MIE1613: Stochastic Simulation
Course Project
Winter 2018

Jonathan Smith
MEng Candidate
Mechanical and Industrial Engineering
University of Toronto
Smithj70
999694521
jon.smith@mail.utoronto.ca

Apr 25, 2018

Table of Contents

Introduction

Modern investors rely on a wide variety of financial products to produce a desired investment outcome. One very broad set of these products is financial options. An option is a contract between a holder and an issuer allowing the holder to buy or sell an underlying asset at maturity according to their discretion. Due to the risky nature of the underlying assets, it is generally not possible to analytically determine a fair price for the contract. Instead, analysts rely on Monte-Carlo simulations to determine a fair price to pay for option contracts.

The value of such an option depends not only on the expected performance of the underlying asset, but on the specifics of the contract as well. There are two general types of options: call options and put options. A call option allows the option holder to purchase the underlying asset, while a put option allows the option holder to sell an underlying asset. There are additional terms which make each contract unique. Three of the most common option contracts are: European, American, Asian. The specifics of each are briefly described below:

European

The option holder may choose to exercise their option only at a predefined maturity date, T. If exercised, the holder may buy or sell the underlying asset from the issuer for a predetermined cost, known as the strike prices, K. This results in a payoff of $\max(0, S_T - K)$ for a call option or $\max(0, K - S_T)$ for a put option. Where S_T is the asset value at maturity.

Asian

Like the European option, the holder may choose to exercise their option only at a predefined maturity date for a predetermined strike price, but payoffs are calculated based on the average strike price from the option issue date to maturity: $\max(0, \overline{S_T} - K)$ for a call option or $\max(0, K - \overline{S_T})$ for a put option.

American

In the case of an American option, the holder may choose to exercise the option at any time before maturity. In this case the payoffs are $\max(0, S_E - K)$ for a call option or $\max(0, K - S_E)$ for a put option.

Investors may buy or sell these options resulting in opposite payoffs. Additionally, when determining an option's value, the time value of money must be accounted for. This means discounting any realized payoffs into present value using a pre-set risk-free rate.

Details on each pricing model are presented in a later section.

Problem Description

There are many existing models related to pricing an individual option, however, investors typically hold a wide variety of both stocks and options. These different financial products compound to produce portfolio payoffs which cannot be analytically predicted. Furthermore, even when a payoff trend can be predicted, such as in the case of owning a single European option, the likelihood of potential payoffs remains unknown.

The purpose of this project is to create a tool which predicts the possible outcomes, as well as the likelihood of outcomes for a portfolio of stocks and options. Features of the tool are as follows:

- Allow the user to specify any number of stocks, and accompanying parameters which may be held in a long or short position
- Allow the user to buy or sell any number of put or call options on an underlying asset
- Provide the fair market value for European, Asian, and American options
- Provide the probability distribution for portfolio payoffs at maturity

- Provide a sensitivity analysis of the portfolio price with respect to the expected return of an underlying asset

Model Description

A description of each option pricing model as well as the portfolio valuation and sensitivity is presented in the following section. See the attached code in the appendix for more detailed implementation.

Option Pricing

The Monte-Carlo pricing methodology relies on generating multiple possible scenarios, and relying on the law of large numbers to make the simulation converge to the correct value. In this case we are using a Geometric Random Walk to simulate the underlying asset value throughout the desired time period with estimated return and variance.

European Option

To simulate a European option price, n sample stock paths must be generated according to the expected return and standard deviation. The stock price at maturity is then used to calculate the option payoff and discounted to the present value:

$$C = e^{-rT}\max(0, S_T - K)$$

$$P = e^{-rT}\max(0, K - S_T)$$

The option price is can then be estimated as the mean option payoff across n replications, with an appropriate confidence interval.

The European option is unique because it has an analytical solution known as the Black-Scholes equation. In order to provide a baseline accuracy for the simulation, this equation is implemented as follows:

$$C(S,t) = \mathcal{N}(d_1) \cdot S - \mathcal{N}(d_2) \cdot K \cdot e^{-r(T-t)}$$
$$P(S,t) = \mathcal{N}(-d_2) \cdot K \cdot e^{-r(T-t)} - \mathcal{N}(-d_1) \cdot S$$

where,

$$d_1 = \frac{1}{\sigma \cdot \sqrt{T-t}}\left[\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right) \cdot (T-t)\right]$$
$$d_2 = d_1 - \sigma \cdot \sqrt{T-t}$$

and $\mathcal{N}(\cdot)$ is the cumulative normal distribution.

Asian Option

The Asian option is very similar to the European option, but instead of using the stock price at maturity, payoff is calculated based on the average stock price until maturity:

$$C = e^{-rT}\max(0, \overline{S_T} - K)$$

$$P = e^{-rT}\max(0, K - \overline{S_T})$$

As in the case of the European option, the option value is the mean payoff across n replications.

American Option

American options are a notoriously difficult product to price because, in addition to the randomness of the underlying asset, there is a game theoretic aspect introduced by the early exercise ability. Many methods have been proposed to price American options, but the most popular is the Least-Square Monte-Carlo method proposed by Longstaff & Schwartz (2001). This algorithm relies on backward induction in order to determine an implied holding value and exercise value at each time step for each path. It aggregates the information in all paths in order to determine an optimal decision at each time step of each path This algorithm is briefly summarized here:

- Generate n random asset paths from current state to expiry
- Determine the payoffs starting at the last time node, t_m and t_{m-1}

- Calculate implied holding value (HV) at t_{m-1} based on t_m payoffs
- Fit quadratic least squares regression across each simulation: $HV = a + bS + cS^2$, representing an estimated holding value E[HV] at time t_{m-1}
- If E[HV] at t_{m-1} is greater than the payoff, overwrite the payoff with E[HV]
- Repeat for all time steps until t_0, then average the values

This code is implemented with iteration instead of recursion for programming simplicity.

Portfolio Valuation

The code utilizes object oriented programming to keep track of products added to the portfolio. When the user buys or shorts a stock or option, the cost of that product at time 0 is calculated as described above and tracked by the portfolio object. The portfolio then simulates potential payoffs of each product and discounts them back to present. The distribution of net portfolio value can then be determined for each replication as:

$$Net\ Payoff = Stock\ Payoff + Put\ Payoff + Call\ Payoff - Portfolio\ Cost$$

Payoff and cost variables are updated each time the user adds a new financial product. The portfolio object also creates plots of payoffs with respect to underlying stock prices and histograms of returns to model the probability of each payoff.

