Procesos Estocásticos y sus Aplicaciones en Finanzas

Pablo Marchesi
Junio 2024

Abstract

El objetivo de este trabajo es presentar una breve introducción a los procesos estocásticos, ampliamente presentes en el mundo de las finanzas, así como sus aplicaciones prácticas más populares. En primer lugar, se partirá de conceptos teóricos para definirlos y posteriormente se estudiarán algunos de los usos más frecuentes de estos, como la simulación de acciones cotizadas, la medición de riesgos financieros a través de los modelos VaR o la valoración de opciones con el modelo *Black-Scholes*. Estas aplicaciones prácticas se ejemplificarán mediante un dashboard programado con Python.

"Algebra is like sheet of music. The important thing isn't can you read music, it's can you hear it. Can you hear the music, Robert?"

— Niels Bohr to J. Robert Oppenheimer, *Oppenheimer* (2023)

Índice

1. Introducción **3**

2. Conceptos Básicos de Probabilidad **4**
2.1. Espacio de Probabilidad 4
2.2. Variable Aleatoria 5
2.3. Distribución de Probabilidad 5
2.4. Esperanza y Varianza de una Variable Aleatoria 6

3. Procesos Estocásticos **6**
3.1. Proceso de Wiener 7
3.2. Simulación del Proceso de Wiener 8
3.3. Proceso de Poisson 9
3.4. Simulación del Proceso de Poisson 10

4. Ecuaciones Diferenciales Estocásticas **11**
4.1. Forma General 11
4.2. Simulación de EDEs 12
4.3. Lema de Itô 13
4.4. Movimiento Browniano Geométrico 15
4.5. Resolución del MBG: Método Euler-Maruyama 15
4.6. Resolución del MBG: Lema de Itô 16
4.7. Procesos de Salto-Difusión 19

5. Estimación de Parámetros **21**
5.1. Modelo Log-Normal 21
5.2. Retornos Logarítmicos 22
5.3. Método de Momentos Estadísticos 23
5.4. Método de Máxima Verosimilitud 24
5.5. Validación del Modelo 25

6. Aplicaciones de los Procesos Estocásticos **26**
6.1. Predicción y Simulación 27
6.2. Riesgos: Value at Risk (VaR) 31
6.3. Valoración: Modelo Black-Scholes 32

7. Creación de un Dashboard **35**
7.1. Librerías y Funciones Empleadas 35
7.2. Funciones Propias 36
7.3. Configuración del Dashboard 38
7.4. Cálculo de los Modelos 40
7.5. Gráficas de los Modelos 41
7.6. Resultado Final y Conclusión 44

8. Anexo: Código del Caso Práctico **45**

1 Introducción

Las matemáticas nos permiten dar explicación a los fenómenos que suceden a nuestro alrededor. Desde la aritmética y el álgebra básica, que nos ayudan solucionar problemas sencillos de la vida cotidiana, hasta las ecuaciones diferenciales, que modelizan complejos sistemas cambiantes, las matemáticas están presentes prácticamente en todas las áreas de nuestra vida.

En el caso de las finanzas, esto no es menos. En las últimas décadas, ha proliferado el uso de métodos matemáticos para abordar problemas financieros, llegando hasta el punto que, hoy en día, las finanzas se encuentran estrechamente vinculadas a las matemáticas y sin ellas, nuestro mundo no funcionaría como lo hace ahora mismo. Sin embargo, las herramientas matemáticas usadas en las finanzas difieren de las que se usan en otras áreas como la ingeniería o el resto de ciencias. Esto se debe a que en el núcleo de las finanzas, se encuentra el comportamiento humano, que es impredecible en su naturaleza.

Es por este motivo, que el aparejo matemático usado en finanzas se basa en la probabilidad y la estadística, ya que solo estas nos permiten explicar o modelizar fenómenos tan poco deterministas como la evolución del precio de una acción o la probabilidad de impago de una hipoteca. En concreto, una de las armas más potentes de las finanzas matemáticas son los procesos estocásticos, tema central de este trabajo, que posteriormente definiremos de forma más rigurosa, pero de momento los podemos entender como una secuencia de variables aleatorias que evolucionan con el tiempo.

A lo largo de este trabajo, iremos profundizando en los procesos estocásticos, con un enfoque práctico e intuitivo, dejando de lado el rigor matemático, con el objetivo final de mostrar algunas de sus aplicaciones prácticas, como son la simulación de acciones cotizadas, la medición de riesgos financieros a través de los modelos VaR o la valoración de derivados financieros, como las opciones, a través del modelo *Black-Scholes*. Por último, desarrollaremos una herramienta gráfica, un dashboard programado con Python, para ejemplificar todas estas aplicaciones y permitir variar los parámetros de entrada de los modelos para considerar diferentes escenarios.

2 Conceptos Básicos de Probabilidad

En esta sección definiremos algunos de los conceptos de probabilidad que emplearemos a lo largo del trabajo.

2.1. Espacio de Probabilidad

En primer lugar, definiremos el concepto de espacio de probabilidad, que nos proporciona un marco matemático para modelar y analizar situaciones aleatorias, como las que están presentes en muchas áreas de las finanzas.

Definición 1 *Sea Ω el conjunto de todos los posibles resultados de un experimento, $\mathcal{F}$ una σ-álgebra de conjuntos medibles en Ω, y P una medida de probabilidad definida en $\mathcal{F}$. Entonces, la tripleta $(\Omega, \mathcal{F}, P)$ constituye un espacio de probabilidad si cumple con las siguientes propiedades:*

1. *Ω es el conjunto de todos los posibles resultados del experimento.*
2. *$\mathcal{F}$ es una σ-álgebra de conjuntos medibles en Ω.*
3. *P es una medida de probabilidad definida en $\mathcal{F}$, tal que:*
   a) *$P(A) \geq 0$ para todo A en $\mathcal{F}$ (la probabilidad de cualquier conjunto es no negativa).*
   b) *$P(\Omega) = 1$ (la probabilidad total es 1).*
   c) *Para conjuntos $A_1, A_2, \ldots$ mutuamente excluyentes en $\mathcal{F}$, la probabilidad de la unión es la suma de las probabilidades individuales:*

$$P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i)$$

De forma más informal, un espacio de probabilidad es un concepto matemático que nos permite modelar un experimento aleatorio. Ω es el conjunto de posibles sucesos que se pueden dar en dicho experimento, $\mathcal{F}$ es una colección de conjuntos que es lo suficientemente grande para abordar todos los escenarios relevantes de dicho experimento y que presenta ciertas propiedades deseables a la hora de trabajar con probabilidades, y finalmente P nos permite asignar un número entre el 0 y el 1 (es decir, una probabilidad) a algún escenario del experimento contenido en $\mathcal{F}$.

Sin entrar mucho más en detalle, esta definición del espacio de probabilidad nos sirve de base para explicar los siguientes conceptos de probabilidad que están presentes en los procesos estocásticos. Para una explicación más en profundidad, consultar [6].

2.2. Variable Aleatoria

Definición 2 *Una variable aleatoria, generalmente denotada por X, es una función real definida en un espacio de probabilidad $(\Omega, \mathcal{F}, P)$, y por tanto asociada a un experimento aleatorio, tal que:*

$$X : \Omega \to \mathbb{R}$$

Podemos entender una variable aleatoria como un valor numérico asignado a un experimento, el cual no podemos conocer con certeza. Sin embargo, si que podemos conocer el rango de valores que tomará dicha variable aleatoria y la probabilidad con la que lo hará mediante su distribución de probabilidad.

2.3. Distribución de Probabilidad

Definición 3 *Una variable aleatoria X tiene una función de densidad f_X, siendo f_X una función no negativa e integrable:*

$$P(a \leq X \leq b) = \int_a^b f_X(x)\, dx$$

donde P es la medida de probabilidad y a, b son valores específicos de X.

De esta forma, conocida la función de densidad de probabilidad (o distribución de probabilidad) podemos calcular la probabilidad de que la variable aleatoria X tome cierto rango de valores entre a y b. La distribución más relevante que usaremos a lo largo del trabajo será la distribución normal.

Definición 4 *La distribución normal, también conocida como distribución gaussiana, es denotada por $\mathcal{N}(\mu, \sigma^2)$, donde μ es la media y σ^2 es la varianza. La función de densidad de probabilidad f_X de una variable aleatoria X con distribución normal es:*

$$f(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Si la variable aleatoria con la que trabajamos es discreta (por ejemplo el número de ocurrencias de un suceso), hablamos de distribuciones de probabilidad discretas (o funciones masa de probabilidad). Un ejemplo de este tipo de distribuciones es la distribución de Poisson, que usaremos posteriormente.

Definición 5 *La distribución de Poisson dentada por Poi(λ) es una distribución de probabilidad discreta que describe el número de eventos que ocurren en un intervalo fijo de tiempo o espacio, si estos eventos ocurren con una tasa promedio constante e independientemente entre sí, denotada como λ. La función de masa de probabilidad de la distribución de Poisson está dada por:*

$$P(X = k) = \frac{e^{-\lambda}\lambda^k}{k!}, \quad k = 0, 1, 2, \ldots$$

2.4. Esperanza y Varianza de una Variable Aleatoria

Durante nuestro análisis de los procesos estocásticos, usaremos dos parámetros que caracterizan a las distribuciones de probabilidad: la esperanza y la varianza.

Definición 6 *La esperanza matemática (o valor esperado) de una variable aleatoria X, denotada como $\mathbb{E}[X]$ o μ, se define como:*

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x\, f_X(x)\, dx$$

Definición 7 *La varianza de una variable aleatoria, denotada por $\mathbb{V}[X]$ se puede definir de la siguiente forma:*

$$\mathbb{V}[X] = \int_{-\infty}^{\infty} (x - \mathbb{E}[X])^2\, f_X(x)\, dx$$

3 Procesos Estocásticos

Podemos definir de forma intuitiva un proceso estocástico como una variable aleatoria que varía en función del tiempo. Un ejemplo de un proceso estocástico podría ser una acción cotizada en bolsa, ya que su precio fluctúa, generalmente, de forma aleatoria y lo hace en función del tiempo. Los procesos estocásticos están presentes en áreas como la biología, las telecomunicaciones o la meteorología, pero en nuestro caso, nos centraremos en su aplicación a las finanzas.

Figura 1: Cotización de Microsoft

Una definición más formal es la siguiente:

Definición 8 *Un proceso estocástico es una colección de variables aleatorias X_t definidas en un espacio de probabilidad común $(\Omega, \mathcal{F}, P)$ e indexadas por algún conjunto, generalmente el tiempo, tal que $t \geq 0$. De forma más compacta, se puede expresar un proceso estocástico como:*

$$\{X_t : t \geq 0\}$$

Para cada instante temporal, tenemos una variable aleatoria con su distribución de probabilidad correspondiente. Es por este motivo, que cuando queramos simular un proceso estocástico en el tiempo, nos vamos a encontrar con la problemática

de que tenemos que tener en cuenta las infinitas realizaciones que presenta el proceso.

Dos procesos estocásticos con ciertas propiedades especiales son:

1. *Proceso Gaussiano:* X_t está distribuido normalmente para todo t.
2. *Proceso de Markov:* La evolución futura del proceso solo depende del valor actual de X_t, es decir, toda la información del pasado está recogida en el presente.

En la siguiente sección hablaremos del proceso de Wiener, que es Gaussiano y Markoviano al mismo tiempo, y se usa ampliamente en finanzas.

Figura 2: Una simulación de un proceso estocástico genérico

3.1. Proceso de Wiener

Para modelar activos como las acciones vamos a necesitar hacer uso de modelos matemáticos que incorporen las dos propiedades que hemos comentado en la sección anterior. Esto se debe a que en primer lugar, las variaciones porcentuales en el precio de las acciones tienden a seguir una distribución normal, los días de grandes subidas y bajadas en el precio de la acción son menos frecuentes que los días en los que la acción varía poco, y además el mercado incorpora toda la información del pasado en el momento actual (aunque también la información futura pero de momento obviaremos este hecho). El proceso de Wiener cumple estas dos condiciones y se puede definir de la siguiente forma:

Definición 9 *Un proceso de Wiener,* $\{W_t : t \geq 0\}$*, es un proceso estocástico continuo en el tiempo con las siguientes propiedades [8]:*

1. $W_0 = 0$ *(comienza en el origen).*
2. $W_t \sim \mathcal{N}(0, t)$ *para todo* $t \geq 0$*. Es decir, para cada* t*, la variable aleatoria* W_t *sigue una distribución normal con media* $E(W_t) = 0$ *y varianza* $Var(W_t) = E(W_t^2) = t$*.*
3. *Todos los incrementos* $\Delta W_t = W_{t+\Delta t} - W_t$ *en intervalos de tiempo no superpuestos son independientes. Es decir, las diferencias* $W_{t_2} - W_{t_1}$ *y* $W_{t_4} - W_{t_3}$ *son independientes para todo* $0 \leq t_1 < t_2 \leq t_3 < t_4$*.*

4. W_t depende de manera continua de t.

Adicionalmente, de la definición anterior se puede deducir la siguiente propiedad que usaremos posteriormente. Para un intervalo temporal $0 \leq s < t$, se cumple que:

$$W_t - W_s \sim \mathcal{N}(0, t-s) \tag{1}$$

Otra propiedad del proceso de Wiener que también nos será de utilidad es:

$$(dW_t)^2 = dt \tag{2}$$

El proceso de Wiener se puede entender también como un paseo aleatorio y es la componente que añadirá el carácter no determinista a los modelos de acciones cotizadas que desarrollaremos posteriormente. Un hecho relevante del proceso de Wiener es que es continuo en todo t pero no derivable en ningún punto, haciendo necesario usar técnicas de cálculo estocástico para poder trabajar con el. Para una explicación más detallada, consultar el capítulo 13 de [3]

3.2. Simulación del Proceso de Wiener

En esta sección simularemos el proceso de Wiener. Para ello es necesario discretizar W_t en forma de una suma de incrementos ΔW_t para un tiempo discreto $t_0, t_1, t_2, ..., t_k$. Dichos incrementos se encuentran normalmente distribuidos, ya que según (1) y para un intervalo temporal $\Delta t = t - s$, donde $0 \leq s < t$, se puede deducir que:

$$\mathbb{E}[\Delta W_t] = \mathbb{E}[W_t - W_s] = \mathbb{E}[W_t] - \mathbb{E}[W_s] = 0 \tag{3}$$

$$\mathbb{V}[\Delta W_t] = \mathbb{V}[W_t - W_s] = t - s = \Delta t \tag{4}$$

Luego podemos afirmar que los incrementos ΔW_t están distribuidos con media nula y varianza Δt. Esto facilita la simulación del proceso de Wiener ya que lo podemos expresar a partir de una variable aleatoria Z distribuida normalmente:

$$\Delta W_t = Z\sqrt{\Delta t} \quad \text{donde} \quad Z \sim \mathcal{N}(0,1) \tag{5}$$

La simulación del proceso de Wiener se puede implementar en Python de la siguiente forma:

Algoritmo 1 *Simulación de un proceso de Wiener*

```
import numpy as np
import plotly.express as px

# Wiener Process
delta_t = 0.01
steps = 1000
paths = 5

fig = px.line(title='Proceso de Wiener', labels={'x': 't', 'y':
    'W(t)'})

for _ in range(paths):
```

```
    t = np.arange(0, steps) * delta_t
    increments = np.random.normal(0, np.sqrt(delta_t), steps)
    wiener_process = np.cumsum(increments)
    fig.add_scatter(x=t, y=wiener_process)

fig.update_layout(xaxis_title='t', yaxis_title='X(t)',
    showlegend=False)
fig.show()
```

Figura 3: Realizaciones de un Proceso de Wiener

3.3. Proceso de Poisson

Otro de los procesos estocásticos frecuentemente usados en finanzas es el proceso de Poisson:

Definición 10 *El proceso estocástico* $\{J_t : t \geq 0\}$ *se llama proceso de Poisson si se cumplen las siguientes condiciones [8]:*

1. $J_0 = 0$
2. $J_t - J_s$ *toma valores enteros para* $0 \leq s < t < \infty$ *y*

$$P(J_t - J_s = k) = \frac{\lambda^k (t-s)^k}{k!} e^{-\lambda(t-s)}$$

   para $k = 0, 1, 2, \ldots$

3. *Los incrementos* $J_{t_2} - J_{t_1}$ *y* $J_{t_4} - J_{t_3}$ *son independientes para todo* $0 \leq t_1 < t_2 < t_3 < t_4$.

