Options pricing with Python - Examples

May 26, 2017

1 The Binomial tree algorithm

1.1 Simulation of the American Roulette

```
In [1]: # Roulette example
        import numpy as np

        rouletteWheel = np.array(range(1,38))

        def playGame(number):

            winningNumber = np.random.choice(rouletteWheel)

            if(number == winningNumber):
                gain = 36
            else:
                gain = 0

            return gain, winningNumber

        number = 12

        gain, winningNumber = playGame(number)

        print("The winning number ", winningNumber)
        print("Your P/L is",gain,"dollars!")
```

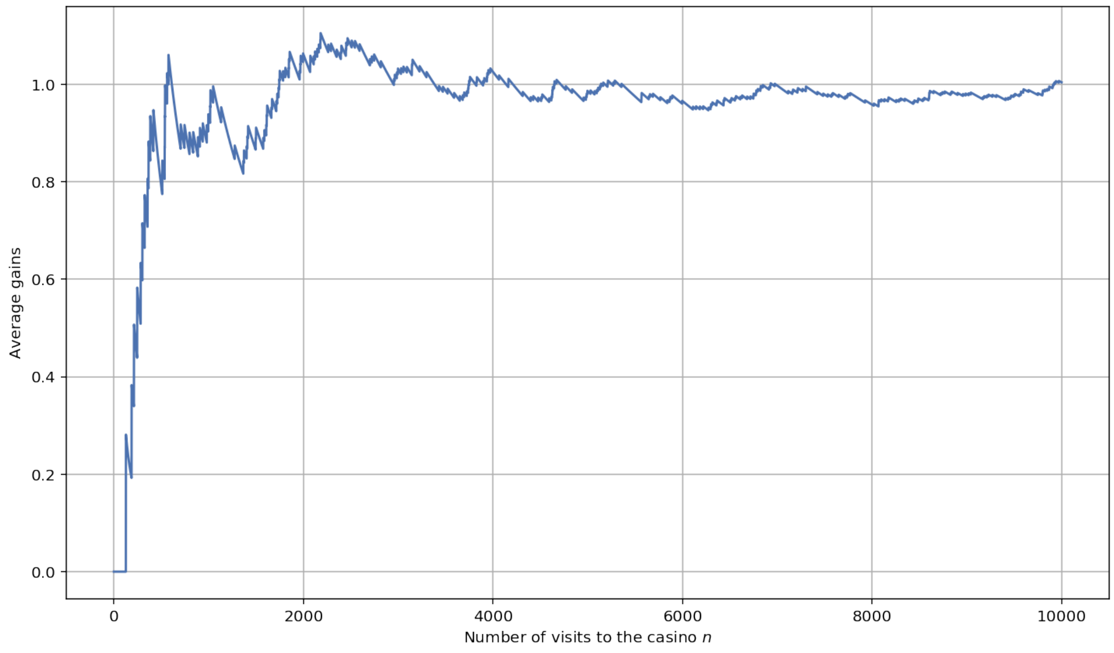
```
The winning number  31
Your P/L is 0 dollars!
```

1.2 Compute the price of the Roulette bet

```
In [2]: # Simulate 10,000 trials of Roulette

        import matplotlib.pyplot as plt
        import numpy as np
```

```
number = 10
winnings = []
meanWinnings = []

for i in range(0,10000):
    gains, winningNumbers = playGame(number)
    winnings.append(gains)
    mean = np.mean(winnings)
    meanWinnings.append(mean)

n = np.arange(10000)
plt.style.use('seaborn-deep')
plt.plot(n,meanWinnings)
plt.xlabel('Number of visits to the casino $n$')
plt.ylabel('Average gains')
plt.grid(True)
plt.show()
```

Out[2]:



2 The binomial tree method

2.1 Binomial tree algorithm to price standard European call and put

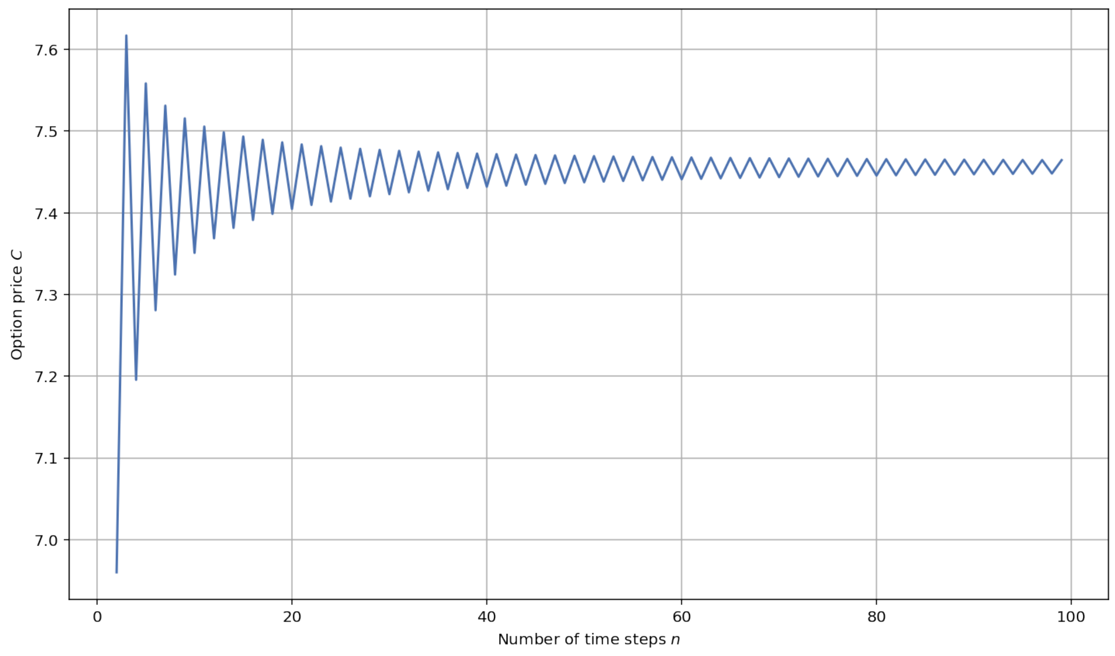
In [4]:
```
# Binomial tree algorithm to price European options
import numpy as np
import math

def buildCRRBinTreeEuropean(spot, strike, rate, vol,
                             T, N, optionType):
    S = np.zeros((N+1,N+1),dtype='float')
    C = np.zeros((N+1,N+1),dtype='float')
    i,j = 0,0

    # Multiplicative binomial tree parameters
    dt = T/N
    u = math.exp(vol*math.sqrt(dt))
    d = 1/u
    disc = math.exp(-rate*dt)

    # Risk-neutral probabilities
    p = (math.exp(rate*dt)-d)/(u-d)
    q = 1-p

    S[0][0] = spot

    # Initialize all the asset prices
    for i in range(N+1):
        for j in range(i+1):
            S[i][j]=S[0][0]*(u**j)*(d**(i-j))


    # Initialize the option values at maturity
    for j in range(N+1):
        if optionType == 'C':
            C[N][j] = max(S[N][j]-strike,0)
        else:
            C[N][j] = max(strike-S[N][j],0)

    # Work backwards through the tree
    for i in range(N-1,-1,-1):
        for j in range(i+1):
            C[i][j]=disc*(p*C[i+1][j+1]+(1-p)*C[i+1][j])

    return S, C

S,C = buildCRRBinTreeEuropean(100,100,0.06,0.10,1,3,'C')

print(C)
```

```
[[  7.61708311   0.           0.           0.        ]
 [  2.49062163  10.48602171   0.           0.        ]
 [  0.           3.84744326  14.22022291   0.        ]
 [  0.           0.           5.9434237   18.91099436]]
```

2.2 Convergence of the binomial method to the Black-Scholes price

```
In [5]: import matplotlib.pyplot as plt

        prices = []
        steps = list(range(2,100))

        for n in range(2,100):
            S,C = buildCRRBinTreeEuropean(100,100,0.06,0.10,1,n,'C')
            prices.append(C[0][0])

        plt.plot(steps,prices)
        plt.xlabel('Number of time steps $n$')
        plt.ylabel('Option price $C$')
        plt.grid(True)
        plt.show()
```

Out[5]:



2.3 Binomial tree algorithm - $O(n)$ memory space and $O(n(n-a))$ running time

```
In [6]: # Binomial tree algorithm to price European options
        # It uses linear space
        import numpy as np
```

```python
import math

def buildCRRBinTreeEuropean(S, X, r, sigma,
                            T, N, optionType):
    C = np.zeros(N+1,dtype='float')

    # i : Time steps [0..N]
    # j : Number of down moves
    i,j = 0,0

    # Multiplicative binomial tree parameters
    dt = T/N
    u = math.exp(sigma*math.sqrt(dt))
    d = 1/u
    disc = math.exp(-r*dt)

    # Risk-neutral probabilities
    p = (math.exp(r*dt)-d)/(u-d)
    q = 1-p

    a = math.log(X/(S*(d**N)))/math.log(u/d)
    a = math.ceil(a)


    # Initialize the option values at maturity
    for j in range(N+1):
        if optionType == 'C':
            C[j] = max((S*(u**(N-j))*(d**j)) - X,0)
        elif optionType == 'P':
            C[j] = max(X - (S*(u**(N-j))*(d**j)),0)

    # Work backwards through the tree
    for i in range(N-1,-1,-1):
        for j in range(min(N-a+1,i+1)):
            C[j]=disc*(p*C[j]+q*C[j+1])

    return S, C[0]

S,C = buildCRRBinTreeEuropean(100,100,0.06,0.10,1,3,'C')

print(C)
```

```
7.61708311062
```

2.4 An optimal algorithm - $O(1)$ memory space and $O(n-a)$ running time

```
In [7]: # Binomial tree algorithm using a single variable instead of
        # a whole array and linear running time

        import numpy as np
        import math

        def optimalEuropean(S, X, r, sigma,
                                 T, N, optionType):

            dt = T/N
            u = math.exp(sigma*math.sqrt(dt))
            d = 1/u
            disc = math.exp(-r*T)

            p = (math.exp(r*dt)-d)/(u-d)
            q = 1-p

            a = math.log(X/(S*(d**N)))/math.log(u/d)
            a = math.ceil(a)

            S = S * (u**a) * (d**(N-a))
            b = (math.factorial(N)/(math.factorial(N-a)*math.factorial(a))) \
            * (p**a) * (q**(N-a))

            C = disc * b * (S-X)

            for j in range(a+1,N+1):
                b = ((p*(N-j+1)) / (q*j)) * b
                S = S * (u/d)
                C = C + disc * b * (S-X)

            return C

        C = optimalEuropean(100,100,0.06,0.10,1,3,'C')

        print(C)
```

```
7.617083110621771
```

2.5 Pricing American options

```
In [8]: # Binomial tree algorithm to price American options
        import numpy as np
        import math

        def buildCRRBinTreeAmerican(spot, strike, rate, vol,
                                 T, N):
            S = np.zeros((N+1,N+1),dtype='float')
            P = np.zeros((N+1,N+1),dtype='float')
            i,j = 0,0

            # Multiplicative binomial tree parameters
            dt = T/N
            u = math.exp(vol*math.sqrt(dt))
            d = 1/u
            disc = math.exp(-rate*dt)

            # Risk-neutral probabilities
            p = (math.exp(rate*dt)-d)/(u-d)
            q = 1-p

            S[0][0] = spot

            # Initialize all the asset prices
            for i in range(N+1):
                for j in range(i+1):
                    S[i][j]=S[0][0]*(u**j)*(d**(i-j))


            # Initialize the option values at maturity
            for j in range(N+1):
                P[N][j] = max(strike-S[N][j],0)

            # Work backwards through the tree
            for i in range(N-1,-1,-1):
                for j in range(i+1):
                    P[i][j]=max(disc*(p*P[i+1][j+1]+(1-p)*P[i+1][j]),
                                strike-S[i][j])

            return S, P

        S,P = buildCRRBinTreeAmerican(100,100,0.06,0.10,1,100)

        print(P[0][0])
```

```
2.22993199989
```

2.6 Optimal algorithm to price down-and-in call european options

```
In [9]: # Optimal algorithm for European down-and-in call barrier  options

        import numpy as np
        import math
```

```
def priceDownAndInCall(S, X, H, r, sigma,
                         T, N, optionType):

    # Binomial tree parameters
    dt = T/N
    u = math.exp(sigma*math.sqrt(dt))
    d = 1/u
    disc = math.exp(-r*T)

    a = math.log(X/(S*(d**N)))/math.log(u/d)
    a = math.ceil(a)
    h = math.log(H/(S*(d**N)))/math.log(u/d)
    h = math.floor(h)

    # Risk-neutral probabilities
    p = (math.exp(r*dt)-d)/(u-d)
    q = 1-p

    # Start at layer 2h
    S = S * (u**(2*h)) * (d**(N-2*h))
    b = 1 * (p**(2*h)) * (q ** (N-2*h))

    C = b * (S-X)

    for j in range(2*h-1,a-1,-1):
        b = (p*(N-2*h+j+1)/(q*(2*h-j)))*b
        S = S * (d/u)
        C = C +  b * (S-X)

    return disc * C

C = priceDownAndInCall(100, 102, 97, 0.05, 0.20, 1, 10, 'C')
print(C)
```

```
1.1782682385438155
```

2.7 Nuts and bolts of GBM

2.7.1 BM

A gambler enters the Bellagio casino and decides to play a game. The gambler's initial wealth is 0. A fair coin is flipped n times. There are two possible outcomes, for each trial. On heads, the gambler gains +\$1, on tails the gambler loses -\$1.

