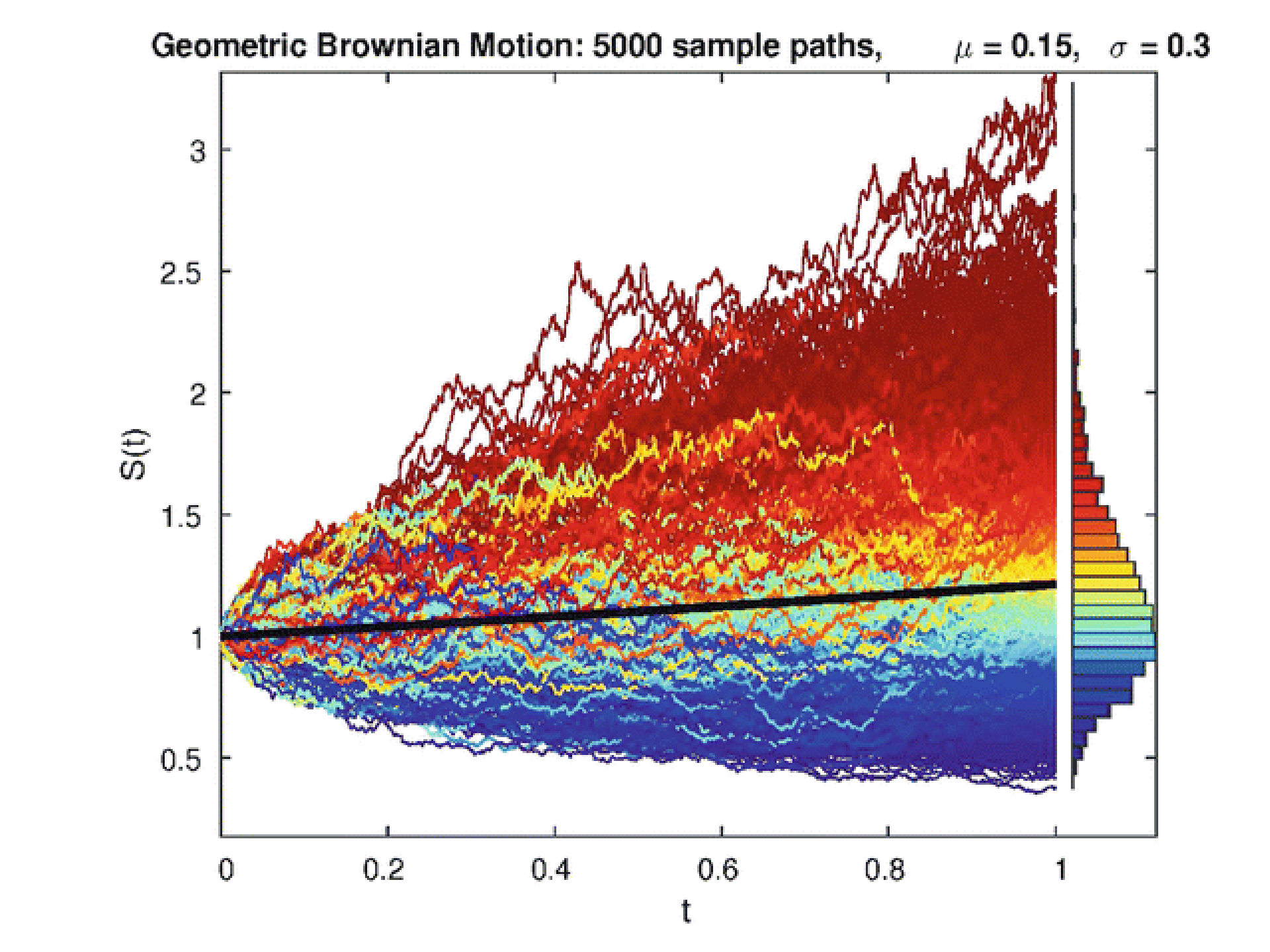
# **Lab 4:** Stochastic Differential Equations

**Stochastic Differential Equations** (SDEs) are extensively used in financial engineering. They are used as models to describe the future behaviour of financial assets, such as the price of a stock, interest rates, or the likelihood of bankruptcy. In this lecture we will study Geometric Brownian Motion (GBM), which is one of the most common types of SDEs.

In particular, we will:

- Describe GBM and its solution via the lognormal distribution
- Look at how GBM the classical approaches to GBM in terms of analytical and numerical methods
- Understand how GBM can be modelled using the tools of quantum computing
- Finally, we will present a number of Python programs implementing the classical and quantum versions of GBM


DIS US $ Market P70.19 / 70.49P 1x2
Prev 70.54 Vol 2,800
DIS US Equity 95) Save As 96) Actions 97) Edit 98) Table Line Chart
12/03/2012 - 12/02/2013 Total Retur Line 11) Compare Mov. Avgs Volume USD
1D 3D 1M 6M YTD 1Y 5Y Max Daily Security/Study Event
Total Return Index (Gross Dividends) 71.6184
High on 11/26/13 72.2682
Average 61.9866
Low on 12/03/12 49.29
71.6184
70
65
60
55
50
Volume 2600
SMAVG (15) 5.281M
30M
20M
10M
5.281M
2600
Dec 2012
Jan
Feb
Mar
Apr
May
Jun
2013
Jul
Aug
Sep
Oct
Nov
Australia 61 2 9777 8600 Brazil 5511 3048 4500 Europe 44 20 7330 7500 Germany 49 69 9204 1210 Hong Kong 852 2977 6000
Japan 81 3 3201 8900 Singapore 65 6212 1000 U.S. 1 212 318 2000 Copyright 2013 Bloomberg Finance L.P.
SN 175660 GMT GMT+0:00 H438-5448-1 02-Dec-2013 13:16:22


