WORLDQUANT UNIVERSITY

Compiled Content Module 3

MScFE 630 Computational Finance

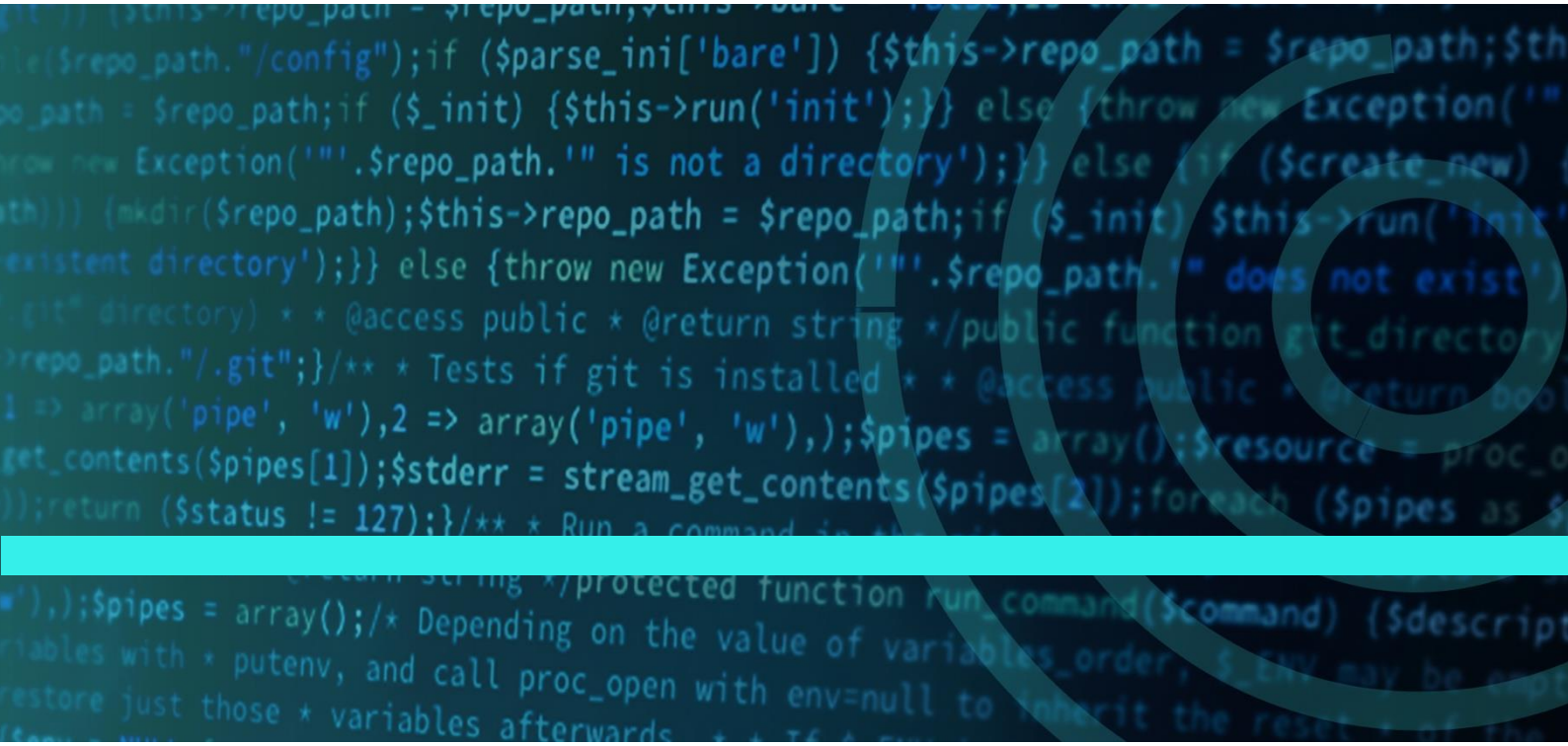
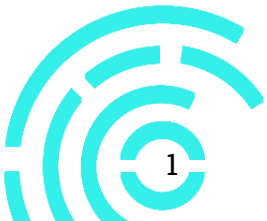


Table of Contents

Revised: 08/19/2019

Module 3: Monte Carlo Methods for Risk Management

Module 3 introduces two ways of accounting for risk in financial transactions: Value at Risk (VaR) and Credit Valuation Adjustment (CVA). The module begins by defining VaR and discusses how to calculate it using Monte-Carlo Methods. Examples of how to implement Monte Carlo Methods in Python for VaR simulation are provided. The module concludes by defining CVA and explaining how it can be calculated using the Black-Scholes model by Merton..



Unit 1: Value at Risk

Unit 1: Notes

Introduction

Risk is everywhere and, if you let it, has the potential to destroy a business. This is particularly the case in the financial sector where markets can move in unpredictable and unexpected ways. So, being able to manage risk is a very important skill for any individual in business, and even more so in the financial sector.

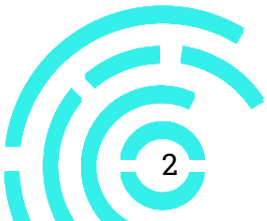
However, managing risk is more than sitting in a room and brainstorming what can go wrong. In order to truly manage it, we need techniques which allow us to introduce a degree of objectivity and consistency when identifying and quantifying the risk that the business or a project faces. The aim of this module will be to expose you to two popular methods for quantifying risk: Value at Risk (VaR) and Credit Valuation Adjustment (CVA).

Definitions

Risk metrics

A technique for quantifying the risk in a portfolio is known as a risk metric. This gives us a way of numerically describing the amount of uncertainty in that portfolio. Note that we will be talking about portfolios throughout this unit. A portfolio is just a group of assets.

A few examples of potential risk metrics include volatility (or variance) and correlation. These give an idea of the extent to which your portfolio's value could decrease, and how susceptible it is to massive swings in value. However, in order for them to fully characterize the risk inherent in the portfolio, you would need the distribution of returns to be multivariate normal, which is very unlikely. Thus, we would need to use a more general metric: Value at Risk.



Value at Risk

Value at Risk gives an indication of how much you stand to lose on a portfolio with a given probability, over a specific time period. This can be measured in absolute terms (we could lose \$100 in portfolio value) or in relative terms (the value of our portfolio will decrease by 5%). Because of the possible usefulness of both measures, we will present both potential definitions (although they are linked regardless).

Now, let our current portfolio value be $X(t)$. We can then define the return on our portfolio, until some fixed time T, as:

$$R_{t,T} = \frac{X(T-t) - X(t)}{X(t)}$$

Note that the α-quantile of a distribution, $x_\alpha(t,T)$, is defined as:

$$\mathbb{P}\left[R_{t,T} < x_\alpha(t,T)\right] = \alpha.$$

Thus, we can define VaR as being:

$$VaR_\alpha(t,T) = \begin{cases} -x_\alpha(t,T) & \text{If we look at portfolio returns} \\ x_\alpha(t,T)X(t) & \text{If we look at portfolio value} \end{cases}$$

We can interpret VaR as being a quantile value: what values does the portfolio take on in the worst 5% of cases? We can also interpret this in terms of confidence intervals. We are $(1-\alpha)$-percent certain that the value of the portfolio will not fall below $VaR_\alpha(t,T)$.