Portfolio Sensitivity

The portfolio also enables a sensitivity analysis. The allows the user to input an alternative estimate for the return on a single stock, and returns the alternate portfolio payoff along with an estimate of the derivative of portfolio value with respect to changes in stock value.

Results

The results of several analyses are shown and discussed below. For complete results please refer to files labeled RESULTS at https://github.com/jonsmith359/MIE1613_Project. Also, note that while the results shown below are for particular stock/option combinations, the same procedures may be followed for any combination of stock/options that the user wishes.

Individual Option Pricing

The first step is to price some individual options using each of the options contracts discussed. The following option valuations are generated using common parameters:

Risk-Free Rate: **r = 0.05**	Expected Return: **mu = 0.05**	Number of repetitions: **reps = 10000**
Maturity(years): **T = 1.0**	Volatility: **sigma = 0.2**	Number of Steps: **steps = 10**
Initial Stock Value: **S0 = 100**	Strike Price**: K = 106**	

Contract Type	Option Type	Exact Value (Black Scholes)	Simulated Value (Monte Carlo)	Confidence Interval
European Option	Call	7.59	7.50	0.25
	Put	8.42	8.22	0.21
Asian Option	Call	-	3.07	0.12
	Put	-	6.37	0.14
American Option	Call	-	7.86	0.26
	Put	-	9.23	0.16

Table 1: Option Pricing Results

As expected, the simulated value of the European option is very close to the value predicted by the Black-Scholes equation. The confidence interval of the Asian option is smaller than the other options because by averaging over each simulation, we are in effect averaging over more data points. We also observe a higher value for the American option than the European option because the early exercise ability increases the payoff opportunities for the option holder.

Option Portfolio Strategies

This section presents figures from several options strategies on a stock with starting price 100, expected return 0.05, and volatility 0.2. Most of these strategies have zero net value, because the option pricing is neutral by nature. The only exception to this is the American option, which naively exercises the option at maturity rather than at an optimal time. The American option pricing script supports payoff calculation with early exercise, but it is up to the user to select an exercise time. The strategies are described in the following table:

Strategy	Description	Expected Net Payoff
Long Stock	Buy 1 stock	Figure 1
Long Call	Buy 1 Call at K=105	Figure 2
Long Put	Buy 1 Put at K=105	Figure 3
Covered Call	Buy 1 stock, Sell 1 Call at K=105	Figure 4
Bear Put Spread European Options	Buy 1 Put at K=120 Sell 1 Put at K=100	Figure 5
Bear Put Spread Asian Options	Buy 1 Put at K=110 Sell 1 Put at K=100	Figure 6
Bear Put Spread American Options	Buy 1 Put at K=120 Sell 1 Put at K=100	Figure 7

Table 2: Option Portfolio Strategy Results

The following figures show portfolio payoff with respect to final stock price. This includes a histogram of net portfolio value, representing the distribution of payoffs. The cumulative distribution represents the likelihood that portfolio return is below some profit. Note that the Asian Bear Put Spread deviates from the linear trend seen in the European version because payoff is not directly related to end price. More payoff examples are available at the link provided.

