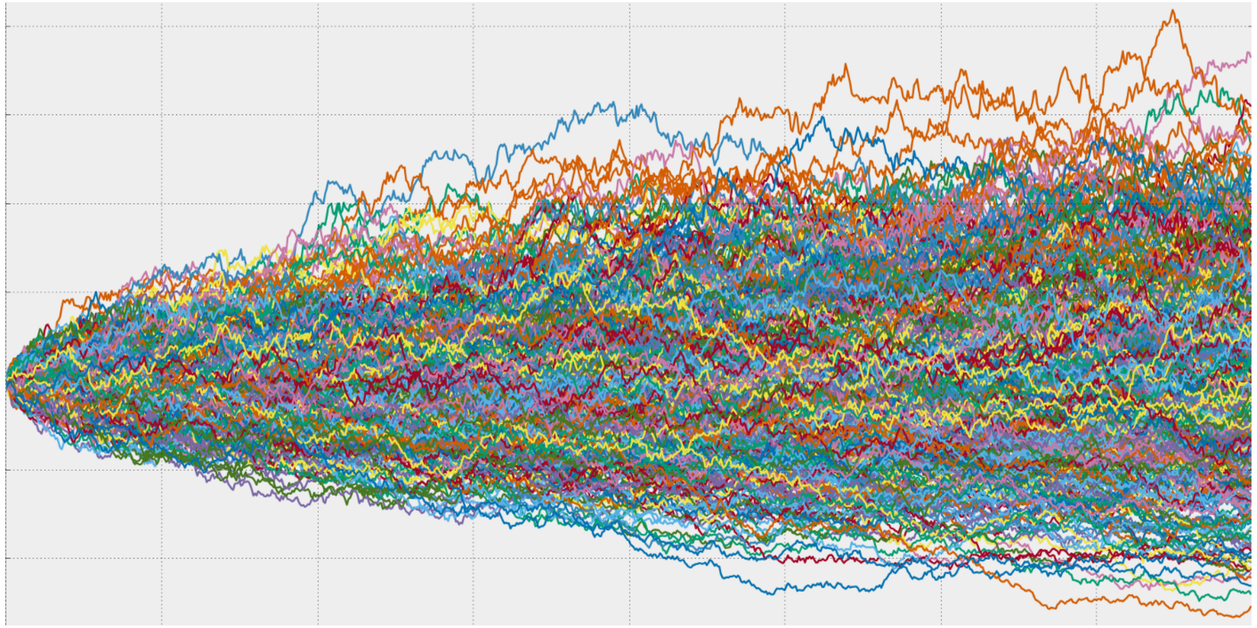
Getting Realistic Market Volatility

A path generator for Monte Carlo simulations using GANs

Ninad Mohale

Why? Monte Carlo methods are used to price financial instruments every day.



https://towardsdatascience.com/monte-carlo-simulation-in-r-with-focus-on-financial-data-ad43e2a4aedf

Random walk - imitates a market property (eg volatility)
An average of thousands of iterations is used to price in that market property.

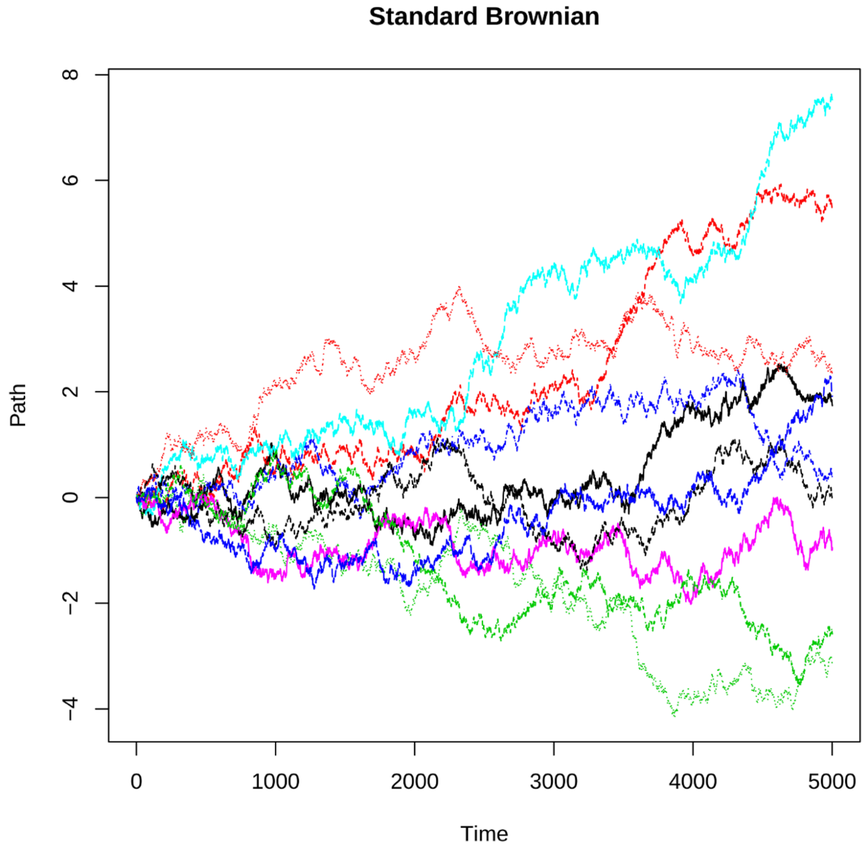
These Monte Carlo methods use path generation algorithms to generate random paths.

