Stochastic Analysis of Random Signal with Noise

Andreas Avgoustis
Department of Informatics
Ionian University
Corfu, Greece
inf.bdn2202@ionio.gr

***Abstract*—Signal processing is essentially an interdisciplinary field, strictly defined by mathematics and its own methodologies and terminology. Its applications are numerous in technological sciences and it forms the basis of fields such as telecommunications, automation, image, video and audio processing, data compression, etc. This paper will focus on the analysis, processing, and cleaning of a random signal using stochastic equations and simple mathematical functions.**

***Index Terms*—Stochastic equations, mathematical functions, signals, signal processing**

I. Introduction

Signal processing involves the analysis and manipulation of signals, where a signal is defined as any function between physical quantities. It is an interdisciplinary scientific field, strictly defined by mathematics and its own methodologies and terminology. The applications are numerous in technological sciences and are fundamental in fields such as telecommunications, automation, image, video and audio processing, data compression, etc. Used in telecommunications systems, signal processing occurs only at the first level of the OSI reference model, the physical layer, and optionally at the sixth and seventh levels of the same model. [1].

In probability theory and related fields, a stochastic or random process is a mathematical object usually defined as a sequence of random variables, where the index of the sequence is interpreted as time. Stochastic processes are widely used as mathematical models for systems and phenomena that appear to vary in a random manner. Examples include the growth of a population of bacteria, an electric current that varies due to thermal noise, or the movement of a gas molecule. Stochastic processes have applications in many fields such as biology, chemistry, ecology, neuroscience, physics, image processing, signal processing, control theory, information theory, computer science, cryptography, and telecommunications. Moreover, the seemingly random changes in financial markets have prompted extensive use of stochastic processes in finance. [8].

The study of such phenomena has inspired the proposal of new stochastic processes. Examples of such processes include the Wiener process and Brownian motion, used by Louis Bachelier to study price changes. These two stochastic processes are considered the most important and central in the theory of stochastic processes and have been discovered independently and repeatedly. [3]

In this paper, a random noise signal is generated and processed and cleaned using Python [10] ϰnd simple mathematical functions. [4] [5]

II. Mathematical Functions

Identifying the laws governing various phenomena and understanding their form often gives us the ability to predict their future evolution.

For phenomena described by deterministic procedures, this is easier. However, most phenomena evolve over time presenting a degree of randomness, and so, an appropriate stochastic model must be found. A stochastic process that has become increasingly common in recent years, especially for modeling economic data, is the Ornstein-Uhlenbeck process [5] , defined as:.

$$dX_t = -\alpha X_t dt + \beta dW_t \tag{1}$$

, where a and β are constants, Xt is the time of the noise, and, w is the weight of the noise. This stochastic process allows us to generate and analyze signals. Every signal contains noise that complicates correct analysis and understanding. Based on this fact, a simple noise reduction technique is used, known as Smoothing Moving Average, defined as: [4]

$$\text{SMA} = \frac{X_1 + X_2 + X_3 + \ldots + X_n}{n} \tag{2}$$

where

- SMA stands for Simple Moving Average
- X1, X2, X3, ..., Xn εare the data points to be measured
- n is the number of data points averaged (known as window size)

III. Methodology Implementation

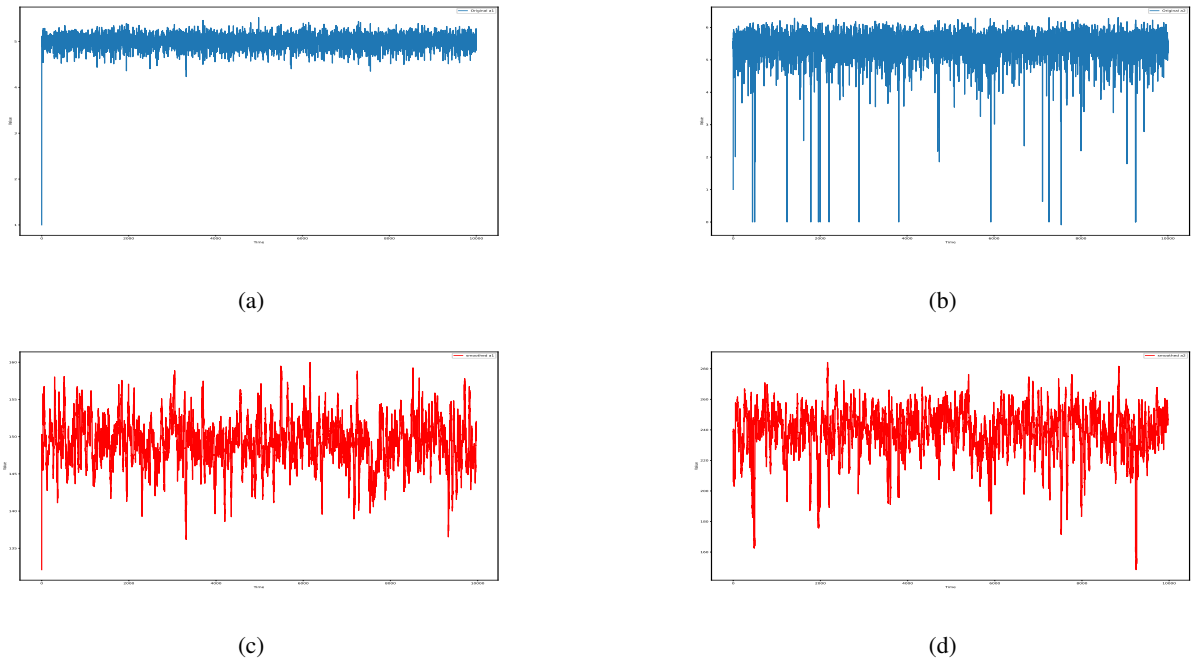
In case the noise cannot be successfully eliminated, logarithmic [6] and exponential operations [9] are used, specifically:

$$\log(a) = \log(b) + \log(c) \tag{3}$$

$$\text{a} = e^{\log(x)} \tag{4}$$

Then, a simple equation is implemented for noise elimination:

$$\alpha = \beta * \gamma \tag{5}$$



(a) (b)

(c) (d)

Fig. 1: (a) Results of a together with the smoothed for s = 30. (b) Results of a together with the smoothed a for s = 120. (c) Results of the smoothed a for s = 30. (d) Results of the smoothed a for s = 120.

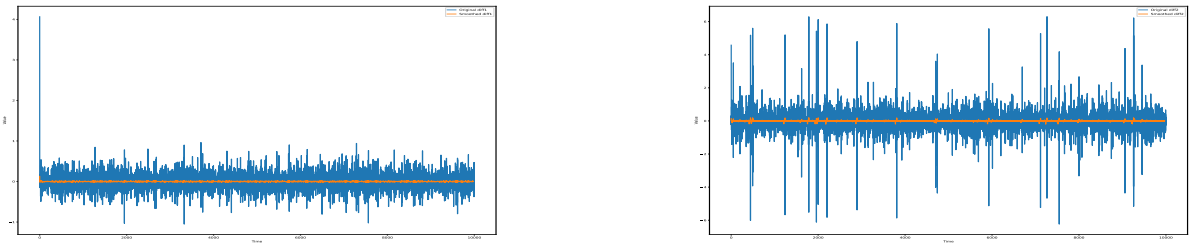
όπου

- β will be the equation for the Ornstein-Uhlenbeck process [7]
- γ will represent additional noise

Using the equations revisited in section II, a random signal can be generated. Random numbers in the range -1 to 1 are used, as shown in the code segment:

```
t1 = np.zeros(10000)
t2 = np.zeros(10000)
b1 = np.zeros(10000)
b2 = np.zeros(10000)
a1 = np.zeros(10000)
a2 = np.zeros(10000)
a = 0.9
s1 = 30
s2 = 120
w1 = np.zeros(10000)
w2 = np.zeros(10000)
g = np.zeros(10000)
for i in range(10000):
    w1[i] = round(random.uniform(-1.00,1.00)↩
        ,1)
    w2[i] = round(random.uniform(-1.00,1.00)↩
        ,1)
    g[i] = round(random.uniform(-1.00,1.00)↩
        ,1)
window_size = 30
```

At the first stage, value arrays with 10,000 positions are created and weights, further noise, and the window for the moving average smoothing are initialized. Then, the equation is executed for every value of i, and the noise is multiplied



(a) (b)

Fig. 2: (a) Results of the difference a together with the smoothed for s = 30. (b) Results of the difference a together with the smoothed a for s = 120.

(used for both constants s1, s2). The result is smoothed by moving average, as seen in the code and image 1.

```
t1[0] = 1
for i in range(1,10000):
    if i != 0 :
        t1[i] = (-a * t1[i-1] + s1 * w1[i])
        b1[i] = t1[i]
        a1[i] = b1[i]*g[i]
smoothed_a1 = np.convolve(a1, np.ones(↩
    window_size) / window_size, mode='valid')

t2[0] = 1
for i in range(1,10000):
    if i != 0 :
        t2[i] = (-a * t2[i-1] + s2 * w2[i])
        b2[i] = t2[i]
        a2[i] = b2[i]*g[i]
smoothed_a2 = np.convolve(a2, np.ones(↩
    window_size) / window_size, mode='valid')
```