Note that the definition of VaR is the negative of the quantile; this is an aesthetic definition so that the actual VaR value is positive.

For example, suppose we have a portfolio that is worth \$100. Say we know that there is a 5% chance that the portfolio will have a value of \$20 or less. Then the 5% VaR for the portfolio is \$80, or 80%. We are 95% confident that the value of our portfolio will not fall by more than \$80.

Note that there is a different VaR for different significance levels, and different time periods. It is very important to note that VaR changes when we change these quantities.

Expected shortfall

Other names for this include conditional VaR or expected tail loss. Expected shortfall tells us how much we stand to lose assuming that our portfolio value does actually fall below our VaR. Put more simply, it is how much we stand to lose, on average, if we only consider the worst α-percentile of cases.

Conditional VaR is calculated as:

$$EL_{\alpha}(t,T) = -\mathbb{E}\left[R_{t,T}\middle|R_{t,T} < -VaR_{\alpha}(t,T)\right] \times X(t),$$

which is the negative of our expected return, given the expected return is less than the VaR, multiplied by the portfolio value. Note that this means that we are dealing with absolute terms, and not relative terms. We are, again, taking the negative for aesthetic reasons (and other sources may define the expected shortfall without the negative).

Another important note here is that, because we are looking at $R_{t,T}$ (the return of the distribution), the VaR value must also be in terms of the portfolio returns, and not the portfolio value.

Calculating VaR

Now that we know what VaR is, we will focus on how to actually calculate VaR. There are many ways to model VaR, so the methods we will present are not an exhaustive list. We will be focusing on the Monte Carlo class of VaR models and historical VaR simulation.

Monte Carlo VaR models

Monte Carlo VaR models make use of Monte Carlo methods to project asset values, in order to get some idea of what happens to your portfolio value in the future. In doing so, you will be able

to build a distribution for your portfolio value. You can then use this distribution to estimate the α-percentile of the portfolio, and to estimate a VaR.

More explicitly, Monte Carlo VaR models work as follows:

- Determine dynamics for the assets in your portfolio.
- Use Monte Carlo simulation to simulate the value of these assets n-days into the future.
- Determine your portfolio value in each scenario.
- Determine the α-quantile of these projected portfolio values.

Historical VaR models

One of the other popular methods to estimate VaR is historical simulation. This simulation process makes no parametric assumptions, and instead entirely uses data to project what may happen. For example, suppose we have three days' worth of share data.

We can work out two days' worth of returns for that share. If we then sample from these two returns, we can then project the value of our share one day into the future. In doing so, we are assuming that the company will be subject to similar conditions as it was in the past, so the past returns will be reflective of future returns. If we were to repeat this process several times (using significantly more data), we could apply a similar process as with Monte Carlo VaR models in order to estimate VaR.

Historical simulation works as follows:

- Source price data for the assets in the portfolio.
- Calculate the daily returns for all assets in the portfolio.
- Randomly sample n many log-returns and use these to grow the respective asset. values in the portfolio n-days into the future.
- Determine your portfolio value in each scenario.
- Determine the α-quantile of these projected portfolio values.

Which method is better?

Each method comes with its own pros and cons. The historical method doesn't require any parametric assumptions regarding the returns of your assets; however, it does need a considerable amount of data to be applied. This may force you to use data from further

into the past than you would be comfortable (arguably the most recent data would be a better reflection of the future asset performance). In fact, many regulators require that the data used for historical VaR calculations include periods of "stress". Thus, the firm may be forced to include data from high stress periods if the most recent data doesn't meet regulatory standards. This implies that the process of selecting appropriate data can be particularly difficult. Monte Carlo methods don't fall prey to this data history issue once you have calibrated your return model; but a model needs to be implied which immediately introduces model risk into your VaR estimate.

Historical simulation focuses on assessing what has happened in the past in order to determine the risk we face in the future. Monte Carlo simulation tries to find a balance between what has happened in the past, and what could potentially happen in the future. It makes use of models which can allow for events which haven't explicitly occurred before, and it also uses historical data to calibrate its model parameters, which allows for it to incorporate what has happened in the past into its estimates.

As is often the case, the best method would be problem specific. In fact, there will likely be legislature which requires a particular method for assessing VaR.

Conditional tail VaR

In the process of calculating VaR (in terms of the methods given in these notes), you will imply a distribution of portfolio values.

In order to determine Conditional Tail VaR, given that you have already determined the VaR, you would just need to work out the average value of the portfolio values which are less the VaR value.

Unit 1 : Video Transcript

In today's video we are going to briefly go through what a risk metric is. We will spend the majority of the video going over what Value at Risk is, and how we can go about estimating its value.

Risk metrics

Before we move on to Value at Risk, though, let's spend a little time on risk metrics. A risk metric is a way of measuring the risk in a portfolio. As you will see, Value at Risk or VaR, is a risk metric. Another example of a risk metric, which will be covered in another video, is a credit valuation adjustment. This gives an idea of how much credit risk your portfolio is exposed to.

VaR

But what is VaR?

VaR tells us how low our portfolio value can fall over a specific period of time, with a given level of certainty. For example, say you are holding a portfolio that is worth $1000. If I told you that your 1% VaR over the next year is $200 then, you can be 95% certain that the value of your portfolio will not be more than $200.

Mathematically, we can describe VaR in terms of quantiles. Suppose we know that the α –quantile of the return distribution of our portfolio over the next n-days is $x_{\alpha,n}$. This means that the worst α –percent of returns will be less than or equal to this value of $x_{\alpha,n}$. Once we have this value, we can then define our VaR as:

$$VaR_\alpha(t,T) = \begin{cases} -x_\alpha(t,T) & \text{If we look at portfolio returns} \\ x_\alpha(t,T)X(t) & \text{If we look at portfolio value} \end{cases}$$

I've included two definitions of VaR here, but one implies the other. The future value of the portfolio implies a specific return on that portfolio, and vice versa. However, we will be using the portfolio return definition of VaR when we calculate VaR in Python, in the next video.

Expected shortfall

A useful extension of VaR is expected shortfall, which is also known as conditional VaR or expected tail loss. This is just the average value that the portfolio takes on, if we only look at the cases where the returns of the portfolio are less than the VaR. This gives us an idea of how bad things can get, if they are worse than the VaR.

Calculating VaR

Now that we have some idea of what VaR is, let's go over how to calculate this value. I will present two methods in this video. It is important to note that there are many variations of these methods, as well as many other techniques that you could use as well.

Monte Carlo methods for estimating vaR

The first method we will cover is Monte Carlo VaR models. This works by assigning dynamics to the assets in your portfolio, projecting their value using Monte Carlo simulation and calculating the future portfolio value, then working out the α-quantile of these simulations. The Python code that would need to be used to produce this VaR estimate will be covered in another video.

Using historical simulation to estimate VaR

The first method we will cover is Monte Carlo VaR models. This works by assigning dynamics to the assets in your portfolio, projecting their value using Monte Carlo simulation and calculating the future portfolio value, then working out the α-quantile of these simulations. The Python code that would need to be used to produce this VaR estimate will be covered in another video.

