Python Project 1:

Discrete-time Signal Classifier: Energy Vs. Power

Submitted by

Group - 5

2. **Aaromal A (24101880)**
15. **Alfin Francis (24101666)**
33. **Devanarayanan C R (24101115)**
39. **G Govardan (24101115)**
49. **Ivana Anto (24101562)**

Signal Classifier: Energy Vs. Power

5 February, 2026

Aim

The aim is to classify a given real-valued signal into energy signal or power signal, or neither of them, using numeric integration, and also to find their energy and power. The signal may be given as a Python function, and the test is done by calling the classifier function..

Tools used

- Python 3.14
- Jupyter notebook
- Numpy and Math module

Method of Classification (Convergence)

The given discrete time signal's energy (E) and power (P) are computed using the summation of squares of samples as:

$$E = \sum_{n=-\infty}^{\infty} |x[n]|^2$$

$$P = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} |x[n]|^2$$

To classify the signal into either category, the summation has to be evaluated and checked if it converges to a non-zero value. The signal is energy signal if $0 < E < \infty$ and it is power signal $0 < P < \infty$. To test for the convergence of the signal, the energy and power are computed within various increasing upper limits L ($L = 10^2, 10^3, \ldots, 10^6$), ie. in $(-10^2, 10^2), \ldots, (-10^6, 10^6)$. From this, a sequence containing differences between each successive limit (energy or power) is obtained and checked if it tends to decrease. Additionally, to account for any rounding error which may result in an oscillatory pattern in the sequence of differences, it is also checked whether each difference is within the average of the whole difference sequence, ie. thus converges to a finite value.

- **Energy Signal:** If the energy converges to a finite value ($0 < E < \infty$) and the average power $P = 0$.
- **Power Signal:** If the energy diverges ($E \to \infty$) but the average power converges to a finite non-zero constant ($0 < P < \infty$).
- **Neither:** If both energy and power diverge, as seen in growing exponential signals that trigger numerical overflows.

Source Code

```python
#jupyter python program to classify discrete time signals as energy or power usinfg convergence

import numpy as np
from math import inf

'''3. It check the stability ie convergence of received energy/power sequence by computing the differences
btw consective elements (energy or power) of the seq, hence analysing it. The sequence is unstable if:
   - the differences of consecutive elements (formed diff list) is inc and any diff in the list is
     greater than avg diff of elements (ie not a finite sequence or have oscialltaionss) or
   - the list of energies contain 0 or infinity or
   - the length of energy/power list is <2.
    Thus returning the respective boolean corresponding to stability of the recievd sequence.
ie the sequence is stable if the difference btw the consecutive elements of the sequence (energy/power) is
decreasing and is less than the average difference ie finite.'''

def is_stable(sequence):
    #Checks if a sequence approaches a non-zero, non-infinite limit.
    if len(sequence) < 2: return False
    if 0 in sequence or inf in sequence: return False

    # Calculate differences between successive calculation steps
    diffs = [abs(sequence[i] - sequence[i-1]) for i in range(1, len(sequence))]
    avg_diff = sum(diffs) / len(diffs)

    for i in range(1, len(diffs)):
        # If the gap between steps grows and is larger than average, hasn't converged
        if diffs[i] > diffs[i-1] and diffs[i] > avg_diff:
            return False
    return True

'''2. Generate sample signal within upper limits -10^2 to 10^2, -10^3 to 10^3, ..... -10^7 to 10^7 for the
signal_fn, compute energies and powers of each interval and pass to is_stable() to check the stability ie
convergence of enegy/power seq, thereby return the classification (stable energies = energy signal and viceversa)'''

def classify_dt_signal(signal_func):
    energies = []
    powers = []

    # Test for increasing 'N' (Number of samples), We check the signal from N=100 to N=1,000,000
    for i in range(2, 7):
        N = 10**i
        n = np.arange(-N, N)
        x_n = signal_func(n)

        E = np.sum(np.abs(x_n)**2) # 1. Compute Signal Energy: Sum of squares
        P = E / len(n) # 2. Compute Average Power: Energy / Total Duration

        energies.append(E)
        powers.append(P)

    # Classification Logic
    if is_stable(energies):
        return f"Energy Signal (E  {energies[-1]:.4f})"
    elif is_stable(powers):
        return f"Power Signal (P  {powers[-1]:.4f})"
    else:
        return "Neither Energy nor Power Signal"
```

Validation against test signals

```
''' 1. Representation of discrete time signals as fns'''

# Rectangular Pulse (Energy Signal)
def rect(n): return np.where((n >= 0) & (n <= 10), 1, 0)

# Unit Step (Power Signal)
def unit_step(n): return np.where(n >= 0, 1, 0)

def dc_signal(n):
    # Constant value of 2 for all indices
    return np.full_like(n, 2, dtype=float)

'''
def growing_exp(n):
    # Returns e^n for n >= 0
    # Note: We cap n to avoid float overflow errors
    return np.where((n >= 0) & (n < 20), np.exp(n), 0)
'''
def growing_exp(n):
    try:
        # A slower growth like 1.1^n helps avoid instant overflow
        # so you can actually see the instability.
        val = 1.1**n
        return np.where(n >= 0, val, 0)
    except:
        return float('inf')

#Decaying Exponential (Energy Signal)
# As n increases, value decreases; energy sum converges.
def decaying_exp(n):
    return np.where(n >= 0, (0.5)**n, 0)

#Discrete Impulse / Kronecker Delta (Energy Signal)
# Only has a value at n=0.
def impulse(n):
    return np.where(n == 0, 1, 0)

#Unit Ramp (Neither Energy nor Power)
# Grows linearly; both E and P will diverge to infinity.
def unit_ramp(n):
    return np.where(n >= 0, n, 0)

# Signum Function (Power Signal)
# Similar to a DC signal but changes polarity at n=0.
def signum_sig(n):
    return np.sign(n)

print(f"Rect Pulse: {classify_dt_signal(rect)}")
print(f"Unit Step: {classify_dt_signal(unit_step)}")
print(f"sine wave: {classify_dt_signal(np.sin)}")
print(f"dc/constant signal: {classify_dt_signal(dc_signal)}")
print(f"growing exponential signal: {classify_dt_signal(growing_exp)}")
print(f"Decaying Exp: {classify_dt_signal(decaying_exp)}")
print(f"Impulse: {classify_dt_signal(impulse)}")
print(f"Unit Ramp: {classify_dt_signal(unit_ramp)}")
print(f"Signum: {classify_dt_signal(signum_sig)}")
```

Output

```
Rect Pulse: Energy Signal (E ≈ 11.0000)
Unit Step: Power Signal (P ≈ 0.5000)
sine wave: Power Signal (P ≈ 0.5000)
dc/constant signal: Power Signal (P ≈ 4.0000)
growing exponential signal: Neither Energy nor Power Signal
Decaying Exp: Energy Signal (E ≈ 1.3333)
Impulse: Energy Signal (E ≈ 1.0000)
Unit Ramp: Neither Energy nor Power Signal
Signum: Power Signal (P ≈ 1.0000)
```