If we let X_i to be the random amount, either +1 or -1, you make on the ith toss, then we have,

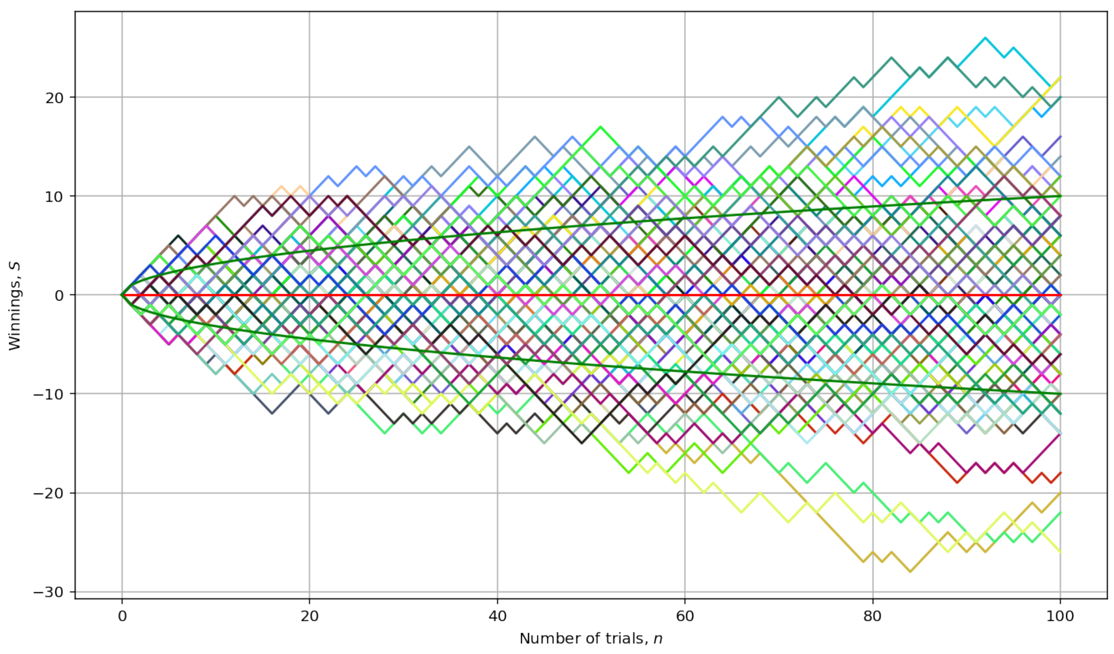
$$E(X_i) = 0, E(X_i^2) = 1, E(X_i X_j) = 0$$

Define $S$ to be the gambler's gain in $n$ trials.

$$S = \sum_{j=1}^{n} X_j$$

Assume the gambler visits the casino on 100 occasions and played $n = 100$ rounds each. A simulation of the gambler's gain would look like this.

```
In [10]: import numpy as np
         import matplotlib.pyplot as plt

         # Input parameters
         sample_space = [1,-1]
         x = list(range(0,101))

         # 100 realizations of the gambler's gain
         for i in range(100):

             x_i = [0]
             s_i = [0]

             # Flip a coin hundred times
             coin_tosses = np.random.choice(sample_space,100)
             x_i = np.append(x_i,coin_tosses)

             # Take the cumulative sum
             s_i = np.cumsum(x_i)

             # Plot the gambler's gains
             plt.plot(x, s_i, color=np.random.rand(3))


         mean = [0]*101
         plus_1_sd = np.sqrt(x)
         minus_1_sd = -np.sqrt(x)

         plt.plot(x,mean,'r-',
                 x,plus_1_sd,'g-',x,minus_1_sd,'g-')
         plt.xlabel('Number of trials, $n$')
         plt.ylabel('Winnings, $S$')
         plt.grid(True)
         plt.show()

Out[10]:
```



Intuitively, the gambler's winnings S is a binomial random variable.
The average winnings $E(S)$ are,

$$E(S) = E(X_1 + X_2 + \ldots + X_n) = 0$$

The variance(dispersion) of the the winnings $E(S^2)$ are,

$$\begin{aligned} E(S^2) &= E[(X_1 + X_2 + \ldots + X_n)^2] \\ &= E(X_1^2 + X_2^2 + \ldots + X_n^2 + 2X_1X_2 + 2X_1X_3 + \ldots) \\ &= E(X_1^2) + E(X_2^2) + \ldots + E(X_n^2) \\ &= n \end{aligned}$$

As the number of time steps t increases, the gains S will be approximately gaussian $N(0, t)$ with mean $E(S) = 0$ and variance $E(S^2) = t$. It's a brownian motion.

2.7.2 BM with a drift

Black-Scholes treats the continuously compounded returns on an asset as a sum of two components: a deterministic return e.g. the interest earned on the money deposited in a bank account and a random return e.g. the winnings of a gambler.

$$\text{Continuously compounded returns } X = \text{Deterministic return} + \text{Random return}$$

We write,

$$\text{Continuously compounded returns } X = \mu T + \sigma N(0, T)$$

Continuously compounded returns are the log of the asset price. So, the natural log of the asset price are gaussian(brownian) $N(\mu T, \sigma^2 T)$.

First and second moments. The first and second moments of BM can be obtained like this. We know, $E(X) = \mu T, Var(X) = \sigma^2 T$. But, $Var(X) = E(X^2) - [E(X)]^2$. Thus, $\sigma^2 T = E(X^2) - \mu^2 T^2$. Therefore,

$$E(X^2) = \sigma^2 T + \mu^2 T^2$$

2.8 Additive binomial tree

Let x be the natural logarithm of the asset price S. At node (i, j), the asset price is

$$x_{i,j} = x + i\Delta x_u + (i - j)\Delta x_d$$

In a discrete time setting, we assume x can either go upto a level $x + \Delta x_u$ with probability p_u or down to a level $x + \Delta x_d$ with probability $p_d = 1 - p_u$. This is an *additive binomial tree*.

In the Black Scholes world, x is normally distributed with first moment $m_1 = \mu\Delta t$ and second moment $m_2 = \sigma^2\Delta t + \mu^2\Delta t^2$.

I equate the mean and variance of the binomial tree with the mean and variance of the continuous time process for x over the interval Δt.

$$E[\Delta x] = p_u\Delta x_u + p_d\Delta x_d = \mu\Delta t$$

$$E[\Delta x^2] = p_u\left(\Delta x_u\right)^2 + p_d\left(\Delta x_d\right)^2 = \sigma^2\Delta t + \mu^2\left(\Delta t\right)^2$$

Equal jump sizes lead to:

$$\begin{aligned} p_u(\Delta x) + p_d(-\Delta x) &= \mu\Delta t \\ (p_u - p_d)\Delta x &= \mu\Delta t \\ (2p_u - 1) &= \frac{\mu\Delta t}{\Delta x} \\ p_u &= \frac{1}{2} + \frac{\mu\Delta t}{2\Delta x} \end{aligned}$$

and

$$\begin{aligned} E[\Delta x^2] &= p_u\left(\Delta x\right)^2 + p_d\left(-\Delta x\right)^2 = \sigma^2\Delta t + \mu^2\left(\Delta t\right)^2 \\ (\Delta x) &= \sqrt{\sigma^2\Delta t + \mu^2\left(\Delta t\right)^2} \end{aligned}$$

2.9 Implementing additive binomial tree to European options

```
In [11]: import numpy as np
         import math

         def buildAddBinomialTree(spot, strike, rate, vol,
                                  T, N, optionType):
             S = np.zeros((N+1,N+1),dtype='float')
             C = np.zeros((N+1,N+1),dtype='float')
             i,j = 0,0

             # Additive binomial tree parameters
```

```
        dt = T/N
        mu = rate - 0.5*(vol ** 2)
        dxu = math.sqrt((vol**2)*(dt)+(mu*dt)**2)
        dxd = -dxu
        disc = math.exp(-rate*dt)

        # Risk-neutral probabilities
        p = (1/2)+(1/2)*((mu * dt)/dxu)
        q = 1-p

        S[0][0] = spot

        # Initialize all the asset prices
        for i in range(N+1):
            for j in range(i+1):
                S[i][j]=S[0][0]*math.exp((j*dxu) + ((i-j)*dxd))


        # Initialize the option values at maturity
        for j in range(N+1):
            if optionType == 'C':
                C[N][j] = max(S[N][j]-strike,0)
            else:
                C[N][j] = max(strike-S[N][j],0)

        # Work backwards through the tree
        for i in range(N-1,-1,-1):
            for j in range(i+1):
                C[i][j]=disc*(p*C[i+1][j+1]+(1-p)*C[i+1][j])

        return x, C

    S,C = buildAddBinomialTree(100,100,0.06,0.20,1,3,'C')

    print('Option price:',C[0][0])
```

```
Option price: 11.5919912079
```

2.10 Trinomial trees

Trinomial trees are used more in practice then binomial trees since they allow for three states : up, down and middle moves. We can approximate diffusions using trinomial random walks. We can price both Eurpopean and American options with trinomial trees.

The stock price at node (i, j) is given by:

$$S_{i,j} = Su^j \text{ j=-i,-i+1,...,0,...,i}$$

A symmetric tree structure is obtained, when $u = e^{\lambda\sigma\sqrt{t}}$ and $d = e^{-\lambda\sigma\sqrt{t}}$. We can compute the option value in a trinomial tree as the discounted expectation of the future payoffs.

$$f_{i,j} = e^{-r\Delta t}(p_u f_{i+1,j+1} + p_m f_{i+1,j} + p_d f_{i+1,j-1})$$

The optimal value of the stretch parameter λ that produces the best convergence rate is

$$\lambda = \sqrt{3/2}$$

3 Monte Carlo method to price spread options

Monte Carlo simulation is a technique for pricing many types of derivatives, when closed-form analytical solutions are not available, for example to price an Asian option or for simulating multi-factor stochastic processes, for example, to price spread options.