De nuevo, este proceso estocástico cumple la propiedad de Markov, y nos será de gran utilidad para modelar, por ejemplo, eventos imprevisibles en el mercado, como noticias, resultados trimestrales, catástrofes naturales... Hablaremos más del proceso de Poisson posteriormente.

3.4. Simulación del Proceso de Poisson

Como hicimos con el proceso de Wiener, podemos discretizar J_t como una suma de incrementos ΔJ_t para un tiempo discreto $t_0, t_1, t_2, ..., t_k$. Los incrementos $\Delta t = t - s$, para $0 \leq s < t$, están distribuidos según la distribución de Poisson, cumpliendo la segunda propiedad de la defínción anterior. Por tanto, se deduce lo siguiente:

$$P(\Delta J_t = k) = \frac{(\lambda \Delta t)^k}{k!} e^{-\lambda \Delta t} \tag{6}$$

De esta forma, podemos simular el proceso de Poisson como una suma de incrementos distribuidos según Poisson con media $\lambda \Delta t$. La implementación en Python es de la siguiente forma:

Algoritmo 2 *Simulación de un proceso de Poisson*

```
import numpy as np
import plotly.express as px

# Poisson Process
delta_t = 0.01
lambd = 10
steps = 100
paths = 1

fig = px.line(title='Proceso de Poisson', labels={'x': 't', 'y'
    : 'J(t)'})

for _ in range(paths):
    t = np.arange(0, steps) * delta_t
    increments = np.random.poisson(delta_t*lambd, steps)
    poisson_process = np.cumsum(increments)
    fig.add_scatter(x=t, y=poisson_process)

fig.update_layout(xaxis_title='t', yaxis_title='J(t)',
    showlegend=False)
fig.show()
```

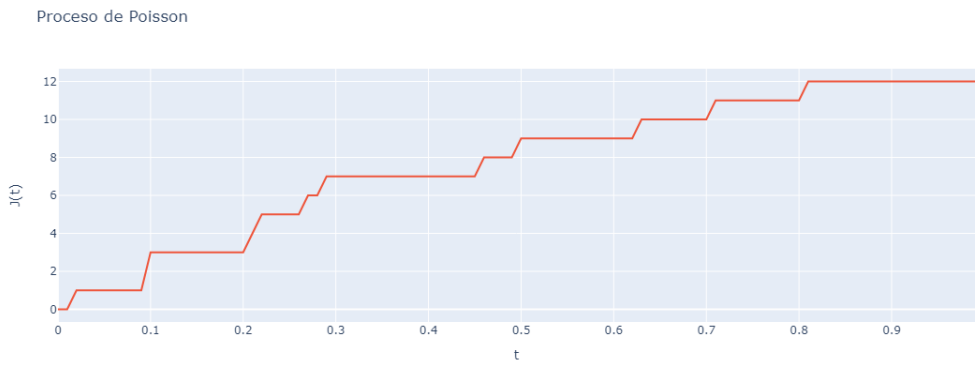

Figura 4: Realización de un Proceso de Poisson

4 Ecuaciones Diferenciales Estocásticas

Las ecuaciones diferenciales son el lenguaje de los fenómenos cambiantes. Nos permiten modelar sistemas dinámicos que cambian con el tiempo o en función de otras variables. Cuando las variaciones se dan respecto a una única variable, hablamos de ecuaciones diferenciales ordinarias (EDOs). Cuando estas variaciones se deben a múltiples variables, dichas ecuaciones reciben el nombre de ecuaciones en derivadas parciales (EDPs). Sin embargo, tanto las EDOs como las EDPs no son aptas para modelar fenómenos no deterministas o aleatorios, como los presentes en el mundo de las finanzas. Es por este motivo que surge la necesidad de combinar las EDPs junto con los procesos estocásticos, dando lugar a las ecuaciones diferenciales estocásticas (EDEs), que nos van a permitir, por ejemplo, explicar las fluctuaciones en el precio de una acción. Para una visión más en detalle de las EDPs, consultar [2].

4.1. Forma General

En su forma más general, podemos definir las EDEs mediante la siguiente ecuación:

$$dX_t = a(X_t, t)\,dt + b(X_t, t)\,dW_t \tag{7}$$

En dicha ecuación se pueden identificar los siguientes términos:

- dX_t: diferencial del proceso estocástico X_t que queremos encontrar.
- $a(X_t, t)$: coeficiente de *drift* o deriva. Puede ser una función de X_t y t.
- $b(X_t, t)$: coeficiente de *diffusion* o difusión. Puede ser una función de X_t y t.
- dW_t: un diferencial de un proceso de Wiener.
- dt: un diferencial de tiempo.

A esta ecuación también se la conoce como ecuación diferencial estocástica de Itô y nos permite trabajar con una amplia variedad de EDEs. Por poner algunos ejemplos, si fijamos $a = 0$ y $b = 1$ obtenemos:

$$dX_t = dW_t$$

por lo que nuestro proceso estocástico X_t es en verdad un proceso de Wiener genérico W_t. Alternativamente, si fijamos $b = 0$ obtenemos una ecuación diferencial determinista ya que al eliminar dW_t estamos eliminando la única componente aleatoria de la ecuación. Si integramos a ambos lados de la ecuación (7) y fijamos $X_{t_0} = X_0$ obtenemos la siguiente expresión:

$$X_t = X_0 + \int_{t_0}^{t} a(X_t, t)\,dt + \int_{t_0}^{t} b(X_t, t)\,dW_t. \tag{8}$$

La primera integral es una integral convencional (Lebesgue o Riemann), mientras que la segunda, al tener un dW_t, se trata de una integral estocástica o integral de Itô. Para resolver esta última integral será necesario emplear técnicas propias del cálculo estocástico (por ejemplo, el lema de Itô, que veremos posteriormente). Dado que las soluciones analíticas de estas ecuaciones son bastante complejas, optaremos por hacer uso de métodos numéricos para resolverlas. Para más información acerca de la integral de Itô, consultar el capítulo 13 de [3].

4.2. Simulación de EDEs

En esta sección emplearemos un conocido método númerico para resolver y posteriormente simualar las EDEs. Se trata del método Euler-Maruyama, una extensión del método de Euler para EDOs:

Definición 11 *La aproximación de Euler-Maruyama para una EDE de la forma* $dX_t = a(X_t, t)\,dt + b(X_t, t)\,dW_t$*, definida en un intervalo* $[0, T]$*, se puede calcular de la siguiente manera:*

1. *Particionar el intervalo* $[0, T]$ *en* N *subintervalos iguales de ancho* $\Delta t > 0$*:* $0 = \tau_0 < \tau_1 < \cdots < \tau_N = T$ *y* $\Delta t = T/N$*;*
2. *Definir* $Y_0 = X_0$*;*
3. *Definir recursivamente* Y_n *para* $0 \leq n \leq N-1$ *por:* $Y_{n+1} = Y_n + a(Y_n, \tau_n)\Delta t + b(Y_n, \tau_n)\Delta W_n$*.*

En otras palabras, el método discretiza la ecuación (7) dejándola de la siguiente forma:

$$\Delta Y_n = a(Y_n, \tau_n)\Delta t + b(Y_n, \tau_n)\Delta W_n \tag{9}$$

Y según (5), podemos expresar la EDE discretizada de forma que:

$$\Delta Y_n = a(Y_n, \tau_n)\Delta t + b(Y_n, \tau_n) Z\sqrt{\Delta t} \quad \text{donde} \quad Z \sim \mathcal{N}(0, 1) \tag{10}$$

Respecto a la implementación, para unos valores de a, b y X_0 conocidos podemos calcular la solución de forma recursiva como en el punto 3 de la definición del método de Euler-Maruyama o como una suma de incrementos ΔY_n según (9). En nuestro caso, hemos optado por calcular la solución de forma recursiva a través del siguiente algoritmo en Python:

Algoritmo 3 *Simulación de una EDE a través del método Euler-Maruyama*

```
# SDEs Euler-Maruyama
def SDE(dt, a, b, X0, steps):

    t = np.arange(0, steps+1)*dt
    Y = np.zeros(steps + 1)
    Y[0] = X0

    for i in range(1, steps+1):
        dW = np.random.normal(0, np.sqrt(dt), 1)
        Y[i] = Y[i - 1] + a*dt + b*dW

    return Y, t

dt = 1/500; a = 2; b = 2; X0 = 50
steps = int(1/dt)
paths = 100

fig = px.line(title='Proceso Estocastico', labels={'x': 't', 'y
    ': 'Y(t)'})
```

```
19
20 for j in range(paths):
21     Y, t = SDE(dt, a, b, X0, steps)
22     fig.add_scatter(x=t, y=Y)
23
24 fig.update_layout(xaxis_title='t', yaxis_title='Y(t)',
       showlegend=False)
25 fig.show()
```

Figura 5: Simulación de una EDE con $a = 2$, $b = 2$ y $X_0 = 50$

4.3. Lema de Itô

Si queremos resolver las EDEs de forma analítica, hemos de recurrir a lo que es posiblemente el arma más potente del cálculo estocástico, es decir, al Lema de Itô. Se trata del equivalente estocástico de la regla de la cadena, propia del cálculo tradicional y nos permite diferenciar procesos estocásticos. El objetivo de esta sección es aportar una demostración informal del Lema de Itô, sin entrar en excesivos detalles técnicos. Supongamos que tenemos una función determinista $f(t, x)$. Su desarrollo en serie de Taylor se puede expresar de la siguiente forma:

$$\begin{aligned} f(t,x) =& f(t_0,x_0) + \frac{\partial f}{\partial t}(t_0,x_0)(t-t_0) + \frac{\partial f}{\partial x}(t_0,x_0)(x-x_0) \\ &+ \frac{1}{2!}\left(\frac{\partial^2 f}{\partial t^2}(t_0,x_0)(t-t_0)^2 + 2\frac{\partial^2 f}{\partial t \partial x}(t_0,x_0)(t-t_0)(x-x_0)\right. \\ &\left. + \frac{\partial^2 f}{\partial x^2}(t_0,x_0)(x-x_0)^2\right) + \cdots \end{aligned} \tag{11}$$

A continuación, diferenciamos ambos lados de la expresión, es decir, consideramos pequeños cambios dt y dx en t y x respectivamente, alrededor del punto (t_0, x_0). De esta forma, y compactando un poco la notación, obtenemos la siguiente expresión:

$$df = \frac{\partial f}{\partial t}\,dt + \frac{\partial f}{\partial x}\,dx + \frac{1}{2}\left(\frac{\partial^2 f}{\partial t^2}\,dt^2 + 2\frac{\partial^2 f}{\partial t \partial x}\,dt\,dx + \frac{\partial^2 f}{\partial x^2}\,dx^2\right) + \ldots \tag{12}$$

Esta ecuación se puede entender como la versión diferencial del desarrollo en serie de Taylor.

Volvamos ahora a la expresión general de las EDEs que hemos enunciado anteriormente, es decir:

$$dX_t = a(X_t, t)\, dt + b(X_t, t)\, dW_t$$

El siguiente paso es sustituir dicha expresión en (12), haciendo que $dx = dX_t$ y $f(t, X_t)$, obteniendo:

$$\begin{aligned} df =& \frac{\partial f}{\partial t}\, dt + \frac{\partial f}{\partial X_t}\, (a(X_t, t)\, dt + b(X_t, t)\, dW_t) \\ &+ \frac{1}{2}\left(\frac{\partial^2 f}{\partial t^2}\, dt^2 + 2\frac{\partial^2 f}{\partial t \partial X_t}\, dt\, (a(X_t, t)\, dt + b(X_t, t)\, dW_t) \right. \\ &\left. + \frac{\partial^2 f}{\partial X_t^2}\, (a(X_t, t)\, dt + b(X_t, t)\, dW_t)^2 \right) + \dots \end{aligned} \tag{13}$$

De este modo, la ecuación contiene únicamente diferenciales dt y dW_t. Operando algunos términos, abreviando $a = a(X_t, t)$ y $b = b(X_t, t)$ y teniendo en cuenta que $(dW_t)^2 = dt$ llegamos a la siguiente expresión:

$$\begin{aligned} df =& \frac{\partial f}{\partial t}\, dt + \frac{\partial f}{\partial X_t}\, (a\, dt + b\, dW_t) + \frac{1}{2}\left(\frac{\partial^2 f}{\partial t^2}\, dt^2 + 2\frac{\partial^2 f}{\partial t \partial X_t}\, (a dt^2 + b\, dW_t\, dt) \right. \\ &\left. + \frac{\partial^2 f}{\partial X_t^2}\, (a^2 dt^2 + b^2\, dt + 2ab\, dt\, dW_t) \right) + \dots \end{aligned} \tag{14}$$

En el límite, cuando dt es suficientemente pequeño, es decir, si $dt \to 0$, podemos despreciar los términos dt^2 y $dt\, dW_t$ (y los términos sucesivos) ya que tienden a cero mucho más rápido que dt. Por tanto, obtenemos:

$$df = \frac{\partial f}{\partial t}\, dt + \frac{\partial f}{\partial X_t}\, (a\, dt + b\, dW_t) + \frac{1}{2}\frac{\partial^2 f}{\partial X_t^2}\, b^2\, dt \tag{15}$$

Para finalizar, el diferencial de la función f, que depende del proceso estocástico X_t se puede expresar de la siguiente forma:

$$df = \frac{\partial f}{\partial t}\, dt + \frac{\partial f}{\partial X_t}\, dX_t + \frac{1}{2}\frac{\partial^2 f}{\partial X_t^2}\, b(X_t, t)^2\, dt \tag{16}$$

Esta es la expresión del famoso Lema de Itô [1] y nos permite diferenciar funciones deterministas que dependen de procesos estocásticos, como puede ser el caso de un derivado financiero, como una opción, cuyo precio depende de la cotización del subyacente. Por hacer una comparativa con el cálculo tradicional, el diferencial de una función $f(x, t)$ se puede expresar de esta forma:

$$df = \frac{\partial f}{\partial x}\, dx + \frac{\partial f}{\partial t}\, dt$$

Por tanto, la versión estocástica añade el término $\frac{1}{2}\frac{\partial^2 f}{\partial X_t^2}\, b(X_t, t)^2\, dt$ a la expresión anterior. En secciones posteriores haremos uso del Lema de Itô para trabajar con EDEs.

