(partial_X1)

print("\nFracoes Parciais de X2(z):")
sp.pprint(partial_X2)

print("\nFracoes Parciais de X3(z):")
sp.pprint(partial_X3)

print("\nFracoes Parciais de X4(z):")
sp.pprint(partial_X4)

print("\nFracoes Parciais de X5(z):")
sp.pprint(partial_X5)
```

Explicação do Código

- Primeiramente, definimos a variável simbólica z utilizando a função `symbols()` da biblioteca SymPy, que nos permite manipular expressões algébricas de forma simbólica.
- Para cada uma das expressões $X(z)$, as frações parciais são calculadas com a função `apart()`, que reescreve a função racional como a soma de frações simples.

- O código então imprime as frações parciais de cada $X(z)$, o que nos permite observar os termos simples que, em seguida, podem ser invertidos usando tabelas conhecidas de transformada Z inversa.
- Para realizar a expansão completa, as expressões obtidas são verificadas quanto aos polos, que nos ajudam a definir a região de convergência (ROC) e a sequência no tempo (se é right-sided ou left-sided).

Solução para as Funções $X(z)$

- Para $X_1(z)$, a sequência é right-sided, e os polos são encontrados usando o método de frações parciais. A partir da decomposição, aplicamos a transformada Z inversa para obter $x_1(n)$.
- Para $X_2(z)$, a sequência é right-sided e absolutamente somável, o que indica que sua região de convergência inclui $z = \infty$. O procedimento de frações parciais é semelhante ao anterior.
- Para $X_3(z)$, a sequência é left-sided, ou seja, ela só existe para $n < 0$. A decomposição em frações parciais leva em conta essa propriedade ao calcular a inversa de $X(z)$.
- Para $X_4(z)$ e $X_5(z)$, as regiões de convergência indicam que as sequências são right-sided para $X_4(z)$ ($|z| > 1$) e left-sided para $X_5(z)$ ($|z| < 0.5$), respectivamente.

Cada função é resolvida de forma semelhante, utilizando o método de frações parciais para identificar os termos que podem ser invertidos diretamente.

Conclusão

O método de frações parciais facilita a obtenção da transformada Z inversa, pois reescreve funções racionais em termos de frações simples.

Cada termo pode ser associado a uma expressão conhecida na tabela de transformadas Z inversas. Ao aplicar este método, conseguimos determinar as sequências no tempo para as funções dadas em $X(z)$.

P4.12

A questão 4.12 trata da sequência dada por:

$$x(n) = A_c(r^n \cos(\pi v_0 n)u(n)) + A_s(r^n \sin(\pi v_0 n)u(n))$$

O objetivo é realizar a transformada Z dessa sequência e determinar os parâmetros A_c, A_s, r, e v_0 em termos dos coeficientes da função racional. A função racional resultante da transformada Z dessa sequência possui um par de polos complexos conjugados. Além disso, a questão pede para desenvolver uma função Python que possa ser utilizada para obter a transformada Z inversa de forma que o resultado não contenha números complexos.

Parte 1: Mostrar que a Transformada Z da Sequência é uma Função Racional

A transformada Z de $x(n)$, dada em (4.30), é:

$$X(z) = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}}, \quad |z| > |r|$$

Onde os coeficientes são:

$$b_0 = A_c, \quad b_1 = r\left[A_s \sin(\pi v_0) - A_c \cos(\pi v_0)\right]$$

$$a_1 = -2r\cos(\pi v_0), \quad a_2 = r^2$$

A função $X(z)$ é uma função racional própria de segunda ordem, o que significa que o grau do polinômio do denominador é maior que o grau do polinômio do numerador. Essa função descreve a relação entre a sequência no domínio do tempo e a transformada Z no domínio da frequência.

Parte 2: Determinação dos Parâmetros de Sinal

A partir de (4.32), podemos determinar os parâmetros de sinal A_c, A_s, r, e v_0 em termos dos parâmetros da função racional b_0, b_1, a_1, a_2.

$$A_c = b_0$$

$$r = \sqrt{a_2}$$

$$v_0 = \frac{\arccos\left(\frac{-a_1}{2r}\right)}{\pi}$$

$$A_s = \frac{b_1/r + A_c \cos(\pi v_0)}{\sin(\pi v_0)}$$

Essas equações nos permitem calcular os parâmetros de sinal com base nos coeficientes da função racional.

Parte 3: Implementação da Função em Python

A função Python, denominada `invCCPP`, é projetada para calcular os parâmetros de sinal A_c, A_s, r, e v_0 com base nos parâmetros b_0, b_1, a_1, a_2.

Aqui está a implementação da função:

```
1 import numpy as np
2
3 def invCCPP(b0, b1, a1, a2):
4     """
```

```
    Calcula os parametros Ac, As, r, v0 a partir dos
coeficientes b0, b1, a1, a2.