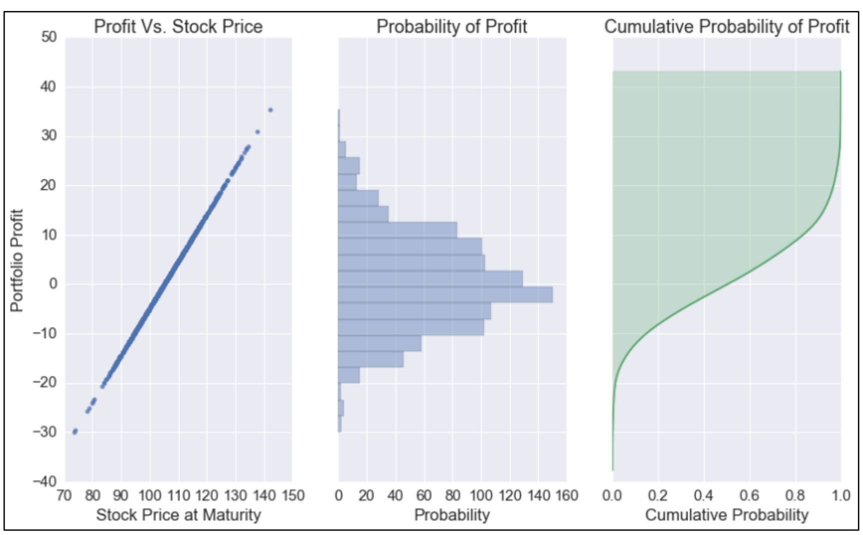

Figure 1: Long Stock Strategy

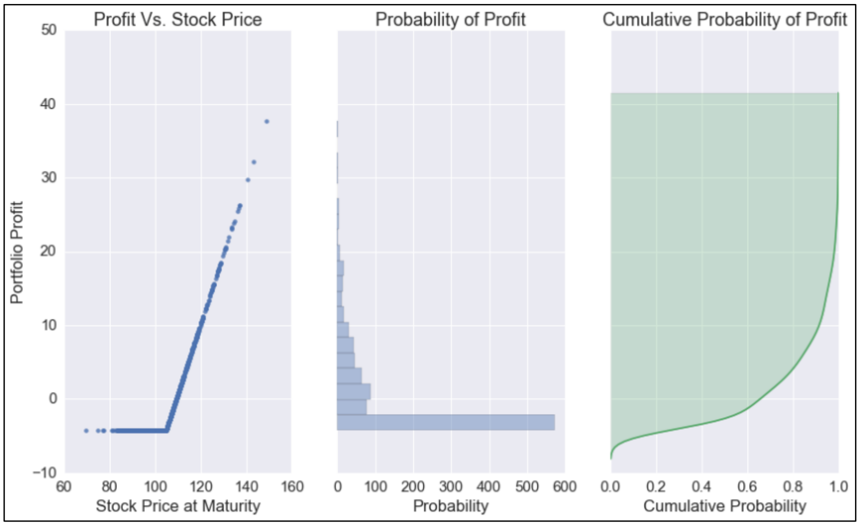

Figure 2: Long Call Strategy

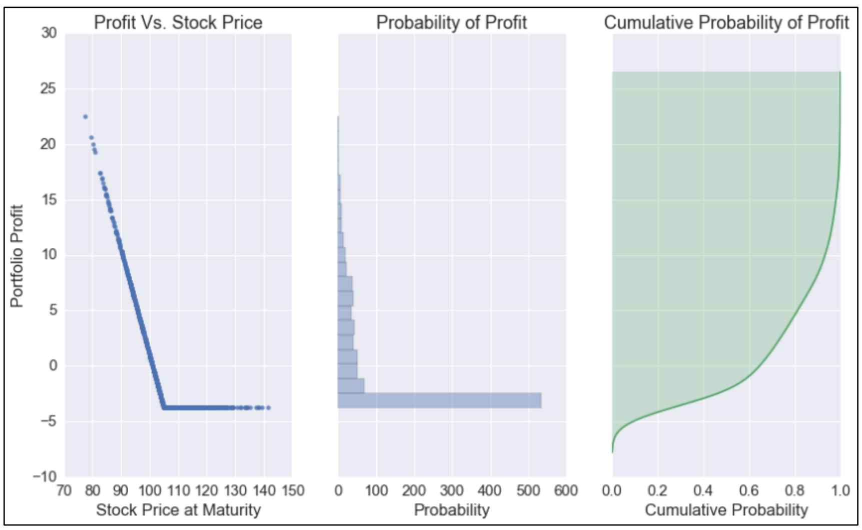

Figure 3: Long Put Strategy

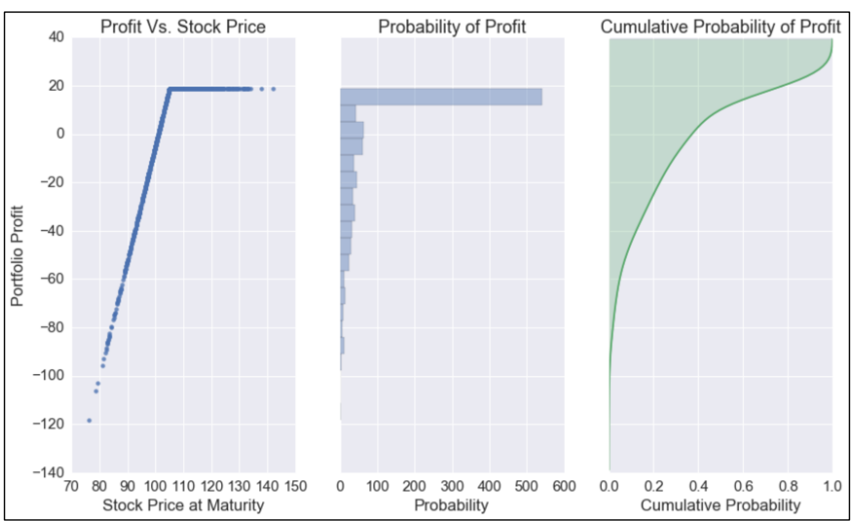

Figure 4: Covered Call Strategy

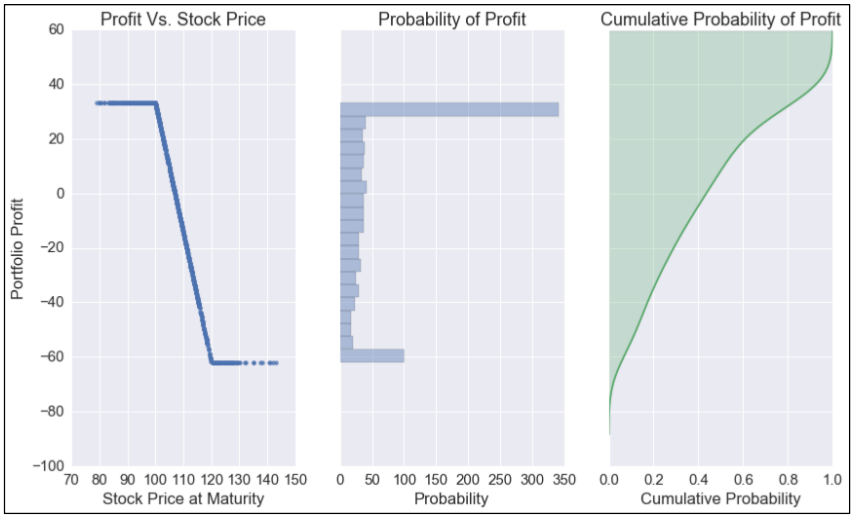

Figure 5: Bear Put Spread Strategy - EuropeanOption

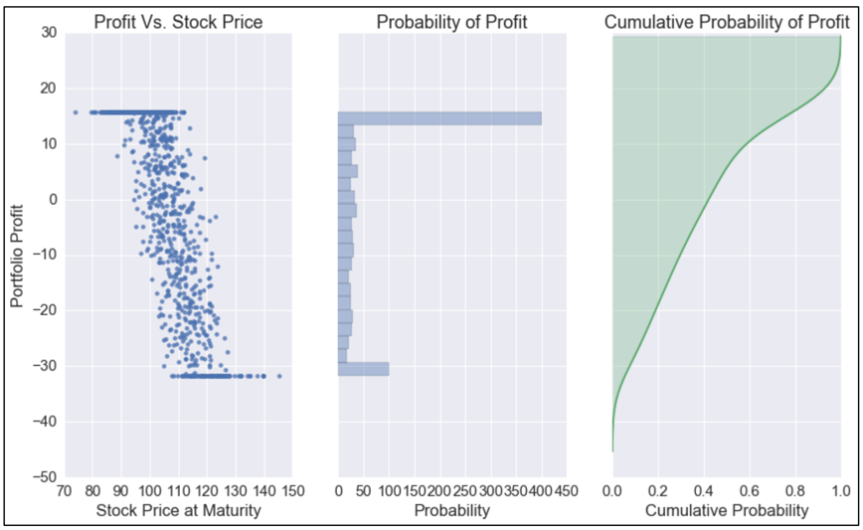

Figure 6: Bear Put Spread Strategy – Asian Option

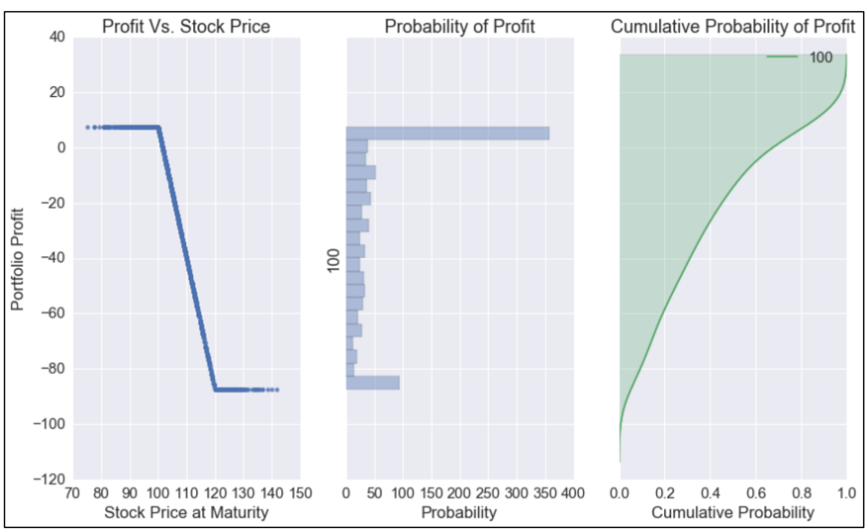

Figure 7: Bear Put Spread Strategy – American Option

Portfolio Sensitivity

The last feature of the portfolio pricing tool is the sensitivity analysis. This allows a user to determine the change in portfolio value if the expected return of a stock is changed. An example is provided on a portfolio of 3 mixed stocks and options. with a perturbation of 0.1 on one stock. This increase changes the expected value of the portfolio from $-0.18 to $2.79. The derivative of portfolio value with respect to stock A is estimated using a finite difference. It is found to be 29.75 +/- 1.43 with 95% confidence. This represents a change in portfolio value of 0.29 for each 1% change in expected return of stock A.

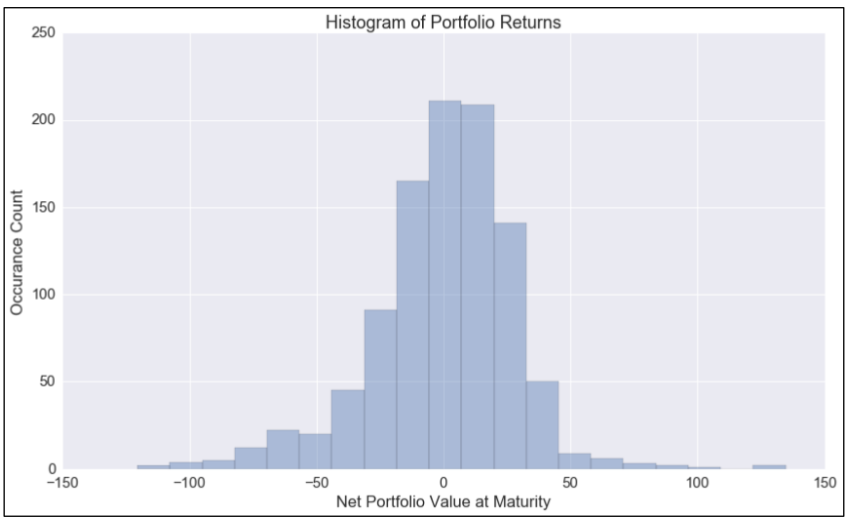

Figure 8: Portfolio Value Before Perturbation

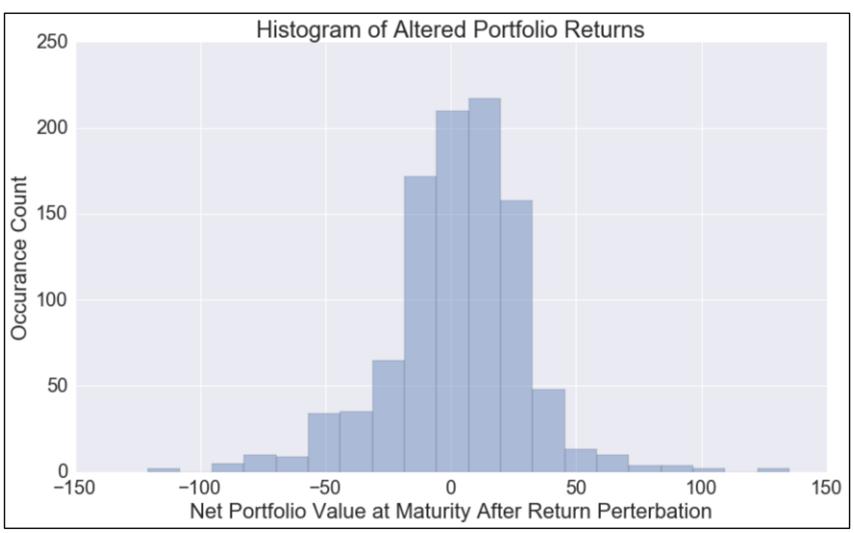

Figure 9: Portfolio Value After Perturbation

Appendix