3.1 Simulating geometric brownian motion

Suppose we wish to simulate a sample path of the geometric brownian motion process for the stock price. We divide the time interval T into n equal steps and simulate the path, starting at the known state(initial price) S_0 at time t_0. The stock price changes according to:

$$S_{i+1} = S_i \exp\left[\mu\Delta t + \sigma\sqrt{\Delta t} \cdot dz_i\right]$$

To generate sample paths, we need to generate a sequence standard normal deviates $[dz_1, dz_2, \ldots, dz_n]$. It would be nice to get a standard normal from a standard uniform by inverting the distribution function, but there is no closed form formula for the distribution function $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{x^2}{2}}$. An amazing trick here saves the day!

3.1.1 Generating standard normal deviates

Imagine throwing darts on a circular dart board with radius 1. A skilled player has a 1/2 chance of hitting the inner bullseye, 1/4 chance of the outer bullseye, 1/8 chance of the triple ring and so forth. The likelihood decreases exponentially, on moving away from the origin. Given a ring on the board, the dart equally likely to land at any angle.

Assume, the dart hits the point $P(\sqrt{2r}, \theta)$. Define a r.v. $\sqrt{2R}$ as the radial distance of the dart from the origin, Θ as the angle with x-axis. Then, R is exponential r.v. and Θ is a uniform r.v. The probability is radially symmetric. We have,

$$R \sim \text{Exponential}(1)$$

and

$$\Theta \sim \text{Uniform}(0, 2\pi)$$

Let's apply the transformation $X = \sqrt{2R}\cos\Theta$ and $Y = \sqrt{2R}\sin\Theta$. We see that,

$$X^2 + Y^2 = 2R(\cos^2\Theta + \sin^2\Theta)$$

and

$$\tan\Theta = \frac{Y}{X}$$

We've transformed the point P from polar co-ordinates $(\sqrt{2R}, \Theta)$ to rectangular co-ordinates (X, Y). Interestingly, we can prove that (X, Y) are gaussian.

The joint PDF of (R, Θ) is:

$$f_{R,\Theta}(r,\theta) = \frac{1}{2\pi}e^{-r}$$

We know that, $r = (x^2 + y^2)/2$, so the joint PDF of (X, Y) is

$$\begin{aligned} f_{X,Y}(x,y) &= \frac{1}{2\pi}e^{-\frac{x^2+y^2}{2}} \\ &= \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} \cdot \frac{1}{\sqrt{2\pi}}e^{-\frac{y^2}{2}} \end{aligned}$$

Thus, (X, Y) are gaussian.

3.1.2 Implementing the Box-Muller transform

Let's see a Python snippet that implements the Box-Muller transformation.

```
In [12]: import numpy as np
         import math
         import matplotlib.pyplot as plt

         def getGaussian(mu,sigma,n):

             # Generate a pair of uniform random variables U,V
             U = np.random.uniform(0,1,n)
             V = np.random.uniform(0,1,n)

             # Get R~Exponential(1) and Theta~Uniform(0,2pi) random variables
             R = -np.log(U)
             Theta = 2*math.pi*V

             # Transform to X and Y
             X = np.multiply(np.sqrt(2*R),np.cos(Theta))
             Y = np.multiply(np.sqrt(2*R),np.sin(Theta))

             # Perform location-scale transform
             X = mu + sigma*X
             Y = mu + sigma*Y

             return X,Y
```

3.1.3 Implementing GBM

Let us define a function to generate Monte Carlo paths for the geometric Brownian motion.

```
In [14]: import numpy as np
         import math

         def gbm_mcs(mu=0.05, sigma=0.10, S0=100, T=1, n=250, noOfPaths=1000):
             S = []
             delta_t = T/n

             # Mean and variance of a lognormal random walk
             mean = S0 * np.exp((mu+(0.5*sigma**2))*T)
             variance = (S0 ** 2)*np.exp(2*mu*T + 2*(sigma ** 2)*T) \
             *(np.exp(sigma**2)*T-1)

             for i in range(noOfPaths):
                 X,Y = getGaussian(0,1,n)
                 dz_t = X
                 dW_t = math.sqrt(delta_t) * dz_t
                 dX_t =  mu * delta_t + sigma * dW_t
                 X_t = np.cumsum(dX_t)
                 S_t = S0 * np.exp(X_t)
                 S.append(S_t)

             S = np.array(S)
             return S, mean, variance
```

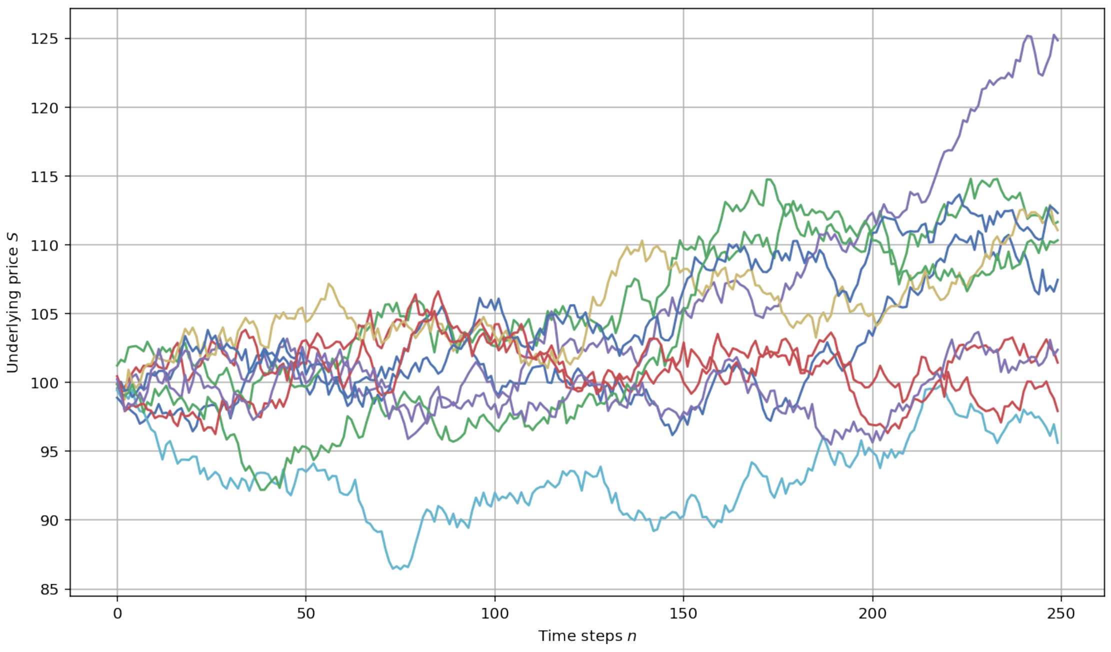
The following is a possible parametrization for the Monte-Carlo simulation, generating 1000 paths, with 250 time steps each:

```
In [15]: S0 = 100
         r = 0.06
         sigma = 0.10
         T = 1
         n = 250
         noOfPaths = 1000
         mu = r-(0.5*sigma**2)

         # Simulate a lognormal random walk
         S,mean,variance = gbm_mcs(mu,sigma,S0,T,n,noOfPaths)
```

The first 10 simulated paths from the simulation:

```
In [16]: time = list(range(250))
         for i in range(10):
             plt.plot(time,S[i])

         plt.xlabel('Time steps $n$')
         plt.ylabel('Underlying price $S$')
         plt.grid(True)
         plt.show()
```

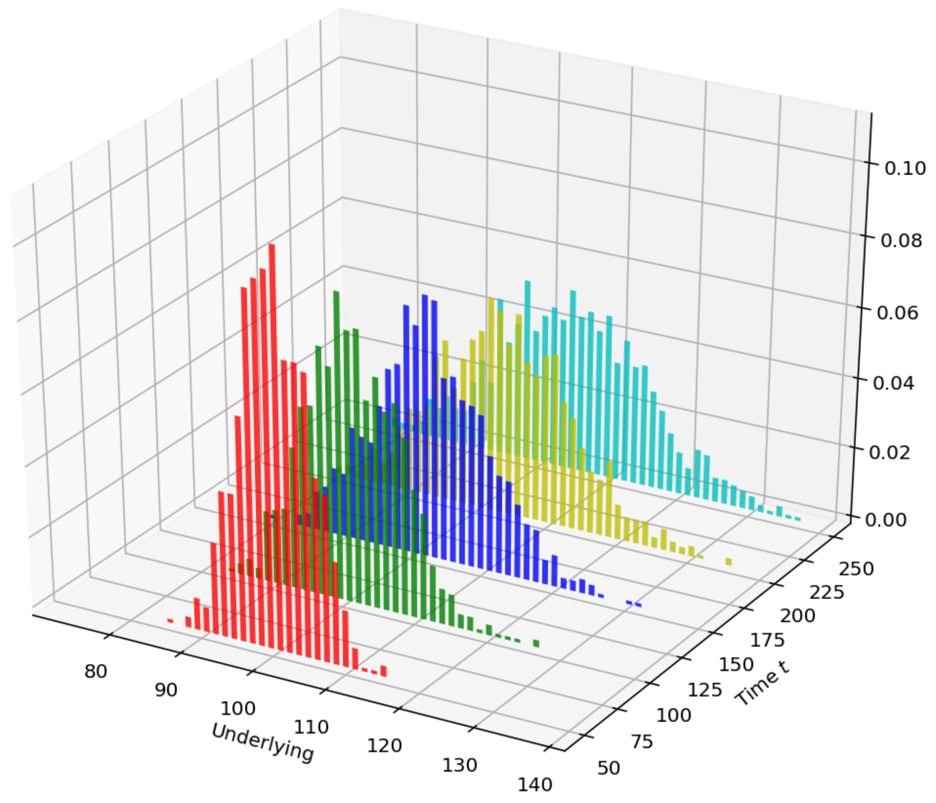
Out[16]:



Our main interest is in the distribution of stock prices that are lognormal with parameters μT and $\sigma\sqrt{T}$.

In [17]:
```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

sd = math.sqrt(variance)

# Density plot of the underlying price
fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111,projection='3d')

x = np.linspace(mean-3*sd,mean+3*sd,51)
time = list(range(250))

for i,c in zip([50,100,150,200,250],['r','g','b','y','c']):
    hist,bins = np.histogram(S[:,i-1],bins = x)
    hist = hist/noOfPaths
    cs = [c] * 50
    ax.bar(x[:-1],hist,i,zdir='y',color=cs,alpha=0.8)

ax.set_xlabel('Underlying')
ax.set_ylabel('Time $t$')
plt.show()
```

Out[17]:



3.2 Simulating correlated brownian motion

3.2.1 Spread options

Spread Trading As in all markets, there are many ways to trade profitably. One type of trading strategy, common among floor traders is scalping. A scalper attempts to buy at the bid price and sell at the offer price as often as possible.

A different type of trading strategy involves speculating on the direction in which the underlying contract will move. If a speculator correctly anticipates the market direction, and takes a position, he can also be expected to show profit.

The majority of successful option traders engage in *spread* trading. Spreading is simply a way of enabling a futures or options trader to take advantage of mispriced instruments. A spread trader takes a simultaneuous but opposing positions in different instruments.

Curve spreads. Say right now, 5 year swaps are trading at 1.50% and 10 year swaps are trading 2.00%. The spread between them aka the slope is +50 bps. If a trader takes a view that spread will widen, he should pay the fixed rate on the 10 year swap and should receive on a 5 year swap. Buy the ten year swap and short an equal number of DV01s of the 5 year swap. Essentially, you are exposed to the slope, you are long the curve.

We always talk in terms of the longest swap in the spread. If a trader is buying(selling), lifting(hitting), putting on a steepener(flattener), he is paying(receiving) fixed on the longest swap. And doing the opposite on the shortest leg.

What if the trader has a view on the shape rather than the steepness of the curve? A butterfly has three legs. The middle leg generally moves in opposite direction to the short and long legs. The middle leg is called the belly of the fly and the other two legs are called wings. There is a double weighting for the intermediate swap rate.

$$Price = 2S_{(0,T2)} - (S_{(0,T1)} + S_{(0,T3)})$$

We always talk in terms of the belly swap of the fly. If the trader is buying(selling), lifting(hitting), taking(giving) then they are paying(receiving) fixed on the belly leg. And doing the opposite on the wings.