    Parametros:
    b0, b1, a1, a2 : float
        Coeficientes da funcao racional.

    Retorna:
    Ac, As, r, v0 : float
        Parametros do sinal calculados.
    """
    # Calcula Ac diretamente a partir de b0
    Ac = b0

    # Calcula r a partir de a2 usando a relacao r = sqrt(
a2)
    r = np.sqrt(a2)

    # Calcula v0 a partir de a1 e r, usando a rela  o
trigonometrica e convertendo para a frequencia
normalizada
    w0 = np.arccos(-a1 / (2*r))

    # Converte w0 para v0 (frequencia normalizada)
    v0 = w0 / np.pi

    # Calcula As a partir de b1, r, Ac e w0 usando uma
combinacao de seno e cosseno
    As = (b1 / r + Ac * np.cos(w0)) / np.sin(w0)

    return Ac, As, r, v0
```

Conclusão

Nesta questão, mostramos que a sequência $x(n)$ pode ser descrita por uma função racional de segunda ordem na transformada Z, com polos

complexos conjugados. Determinamos os parâmetros A_c, A_s, r, e v_0 em termos dos coeficientes da função racional e desenvolvemos uma função Python para calcular esses parâmetros de forma eficiente.

P4.13

Nesta questão, a função $X(z)$ é dada por:

$$X(z) = \frac{2 + 3z^{-1}}{1 - z^{-1} + 0.81z^{-2}}, \quad |z| > 0.9$$

O objetivo é usar a função `invCCPP` definida na questão anterior para determinar $x(n)$ de forma que não contenha números complexos. Em seguida, queremos calcular os primeiros 20 valores de $x(n)$.

Parte 1: Determinação de $x(n)$

Usando a função `invCCPP`, podemos determinar os parâmetros A_c, A_s, r, e v_0 que correspondem à sequência no tempo associada a $X(z)$.

Os coeficientes de $X(z)$ são:

$$b_0 = 2, \quad b_1 = 3$$

$$a_1 = -1, \quad a_2 = 0.81$$

A função `invCCPP` calculará os parâmetros de sinal A_c, A_s, r, e v_0 a partir desses coeficientes, resultando em uma sequência no tempo que contém apenas números reais.

Parte 2: Cálculo dos Primeiros 20 Amostras de $x(n)$

Após determinar os parâmetros de sinal, podemos calcular os primeiros 20 valores de $x(n)$ usando Python. Isso será feito utilizando os parâmetros A_c, A_s, r, e v_0 gerados na primeira parte da questão.

```
import numpy as np

def invCCPP(b0, b1, a1, a2):
    """
    Calcula os parametros Ac, As, r, v0 a partir dos coeficientes b0, b1, a1, a2.

    Parametros:
    b0, b1, a1, a2 : float
        Coeficientes da funcao racional.

    Retorna:
    Ac, As, r, v0 : float
        Parametros do sinal calculados.
    """
    # Calcula Ac diretamente a partir de b0
    Ac = b0

    # Calcula r a partir de a2 usando a relacao r = sqrt(a2)
    r = np.sqrt(a2)

    # Calcula v0 a partir de a1 e r, usando a relacao trigonometrica e convertendo para a frequencia normalizada
    w0 = np.arccos(-a1 / (2*r))

    # Converte w0 para v0 (frequencia normalizada)
    v0 = w0 / np.pi

```

```
27      # Calcula As a partir de b1, r, Ac e w0 usando uma
    combinacao de seno e cosseno
28      As = (b1 / r + Ac * np.cos(w0)) / np.sin(w0)
29
30      return Ac, As, r, v0
31
32  def calculate_xn(Ac, As, r, v0, n_samples=20):
33      """
34      Calcula os primeiros n_samples valores de x(n) para n
        variando de 0 a n_samples-1.
35
36      Parametros:
37      Ac, As, r, v0 : float
38          Parametros do sinal.
39      n_samples : int
40          Numero de amostras de x(n) a serem calculadas.
41
42      Retorna:
43      x_n : array
44          Valores de x(n) calculados.
45      """
46      n = np.arange(n_samples)
47      x_n = Ac * (r ** n) * np.cos(np.pi * v0 * n) + As * (
    r ** n) * np.sin(np.pi * v0 * n)
48      return x_n
49
50  # Coeficientes de X(z)
51  b0 = 2
52  b1 = 3
53  a1 = -1
54  a2 = 0.81
55
56  # Calcula os parametros usando a funcao invCCPP
57  Ac, As, r, v0 = invCCPP(b0, b1, a1, a2)
58
59  # Calcula os primeiros 20 valores de x(n)
60  x_n = calculate_xn(Ac, As, r, v0, n_samples=20)
```

