**Aim:** Analysis of Discrete-Time Signals Using Z-Transform

## IDE:

Install Library pip install sympy Example 1:

import sympy as sp # Define symbols

n, z, a = sp.symbols('n z a')

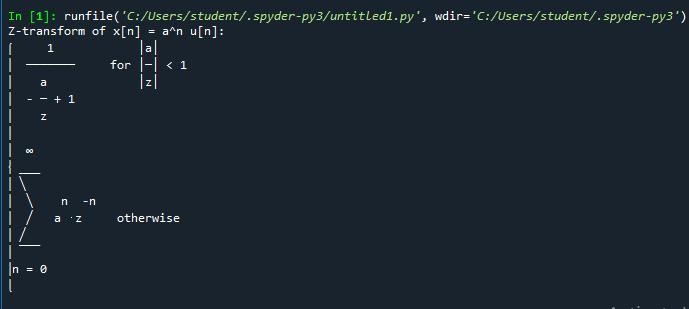
# Define the signal x[n] = a^n \* u[n] x\_n = a\*\*n

# Compute the Z-transform

X\_z = sp.summation(x\_n \* z\*\*(-n), (n, 0, sp.oo)) # Print the result

print("Z-transform of x[n] = a^n u[n]:") sp.pprint(X\_z, use\_unicode=True)

# OUTPUT:



Example 2:

# Define symbols

n, z, a = sp.symbols('n z a')

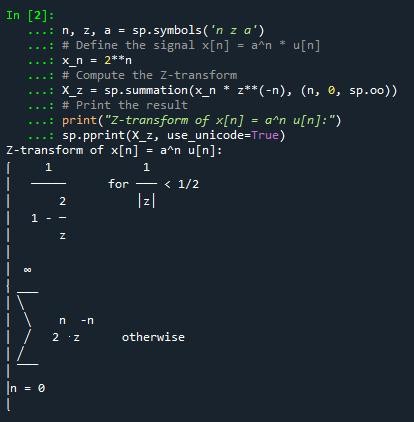
# Define the signal x[n] = a^n \* u[n] x\_n = 2\*\*n

# Compute the Z-transform

X\_z = sp.summation(x\_n \* z\*\*(-n), (n, 0, sp.oo)) # Print the result

print("Z-transform of x[n] = a^n u[n]:") sp.pprint(X\_z, use\_unicode=True)

# OUTPUT:



Example 3:

import sympy as sp # Define symbols

n, z = sp.symbols('n z')

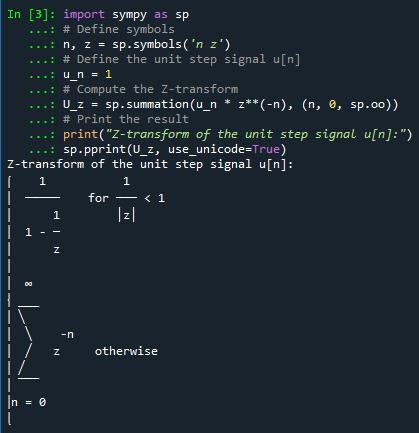
# Define the unit step signal u[n] u\_n = 1

# Compute the Z-transform

U\_z = sp.summation(u\_n \* z\*\*(-n), (n, 0, sp.oo)) # Print the result

print("Z-transform of the unit step signal u[n]:") sp.pprint(U\_z, use\_unicode=True)

# OUTPUT:



Example 4:

import sympy as sp # Define symbols

n, z, alpha = sp.symbols('n z alpha')

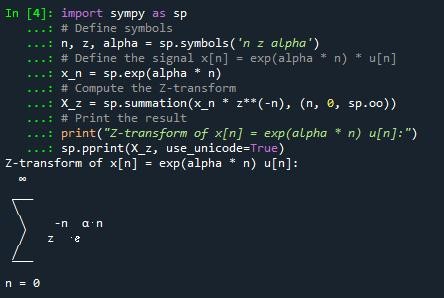
# Define the signal x[n] = exp(alpha \* n) \* u[n] x\_n = sp.exp(alpha \* n)

# Compute the Z-transform

X\_z = sp.summation(x\_n \* z\*\*(-n), (n, 0, sp.oo)) # Print the result

print("Z-transform of x[n] = exp(alpha \* n) u[n]:") sp.pprint(X\_z, use\_unicode=True)