All source code used in this project is presented below. For a better formatted version, please see files at: https://github.com/jonsmith359/MIE1613_Project.

mean_confidence_interval.py

```
#Confidence Interval Function

import numpy as np
import scipy.stats as stats
from math import sqrt
def CI(data, confidence=0.95):
    '''
    Returns the mean of a data set as well as the confidence band, h at specified conf
idence level (default 95%)
    '''
    a = 1.0*np.array(data)
    n = len(a)
    mu,se = np.mean(a),stats.sem(a)
    # z = stats.t.ppf(confidence, n)
    z1,z2 = stats.norm.interval(confidence, loc=0, scale=1)
    h=z2*se
    return mu, h
```

geometric_brownian_motion.py

```
# Geometric Brownian Motion Function

import numpy as np
from scipy.stats import norm

def BRW(drift, sigma, S0, T, paths, steps):
        '''
        Function to generate geometric random brownian motion
        drift - expected return
        sigma - volatility
        S0 - starting price
        T - maturity
        paths - number of replications
        steps - number of discretizations
        '''
        # T = float(T)
        # steps = steps
        interval = float(T)/float(steps)

        RW = np.zeros((paths,steps+1))

        for i in range (paths):
                RW[i,0] = S0
                for j in range (steps):
                        Z = norm.rvs()
                        RW[i,j+1] = RW[i,j]*np.exp((drift - sigma**2/2) * interval +
sigma * np.sqrt(interval) * Z)
        return RW
```