US Dollar Swap spreads. Any swap trader will make or lose money as the swap rates go up or down. But, what if he doesn't have an opinion on whether the rates will go up or down? Perhaps, he thinks that the credit-worthiness of the financial industry will change relative to sovereign debt, or maybe he thinks that the markets a poised for a flight to quality. Consider a trader who believes that the US-Dollar swap spread will narrow. Then, he should receive the fixed rate on the ten-year swap and sell an equal number of DV01s of the 10 year government bond. If he feels that 10 year swap rates will go up by more than 10 year government bond yields, he should pay the fixed rate on the ten year swap and buy an equal number of DV01s of the 10 year government bonds.

Crack spreads. Benchmark oils are used as references when pricing oil. Western Texas Intermediate(WTI) is a light sweet North-American crude oil, which is traded for physical delivery to the transshipment point of Cushing, West Oklahoma. Brent Crude is a blend of various deliverable from the North sea oil-fields - Brent, Forties, Oseberg and Ekofisk. Crude oil futures trade on NYMEX, ICE for delivery of 1000 barrels.

It is quite common for heavier petrochemicals to be cracked into more valuable lighter products such as gasoline and middle distillates(jet fuel, diesel, kerosene and heating oil). Gasoline traded in markets such as NYMEX is a blendstock which needs to have an oxygenating product added to it to raise the octane number before it can be used in internal combustion engines. This gasoline blendstock is denoted by RBOB (Reformulated Gasoline Blendstock for Oxygenate Blending). Both RBOB and heating oil futures trade on the NYMEX.

It is relatively common to trade the *crack spread*. The gasoline crack spread for example, is the price differential between 1 barrel of RBOB gasoline and 1 barrel of WTI crude oil. A more typical crack spread is the 3 : 2 : 1 crack. For every 3 barrels of crude oil the refinery processes, it makes 2 barrels of gasoline and 1 barrel of heating oil. A refinery is naturally long 2 barrels of gasoline and 1 barrel of heating oil and short 3 barrels of crude oil. Crack spread futures and options trade widely on the New York Mercantile Exchange(NYMEX) and the Intercontinental Exchange (ICE).

This code snippet shows the WTI-Brent crude spread.

```
In [18]: import quandl
         import matplotlib.pyplot as plt
         import pandas as pd

         brentColor = np.array([31, 119, 180])/255
```

```
WTIColor = np.array([174, 199, 232])/255


startdate = '2012-04-01'
enddate = '2017-04-30'

BrentCrude = quandl.get("COM/WLD_CRUDE_BRENT", start_date=startdate)
WTICrude = quandl.get("COM/WLD_CRUDE_WTI",start_date=startdate)

mydates = pd.date_range(startdate, enddate, freq='M').tolist()

plt.figure(figsize=(8,5))
plt.plot(mydates,BrentCrude.iloc[:],color=brentColor,label='Brent/BZ')
plt.plot(mydates,WTICrude.iloc[:],color=WTIColor,label='WTI/CL')
plt.xlabel('Year')
plt.ylabel("US Dollars/barrel")
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```

```
    ---------------------------------------------------------------------------

    KeyboardInterrupt                         Traceback (most recent call last)

    <ipython-input-18-361ecce03833> in <module>()
     10 enddate = '2017-04-30'
     11 
---> 12 BrentCrude = quandl.get("COM/WLD_CRUDE_BRENT", start_date=startdate)
     13 WTICrude = quandl.get("COM/WLD_CRUDE_WTI",start_date=startdate)
     14 


    /projects/anaconda3/lib/python3.5/site-packages/quandl/get.py in get(dataset, **kwargs)
     46         if dataset_args['column_index'] is not None:
     47             kwargs.update({'column_index': dataset_args['column_index']})
---> 48         data = Dataset(dataset_args['code']).data(params=kwargs, handle_column_not_f
     49     # Array
     50     elif isinstance(dataset, list):


    /projects/anaconda3/lib/python3.5/site-packages/quandl/model/dataset.py in data(self, **
     45         updated_options = Util.merge_options('params', params, **options)
     46         try:
---> 47             return Data.all(**updated_options)
     48         except NotFoundError:
     49             if handle_not_found_error:
```

```
/projects/anaconda3/lib/python3.5/site-packages/quandl/operations/list.py in all(cls, **
 12             options['params'] = {}
 13         path = Util.constructed_path(cls.list_path(), options['params'])
---> 14         r = Connection.request('get', path, **options)
 15         response_data = r.json()
 16         Util.convert_to_dates(response_data)


/projects/anaconda3/lib/python3.5/site-packages/quandl/connection.py in request(cls, htt
 34         abs_url = '%s/%s' % (ApiConfig.api_base, url)
 35
---> 36         return cls.execute_request(http_verb, abs_url, **options)
 37
 38     @classmethod


/projects/anaconda3/lib/python3.5/site-packages/quandl/connection.py in execute_request(
 40         try:
 41             func = getattr(requests, http_verb)
---> 42             response = func(url, **options)
 43             if response.status_code < 200 or response.status_code >= 300:
 44                 cls.handle_api_error(response)


/projects/anaconda3/lib/python3.5/site-packages/requests/api.py in get(url, params, **kw
 68
 69     kwargs.setdefault('allow_redirects', True)
---> 70     return request('get', url, params=params, **kwargs)
 71
 72


/projects/anaconda3/lib/python3.5/site-packages/requests/api.py in request(method, url,
 54     # cases, and look like a memory leak in others.
 55     with sessions.Session() as session:
---> 56         return session.request(method=method, url=url, **kwargs)
 57
 58


/projects/anaconda3/lib/python3.5/site-packages/requests/sessions.py in request(self, me
486         }
487         send_kwargs.update(settings)
--> 488         resp = self.send(prep, **send_kwargs)
489
490         return resp
```

```
/projects/anaconda3/lib/python3.5/site-packages/requests/sessions.py in send(self, reque
    607
    608         # Send the request
--> 609         r = adapter.send(request, **kwargs)
    610
    611         # Total elapsed time of the request (approximately)


/projects/anaconda3/lib/python3.5/site-packages/requests/adapters.py in send(self, reque
    421                     decode_content=False,
    422                     retries=self.max_retries,
--> 423                     timeout=timeout
    424                 )
    425


/projects/anaconda3/lib/python3.5/site-packages/requests/packages/urllib3/connectionpool
    598                                                   timeout=timeout_obj,
    599                                                   body=body, headers=headers,
--> 600                                                   chunked=chunked)
    601
    602             # If we're going to release the connection in ``finally:``, then


/projects/anaconda3/lib/python3.5/site-packages/requests/packages/urllib3/connectionpool
    343         # Trigger any extra validation we need to do.
    344         try:
--> 345             self._validate_conn(conn)
    346         except (SocketTimeout, BaseSSLError) as e:
    347             # Py2 raises this as a BaseSSLError, Py3 raises it as socket timeout.


/projects/anaconda3/lib/python3.5/site-packages/requests/packages/urllib3/connectionpool
    842         # Force connect early to allow us to validate the connection.
    843         if not getattr(conn, 'sock', None):  # AppEngine might not have  `.sock`
--> 844             conn.connect()
    845
    846         if not conn.is_verified:


/projects/anaconda3/lib/python3.5/site-packages/requests/packages/urllib3/connection.py
    282     def connect(self):
    283         # Add certificate verification
--> 284         conn = self._new_conn()
    285
    286         hostname = self.host
    /projects/anaconda3/lib/python3.5/site-packages/requests/packages/urllib3/connection.py
    139         try:
    140             conn = connection.create_connection(
--> 141                 (self.host, self.port), self.timeout, **extra_kw)
    142
    143         except SocketTimeout as e:


    /projects/anaconda3/lib/python3.5/site-packages/requests/packages/urllib3/util/connectio
     71             if source_address:
     72                 sock.bind(source_address)
---> 73             sock.connect(sa)
     74             return sock
     75


    KeyboardInterrupt:
```

3.2.2 Generating correlated normal deviates

Joint distribution. The distribution of a single r.v. X provides complete information about the probability of X falling into any subset of the real line $\mathbb{R}$. Analogously, the joint distribution of two r.v.s X and Y provides complete information about the probability of the random vector (X, Y) falling into any subset of the real plane $\mathbb{R}$.

Covariance. Just as mean and variance provide single-number summaries of the distribution of a single r.v., covariance is a single number summary of the joint distribution of two r.v.s. Roughly speaking, covariance measures a tendency of the two r.v.s to go up or down together, relative to their expected values: positive covariance between X and Y indicates that when X goes up, Y also tends to go up, and negative covariance indicates that when X goes up, Y tends to go down.

The covariance between r.v.s X and Y is:

$$Cov(X, Y) = E[(X - EX)(Y - EY)]$$

Multivariate normal. A random vector $X = (X_1, \ldots, X_k$ is said to have a *multi-variate* normal distribution if every linear combination of the X_j has a normal distribution. That is, we require

$$t_1 X_1 + \ldots + t_k X_k$$

to have a normal distribution for any choice of the constants $t_1, \ldots, t_k$. An important special case is $k = 2$; this distribution is called the *Bivariate Normal*(BVN).

If $(X_1, X_2, \ldots, X_k)$ is MVN, then the marginal distribution of X_1 is normal, since we can take t_1 to be one and all other t_j to be zero. Similarly, the marginal distribution of each X_j is normal. However, the converse is false: it is possible to have normally distributed r.v.s such that $(X_1, X_2, \ldots, X_k)$ is not multivariate normal.

Example.(Non-example of MVN). Here is an example of two r.v.s whose marginal distributions are normal but whose joint distribution is not bivariate normal. Let $X \sim \Phi(0,1)$ and let $S = 1$ with probability $1/2$, $S = -1$ with probability $1/2$ be a random *sign* independent of X. Then, $Y = SX$ is a standard normal random variable, due to the symmetry of the normal distribution. However, (X, Y) is not bivariate normal because $P(X + Y = 0) = P(S = -1) = 1/2$, which implies that $X + Y$ can't be gaussian(or for that matter, have any continuous distribution). Since, $X + Y$ is a linear combination of X and Y that is not normally distributed, (X, Y) is not bivariate normal.

Example.(Actual MVN). For $Z, W \sim \Phi(0,1)$, (Z, W) is bivariate normal because the sum of independent Normals is Normal. Also, $(Z + 2W, 3Z + 5W)$ is bivariate normal, since an arbitrary linear combination

$$t_1(Z + 2W) + t_2(3Z + 5W)$$

can also be written as a linear combination of Z and W,

$$(t_1 + 3t_2)Z + (2t_1 + 5t_2)W$$

,

which is normal.

A Multivariate normal distribution is fully specified by knowing the mean of each component, the variance of each component and the covariance or the correlation between any two components. Another way to say this is that the parameters of an MVN random vector $(X_1, \dots, X_k)$ are as follows :

1. The mean vector $(\mu_1, \dots, .\mu_k)$, where $E(X_j) = \mu_j$
2. The covariance matrix, which is the $k \times k$ matrix of covariances between components, arranged so that the row i, column j entry is $Cov(X_i, X_j)$

For example, in order to fully specify a Bivariate normal distribution for (X, Y), we need to know the five parameters:

1. The means $E(X)$,$E(Y)$.
2. The variances $Var(X)$, $Var(Y)$.
3. The correlation $Corr(X, Y)$.

Theorem. The joint PDF of a Bivariate normal (Z, W) with $\Phi(0,1)$ marginal distributions and correlation $\rho \in (1, 1)$ is

$$f_{Z,W}(z, w) = \frac{1}{2\pi\tau} \exp\left(-\frac{1}{2\tau^2}(z^2 + w^2 - 2\rho zw)\right)$$

Proof. Let (Z, W) be BVN with $\Phi(0,1)$ marginals and $Corr(Z, W) = \rho$. As shown in the next example, we can construct (Z, W) as

$$Z = X$$

$$W = \rho X + \tau Y$$

with $\tau = \sqrt{1 - \rho^2}$ and X, Y are i.i.d $\Phi(0,1)$. We also need the inverse transformation. Solving $Z = X$ for X, we have $X = Z$. Plugging this into $W = \rho X + \tau Y$ and solving for Y, we get

$$X = Z$$

$$Y = -\frac{\rho}{\tau} Z + \frac{1}{\tau} W$$

The Jacobian is

$$\begin{aligned}\frac{\partial(x,y)}{\partial(z,w)} &= \begin{pmatrix} \frac{\partial x}{\partial z} & \frac{\partial x}{\partial w} \\ \frac{\partial y}{\partial z} & \frac{\partial y}{\partial w} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ -\frac{\rho}{\tau} & \frac{1}{\tau} \end{pmatrix}\end{aligned}$$

which has absolute determinant $1/\tau$. So, by change of variables formula,

$$\begin{aligned} f_{Z,W}(z,w) &= f_{X,Y}(x,y)\frac{\partial(x,y)}{\partial(z,w)} \\ &= \frac{1}{2\pi\tau}\exp\left(-\frac{1}{2}(x^2+y^2)\right) \\ &= \frac{1}{2\pi\tau}\exp\left(-\frac{1}{2}(z^2+(-\frac{\rho}{\tau}z+\frac{1}{\tau}w)^2)\right) \\ &= \frac{1}{2\pi\tau}\exp\left(-\frac{1}{2\tau^2}(z^2(1-\rho^2)+\rho^2z^2+w^2-2\rho zw)\right) \\ &= \frac{1}{2\pi\tau}\exp\left(-\frac{1}{2\tau^2}(z^2+w^2-2\rho zw)\right) \end{aligned}$$

Bivariate normal generation. Suppose that we have access to i.i.d r.v.s $X, Y \sim \Phi(0,1)$, but we want to generate a Bivariate normal (Z, W) with $Corr(Z,W) = \rho$ and Z, W marginally $\Phi(0,1)$, for the purpose of running a simulation. How can we construct Z and W from linear combinations of X and Y?

Solution.

By definition of multivariate normal, any (Z, W) of the form

$$Z = aX + bY$$

$$W = cX + dY$$

will be bivariate normal. So, let's try to find a suitable a, b, c, d. The means are already 0. Setting the variances equal to 1 gives,

$$a^2 + b^2 = 1$$

$$c^2 + d^2 = 1$$

The covariance of Z and W is,

$$\begin{aligned} Cov(Z,W) &= E[(Z-EZ)(W-EW)] \\ &= E(ZW) - E(Z)E(W) \\ &= E((aX+bY)(cX+dY)) - 0 \\ &= E(acX^2 + adXY + bcXY + bdY^2) \\ &= acE(X^2) + bdE(Y^2) + (ad+bc)E(XY) \\ &= ac(1) + bd(1) + (ad+bc)(0) \\ &= ac + bd \end{aligned}$$

Setting the covariance of Z and W equal to ρ yields,

$$ac + bd = \rho$$

There are three equations and four unknowns, and we need just one solution. To simply, let's look for a solution with $b = 0$. Then, $a^2 = 1$. Now, $ac + bd = \rho$ reduces to $c = \rho$ and then we can use $c^2 + d^2 = 1$ to find a suitable d. Putting everything together, we can generate (Z, W) as:

$$Z = X$$

$$W = \rho X + \sqrt{1-\rho^2} Y$$

3.3 Pricing a spread option in Python