4.4. Movimiento Browniano Geométrico

Una de las EDEs más utilizadas en las finanzas matemáticas es el Movimiento Browniano Geométrico (MBG). Esta ecuación es la base de todas las aplicaciones prácticas que desarrollaremos en los siguientes capítulos ya que nos permite caracterizar el comportamiento de activos cotizados. Supongamos el caso de activo (subyacente) cotizado con una tasa de retorno constante. La expresión que define el comportamiento del activo es la siguiente:

$$dS_t = \mu S_t dt \tag{17}$$

Donde S_t es el precio del activo y μ es la tasa de retorno. Se trata de una EDO de primer orden. Reordenando e integrando a ambos lados obtenemos:

$$\int \frac{1}{S_t} dS_t = \int \mu dt \tag{18}$$

Y haciendo $S(0) = S_0$ llegamos a la siguiente solución final:

$$S_t = S_0 e^{\mu t} \tag{19}$$

En otras palabras, la ecuación (17) representa la fórmula de la capitalización compuesta (en tiempo continuo) para una inversión inicial S_0, que crece con tasa μ. Esta EDO nos permite modelar activos con retornos deterministas, pero falla a la hora de explicar activos que presentan retornos aleatorios como por ejemplo las acciones. Es por esto que debemos de añadir un término a la ecuación que nos permita dotarla del carácter no determinista que buscamos. Aquí es donde entra el Movimiento Browniano Geométrico:

Definición 12 *Un proceso estocástico S_t se denomina Movimiento Geométrico Browniano (MGB) si satisface la siguiente ecuación diferencial estocástica [5]:*

$$dS_t = \mu S_t dt + \sigma S_t dW_t \tag{20}$$

Lo que diferencia a esta ecuación de la (17) es que hemos añadido un término adicional que se compone de un diferencial de un proceso de Wiener, dW_t, multiplicado por una constante σ, que representa la volatilidad del activo, y por el propio proceso estocástico S_t. Al añadir este término adicional conseguimos dotar a la ecuación de ese carácter aleatorio que buscábamos, ya que hemos añadido un componente estocástico. El MBG es un caso particular de (7), en el que $a(X_t, t) = \mu S_t$ y $b(X_t, t) = \sigma S_t$. En las siguientes secciones resolveremos la EDE mediante el uso de métodos numéricos (usando el método Euler-Maruyama) y de forma analítica vía Lema de Itô.

4.5. Resolución del MBG: Método Euler-Maruyama

Partiendo de la expresión (10), podemos definir el MBG discretizado de la siguiente forma:

$$\Delta S_n = \mu S_n \Delta t + \sigma S_n Z \sqrt{\Delta t} \quad \text{donde} \quad Z \sim \mathcal{N}(0,1) \tag{21}$$

A continuación presentamos el código en Python para simular dicho proceso estocástico:

Algoritmo 4 *Simulación del MBG a través del método Euler-Maruyama*

```python
import numpy as np
import plotly.graph_objects as go

# GBM Euler-Maruyama
def GBM_num(mu, sigma, S0, steps):
    dt = 1 / steps
    t = np.arange(0, steps + 1) * dt

    S = np.zeros(steps + 1)
    S[0] = S0

    for i in range(1, steps + 1):
        dW = np.random.normal(0, np.sqrt(dt), 1)
        S[i] = S[i - 1] + mu * S[i - 1] * dt + sigma * S[i - 1]
            * dW
    return S, t

mu = 0.1; sigma = 0.2; S0 = 50; steps = 1000; paths = 20

fig = go.Figure()

for j in range(paths):
    S, t = GBM_num(mu, sigma, S0, steps)
    fig.add_trace(go.Scatter(x=t, y=S, mode='lines', name=f'
       Path {j+1}'))

fig.update_layout(title='Movimiento Browniano Geometrico',
    xaxis_title='t',yaxis_title='S(t)',showlegend=False).show()
```

Figura 6: MBG con $S_0 = 50$, $\sigma = 0.2$, $\mu = 0.1$

4.6. Resolución del MBG: Lema de Itô

Si aplicamos las técnicas del cálculo tradicional para resolver (20), nos encontramos con la siguiente expresión:

$$\int_{t_0}^{t} dS_t = \int_{t_0}^{t} \mu S_t dt + \int_{t_0}^{t} \sigma S_t dW_t$$

Las dos primeras integrales son deterministas y se pueden calcular de forma analítica sin problema. Sin embargo, la última integral no tiene solución debido a que estamos integrando respecto a un término estocástico dW_t, que no tiene derivada, es decir, se trata de una integral estocástica. Una forma alternativa de resolver la ecuación del MBG es a través del Lema de Itô, cuya expresión hemos deducido anteriormente y es la siguiente:

$$df = \frac{\partial f}{\partial t}\,dt + \frac{\partial f}{\partial X_t}\,dX_t + \frac{1}{2}\frac{\partial^2 f}{\partial X_t^2}\,b(X_t,t)^2\,dt \tag{22}$$

Para ello necesitamos una función determinista suficientemente diferenciable [7], que en nuestro caso será:

$$f(x,t) = ln(x)$$

Las derivadas de dicha función son las siguientes:

$$\begin{cases} f(x) & = \frac{1}{x} \\ f''(x) & = -\frac{1}{x^2} \\ f'(t) & = 0 \end{cases}$$

Por tanto, la expresión del Lema de Itô quedará de la siguiente forma:

$$d\left(\ln(S_t)\right) = \left(0 + \mu S_t\frac{1}{S_t} + \frac{1}{2}\left(\sigma S_t\right)^2\left(-1\left(\frac{1}{S_t}\right)^2\right)\right)dt + \sigma S_t\frac{1}{S_t}dW_t \tag{23}$$

A continuación, haciendo algunas manipulaciones algebraicas, llegamos a:

$$d\left(\ln(S_t)\right) = (\mu - \frac{1}{2}\sigma^2)dt + \sigma dW_t \tag{24}$$

Para resolver la ecuación, supongamos, sin pérdida de generalidad, que los límites de integración son 0 y un instante temporal T. Por tanto, obtenemos:

$$\int_0^T d(\ln(S_t)) = \int_0^T \left(\mu - \frac{1}{2}\sigma^2\right)dt + \int_0^T \sigma dW_t \tag{25}$$

En este caso, la integral de la derecha si que se puede obtener de forma analítica, ya que solo es el diferencial del proceso de Wiener, dW_t. Hemos resuelto el problema de la integral estocástica mediante el Lema de Itô. Integrando, llegamos a la siguiente expresión:

$$\ln(S_T) - \ln(S_0) = \left(\mu - \frac{1}{2}\sigma^2\right)T + \sigma\left(W_T - W_0\right) \tag{26}$$

Aplicando la propiedad $W_0 = 0$, deshaciendo el logaritmo y sustituyendo T por t (por conveniencia) se obtiene:

$$S_t = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma W_t} \tag{27}$$

Esta es la solución analítica del MBG. Dicha expresión implica $S_t > 0$, lo cual tiene bastante sentido ya que los subyacentes cotizados no tienen precios negativos. Podemos identificar dos términos principales, uno determinista, que tomando logaritmos correspondería con $(\mu - \frac{1}{2}\sigma^2)t$, y otro estocástico, σW_t. Esto es precisamente lo que buscábamos inicialmente, un modelo que incorporase variaciones aleatorias a

un activo con cierta tasa de retorno. Podemos simular también esta ecuación discretizando W_t como hicimos en (5):

$$S_t = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma Z\sqrt{\Delta t}} \quad \text{donde} \quad Z \sim \mathcal{N}(0,1) \tag{28}$$

Podemos simular la solución analítica del MBG de la siguiente forma:

Algoritmo 5 *Simulación del MBG a través de su solución analítica*

```
# GBM analytical solution:
import plotly.express as px
import numpy as np

def GBM(mu,sigma,S0,steps,paths):

    dt = 1/steps
    S = np.zeros((steps+1, paths))
    S[0] = S0

    for t in range(1,steps+1):
        S[t] = S[t-1] * np.exp((mu-0.5*sigma**2)*dt + sigma*np.
           sqrt(dt)*np.random.standard_normal(paths))

    return S

mu = 0.1; sigma = 0.2; S0 = 50; steps = 1000; paths = 20
GBM(mu, sigma, S0, steps, paths)

fig = px.line(S,title='Movimiento Browniano Geometrico')
fig.update_layout(xaxis_title='t', yaxis_title='S(t)',
    showlegend=False)
fig.show()
```

Figura 7: MBG con $S_0 = 50$, $\sigma = 0.2$, $\mu = 0.1$

4.7. Procesos de Salto-Difusión

En esta sección exploraremos otro proceso estocástico muy usado en finanzas, el proceso de Salto-Difusión (SD). Se trata de una variante del MBG que añade un componente de "salto"que ocurre con cierta frecuencia y una intensidad variable. Esta variación brusca nos permite modelar eventos inesperados en los mercados financieros. Por ejemplo, cuando una empresa cotizada bate resultados trimestrales, superando las expectativas, o cuando se publica una noticia que perjudica a una compañía, o también ante la publicación de datos macroeconómicos que se alejan de las previsiones. Podemos ver este tipo de fenómenos a diario en los mercados. El proceso de salto difusión se modela mediante la siguiente EDE:

Definición 13 *Un proceso estocástico S_t se denomina Proceso de Salto-Difusión (SD) si satisface la siguiente ecuación diferencial estocástica [8]:*

$$dS_t = S_t\mu dt + S_t\sigma dW_t + S_t(Q_t - 1)dJ_t \tag{29}$$

La ecuación anterior consiste en un MBG con un término adicional $S_t(Q_t-1)dJ_t$ que es el encargado de añadir el componente de salto. Este término consta de dos procesos estocásticos (sin tener en cuenta S_t). En primer lugar, J_t, que es un proceso de Poisson (10) y representa la frecuencia del salto, y en segundo lugar, Q_t, que tiene una distribución lognormal (más sobre esta distribución en la siguiente sección), que nos habla de la intensidad del salto. La expresión $S_t(Q_t - 1)dJ_t$ se deduce de la siguiente forma [8]. Consideremos un salto como una diferencia de precios en S_τ:

$$\Delta S_\tau = S_{\tau^+} - S_{\tau^-}$$

Donde S_{τ^+} es el momento posterior al salto y S_{t^-} el momento anterior a este. Podemos considerar que se trata de un salto proporcional al precio, luego:

$$S_{\tau^+} = qS_{\tau^-}$$

Siendo $q > 0$. De esta forma, se deduce que:

$$\Delta S = (q - 1)S_{\tau^-}$$

Si queremos generalizar este modelo a varios saltos, cuya frecuencia de ocurrencia viene dada por un proceso de Poisson J_t y su intensidad por un proceso con distribución lognormal Q_t obtenemos la siguiente expresión:

$$dS_t = S_t(Q_t - 1)dJ_t$$

Respecto a la solución analítica del proceso SD, podemos reescribir la ecuación (29) de la siguiente forma:

$$S_t = S_0 + \int_0^t S_t(\mu \mathrm{d}t + \sigma \mathrm{d}W_t) + \sum_{j=1}^{J_t} S_{t_j}(Q_{t_j} - 1) \tag{30}$$

Podemos ver la integral del MBG que hemos resuelto vía Lema de Itô, y un proceso de Poisson compuesto que es el responsable de dotar de saltos al modelo. Alternativamente y haciendo $Y_t = \ln(S_t)$ podemos expresar la ecuación anterior de la siguiente forma:

$$Y_t = Y_0 + \left(\mu - \frac{\sigma^2}{2}\right) t + \sigma W_t + \sum_{j=1}^{J_t} \ln(Q_{t_j}) \tag{31}$$

Esta es una forma más compacta de expresar el proceso de SD y nos servirá para simularlo de una forma más sencilla. Para ello, discretizaremos (5) los dos procesos estocásticos gaussianos W_t y $\ln(Q_t)$ de la siguiente forma:

$$Y_t = Y_0 + \left(\mu - \frac{\sigma^2}{2}\right) t + \sigma Z_1 \sqrt{\Delta t} + \sum_{j=1}^{J_1} \mu_{J_j} + \sigma_{J_j} Z_{2_j} \tag{32}$$

Donde Z_1 y $Z_2 \sim \mathcal{N}(0,1)$ y μ_J, σ_J son la media y la desviación estándar del salto. Además $J_1 \sim Poi(\lambda\sqrt{\Delta t})$, siendo λ el parámetro que define la frecuencia de ocurrencia de los saltos. Para llegar a esta expresión hemos hecho:

$$\ln(Q_t) = \mu_J + \sigma_J Z_2$$

Es necesario destacar cómo en el proceso de SD tenemos 5 parámetros $(\mu, \mu_J, \sigma, \sigma_J, \lambda)$ en lugar de los dos del MBG (μ, σ). Una vez hemos discretizado el proceso de SD, estamos listos para simularlo en Python:

Algoritmo 6 *Simulación del proceso Salto-Difusión a través de su solución analítica*

```
import numpy as np
import plotly.express as px

def JumpDiff_Merton(S0, mu, sigma, lamb, mu_J, sigma_J, steps,
    paths):

    dt = 1/steps
    r_J = lamb*(np.exp(mu_J + 0.5*sigma_J**2)-1)
    # Correccion del drift para mantener neutralidad al riesgo

    Z1 = np.random.standard_normal((steps+1, paths))
    Z2 = np.random.standard_normal((steps+1, paths))
    J1 = np.random.poisson(lamb*dt, (steps+1, paths))

    cum_poi = np.multiply(J1, mu_J + sigma_J*Z2).cumsum(axis =
       0)
    gbm = np.cumsum(((mu -  sigma**2/2 -r_J)*dt + sigma*np.sqrt
       (dt) * Z1), axis=0)

    return np.exp(cum_poi + gbm)*S0

S0 = 100; mu = 0.05; sigma = 0.25; lamb = 1; mu_J = 0.025;
    sigma_J = 0.15; steps = 500; paths = 10

S = JumpDiff_Merton(S0, mu, sigma, lamb, mu_J, sigma_J, steps,
    paths)

fig = px.line(S,title='Proceso de Salto-Difusion')
fig.update_layout(xaxis_title='t', yaxis_title='S(t)',
    showlegend=False)
fig.show()
```

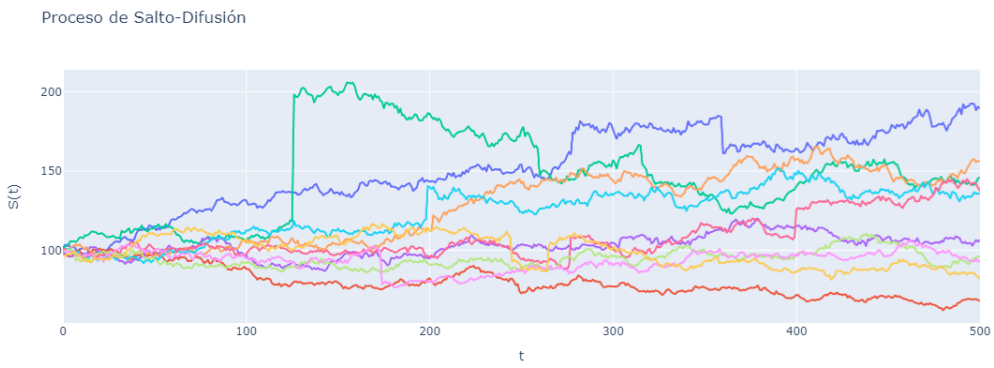

Figura 8: Proceso SD con $S_0 = 100$, $\sigma = 0.25$, $\mu = 0.05$, $\mu_J = 0.025$, $\sigma_J = 0.15$ y $\lambda = 1$

5 Estimación de Parámetros

En esta sección profundizaremos en los modelos del Movimiento Browniano Geométrico y en los de Salto-Difusión mediante la estimación de sus parámetros, que hasta ahora hemos definido de forma arbitraria para las simulaciones. Como hemos comentado anteriormente, el MBG consta de dos parámetros (μ, σ) y el de SD tiene 5 $(\mu, \mu_J, \sigma, \sigma_J, \lambda)$. Para ello usaremos dos técnicas, la de momentos estadísticos y la de máxima verosimilitud. Pero antes, hablaremos del modelo log-normal y de los retornos logarítmicos.