Geometric Brownian Motion: 5000 sample paths, $\mu$ = 0.15, $\sigma$ = 0.3
S(t)
t
0
0.2
0.4
0.6
0.8
1
0.5
1
1.5
2
2.5
3

# The Problem: Geometric Brownian Motion

**Geometric Brownian** motion is a type of stochastic process.

Stochastic processes are ideal mathematical tools to describe the behavior of financial variables. Remember watching the graphs of a stock index such as the Dow Jones or the EUR/GBP exchange rates.

What do they have in common?

Clearly their evolution is not smooth.

Financial graphs are full of ups and downs, and broken features, dramatic changes. In short they are what mathematicians call **fractals**.

**And how we can model fractals?**

Well, using stochastic differential equations. It was the **French mathematician Bachelier** who first introduced fractals via Brownian motion as a model to capture this “broken” behavior observed in the financial markets.

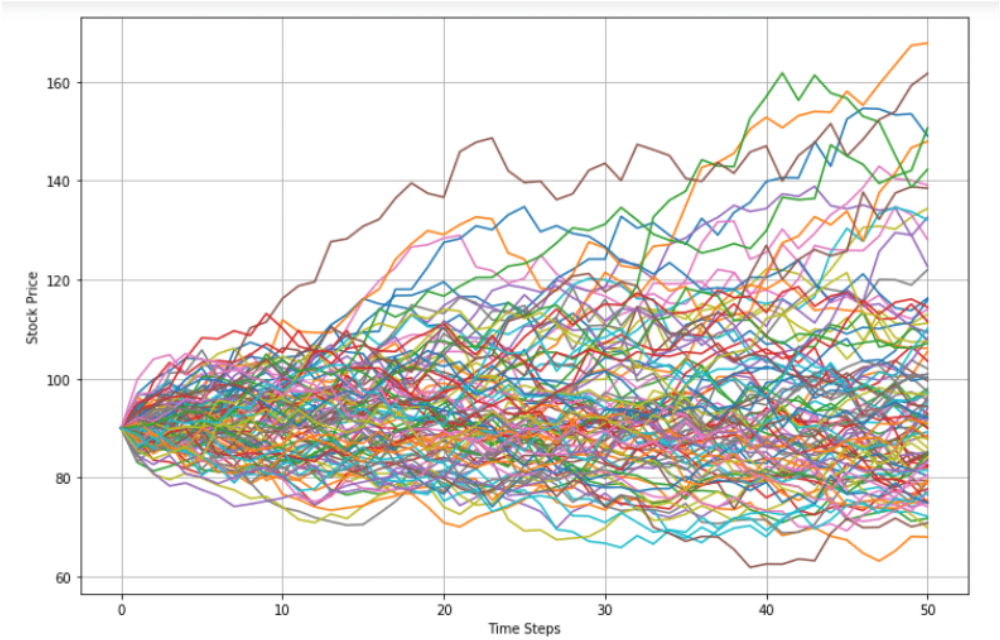
This was in 1901 and largely forgotten. It was not until the late 20th century that the model was revised and it became the basis of such important financial theories such as the pricing of financial derivatives. In fact, GBM is at the heart of the **Black Scholes** option pricing model of 1973.

Even though stochastic differential equations in general, or GBM in particular, may seem esoteric this is very far from the truth.

As we will showing this lecture, it is very easy to construct a simple stochastic process in a computer program using random numbers generators and Monte Carlo simulation.

Our point of departure is then the GBM stochastic process of the form:

$$dS = rSdt + vSdW$$




We are going to use GBM as a way to **project into the future** information we have of today.

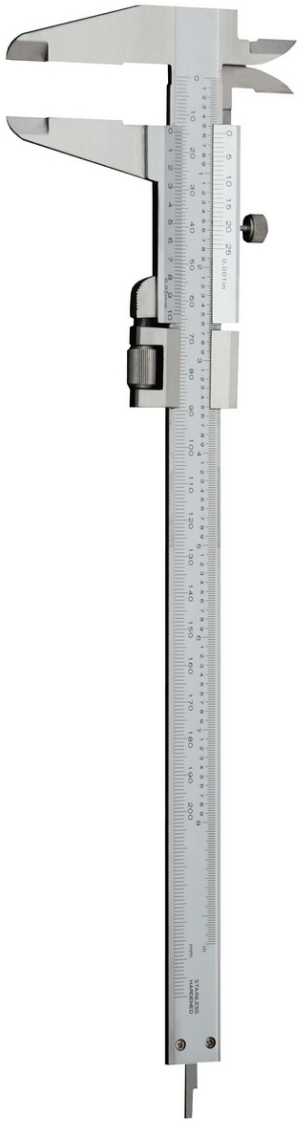
For example if we have the value of a stock price today, **we will use GBM** to generate many possible ways in which this stock price can evolve into the future. We can say that these are different possible futures or scenarios that the stock can take in the next week, month, or years.

