UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

APLICACIONES DE FÍSICA ESTADÍSTICA

DR. OMAR GONZÁLEZ AMEZCUA

PROYECTO 1

SIMULACIÓN DE PROCESO DE ORNSTEIN–UHLENBECK

JOSÉ GUADALUPE ARELLANO EMILIANO	1941509
KEVIN ALFREDO CANSINO TORTOLEDOS	1941507
SEBASTIÁN MENDOZA GARCÍA	1941525

MARZO 2023

Índice

1. Introducción

El proceso de Ornstein-Uhlenbeck es un modelo matemático utilizado en física y en otras áreas para describir el movimiento de un objeto sujeto a fricción. Se define mediante la siguiente ecuación estocástica:

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t$$

donde X_t es el valor del proceso en el tiempo t, θ es una constante que representa la velocidad de reversión a la media, μ es la media del proceso, σ es la volatilidad del proceso y W_t es un proceso de Wiener estándar.

El proceso de Ornstein-Uhlenbeck tiene propiedades interesantes, como la reversión a la media y la dependencia del estado anterior. También se utiliza para modelar la evolución de la posición de una partícula en un medio con fricción.

Con la finalidad de encontrar una representación a los procesos estocásticos, se ha realizado una simulación o representación de gráficas propuestas a realizar. Todo ello utilizando conocimientos del tema. Lo fundamental a considerar es el hecho de que nuestro proceso estocástico sigue un parámetro de DISTRIBUCIÓN NORMAL, siendo esto el pivote a basarnos para que nuestras gráficas se comporten de la manera que se desea.

Para la realización de este proyecto se utilizó como herramienta clave el lenguaje de programación de Python debido a su accesibilidad en cuanto a información sobre las herramientas que ofrece, así como su repertorio de bibliotecas de diversas orientaciones como matemáticas, estadísticas y gráficas.

2. Implementación numérica

Se utilizó Python para la simulación de este proyecto.

Primero se importan las librerías y funciones necesarias.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
```

Se crea una lista vacía donde se almacenarán los números aleatorios que siguen una distribución normal. Estos 50 "números"son los puntos que se logran observar en la gráfica dada en la imagen a referir.

`numale` es la lista vacía en la cual se registrarán los números aleatorios respecto la Normal, que son 50. Dando saltos de 0.2 a 0.2

```
numale = []
numale = np.random.normal(0, 0.2, 50)
```

En la imagen a representar de la tarea tenemos dos gráficas diferentes, por lo que se optó una interfaz de interacción con el usuario para decidir cuál gráfica generar.

A continuación se definen los valores mediante un input del usuario.

Para V_0:

```
velin = int(input())
```

Para V_d:

```
velfn = int(input())
```

Se inicializan las listas donde se almacenarán los valores para la gráfica. Por orden descendente son la función de velocidad de cada punto, el promedio de la velocidad, la desviación superior y la desviación inferior.