After the equations are created, we use their results to generate the difference between the current state of noise and the previous one, as shown in the following code. The result must be smoothed using the moving average, as illustrated in Figure 2. For effective noise smoothing in our case, we need to use the property of the logarithm [2] :

$$\log(\alpha) = \log(\beta) + \log(\gamma) \tag{6}$$

The result of the above equation must be returned to its original form so that we can make it correlated. In this case, we will use the property of the exponential function with a logarithmic exponent. This property will be applied as long as we use it on the smoothed value of alpha. :

$$\alpha = e^{\log(\alpha)} \tag{7}$$

```
diff1 = [a1[i] - a1[i-1] for i in range(1,↩
    len(a1))]
diff2 = [a2[i] - a2[i-1] for i in range(1,↩
    len(a2))]
smoothed_diff1 = np.convolve(diff1, np.ones(↩
    window_size) / window_size, mode='valid')
smoothed_diff2 = np.convolve(diff2, np.ones(↩
    window_size) / window_size, mode='valid')
```

Finally, we create the autocorrelation diagram for s = 30 and s = 120. The autocorrelation shows us how quickly the noise approaches zero. We set the autocorrelation with a lag of 50 as shown in Figure 3 and observe the rate at which the noise diminishes to zero.

IV. Conclusion

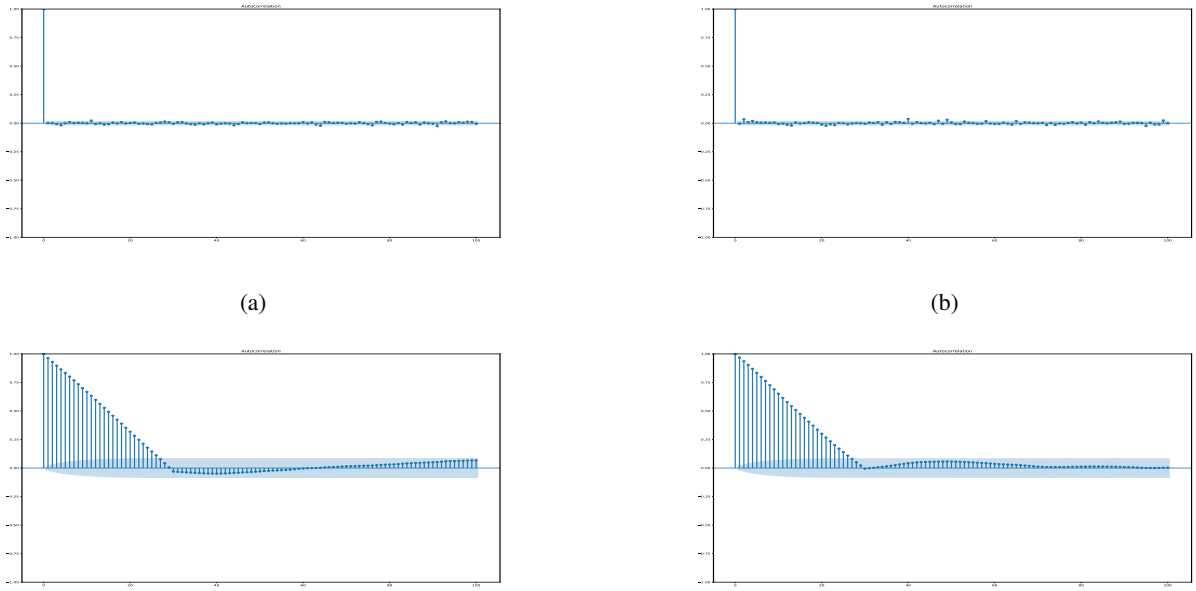
Based on the above implementation and using basic stochastic processes, it is observed that the signal, even after smoothing by moving average, does not completely clean the signal except for the use of differences, where signal cleaning is evident. With these methods, satisfactory results can be achieved in signal analysis and processing for noise smoothing.

V. Complete Implementation Code

Below is the total implementation code:

```
import numpy as np
import matplotlib.pyplot as plt
import random
import csv
from statsmodels.graphics.tsaplots import ↩
    plot_acf
plt.rcParams[ figure.figsize ] = (20,15)
t1 = np.zeros(10000)
t2 = np.zeros(10000)
b1 = np.zeros(10000)
b2 = np.zeros(10000)
a1 = np.zeros(10000)
a2 = np.zeros(10000)
a = 0.8
s1 = 30
s2 = 120
for i in range(10000):
    t1[i] = 1
    t1[i] = 1
    b1[i] = 1
    b2[i] = 1
    a1[i] = 1
    a2[i] = 1
w1 = np.zeros(10000)
w2 = np.zeros(10000)
g1 = np.zeros(10000)
g2 = np.zeros(10000)
for i in range(10000):
    w1[i] = round(random.uniform(-1.00,1.00)↩
        ,1)
    w2[i] = round(random.uniform(-1.00,1.00)↩
        ,1)
    g1[i] = round(random.uniform(-1.00,1.00)↩
        ,1)
    g2[i] = round(random.uniform(-1.00,1.00)↩
        ,1)
window_size = 30
for i in range(1,10000):
    if i != 0 :
        t1[i] = (-a * t1[i-1] + s1 * w1[i])
        b1[i] = t1[i]
        a1[i] = (b1[i] * g1[i]) +150
        a1[i] = np.log(a1[i])

nan_indices = np.isnan(a1)
a1[nan_indices] = 0
smoothed_a1 = np.convolve(a1, np.ones(↩
    window_size) / window_size, mode='valid')
smoothed_a1 = np.exp(smoothed_a1)
plt.plot(a1, label='Original a1')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.savefig('a1.pdf', format='pdf')
plt.show()
plt.plot(smoothed_a1, label='smoothed a1',↩
    color = 'red')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.savefig('smoothed_a1.pdf', format='pdf')
plt.show()
diff1 = [a1[i] - a1[i-1] for i in range(1,↩
```