**How do we use GBM in practice?**

Well we would need to solve it. In the case of the standard plain GBM, we are fortunate to have an analytical solution. This is very important for us at this stage because we will use it as a benchmark.

However in practice, we rarely have the luxury to have analytical solutions. The vast majority of financial calculations are done using numerical approximations.

This could include Monte Carlo simulation, like we will describe in this book, but also other types of numerical techniques such as binomial trees, partial differential equations, machine learning, and others.



Important: in financial engineering, we may used different models to describe stocks, interest rates or forex rates, but what is common is that our main interest is to find the expected value of an statistical distribution. This is what we are going to do in this chapter.

In the classical approach we will do this with Monte Carlo simulation which converges slowly to this expected value.

Quantum computing solution that we will shows does the same thing, i.e. find the expected value of an statistical distribution, but it does it much faster – in fact with quadratic improvement.

# The Classical Solution

## Solution 1: analytical solution

GBM (equation 1) can be shown to have the solution

$$f(S;\mu,\sigma) = \frac{1}{S\sigma\sqrt{2\pi}}\exp\left[-\frac{(\log(S)-\mu)^2}{2\sigma^2}\right]$$

Which depends on parameters mu and sigma, whilst S is the independent variable (which can represent stock price for example). The above equation can be shown to be the probability density function of the lognormal distribution,

$$f(S;\mu,\sigma) = PDF_{LOGNORM}\,(S;\mu,\sigma)$$

The mapping between the variables used in the GBM (r,v) Andy lognormal distribution(mu, sigma) is

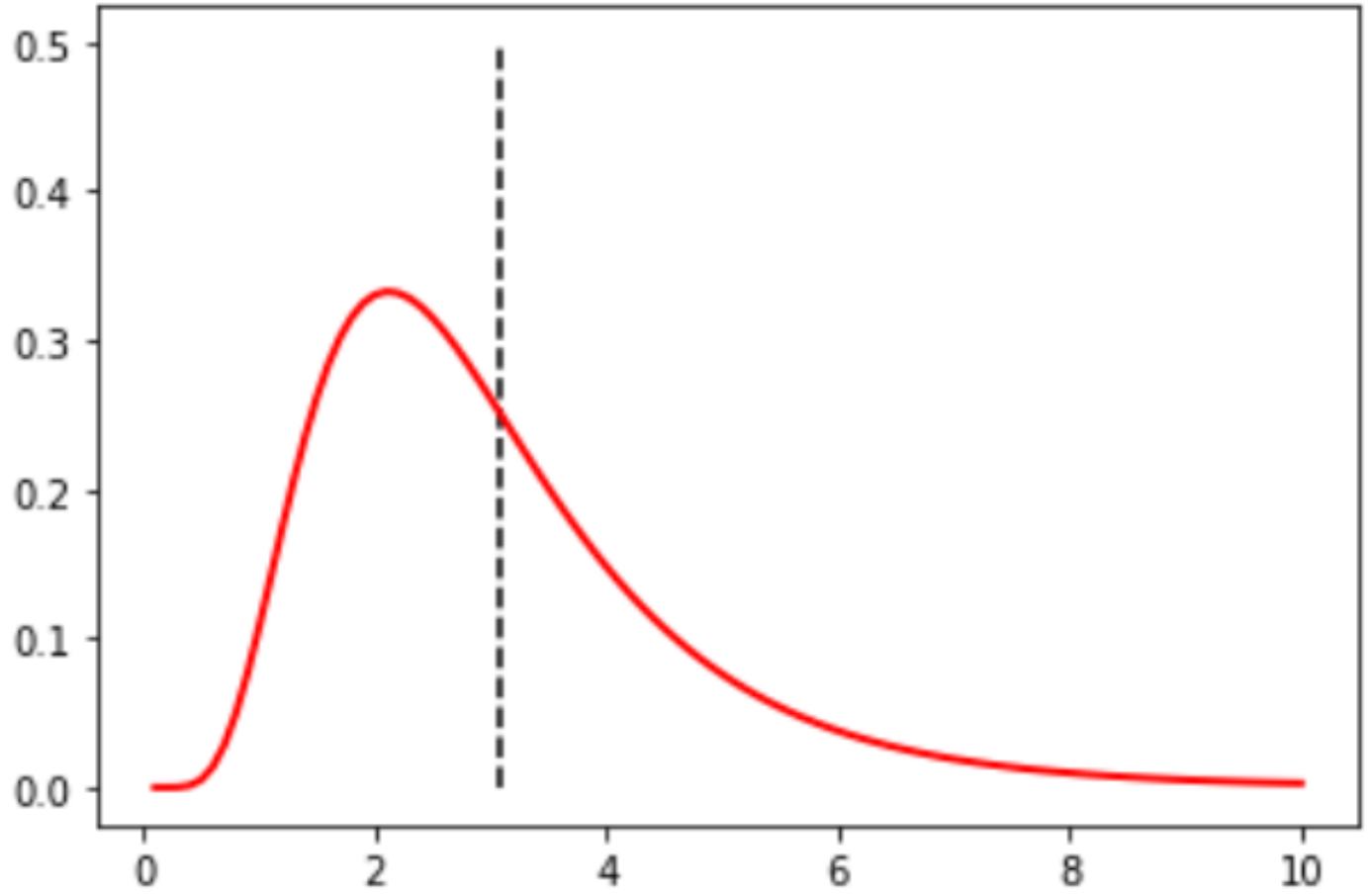
$$\mu = \left[\left(r - \frac{1}{2}v^2\right)T + \log\left(S_0\right)\right]$$

$$\sigma = v\sqrt{T}$$

Finally, the expected value, or the mean of the lognormal distribution can be computed as:

$$\bar{S} = \mathrm{E}[S] = \exp\left(\mu + \frac{1}{2}\sigma^2\right)$$

```
# CODE 4.1 Lognormal distribution PDF graph
import numpy as np
import matplotlib.pyplot as plt
```




```
_____________ parameters _____________
mu =  1.0  sigma =  0.5

_____________ results _____________
exact_mean =  3.080216848918031
```