Calculating conditional VaR

The last thing we will cover today is how to calculate the conditional VaR for a portfolio. This is relatively easily done once you have calculated your VaR. Assuming that you have in fact

calculated your VaR, you can then take the average return from all the simulated returns less than the VaR estimate, multiply this with your current portfolio value and you will have worked out your conditional VaR.

.

Unit 2: Value at Risk in Python

Unit 2 : Notes

The code in this unit is particularly dense. I would strongly recommend going through it with Jupyter Notebook open and checking each line as you go along. Being able to visualize the data helps significantly in terms of understanding what the code is trying to achieve.

Applying Monte Carlo VaR simulation in Python

The goal in this part of the notes will be to develop code for generating Monte Carlo estimates for the VaR.

The first thing to do is import all the libraries that we will use for this unit:

In [ ]:
```
import numpy as np
from scipy.stats import norm
import numpy.matlib
from scipy.stats import uniform
import matplotlib.pyplot as plt
import math
import random
```

Now we need to set up all the relevant share information. These values were chosen just for this example. The code below creates column vectors for the initial share values, and the volatilities for each share. We also have a 3x3 correlation matrix, and a Cholesky decomposition of this matrix (saved as "L").

In [ ]:
```
#General share information
S0 = np.array ([[100], [95], [50]])
sigma = np.array ([[0.15], [0.2], [0.3]])
cor_mat = np.array ([[1, 0.2, 0.4], [0.2, 1, 0.8], [0.4, 0.8, 1]])
L = np.linalg.cholesky(cor_mat) #Cholesky decomposition
r = 0.1
T = 1
```

Now we can set up information that is more relevant to our problem. We need to decide on our total number of simulations and the level of confidence for our VaR estimate.

Once this is done, we can calculate our initial portfolio value (assuming that we hold one unit of each of the three assets) as the sum of the current asset values.

In [ ]:
```
#Applying Monte Carlo estimation of VaR
np.random.seed (0)
t_simulations = 10000
alpha = 0.05
#Current portfolio value
portval_current = np.sum(S0)
```

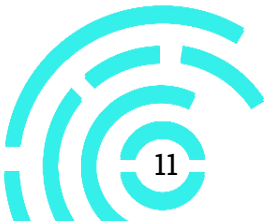
We then import our terminal share price function from the previous module:

In [ ]:
```
#Terminal share function
def terminal_shareprice (S_0, risk_free_rate, sigma, Z, T):
    """Generates the terminal share price given some random normal values, Z"""
    return S_0*np.exp((risk_free_rate-sigma**2/2)*T+sigma*np.sqrt(T)*Z)
```

The next step is to simulate future share values and portfolio values (assuming that we keep our holdings constant). The "Z" variable is a correlated multivariate normal sample.

In [ ]:
```
#Creating 10000 simulations for future portfolio values
Z = np.matmul (L, norm.rvs(size = [3, t_simulations]))
portval_future = np.sum (terminal_shareprice (S0, r, sigma, Z, T), axis = 0 )
```

We can now calculate the expected portfolio returns given our simulations. Before trying to calculate the VaR, it would make things significantly easier to sort this list. Otherwise, we would need to run some form of search algorithm to find the smallest portfolio returns. Lastly, we determine our VaR as the negative of the α-quantile, where our α is 5%. We use the int function here because, when using matrix indexation, python can only index using integer values. The int function just makes sure that the script doesn't error out.



In [ ]:
```
#Calculating portfolio returns
portreturn = (portval_future - portval_current)/portval_current

#Sorting returns
portreturn = np.sort(portreturn)

#Determining VaR
mVaR_estimate = -portreturn[int (np.floor (alpha*t_simulations))-1]
```

Applying historical simulation to estimate VaR

We now move on to estimating our VaR using historical simulation. We will use the same libraries as the previous unit, plus one extra:

In [ ]:
```
from scipy. stats import uniform
```

Once again, we set up our preliminary information.

In [ ]:
```
#Historical simulation
random.seed (0)

t_simulations = 10000
alpha = 0.05

hist_S0 = price_path[-1]
hist_portval = np.sum(hist_S0)
hist_portret = [None]*t_simulations
```

At this point, we would source historical share data. However, in this example we will rather generate our own. First, we need a function which produces a share price path. This works in a similar way to our terminal share value function above; the big difference is that it keeps track of the share value on a daily basis. Note the use of the `cumsum` function within the `share_path` function.

This is a very convenient function which returns the sum of the elements in a matrix at that point in the matrix (so the first element of the resulting matrix will be the same as

the original matrix, but the last element in the `cumsum` matrix will be the sum of the elements up until that point). This allows us to write share prices as being a function of the individual changes up until that point.

In [ ]:
```
def share_path (S_0, risk_free_rate, sigma, Z, dT):
    """Generates the terminal share price given some random normal values, Z"""
    return S_0*np.exp(np.cumsum((risk_free_rate -sigma**2/2)*dT + sigma*np.sqrt(dT)*Z,1))
```

This code generates our price path. The `price_path` variable can be viewed as a synthetic version of the historical data that we would normally use.

In [ ]:
```
#Generating synthetic share data
random.seed (0)

Z = norm.rvs(size = [3,5*365])
corr_Z = np.transpose(np.matmul(L,Z))
price_path = share_path(S0, r, sigma, corr_Z, dT)
```

We now determine our current portfolio value as the sum of the most recent share prices. We also initialize a vector to capture our simulated portfolio returns. The last step is to determine our share price log returns. We find the log-differences between them to get the daily changes; in other words, we are working out the continuous changes in the share price between days. This is mostly done because it is easier to code than working with non-log based changes. The hist_lret variable gives us the daily returns on our shares.

In [ ]:
```
hist_S0 = price_path[-1]
hist_portval = np.sum(hist_S0)
hist_portret = [None]*t_simulations

#Determining historical log returns
hist_lret = np.log(price_path[1:]) - np.log(price_path[0: -1])
```

We now want to randomly sample from our pool of share returns. We generate a sample from a $U(0,1854)$ distribution (the size of the `hist_lret` vector is 3x1854). Before using the values in `rand_samp`, we need to convert each element into an integer before using it as an index

to find the sampled log-returns. Then, because we are dealing with log-returns, we need to exponentiate the returns to grow the share prices. Finally, we apply a similar method as we did with Monte Carlo estimation.

In [ ]:
```
#Using historical returns to project future returns
for i in range (t_simulations):
    rand_samp = uniform.rvs(size = 365)*(len(price_path)-1)
    rand_samp = [int(x) for x in rand_samp]
    share_returns = hist_lret [rand_samp]
    s_term = hist_S0*np.exp(np.sum(share_returns, axis = 0))
    hist_portret[i] = (np.sum(s_term) - hist_portval)/hist_portval
```

This is the same final step as the Monte Carlo estimation.

In [ ]:
```
#Sorting portfolio returns
hist_portret = np.sort(hist_portret)

#VaR estimate
hVaR_estimate = -hist_portret[int(np.floor(alpha*t_simulations))-1]
```


Unit 2: Video Transcript

In this video, we will go through the Python code that uses Monte Carlo and historical simulation to estimate VaR.

Before we start going through code, there are two things that you need to note. Our portfolio contains one share. We have simulated the data that we are going to use for historical simulation. In order to simulate our data, we generated a share price path assuming geometric Brownian Motion.