```
In [19]: import numpy as np
         import math

         def genCorrelatedGaussianDeviates(rho,n):

             X,Y = getGaussian(0,1,n)
             Z = X
             W = rho*X + math.sqrt(1-rho**2)*Y
             return Z,W

         def computeMCEuroSpreadOption(S1,S2,K,r,sigma1,sigma2,
                                       rho,T,n,noOfPaths,optType):
             dt = T/n
             mu1 = r - 0.5 * (sigma1**2)
             mu2 = r - 0.5 * (sigma2**2)
             S1_paths = []
             S2_paths = []
             sum = 0
             disc = math.exp(-r*T)

             for i in range(noOfPaths):
                 dz1,dz2 = genCorrelatedGaussianDeviates(rho,n)
                 dB1 = math.sqrt(dt)*dz1
```

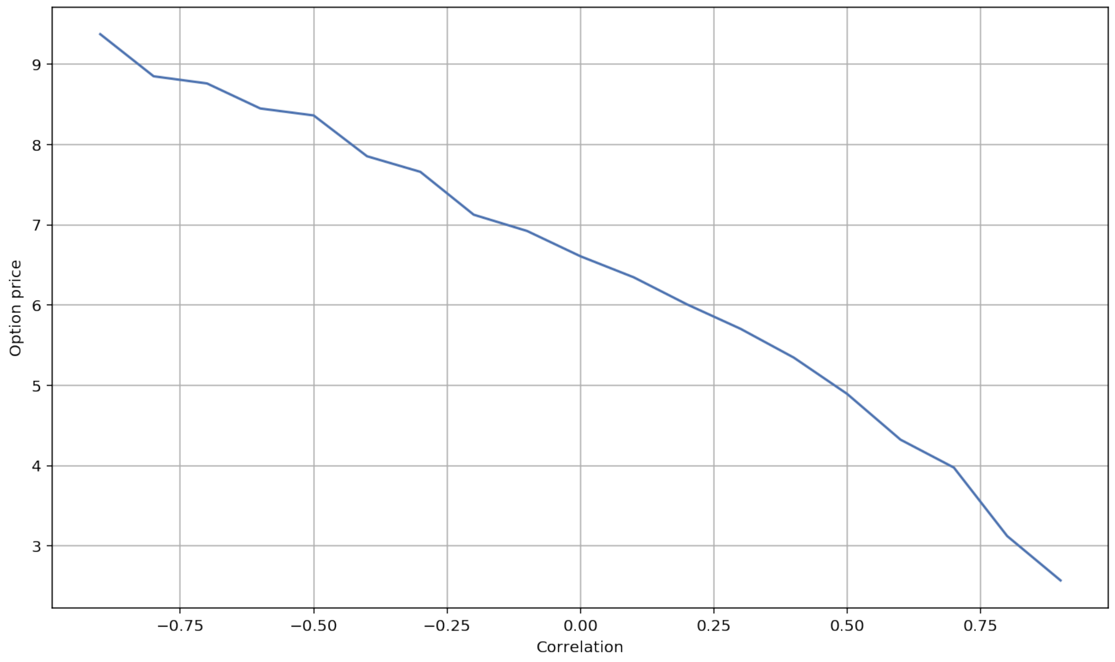
```
                dB2 = math.sqrt(dt)*dz2
                dX1 = mu1 * dt + sigma1 * dB1
                dX2 = mu2 * dt + sigma2 * dB2
                X1_t= np.cumsum(dX1)
                X2_t= np.cumsum(dX2)
                S1_t= S1 * np.exp(X1_t)
                S2_t= S2 * np.exp(X2_t)

                if (optType == 'C'):
                    value = max(S1_t[-1] - S2_t[-1] - K,0)
                elif (optType == 'P'):
                    value = max(K - S1_t[-1] - S2_t[-1],0)

                sum = sum + value

                S1_paths.append(S1_t)
                S2_paths.append(S2_t)

            S1_paths = np.array(S1_paths)
            S2_paths = np.array(S2_paths)
            C = disc * (sum/noOfPaths)

            return C,S1_paths,S2_paths

        C,S1_hist, S2_hist = computeMCEuroSpreadOption(50,50,1,0.06,0.30,0.20,
                                                       0.75,1,250,1000,'C')
        print(C)
```

```
3.78107571479
```

Notice that as the correlation increases, the option price (monotonically) decreases for both call and put spread options.

```
In [20]: import matplotlib.pyplot as plt

         correlations = np.arange(-0.90,1,0.10)
         callprices = np.array([])

         for rho in correlations:
             C,S1_histc, S2_histc = computeMCEuroSpreadOption(50,50,1,0.06,0.30,
                                                    0.20,rho,1,250,10000,'C')
             callprices = np.append(callprices,C)

         plt.plot(correlations,callprices)
         plt.xlabel('Correlation')
         plt.ylabel('Option price')
         plt.grid(True)
         plt.show()
```

```
Out[20]:
```



4 Smile pricing

4.1 The vol smile

The FX options market is one of the largest and most liquid OTC derivatives market in the world. The Black-Scholes assumption of *constant volatility* for all options is wrong.

In professional FX markets, options with different moneyness levels and maturities are usually priced with different volatilities. A *smile* is a set of *market implied volatilities* plotted as function of *the moneyness of the option*, holding maturity constant. The vol smile evolves over time as options approach expiration. So, professional FX traders construct a *vol surface*, where implied volatilies are plotted as a function of moneyness and maturities.

If the Black Scholes model was correct, then the vol surface would be flat.

4.2 Reasons for the vol smile

4.3 Bisection method to find the implied vol

The analytical Black-Scholes formula for European calls and puts is:

$$Call(\sigma) = DF_d(t)[F\Phi(d_+) - K\Phi(d_-)]$$

$$Put(\sigma) = DF_d(t)[F\Phi(-d_+) - K\Phi(-d_-)]$$

where

$$d_{\pm} = \frac{\ln F/K \pm \frac{\sigma^2}{2}}{\sigma\sqrt{t}}$$

The implied volatility $\sigma^{implied}$ is such a value of the volatility parameter σ that the Black Scholes price *matches* the observed market price:

$$C_{BS}(\sigma^{implied}) = C_{market}$$

Define $f(\sigma) = C_{BS}(\sigma) - C_{market}$. Any root finding algorithm e.g. the *bisection method* can be used to find the roots of the equation $f(\sigma) = 0$ is as follows:

1. Choose a subinterval $[\sigma_L, \sigma_H]$ for example, $[0.01, 1.00]$, and calculate the two option prices $C_{BS}(\sigma_L), C_{BS}(\sigma_H)$, such that $C_{BS}(\sigma_L) < C_{market}$ and $C_{BS}(\sigma_H) > C_{market}$.

2. Within this sub-interval, we know that the function $f(\sigma)$ is monotonically increasing, and at the end points $f(\sigma_L) < 0$ and $f(\sigma_H) > 0$. Since the function $f(\sigma)$ is continuous on the interval $[\sigma_L, \sigma_H]$, its graph will cut the x-axis in some one point between σ_L and σ_H.

3. Draw a chord AB connecting the end points of the curve $y = f(\sigma)$, which corresponds to the abscissas σ_L and σ_H. Then the abscissa σ_1 of the point of intersection of this chord with the x-axis will be the approximate root. In order to find this approximate value, let us write the equation of the straight line AB that passes through the given points $A[\sigma_L, f(\sigma_L)]$ and $B[\sigma_H, f(\sigma_H)]$:

$$\frac{y - f(\sigma_L)}{x - \sigma_L} = \frac{f(\sigma_H) - f(\sigma_L)}{\sigma_H - \sigma L}$$

Since, $y = 0$ at $x = \sigma_1$, we get:

$$\frac{-f(\sigma_L)}{\sigma_1 - \sigma_L} = \frac{f(\sigma_H) - f(\sigma_L)}{\sigma_H - \sigma L}$$

so,

$$\begin{aligned}\sigma_1 &= \sigma L - \frac{(\sigma_H - \sigma_L) f(\sigma_L)}{f(\sigma_H) - f(\sigma_L)} \\ &= \sigma_L - (C_{BS}(\sigma_L) - C_{market}) \frac{(\sigma_H - \sigma_L)}{C_{BS}(\sigma_H) - C_{BS}(\sigma_L)}\end{aligned}$$

4. To obtain a more exact value of the root, we determine $f(\sigma_1)$. If $f(\sigma_1) < 0$, then repeat the same procedure applying the bisection formula to the interval $[\sigma_1, \sigma_H]$. If $f(\sigma_1) > 0$, then apply this formula to the interval $[\sigma_L, \sigma_1]$. By repeating this procedure several times more, we obvious obtain more and more previse values of the root σ_2, σ_3 etc. This is continued until the desired accuracy is achieved:

$$C_{market} - C_{BS}(\sigma_i) < \epsilon$$

4.4 Implementation of the bisection method

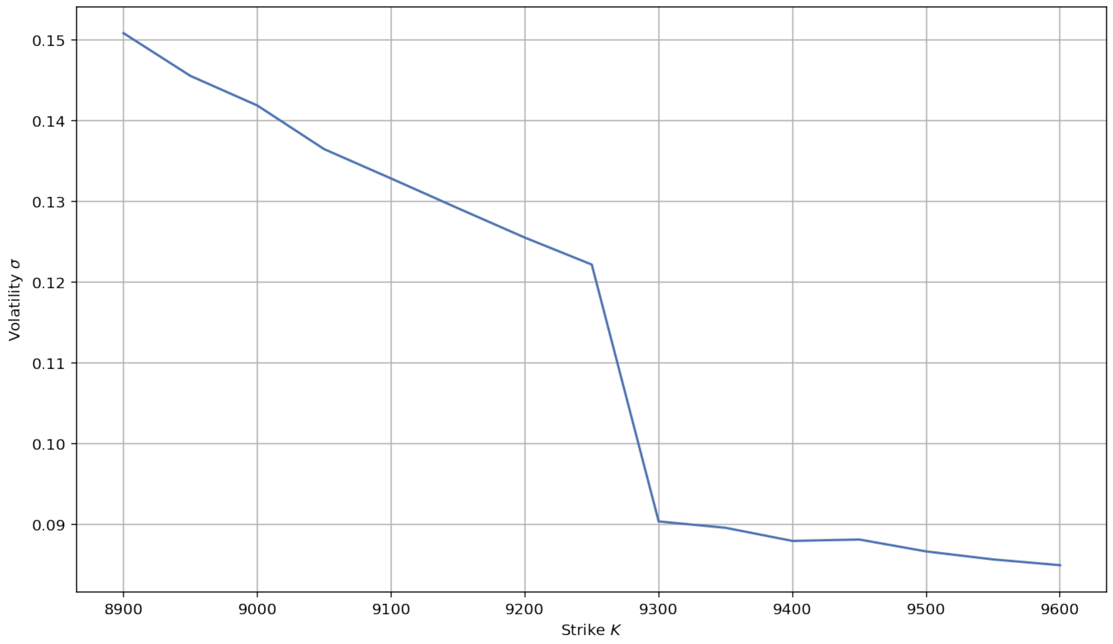
```
In [21]: import numpy as np
         import math
         import scipy.stats as stats