## Solution 2: numerical solution

The numerical solution of GBM can be obtained by discretizing GBM as:

$$dS = rSdt + vSdW$$

$$\Delta S = rS\Delta t + vS\Delta W$$

$$S^{i+1} - S^i = rS^i\Delta t + vS^i\varepsilon_i\sqrt{\Delta t}$$

$$S^{i+1} = S^i + rS^i\Delta t + vS^i\varepsilon_i\sqrt{\Delta t}$$

$$S^{i+1} = S^i\left(1 + r\Delta t + v\varepsilon_i\sqrt{\Delta t}\right)$$

Which can then be used step by step to move forward from time zero to time T. Once this process is repeated many times, we can determine the final value as the average of the ST generated.

$$S_T^j = S^N$$

$$\bar{S} = \mathrm{E}[S] = \frac{1}{M}\sum_{j=1}^{M} S_T^j$$

# The Composition

In both cases, analytical and numerical, we obtain an estimate of the underlying variable of interest at some future date T. As mentioned before, we are also interested to know what is the expected value of the contract and that future date. This means that using each individual estimate of the underlying, we will then evaluate the contract using a payoff function H, and then take the average. This will results in the expected price of the contract and not only the expected price of the underlying.

In the example we use in this chapter, the payoff H will be represented by a simple linear function. This has the form alpha times X plus beta.

$$H = H(x; parameters)$$

$$H(x; \alpha, \beta) = \alpha x + \beta$$

$$H(x; 1,0) = x$$

As before we can compute the expected value as the average of the terminal values for the estimates of the underlying

$$\bar{S} = \mathrm{E}[S_T] = \frac{1}{M}\sum_{j=1}^{M} S_T^j$$

And we can do the same thing for the expected payoff:

$$\bar{H} = \mathrm{E}[H(S_T)] = \frac{1}{M}\sum_{j=1}^{M} H\left(S_T^j\right)$$

# Python Lab

# LABORATORY 1: Classical Analytical Solution

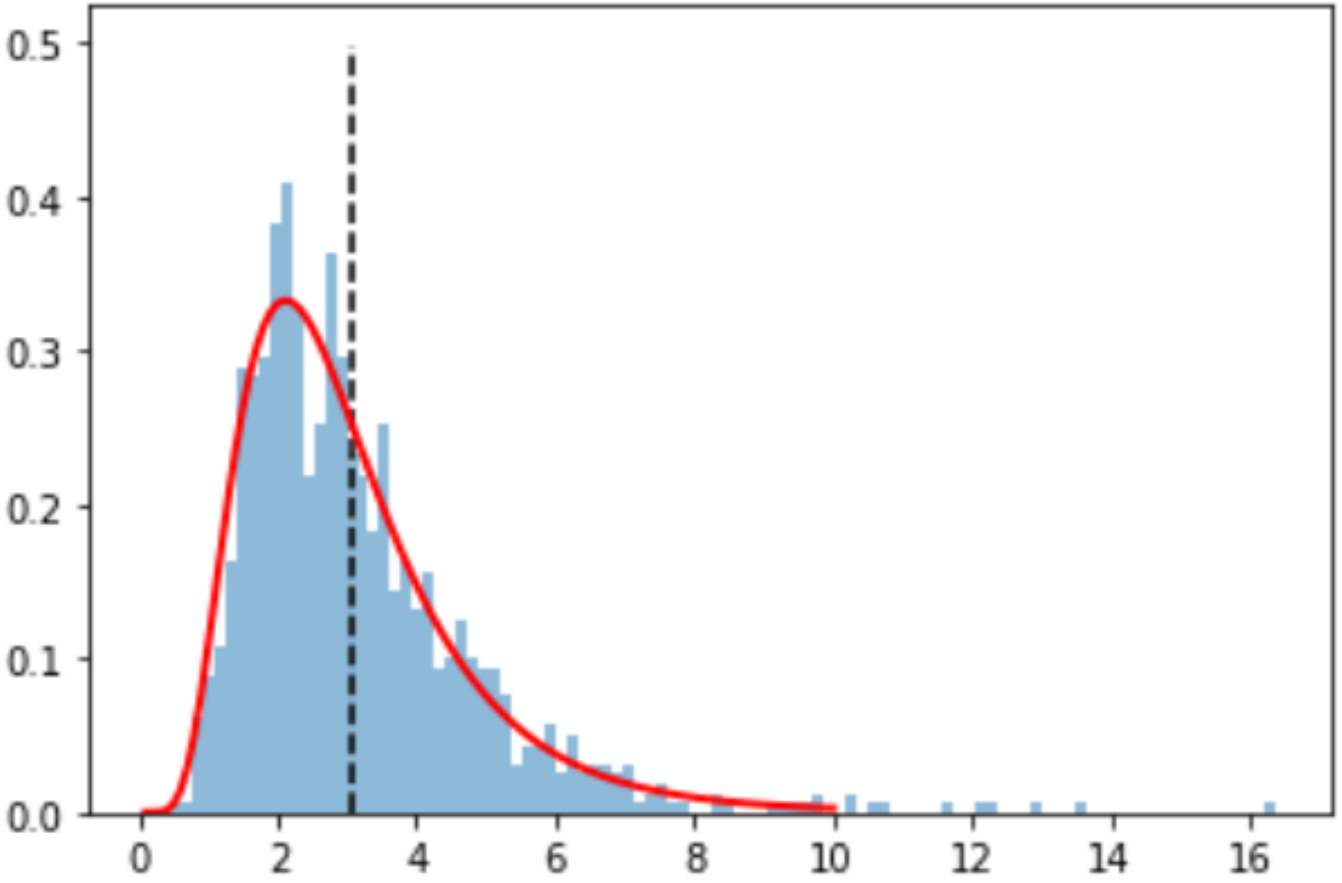
This code illustrates the analytical methods to obtain the estimated mean value of the Lognormal distribution. We have used parameters mu=1 and sigma=0.5. The technique is based on the sampling of distribution based on the Python function xxxx. We have sampled the distribution M=100 times and then simply taking the average of samples. As can be seen, the exact meaning of this tuition is 3.0802, which compares with the estimated value of the Monte Carlo sampling of 3.1801.