Using Monte Carlo and historical simulation for VaR estimation

The code I will be referring to in this unit is:

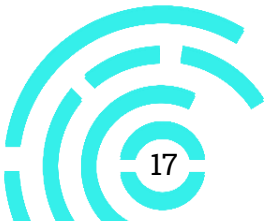
In[]:
```
#Determining tail VaR of given portfolio
import numpy as np
from scipy.stats import norm
from scipy.stats import uniform
import matplotlib.pyplot as plt
import random

#Share information
S0 = 25
sigma = 0.1
mu = 0.1
T = 1
alpha = 0.01

# Setting our seed before generating our synthetic data
np.random.seed(0)

dT = 1/365

def share_path(S_0, risk_free_rate,sigma,Z,dT):
    """Generates the terminal share price given some random normal values, Z"""
    return S_0*np.exp(np.cumsum((risk_free_rate-sigma**2/2)*dT+sigma*np.sqrt(dT)*Z,1))

Z_histdata = norm.rvs(size = [1,10*365])
price_path = share_path(S0, mu,sigma,Z_histdata,dT)

hist_lret = np.log(price_path[0,1:]) - np.log(price_path[0,0:-1])
mc_mu = np.mean(hist_lret)*365
mc_sigma = np.std(hist_lret)*np.sqrt(365)

def terminal_shareprice_adj(S_0, mu,sigma,Z,T):
    """Generates the terminal share price given some random normal values, Z"""
    return S_0*np.exp((mu)*T+sigma*np.sqrt(T)*Z)

alpha = 0.01
S0 = price_path[0,-1]
mportval_current = S0

mV_estimate = [None]*50
histV_estimate = [None]*50

for i in range(1,51):
    Z = norm.rvs(size = i*1000)
    montportval_future = terminal_shareprice_adj(S0,mc_mu,mc_sigma,Z,T)

    hist_portret = [None]*i*1000
    for j in range(1,i*1000+1):
        rand_samp = uniform.rvs(size = 365)*(len(price_path[0])-1)
        rand_samp = [int(x) for x in rand_samp]
        share_returns = hist_lret[rand_samp]
        s_term = S0*np.exp(np.sum(share_returns, axis = 0))
        hist_portret[j-1] = (s_term - mportval_current)/mportval_current

    montport_return = np.sort((montportval_future - mportval_current)/mportval_current)
    hist_portret = np.sort(hist_portret)

    mV_estimate[i-1] = -montport_return[int(alpha*i*1000)-1]
    histV_estimate[i-1] = -hist_portret[int(np.floor(alpha*i*1000))-1]
```

Line 1 to Line 16 is essentially just pre-amble (they shouldn't be anything you haven't seen before). From lines 18 to 25, we implement code to generate synthetic historical data. More detail on the mechanics for synthetic data generation can be found in the notes. In lines 31 to 33, we define an adjusted terminal share price function. The reason we need this is because we will be using our historical returns to estimate the drift. When doing this, the average historical return is estimating the whole $\mu - \frac{1}{2}\sigma^2$ and not just μ, which is calculated in line 28. The adjusted terminal share price function accounts for this.

In line 36 we set our current share price to the latest value in our synthetic historical data, and in line 37 we set up the initial value of the portfolio as the current price of the share. This is assuming that we are holding one unit in the share.

Note that we will be working out VaR as a function of sample size for both methods. So, we pre-allocate space to store these values, which we do in lines 39 and 40. We also use our sample mean and volatility to project our share value. We do this so that our Monte Carlo and historical simulation estimates will be more similar. They will likely still differ slightly, however, because we will essentially be comparing a sample of returns against the mean return for the sample as a whole. These estimations happen in lines 28 and 29.

From Line 42 to 58, we loop through sample size and calculate VaR under both methods. Lines 43, 44, 54, and 57 are relevant to Monte Carlo simulation. The rest are relevant to historical simulation. The Monte Carlo unit samples from a random normal distribution, calculates the future portfolio value as the future share price, calculates the portfolio return, then calculates the VaR. The historical simulation parts are a bit more involved. The nested loop is necessary to generate a distribution for the future portfolio values through the historical simulation method. Briefly, we sample from a uniform distribution, use this sample to pick from our historical returns, use these returns to project our share price one year forward, and calculate the future portfolio value.



Finally, under both methods, we calculate the returns on these portfolios, work out the 1% quantile, and determine the Value at Risk. If we plot the results, we should get the following graph:

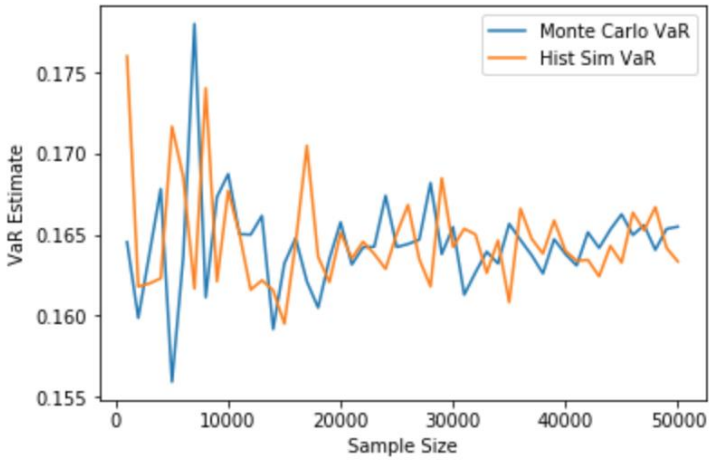


Figure 1: Monte Carlo and historical simulation VaR estimate with respect to sample size

Unit 3: Credit Valuation Adjustment

Unit 3: Notes

Up until this point, we have been running our Monte Carlo simulations within a Black-Scholes framework. This has been useful because of the ease with which we can implement it; however, it does rely on some unrealistic assumptions. One of these is that there is no risk of default. In this set of notes, we will introduce the concept of credit risk, and give an overview of how it can be accounted for.

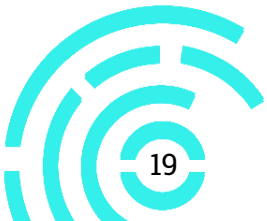
Definitions

When contracts are traded on an exchange, the exchange itself takes responsibility for ensuring that the parties receive the payments they are due. This is not the case for over-the-counter (OTC) contracts. For example, suppose you entered into an agreement for a company to buy an asset from you at some point in the future. If this company were to become bankrupt, you would miss out on any profit you were entitled to. This danger is known as counterparty credit risk. It is also often called default risk. We will use these two terms interchangeably. It is important to note that counterparty risk is faced by both parties in any transaction. In our above example, you could have insufficient funds to buy the asset to sell it to the company – in this case, they could lose out on profit. Since both parties are exposed to default risk, it is bilateral.

We now introduce a concept known as exposure. Your exposure at any time t is what you are at risk of losing, were the counterparty to default. It is the higher of the value of the portfolio and 0, i.e.

$$E(t) = \max(V(t), 0) \tag{1}$$

You can think of it this way: if your portfolio has a negative value (meaning you would owe the other party money), and they default, you don't lose out on any profit. If the value of the portfolio was positive and they defaulted, you would lose the value of the portfolio (the amount they owed you).