```
velad = []
prvd = []
desup = []
desdwn = []
```

En el siguiente ciclo se «rellenan» las listas vacías con los datos.

Se tiene un un contador `t`, el tiempo, que comenzará en 1 y terminará en 50. También se tiene un *delta de t* `dt` que aparece en la imagen, dando así tiempos de 0.1 segundos cada intervalo hasta llegar a los 5 segundos.

Nota: La sangría en el código dentro del ciclo es porque la línea es tan extensa que le falta espacio en la página para los caracteres. En el código es una sola línea.

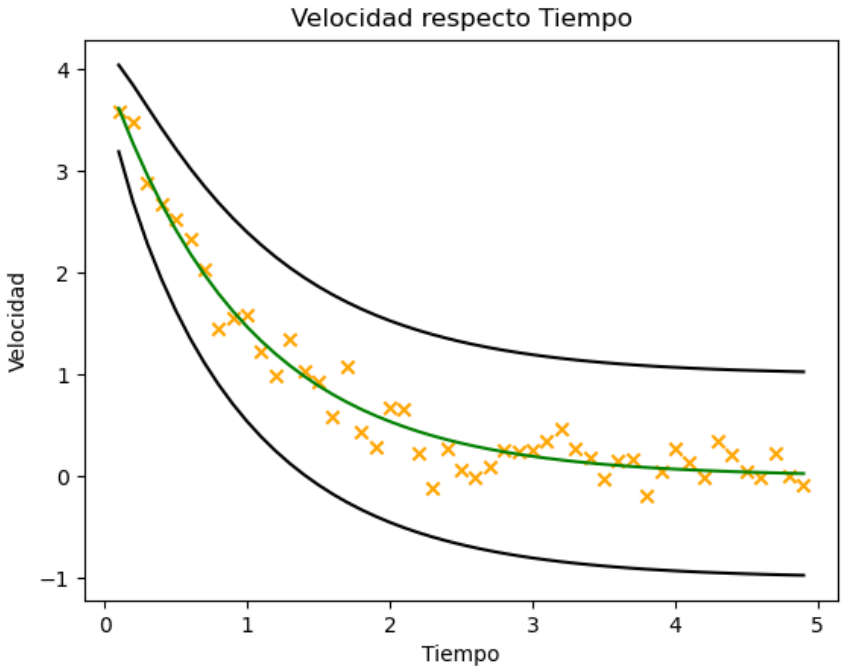
```
for t in range(1, 50):
    dt = t / 10
    velad.append(velin * (np.exp(-dt)) + velfn * (1 - np.exp(-dt)) + np.
   sqrt(1 - np.exp(-2 * dt)) * (numale[t]))
    prvd.append(velin * (np.exp( - dt)) + velfn * (1 - np.exp( - dt)))
    desup.append(velin * (np.exp( - dt)) + velfn * (1 - np.exp( - dt)) +
   np.sqrt(1 - np.exp( - 2 * dt)))
    desdwn.append(velin * (np.exp( - dt)) + velfn * (1 - np.exp( - dt)) -
   np.sqrt(1 - np.exp( - 2 * dt)))
```

Con las listas preparadas se procede a generar la gráfica.

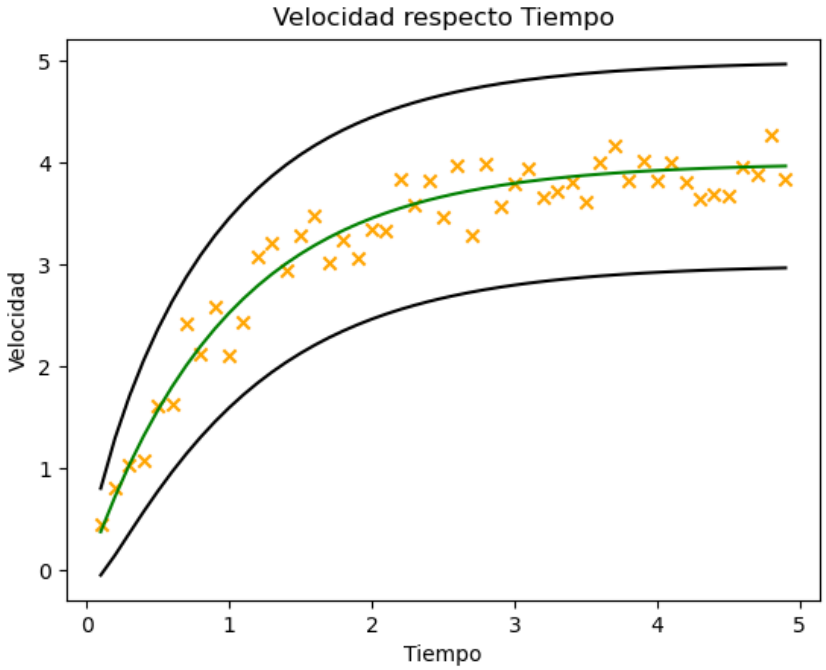
```
plt.scatter(np.arange(1, 50) / 10, velad, color = "orange", marker = "x")
plt.plot(np.arange(1, 50) / 10, prvd, color = "green")
plt.plot(np.arange(1, 50) / 10, desup, np.arange(1, 50) / 10, desdwn,color
    = 'black')
plt.title("Velocidad respecto Tiempo")
plt.xlabel("Tiempo")
plt.ylabel("Velocidad")
plt.show()
```

3. Gráficas

Para $V_0 = 4$ y $V_d = 0$:



Para $V_0 = 0$ y $V_d = 4$:



4. Conclusiones

Se llegó al resultado esperado, todo ello gracias a seguir la distribución normal como base en el desarrollo del código. Múltiples herramientas fueron necesarias para su implementación en diferentes secciones del código. Es por ello que es fundamental que durante un proceso estocástico se conozca bajo cual parámetro o argumento es que se comportan las variables aleatorias.

Referencias

Klenke, A. (2013). *Probability theory: a comprehensive course*. Springer Science & Business Media.

Le Gall, J.-F. (2016). *Brownian motion, martingales, and stochastic calculus*. Springer.

Lemons, D. S. (2003). *An introduction to stochastic processes in physics*. American Association of Physics Teachers.

Stickler, B. A. and Schachinger, E. (2016). *Basic concepts in computational physics*. Springer.