5.1. Modelo Log-Normal

Recordemos la solución analítica del MBG (27):

$$S_t = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma W_t}$$

Si calculamos la distribución de S_t podemos apreciar lo siguiente:

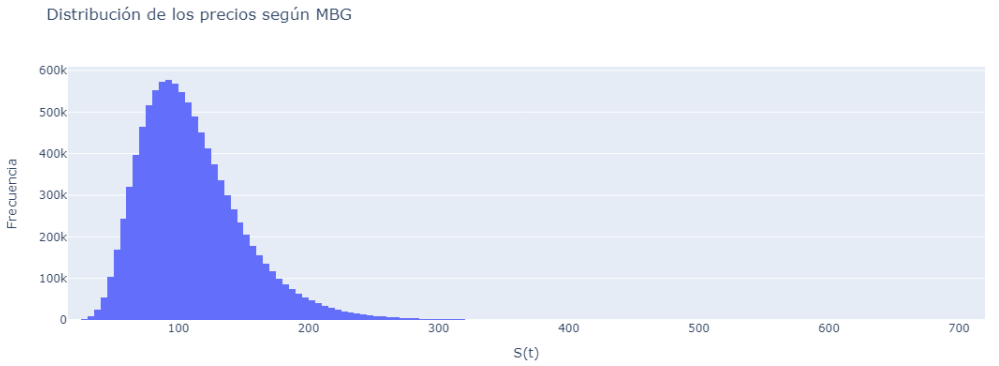

Figura 9: Distribución de los precios de un MBG

Es decir, los precios se distribuyen según una distribución log-normal. Por tanto, si tomamos el logaritmo de dicha distribución obtenemos una distribución gaussiana:

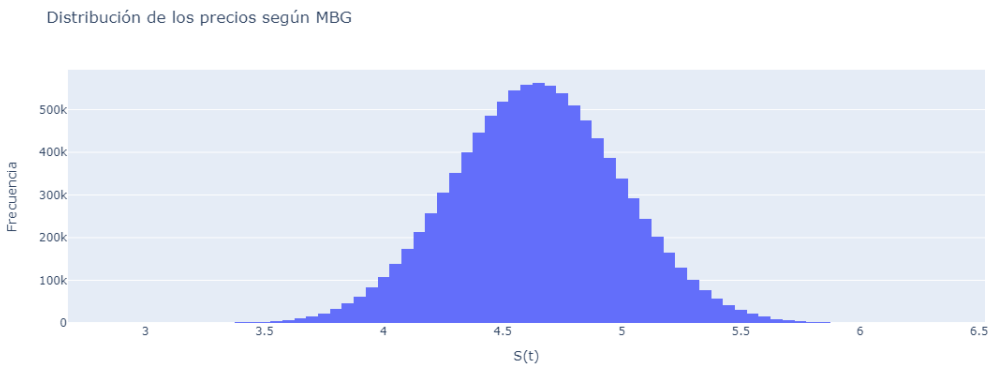

Figura 10: Distribución del logaritmo de un MBG

Por lo general, vamos a preferir trabajar con distribuciones normales, luego una práctica frecuente será tomar el logaritmo de los precios como hemos hecho ahora con el MBG y como hicimos en (31) para el proceso de SD. Esto viene motivado por el hecho de que muchos teoremas de las finanzas matemáticas asumen normalidad. Es por este motivo (y por otros que veremos a continuación), que trabajaremos con los retornos logarítmicos en lugar de los lineales.

5.2. Retornos Logarítmicos

Como hemos comentado, es una práctica bastante común en finanzas utilizar los retornos logarítmicos para medir las variaciones relativas en el precio de los activos. Podemos calcular los retornos logarítmicos de la siguiente forma:

$$R_i = \ln\left(\frac{S_{i+1}}{S_i}\right) \tag{33}$$

Se suelen emplear los retornos logarítmicos en lugar de los lineales ya que tienen diversas ventajas. Una de ellas es que al tomar logaritmos, los retornos se hacen aditivos en el tiempo, es decir, podemos aplicar el *Teorema Central del Límite* ya que la suma de estos retornos logarítmicos (aleatorios), se distribuirá normalmente. Además, a nivel de cálculo numérico, es más sencillo operar con estos retornos ya que presentan mayor estabilidad numérica, ya que para obtener el retorno de un periodo solo hay que sumar los retornos en lugar de multiplicarlos como haríamos con los lineales. En el caso del MBG, podemos calcular sus retornos (logarítmicos) como [7]:

$$R_i = \ln\left(\frac{S(t_{i+1})}{S(t_i)}\right) = \left(\mu - \frac{1}{2}\sigma^2\right)(t_{i+1} - t_i) + \sigma(W(t_{i+1}) - W(t_i)) \tag{34}$$

Donde hemos evaluado el retorno teniendo en cuenta dos instantes temporales t_i y t_{i+1}. Si seguimos operando, haciendo $\Delta t = t_{i+1} - t_i$ y aplicamos algunas de las propiedades del proceso de Wiener (9), llegamos a la siguiente expresión:

$$R_i = \left(\mu - \frac{1}{2}\sigma^2\right)\Delta t + \sigma Z\sqrt{\Delta t} \quad \text{donde} \quad Z \sim \mathcal{N}(0,1) \tag{35}$$

Esta es la formula para calcular el retorno entre dos instantes temporales t_i y t_{i+1}, y presenta una distribución normal.

Si atendemos a (35), podemos observar que $R_i \sim \mathcal{N}\left(\left(\mu - \frac{1}{2}\sigma^2\right)\Delta t, \sigma^2\Delta t\right)$, luego podemos definir la esperanza matemática y la varianza de los retornos del MBG de esta forma:

$$\begin{cases} \mathbb{E}[R_i] = (\mu - \frac{1}{2}\sigma^2)\Delta t \\ \mathbb{V}[R_i] = \sigma^2\Delta t \end{cases} \tag{36}$$

Acabamos de deducir los momentos estadísticos del MBG, que nos serán de gran ayuda para estimar sus parámetros (μ, σ). Respecto a los momentos estadísticos del proceso de SD, podemos calcular la esperanza y la media de los retornos de dicho proceso como [11]:

$$\begin{cases} \mathbb{E}[R_i] = (\mu - \frac{1}{2}\sigma^2)\Delta t + \mu_J\lambda\Delta t \\ \mathbb{V}[R_i] = \sigma^2\Delta t + (\sigma_J^2 + \mu_J^2)\lambda\Delta t \end{cases} \tag{37}$$

5.3. Método de Momentos Estadísticos

Llegó el momento de estimar los parámetros de ambos modelos. Lo haremos usando dos técnicas estadísticas como son el Método de Momentos Estadísticos (MME) y el Método de Máxima Verosimilitud (MMV). En el primer método, nuestro objetivo va a ser igualar los momentos estadísticos anteriormente calculados a los muestrales. Esto da lugar a un sistema de ecuaciones que debemos resolver para obtener las estimaciones de los parámetros. La media y la varianza muestrales de un conjunto de datos de longitud N se calculan de la siguiente forma [7]:

$$\bar{u} = \frac{1}{N}\sum_{i=0}^{N-1} R_i, \quad s^2 = \frac{1}{N-1}\sum_{i=0}^{N-1}(R_i - \bar{u})^2 \tag{38}$$

Para el caso del MBG, si igualamos (38) a (36) obtenemos el siguiente sistema de ecuaciones:

$$\begin{cases} \bar{u} = (\hat{\mu} - \frac{1}{2}\sigma^2)\Delta t \\ s^2 = \hat{\sigma}^2\Delta t \end{cases} \tag{39}$$

Y resolviendo es sistema, obtenemos las estimaciones de los parámetros $(\hat{\mu}, \hat{\sigma})$:

$$\hat{\mu} = \frac{\bar{u} + \frac{s^2}{2}}{\Delta t}, \quad \hat{\sigma} = \frac{s}{\sqrt{\Delta t}} \tag{40}$$

La elección de Δt depende del periodo temporal de estudio, en el caso de datos diarios, podemos tomar el número de días al año que cotizan las bolsas, es decir, $\Delta t = \frac{1}{252}$. Además, cuanto mayor sea N, mejores serán las estimaciones de los parámetros $(\hat{\mu}, \hat{\sigma})$. Otro tema un tanto más complejo es el de estimar los parámetros del modelo de SD. Para una visión más en detalle, consultar el capítulo 7.3 de [11], ya que aquí solo daremos unas pinceladas. La cuestión es estimar tres de los cinco parámetros $(\hat{\mu_J}, \hat{\sigma_J}, \hat{\lambda})$ ya que los dos restantes se estiman igual para el MBG y para SD. Empecemos por estimar $\hat{\lambda}$, podemos hacerlo de la siguiente forma:

$$\hat{\lambda} = \frac{\text{Número total de saltos}}{\text{Duración de la muestra en años}}$$

Aquí se nos plantea un importante problema: ¿Qué es un salto? De forma intuitiva, podemos pensar que es un retorno (logarítmico) anómalo que hace que el precio

suba o baje de una forma abrupta. Si intentamos definir con rigor esta idea, podemos considerar una tolerancia ϵ a partir de la cual los retornos se consideran con salto. Por tanto, podemos separar los retornos R_i en retornos con salto R_i^J, y retornos sin salto R_i^D. Cuando no hay salto, estimaremos $(\hat{\mu}, \hat{\sigma})$, como hemos hecho hasta ahora (40). Para los retornos con salto tenemos que:

$$\begin{cases} \mathbb{E}[R_i^J] = (\mu - \frac{1}{2}\sigma^2)\Delta t + \mu_J \\ \mathbb{V}[R_i^J] = \sigma^2 \Delta t + \sigma_J^2 \end{cases} \tag{41}$$

Y definiendo los momentos muestrales del conjunto de datos con salto como $\bar{u}_J$ y s_J^2, que se calculan igual que en (38) e igualándolos a (41) tenemos que:

$$\begin{cases} \bar{u}_J = (\hat{\mu} - \frac{1}{2}\hat{\sigma}^2)\Delta t + \hat{\mu}_J \\ s_J^2 = \hat{\sigma}^2 \Delta t + \hat{\sigma}_J^2 \end{cases} \tag{42}$$

Despejando y sustituyendo las expresiones de (40) llegamos finalmente al siguiente resultado:

$$\hat{\mu}_J = \bar{u}_J - \bar{u}, \quad \hat{\sigma}_J = \sqrt{s_J^2 - s^2} \tag{43}$$

Por tanto, para estimar los parámetros del proceso SD solo tendremos que dividir el conjunto de datos en R_i^J y R_i^D según una tolerancia ϵ, obtener λ en base a esto, calcular los momentos muestrales de cada conjunto, definir un Δt apropiado al tamaño de los datos, y usar las expresiones (40) y (43) para obtener las estimaciones. La elección de ϵ, es por tanto un asunto de suma importancia ya que define tres de los cinco parámetros del modelo. No hemos podido encontrar ninguna expresión analítica para un ϵ óptimo, por lo que a priori lo estimaremos de forma empírica. Posteriormente, en la sección 6, trataremos este tema más en detalle.

5.4. Método de Máxima Verosimilitud

En esta sección aplicaremos un método estadístico un tanto más sofisticado para estimar los parámetros del modelo de Salto-Difusión. Se trata de el Método de Máxima Verosimilitud y consiste en maximizar una función llamada función de verosimilitud que representa la probabilidad de observar un conjunto de datos $X = \{x_1, x_2, \cdots, x_n\}$, dado una serie de parámetros θ, que en nuestro caso será $\theta = (\hat{\mu}, \hat{\mu_J}, \hat{\sigma}, \hat{\sigma_J}, \hat{\lambda})$. Los valores de X serán los retornos logarítmicos. A continuación, definiremos la función de verosimilitud:

Definición 14 *La función de verosimilitud $L(X;\theta)$ se puede definir como la probabilidad de observar el conjunto de datos X dados los parámetros θ y su expresión es la siguiente [12]:*

$$L(X;\theta) = f(x_1;\theta)f(x_2;\theta)\cdots f(x_n;\theta) = \prod_{i=1}^{n} f(x_i;\theta) \tag{44}$$

Donde $f(x;\theta)$ es la función densidad de probabilidad con la que pretendemos modelar la distribución de X, con parámetros θ. Es una práctica bastante común la de tomar el logaritmo de dicha función, cambiar su signo y minimizarla. Por tanto, podemos reescribir la ecuación anterior como:

$$-\ln L(\theta; x) = -\sum_{i=0}^{n} \ln f(x_i; \theta) \tag{45}$$

Sin entrar mucho en detalle (consultar [11] para más), podemos definir la función de densidad de probabilidad de los retornos (logarítmicos) del proceso SD como:

$$f(x; \theta) = \sum_{k=0}^{\infty} p_k(\lambda \Delta t) \varphi\big(x; (\mu - \frac{\sigma^2}{2})\Delta t + \mu_J k, \sigma^2 \Delta t + \sigma_J^2 k\big) \tag{46}$$

De cara a la implementación, lo que haremos será dar unas estimaciones de parámetros iniciales, θ_0, obtenidos con el MME, y posteriormente minimizar la función $-\ln L(\theta; x)$ a través de métodos numéricos, para obtener unos resultados más precisos. Para ello, daremos diferentes valores a ϵ y comprobaremos cómo varían los parámetros calculados.

5.5. Validación del Modelo

Una vez hemos estimado los parámetros del modelo, es hora de poner a prueba su validez estadística [7]. Para ello simularemos varias realizaciones del proceso de salto-difusión y calcularemos las log-cotizaciones y la media aritmética de estas para cada instante t_i, denotándolas como $\tilde{\nu}_i$. Compararemos dichos valores con las log-cotizaciones reales $\nu_i = \ln S_i$, usando estas dos medidas de bondad del ajuste:

$$\text{RSME (Root Square Mean Error)} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (\nu_i - \tilde{\nu}_i)^2} \tag{47}$$

$$\text{MAPE (Mean Absolute Porcentual Error)} = \frac{1}{N}\sum_{i=1}^{N} \frac{|\nu_i - \tilde{\nu}_i|}{\nu_i} \tag{48}$$

La primera es una suma total de distancias euclídeas entre los datos reales y los aportados por el modelo y la segunda medida es una suma total de errores relativos entre los datos reales y los aportados por el modelo. Buscaremos siempre que el RSME sea lo menor posible y que el MAPE no supere el 5 %.

6 Aplicaciones de los Procesos Estocásticos

En esta última parte, aplicaremos los conocimientos que hemos ido adquiriendo a lo largo de todo este proyecto y los emplearemos en un caso real. Nos pondremos en la piel de un analista cuantitativo de un importante banco americano, y llevaremos a cabo las siguientes tareas:

1. **Predicción y Simulación:** Construcción una cartera de un único activo, el cual modelaremos con un proceso SD. Para ello, será necesario estimar los parámetros del modelo a través de los datos históricos, por medio de los métodos de momentos estadísticos y de máxima verosimilitud. Posteriormente simularemos su comportamiento futuro y comprobaremos cómo de válido es a nivel estadístico mediante el MAPE y el RSME.

2. **Medición de Riesgos (VaR):** Una vez listo el modelo de simulación, supondremos que la cartera pertenece al banco americano. Tendremos que estimar las posibles pérdidas derivadas de las fluctuaciones del precio del activo en la cartera. Construiremos un modelo *Value at Risk* (VaR) para estimar las pérdidas potenciales dado un nivel de confianza. Posteriormente usaremos un modelo CVAR para obtener una visión más precisa del riesgo de pérdida.