(a) (b)

(c) (d)

Fig. 3: (a) Autocorrelation for the results of a with s = 30. (b) Autocorrelation for the results of a with s = 120. (c) Autocorrelation for the results of the smoothed a with s = 30. (d) Autocorrelation for the results of the smoothed a with s = 120.

```
    len(a1))]
for i in range(1,10000):
    if i != 0 :
        t2[i] = -a*t2[i-1] + s2*w2[i]
        b2[i] = t2[i]
        a2[i] = (b2[i] * g2[i] ) + 250
        a2[i] = np.log(a2[i])

nan_indices = np.isnan(a2)
a2[nan_indices] = 0
smoothed_a2 = np.convolve(a2, np.ones(↩
    window_size) / window_size, mode='valid')
smoothed_a2 = np.exp(smoothed_a2)
plt.plot(a2, label='Original a2')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.savefig('a2.pdf', format='pdf')
plt.show()
plt.plot(smoothed_a2, label='smoothed a2',↩
    color = 'red')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.savefig('smoothed_a2.pdf', format='pdf')
plt.show()
diff2 = [a2[i] - a2[i-1] for i in range(1,↩
    len(a2))]
smoothed_diff1 = np.convolve(diff1, np.ones(↩
    window_size) / window_size, mode='valid')
smoothed_diff2 = np.convolve(diff2, np.ones(↩
    window_size) / window_size, mode='valid')
plt.plot(diff1, label='Original diff1')
plt.plot(smoothed_diff1, label='Smoothed ↩
    diff1')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.savefig('diff1.pdf', format='pdf')
plt.show()
plt.plot(diff2, label='Original diff2')
plt.plot(smoothed_diff2, label='Smoothed ↩
    diff2')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.savefig('diff2.pdf', format='pdf')
plt.show()
plot_acf(a1,lags=100)
plt.savefig('autoc1.pdf', format='pdf')
plt.show()
plot_acf(smoothed_a1,lags=100)
plt.savefig('auto1.pdf', format='pdf')
plt.show()
plot_acf(a2,lags=100)
plt.savefig('autoc2.pdf', format='pdf')
plt.show()
plot_acf(smoothed_a2,lags=100)
plt.savefig('auto2.pdf', format='pdf')
plt.show()
```

References

[1] Σοφία Εφραιμία Βρεττού. *στοχαστική προσομοίωση της φόρτισης ανέμου και κυμάτων στα υπεράκτια αιολικά πάρκα*. 2023.

[2] Κωνσταντίνος Δραζιώτης. *παραγοντοποίηση & διακριτός λογάριθμος*. 2022.

[3] Παπαϊωάννου Γεώργιος. *Στοχαστικά Μοντέλα Συνεχούς Χρόνου για την Αποτίμηση Αξιογράφων που Ενέχουν*

Πιστωτικό Κίνδυνο. PhD thesis, University of Piraeus (Greece), 2022.
[4] Rob J Hyndman. Moving averages., 2011.
[5] Ross A Maller, Gernot Müller, and Alex Szimayer. Ornstein–uhlenbeck processes and extensions. *Handbook of financial time series*, pages 421–437, 2009.
[6] Μανώλης Δ Μαραγκουδάκης. διακριτός λογάριθμος. B.S. thesis, 2014.
[7] Μαρία Νικολάου Μιχάλη. Διαδικασία *Ornstein-Uhlenbeck*. PhD thesis, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης, 2019.
[8] Κυριάκος Μουστάκας. Διαγνωστική μελέτη για την ακρίβεια εκτίμησης παραμέτρων σε γραμμικά και μη-γραμμικά μοντέλα γεωδαιτικών χρονοσειρών θέσης. PhD thesis, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης, 2022.
[9] Βασιλική Πλεύρη. Η συμπεριφορά του επενδυτή με εκθετική συνάρτηση χρησιμότητας. 2020.
[10] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.