The graph illustrates an overlay of the lognormal probability density function in red, the histogram of the samples, and the exact meaning of distribution indicated by the dashed vertical line.

## Code 4.2 Classical Analytical Solution

In the following code, we present the use of method `lognormal()` from module `random` to generate random numbers from a lognormal distribution.

```
# CODE 4.2 Lognormal distribution MC sampling
import numpy as np
import matplotlib.pyplot as plt
```




```
____________ parameters ____________
mu =  1.0  sigma =  0.5

_____________ results _______________
Exact_mean       =  3.080216848918031
MC_sample_mean   =  3.180187168249 46

_____________________________________
```

## LABORATORY 2: Numerical Solution

In this laboratory we ought to follow the trajectory of the underlying variable up to some future time T. Each path represents a possible scenario or evolution of the underlying price into the future. The code that we present here implements the iterated process described in the numerical solution section described before.

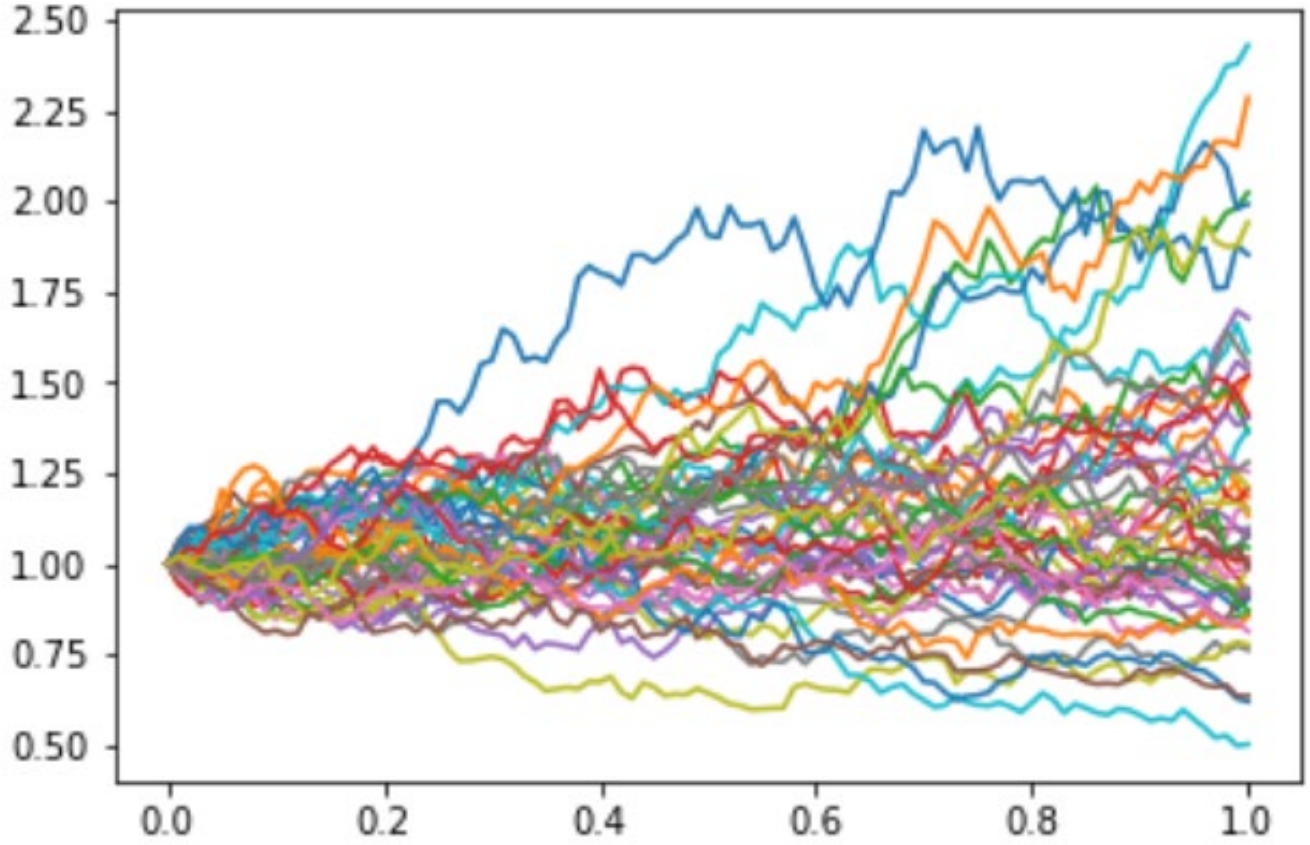
This is  based in both random number generation and Monte Carlo simulation. The code is composed of three parts:

**The first Part,** the necessary libraries are loaded and the parameters of the problem are specified.

**The second part** represents the iterated process that is required to advance from time T = 0 to T. And the the M repetitions required. These are captured using a double for loop.

**The third** and final part is where the average value is computed and the results presented in graphical form.

```
# CODE 4.3 Geometric Brownian Motion with Monte Carlo
import math
import scipy as sp
import numpy as np
import matplotlib.pyplot as plt
```

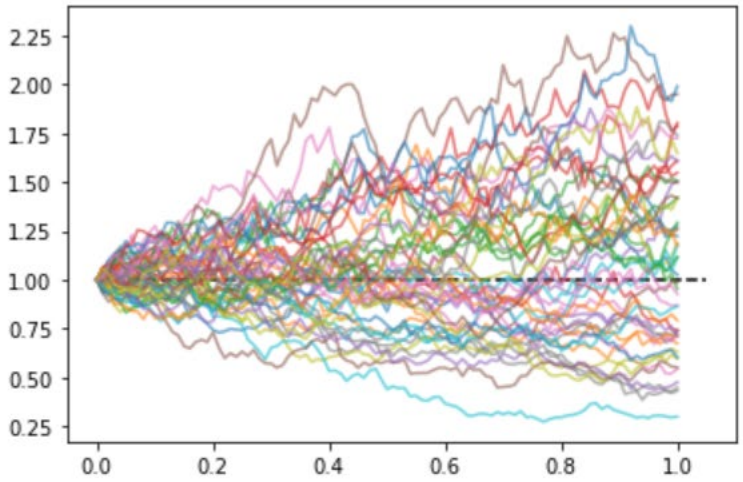


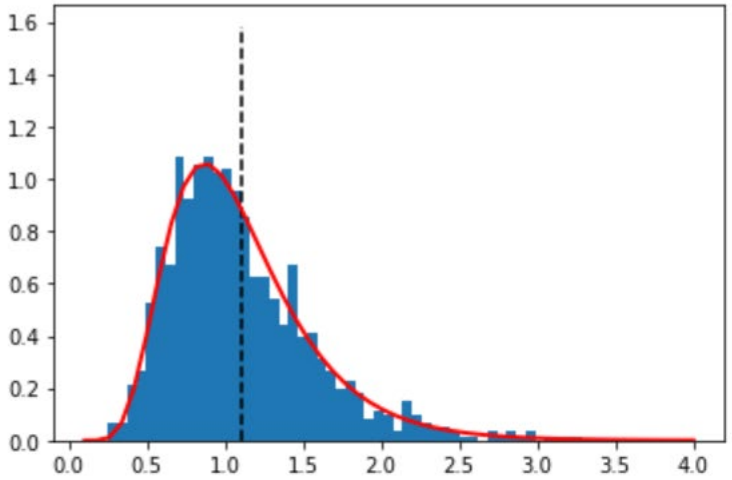

```
____________ parameters ____________
S0= 1  vol= 0.3  r= 0.1  T= 1
N =  100  M =  1000
mu =  0.0550000000000001  sigma =  0.3

____________ results _____________
Exact_mean =  1.1051709180756477
MC_ST_mean =  1.1043202409665795

_____________________________________
```

## Code 4.4 Classical Numerical Solution: distribution

We can explore in more detail the behavior of the underlying at maturity with plot a histogram of the values at this time. In the next code we again generate multiple trajectories of the evolution of the underlying variable, but this time we collect the values of the underlying at maturity and generates an histogram of them. This is illustrated in the next code and figure.

```
# CODE 4.4 GBM PATH MC SAMPLING
import math
import scipy as sp
import numpy as np
import matplotlib.pyplot as plt
```




```
___________ parameters ___________
S0= 1  vol= 0.4  r= 0.1  T= 1
N =  100  M =  1000
mu =  0.01999999999999999  sigma =  0.4

____________ results _____________
Exact_mean      =  1.1051709180756477
MC_sample_mean  =  1.1012865878720308
__________________________________
```

# The Quantum Solution

# The Quantum Computing Solution

How can we approach the problem of computing the expected value of the statistical distribution using quantum computing? The quantum solution relies on a very different approach than the classical solution. Quantum solution is going to be based on the Grover's search algorithm and its generalization, the so-called quantum amplitude estimation. On the one hand this are advanced tools in quantum computing, but on the other they are easy to use.

Fundamentally the classical and the quantum solutions are similar. Conceptually they are trying to do the same: estimate the expected value of the mean of a statistical distribution. However, the approach the solution in very different ways.