3. **Valoración: Modelo *Black-Scholes***: Supongamos ahora que dicho banco quiere ofrecer un derivado financiero, en concreto una opción *call* o *put*, a un cliente. Calcularemos el precio de dicha opción a través del modelo *Black-Scholes*, usando para ello los modelos de simulación que hemos desarrollado. El precio de la opción o prima será calculado mediante el descuento del valor esperado de los payoffs de las opciones para cada realización.

4. **Creación de un Dashboard:** Por último, crearemos una herramienta gráfica, un dashboard programado con Python, donde podremos monitorizar todos los modelos que hemos llevado a cabo. Se tratará de un dashboard interactivo donde podremos variar los parámetros de entrada para considerar escenarios diferentes a los anteriormente calculados.

Respecto a los datos empleados, hemos elegido el activo MSFT (la acción de Microsoft). Analizaremos 1000 días de datos históricos que obtendremos de la librería de Python *yfinance*. Para la realización del caso práctico, se ha usado de apoyo la siguiente referencia [4]. Los códigos para todos los cálculos se pueden encontrar en el anexo.

6.1. Predicción y Simulación

Comenzaremos descargando los datos de la cotización de MSFT a través de la librería yfinance. Para ello, usaremos los datos de cierre diarios de los últimos 1000 días. A continuación se muestra una gráfica de la cotización:

Figura 11: Cotización de MSFT

A continuación, calcularemos los retornos logarítmicos usando (33). La media y la varianza muestrales de este conjunto de datos (38) son las siguientes:

$$\bar{u} = 0.00047, \quad s^2 = 0.00032$$

Si representamos la distribución de los log-retornos, podemos observar como se distribuyen de forma similar a la distribución normal:

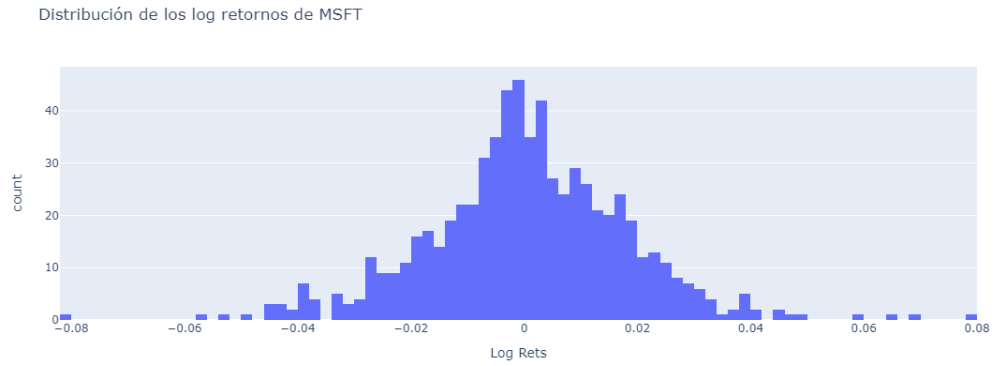

Figura 12: Distribución de los log-retornos de MSFT

El siguiente paso es estimar los parámetros del modelo a través del MME, mediante las expresiones (40) y (43). Respecto a la tolerancia ϵ, probaremos con una serie de valores que serán múltiplos de la desviación estándar muestral s. Posteriormente, y usando el MMV, intentaremos obtener un ϵ óptimo. Hemos tomado $\Delta t = \frac{1}{252}$ ya que MSFT cotiza 252 días al año. Tras realizar los cálculos para diferentes tolerancias, obtenemos los siguientes resultados:

ϵ	Veces s	μ	μ_J	σ	σ_J	λ
0.0267	1.5	0.190	-0.002	0.196	0.037	30.401
0.0356	2.0	0.225	-0.006	0.226	0.045	14.651
0.0445	2.5	0.103	0.009	0.254	0.057	5.128
0.0534	3.0	0.103	0.023	0.265	0.069	2.198
0.0623	3.5	0.106	0.033	0.269	0.074	1.465
0.0713	4.0	0.156	-0.001	0.275	0.111	0.733

Cuadro 1: Parámetros estimados con MME para diferentes valores de ϵ

De la tabla se puede deducir cómo a mayor tolerancia, menor número de saltos al año. El siguiente paso será estimar los parámetros con mayor precisión a través del MMV. Para ello, necesitaremos la función densidad de probabilidad del proceso de salto-difusión (46). Si simulamos dicha distribución para diferentes valores de ϵ, obtenemos distribuciones normales, con una misma media, pero con una desviación estándar variable que disminuye conforme aumentamos la tolerancia.

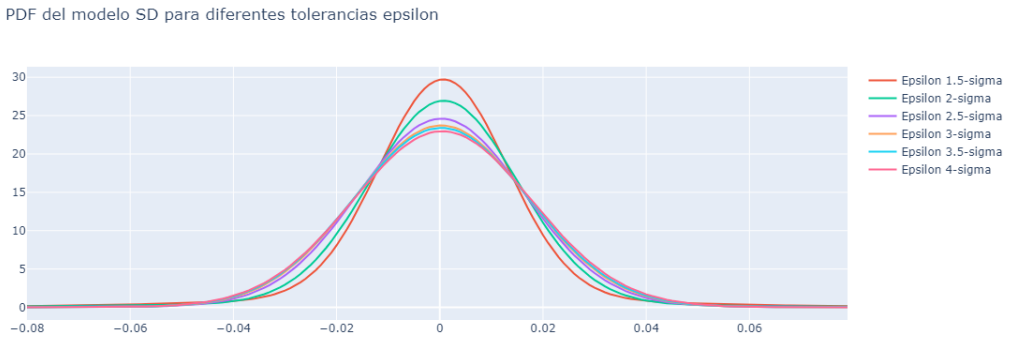

Figura 13: PDF del proceso SD para diferentes tolerancias ϵ

Tomando el logaritmo de las distribuciones podemos extraer información relevante acerca de estas PDFs. Como se puede observar en la siguiente imagen, las colas de la distribución varían bastante dependiendo de la tolerancia que elijamos. Para una tolerancia ϵ más elevada, las colas son menos pesadas ya que hay menos saltos o eventos extremos. En el límite cuando $\epsilon \to \infty$ no hay saltos y la PDF es la de una distribución normal. Para el caso de tolerancias bajas, hay muchos más saltos y las colas se hacen más pesadas, ya que los eventos extremos se acumulan en los laterales de la distribución.

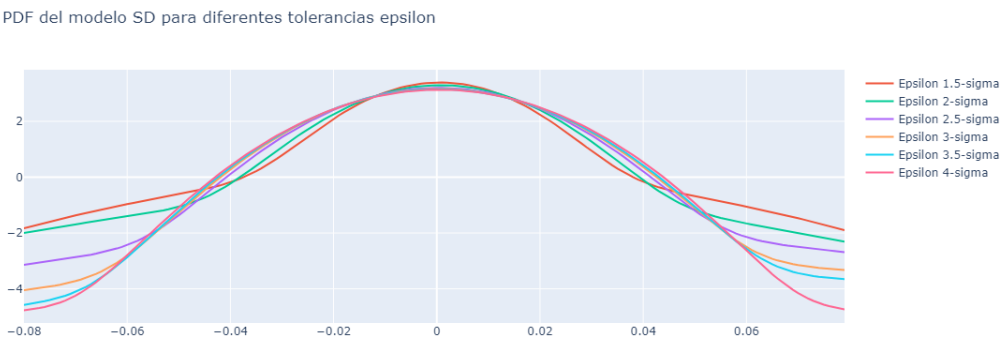

Figura 14: Logaritmo de la PDF del proceso SD para diferentes tolerancias

Calculemos ahora la función de verosimilitud, para ello, tomaremos su logaritmo y cambiaremos el signo de la función (45). A continuación, minimizaremos la función usando la librería de Python *scipy.optimize* y en concreto, mediante la función *minimize*. Definiremos unas restricciones, que en nuestro caso serán que σ, σ_J y λ solo pueden tomar valores positivos y distintos de cero. Además, usaremos como estimaciones iniciales, θ_0, las calculadas en la tabla (para diferentes tolerancias) y veremos a qué conclusiones llegamos:

ϵ	Veces s	μ	μ_J	σ	σ_J	λ
0.0267	1.5	0.191	-0.001	0.240	0.027	30.407
0.0356	2.0	0.136	0.001	0.252	0.033	14.669
0.0445	2.5	0.167	0.005	0.262	0.042	5.534
0.0534	3.0	-0.015	0.011	0.268	0.048	6.144
0.0623	3.5	0.106	0.033	0.269	0.074	1.465
0.0713	4.0	0.156	-0.001	0.275	0.111	0.733

Cuadro 2: Parámetros estimados con MMV para diferentes valores de ϵ

En general, los parámetros estimados con el MMV no difieren en exceso de los calculados inicialmente con el MME. En algunos casos son hasta los mismos valores. Esto nos indica que las estimaciones iniciales eran ya bastante precisas en la mayoría de casos. Idealmente, buscamos que los parámetros del modelo sean iguales para cada valor de ϵ, algo que no ocurre en nuestro caso (salvo en el parámetro σ), luego podemos afirmar que el modelo es dependiente de la tolerancia que elijamos.

Para comprobar cómo de válidos son estos parámetros, calcularemos el MAPE y el RSME para cada tolerancia ϵ y elegiremos el conjunto de parámetros que mejor se ajuste a los datos.

ϵ	Veces s	MAPE (%)	RSME	μ	μ_J	σ	σ_J	λ
0.0267	1.5	9.345	0.532	0.191	-0.001	0.240	0.027	30.407
0.0356	2.0	8.207	0.466	0.136	0.001	0.252	0.033	14.669
0.0445	2.5	8.484	0.482	0.167	0.005	0.262	0.042	5.534
0.0534	3.0	4.860	0.231	-0.015	0.011	0.268	0.048	6.144
0.0623	3.5	7.071	0.400	0.106	0.033	0.269	0.074	1.465
0.0713	4.0	8.078	0.458	0.156	-0.001	0.275	0.111	0.733

Cuadro 3: Parámetros estimados y métricas de error para diferentes valores de ϵ

Tras analizar los datos, tomaremos una tolerancia $\epsilon = 0.0534$ por ser la que menor MAPE posee (menor de nuestro 5 % objetivo) y menor RSME. Los parámetros del modelo serán por tanto, lo de la fila correspondiente a esa tolerancia. Una vez calculados los parámetros óptimos, ya estamos en disposición de realizar simulaciones del precio:

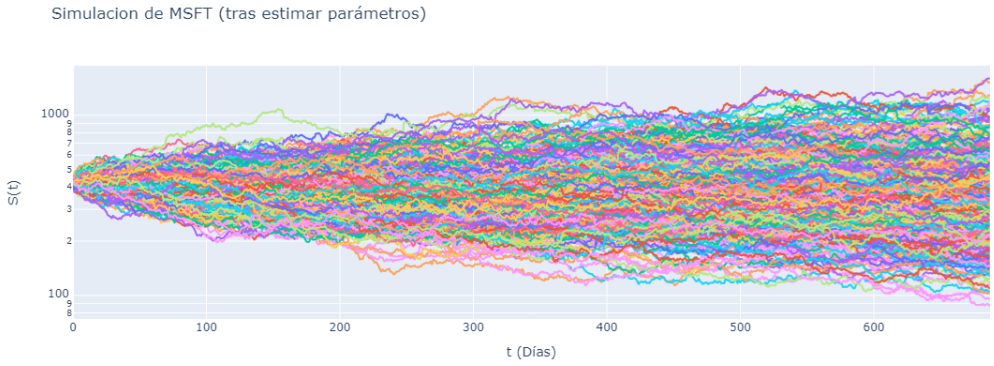

Figura 15: Simulación de MSFT con los parámetros óptimos

En este caso hemos optado por una simulación de 365 días y 500 realizaciones. Hemos tomado los siguientes parámetros de la tabla: $\epsilon = 0.0534$, $\mu = -0.015$, $\mu_J = 0.011$, $\sigma = 0.268$, $\sigma_J = 0.048$ y $\lambda = 6.144$.

Por último, compararemos la PDF estimada con nuestro modelo con la empírica obtenida a partir de los datos de la cotización (los log-retornos en concreto). Podemos observar en la siguiente imagen cómo son similares ambas PDFs:

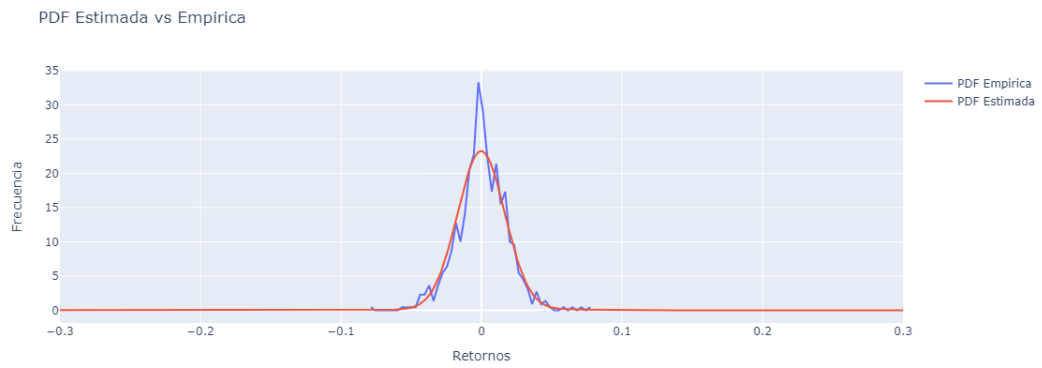

Figura 16: Comparativa de las PDFs empírica vs estimada

6.2. Riesgos: Value at Risk (VaR)

Una vez tenemos el modelo listo y calibrado, podemos crear una métrica de riesgos a partir de el. Supongamos que el banco americano quiere crear una cartera con nuestro activo, en el que invertirá 1 millón de dólares. El banco nos ha pedido estimar las posibles pérdidas a un año vista mediante un modelo *Value at Risk* o VaR con un intervalo de confianza del 95 %. Para ello, seguiremos los siguientes pasos:

1. Simulación del activo para $T = 365$ días, con un número elevado de realizaciones (por ejemplo 100.000).

2. Cálculo de los retornos en el instante final, a partir de las diferentes cotizaciones en ese momento, S_T. En concreto:

$$R_T = \frac{S_T - S_0}{S_0}$$

3. Cálculo del percentil correspondiente al nivel de confianza deseado, denominado α (en nuestro caso $\alpha = 5\,\%$), a partir de la distribución de los retornos R_T. Dicho valor del percentil es lo que se denomina *Value at Risk* o VaR.

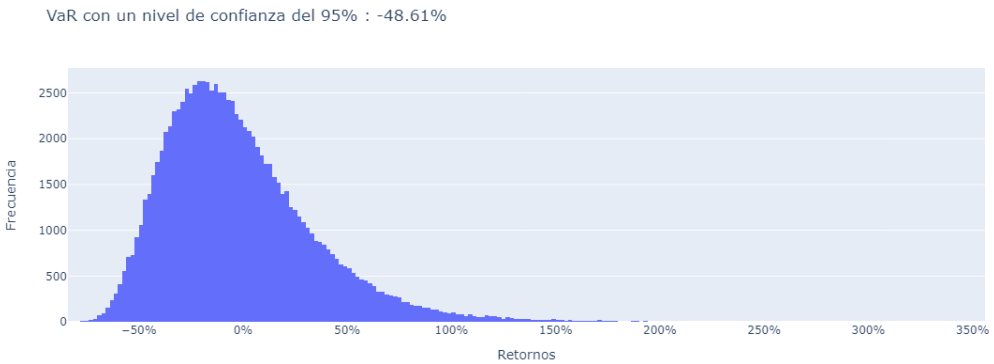

Figura 17: VaR para un intérvalo de confianza del 95 %

Siguiendo estos pasos y usando como simulación la que hemos empleado en el apartado anterior, obtenemos que el VaR es del -48.61 %, que para nuestra cartera de un millón de dólares supone una pérdida potencial al final del periodo de 486.100 dólares.

