

# Price a European Up-and-out Call Option

Ng, Joe Hoong  
ng\_joehoong@hotmail.com

Nguyen, Dang Duy Nghia  
nghia002@e.ntu.edu.sg

Ansari, Zain Us Sami Ahmed  
zainussami@gmail.com

Thorne, Dylan  
dylan.thorne@gmail.com

April. 8, 2020

**Keywords:** Exotic Options, European up-and-out call option, Black-Scholes-Merton model and Geometric Brownian Motion.

## Abstract

This report presents the results of the simulation of a European up-and-out call option over twelve months. The Black-Scholes-Merton model will be used to implement Monte Carlo estimations using Python code. The estimations will simulate paths for the underlying share and the counterparty's firm value using varying sample sizes, and determine estimates of both the default-free value of the option and the Credit Valuation Adjustment (CVA). Following from these it is simple to price the option incorporating counterparty risk, given by the default-free price less the CVA.

## 1 Introduction

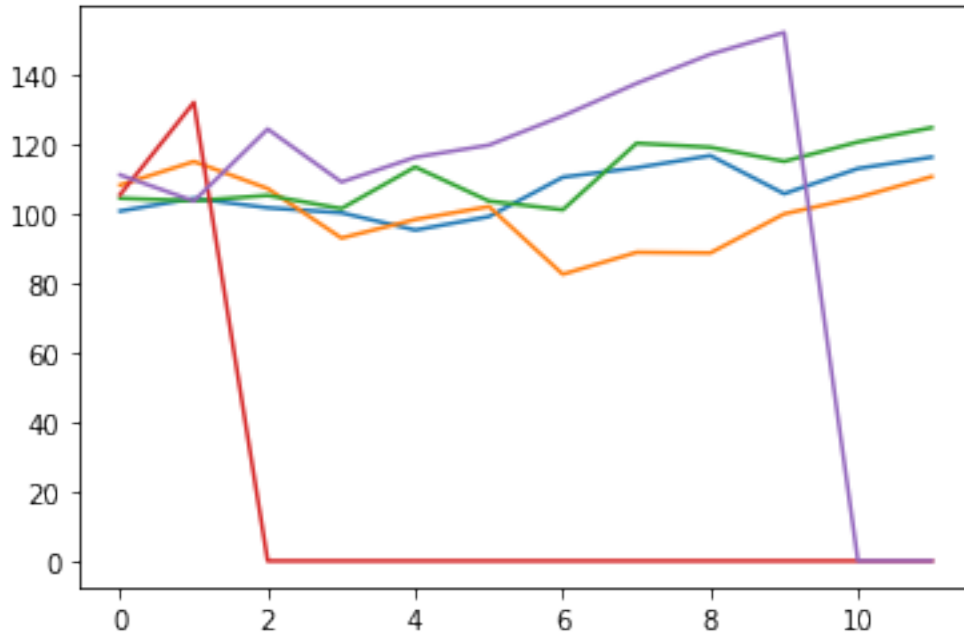
A European up-and-out call option is type of a Barrier Option, these options are consider path-dependant because barrier option's payoff is based on the underlying stock's price path. The payoff of an up-and-out option at maturity  $T$  is given by,

$$Payoff_T = (S_T - K)^+ \text{ given, } \max_{t \in [0, T]} S_t < L$$

Where, where  $K$  is the strike of the option,  $L$  is the barrier level, and  $S_t$  is the share price at time  $t$ . This is a type of call option whose payoff is reduced to 0 if the share price crosses the barrier level.

The seminal work of Merton [1] pioneered the formula for pricing barrier options. This led pricing formulas under the geometric Brownian motion (GBM) framework for one-asset barrier options by Rich [2] and multi-asset barrier options by Wong and Kwok [3]. Although the return dynamics of underlying shares are not sufficiently well described by the GBM process proposed by Black and Scholes [4], we are going to assume both the stock and counterparty firm values follow GBM with constant drift and volatilities and default only occurs at maturity.

Possible paths for the up-and-out barrier option payoff would look similar to the chart below, where two of the paths have crossed the barrier and resulted in zero payoff, but the other three have not crossed the barrier and behave like a vanilla option.