Whilst the classical solution samples statistical distribution using random numbers and therefore constructs a set of samples from which we can then calculate statistical properties, such as the mean. In contrast, the quantum solution relies on a search algorithm, the Grovers search algorithm, which when implemented in quantum computers turns out to be extremely efficient.

In terms of computational efficiency, it has been shown that the classical Monte Carlo sampling of in a statistical solution is proportional to the square root of the number of samples taken. On the contrary, Montanaro has shown that quantum amplitude estimation technique converges according to the inverse of the number of samples done. This represents a quadratic improvement, which is extremely significant.

# Qiskit Lab

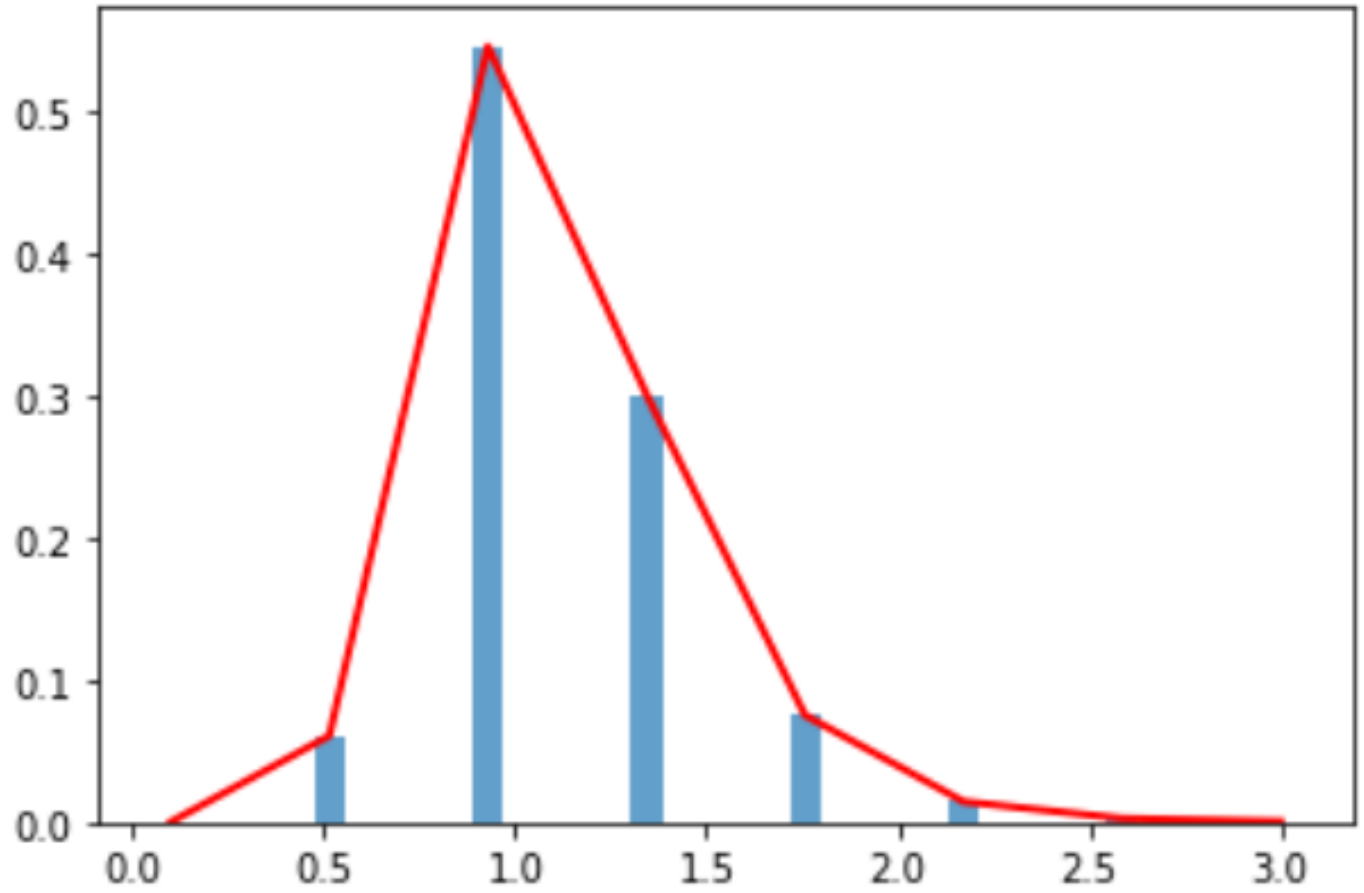
# LABORATORY 3: Quantum Uncertainty Model

In order to capture the lognormal distribution in qiskit, we will use the function uncertainty model with a specified number of qubits. This function allows for the description of not only the lognormal distribution, but also of the normal distribution and the uniform distribution. This flexibility is quite useful in financial calculations. It allows us to describe the statistical behaviour of some underlying variable. Whilst in some cases a live demonstration will be an adequate, like in stock price modelling, in other cases a normal distribution will be more appropriate, such as some types of interest rate models, as will be discussed in a future.