         def BS(S0,K,r,q,T,sigma,optType):
             dplus = (math.log(S0/K)+((r - q) + (sigma ** 2)/2)*T) \
             /(sigma * math.sqrt(T))

             dminus = (math.log(S0/K)+((r - q) - (sigma ** 2)/2)*T) \
             /(sigma * math.sqrt(T))

             discD = math.exp(-r*T)
             discF = math.exp(-q*T)

             if optType == 'C':
                 price = (S0 * discF * stats.norm(0,1).cdf(dplus)) \
                 - (K * discD * stats.norm(0,1).cdf(dminus))
             else:
                 price = -((S0 * discF * stats.norm(0,1).cdf(-dplus)) \
                           -(K * discD * stats.norm(0,1).cdf(-dminus)))

             return price

         def findImpliedVol(S0,K,r,q,T,optType,Call_market):

             # Parameters
             sigma_Lo = 0.01
             sigma_Hi = 1.00
             epsilon = 0.001

             Call_BS_Lo = BS(S0,K,r,q,T,sigma_Lo,optType)
             Call_BS_Hi = BS(S0,K,r,q,T,sigma_Hi,optType)
             Call_BS_i = Call_BS_Lo

             # Iteratively find a root using bisection method
             while abs(Call_BS_i - Call_market) > epsilon:
                 sigma_i = sigma_Lo \
                 -((Call_BS_Lo - Call_market)\
                   *((sigma_Hi - sigma_Lo)/(Call_BS_Hi - Call_BS_Lo)))

                 Call_BS_i = BS(S0,K,r,q,T,sigma_i,optType)

                 if Call_BS_i > Call_market:
                     Call_BS_Hi = Call_BS_i
                     sigma_Hi = sigma_i
                 else:
```

```
                Call_BS_Lo = Call_BS_i
                sigma_Lo = sigma_i

            return sigma_i
```

Equity indices have a volatility skew - puts have higher higher volatility then calls. The following code snippet plots the vol skew of May NIFTY options. NIFTY spot was at 9285.30 on 05th May, 2017.

```
In [22]: import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd

         d = {'call': pd.Series([420.875,379.45,328.325,277.775,240.475,\
                                 199.3,162.425,128.325,97.575,71.1,48.975, \
                                 33.2, 20.625, 12.2, 6.85],
                                index=[8900,8950,9000,9050,9100,9150,9200, \
                                       9250,9300,9350,9400,9450,9500,9550,9600]),
             'put':pd.Series([12.95,16,20.6,25.525,32.825,42,53.525,68.05,\
                              86.925,107.625,134.95,167.1,203.95,250.65,289.75],
                             index=[8900,8950,9000,9050,9100,9150,9200,\
                                    9250,9300,9350,9400,9450,9500,9550,9600])}

         optionPrices = pd.DataFrame(d)
         optionPrices

In [23]: iv = []

         # Find the IV of OTM puts
         for strike in range(8900,9300,50):
             vol = findImpliedVol(9285.30,strike,
                                  0.10,0,0.05479,'P',
                                  optionPrices['put'][strike])
             iv.append(vol)

         # Find the IV of OTM calls
         for strike in range(9300,9650,50):
             vol = findImpliedVol(9285.30,strike,
                                  0.10,0,0.05479,'C',
                                  optionPrices['call'][strike])
             iv.append(vol)

         # Plot the volatility skew as function of strikes
         strikes = list(range(8900,9650,50))
         plt.grid(True)
         plt.xlabel('Strike $K$')
         plt.ylabel('Volatility $\sigma$')
         plt.plot(strikes,iv)
         plt.show()
```

```
Out[23]:
```



4.5 Constructing the vol surface - Vanna Volga Method

In FX markets, there are only three active quotes for each maturity, the 0Δ straddle, the 25Δ risk reversals and the 25Δ vega weighted butterfly. We know that,

$$\sigma_{ATM} = \frac{1}{2}(\sigma_{50\Delta C} + \sigma_{50\Delta P})$$

$$\sigma_{RR} = \sigma_{25\Delta C} - \sigma_{25\Delta P}$$

$$\sigma_{BF} = -\sigma_{ATM} + \frac{1}{2}(\sigma_{25\Delta C} + \sigma_{25\Delta P})$$

Solving for the 25Δ-call and 25Δ-put volatilities, we have:

$$\sigma_{25\Delta C} = \sigma_{RR} + \sigma_{BF} + \frac{\sigma_{ATM}}{2}$$

$$\sigma_{25\Delta P} = \sigma_{RR} + \sigma_{BF} - \frac{\sigma_{ATM}}{2}$$

The other implied volatilities need to be determined. FX brokers and traders use a fast method - the *Vanna-Volga* (VV) also known as the *trader's rule of thumb* to construct the whole smile for a given maturity.

In a nutshell, the vanna-volga(VV) method is based on adding an analytically derived correction to the Black-Scholes price of the instrument. The method constructs a hedging portfolio that zeroes out the Black-Scholes greeks that measure the option's sensitivity with respect to the volatility i.e. the vega, vanna and volga of the option. In this way, VV method produces smile-consistent values.

The strikes corresponding to ATM, 25Δ call and 25Δ put can be obtained as:

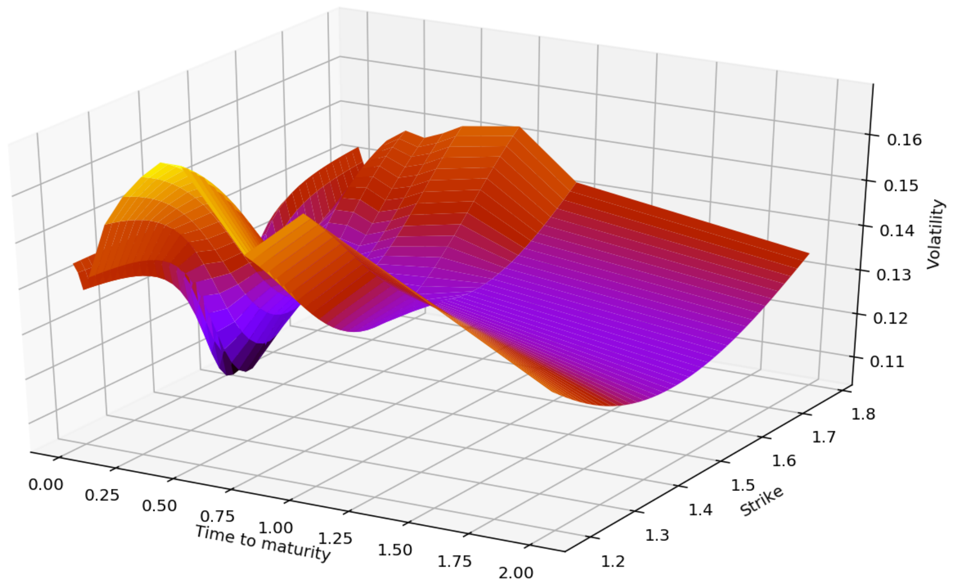
$$\begin{aligned}
D_f\Phi(d_+) + D_f(\Phi(d_+) - 1) &= 0 \\
\Phi(d_+) &= 0.50 \\
d_+ &= 0 \\
\frac{\log\left(\frac{F}{K_{atm}}\right) + \sigma_{atm}^2 t}{\sigma_{atm}\sqrt{t}} &= 0 \\
K_{atm} &= F\exp\frac{1}{2}\sigma_{atm}^2 t
\end{aligned}$$

```
In [24]: import pandas as pd
         import numpy as np
         import math
         import scipy.stats as sct
         import matplotlib.pyplot as plt
         from mpl_toolkits.mplot3d import Axes3D
         from matplotlib import cm

         # Market quotes for maturities ranging from 1 week to 2 years
         T = np.array([0.0192,0.0384,0.0575,0.0833,0.1667,0.2500,
                       0.3333,0.4167,0.5000,0.7500,1.0000,2.0000 ])
         n = len(T)
         sigma25DeltaPut=np.array([0.1170,0.1121,0.1068,0.1130,0.1210,0.1283,
                          0.1341,0.1372,0.1369,0.1263,0.1396,0.1234])
         sigmaATM=np.array([0.1105,0.1100,0.1048,0.1060,0.1130,0.1203,
                    0.1247,0.1270,0.1273,0.1238,0.1300,0.1213])
         sigma25DeltaCall=np.array([0.1090,0.1129,0.1088,0.1050,0.1130,0.1203,
                           0.1237,0.1265,0.1286,0.1328,0.1314,0.1291])
         r = np.array([0.0023,0.0023,0.0024,0.0024,0.0026,0.0028,
              0.0036,0.0044,0.0055,0.0089,0.0119,0.0124])
         q = np.array([0.0027,0.0027,0.0028,0.0032,0.0038,0.0042,
              0.0050,0.0060,0.0073,0.0110,0.0141,0.0139])

         Dd = np.exp(-np.multiply(r,T))
         Df = np.exp(-np.multiply(q,T))
         mu = r - q

         S0 = 1.4844
         delta = 0.25
         F = S0*np.exp(np.multiply(mu,T))
         alpha = -sct.norm.ppf(delta*np.reciprocal(Df))
         K25DCall = np.array([])
         KATM = np.array([])
         K25DPut = np.array([])
```