European_Option.py

```
# European Option Pricing Function
```

```
import numpy as np
from scipy.stats import norm
import mean_confidence_interval as conf
import geometric_brownian_motion as gbm

class EuropeanOption(object):
    '''
    Class for European Option valuation
    contract - option contract (put or call)
    S0 - initial stock value
    K - strike price
    T - time to maturity (years)
    r - annual risk free rate
    mu - expected return
    sigma - volatility
    steps - number of steps in discretization
    reps - number of simulations
    '''
    # Constructor
    def __init__(self,contract,S0,K,T,r,mu,sigma,paths):
        self.contract = contract
        self.S0 = float(S0)
        self.K = float(K)
        self.T = float(T)
        self.r = float(r)
        self.mu = float(mu)
        self.sigma = float(sigma)
        self.paths = paths
        if (contract != 'call') & (contract != 'put'):
            raise ValueError('Invalid Contract Type. Specify <call> or <put>')
        self.value = self.Sim_value()
    def BS_value(self):
        '''
        Return European option value using Black-Scholes equation
        '''
        d1 = (1/(self.sigma*self.T))*(np.log(self.S0/self.K)+(self.r+0.5*self.sigma**2
)*self.T)
        d2 = d1 - self.sigma*self.T
        if self.contract =='call':
            value = norm.cdf(d1)*self.S0 - norm.cdf(d2)*self.K*np.exp(-self.r*self.T)
        elif self.contract =='put':
            value = norm.cdf(-d2)*self.K*np.exp(-self.r*self.T)-norm.cdf(-d1)*self.S0
        return value
    def Sim_value(self):
        '''
        Return European option value using Brownian Random Walk Monte-Carlo simulation
        '''
        self.final_price = self.paths[:,-1]
        self.values=[]
        for val in self.final_price:
            if self.contract =='call':
                self.values.append(np.exp(-self.r*self.T)*np.maximum(0.0,val - self.K)
)
            elif self.contract =='put':
                self.values.append(np.exp(-self.r*self.T)*np.maximum(0.0,self.K - val)
)
        value, CI_95 = conf.CI(self.values)
        return value, CI_95
```

Asian_Option.py

```
# Asian Option Pricing Function
```

```
import numpy as np
from scipy.stats import norm
import mean_confidence_interval as conf
import geometric_brownian_motion as gbm

class AsianOption(object):
    '''
    Class for Asian Option valuation
    contract - option contract (put or call)
    S0 - initial stock value
    K - strike price
    T - time to maturity (years)
    r - annual risk free rate
    mu - expected return
    sigma - volatility
    steps - number of steps in discretization
    reps - number of simulations
    '''
    # Constructor
    def __init__(self,contract,S0,K,T,r,mu,sigma,paths):
        self.contract = contract
        self.S0 = float(S0)
        self.K = float(K)
        self.T = float(T)
        self.r = float(r)
        self.mu = float(mu)
        self.sigma = float(sigma)
        self.paths = paths
        if (contract != 'call') & (contract != 'put'):
            raise ValueError('Invalid Contract Type. Specify <call> or <put>')
        self.value = self.Sim_value()
    def Sim_value(self):
        '''
        Return Asian option value using Brownian Random Walk Monte-Carlo simulation
        '''
        ave_price = self.paths.mean(axis=1)
        self.final_price = self.paths[:,-1]
        self.values=[]
        for val in ave_price:
            if self.contract =='call':
                self.values.append(np.exp(-self.r*self.T)*np.maximum(0.0,val - self.K)
)
            elif self.contract =='put':
                self.values.append(np.exp(-self.r*self.T)*np.maximum(0.0,self.K - val)
)
        value, CI_95 = conf.CI(self.values)
        return value, CI_95
```

American_Option.py

```
# American Option Pricing Function

import pandas as pd
import numpy as np
import math
from scipy.stats import norm
import mean_confidence_interval as conf
import geometric_brownian_motion as gbm

import matplotlib.pyplot as plt

import warnings
```

```
warnings.filterwarnings("ignore")

class AmericanOption(object):
    '''
    Class for American Option valuation
    contract - option contract (put or call)
    S0 - initial stock value
    K - strike price
    T - time to maturity (years)
    r - annual risk free rate
    mu - expected return
    sigma - volatility
    steps - number of steps in discretization
    reps - number of simulations
    '''
    # Constructor
    def __init__(self,contract,S0,K,T,r,mu,sigma,paths,**exercise):
        self.contract = contract
        self.S0 = float(S0)
        self.K = float(K)
        self.T = float(T)
        self.steps = paths.shape[1]-1
        self.r = float(r)/self.steps
        self.mu = float(mu)
        self.sigma = float(sigma)
        self.paths = paths
        self.exercise = float(exercise['exercise']['exercise'])
        try:
            self.ex_step = int(math.floor(self.steps * self.exercise/self.T))
        except:
            self.ex_step = int(self.steps)
        if (contract != 'call') & (contract != 'put'):
            raise ValueError('Invalid Contract Type. Specify <call> or <put>')
        self.value = self.Sim_value()

    def Sim_value(self):
        '''
        Return American option value using Brownian Random Walk Monte-Carlo simulation
        '''
        self.paths = pd.DataFrame(self.paths)
        if self.contract =='call':
            payout = self.paths - self.K
        elif self.contract =='put':
            payout = self.K - self.paths

        # store values at exercise
        self.values = np.exp(-self.r * self.ex_step) * np.maximum(0,payout.iloc[:,self
.ex_step])

        # Set negative payoffs to 0, reverse order of dataframes along time axis
        payout[payout < 0] = 0
        paths_rev = self.paths.iloc[:, ::-1]
        payout_rev = payout.iloc[:, ::-1]
        for i in range(self.steps):
            payout_1 = payout_rev.iloc[:,i]
            payout_2 = payout_rev.iloc[:,i+1]

            # x - prices of stocks at timestep t, if non-zero payout at time t-1
            x = paths_rev.iloc[:,i+1].iloc[payout_2.nonzero()]

            # y - holding value from time t-1 to t
            HV = np.exp(-self.r)*payout_1.iloc[payout_2.nonzero()]
```

```
            # Fit quadratic regression
            try:
                c,b,a = np.polyfit(x,HV,2)
            # polyfit will fail in the case of no non-zero payouts:
            except:
                c,b,a = 0.0,0.0,0.0

            # Find expected holding value based on regression
            E_HV = a + b * x + c * np.square(x)

            # Find Exercise value at time t-1
            EV = payout_2.iloc[payout_2.nonzero()]

            # indexes of EV>E_HV
            for pos,ev in EV.iteritems():
                # if EV>E_HV, payout at t-1 is corresponding EV, and payout at t = 0
                if ev>E_HV[pos]:
                    payout_1[pos]=0
                    payout_2[pos]=EV[pos]
                # if EV<E_HV, payout (value) at t-1 is corresponding HV
                else:
                    payout_2[pos]=HV[pos]

            # Find cases where holding is optimal, and overwrite t-1 payout with disco
unted t HV
            for pos,p1 in payout_1.iteritems():
                if p1>payout_2[pos]:
                    payout_2[pos] = np.exp(-self.r)*payout_1.iloc[pos]

        end_values = payout_rev.iloc[:,-1]
        value, CI_95 = conf.CI(end_values)

        return value, CI_95
```