We initialize most variables as given by the question. We assume the counterparty firm's value is at 200, as this is above the counterparty's debt due in one year. Also, we calculate the default-free value of our option to be 6.60, so we select a firm value not too distant such that the CVA value we calculate is not negligible. For example, if we chose counterparty firm value as 1,000, the CVA would be 0.000.

```
[2]: ### Initialize problem parameters
T = 1 # option maturity
L = 150 # up-and-out barrier
S0 = 100 # current share price
K = 100 # strike price, at-the-money
risk_free = .08 # risk-free rate
sigma = .3 # volatility
v_0 = 200 # counterparty firm current value (Our assumption)
sigma_firm = .25 # volatility for the counterparty's firm
debt = 175 # counterparty's debt, due in one year
corr = .2 # correlation
recovery_rate = 0.25 # recovery rate
#####
corr_matrix = np.array([[1, corr], [corr, 1]])
sample_sizes = range(1000, 50001, 1000)
```

## 2 Simulated Paths for the Underlying Share and Counterparty Firm

In order to simulate paths for the underlying share and the firm, we use a Cholesky decomposition to generate the correlated price paths.

```
[3]: def share_path(S_0, risk_free_rate, sigma, Z, dT):
    return S_0*np.exp(np.cumsum((risk_free_rate-sigma**2/2)*dT + sigma*np.
    →sqrt(dT)*Z,1))

def generate_share_and_firm_price(S0, v_0, risk_free, sigma, sigma_firm, corr,
    →T, sample_size = 10, timesteps = 12):
    corr_matrix = np.array([[1, corr], [corr, 1]])
    norm_matrix = stats.norm.rvs(size = np.array([sample_size, 2, timesteps]))
    corr_norm_matrix = np.matmul(np.linalg.cholesky(corr_matrix), norm_matrix)

    share_price_path = pd.DataFrame(share_path(S0, risk_free, sigma,
    →Z=corr_norm_matrix[:,0,], dT=1/timesteps))
    share_price_path = share_price_path.transpose()

    firm_price_path = pd.DataFrame(share_path(v_0, risk_free, sigma_firm,
    →Z=corr_norm_matrix[:,1,], dT=1/timesteps))
    firm_price_path = firm_price_path.transpose()

    return [share_price_path,firm_price_path]
```

To double check that the stock prices and firm values monthly returns are correlated, we check them as follows, generating 20 different price paths with 10,000 timesteps.

```
[4]: #Testing share and firm price correlation
sample_size = 20
test = generate_share_and_firm_price(S0, v_0, risk_free, sigma, sigma_firm,
    →corr, T, sample_size, timesteps = 10000)

share_ret = np.log(test[0])

for i in range(sample_size):
    test[0]['sharelog'] = np.log(test[0][i])
    test[1]['firmlog'] = np.log(test[1][i])
    pearson, p_value = stats.pearsonr(test[0]['sharelog'].diff().dropna(),
    →test[1]['firmlog'].diff().dropna())
    print("Pearson correlation coefficient : {:.3f}, Two-tailed p-value {:.3f}".
    →format(pearson, p_value))
```

```
Pearson correlation coefficient : 0.208, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.198, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.184, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.208, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.196, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.203, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.204, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.209, Two-tailed p-value 0.000
```

```

Pearson correlation coefficient : 0.198, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.215, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.187, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.196, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.208, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.196, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.196, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.192, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.186, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.204, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.205, Two-tailed p-value 0.000
Pearson correlation coefficient : 0.188, Two-tailed p-value 0.000

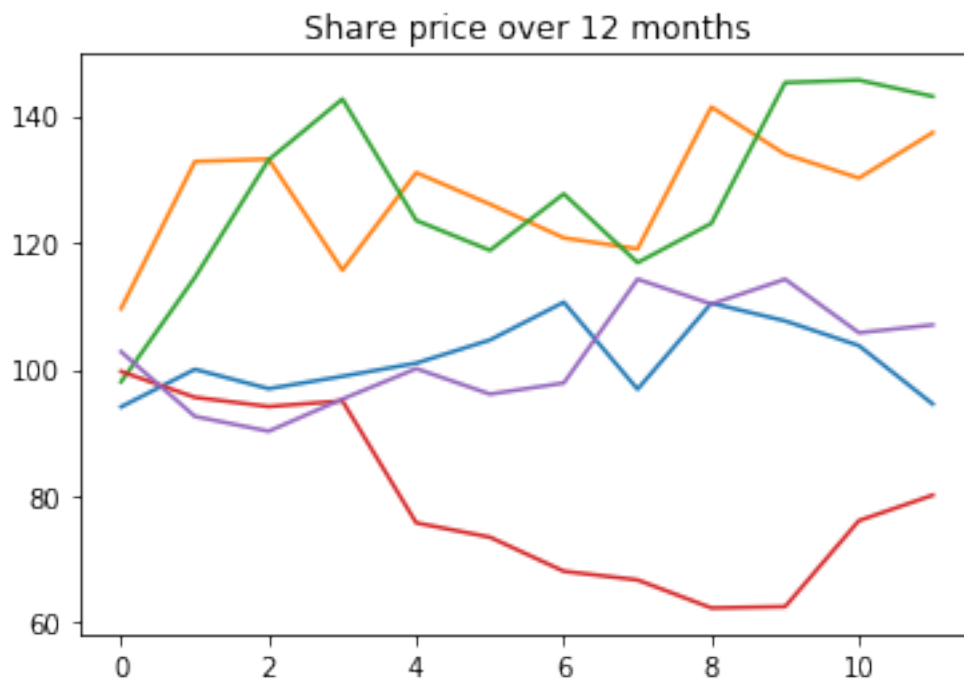
```