There are a number of further concepts which follow on from exposure:

- Current exposure, which is the exposure when t is the current time,
- Expected exposure, which is what you predict the exposure to be at a future time t,
- Potential future exposure, which confidence level on exposure at some future time (this is very similar in concept to Value at Risk).

With these ideas of counterparty risk and exposure, we can now define credit valuation adjustments (CVA). CVA is calculated as the difference between the value of a portfolio which we assume is risk-free, and a portfolio where we account for default risk. As such, CVA can be thought of as the market value of the counterparty credit risk. As you might expect, the higher your exposure, the higher your CVA, since you stand to lose more.

For a mathematical definition of counterparty risk, we need to define two variables: δ and τ. δ will be our recovery rate – this is the fraction of our portfolio that we would receive if the counterparty defaults. In other words, if our portfolio is worth $V(t)$ at time t and the counterparty defaults at time t, we would only receive $\delta V(t)$. So, we would lose $(1-\delta)V(t)$. Our second variable, τ, is a stopping time. This will be the time that the counterparty defaults. If the counterparty never defaults, $\tau = \infty$. We only lose out if the counterparty defaults before the time we close-out our position with them. Our CVA is thus as follows:

$$\text{CVA} = \mathbb{E}^{\mathbb{Q}}\left[e^{-r\tau}(1-\delta)V(\tau)\mathbb{I}_{\{\tau \leq T\}}\right], \tag{2}$$

where r is our risk-free interest rate and T is the close-out time for our portfolio.
Let's go through each element in this expression.

- $\mathbb{E}^{\mathbb{Q}}$: we are calculated our expectation under the risk-neutral measure, as we always do for pricing.
- $e^{-r\tau}$: we are discounting our value to the present time.
- $1-\delta$: this is the proportion of our portfolio that we lose if there is a default.

- $V(\tau)$: this is how much our portfolio is worth in total at the time of default
- $\mathbb{I}_{\{\tau \leq T\}}$: we only lose out if the counterparty defaults before we have closed-out our position with them.

The biggest issue we face with CVA is the difficulty in calculating it, as there are a number of factors which affect it, such as credit spreads and market factors. As a result, we introduce an extension to the Black-Scholes model, known as the Merton model.

Merton model

In order to account for the risk of default, we introduce the notion of firm value at time t. We keep all the assumptions which we had in the Black-Scholes model (besides for the no risk of default) and treat our firm value as an asset. This means it follows geometric Brownian motion and has the following SDE under the risk-neutral measure:

$$dV_t = rV_t dt + \sigma V_t dW_t; V_0 = V \tag{3}$$

where V_t is the value of the firm at time t. We assume that the value of the firm is the maximum amount which they can pay towards settling their debt. For the sake of modeling, we assume that the total debt which the firm has is some fixed amount, D, which is due at some future time T, and that default can only happen at time T. This means that, at time T, the debt holders receive

$$\min(V_T, D) = D - (D - V_T)^+$$

You'll notice that the positive-only portion, $(D - V_T)^+$, is the same payoff as a put on the firm value. The expected present value of the debt is thus equal to:

$$De^{-rT} - \mathrm{P}(V_0, r, \sigma, D, T) \tag{4}$$

where $\mathrm{P}(V_0, r, \sigma, D, T)$ is the Black-Scholes price for a put with initial asset value V_0, risk-free rate r, volatility σ, strike D, and maturity T. Using our closed-form solution for the price of a put in Black-Scholes, we get:

$$V_0\Phi(-d_1) + \big(1 - \Phi(-d_2)\big)De^{-rT} \tag{5}$$

where

$$d_1 = \frac{\ln\left(\frac{V_0}{D}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} \text{ and } d_2 = d_1 - \sigma\sqrt{T}$$

In this model, the probability of default is given by:

$$\mathbb{Q}[V_T < D] = \Phi(-d_2) \tag{6}$$

These two formulas allow us to give values to the total debt of the firm. Now, let's consider the value of CVA for a particular portfolio. Suppose our portfolio has value X_t at time t, where X_t is some random variable with maturity time T. Now, implementing our assumption that default only happens at time T, our CVA given by equation (2) becomes:

$$\begin{aligned}\text{CVA} &= \mathbb{E}^{\mathbb{Q}}\left[e^{-rT}(1-\delta)X_T\mathbb{I}_{\{\tau=T\}}\right] \\ &= \mathbb{E}^{\mathbb{Q}}\left[e^{-rT}(1-\delta)X_T\mathbb{I}_{\{V_T<D\}}\right]\end{aligned} \tag{7}$$

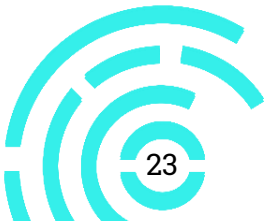
If the value of our portfolio is independent of the value of the counterparty firm, we can calculate this expectation as:

$$(1-\delta)\mathbb{E}^{\mathbb{Q}}[e^{-rT}X_T]\mathbb{Q}[V_T < D] = (1-\delta)X_0\Phi(-d_2),$$

which follows from the fact that discounted assets are martingales under the risk-neutral measure.

In the case of correlated portfolio values and firm values, we will need to resort to Monte Carlo methods. This will be done in the next unit, but we briefly introduce the concept of right way and wrong way risk. Wrong way risk is where, as the value of your portfolio increases relative to the counterparty, the likelihood of the counterparty defaulting increases. This means that the more you stand to gain, the more risk of default you face. Right way risk is simply the opposite: the more valuable your portfolio, the less likely the counterparty will default. The more negative this correlation, the more careful we have to be when performing our valuations.

In the next unit, we will be implementing the Monte Carlo algorithms for CVA. Refer back to these notes as necessary when doing the coding yourself.



Unit 3 : Video Transcript

In this presentation and the associated notes, we are going to be introducing some new concepts and an adjustment to our Black-Scholes market model in order to account for the risk of default in our modeling.

Whenever you enter into a contract directly with another party (i.e. not through an exchange), you are exposed default risk. This is the risk that the other party you are dealing with (referred to as the counterparty) doesn't pay you money that you are owed. Of course, this counterparty is also at risk that you won't pay them money you owe. This means that default risk is bilateral – both parties are exposed to it.

In order to model the default risk we face, we need to account for something known as exposure. Exposure is the amount which you stand to lose out on at any time, were the counterparty to default. It is given by the equation:

$$E(t) = \max(V(t), 0)$$

where $V(t)$ is the value of your portfolio at time t. Think about it this way – if your portfolio is worth a negative amount, the counterparty doesn't owe you anything; so, if they default, you won't lose out on any money. If your portfolio is worth a positive amount, and the counterparty defaults, the value of your portfolio is what you would lose.

With our new idea of exposure, we can now introduce a concept known as Credit Valuation Adjustment, or CVA. This value is calculated as the difference between the value of a portfolio which we assume is risk-free, and the same portfolio if we account for default risk.

As a result, it can be thought of as the market value of risk for that portfolio. As you'd expect, the higher your exposure, the higher your CVA.