Option_Portfolio.py

```
# Standard Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# %matplotlib inline

# Custom Libraries
import geometric_brownian_motion as gbm

from European_Option import EuropeanOption
from Asian_Option import AsianOption
from American_Option import AmericanOption
from mean_confidence_interval import CI
class OptionPortfolio(object):
    '''
    Class to track value of stock/option portfolio on single underlying asset
    stocks - dictionary containing stock information in form:
        stocks ={
            'A':{'S0':100., 'mu':0.05, 'sigma':0.1},
            'B':{'S0':50., 'mu':0.1, 'sigma':0.2}
            }
        where
        s0 - starting stock price
        mu - expected return
        sigma - expected volatility
    r - risk-free rate
```

```
    T - option maturity
    reps - number of simulations
    steps - number of steps in discretization
    --------------------------------------------------------
    self.put_payoff - payoff of all put options in each scenario
    self.call_payoff - payoff of all call options in each scenario
    self.cost - cost of all products in the portfolio
    self.products - Dataframe storing all products in portfolio

    '''
    def __init__(self, stocks, r, T, reps, steps, sensitivity=False, **delta):
        self.stocks = stocks
        self.r = r
        self.T = T
        self.put_payoff = np.zeros(reps)
        self.call_payoff = np.zeros(reps)
        self.cost = 0
        self.products = pd.DataFrame(columns=['stock','current price','product','type'
,'strike','cost','count','total cost',])

        # Generate sample paths for each stock
        for i,j in stocks.items():
            self.stocks[i]['paths'] = gbm.BRW(stocks[i]['mu'],stocks[i]['sigma'],stock
s[i]['S0'],T,reps,steps)
            self.stocks[i]['count'] = 0

        # Create parameters for sensitivity analysis portfolio
        self.sensitivity = sensitivity
        # If sensitivity analysis is specified, create alternate portfolio attributes
ending in _delta
        # Generate alternative paths according to alternate expected return
        try:
            self.delta = delta['delta']
            self.stock_delta = delta['stock']
        except:
            self.delta = 0
            self.stock_delta = 0

        if self.sensitivity==True:
            self.put_payoff_delta = np.zeros(reps)
            self.call_payoff_delta = np.zeros(reps)
            self.cost_delta = 0
            self.products_delta = pd.DataFrame(columns=['stock','current price','produ
ct','type','strike','cost','count','total cost',])
            self.stocks[self.stock_delta]['paths_delta'] = gbm.BRW((1.+self.delta)*sto
cks[self.stock_delta]['mu'],stocks[self.stock_delta]['sigma'],stocks[self.stock_delta]
['S0'],T,reps,steps)

    def add_stock(self,stock,num,sense='long'):
        '''
        Add specified stock to portfolio
        and update all portfolio parameters
        '''

        # Select stock parameters
        S0 = self.stocks[stock]['S0']

        # Update portfolio parameters

        if sense == 'long':
            self.cost += num*S0
            self.stocks[stock]['count'] = self.stocks[stock]['count'] + num
```

```
            self.products = self.products.append({'stock':stock,'current price':S0,'pr
oduct':'stock','type':sense,'strike':'-','cost':S0,'count':num,'total cost':num*S0}, i
gnore_index=True)
        elif sense == 'short':
            self.cost -= num*S0
            self.stocks[stock]['count'] = self.stocks[stock]['count'] - num
            self.products = self.products.append({'stock':stock,'current price':S0,'pr
oduct':'stock','type':sense,'strike':'-','cost':-S0,'count':num,'total cost':-num*S0},
ignore_index=True)

        # Generate sensitivity analysis portfolio

        if (self.sensitivity==True):
            if sense == 'long':
                self.cost_delta += num*S0
                self.products_delta = self.products_delta.append({'stock':stock,'curre
nt price':S0,'product':'stock','type':sense,'strike':'-','cost':S0,'count':num,'total
cost':num*S0}, ignore_index=True)
            elif sense == 'short':
                self.cost_delta -= num*S0
                self.products_delta = self.products_delta.append({'stock':stock,'curre
nt price':S0,'product':'stock','type':sense,'strike':'-','cost':-S0,'count':num,'total
cost':-num*S0}, ignore_index=True)

    def add_put(self,stock,num,K,sense='buy',op_type='european',**exercise):
        '''
        Add specified put option to portfolio
        and update all portfolio parameters
        '''

        # Select stock parameters
        S0 = self.stocks[stock]['S0']
        mu = self.stocks[stock]['mu']
        sigma = self.stocks[stock]['sigma']
        paths = self.stocks[stock]['paths']

        # Generate price of option
        if op_type == 'european':
            put = EuropeanOption(contract='put',S0=S0,K=K,T=self.T,r=self.r,mu=mu,sigm
a=sigma,paths=paths)
        elif op_type == 'asian':
            put = AsianOption(contract='put',S0=S0,K=K,T=self.T,r=self.r,mu=mu,sigma=s
igma,paths=paths)
        elif op_type =='american':
            put = AmericanOption(contract='put',S0=S0,K=K,T=self.T,r=self.r,mu=mu,sigm
a=sigma,paths=paths,exercise=exercise)

        # Update portfolio parameters
        if sense=='buy':
            self.cost += num*put.value[0]
            self.put_payoff = self.put_payoff + np.multiply(num,put.values)
            self.products = self.products.append({'stock':stock,'current price':S0,'pr
oduct':op_type+' put option','type':sense,'strike':K,'cost':put.value[0],'count':num,'
total cost':num*put.value[0]}, ignore_index=True)

        elif sense=='sell':
            self.cost -= num*put.value[0]
            self.put_payoff = self.put_payoff - np.multiply(num,put.values)
            self.products = self.products.append({'stock':stock,'current price':S0,'pr
oduct':op_type+' put option','type':sense,'strike':K,'cost':-put.value[0],'count':num,
'total cost':-num*put.value[0]}, ignore_index=True)

        # Generate sensitivity analysis portfolio
```

```