Let's try to simulate the share price with a small number of sample paths and visualize them over the course of 12 months

```

[5]: share_and_firm_price_12_months = generate_share_and_firm_price(S0, v_0,
    ↪risk_free, sigma, sigma_firm, corr, T, sample_size = 5, timesteps = 12)
share_price_12_months = share_and_firm_price_12_months[0]
share_price_12_months.plot(title='Share price over 12 months', legend=False);

```

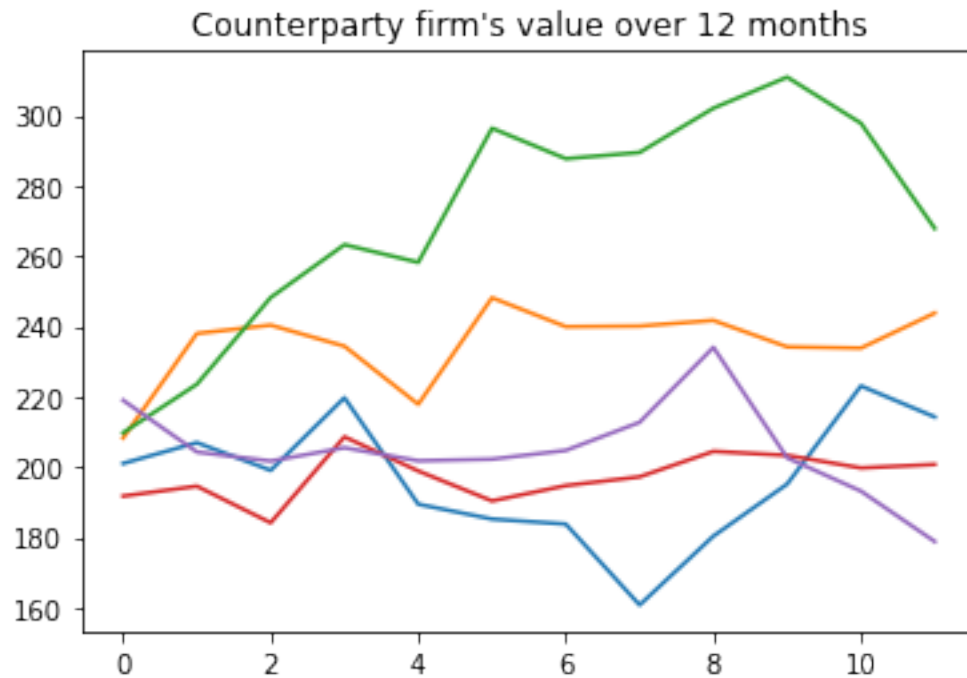


We can do the same thing to simulate counterparty firm's value

```

[6]: firm_value_12_months = share_and_firm_price_12_months[1]
firm_value_12_months.plot(title="Counterparty firm's value over 12 months",
    ↪legend=False);

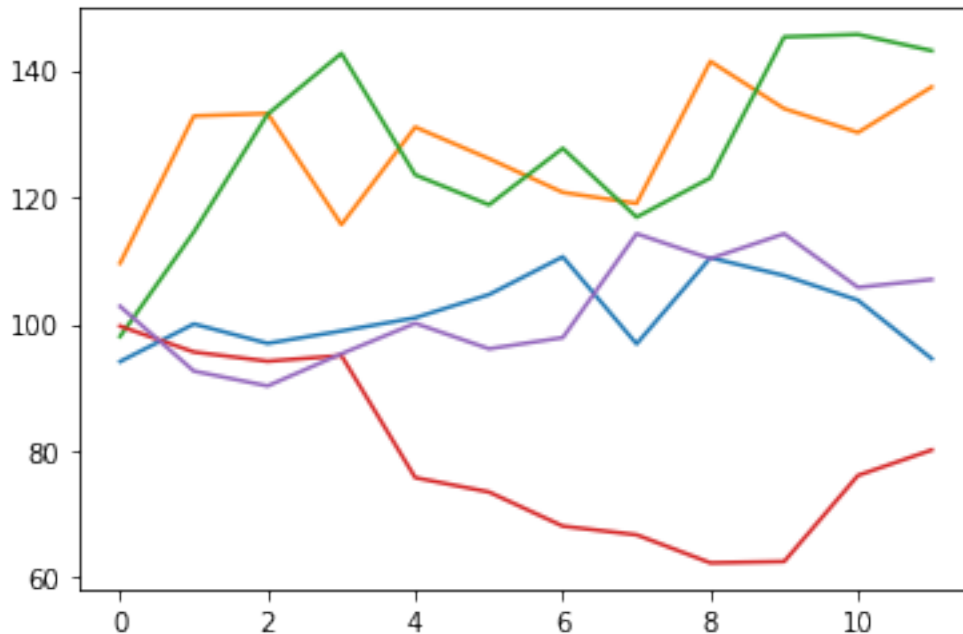
```