# OUTPUT:



Example 5:

import sympy as sp # Define symbols

n, z = sp.symbols('n z')

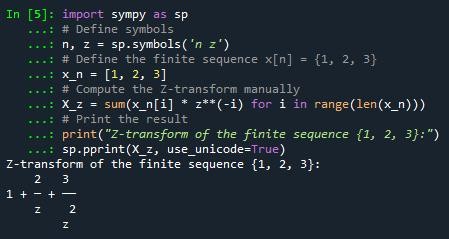
# Define the finite sequence x[n] = {1, 2, 3} x\_n = [1, 2, 3]

# Compute the Z-transform manually

X\_z = sum(x\_n[i] \* z\*\*(-i) for i in range(len(x\_n))) # Print the result

print("Z-transform of the finite sequence {1, 2, 3}:") sp.pprint(X\_z, use\_unicode=True)

# OUTPUT:



Example 6

import sympy as sp # Define symbols

n, z, omega = sp.symbols('n z omega')

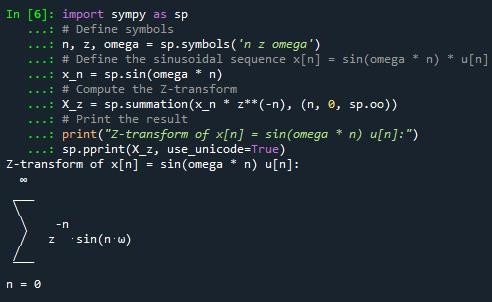
# Define the sinusoidal sequence x[n] = sin(omega \* n) \* u[n] x\_n = sp.sin(omega \* n)

# Compute the Z-transform

X\_z = sp.summation(x\_n \* z\*\*(-n), (n, 0, sp.oo)) # Print the result

print("Z-transform of x[n] = sin(omega \* n) u[n]:") sp.pprint(X\_z, use\_unicode=True)

# OUTPUT:



## Post Lab Exercise:

* Using Python, compute the Z-transform of the sequence  = 3[].

# INPUT:

import sympy as sp

# Define symbols

n, z = sp.symbols('n z')

# Set alpha = ln(3) alpha = sp.log(3)

# Define the signal x[n] = 3^n \* u[n] = exp(alpha \* n) \* u[n] x\_n = sp.exp(alpha \* n)

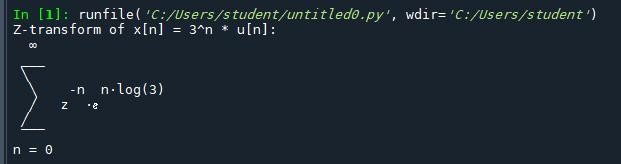
# Compute the Z-transform: sum from n = 0 to ∞ of x[n] \* z^(-n) X\_z = sp.summation(x\_n \* z\*\*(-n), (n, 0, sp.oo))

# Print the result

print("Z-transform of x[n] = 3^n \* u[n]:")

sp.pprint(X\_z, use\_unicode=True)

# OUTPUT:



* Using Python, compute the Z-transform of the sequence  = cos ()[].

# INPUT:

import sympy as sp

# Define symbols

n, z, w = sp.symbols('n z w', real=True)

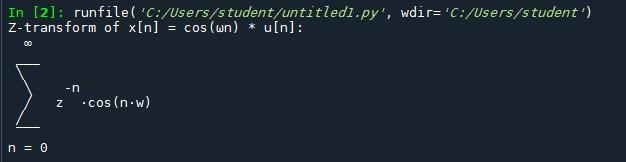
# Define the signal x[n] = cos(w \* n) \* u[n] x\_n = sp.cos(w \* n)

# Compute the Z-transform: sum from n = 0 to ∞ of x[n] \* z^(-n) X\_z = sp.summation(x\_n \* z\*\*(-n), (n, 0, sp.oo))

# Print the result

print("Z-transform of x[n] = cos(ωn) \* u[n]:") sp.pprint(X\_z.simplify(), use\_unicode=True)

OUTPUT:



github:

<https://github.com/hemanth-singampalli/pwp.git>