Una medida de riesgo un tanto más sofisticada es el CVaR o *Conditional Value at Risk*, que considera la magnitud de las pérdidas esperadas que esceden el umbral del VaR. Matemáticamente, se puede calcular como la esperanza de los retornos que superan el nivel VaR. Además, el CVaR es menos sensible al sesgo de las distribuciones de pérdidas que el VaR, lo que lo hace más adecuado para modelar escenarios extremos y situaciones en las que las distribuciones no son simétricas. En nuestro caso, si calculamos el CVaR obtenemos una magnitud de pérdidas esperada del 55.17 %, lo que se traduce en unas pérdidas de 551.700 dólares.

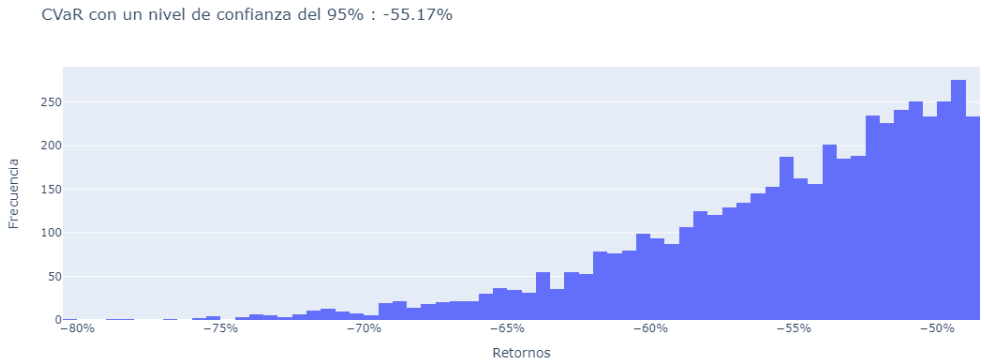

Figura 18: CVaR para un intérvalo de confianza del 95 %

En la figura se muestra la cola de la distribución de los retornos una vez superado el nivel del VaR, que como hemos calculado, es del -48.61 %.

6.3. Valoración: Modelo Black-Scholes

En la presente sección introduciremos otra de las aplicaciones de los procesos estocásticos en las finanzas: la valoración de derivados. En concreto, nos centraremos en las opciones, que se pueden definir como el derecho (pero no la obligación) de comprar o vender un activo con cierto riesgo a un precio fijo preestablecido dentro de un período especificado [8]. Pongamos que nuestro banco quiere ofrecer dicho instrumento financiero a uno de sus clientes, es decir, quiere vender una opción *call* o *put* a cambio de una prima. Desde la perspectiva del cliente, la función que define su beneficio o pérdida al comprar dicho derivado financiero al banco es la siguiente (para el caso de una *call*):

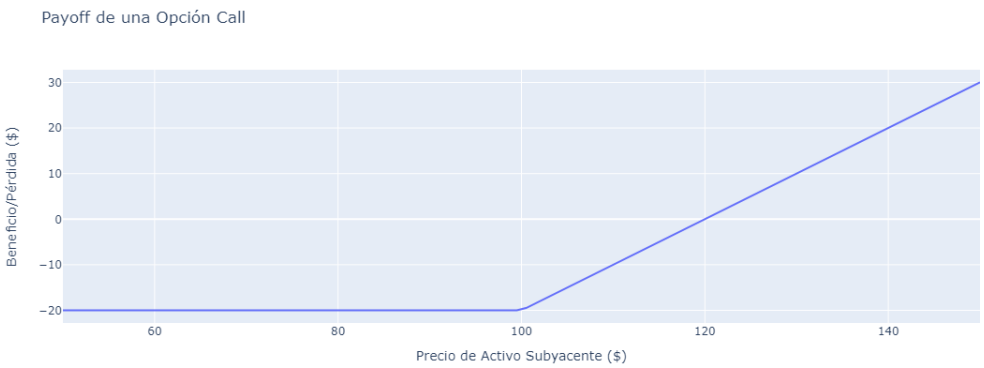

Figura 19: Función de *payoff* de la compra de una *call*

Esta gráfica corresponde al momento de vencimiento T de la opción *call* (si fuese una *put* tendríamos el eje horizontal invertido). En nuestro caso trabajaremos siempre con opciones europeas (que solo se pueden ejercer al vencimiento). La expresión analítica es la siguiente:

$$V(S_T, T) = \text{máx}(S_T - K, 0) - V(S_0, 0) \tag{49}$$

Donde K es el *strike*, o precio de ejercicio, y $V(S, t)$ el precio de la opción. Una aspecto muy relevante al que vamos a dedicar la mayor parte de esta sección es calcular la prima o el precio de venta de la opción $V(S_0, 0)$. Esto es de vital importancia ya que el banco debe de obtener una compensación por el riesgo que asume estando en la posición corta del contrato (que para el caso de una *call* puede implicar pérdidas ilimitadas).

El precio de una opción Europea $V(S, t)$ se puede calcular resolviendo la famosísima ecuación de *Black-Scholes*:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \tag{50}$$

Dicha ecuación se sustenta sobre una serie de hipótesis, conocidas como el modelo de mercado *Black-Scholes*, y son las siguientes [8]:

1. No hay oportunidades de arbitraje.
2. El mercado es libre de fricciones. Esto significa que no hay costes de transacción, los tipos de interés para pedir prestado y prestar dinero son iguales, todas las partes tienen acceso inmediato a cualquier información, y todos los valores y créditos están disponibles en cualquier momento y en cualquier tamaño. En consecuencia, todas las variables son perfectamente divisibles, es decir, pueden tomar cualquier número real. Además, las operaciones individuales no influirán en el precio.
3. El precio del activo sigue un Movimiento Browniano Geométrico.
4. r y σ, el tipo de interés de mercado y la volatilidad implícita (respectivamente), son constantes para $0 \leq t \leq T$. No se pagan dividendos en ese período de tiempo. La opción es europea.

Para este caso práctico, vamos a considerar que el precio del activo sigue un proceso de Salto-Difusión, en lugar de un Movimiento Browniano Geométrico, lo cual es más realista. Una forma sencilla de calcular la prima de la opción $V(S_0, 0)$ es la simulación de múltiples trayectorias del activo subyacente, como hemos hecho en la primera parte del caso. Los pasos a seguir son los siguientes:

1. Parámetros de la opción: Definimos el precio de ejercicio K, el tiempo hasta el vencimiento T, la tasa libre de riesgo r y los parámetros estimados del proceso de Salto-Difusión.
2. Simulación: Realizamos un número suficiente de simulaciones (por ejemplo 100.000) de precios futuros del activo subyacente S_t.
3. Payoff: Calculamos el *payoff* $V(S_T, T)$ de la opción *call* para cada simulación.
4. Descuento al valor presente: Calculamos el valor presente esperado (la media aritmética) de los *payoffs* $V(S_T, T)$ para obtener el precio de la opción, es decir, la prima $V(S_0, 0)$.

A modo de resumen, podemos calcular la prima a través de la siguiente ecuación:

$$V(S_0, 0) = e^{-rT}\mathbb{E}_{\mathbb{Q}}[V(S_T, T)] \tag{51}$$

Donde $\mathbb{E}_{\mathbb{Q}}$ representa la esperanza bajo la medida de probabilidad riesgo-neutral (para más información sobre este tema consultar [10] o [9]). Calculemos ahora varias primas para diferentes vencimientos y *strikes* (para una opción *call* en MSFT). Tomaremos $r = 5.5\,\%$, el tipo de interés de la Reserva Federal (en Mayo de 2024) y $S_0 = 430.16\$$. Si consideramos un *strike* fijo, por ejemplo $K = 500\$$ podemos observar lo siguiente:

Strike ($)	Vencimiento (días)	Valor de la prima ($)
500	365	34.882
500	180	18.820
500	90	9.074
500	30	1.719

Cuadro 4: Tabla de valores de prima para diferentes vencimientos

El valor de la opción decae conforme se acerca el vencimiento. Esto tiene sentido ya que a mayor vencimiento, mayor probabilidad de que se produzca un movimiento en el activo subyacente que haga que la opción se acabe ejerciendo por el comprador al superar el *strike*. Si consideramos diferentes *strikes*:

Strike ($)	Vencimiento (días)	Valor de la prima ($)
431	365	57.016
500	365	34.421
600	365	16.234
700	365	7.436

Cuadro 5: Tabla de valores de prima para diferentes *strikes*

En este caso se observa que si el *strike* se aleja mucho del precio actual de la acción, la prima será baja ya que es improbable que ese suceso ocurra.

7 Creación de un Dashboard

Para finalizar el trabajo, elaboraremos un dashboard con Python donde implementaremos todos los modelos que hemos ido construyendo, permitiendo variar los parámetros de entrada para poder hacer un análisis más completo. El dashboard incluirá los siguientes puntos:

1. **Simulación:** Se incluirá una gráfica que muestre trayectorias de un activo modelado con un proceso de salto-difusión. El activo, los días de simulación y el número de trayectorias serán parámetros de entrada del dashboard.

2. **Riesgos:** Incluiremos un modelo VaR para estimar las posibles pérdidas en un activo dado un nivel de confianza. Dicho nivel de confianza, así como el activo, el número de realizaciones, y el periodo temporal de estudio serán parámetros de entrada.

3. **Valoración:** El dashboard tendrá un gráfico interactivo donde se podrán visualizar varios precios de opciones *call* o *put* sobre un activo a elegir, para varios *strikes*. Permitiremos variar el activo, el tipo de interés de mercado, el tipo de opción y los días hasta el vencimiento.

Incluiremos también un gráfico de la cotización histórica del activo a modelar.

7.1. Librerías y Funciones Empleadas

Usaremos principalmente las librerías *plotly* y *dash*, que nos permitirán elaborar los gráficos y el entorno del dashboard respectivamente. Por simplicidad, emplearemos datos diarios de la librería *yfinance*. Por otro lado, la mayoría de cálculos numéricos se llevarán a cabo usando *numpy* y *scipy*. Abajo se adjuntan todas las librerías y funciones que emplearemos en este proyecto:

```
import dash
import plotly.express as px
from dash import dcc, html
from dash.dependencies import Input, Output, State
import plotly.graph_objs as go
import numpy as np
import yfinance as yf
from datetime import datetime, timedelta
from scipy.stats import norm
import math
from scipy.optimize import minimize
import warnings
warnings.filterwarnings("ignore")
```

7.2. Funciones Propias

Por otro lado, para la realización del dashboard, también emplearemos funciones propias para realizar todas las simulaciones. En primer lugar, usaremos estas dos funciones para obtener los datos diarios de la acción deseada para un número de días dado:

```
# Obten la fecha de hace n dias
def date_n_days_ago(n):
    today = datetime.now()
    delta = timedelta(days=n)
    result = today - delta
    formatted_result = result.strftime('%Y-%m-%d')

    return formatted_result

# Extrae datos de yfinance
def get_stock_data(stock_symbol, n):
    stock = yf.download(stock_symbol, start = date_n_days_ago(n
        ), progress = False)

    return stock
```

Respecto a la simulación del proceso de salto-difusión, emplearemos la función que definimos en (6), que queda de la siguiente forma:

```
# Proceso de salto-difusion
def JumpDiff_Merton(S0, mu, sigma, lamb, mu_J, sigma_J, steps,
    paths, Delta_t):

    # Correccion del drift para mantener neutralidad al riesgo
    r_J = lamb*(np.exp(mu_J + 0.5*sigma_J**2)-1)

    Z1 = np.random.standard_normal((steps+1, paths))
    Z2 = np.random.standard_normal((steps+1, paths))
    Y = np.random.poisson(lamb*Delta_t, (steps+1, paths))

    cum_poi = np.multiply(Y, mu_J + sigma_J*Z2).cumsum(axis =
        0)
    gbm = np.cumsum(((mu -  sigma**2/2 -r_J)*Delta_t+ sigma*np.
        sqrt(Delta_t) * Z1), axis=0)

    return np.exp(cum_poi + gbm)*S0
```

Para la optimización de parámetros, necesitaremos dos funciones, una que inicialice la función densidad de probabilidad del proceso salto-difusión y otra que calcule la función de verosimilitud (que posteriormente minimizaremos):

```
# PDF del proceso salto-difusion vectorizada
def jump_diffusion_pdf_vector(x, Delta_t, mu, sigma, lambd,
    mu_J, sigma_J):
    k = np.arange(100)
    t = np.array([math.factorial(f) for f in k])
```

```
    pk = np.exp(-lambd * Delta_t) * ((lambd * Delta_t) ** k) / t
    mu_phi = (mu - (sigma ** 2) / 2) * Delta_t + mu_J * k
    sigma_phi = np.sqrt((sigma ** 2) * Delta_t + (sigma_J ** 2) * k)

    pdf_contributions = pk * norm.pdf(x, loc=mu_phi, scale=sigma_phi)
    pdf = sum(pdf_contributions)

    return np.log(pdf)

# Funcion de verosimilitud
def log_likelihood(theta, rets, Delta_t):
    mu, sigma, lambd, mu_J, sigma_J = theta
    lnL = 0

    for x in rets:
        pdf = jump_diffusion_pdf_vector(x, Delta_t, mu, sigma, lambd, mu_J, sigma_J)
        lnL += pdf

    return -lnL
```

Para el caso del cálculo del VaR, emplearemos esta simple función:

```
# Calcula el VaR para un intervalo de confianza dado
def calculate_var(returns, alpha):
    var = np.percentile(returns, alpha * 100)

    return var
```

Y por último, para la valoración de opciones vía montecarlo, usaremos el siguiente código (que nos calcula varios precios de opciones para diferentes *strikes*, descontando la esperanza de los *payoffs* simulados):

```
# Funcion para calcular el precio de la opcion via montecarlo (T en dias)
def mc_option_valuation(S0, strikes, T, r, mu, sigma, lambd, mu_J, sigma_J, paths, option_type):
    S_t = JumpDiff_Merton(S0, mu, sigma, lambd, mu_J, sigma_J, steps = T, paths = paths, Delta_t = 1/252)
    S_T = S_t[-1]

    results = {}
    for K in strikes:
        if option_type == 'call':
            payoffs = np.maximum(S_T - K, 0)
        elif option_type == 'put':
            payoffs = np.maximum(K - S_T, 0)
        else:
            raise ValueError("Invalid option type. Use 'call' or 'put'")

        disc_payoffs = np.exp(-r * T/365) * payoffs
```

```
        option_price = np.mean(disc_payoffs)
        results[K] = option_price

    return results
```

7.3. Configuración del Dashboard

En esta sección definiremos la apariencia visual del dashboard, así como sus parámetros de entrada. Estos últimos serán:

1. **Ticker:** Abreviatura que se usa en los mercados de valores para hacer referencia a un activo, en nuestro caso, a una acción.
2. **Días:** Número de días de simulación, este dato de entrada valdrá para todos los gráficos (tanto para el modelo VaR como para la valoración de las opciones).
3. **Trayectorias:** Número de simulaciones del activo elegido, a mayor número de estas, mayor precisión en los modelos pero también mayor tiempo de computación.
4. **Opción:** Tipo de opción, *call* o *put* para el modelo de valoración de opciones.
5. **Tipo de Interés:** Tipo de interés de mercado que usaremos para descontar los *payoffs* esperados en el modelo de valoración.
6. **Nivel de Confianza:** Percentil sobre la distribución de los retornos que nos indicará la pérdida esperada para ese umbral. Se usa en el modelo VaR.