Let's visualize when the stopped process is applied

```
[8]: def stop(s, cond):
      ret = s.copy()
      r = ret[cond]
      if len(r) > 0:
          print(r)
          ret[r.idxmin():] = 0
      return ret

[9]: share_price_12_months.apply(lambda s: stop(s, s>L), axis=0).plot(legend=False);
```



We then define a function to calculate the payoff for the up-and-out call option.

```
[10]: # define payoff for up-and-out call option
def payoff(S_t, K, L):
    stopped_S = S_t.iloc[-1].where((S_t < L).all(), 0)
    return np.maximum(stopped_S - K, 0).to_numpy()

payoff(share_price_12_months, K, L)
```

```
[10]: array([ 0.          , 37.34484527, 43.05073004,  0.          ,  7.00008561])
```

We now create the share and firm price paths for sample sizes 1000, 2000, ..., 50000, as required for part 1 of the assignment.

```
[11]: share_price_paths = {}
firm_val_paths = {}

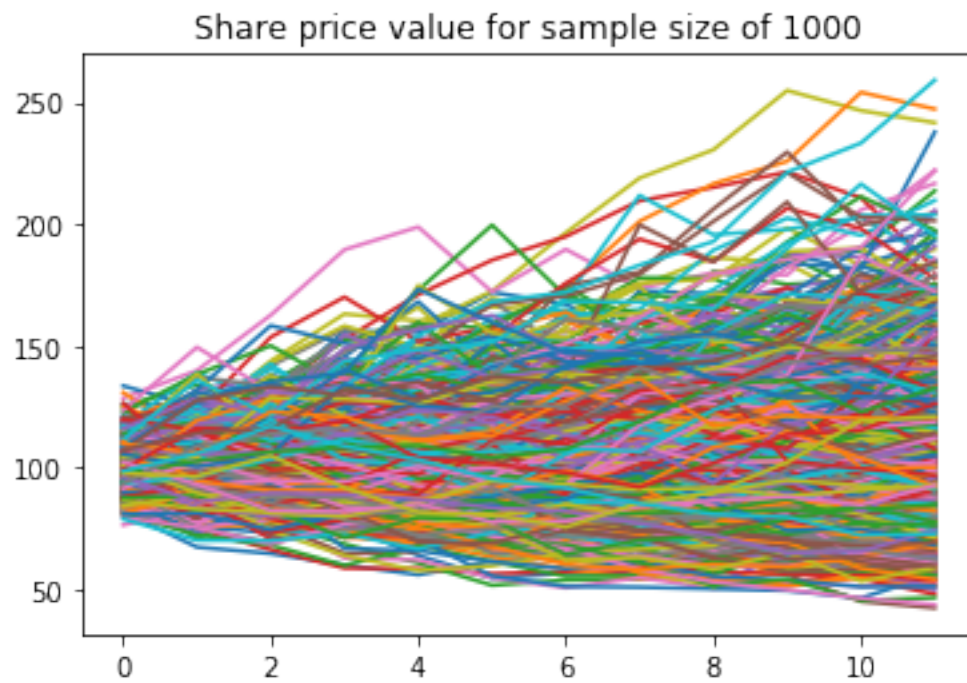
sample_sizes = range(1000, 50001, 1000)

for sample_size in sample_sizes:
    share_val, firm_val = generate_share_and_firm_price(S0, v_0, risk_free, u,
    ↪sigma, sigma_firm, corr, T, sample_size = sample_size, timesteps = 12)

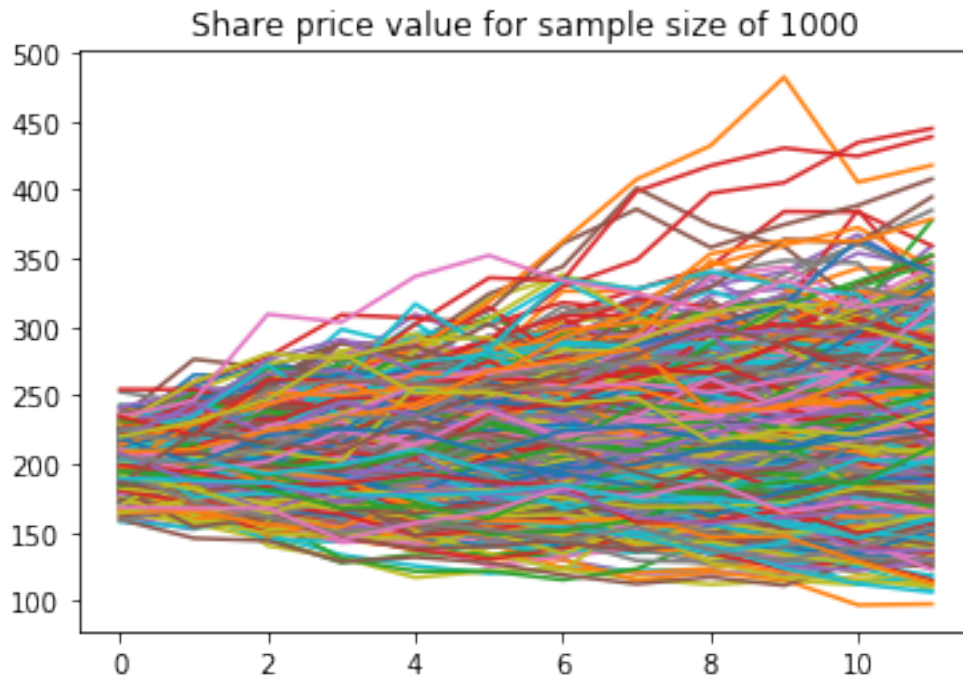
    share_price_paths[sample_size] = share_val
    firm_val_paths[sample_size] = firm_val
```