To make an equation for CVA, we need two new variables: τ and δ. τ is the time of default of the counterparty – so if they never default, $\tau = \infty$. δ is called the recovery rate – this is the

proportion of our portfolio which we receive were the counterparty to default. We also assume that at some point capital T, we would close-out our position with the counterparty; if they default after this point, it doesn't affect us. With these variables in mind, the equation for calculating CVA is as follows:

$$\text{CVA} = \mathbb{E}^{\mathbb{Q}}\left[e^{-r\tau}(1-\delta)V(\tau)\mathbb{I}_{\{\tau \leq T\}}\right]$$

Let's go through each part of this equation.

- $\mathbb{E}^{\mathbb{Q}}$: we are calculated our expectation under the risk-neutral measure, as we always do for pricing.
- $e^{-r\tau}$: we are discounting our value to the present time.
- $1-\delta$: this is the proportion of our portfolio that we lose if there is a default.
- $V(\tau)$: this is how much our portfolio is worth in total at the time of default.
- $\mathbb{I}_{\{\tau \leq T\}}$: we only lose out if the counterparty defaults before we have closed-out our position with them.

In order to calculate CVA, we are going to implement the Merton model. This model has all the same assumptions as the Black-Scholes model, except for the no risk of default assumption. To calculate the risk of default, we introduce the counterparty (or firm) value, which we model in the same way as any asset i.e.

$$dV_t = rV_t dt + \sigma V_t dW_t; V_0 = V$$

We make two assumptions about default:

1. Default only occurs at the time at which we close-out our position with the counterparty, capital T.
2. The firm defaults if their value is below the firm debt amount, capital D.

The firm debt amount is simply a constant known in advance. We won't go into the mathematics here, but under these assumptions, the probability of default is given by:

$$\mathbb{Q}[V_T < D] = \Phi(-d_2)$$

where

$$d_1 = \frac{\ln\left(\frac{V_0}{D}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} \text{ and } d_2 = d_1 - \sigma\sqrt{T}$$

And CVA can be calculated as:

$$\mathbb{E}^{\mathbb{Q}}\left[e^{-rT}(1-\delta)X_T\mathbb{I}_{\{V_T<D\}}\right]$$

Where X_T is the value of our portfolio at time T. Note that this is essentially our original CVA equation, but with all the τ's replaced by capital T's, since we are assuming that the only possible default time is capital T.

We end off this video by quickly mentioning wrong-way and right-way risk. Wrong-way risk is where, as your portfolio becomes more valuable, the risk of default becomes greater. This will be when firm value and portfolio value have a negative correlation. Right-way risk is simply the opposite of this.

In the next unit, we will be implementing a CVA calculation in Python using Monte Carlo simulation.

Unit 4 : CVA in Python

Unit 4 : Notes

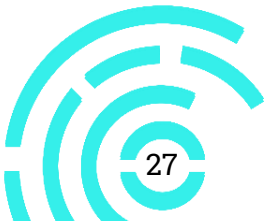
A number of the concepts introduced in the previous set of notes can be difficult to grasp at first. In this set of notes, we are going to be going through code that implements a Monte Carlo simulation for calculating CVA – this should help to make things a bit more concrete.

We are going to be considering the following scenario: you hold a call option on a stock, with an initial stock value of $100, a volatility of 30%, a strike of $110, and a term of one year. The continuously-compounded risk-free rate prevalent in the market is 10%. The counterparty to your call has some probability of default. The firm/counterparty is currently valued at $200, with a volatility of 25%, and has a debt of $180 which it owes in one year. If the firm were to default, you would receive 20% of what they owe you – this is their recovery rate.

We are going to be modeling your CVA for a number of different levels of correlation between the stock on which the option is written and the value of the counterparty. Let's go through the code that does this.

In [ ]:
```
1 import numpy as np
2 from scipy.stats import norm
3 import matplotlib.pyplot as plt
4 import random
```

As per usual, we import the libraries necessary. Since we are working within the Merton model, we are going to be simulating our assets using a standard normal distribution – this is done in the same way as in the Black-Scholes model.



In [ ]:

```
#Market information
risk_free = 0.1

#Share specific information
S_0 = 100
sigma = 0.3

#Call Option specific information
strike = 110
T = 1

#Firm specific information
V_0 = 200
sigma_firm = 0.25
debt = 180
recovery_rate = 0.2
```

Here we are capturing the information as per the problem statement. The share is the asset on which we hold a call option, and the firm is the counterparty for the option.

In [ ]:

```
# Functions for later valuations
def terminal_value(S_0, risk_free_rate,sigma,Z,T): #applies to both firm and stock
    """Generates the terminal share price given some random normal values, Z"""
    return S_0*np.exp((risk_free_rate-sigma**2/2)*T+sigma*np.sqrt(T)*Z)

def call_payoff(S_T,K):
    """Function for evaluating the call price in Monte Carlo Estimation"""
    return np.maximum(S_T-K,0)
```

It is necessary to create the functions which transform our random standard normals into terminal values. Note that we are going to be modeling our firm value in the same way which we have been modeling stock values – hence, we are going to be using the `terminal_value` function for both our stock and our firm. The `call_payoff` function finds the payoff for a call option at terminal time.

In [ ]:

```
np.random.seed(0)

corr_tested = np.linspace(-1,1,21)
cva_estimates = [None]*len(corr_tested)
cva_std = [None]*len(corr_tested)
```

We set our seed so that we can easily compare our answers. Line 3 creates an array of the correlations we are going to be testing. Run this code and have a look at the array created – it has numbers from -1 to 1, going up in steps of 0.1.

For each correlation we test, we will have a CVA estimate and a CVA standard deviation.

In [ ]:
```
1  for i in range(len(corr_tested)):
2
3      correlation = corr_tested[i]
4      if (correlation == 1 or correlation == -1):
5          norm_vec_0 = norm.rvs(size = 50000)
6          norm_vec_1 = correlation*norm_vec_0
7          corr_norm_matrix = np.array([norm_vec_0,norm_vec_1])
8
9      else:
10         corr_matrix = np.array([[1,correlation],[correlation,1]])
11         norm_matrix = norm.rvs(size = np.array([2,50000]))
12         corr_norm_matrix = np.matmul(np.linalg.cholesky(corr_matrix),norm_matrix)
13
14     term_stock_val = terminal_value(S_0,risk_free,sigma,corr_norm_matrix[0,],T)
15     call_val = call_payoff(term_stock_val,strike)
16     term_firm_val = terminal_value(V_0,risk_free,sigma_firm,corr_norm_matrix[1,],T)
17     amount_lost = np.exp(-risk_free*T)*(1-recovery_rate)*(term_firm_val < debt)*call_val
18     cva_estimates[i] = np.mean(amount_lost)
19     cva_std[i] = np.std(amount_lost)/np.sqrt(50000)
```

In this loop we are running our Monte Carlo estimation. We are going to be using a sample size of 50 000 – this means that we have 50 000 estimates for our stock price and 50 000 estimates for our firm value. If our correlation is equal to 1 or -1, our standard normal numbers for each asset (the stock and the firm) are generated as follows:

- Generate 50 000 standard normal random variables,
- Take the vector of these standard normals and multiply them by the correlation, and
- Combine these two vectors into a matrix with each array making up a row.

Ensure that you understand why this works. Try a few smaller examples in your own code, and check the matrices generated.