A continuación, se adjunta la sección correspondiente a la inicialización del dashboard y a la configuración de los inputs:

```
# Inicializacion de la app de Dash
app = dash.Dash(__name__)
app.title = 'Dashboard'

# Apariencia del dashboard
app.layout = html.Div([
    html.H1("Dashboard de Simulaci n, Riesgos y Valoraci n de
        Opciones"),

    html.Div([
        html.Label('Ticker: '),
        dcc.Input(id='input-stock', type='text', value='MSFT',
            style={'width': '45px', 'margin-right': '10px'}),

        html.Label('D as: '),
        dcc.Input(id='input-steps', type='number', value=500,
            style={'width': '45px', 'margin-right': '10px'}),

        html.Label('Trayectorias: '),
        dcc.Input(id='input-paths', type='number', value =
            10000, style={'width': '55px', 'margin-right': '10px
            '}),

```

```
        html.Label('Opci n: '),
        dcc.Input(id='input-option_type', type='text', value='
            call', style={'width': '40px', 'margin-right': '10px
            '}),

        html.Label('Tipo de Inter s: '),
        dcc.Input(id='input-r', type='number', value = 0.055,
            step = 0.001, style={'width': '50px', 'margin-right'
            : '10px'}),

        html.Label('Nivel de Confianza: '),
        dcc.Input(id='input-alpha', type='number', value =
            0.95, step = 0.01, style={'width': '50px', 'margin-
            right': '10px'}),

        html.Button('Simulate', id ='simulate-button', n_clicks
            =0),

        ], style={'display': 'inline-block'}),
```

Como ya hemos comentado incluiremos una gráfica de la cotización histórica del activo a estudiar, otra que muestre las simulaciones de dicho activo a través del proceso de salto-difusión, un gráfico que ejemplifique las potenciales pérdidas dado un nivel de confianza (modelo VaR) y por último un gráfico del precio de una opción *call* o *put* sobre el activo elegido, para diferentes vencimientos. El código para inicializar los gráficos y el *callback* para actualizar los *inputs* y *outputs* es el siguiente:

```
        html.Div([
        dcc.Graph(id='stock-graph', style={'width': '50%', '
            display': 'inline-block'}),
        dcc.Graph(id='sim-graph', style={'width': '50%', '
            display': 'inline-block'}),]),

    html.Div([
    dcc.Graph(id='var-histogram', style={'width': '50%', '
        display': 'inline-block'}),
    dcc.Graph(id='option-valuation', style={'width': '50%', '
        display': 'inline-block'}),
    ]),
])

# Callback para actualizar el dashboard segun los inputs
@app.callback(
    [Output('stock-graph', 'figure'),
     Output('sim-graph', 'figure'),
     Output('option-valuation','figure'),
     Output('var-histogram', 'figure')],
    [Input('simulate-button', 'n_clicks')],
    [State('input-stock', 'value'),
     State('input-steps', 'value'),
     State('input-paths', 'value'),
     State('input-option_type', 'value'),
     State('input-r', 'value'),
     State('input-alpha', 'value')])
```

7.4. Cálculo de los Modelos

En esta parte del código nos centraremos en estimar los parámetros del modelo de salto-difusión para poder realizar las simulaciones. Hemos decidido situar la tolerancia ϵ en 3 desviaciones estándar sobre la desviación estándar muestral. El resto de cálculos corresponden al caso práctico que hemos llevado a cabo en la sección anterior:

```
def update_graph(n_clicks, stock_symbol, steps, paths,
    option_type, r, alpha):
     if n_clicks >= 0:

         # Definimos algunos parametros iniciales
         Delta_t = 1/252
         alpha = 1 - alpha

         # Descarga de datos
         S = get_stock_data(stock_symbol, steps)
         S['Log Rets'] = np.log(S['Close']/S['Close'].shift(1))
         S0 = S['Close'].iloc[-1]

         # Separamos en retornos con salto y sin salto
         eps = 3*np.std(S['Log Rets'])
         R_J = S[np.abs(S['Log Rets']) >= eps]['Log Rets']
         R_D = S[np.abs(S['Log Rets']) < eps]['Log Rets']

         # Estimamos los parametros mu y sigma a partir de los
            retornos sin salto
         u_D = np.mean(R_D); s_D = np.var(R_D, ddof = 1)
         mu_0 = (u_D + 0.5*s_D)/Delta_t
         sigma_0 = np.sqrt(s_D/Delta_t)

         # Estimamos lambda (en saltos/a o)
         lambd_0 = len(R_J)/(len(S)*Delta_t)

         # Estimamos mu_J y sigma_J a partir de los retornos con
            salto
         u_J = np.mean(R_J); s_J = np.var(R_J, ddof = 1)
         mu_J_0 = u_J - u_D
         sigma_J_0 = np.sqrt(s_J - s_D)

         # Parametros inciales y restricciones
         theta_0 = [mu_0, sigma_0, lambd_0, mu_J_0, sigma_J_0]
         bounds = [(-1.5, 1.5), (0.01, 1), (0.5, 50), (-0.5,
            0.5), (0.01, 0.5)]

         # Optimizacion
         res = minimize(log_likelihood, theta_0, args=(S['Log
            Rets'], Delta_t), method='L-BFGS-B', bounds=bounds,
            options={"maxiter":20})
         mu, sigma, lambd, mu_J, sigma_J = res.x
```

7.5. Gráficas de los Modelos

Por último, una vez tenemos los *inputs* de los modelos y las simulaciones calculadas pasaremos a la parte más visual del proyecto, las gráficas con los modelos. A continuación se muestran los resultados:

Figura 20: Gráfico 1: Cotización de la acción

En primer lugar, la gráfica de la cotización del activo, definida por la variable **Ticker**, mostrando un rango de datos históricos definido por la variable **Días**.

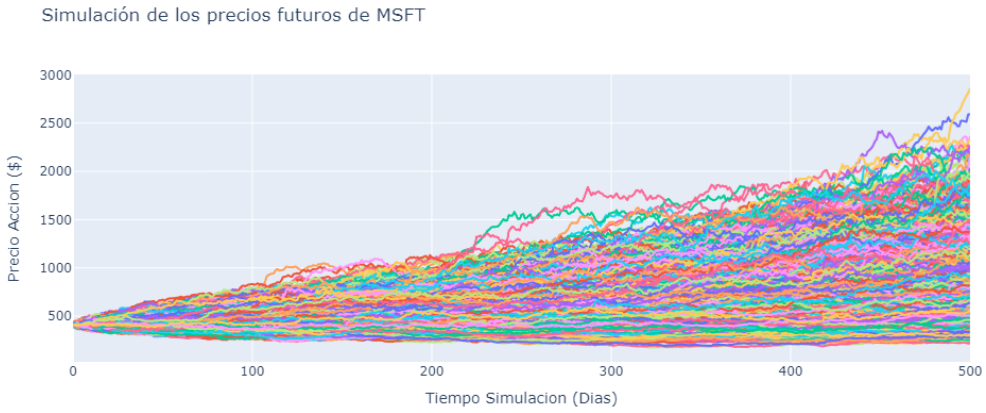

Figura 21: Gráfico 2: Simulaciones

En segundo lugar, la gráfica de las simulaciones del proceso Salto-Difusión, definida por las variables **Trayectorias**, **Días** y **Ticker**.

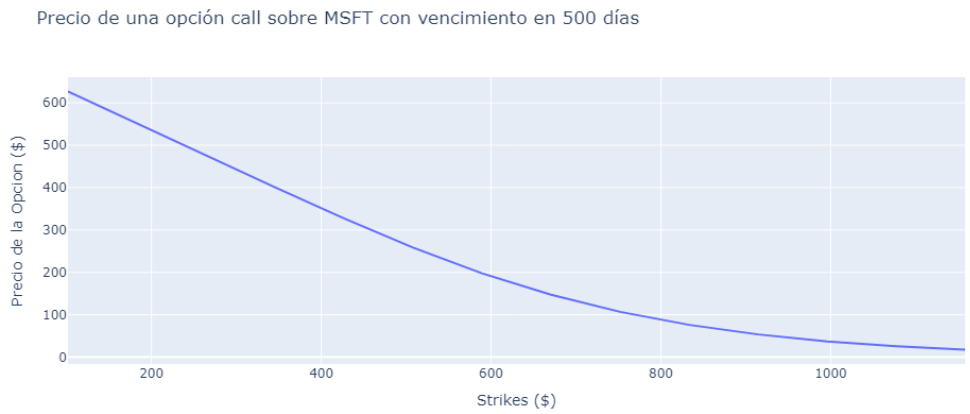

Figura 22: Gráfico 3: Valoración de opciones

En tercer lugar, la gráfica del modelo de valoración de opciones, con parámetros **Ticker**, **Días**, **Trayectorias**, **Opción** y **Tipo de Interés**.

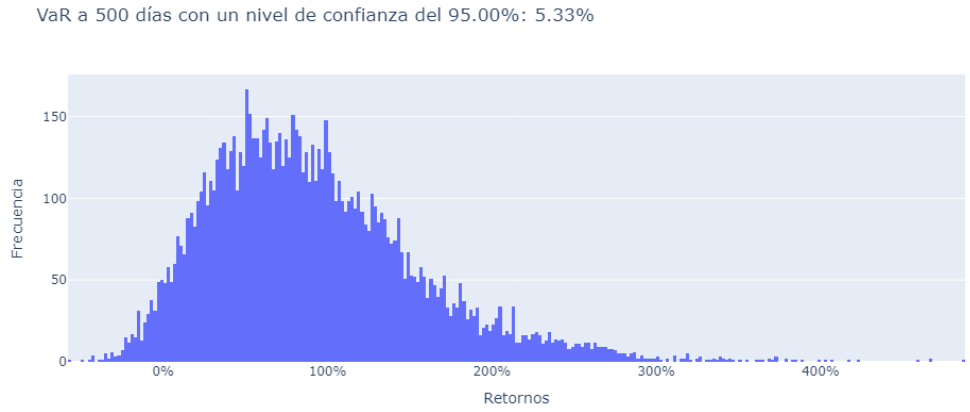

Figura 23: Gráfico 4: Modelo VaR

Por último, tenemos el gráfico del modelo de riesgos VaR, asociado a los *inputs* **Ticker**, **Días**, **Trayectorias** y **Nivel de Confianza**.

A continuación, el código empleado para generar las gráficas:

```
1         # Grafico de la accion
2         stock_fig = go.Figure()
3         stock_fig.add_trace(go.Scatter(x = S.index, y = S['
              Close'], mode='lines', name='Stock Price'))
4         stock_fig.update_layout(title=f"Cotizaci n de {
              stock_symbol}", xaxis_title="Fecha", yaxis_title="
              Precio Accion ($)")
5
6         # Grafico simulaciones
7         S_t = JumpDiff_Merton(S0, mu, sigma, lambd, mu_J,
```

```
            sigma_J, steps = steps, paths = paths, Delta_t =
                Delta_t)
8           sim_fig = px.line(S_t, title = f'Simulaci n de los
                precios futuros de {stock_symbol}')
9           sim_fig.update_layout(xaxis_title='Tiempo Simulacion (
                Dias)', yaxis_title='Precio Accion ($)', showlegend=
                False)
10
11          # Grafico del VaR
12          retornos = (S_t[-1,:] - S0)/S0
13          var = calculate_var(retornos, alpha)
14          var_fig = go.Figure()
15          var_fig.add_trace(go.Histogram(x = retornos, nbinsx =
                500, showlegend = False))
16          var_fig.update_layout(title=f'VaR a {steps} d as con
                un nivel de confianza del {1-alpha:.2%}: {var:.2%}',
                xaxis=dict(title = "Retornos", tickformat = ".0%",),
                 yaxis_title = "Frecuencia")
17
18          # Grafico valoracion opciones
19          strikes = S0*np.arange(0.25, 3, 0.20)
20          V = mc_option_valuation(S0, strikes, steps, r, mu,
                sigma, lambd, mu_J, sigma_J, paths, option_type)
21          option_fig = px.line(x = strikes, y = V, title = f'
                Precio de una opci n {option_type} sobre {
                stock_symbol} con vencimiento en {steps} d as')
22          option_fig.update_layout(xaxis_title = 'Strikes ($)',
                yaxis_title = 'Precio de la Opcion ($)', showlegend
                = False)
23
24      return stock_fig, sim_fig, var_fig, option_fig
25
26  # Mantenemos el dashboard en constante actualizacion (al
        realizar cambios)
27  if __name__ == '__main__':
28      app.run_server(debug=True)
```

7.6. Resultado Final y Conclusión

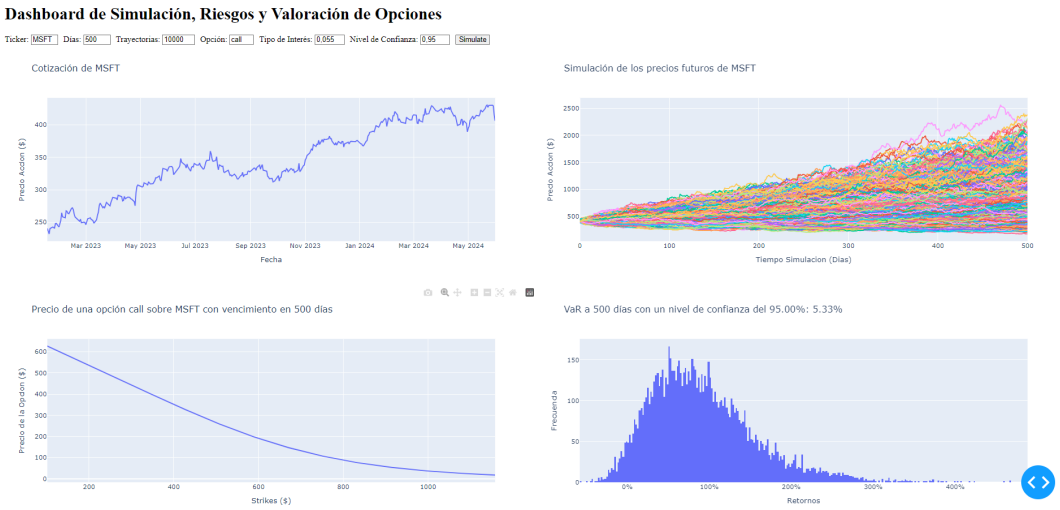

Figura 24: Resultado final del dashboard

El resultado final de este trabajo es el dashboard que se muestra arriba. Hemos conseguido implementar tres modelos financieros, cada uno focalizado en un área concreta de las finanzas matemáticas (simulación, valoración y riesgos) usando herramientas del cálculo estocástico.