We now plot the share and firm price paths for sample size of 1000.

```
[12]: share_price_paths[1000].plot(title="Share price value for sample size of 1000",  
→legend=False);
```



```
[13]: firm_val_paths[1000].plot(title="Share price value for sample size of 1000",  
→legend=False);
```



### 3 Monte Carlo Estimations

#### 3.1 Default-Free Value Estimation

To calculate Monte Carlo estimate of the default-free option value, we calculate the average payoff of the 1000s of sample price paths, to estimate the price of the option

```
[14]: # Estimate the default-free value of the option:
option_estimate = []
option_std = []

for sample_size, paths in share_price_paths.items():
    payoffs = payoff(paths, K, L)
    option_price = np.exp(-risk_free*T)*payoffs
    option_estimate.append(option_price.mean())
    option_std.append(option_price.std()/np.sqrt(sample_size))
```

Prices of default-free option at various sample sizes:

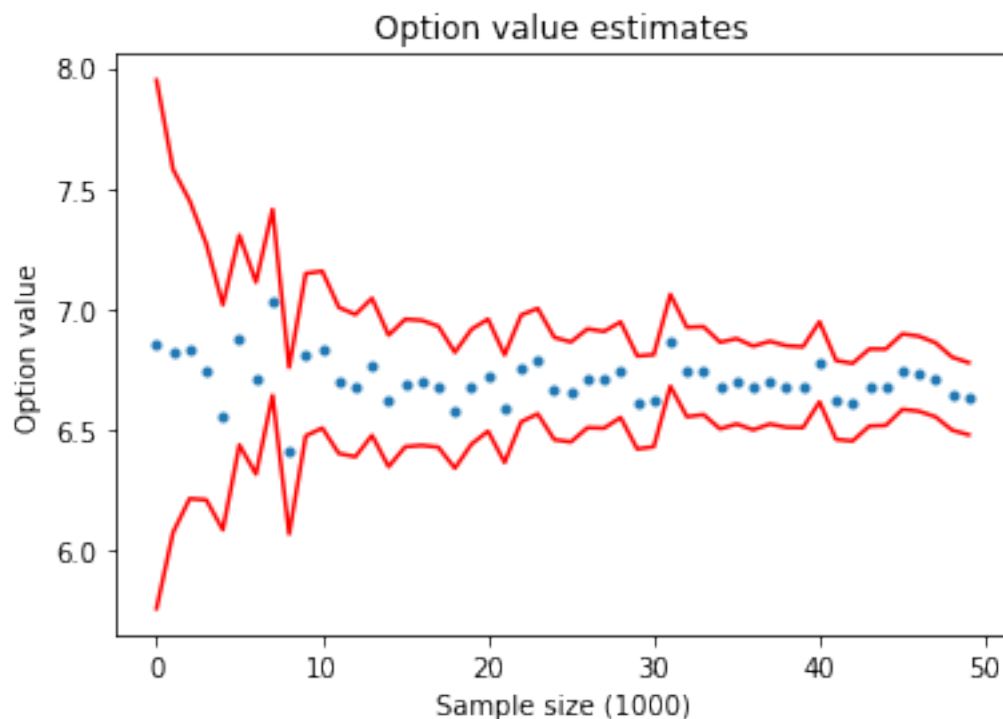
```
[15]: for i in range(len(option_estimate)):
    print("sample size: {}, Option value: {:.3f}".
        →format((i+1)*1000,option_estimate[i]))
```