If the correlation lies between -0.9 and 0.9 inclusive, we generate the correlated standard normal numbers using a Cholesky decomposition.

Once we have our matrix of standard normal random numbers, we can generate our terminal stock and firm values. Line 14 creates an array of stock values using the first row of the matrix of correlated standard normals. Note that this will create an array of 50 000 stock values. Line 16 creates an array of terminal firm values.

With these terminal values generated, we can calculate our lost profit in each case. Line 15 creates an array of call values for the given stock values. Line 17 using these call values in the formula:

$$e^{-rT}(1-\delta)X_T\mathbb{I}_{\{V_T<D\}}$$

where $X_T$are the call values, V_T is the firm value, D is the debt of the firm, δ is the recovery rate, and r is the risk-free rate.

Our CVA estimate and standard deviation are calculated in the same way in which our usual Monte Carlo estimates are calculated. Before plotting our CVA estimates, we are going to calculate our CVA assuming that the correlation is 0.

In [ ]:
```
#Code to calculate default probability
d_1 = (np.log(V_0/debt)+(risk_free + sigma_firm**2/2)*(T))/(sigma_firm*np.sqrt(T))
d_2 = d_1 - sigma_firm*np.sqrt(T)

default_prob = norm.cdf(-d_2)
```

This code calculates the probability of default, which is given by:

$$\Phi(-d_2)$$

In [ ]:
```
# Code for analytical solution for vanilla European Call option
d_1_stock = (np.log(S_0/strike)+(risk_free + sigma**2/2)*(T))/(sigma*np.sqrt(T))
d_2_stock = d_1_stock - sigma*np.sqrt(T)

analytic_callprice = S_0*norm.cdf(d_1_stock)-strike*np.exp(-risk_free*(T))*norm.cdf(d_2_stock)
```

Here we have calculated our closed-form solution for the call option on the share, as per our usual Black-Scholes formula.

In [ ]:
```
uncorr_cva = (1-recovery_rate)*default_prob*analytic_callprice
```

We are finally able to compute our CVA assuming 0 correlation. We use the formula given by:

$$(1-\delta)X_0\Phi(-d_2),$$

where X_0 is the closed-form solution to the price of the call.

In [ ]:
```
plt.plot(corr_tested,[uncorr_cva]*21)
plt.plot(corr_tested,cva_estimates,'.')
plt.plot(corr_tested,cva_estimates+3*np.array(cva_std),'r')
plt.plot(corr_tested,cva_estimates-3*np.array(cva_std),'r')
plt.xlabel("Correlation")
plt.ylabel("CVA")
plt.show()
```

Finally, we plot our Monte Carlo CVA estimates for different correlations, alongside three standard deviation error bounds and the 0 correlation CVA. If you have implemented your code properly, your plot should look like the one in Figure 2.

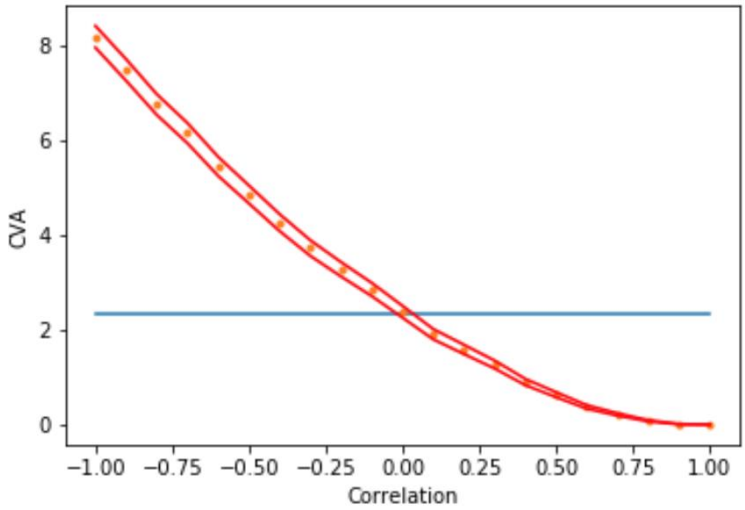


Figure 2: Monte Carlo Estimates of CVA

Unit 4: Video Transcript

In this video, we are going to be calculating CVA in Python using Monte Carlo. It is recommended that you keep your notes from the previous unit on hand, and pause the video as we go, so you can make sure everything is making sense.

We are going to be using the Merton model and calculating CVA for the following scenario: you hold a call on an asset with a term of 1 year. The counterparty/firm with which you hold this call has a risk of default. The relevant variables are:

- Continuously-compounded risk-free rate, r, of 10%
- Initial asset value, S_0 of \$100 and initial firm value, V_0 of \$200
- Asset volatility of 30% and firm volatility of 25%
- Option strike price, K, of \$110
- Firm debt, D, of \$180
- Recovery rate, δ, of 20%

Make sure you're comfortable with all these terms before moving on.

Let's look at the first part of the code which will calculate our CVA:

In [ ]:

```
#Market information
risk_free = 0.1

#Share specific information
S_0 = 100
sigma = 0.3

#Call Option specific information
strike = 110
T = 1

#Firm specific information
V_0 = 200
sigma_firm = 0.25
debt = 180
recovery_rate = 0.2
```

Here we've just imported our relevant libraries and created variables for the information we are given in the question. Note that, since the Merton model models assets in the same way as the Black-Scholes model, we are going to be using standard normal random variables to simulate our asset values.

In [ ]:
```
# Functions for later valuations
def terminal_value(S_0, risk_free_rate,sigma,Z,T): #applies to both firm and stock
    """Generates the terminal share price given some random normal values, Z"""
    return S_0*np.exp((risk_free_rate-sigma**2/2)*T+sigma*np.sqrt(T)*Z)

def call_payoff(S_T,K):
    """Function for evaluating the call price in Monte Carlo Estimation"""
    return np.maximum(S_T-K,0)
```

The first function, `terminal_value`, calculates values for both our asset and the firm at maturity – you can see it's the exact same formula as we have been using in our Black-Scholes modeling. The second function, `call_payoff`, calculates call payoffs for a given terminal stock price.

In [ ]:
```
np.random.seed(0)

corr_tested = np.linspace(-1,1,21)
cva_estimates = [None]*len(corr_tested)
cva_std = [None]*len(corr_tested)
```

We are going to be calculating our CVA for a number of different correlations between our stock price and firm value – this will show the effects of right way and wrong way risk. The `corr_tested` variable has all the different correlations we are going to test, from -1 to 1 going up in steps of 0.1.