        if (self.sensitivity==True) & (stock==self.stock_delta) :
            # if option is on sensitivity stock, new option prices must be generated
            paths = self.stocks[stock]['paths_delta']
            if op_type == 'european':
                put = EuropeanOption(contract='put',S0=S0,K=K,T=self.T,r=self.r,mu=mu,
sigma=sigma,paths=paths)
            elif op_type == 'asian':
                put = AsianOption(contract='put',S0=S0,K=K,T=self.T,r=self.r,mu=mu,sig
ma=sigma,paths=paths)
            elif op_type =='american':
                put = AmericanOption(contract='put',S0=S0,K=K,T=self.T,r=self.r,mu=mu,
sigma=sigma,paths=paths,exercise=exercise)

            if sense=='buy':
                self.cost_delta += num*put.value[0]
                self.put_payoff_delta = self.put_payoff_delta + np.multiply(num,put.va
lues)
                self.products_delta = self.products_delta.append({'stock':stock,'curre
nt price':S0,'product':op_type+' put option','type':sense,'strike':K,'cost':put.value[
0],'count':num,'total cost':num*put.value[0]}, ignore_index=True)

            elif sense=='sell':
                self.cost_delta -= num*put.value[0]
                self.put_payoff_delta = self.put_payoff_delta - np.multiply(num,put.va
lues)
                self.products_delta = self.products_delta.append({'stock':stock,'curre
nt price':S0,'product':op_type+' put option','type':sense,'strike':K,'cost':-put.value
[0],'count':num,'total cost':-num*put.value[0]}, ignore_index=True)

        elif self.sensitivity==True:
            # if option is not on sensitivity stock, use existing option costs
            if sense=='buy':
                self.cost_delta += num*put.value[0]
                self.put_payoff_delta = self.put_payoff_delta + np.multiply(num,put.va
lues)
                self.products_delta = self.products_delta.append({'stock':stock,'curre
nt price':S0,'product':op_type+' put option','type':sense,'strike':K,'cost':put.value[
0],'count':num,'total cost':num*put.value[0]}, ignore_index=True)

            elif sense=='sell':
                self.cost_delta -= num*put.value[0]
                self.put_payoff_delta = self.put_payoff_delta - np.multiply(num,put.va
lues)
                self.products_delta = self.products_delta.append({'stock':stock,'curre
nt price':S0,'product':op_type+' put option','type':sense,'strike':K,'cost':-put.value
[0],'count':num,'total cost':-num*put.value[0]}, ignore_index=True)

    def add_call(self,stock,num,K,sense='buy',op_type='european',**exercise):
        '''
        Add specified call option to portfolio
        and update all portfolio parameters
        '''

        # Select stock parameters
        S0 = self.stocks[stock]['S0']
        mu = self.stocks[stock]['mu']
        sigma = self.stocks[stock]['sigma']
        paths = self.stocks[stock]['paths']

        # Generate price of option
        if op_type == 'european':
            call = EuropeanOption(contract='call',S0=S0,K=K,T=self.T,r=self.r,mu=mu,si
gma=sigma,paths=paths)
```

```
        elif op_type == 'asian':
            call = AsianOption(contract='call',S0=S0,K=K,T=self.T,r=self.r,mu=mu,sigma
=sigma,paths=paths)
        elif op_type =='american':
            call = AmericanOption(contract='call',S0=S0,K=K,T=self.T,r=self.r,mu=mu,si
gma=sigma,paths=paths,exercise=exercise)

        # Update portfolio parameters
        if sense=='buy':
            self.cost += num*call.value[0]
            self.call_payoff = self.call_payoff + np.multiply(num,call.values)
            self.products = self.products.append({'stock':stock,'current price':S0,'pr
oduct':op_type+' call option','type':sense,'strike':K,'cost':call.value[0],'count':num
,'total cost':num*call.value[0]}, ignore_index=True)

        elif sense=='sell':
            self.cost -= num*call.value[0]
            self.call_payoff = self.call_payoff - np.multiply(num,call.values)
            self.products = self.products.append({'stock':stock,'current price':S0,'pr
oduct':op_type+' call option','type':sense,'strike':K,'cost':-call.value[0],'count':nu
m,'total cost':-num*call.value[0]}, ignore_index=True)

        # Generate sensitivity analysis portfolio
        if (self.sensitivity==True) & (stock==self.stock_delta) :
            # if option is on sensitivity stock, new option prices must be generated
            paths = self.stocks[stock]['paths_delta']

            if op_type == 'european':
                call = EuropeanOption(contract='call',S0=S0,K=K,T=self.T,r=self.r,mu=m
u,sigma=sigma,paths=paths)
            elif op_type == 'asian':
                call = AsianOption(contract='call',S0=S0,K=K,T=self.T,r=self.r,mu=mu,s
igma=sigma,paths=paths)
            elif op_type =='american':
                call = AmericanOption(contract='call',S0=S0,K=K,T=self.T,r=self.r,mu=m
u,sigma=sigma,paths=paths,exercise=exercise)

            if sense=='buy':
                self.cost_delta += num*call.value[0]
                self.call_payoff_delta = self.call_payoff_delta + np.multiply(num,call
.values)
                self.products_delta = self.products_delta.append({'stock':stock,'curre
nt price':S0,'product':op_type+' call option','type':sense,'strike':K,'cost':call.valu
e[0],'count':num,'total cost':num*call.value[0]}, ignore_index=True)

            elif sense=='sell':
                self.cost_delta -= num*call.value[0]
                self.call_payoff_delta = self.call_payoff_delta - np.multiply(num,call
.values)
                self.products_delta = self.products_delta.append({'stock':stock,'curre
nt price':S0,'product':op_type+' call option','type':sense,'strike':K,'cost':-call.val
ue[0],'count':num,'total cost':-num*call.value[0]}, ignore_index=True)

        elif self.sensitivity==True:
            # if option is not on sensitivity stock, use existing option costs
            if sense=='buy':
                self.cost_delta += num*call.value[0]
                self.call_payoff_delta = self.call_payoff_delta + np.multiply(num,call
.values)
                self.products_delta = self.products_delta.append({'stock':stock,'curre