sample size: 1000, Option value: 6.853  
sample size: 2000, Option value: 6.826  
sample size: 3000, Option value: 6.831  
sample size: 4000, Option value: 6.739  
sample size: 5000, Option value: 6.551  
sample size: 6000, Option value: 6.871  
sample size: 7000, Option value: 6.713  
sample size: 8000, Option value: 7.028  
sample size: 9000, Option value: 6.412  
sample size: 10000, Option value: 6.810  
sample size: 11000, Option value: 6.831  
sample size: 12000, Option value: 6.704  
sample size: 13000, Option value: 6.682  
sample size: 14000, Option value: 6.760  
sample size: 15000, Option value: 6.619  
sample size: 16000, Option value: 6.693  
sample size: 17000, Option value: 6.694  
sample size: 18000, Option value: 6.678  
sample size: 19000, Option value: 6.580  
sample size: 20000, Option value: 6.678  
sample size: 21000, Option value: 6.726  
sample size: 22000, Option value: 6.587  
sample size: 23000, Option value: 6.754  
sample size: 24000, Option value: 6.784  
sample size: 25000, Option value: 6.670  
sample size: 26000, Option value: 6.656  
sample size: 27000, Option value: 6.713  
sample size: 28000, Option value: 6.706  
sample size: 29000, Option value: 6.748  
sample size: 30000, Option value: 6.612  
sample size: 31000, Option value: 6.620  
sample size: 32000, Option value: 6.870  
sample size: 33000, Option value: 6.738  
sample size: 34000, Option value: 6.744  
sample size: 35000, Option value: 6.683  
sample size: 36000, Option value: 6.700  
sample size: 37000, Option value: 6.672  
sample size: 38000, Option value: 6.694  
sample size: 39000, Option value: 6.678  
sample size: 40000, Option value: 6.676  
sample size: 41000, Option value: 6.781  
sample size: 42000, Option value: 6.623  
sample size: 43000, Option value: 6.614  
sample size: 44000, Option value: 6.674  
sample size: 45000, Option value: 6.676  
sample size: 46000, Option value: 6.740  
sample size: 47000, Option value: 6.732  
sample size: 48000, Option value: 6.706

sample size: 49000, Option value: 6.649  
sample size: 50000, Option value: 6.628

```
[16]: plt.plot(option_estimate, '.')
plt.plot(option_estimate + 3 * np.array(option_std), 'r')
plt.plot(option_estimate - 3 * np.array(option_std), 'r')
plt.title("Option value estimates")
plt.xlabel("Sample size (1000)")
plt.ylabel("Option value")
plt.show()
```



### 3.2 Credit Valuation Adjustment Estimation

To calculate Monte Carlo estimate of the credit value adjustment, we calculate the average loss of the 1000s of sample price paths, which happens when we both see a gain in the option with hold and when the counterparty firm value is less than its debt.

As per notes, we assume that default can only occur at time  $T$ , and firm defaults if the firm value is below firm debt amount

```
[17]: def terminal_value(S_0, risk_free_rate, sigma, Z, T): #applies to both firm and
      ↪ stock
```

```

    return S_0 * np.exp((risk_free_rate - sigma**2/2) * T + sigma * np.sqrt(T) *
→Z)

```

```

[18]: cva_estimate = []
      cva_std = []

      for sample_size, paths in share_price_paths.items():
          payoffs = payoff(paths, K, L)
          term_firm_vals = firm_val_paths[sample_size].iloc[-1].to_numpy()
          amount_lost = np.exp(-risk_free*T)*(1-recovery_rate)*(term_firm_vals <
→debt)*payoffs
          cva_estimate.append(amount_lost.mean())
          cva_std.append(amount_lost.std()/np.sqrt(sample_size))

```

Credit value adjustment at various sample sizes:

```

[19]: for i in range(len(cva_estimate)):
      print("Sample size: {}, CVA: {:.3f}".format((i+1)*1000,cva_estimate[i]))