```
61
62 # Exibe os valores de x(n)
63 print("Primeiros 20 valores de x(n):")
64 print(x_n)
```

Conclusão

Utilizando a função `invCCPP`, conseguimos determinar $x(n)$ de maneira eficiente, garantindo que a sequência no tempo seja composta apenas por números reais. O código Python também nos permite calcular as primeiras 20 amostras de $x(n)$ e compará-las com o resultado obtido na parte anterior.

P4.15

A questão P4.15 apresenta sistemas lineares e invariantes no tempo descritos pelas seguintes respostas impulsivas $h(n)$. Para cada uma delas, são solicitadas as seguintes tarefas:

- (i) Determinar a representação da função do sistema $H(z)$.
- (ii) Determinar a equação de diferença do sistema.
- (iii) Traçar o diagrama de polos e zeros.
- (iv) Calcular a saída $y(n)$, dado que a entrada é $x(n) = \left(\frac{1}{4}\right)^n u(n)$.

1. $h(n) = 5\left(\frac{1}{4}\right)^n u(n)$

(a) (i) A transformada Z de $h(n)$ é dada por:

$$H(z) = 5 \cdot \frac{1}{1 - \frac{1}{4}z^{-1}}, \quad |z| > \frac{1}{4}$$

(b) (ii) A equação de diferença correspondente é:

$$y(n) - \frac{1}{4}y(n-1) = 5x(n)$$

(c) (iii) O diagrama de polos e zeros é calculado usando a função $H(z)$. O polo está localizado em $z = \frac{1}{4}$ e o zero está em $z = 0$.

(d) (iv) A saída $y(n)$ pode ser calculada usando a convolução de $h(n)$ com $x(n)$. O resultado é:

$$y(n) = \frac{5}{3}\left(\frac{1}{4}\right)^n u(n)$$

2. $h(n) = n\left(\frac{1}{3}\right)^n u(n) + \left(-\frac{1}{4}\right)^n u(n)$

(a) (i) A transformada Z de $h(n)$ é dada por:

$$H(z) = \frac{\frac{1}{3}}{\left(1 - \frac{1}{3}z^{-1}\right)^2} + \frac{1}{1 + \frac{1}{4}z^{-1}}, \quad |z| > \frac{1}{3}$$

(b) (ii) A equação de diferença correspondente é:

$$y(n) - \frac{1}{3}y(n-1) - \frac{1}{9}y(n-2) = x(n)$$

(c) (iii) O diagrama de polos e zeros mostra polos em $z = \frac{1}{3}$ (de ordem 2) e $z = -\frac{1}{4}$, e zeros em $z = 0$.

(d) (iv) A saída $y(n)$ é calculada como:

$$y(n) = \frac{\left(\frac{1}{4}\right)^n}{3n + 1}$$

Implementação em Python

O código Python abaixo calcula os diagramas de polos e zeros e a saída $y(n)$ para cada um dos sistemas:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

# Definir os coeficientes para os diferentes sistemas
systems = [
    {"b": [5], "a": [1, -1/4]},   # Sistema 1
    {"b": [1/3, 1], "a": [1, -1/3, -1/9]},  # Sistema 2
    {"b": [3], "a": [1, -0.9**2]},  # Sistema 3
    {"b": [1/2], "a": [1, -1/2]},  # Sistema 4
    {"b": [2], "a": [1, -1]},  # Sistema 5
]

# Funcao para plotar polos e zeros
def plot_pole_zero(b, a, system_num):
    system = signal.TransferFunction(b, a, dt=True)
    zeros, poles, gain = signal.tf2zpk(b, a)  # Ajuste
    aqui, capturando o ganho tamb m
    plt.figure()
    plt.scatter(np.real(poles), np.imag(poles), marker='x
    ', label='Polos')
    plt.scatter(np.real(zeros), np.imag(zeros), marker='o
    ', label='Zeros')
    plt.title(f'Diagrama de Polos e Zeros - Sistema {
    system_num}')
    plt.axhline(0, color='black')
    plt.axvline(0, color='black')
    plt.grid(True)
    plt.legend()
    plt.show()

# Exemplo de calculo para o sistema 1
```

```
for i, sys in enumerate(systems):
    plot_pole_zero(sys["b"], sys["a"], i + 1)

    # Exemplo de calculo da saida y(n) para os primeiros 10 valores
    n = np.arange(0, 10)
    x_n = (1/4)**n
    y_n = signal.lfilter(sys["b"], sys["a"], x_n)
    print(f"Saida y(n) para o sistema {i+1}: {y_n}")
```

Referências