```python
for i in range(12):
    K1 = F[i]*math.exp(-(alpha[i]*sigma25DeltaPut[i]*math.sqrt(T[i]))+0.5*(sigma25Delta
    K25DPut = np.append(K25DPut,K1)
    K2 = F[i]*math.exp(0.5*(sigmaATM[i]**2)*T[i])
    KATM = np.append(KATM,K2)
    K3 = F[i]*math.exp((alpha[i]*sigma25DeltaCall[i]*math.sqrt(T[i]))+0.5*(sigma25Delta
    K25DCall = np.append(K25DCall,K3)

def d1(F,K,sigma,t):
    dplus = (math.log(F/K) + 0.5 *(sigma**2)*t)/(sigma*math.sqrt(t))
    return dplus

def d2(F,K,sigma,t):
    dminus = d1(F,K,sigma,t)-sigma*math.sqrt(t)
    return dminus

def VannaVolgaImpliedVol(F,K,t,K1,K2,K3,
              sigma1,sigma2,sigma3):

    y1 = (math.log(K2/K) * math.log(K3/K))/(math.log(K2/K1) * math.log(K3/K1))
    y2 = (math.log(K/K1) * math.log(K3/K))/(math.log(K2/K1) * math.log(K3/K2))
    y3 = (math.log(K/K1) * math.log(K/K2))/(math.log(K3/K1) * math.log(K3/K2))

    P = (y1*sigma1 + y2*sigma2 + y3 *sigma3) - sigma2
    Q = y1 * d1(F,K1,sigma1,t) * d2(F,K1,sigma1,t) * ((sigma1-sigma2)**2) \
    + y2 * d1(F,K2,sigma2,t) * d2(F,K2,sigma2,t) * ((sigma2-sigma2)**2) \
    + y3 * d1(F,K3,sigma3,t) * d2(F,K3,sigma3,t) * ((sigma3-sigma2)**2)

    d1d2 = d1(F,K,sigma2,t) * d2(F,K,sigma2,t)

    sigma = sigma2 + (-sigma2 + math.sqrt(sigma2**2 + d1d2 *(2*sigma2*P+Q)))/(d1d2)

    return sigma

strike = (1+np.arange(-0.20,0.21,0.01))*S0
VVImpliedVol = np.zeros((41,12),dtype=float)

for i in range(41):
    for j in range(12):
        VVImpliedVol[i][j]=VannaVolgaImpliedVol(F[j],strike[i],T[j],
                                        K25DPut[j],KATM[j],
                                        K25DCall[j],sigma25DeltaPut[j],
                                        sigmaATM[j],sigma25DeltaCall[j])