```

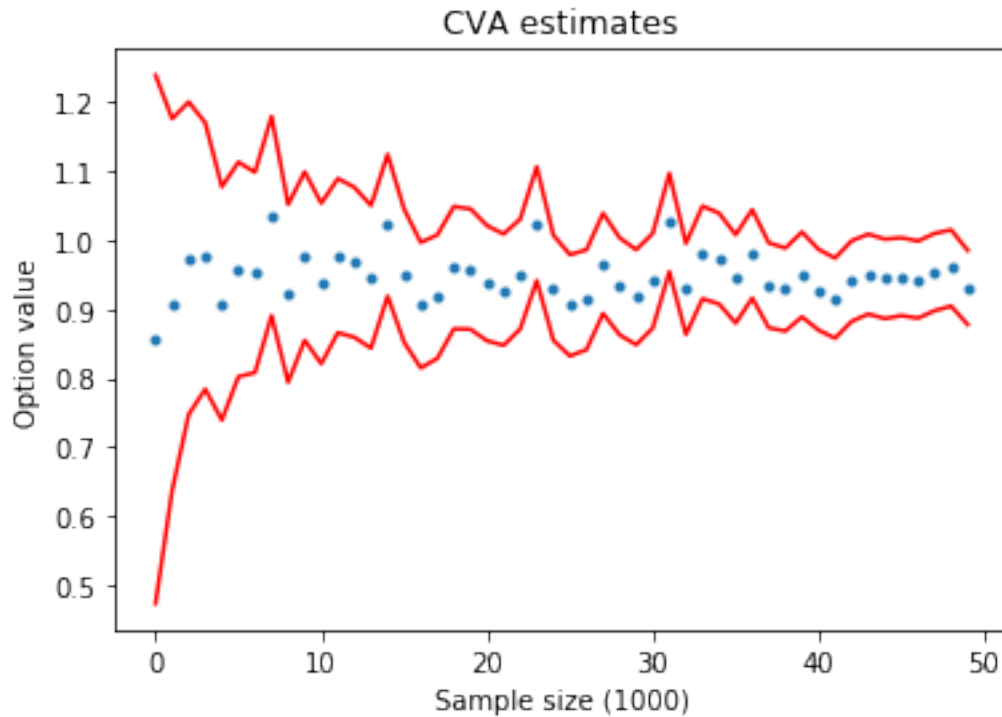
```

Sample size: 1000, CVA: 0.856
Sample size: 2000, CVA: 0.906
Sample size: 3000, CVA: 0.974
Sample size: 4000, CVA: 0.977
Sample size: 5000, CVA: 0.908
Sample size: 6000, CVA: 0.957
Sample size: 7000, CVA: 0.953
Sample size: 8000, CVA: 1.035
Sample size: 9000, CVA: 0.923
Sample size: 10000, CVA: 0.977
Sample size: 11000, CVA: 0.937
Sample size: 12000, CVA: 0.977
Sample size: 13000, CVA: 0.968
Sample size: 14000, CVA: 0.947
Sample size: 15000, CVA: 1.021
Sample size: 16000, CVA: 0.948
Sample size: 17000, CVA: 0.906
Sample size: 18000, CVA: 0.918
Sample size: 19000, CVA: 0.960
Sample size: 20000, CVA: 0.958
Sample size: 21000, CVA: 0.937
Sample size: 22000, CVA: 0.928
Sample size: 23000, CVA: 0.951
Sample size: 24000, CVA: 1.024
Sample size: 25000, CVA: 0.931
Sample size: 26000, CVA: 0.905
Sample size: 27000, CVA: 0.913

```

Sample size: 28000, CVA: 0.966  
Sample size: 29000, CVA: 0.933  
Sample size: 30000, CVA: 0.917  
Sample size: 31000, CVA: 0.941  
Sample size: 32000, CVA: 1.025  
Sample size: 33000, CVA: 0.929  
Sample size: 34000, CVA: 0.982  
Sample size: 35000, CVA: 0.973  
Sample size: 36000, CVA: 0.944  
Sample size: 37000, CVA: 0.980  
Sample size: 38000, CVA: 0.934  
Sample size: 39000, CVA: 0.928  
Sample size: 40000, CVA: 0.950  
Sample size: 41000, CVA: 0.928  
Sample size: 42000, CVA: 0.916  
Sample size: 43000, CVA: 0.941  
Sample size: 44000, CVA: 0.951  
Sample size: 45000, CVA: 0.944  
Sample size: 46000, CVA: 0.947  
Sample size: 47000, CVA: 0.943  
Sample size: 48000, CVA: 0.954  
Sample size: 49000, CVA: 0.959  
Sample size: 50000, CVA: 0.931

```
[20]: plt.plot(cva_estimate, '.')  
plt.plot(cva_estimate + 3 * np.array(cva_std), 'r')  
plt.plot(cva_estimate - 3 * np.array(cva_std), 'r')  
plt.title("CVA estimates")  
plt.xlabel("Sample size (1000)")  
plt.ylabel("Option value")  
plt.show()
```



## 4 Incorporating Counterparty Risk

We calculate the Monte Carlo estimates for the price of the option incorporating counterparty risk, given by the default-free price less the CVA.

```
[21]: option_cva_adjusted_prices = []
      option_cva_adjusted_std = []

      for sample_size, paths in share_price_paths.items():
          payoffs = payoff(paths, K, L)
          option_price = np.exp(-risk_free*T)*payoffs

          term_firm_vals = firm_val_paths[sample_size].iloc[-1].to_numpy()
          amount_lost = np.exp(-risk_free*T)*(1-recovery_rate)*(term_firm_vals <
          ↳debt)*payoffs

          option_cva_price = option_price - amount_lost

          option_cva_adjusted_prices.append(option_cva_price.mean())
          option_cva_adjusted_std.append(option_cva_price.std()/np.sqrt(sample_size))
```