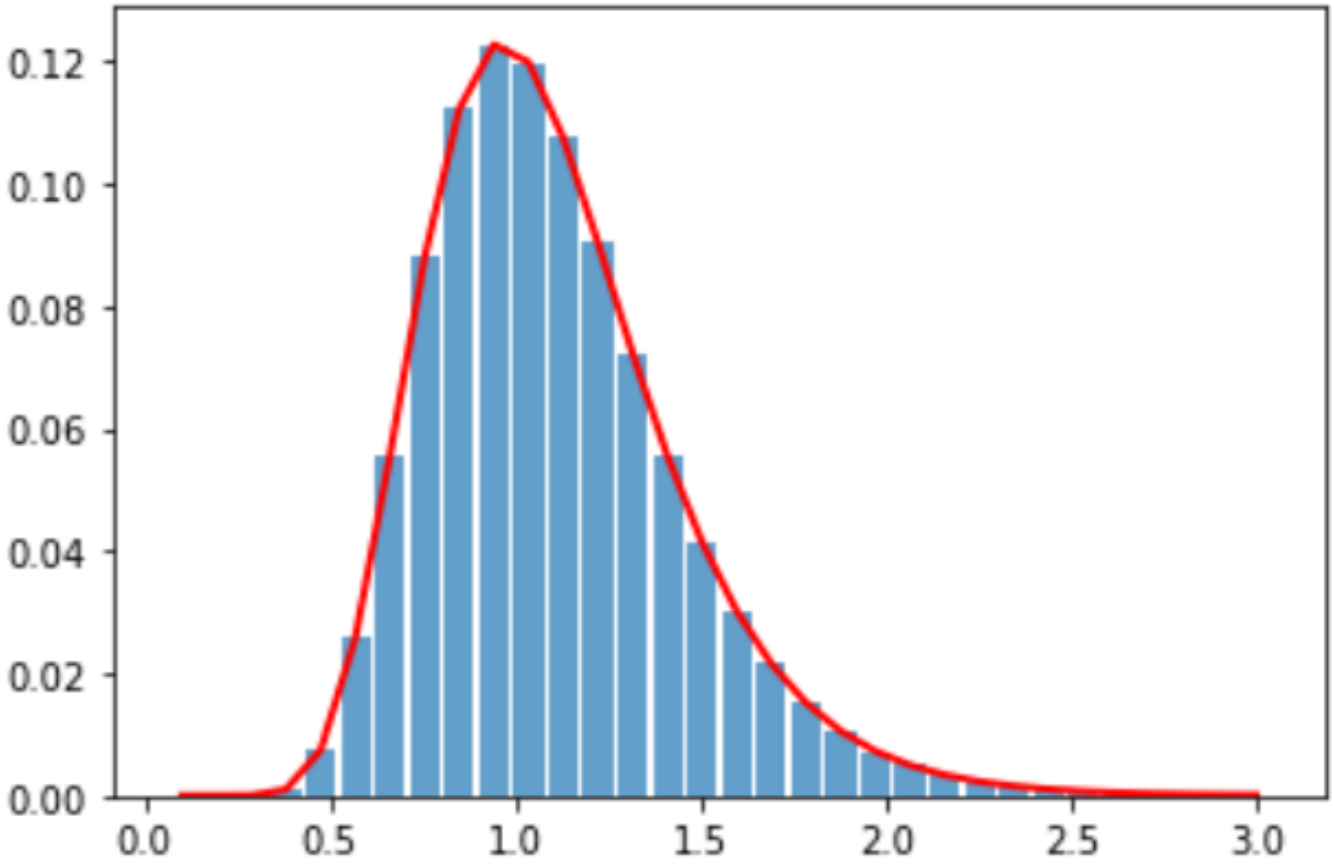
In the accompanying Python code illustrates the amount distribution in qiskit using three qubits. The resultant number of combinations of grid points is 2 elevated t power of 3,  and therefore eight.

## Code 4.6 Quantum LogNormal FEW QUBITS

```
# CODE_4_6_QUANTUM_LOGNORMAL_FEW
import matplotlib.pyplot as plt
import numpy as np
from qiskit import Aer, QuantumCircuit
```




```
_____________ parameters _____________
S0= 1  vol= 0.3  r= 0.1  T= 1
mu =  0.05500000000000001  sigma =  0.3
_____________ quantum LOGN _____________
number qubits  =  3
resulting grid =  8
```




```
____________ parameters ____________
S0= 1  vol= 0.3  r= 0.1  T= 1
mu =  0.05500000000000001  sigma =  0.3
____________ quantum LOGN ____________
number qubits  =  5
resulting grid =  32

_____________________________________
```

# References

The theory of stochastic processes as used finance is extremely complex. I invite Grinko, D., Gacon, J., Zoufal, C. et al. Iterative quantum amplitude estimation. npj Quantum Inf 7, 52 (2021). https://doi.org/10.1038/s41534-021-00379-1

- Montanaro A, Quantum speedup of Monte Carlo methods, Proc. Roy. Soc. Ser. A, vol. 471 no. 2181, 20150301, 2015.
- Glasserman P. Monte Carlo Methods in Financial Engineering, Springer 2003.
- Wilmott P, The Mathematics of Financial Derivatives: A Student Introduction, Cambridge University Press, 1995.
- Damiano B, Mercurio F. Interest Rate Models-theory and Practice: With Smile, Inflation and Credit. Springer Nature, 2007.
- Bolder D. Credit Risk Modelling: Theoretical Foundations, Diagnostic Tools, Practical Examples, and Numerical Recipes in Python. Springer Nature, 2018.