X = T
Y = strike
X, Y = np.meshgrid(X,Y)
```

```
Z = VVImpliedVol

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.set_xlabel('Time to maturity')
ax.set_ylabel('Strike')
ax.set_zlabel('Volatility')
surf = ax.plot_surface(X, Y, Z, cmap=cm.gnuplot,linewidth=0)
plt.show()
```

Out[24]:



5 Appendix A - Ito's Lemma

5.1 Simple Ito Formula

The Taylor's series polynomial that approximates the change in the (deterministic) function f, for a small change dx in the argument x is,

$$df(x) = f'(x)(dx) + \frac{1}{2!}f''(x)(dx)^2 + \frac{1}{3!}f'''(x)(dx)^3 + \frac{1}{4!}f^{(4)}(x)(dx)^4 + \dots$$

Let us now apply Taylor's formula to Brownian motion, taking

$$dB_t = B_{t+dt} - B_t$$

we have

$$df(B_t) = f'(B_t)(dB_t) + \frac{1}{2}f''(B_t)(dB_t)^2 + \frac{1}{3!}f'''(B_t)(dB_t)^3 + \frac{1}{4!}f^{(4)}(B_t)(dB_t)^4 + \dots$$

In the construction of Brownian motion, its small increment $dB_t = \pm\sqrt{dt}$ is a Bernoulli random variable. It turns out that the terms in $(dt)^2$ and $dtdB_t = \pm(dt)^{3/2}$ can be neglected in the Taylor's formula at the first order of approximation in dt. However, the term in order two,

$$(dB_t)^2 = (\pm\sqrt{dt})^2 = dt$$

can no longer be neglected in front of dt.

For a function f, Taylor's formula written at the second order of Brownian motion reads

$$df(B_t) = f'(B_t)dB_t + \frac{1}{2}f''(B_t)dt$$

5.2 Ito's formula for Ito processes

The Ito's Lemma applies to processes of the form,

$$dX_t = v_t dt + u_t dB_t$$

Given $(t,x) \to f(t,x)$ a smooth function of two variables $\mathbb{R}_+ \times \mathbb{R}$, the little change in the function f of any Ito process, is given by

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

6 Appendix B - Black Scholes PDE

Let the risky asset process $(S_t)_{t\in\mathbb{R}_+}$ be defined as,

$$dS_t = rS_t dt + \sigma S_t dB_t$$

We are interested in how the asset price S_t evolves with time. Our aim is therefore, to solve the above stochastic differential equation. It can be solved by applying the Ito's formula to the Ito process $(S_t)_{t\in\mathbb{R}_+}$ with $v_t = rS_t$ and $u_t = \sigma S_t$, and by taking $f(S_t) = \log(S_t)$.

$$\begin{aligned} d(\log(S_t)) &= \frac{1}{S_t}dS_t + \frac{1}{2}\sigma^2 S^2 \cdot \left(\frac{-1}{S^2}\right)dt \\ &= rdt + \sigma dB_t - \frac{1}{2}\sigma^2 dt \\ &= \left(r - \frac{\sigma^2}{2}\right)dt + \sigma dB_t \end{aligned}$$

Hence,

$$\begin{aligned} \log(S_t) - \log(S_0) &= \int_0^t d(\log(S_r)) \\ &= \left(r - \frac{\sigma^2}{2}\right)\int_0^t dr + \sigma\int_0^t dB_r \\ &= \left(r - \frac{\sigma^2}{2}\right)t + \sigma B_t \end{aligned}$$

and

$$S_t = S_0 \exp\left(r - \frac{\sigma^2}{2}\right) t + \sigma B_t$$

Consider a derivative $y = f(t,x)$ of the risky asset x at time t. The underlying follows a geometric brownian motion with a drift as above. Suppose we construct a portfolio Π containing a short position in one option f and a certain number of shares Δ:

$$\Pi = \Delta x - f(t,x)$$

From Ito's Lemma, a change in the portfolio value is given by:

$$\begin{aligned}
d\Pi &= \Delta dx - df(t,x) \\
&= \Delta rxdt + \sigma xdB_t - \left(\frac{\partial f}{\partial t}(t,x)dt + \frac{\partial f}{\partial x}(t,x)dx + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 f}{\partial x^2}(t,x)dt\right) \\
&= \Delta rxdt + \sigma xdB_t - \left(\frac{\partial f}{\partial t}(t,x)dt + \frac{\partial f}{\partial x}(t,x)(rxdt + \sigma xdB_t) + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 f}{\partial x^2}(t,x)dt\right) \\
&= \left(rx\left(\Delta - \frac{\partial f}{\partial x}(t,x)\right) - \frac{\partial f}{\partial t}(t,x) - \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 f}{\partial x^2}(t,x)\right) dt + \sigma x\left(\Delta - \frac{\partial f}{\partial x}(t,x)\right) dB_t
\end{aligned}$$

Let's select the number of shares to hold (the hedge ratio) Δ so that:

$$\Delta = \frac{\partial f}{\partial x}$$

This selection makes our portfolio instantaneously riskless - the term with the brownian motion dB_t(risk) falls out of the expression. However, the portfolio is riskless only for an infinitesimal time period dt, since we have fixed our hedge ratio Δ. To keep the portfolio riskless through the next time period dt, we will need to rebalance - to change the delta to reflect the changing stock price.

Since our portfolio is now instantaneously riskless(over an infinitesimal time period dt), its rate of return must be equal to the risk-free rate r (otherwise, there is a clear arbitrage opportunity). The interest that accrues on our portfolio during an infinitesimal time period dt is:

$$d\Pi = r\Pi dt$$

Thus, we have :

$$\begin{aligned}
\left(\frac{\partial f}{\partial t}(t,x) + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 f}{\partial x^2}(t,x)\right) dt &= r\Pi(dt) \\
-\frac{\partial f}{\partial t}(t,x) - \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 f}{\partial x^2}(t,x) &= r(\Delta x - f(t,x)) \\
&= rx\frac{\partial f}{\partial x}(t,x) - rf(t,x) \\
\frac{\partial f}{\partial t}(t,x) + rx\frac{\partial f}{\partial x}(t,x) + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 f}{\partial x^2}(t,x) &= rf(t,x)
\end{aligned}$$

Consequently, the option price f must satisfy this partial differential equation. This is the Black-Scholes PDE. It is similar to the Fourier's law of heat conduction in Physics or the Navier Stokes's equation for the motion of a viscous fluid.

7 Appendix C - Geometric Brownian Motion

7.1 Introduction

Note that since BM can take on negative values, using it directly for modeling stock prices is questionable. Instead, we use a non-negative variation of BM called geometric Brownian motion $S(t)$, which is defined by

$$S_t = S_0 e^{X(t)}$$

where $X(t) = \mu t + \sigma B(t)$ is BM with drift μ.

7.2 Moment generating function of a normal r.v.

The moment generating function of a normal r.v. $X \sim N(\mu, \sigma^2)$ is given by,

$$M_X(s) = E(e^{sX}) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp sx \cdot \exp -\frac{(x-\mu)^2}{2\sigma^2} dx$$

We can evaluate the integral by completing the square:

$$\begin{aligned}
&\exp sx \cdot \exp -\frac{(x-\mu)^2}{2\sigma^2} \\
&= \exp \frac{2\sigma^2 sx}{2\sigma^2} \cdot \exp -\frac{x^2 - 2\mu x + \mu^2}{2\sigma^2} \\
&= \exp -\frac{x^2 - 2\mu x - 2\sigma^2 sx + \mu^2}{2\sigma^2} \\
&= \exp -\frac{x^2 - 2x(\mu + \sigma^2 s) + (\mu + \sigma^2 s)^2 - 2\mu\sigma^2 s - \sigma^4 s^2}{2\sigma^2} \\
&= \exp -\frac{(x - (\mu + \sigma^2 s))^2}{2\sigma^2} \exp \left(\mu s + \frac{\sigma^2 s^2}{2} \right)
\end{aligned}$$

Thus, we have:

$$\begin{aligned}
M_X(s) = E(e^{sX}) &= \exp \left(\mu s + \frac{\sigma^2 s^2}{2} \right) \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(x - (\mu + \sigma^2 s))^2}{2\sigma^2} dx \\
&= \exp \left(\mu s + \frac{\sigma^2 s^2}{2} \right)
\end{aligned}$$

7.3 Computing moments for Geometric BM

As described above, the MGF of a normal r.v. $X \sim N(\mu, \sigma^2)$ is given by,

$$M_X(s) = E(e^{sX}) = \exp \left(\mu s + \frac{\sigma^2 s^2}{2} \right)$$

Thus, for BM with drift since $X(t) \sim N(\mu t, \sigma^2 t)$,

$$M_{X(t)}(s) = E(e^{sX(t)}) = \exp \left(\mu t s + \frac{\sigma^2 t s^2}{2} \right)$$

This allows us to immediately compute moments and variance of geometric BM, by using the values $s = 1, 2$ and so on. For example,

$$E(S(t)) = E(S_0 e^{X(t)}) = S_0 e^{\left(\mu t + \frac{\sigma^2 t}{2}\right)}$$

$$E(S(t)^2) = E(S_0^2 e^{2X(t)}) = S_0^2 e^{\left(2\mu t + 2\sigma^2 t\right)}$$

$$Var(S(t)) = S_0^2 e^{(2\mu t + \sigma^2 t)}(e^{\sigma^2 t} - 1)$$

7.4 Convergence of the binomial random walk to the Geometric BM

The binomial lattice model that I used earlier is in fact an approximation to the geometric BM, and I would proceed to explain the details here.

Recall that for BLM, $S_t = S_0 Y_1 Y_2 \ldots Y_n$, where the Y_i are i.i.d r.v.s distributed as $P(Y = u) = p$ and $P(Y = d) = 1 - p$. The initial value S_0, the parameters d, u, and $0 < p < 1$ completely determine this model.

In the Black-Scholes model, the continuously compounded returns $X(t_1)$, $X(t_2) - X(t_1)$ are independent and normally distributed due to normal independent increments property of BM. Therefore, $e^X(t_1)$, $e^{X(t_2)-X(t_1)}$ are independent lognormal random variables.

Define $L_1 := S_1/S_0 = e^X(t_1)$, $L_2 := S_2/S_1 = e^{X(t_2)-X(t_1)}$ as pecentage changes in the stock price.

The percentage changes in the stock price (not the actual changes $S_i - S_{i-1}$) are independent and lognormal.

We can re-write the terminal stock price,

$$\begin{aligned} S_t &= S_0 e^{X(t)} \\ &= S_0 e^{X(t_0)} . e^{X(t_2)-X(t_1)} \ldots e^{X(t_n)-X(t_{n-1})} \\ &= S_0 L_1 L_2 \ldots L_n \end{aligned}$$

as a product of n lognormal random variables.

Thus, we can approximate geometric BM over the time interval $(0, t]$ by the binomial lattice model, if we approximate the lognormal L_i with the binomial Y_i. To do so, we will just match the mean and variance:

$$pu + (1-p)d = e^{r\Delta t}$$

and

$$pu^2 + (1-p)d^2 = e^{(2\mu\Delta t + 2\sigma^2 \Delta t)}$$

Solving the first equation for the probability p yields,

$$p = \frac{e^{r\Delta t} - d}{u - d}$$

Substitution of p in the second expression yields,

$$
\begin{aligned}
p(u^2 - d^2) + d^2 &= e^{(2\mu\Delta t + 2\sigma^2\Delta t)} \\
\frac{e^{r\Delta t} - d}{u - d}(u^2 - d^2) + d^2 &= e^{(2\mu\Delta t + 2\sigma^2\Delta t)} \\
(e^{r\Delta t} - d)(u + d) + d^2 &= e^{(2\mu + 2\sigma^2)\Delta t} \\
e^{r\Delta t}(u + d) - du &= e^{(2\mu + 2\sigma^2)\Delta t}
\end{aligned}
$$

To solve for u and d in terms of r and σ, we need an additional equation, since we have one equation in two unknowns. There are two choices for a second equation : (1) Cox-Ross-Rubinstein (CRR) approach (2) Jarrow-Rudd approach.

7.5 Coxx-Ross-Rubinstein binomial tree

Coxx, Ross and Rubinstein in their original model assumed the identity $ud = 1$, so $u = 1/d$. Substituting this identity for u and d into the equation, we get

$$
\begin{aligned}
e^{r\Delta t}\left(u + \frac{1}{u}\right) - 1 &= e^{(2\mu + 2\sigma^2)\Delta t} \\
(u^2 + 1) - ue^{-r\Delta t} &= ue^{(r+\sigma^2)\Delta t} \\
u^2 - u(e^{-r\Delta t} + e^{(r+\sigma^2)\Delta t}) + 1 &= 0 \\
u^2 - 2Au + 1 &= 0
\end{aligned}
$$

where,

$$A = \frac{1}{2}(e^{-r\Delta t} + e^{(r+\sigma^2)\Delta t})$$

The roots of this quadratic equation are as follows,

$$
\begin{aligned}
&(u - A)^2 - A^2 + 1 = 0 \\
&u = A \pm \sqrt{A^2 - 1}
\end{aligned}
$$

Thus, we have specified all the parameters u, d and p in terms of the risk-free rate r and the volatility σ.

The Maclaurin series expansion for exponential functions is:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Ignoring the terms of higher order than x, we get:

$$e^{(r+\sigma^2)\Delta t} = 1 + r\Delta t + \sigma^2\Delta t$$

$$e^{-r\Delta t} = 1 - r\Delta T$$

Therefore,

$$\begin{aligned}
A &= 1+\frac{\sigma^2\Delta t}{2}\\
A^2 &= 1+\sigma^2\Delta t+\frac{\sigma^4\Delta t^2}{4}\\
A^2 &\approx 1+\sigma^2\Delta t\\
A^2-1 &= \sigma^2\Delta t\\
\sqrt{A^2-1} &= \sigma\sqrt{\Delta t}\\
u = A+\sqrt{A^2-1} &= 1+\frac{\sigma^2\Delta t}{2}+\sigma\sqrt{\Delta t}\\
d = A-\sqrt{A^2-1} &= 1+\frac{\sigma^2\Delta t}{2}-\sigma\sqrt{\Delta t}
\end{aligned}$$

Ignoring terms of order Δt and higher, we have:

$$u = 1+\sigma\sqrt{\Delta t} = e^{+\sigma\sqrt{\Delta t}}$$

$$d = 1-\sigma\sqrt{\Delta t} = e^{-\sigma\sqrt{\Delta t}}$$

7.6 Jarrow Rudd binomial tree

8 Appendix B - Greek Letters

8.1 Delta

Delta Δ is the first derivative of the option price with respect to the underlying spot S. Delta is a measure of the moneyness of the option. Let's first summarize the following relationships.

$\Phi(-x) = 1-\Phi(x)$

Also,

$d_+ = \dfrac{\ln\frac{F}{K}+\frac{\sigma^2}{2}T}{\sigma\sqrt{T}}$

$d_- = \dfrac{\ln\frac{F}{K}-\frac{\sigma^2}{2}T}{\sigma\sqrt{T}}$

Squaring the above expressions and subtracting,

$d_+^2-d_-^2 = \dfrac{4\ln\frac{F}{K}\left(\frac{\sigma^2}{2}\right)T}{\sigma^2 T}$

$\Rightarrow \dfrac{d_+^2}{2}-\dfrac{d_-^2}{2} = \ln\dfrac{F}{K}$

$\Rightarrow -\dfrac{d_-^2}{2} = -\dfrac{d_+^2}{2}+\ln\dfrac{F}{K}$

$\Rightarrow \exp\left(-\dfrac{d_-^2}{2}\right) = \dfrac{F}{K}\exp\left(-\dfrac{d_+^2}{2}\right)$

$\Rightarrow \dfrac{1}{\sqrt{2\pi}}\exp\left(-\dfrac{d_-^2}{2}\right) = \dfrac{F}{K}\dfrac{1}{\sqrt{2\pi}}\exp\left(-\dfrac{d_+^2}{2}\right)$

$\Rightarrow \phi\left(d_-\right) = \dfrac{F}{K}\phi\left(d_+\right)$

Let's now find the derivatives of $\Phi(d_+)$ and $\Phi(d_-)$.

$$\frac{\partial \Phi(d_+)}{\partial d_+} = \phi(d_+) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{{d_+}^2}{2}\right)$$

$$\frac{\partial \Phi(d_-)}{\partial d_-} = \phi(d_-) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{{d_-}^2}{2}\right) = \frac{F}{K}\phi\left(d_+\right)$$

Moreover,

$$\frac{\partial d_+}{\partial S} = \frac{\partial d_-}{\partial S} = \frac{1}{S\sigma\sqrt{T}}$$

Using the above identities, we can easily derive the call and put delta's as:

$$\begin{aligned}
\Delta_C &= \frac{\partial C}{\partial S} \\
&= D_f\Phi(d_+) + D_f S\phi(d_+)\frac{\partial d_+}{\partial S} - D_d K\phi(d_-)\frac{\partial d_-}{\partial S} \\
&= D_f\Phi(d_+) + D_f S\phi(d_+)\frac{1}{S\sigma\sqrt{T}} - D_d K \cdot \frac{F}{K}\phi(d_+)\frac{1}{S\sigma\sqrt{T}} \\
&= D_f\Phi(d_+) + D_f S\phi(d_+)\frac{1}{S\sigma\sqrt{T}} - D_f S\phi(d_+)\frac{1}{S\sigma\sqrt{T}} \\
&= D_f\Phi(d_+)
\end{aligned}$$

$$\begin{aligned}
\Delta_P &= \frac{\partial P}{\partial S} \\
&= -\left(D_f\Phi(-d_+) - D_f S\phi(-d_+)\frac{\partial d_+}{\partial S} + D_d K\phi(-d_-)\frac{\partial d_-}{\partial S}\right) \\
&= -\left(D_f\Phi(-d_+) - D_f S\phi(-d_+)\frac{1}{S\sigma\sqrt{T}} + D_d K \cdot \frac{F}{K}\phi(-d_+)\frac{1}{S\sigma\sqrt{T}}\right) \\
&= -\left(D_f\Phi(-d_+) - D_f S\phi(-d_+)\frac{1}{S\sigma\sqrt{T}} + D_f S\phi(-d_+)\frac{1}{S\sigma\sqrt{T}}\right) \\
&= -D_f\Phi(-d_+) \\
&= D_f(1 - \Phi(d_+))
\end{aligned}$$

8.2 Theta

Theta Θ is the first derivative of the option price with respect to the initial time t. We have,

$$\Theta = -\frac{\partial C}{\partial t}$$

The derivatives of d_+ and d_- with respect to time t are:

$$
\begin{aligned}
d_+ &= \frac{\ln\left(\frac{S}{K}\right) + \left(r - q + \frac{\sigma^2}{2}\right) t}{\sigma\sqrt{t}} \\
&= \frac{1}{\sigma\sqrt{t}} \ln\left(\frac{S}{K}\right) + \left(\frac{r-q}{\sigma} + \frac{\sigma}{2}\right)\sqrt{t} \\
\frac{\partial d_+}{\partial t} &= \left(\frac{r-q}{\sigma} + \frac{\sigma}{2}\right)\frac{1}{2\sqrt{t}} - \frac{1}{2\sigma\sqrt{t^3}} \ln\left(\frac{S}{K}\right) \\
&= \frac{r-q}{2\sigma\sqrt{t}} + \frac{\sigma}{4\sqrt{t}} - \frac{1}{2\sigma\sqrt{t^3}} \ln\left(\frac{S}{K}\right) \\
\frac{\partial d_-}{\partial t} &= \frac{r-q}{2\sigma\sqrt{t}} - \frac{\sigma}{4\sqrt{t}} - \frac{1}{2\sigma\sqrt{t^3}} \ln\left(\frac{S}{K}\right)
\end{aligned}
$$

The theta can then be derived as,

$$
\begin{aligned}
\Theta_C &= -\frac{\partial C}{\partial T} \\
&= D_f r_f S\Phi(d_+) - D_f S\phi(d_+)\frac{\partial d_+}{\partial t} - r_d D_d K\Phi(d_-) + D_d K\phi(d_-)\frac{\partial d_-}{\partial t} \\
&= r_f D_f S\Phi(d_+) - D_f S\phi(d_+)\left(\frac{r-q}{2\sigma\sqrt{t}} + \frac{\sigma}{4\sqrt{t}} - \frac{1}{2\sigma\sqrt{t^3}}\ln(S/K)\right) \\
&- r_d D_d K\Phi(d_-) + D_f S\phi(d_+)\left(\frac{r-q}{2\sigma\sqrt{t}} - \frac{\sigma}{4\sqrt{t}} - \frac{1}{2\sigma\sqrt{t^3}}\ln(S/K)\right) \\
&= r_f D_f S\Phi(d_+) - r_d D_d K\Phi(d_-) - D_f S\phi(d_+)\frac{\sigma}{2\sqrt{t}}
\end{aligned}
$$