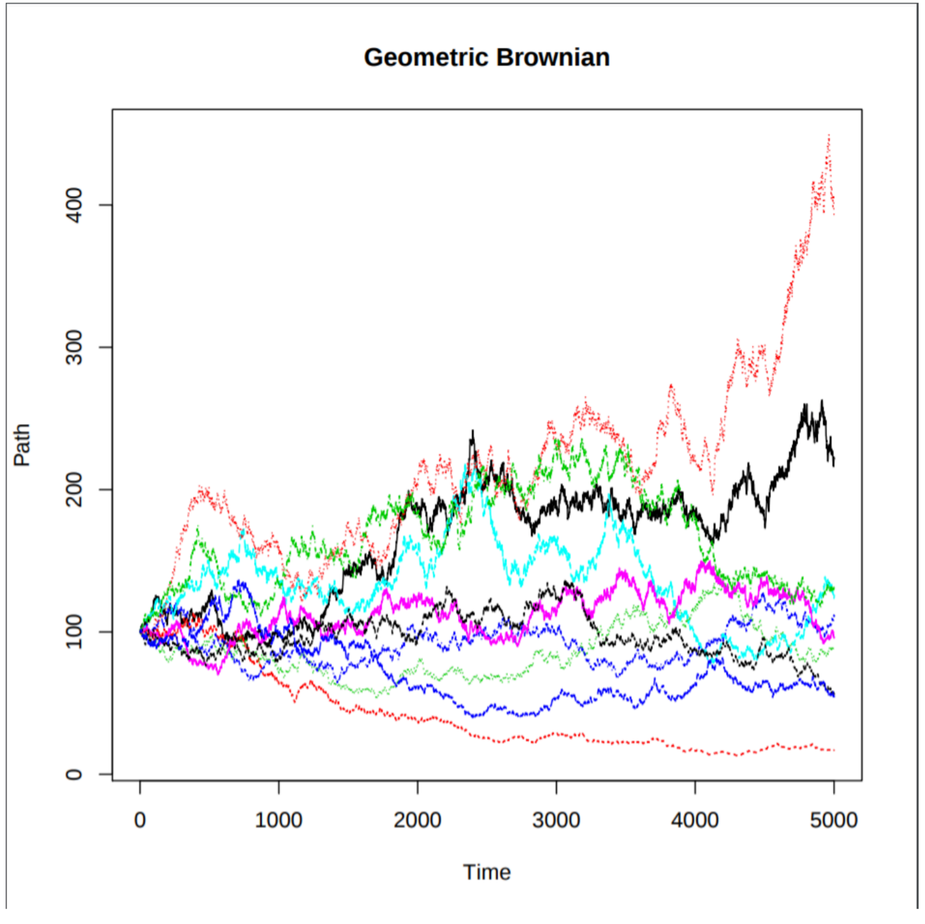
$$X(t_{i+1}) = X(t_i) + \int_{t_i}^{t_{i+1}} \mu(s)\,ds + \sqrt{\int_{t_i}^{t_{i+1}} \sigma^2(u)\,du}Z_{i+1},$$