Credit value adjustment at various sample sizes:

```
[22]: for i in range(len(option_cva_adjusted_prices)):
        print("Sample size: {}, CVA-adjusted option value: {:.3f}".
              ↪format((i+1)*1000,option_cva_adjusted_prices[i]))
```

```
Sample size: 1000, CVA-adjusted option value: 5.997
Sample size: 2000, CVA-adjusted option value: 5.920
Sample size: 3000, CVA-adjusted option value: 5.857
Sample size: 4000, CVA-adjusted option value: 5.763
Sample size: 5000, CVA-adjusted option value: 5.643
Sample size: 6000, CVA-adjusted option value: 5.913
Sample size: 7000, CVA-adjusted option value: 5.760
Sample size: 8000, CVA-adjusted option value: 5.993
Sample size: 9000, CVA-adjusted option value: 5.490
Sample size: 10000, CVA-adjusted option value: 5.834
Sample size: 11000, CVA-adjusted option value: 5.894
Sample size: 12000, CVA-adjusted option value: 5.726
Sample size: 13000, CVA-adjusted option value: 5.715
Sample size: 14000, CVA-adjusted option value: 5.814
Sample size: 15000, CVA-adjusted option value: 5.597
Sample size: 16000, CVA-adjusted option value: 5.745
Sample size: 17000, CVA-adjusted option value: 5.788
Sample size: 18000, CVA-adjusted option value: 5.760
Sample size: 19000, CVA-adjusted option value: 5.620
Sample size: 20000, CVA-adjusted option value: 5.720
Sample size: 21000, CVA-adjusted option value: 5.788
Sample size: 22000, CVA-adjusted option value: 5.659
Sample size: 23000, CVA-adjusted option value: 5.803
Sample size: 24000, CVA-adjusted option value: 5.760
Sample size: 25000, CVA-adjusted option value: 5.739
Sample size: 26000, CVA-adjusted option value: 5.751
Sample size: 27000, CVA-adjusted option value: 5.799
Sample size: 28000, CVA-adjusted option value: 5.740
Sample size: 29000, CVA-adjusted option value: 5.815
Sample size: 30000, CVA-adjusted option value: 5.695
Sample size: 31000, CVA-adjusted option value: 5.678
Sample size: 32000, CVA-adjusted option value: 5.845
Sample size: 33000, CVA-adjusted option value: 5.809
Sample size: 34000, CVA-adjusted option value: 5.762
Sample size: 35000, CVA-adjusted option value: 5.709
Sample size: 36000, CVA-adjusted option value: 5.756
Sample size: 37000, CVA-adjusted option value: 5.693
Sample size: 38000, CVA-adjusted option value: 5.760
Sample size: 39000, CVA-adjusted option value: 5.750
Sample size: 40000, CVA-adjusted option value: 5.726
Sample size: 41000, CVA-adjusted option value: 5.853
Sample size: 42000, CVA-adjusted option value: 5.707
Sample size: 43000, CVA-adjusted option value: 5.673
```

Sample size: 44000, CVA-adjusted option value: 5.724  
 Sample size: 45000, CVA-adjusted option value: 5.732  
 Sample size: 46000, CVA-adjusted option value: 5.793  
 Sample size: 47000, CVA-adjusted option value: 5.789  
 Sample size: 48000, CVA-adjusted option value: 5.753  
 Sample size: 49000, CVA-adjusted option value: 5.689  
 Sample size: 50000, CVA-adjusted option value: 5.697

```

[23]: plt.plot(option_cva_adjusted_prices, '.')
plt.plot(option_cva_adjusted_prices + 3 * np.array(option_cva_adjusted_std), 'r')
plt.plot(option_cva_adjusted_prices - 3 * np.array(option_cva_adjusted_std), 'r')
plt.title("CVA-adjusted option prices estimates")
plt.xlabel("Sample size (1000)")
plt.ylabel("Option value")
plt.show()

```



## 5 Conclusion

We are well aware that Black-Scholes formula was derived under assumptions which might not hold in a real market, as an example, if the share price deviates from the geometric Brownian motion, then our estimates can be wrong by a big margin. However with the assumptions of Black-Scholes-Merton model we were able to price a European Up-and-out Call Option. We were

able to use knowledge from module 2 and 3 to generate Python code for monthly simulations for the lifetime of the option. Also, our Monte Carlo estimates are highly dependant on parameters such as firm value if firm value is well above debt value and the volatility of the firm is low, the chance of default is really low.



## References

- [1] Merton, Robert C. "Theory of rational option pricing." *The Bell Journal of economics and management science* (1973): 141-183.
- [2] Rich, Don R. "The mathematical foundations of barrier option-pricing theory." *Advances in futures and options research* 7 (1994).
- [3] Wong, Hoi Ying, and Yue-ÅKuen Kwok. "Multi-asset barrier options and occupation time derivatives." *Applied Mathematical Finance* 10.3 (2003): 245-266.
- [4] Black, Fisher, and Myron Scholes. "The pricing and Corporate Liabilities." *Journal of Political Economy* 81 (1973).
- [5] Yousuf, M. "A fourth-order smoothing scheme for pricing barrier options under stochastic volatility." *International Journal of Computer Mathematics* 86.6 (2009): 1054-1067.