In [ ]:
```
for i in range(len(corr_tested)):

    correlation = corr_tested[i]
    if (correlation == 1 or correlation == -1):
        norm_vec_0 = norm.rvs(size = 50000)
        norm_vec_1 = correlation*norm_vec_0
        corr_norm_matrix = np.array([norm_vec_0,norm_vec_1])

    else:
        corr_matrix = np.array([[1,correlation],[correlation,1]])
        norm_matrix = norm.rvs(size = np.array([2,50000]))
        corr_norm_matrix = np.matmul(np.linalg.cholesky(corr_matrix),norm_matrix)

    term_stock_val = terminal_value(S_0,risk_free,sigma,corr_norm_matrix[0,],T)
    call_val = call_payoff(term_stock_val,strike)
    term_firm_val = terminal_value(V_0,risk_free,sigma_firm,corr_norm_matrix[1,],T)
    amount_lost = np.exp(-risk_free*T)*(1-recovery_rate)*(term_firm_val < debt)*call_val
    cva_estimates[i] = np.mean(amount_lost)
    cva_std[i] = np.std(amount_lost)/np.sqrt(50000)
```

In this loop, we are actually calculating our CVA for different correlations. We are simulating 50 000 stock values, and an associated 50 000 firm values. The if-else statement in lines 4 to 12 creates a $2 \times 50\,000$ array of standard normal random variables with a given correlation. Line 14 calculates stock values using the first row of the normal random variables array, and line 15 finds the discounted call payoffs for each of these. Line 16 calculates the terminal firm values using the second row of the normal random variables array. These call payoffs and firm values are then used to calculate how much is lost in each case in line 17.

The mean and standard deviations for these losses are calculated in the usual Monte Carlo way, to give us estimates for CVA and deviations of these estimates.

In [ ]:
```
#Code to calculate default probability
d_1 = (np.log(V_0/debt)+(risk_free + sigma_firm**2/2)*(T))/(sigma_firm*np.sqrt(T))
d_2 = d_1 - sigma_firm*np.sqrt(T)

default_prob = norm.cdf(-d_2)
```

Here we have just calculated the default probability, which we will need in calculating our 0 correlation closed-form solution for CVA.

In [ ]:
```
# Code for analytical solution for vanilla European Call option
d_1_stock = (np.log(S_0/strike)+(risk_free + sigma**2/2)*(T))/(sigma*np.sqrt(T))
d_2_stock = d_1_stock - sigma*np.sqrt(T)

analytic_callprice = S_0*norm.cdf(d_1_stock)-strike*np.exp(-risk_free*(T))*norm.cdf(d_2_stock)
```

Here we calculate the analytic price of the call option, which is combined with our default probability to give the 0 correlation CVA:

In [ ]:
```
uncorr_cva = (1-recovery_rate)*default_prob*analytic_callprice
```

Finally, we can graph our CVA estimates for different correlations

In [ ]:
```
plt.plot(corr_tested,[uncorr_cva]*21)
plt.plot(corr_tested,cva_estimates,'.')
plt.plot(corr_tested,cva_estimates+3*np.array(cva_std),'r')
plt.plot(corr_tested,cva_estimates-3*np.array(cva_std),'r')
plt.xlabel("Correlation")
plt.ylabel("CVA")
plt.show()
```

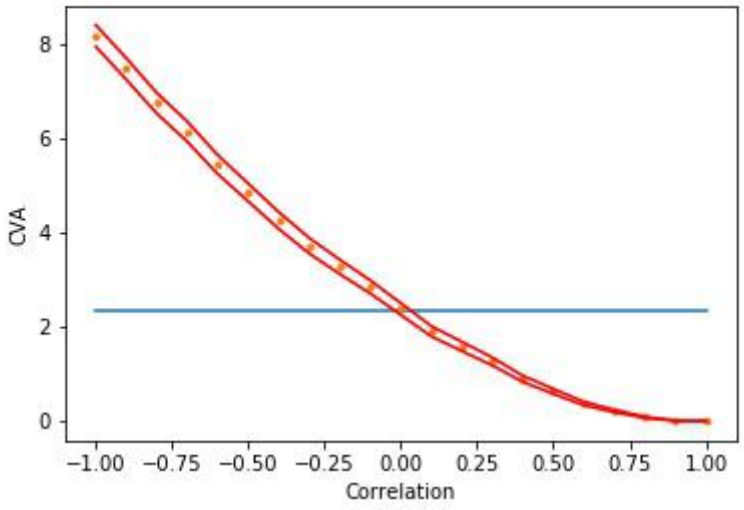


Figure 3: Monte Carlo Estimates of CVA

We have plotted error bounds and the closed-form solution for 0 correlation CVA.As you can see in this figure, as our correlation gets smaller, our CVA gets larger. This is because as our

stock performs well, the value of our call option grows, but firm is more likely to perform poorly. A negative correlation would thus result in wrong way risk, while a positive one results in right way risk.

Bibliography

Hilpisch, Y. (2014). *Python for Finance: Analyze big financial data*. " O'Reilly Media, Inc.".

Hilpisch, Y. (2015). Derivatives analytics with Python: data analysis, models, simulation, calibration and hedging. John Wiley & Sons.

Yan, Y. (2017). *Python for Finance*. Packt Publishing Ltd.

Appendix

CVA in real situations:

- CVA is the price one would pay to hedge the portfolio of derivative instruments against counterparty credit risk
- It was introduced in 2007 in order to enhance fair value accounting
- It is largely used in the derivatives markets
- There is not a standardized methodology for CVA as financial companies use different versions of CVA depending on their needs and complexity required
- Complex CVA approaches require market risk factor simulations and different market scenarios
- Large financial institutions that have a high share of derivatives in their portfolio invest heavily in CVA analysis and even have a separate CVA trading desk

Example with Poisson distribution:

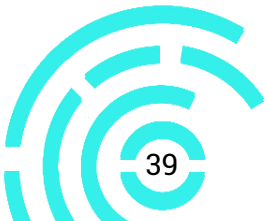
Blackberry stock price on 27 August 2019 – 6.86 USD.

We calculate CVaR and CVA for this portfolio.

```
import numpy as np
import numpy.random as npr
from pylab import *
import matplotlib.pyplot as plt
```

Consider the Black-Scholes-Merton model with the following parameters:

```
S0 = 6.86
r1 = 0.01
sigma1 = 0.2
T1 = 1.
I1 = 400
ST1 = S0 * np.exp((r1 - 0.5 * sigma1 ** 2) * T1 + sigma1 * np.sqrt(T1) *
npr.standard_normal(I1))
```

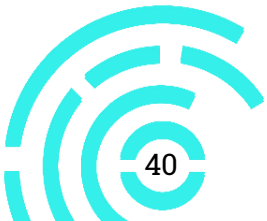
We simulate BlackBerry stock price movements.

```
ST1
L1 = 0.5 #fixed (average) loss level
p1 = 0.01 #probability of default
#Generate default scenarios using Poisson distribution
```

Poisson distribution is used in modelling rare events, in our case if the counterparty goes bankrupt.

Risk-neutral value of the future index level is equal to the current value of the asset today if there is no default.

```
D = npr.poisson(p1 * T1, I1)
D = np.where(D > 1, 1, D)

#Risk-neutral value
np.exp(-r1 * T1) * 1 / I1 * np.sum(ST1)

CVaR1 = np.exp(-r1 * T1) * 1 / I1 * np.sum(L1 * D * ST1)
CVaR1

#Present value of the asset adjusted for the credit risk
S0_CVA1 = np.exp(-r1 * T1) * 1 / I1 * np.sum((1 - L1 * D) * ST1)
S0_CVA1

#Another way for calculating the present value of the asset
```

CVaR = Risk free value – risk-adjusted value

```
S0_adj1 = S0 - CVaR1
S0_adj1

#The number of possible losses
np.count_nonzero(L1 * D * ST1)
```

We observe only 4 losses due to credit risk given a number of 400 simulations and 1 % assumed default probability.