Solution of Stochastic Differential Equation for Brownian Motion

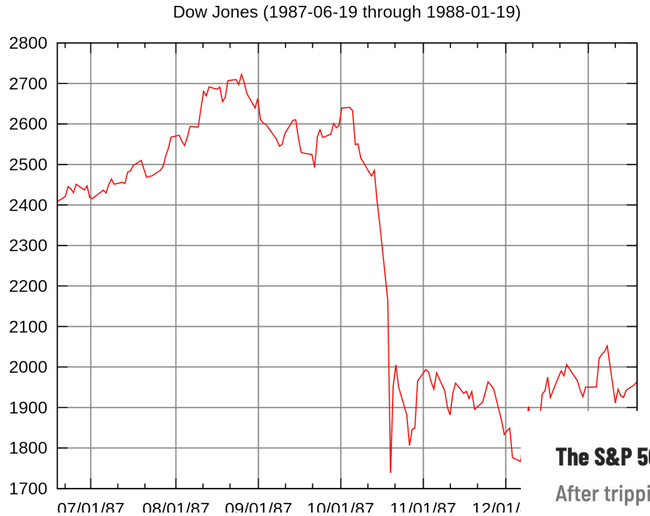
$$S(t) = S(0)\exp\left([\mu - \tfrac{1}{2}\sigma^2]t + \sigma W(t)\right)$$

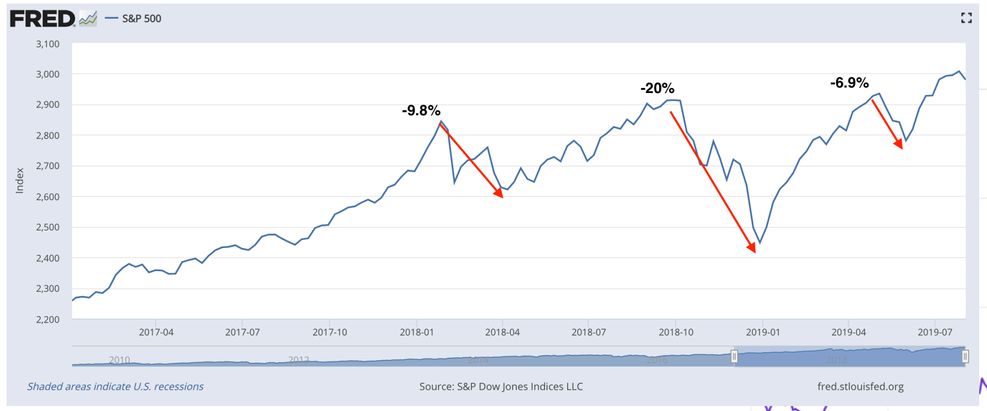
Solution of S.D.E for Geometric Brownian Motion





Market volatilities can happen outside of these ‘ranges’. eg. Nicholas Taleb’s Black Swan events, 7σ events.





These events have extreme effects on financial decisions

My model will give more realistic random paths, which could be used to better price financial instruments.

How?

I will train a Generative Adversarial Network with real stock price movement data, volatility index movement data, risk free return rate data.

- The adversarial network will be able to identify realistic volatility movement
- The generative network will generate random paths and send them to the adversarial network
- Error from adversarial network will train generative network, until both get better and there is just marginal improvement.

In [ ]:
```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout, LeakyReLU
```

In [ ]:
```
# Sample GAN
```

In [ ]:
```
def generator(Z,hsize=[16, 16],reuse=False):
    with tf.variable_scope("GAN/Generator",reuse=reuse):
        h1 = tf.layers.dense(Z,hsize[0],activation=tf.nn.leaky_relu)
        h2 = tf.layers.dense(h1,hsize[1],activation=tf.nn.leaky_relu)
        out = tf.layers.dense(h2,2)

    return out
```

In [ ]:
```
def discriminator(X,hsize=[16, 16],reuse=False):
    with tf.variable_scope("GAN/Discriminator",reuse=reuse):
        h1 = tf.layers.dense(X,hsize[0],activation=tf.nn.leaky_relu)
        h2 = tf.layers.dense(h1,hsize[1],activation=tf.nn.leaky_relu)
        h3 = tf.layers.dense(h2,2)
        out = tf.layers.dense(h3,1)

    return out, h3
```