Python Project:

Signal Classification System: Energy Vs. Power

Submitted by

Group - 5

Roopesh O R

Roshna Palatty Santhosh

Merella Jobi

Signal Classification System: Energy Vs. Power

March 30, 2025

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

- Jupyter notebook
- `math` module
- `inspect` module (for testing purpose)

Method of classification

The given signal's energy (E) and power (P) are computed using definite integrals:

$$E = \int_{-L}^{L} f^2(x)dt$$

$$P = \frac{1}{2L}\int_{-L}^{L} f^2(x)dx$$

with a large upper limit of L

The integration is computed numerically using the Trapezoidal rule (https://en.wikipedia.org/wiki/Trapezoidal_rule#Uniform_grid).

To classify the signal into either category, the integral 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 if $0 < P < \infty$

To test for the convergence of integral, integrals for energy and power are computed with various increasing upper limits L ($L = 10^2, 10^3, \cdots 10^6$). From this, a sequence containing differences between each successive limit 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.

Numeric integration

In the program, numeric integration is performed with the help of a wrapper function $e(x)$. This wrapper function evaluates the signal function at both positive and negative input values and also checks for any overflows or zero divisions and handles them appropriately:

$$e(x) = \begin{cases} \infty & \text{, if } f^2(x) \text{ or } f^2(-x) \text{ overflows or it results in zero division} \\ f(x)^2dx + f(-x)^2dx & \text{, otherwise} \end{cases}$$

Here, stepsize dx is choosen to 0.1

The summation is done by:

$$E = \frac{e(0) + e(L)}{2} + \sum_{dx \le x \le L - dx} e(x)$$

$$P = \frac{e(0) + e(L)}{2T} + \sum_{dx \le x \le L - dx} \frac{e(x)}{T}$$

Where $T = 2L$

For readability, in the program `end` is used to denote L and `duration` is used to denote T

Source code

[1]:
```
from math import inf
def compute_E_P(f, end, dx):
    """
    Computes energy and power of given signal f and integrates
    it from -end to end with step size dx
    """
    duration  = end * 2
    power_overflown = False

    def e(x):
        """ wrapper function to compute integrants.
        It also check overlfow and zero division
        """
        try: return f(x) ** 2 * dx + f(-x) ** 2 * dx
        except (OverflowError, ZeroDivisionError):
            return inf

    # using trapezoidal rule to integerate

    # add initial parts
    # e(0) is divided by 2 to remove duplicate summation
    E = (e(0) + e(end)) / 2
    P = E / duration

    x = dx
    while (
        x < end and
        P != inf # check if power has overflown, if so break the loop
    ):
        i = e(x)
        E += i
```

```
        P += i / duration

        x += dx

    return (E, P)

def is_converging_nonzero(sequence):
    """
        returns true if sequence tend to approach a
        value that is not zero or infinity
    """

    # Check if sequence contains zero or infinity
    if 0 in sequence or inf in sequence: return False

    """
    Check if function stabilizes to some value
    This is done by finding absolute differences of successive
    elements and checking whether each difference is smaller than
    the previous difference.
    """
    diffs = [
        abs(sequence[i-1] - sequence[i]) for i in range(1, len(sequence))
    ]
    # and their average
    avg_diff = sum(diffs) / len(diffs)

    """
    Here the sequence is tested for instability.
    The sequence considered unstable if any of difference is larger
    than the previous one and the difference is larger than the
    average of all differences.

    The condition is checked to ensure that sequence is not
    marked unstable due to some roundoff error in summation.
    """
    for i in range(1, len(diffs)):
        if (diffs[i-1] < diffs[i] and diffs[i] > avg_diff):
            return False
    return True

def classify_signal(f):
    """
    Function that does the final classification
    Classification is done by finding integrals from -10^2 to 10^2,
    -10^3 to 10^3, upto -10^6 to 10^6 and checking if they
    converge to some value (not equal to 0)
    """
    Energies = []
```

```
    Powers = []
    for i in range(2, 7):
        EP = compute_E_P(f, 10**i, 0.1)
        Energies.append(EP[0])
        Powers.append(EP[1])

    if is_converging_nonzero(Energies):
        print("Energy signal, energy:", Energies[-1])
    elif is_converging_nonzero(Powers):
        print("Power signal, power:", Powers[-1])
    else:
        print("Neither energy nor power signal")
```

Testing against some signals

[2]:
```
from math import sin, e
from inspect import getsource

def f1(x): return sin(x) * e**x
def f2(x): return 1/x
def exp(x): return e**x
def unit_step(x): return 1 if x >= 0 else 0
def rect(x): return 1 if x>=0 and x <= 5 else 0
def linear(x): return x
def gaussian(x): return e**(-x**2)
def sigmoid(x): return 1/(1+e**(-x))
def sqrt(x): return x ** 0.5
def sine(x): return sin(x)


def print_function(f):
    """
    Prints the expression inside a given function
    """
    print(
        "f(x) =",
        getsource(f).split("return")[1].strip()
    )

fx = [
    f1,
    f2,
    exp,
    unit_step,
    linear,
    sqrt,
    sine,
    gaussian,
```

```
    sigmoid,
    rect
]

for f in fx:
    print_function(f)
    classify_signal(f)
    print("\n")
```

Output

```
f(x) = sin(x) * e**x
Neither energy nor power signal


f(x) = 1/x
Neither energy nor power signal


f(x) = e**x
Neither energy nor power signal


f(x) = 1 if x >= 0 else 0
Power signal, power: 0.500000074875085


f(x) = x
Neither energy nor power signal


f(x) = x ** 0.5
Neither energy nor power signal


f(x) = sin(x)
Power signal, power: 0.5000001756971644


f(x) = e**(-x**2)
Energy signal, energy: 1.2533141373155003


f(x) = 1/(1+e**(-x))
Neither energy nor power signal


f(x) = 1 if x>=0 and x <= 5 else 0
```