8 Anexo: Código del Caso Práctico

```
#-----------------------------------------------------
# Autor: Pablo Marchesi Selma
# Universidad Politecnica de Valencia
# Mayo 2024
# pablomarchesiselma@gmail.com
#-----------------------------------------------------

import numpy as np
import yfinance as yf
import plotly.express as px
import plotly.graph_objs as go
import pandas as pd
import math
from scipy.stats import norm
from scipy.optimize import minimize
import warnings
from datetime import datetime, timedelta
warnings.filterwarnings("ignore")

# %%
# Obten la fecha de hace n d as
def date_n_days_ago(n):
    today = datetime.now()
    delta = timedelta(days=n)
    result = today - delta
    formatted_result = result.strftime('%Y-%m-%d')
    return formatted_result

# %% [markdown]
# ### 1. Prediccion y Simulacion:

# %%
# Descargamos los datos de yfinance y calculamos los retornos
    logaritmicos
ticker = 'MSFT'
N = 1000
S = yf.download(ticker, start = date_n_days_ago(N), interval =
    '1d', progress = False )
S['Log Rets'] = np.log(S['Close']/S['Close'].shift(1)).dropna()
S = S.dropna()

# Plot de la cotizacion
px.line(S, y = 'Close', title = f'Cotizaci n de {ticker}',
    log_y= True)

# %%
# Calculamos la media y la varianza muestral de los log-
    retornos
u = np.mean(S['Log Rets'])
s_2 = np.var(S['Log Rets'], ddof = 1)
print(f'Media: {round(u,5)}, Varianza: {round(s_2,5)}')

# Plot de la distribucion de los retornos
```

```
50 px.histogram(S, x = S['Log Rets'], nbins= 100, title = f'
       Distribuci n de los log retornos de {ticker}')
51
52 # %%
53 # Estimamos los parametros, para ello:
54 # 1) Establecemos una tolerancia epsilon para los saltos
55 # 2) Dividimos el conjunto de datos
56 # 3) Iteramos para diferentes valores de epsilon
57
58 mu = []; mu_J = []; sigma = []; sigma_J = []; lambd = [];
       epsilon = []
59
60 Delta_t = 1/252
61 num_std = [1.5, 2, 2.5, 3, 3.5, 4] # Numero de veces la
       desviacion estandar de los log retornos
62
63 for n in num_std:
64
65     eps = n*np.std(S['Log Rets'])
66     epsilon.append(round(eps, 4))
67     R_J = S[np.abs(S['Log Rets']) >= eps]['Log Rets']
68     R_D = S[np.abs(S['Log Rets']) < eps]['Log Rets']
69
70     # Estimamos los par metros mu y sigma a partir de los
          retornos sin salto
71     u_D = np.mean(R_D); s_D = np.var(R_D, ddof = 1)
72
73     mu.append((u_D + 0.5*s_D)/Delta_t)
74     sigma.append(np.sqrt(s_D/Delta_t))
75
76     # Estimamos lambda (en saltos/a o)
77     lambd.append(len(R_J)/(len(S)*Delta_t))
78
79     # Estimamos mu_J y sigma_J a partir de los retornos con
          salto
80     u_J = np.mean(R_J); s_J = np.var(R_J, ddof = 1)
81
82     mu_J.append(u_J - u_D)
83     sigma_J.append(np.sqrt(s_J - s_D))
84
85 params = pd.DataFrame({'Evento n-sigma':num_std, 'mu':mu, 'mu_J
       ':mu_J, 'sigma':sigma, 'sigma_J':sigma_J, 'lambda':lambd},
       index = epsilon)\
86             .rename_axis('epsilon').round(3)
87 params
88
89 # %%
90 # Definimos el proceso de salto-difusion
91
92 def JumpDiff_Merton(S0, mu, sigma, lamb, mu_J, sigma_J, steps,
       paths, Delta_t):
93
94     # Correccion del drift para mantener neutralidad al riesgo
95     r_J = lamb*(np.exp(mu_J + 0.5*sigma_J**2)-1)
96
97     Z1 = np.random.standard_normal((steps+1, paths))
```

```
    Z2 = np.random.standard_normal((steps+1, paths))
    Y = np.random.poisson(lamb*Delta_t, (steps+1, paths))

    cum_poi = np.multiply(Y, mu_J + sigma_J*Z2).cumsum(axis = 0)
    gbm = np.cumsum(((mu -  sigma**2/2 -r_J)*Delta_t+ sigma*np.sqrt(Delta_t) * Z1), axis=0)

    return np.exp(cum_poi + gbm)*S0

# %%
# Funcion de densidad de probabilidad del proceso de salto-difusi n

def jump_diffusion_pdf(x, Delta_t, mu, sigma, lambd, mu_J, sigma_J):
    pdf = np.zeros_like(x)
    nsaltos = 100

    # Sumamos un numero suficientemente grande de saltos
    for k in range(nsaltos):
        pk = np.exp(-lambd * Delta_t) * ((lambd * Delta_t) ** k) / math.factorial(k)
        mu_phi = (mu - (sigma ** 2) / 2) * Delta_t + mu_J * k
        sigma_phi = np.sqrt((sigma ** 2) * Delta_t + (sigma_J ** 2) * k)

        # Actualizamos pdf para cada salto
        pdf += pk*norm.pdf(x, loc=mu_phi, scale=sigma_phi)

    return pdf

# %%
# Comparativa de la PDF del proceso de SD para diferetes tolerancias (epsilon)
x = np.arange(-0.08, 0.08, 0.001)
fig = px.line(title = 'PDF del modelo SD para diferentes tolerancias epsilon')

for n in range(len(num_std)):
    pdf = jump_diffusion_pdf(x, Delta_t, mu[n], sigma[n], lambd[n], mu_J[n], sigma_J[n])
    fig.add_scatter(x=x, y = np.log(pdf), mode='lines', name=f'Epsilon {num_std[n]}-sigma')

fig.show()

# %%
# PDF vectorizada mucho m s r pida
def jump_diffusion_pdf_vector(x, Delta_t, mu, sigma, lambd, mu_J, sigma_J):
    k = np.arange(100)
    t = np.array([math.factorial(f) for f in k])
    pk = np.exp(-lambd * Delta_t) * ((lambd * Delta_t) ** k) / t
    mu_phi = (mu - (sigma ** 2) / 2) * Delta_t + mu_J * k
```

```
142     sigma_phi = np.sqrt((sigma ** 2) * Delta_t + (sigma_J ** 2)
            * k)
143
144     pdf_contributions = pk * norm.pdf(x, loc=mu_phi, scale=
            sigma_phi)
145     pdf = sum(pdf_contributions)
146
147     return np.log(pdf)
148
149 # Definimos la funcion de verosimilitud
150 def log_likelihood(theta, rets, Delta_t):
151     mu, sigma, lambd, mu_J, sigma_J = theta
152     lnL = 0
153
154     for x in rets:
155         pdf = jump_diffusion_pdf_vector(x, Delta_t, mu, sigma,
                lambd, mu_J, sigma_J)
156         lnL += pdf
157
158     return -lnL
159
160 # %%
161 # Minimizamos la funcion
162 mu_est = []; sigma_est = []; lambda_est = []; mu_J_est = [];
        sigma_J_est = []
163
164 for n in range(len(num_std)):
165
166     print(f'Estimaci n  n mero {n+1}')
167
168     # Estimaciones iniciales y restricciones
169     theta_0 = [mu[n], sigma[n], lambd[n], mu_J[n], sigma_J[n]]
170     bounds = [(-1.5, 1.5), (0.01, 1), (0.5, 50), (-0.5, 0.5),
            (0.01, 0.5)]
171
172     # Optimization
173     res = minimize(log_likelihood, theta_0, args = (S['Log Rets
            '], Delta_t), method = 'L-BFGS-B', bounds = bounds,
            options = {"maxiter":20})
174     mu_est.append(res.x[0]); sigma_est.append(res.x[1]);
            lambda_est.append(res.x[2]); mu_J_est.append(res.x[3]);
            sigma_J_est.append(res.x[4])
175
176 # Comparamos las estimaciones iniciales por MME con las
        estimaciones de MMV
177 params_est = pd.DataFrame({'Evento n-sigma': num_std, 'mu (
        inicial)':mu, 'mu (estimado)':mu_est,
178 'sigma (inicial)': sigma, 'sigma (estimado)': sigma_est,
179 'lambda (inicial)':lambd, 'lambda (estimado)':lambda_est,
180 'mu_J (inicial)': mu_J,'mu_J (estimado)':mu_J_est,
181 'sigma_J (inicial)': sigma_J, 'sigma_J (estimado)': sigma_J_est
        ,},
182 index = epsilon).rename_axis('epsilon').round(3)
183 params_est
184
185 # %%
```

```
# Definimos el MAPE y el RSME
def RSME(v_i, v_tilde):
    return np.sqrt(np.mean(v_i - v_tilde)**2)

def MAPE(v_i, v_tilde):
    return np.mean(np.abs(v_i - v_tilde)/v_i)

# %%
# Calculamos el MAPE y el RSME para cada tolerancia
v_i = np.log(S['Close'])
S0 = S['Close'].iloc[-1]; steps = len(S['Close'])-1; paths =
    500
MAPE_list = []; RSME_list = []

for n in range(len(num_std)):
    S_t = np.log(JumpDiff_Merton(S0, mu_est[n], sigma_est[n],
        lambda_est[n], mu_J_est[n], sigma_J_est[n], steps, paths
        , Delta_t))
    v_tilde = np.mean(S_t, axis = 1)
    MAPE_list.append(MAPE(v_i, v_tilde)*100)
    RSME_list.append(RSME(v_i, v_tilde))

params_fit =  pd.DataFrame({'Evento n-sigma': num_std, 'MAPE
    (%)': MAPE_list, 'RSME': RSME_list,
'mu (estimado)':mu_est, 'sigma (estimado)': sigma_est,'lambda (
    estimado)':lambda_est,
'mu_J (estimado)':mu_J_est,'sigma_J (estimado)': sigma_J_est,},
     index = epsilon).rename_axis('epsilon').round(3)

params_fit

# %%
# Simulamos en funcion de las estimaciones optimas
# Elegimos n = 3 por tener el menor MAPE
n = 3
S0 = S['Close'].iloc[-1]; steps = len(S['Close'])-1;

S_t = JumpDiff_Merton(S0, mu_est[n], sigma_est[n], lambda_est[n
    ], mu_J_est[n], sigma_J_est[n], steps, paths, Delta_t)

fig = px.line(S_t, title = f'Simulacion de {ticker} (tras
    estimar par metros)', log_y = True )
fig.update_layout(xaxis_title='t (D as)', yaxis_title='S(t)',
    showlegend=False)
fig.show()

# %%
# Comparamos la PDF obtenida a partir de los datos (emp rica)
    con la estimada con los par metros  ptimos
x = np.linspace(-0.5, 0.5, len(S['Log Rets']));
empirical_pdf, bin_edges = np.histogram(S['Log Rets'], bins=50,
     density=True)
bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

pdf_fig = go.Figure()
```

```
pdf_fig.add_trace(go.Scatter(x=bin_centers, y = empirical_pdf, mode='lines', name='PDF Empirica'))

pdf = jump_diffusion_pdf(x, Delta_t, mu_est[n], sigma_est[n], lambda_est[n], mu_J_est[n], sigma_J_est[n])
pdf_fig.add_trace(go.Scatter(x = x, y = pdf, mode='lines', name = 'PDF Estimada'))

pdf_fig.update_layout(title = 'PDF Estimada vs Empirica', xaxis_title="Retornos", yaxis_title="Frecuencia")
pdf_fig.update_xaxes(range=[-0.3, 0.3])

# %% [markdown]
# ### 2. Riesgos: Modelo Value at Risk (VaR)

# %%
# Creamos un modelo VaR a partir del proceso de SD para estimar las posibles perdidas tras T d as
T = 365; paths = 100000;
S_t = JumpDiff_Merton(S0, mu_est[n], sigma_est[n], lambda_est[n], mu_J_est[n], sigma_J_est[n], steps = T, paths = paths, Delta_t = 1/252)
rets = (S_t[-1,:] - S0)/S0

alpha = 0.05
var = np.percentile(rets, alpha * 100)

var_fig = go.Figure()
var_fig.add_trace(go.Histogram(x = rets, nbinsx = 300, showlegend=False))
var_fig.update_layout(title=f'VaR con un nivel de confianza del {1-alpha:.0%} : {var:.2%}',
                      xaxis=dict(title = "Retornos", tickformat = ".0%",), yaxis_title = "Frecuencia"
                     )

# %%
# C lculo de CVaR
cvar = np.mean(rets[rets <= var])
cvar_fig = go.Figure()
cvar_fig.add_trace(go.Histogram(x = rets[rets <= var], nbinsx = 100, showlegend=False))
cvar_fig.update_layout(title = f'CVaR con un nivel de confianza del {1-alpha:.0%} : {cvar:.2%}',
                      xaxis=dict(title = "Retornos", tickformat = ".0%",), yaxis_title = "Frecuencia"
                     )

# %% [markdown]
# ### 3. Valoraci n de Opciones: Modelo Black-Scholes

# %%
# Grafico Payoff Opcion Call
K = 100
S_min = 50
S_max = 150
```

```
V_0 = 20

S_T = np.linspace(S_min, S_max, 100)
payoff_call = np.maximum(S_T - K, 0) - V_0
fig = px.line(x = S_T, y = payoff_call, title = 'Payoff de una Opci n Call')
fig.update_layout(xaxis_title = 'Precio de Activo Subyacente ($)', yaxis_title = 'Beneficio/P rdida ($)', showlegend = False)

# %%
# Funcion para calcular el precio de la opcion via montecarlo (T en dias)
def mc_option_valuation(S0, strikes, T, r, mu, sigma, lambd, mu_J, sigma_J, paths, option_type='call'):
    S_t = JumpDiff_Merton(S0, mu, sigma, lambd, mu_J, sigma_J, steps = T, paths = paths, Delta_t = 1/252)
    S_T = S_t[-1]

    results = {}
    for K in strikes:
        if option_type == 'call':
            payoffs = np.maximum(S_T - K, 0)
        elif option_type == 'put':
            payoffs = np.maximum(K - S_T, 0)
        else:
            raise ValueError("Invalid option type. Use 'call' or 'put'")

        disc_payoffs = np.exp(-r * T/365) * payoffs
        option_price = np.mean(disc_payoffs)
        results[K] = option_price

    return results

# %%
r = 0.055; option_type = 'call'; T = 365
strikes  = S0*np.arange(0.2, 3, 0.10)
V = mc_option_valuation(S0, strikes, T, r, mu_est[n], sigma_est[n], lambda_est[n], mu_J_est[n], sigma_J_est[n], paths, option_type)
option_fig = px.line(x = strikes, y = V, title = f'Precio de una opci n {option_type} sobre {ticker}')
option_fig.update_layout(xaxis_title = 'Strikes ($)', yaxis_title = 'Precio de la Opcion ($)', showlegend = False)
```

Referencias

[1] Iván Atienza. Charla sobre finanzas quantitativas, 2024. `https://www.youtube.com/watch?v=nJNuos6hf6o&t=2495s`.

[2] Stanley J Farlow. *Partial differential equations for scientists and engineers*. Courier Corporation, 1993.

[3] Geoffrey Grimmett and David Stirzaker. *Probability and random processes*. Oxford university press, 2020.

[4] Yves Hilpisch. *Python for Finance: Analyze big financial data*. O'Reilly Media, Inc., 2014.

[5] Gautam Iyer. Stochastic calculus for finance brief lecture notes. *Carnegie Mellon University*, 2017.

[6] Jean Jacod and Philip Protter. *Probability essentials*. Springer Science & Business Media, 2004.

[7] Rafael J. Villanueva Juan Carlos Cortés, Cristina Santamaría. Unidad 1: Modelización matemática de subyacentes cotizados.

[8] Rüdiger Seydel and Rudiger Seydel. *Tools for computational finance*, volume 3. Springer, 2006.

[9] Steven Shreve. *Stochastic calculus for finance I: the binomial asset pricing model*. Springer Science & Business Media, 2005.

[10] Steven E Shreve et al. *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer, 2004.

[11] Furui Tang. Merton jump-diffusion modeling of stock price data, 2018.

[12] The Pennsylvania State University. Introduction to mathematical statistics: Maximum likelihood estimation, 2024. `https://online.stat.psu.edu/stat415/lesson/1/1.2`.