nt price':S0,'product':op_type+' call option','type':sense,'strike':K,'cost':call.valu
e[0],'count':num,'total cost':num*call.value[0]}, ignore_index=True)
```

```
            elif sense=='sell':
                self.cost_delta -= num*call.value[0]
                self.call_payoff_delta = self.call_payoff_delta - np.multiply(num,call
.values)
                self.products_delta = self.products_delta.append({'stock':stock,'curre
nt price':S0,'product':op_type+' call option','type':sense,'strike':K,'cost':-call.val
ue[0],'count':num,'total cost':-num*call.value[0]}, ignore_index=True)

    def net_value(self):
        '''
        Return net value of the portfolio
        '''
        self.stock_value = 0
        for i,j in self.stocks.items():
            self.stock_value += np.exp(-self.r*self.T)*self.stocks[i]['paths'][:,-1]*s
elf.stocks[i]['count']
        self.put_value = self.put_payoff
        self.call_value = self.call_payoff
        self.net = self.stock_value + self.put_value + self.call_value - self.cost
        self.port_ave, self.port_ci = CI(self.net)

        return self.net, self.port_ave, self.port_ci

    def sensitivity_analysis(self):
        '''
        Return net value of the alternative portfolio (ie with perturbed stock)
        and the estimated derivative of effect expected net portfolio with respect to
perturbed stock
        '''
        portfolio = np.sort(self.net_value()[0])
        # portfolio = self.net_value()[1]

        # Calculate sensitivity stock returns
        self.stock_value_delta = 0
        for i,j in self.stocks.items():
            if i == self.stock_delta:
                self.stock_value_delta += np.exp(-self.r*self.T)*self.stocks[i]['paths
_delta'][:,-1]*self.stocks[i]['count']
            else:
                self.stock_value_delta += np.exp(-self.r*self.T)*self.stocks[i]['paths
'][:,-1]*self.stocks[i]['count']
        self.put_value_delta = self.put_payoff_delta
        self.call_value_delta = self.call_payoff_delta
        self.net_delta = self.stock_value_delta + self.put_value_delta + self.call_val
ue_delta - self.cost_delta

        self.port_ave_delta, self.port_ci_delta = CI(self.net_delta)

        # Calculate finite difference
        FD = (np.sort(self.net_delta) - portfolio)/self.delta
        # FD = (self.port_ave_delta - portfolio)/self.delta
        FD_ave,FD_CI = CI(FD)
        return  self.net_delta, self.port_ave_delta, self.port_ci_delta, FD, FD_ave, F
D_CI

    def stock_plot(self,stock, cumulative=False):
        '''
        Plot portfolio payoff vs stock price at maturity
        Also plot vertical histogram of outcomes
        If cumulative=True cumulative return distribution is plotted as well
        '''
        x = self.stocks[stock]['paths'][:,-1]
        y = self.net_value()
```

```
        y = y[0]

        sns.set(rc={'figure.figsize':(14,8)})
        sns.set(font_scale=1.5)

        if cumulative:
            fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, sharey=True, sharex=False)
            sns.regplot(x,y, fit_reg=False, ax=ax1)
            sns.distplot(y, vertical=True, bins=20, kde=False, norm_hist=False, ax=ax2
)
            sns.kdeplot(y, shade=True, vertical=True, ax=ax3, cumulative=True, gridsiz
e=100)
            ax1.xaxis.set_label_text('Stock Price at Maturity')
            ax1.yaxis.set_label_text('Portfolio Profit')
            ax1.set_title('Profit Vs. Stock Price')
            ax2.xaxis.set_label_text('Probability')
            ax2.set_title('Probability of Profit')
            ax3.xaxis.set_label_text('Cumulative Probability')
            ax3.set_title('Cumulative Probability of Profit')
            ax3.set_xlim(0,1)
        else:
            fig, (ax1, ax2) = plt.subplots(ncols=2, sharey=True, sharex=False)
            sns.regplot(x,y, fit_reg=False, ax=ax1)
            sns.distplot(y, vertical=True, bins=20, kde=False, norm_hist=False, ax=ax2
)

            ax1.xaxis.set_label_text('Stock Price at Maturity')
            ax1.yaxis.set_label_text('Portfolio Profit')
            ax1.set_title('Profit Vs. Stock Price')
            ax2.xaxis.set_label_text('Probability')
            ax2.set_title('Probability of Profit')

    def hist_plot(self):
        '''
        Plot histogram of primary portfolio
        '''
        y = self.net_value()
        y = y[0]
        sns.set(rc={'figure.figsize':(14,8)})
        sns.set(font_scale=1.5)
        ax = sns.distplot(y, vertical=False, bins=20, kde=False, norm_hist=False)
        ax.xaxis.set_label_text('Net Portfolio Value at Maturity')
        ax.yaxis.set_label_text('Occurance Count')
        ax.set_title('Histogram of Portfolio Returns')

    def sensitivity_hist_plot(self):
        '''
        Plot histogram of perturbed portfolio
        '''
        y = self.sensitivity_analysis()[0]
        sns.set(rc={'figure.figsize':(14,8)})
        sns.set(font_scale=2)
        ax = sns.distplot(y, vertical=False, bins=20, kde=False, norm_hist=False)
        ax.xaxis.set_label_text('Net Portfolio Value at Maturity After Return Perterba
tion')
        ax.yaxis.set_label_text('Occurance Count')
        ax.set_title('Histogram of Altered Portfolio Returns')

    def stock_plot_reg(self,stock):
        '''
        Plot portfolio payoff vs stock price at maturity
        Also plot vertical histogram of outcomes
        If cumulative=True cumulative return distribution is plotted as well
```

```
        '''
        x = self.stocks[stock]['paths'][:,-1]
        y = self.net_value()
        y = y[0]

        sns.set(rc={'figure.figsize':(14,8)})
        sns.set(font_scale=1.5)

        fig, (ax1, ax2) = plt.subplots(ncols=2, sharey=True, sharex=False)
        sns.regplot(x,y, order=4, ci=None, truncate=True , ax=ax1)
        sns.distplot(y, vertical=True, bins=20, kde=False, norm_hist=False, ax=ax2)

        ax1.xaxis.set_label_text('Stock Price at Maturity')
        ax1.yaxis.set_label_text('Portfolio Profit')
        ax1.set_title('Profit Vs. Stock Price')
        ax2.xaxis.set_label_text('Probability')
        ax2.set_title('Probability of Profit')
```