# IE 420 Final Project

Julius Olson

April 2020

## Question 1

The implementation was done in `C++` to ensure efficient calculations. To make the program more easily readable and to reduce sources of error, the function signature presented in the assignment was altered to `binomial(option &opt)`, i.e. a reference to an option struct is sent to the function. The option struct contains (see Appendix A.3) all parameters needed by the CRR binomial function.

## Question 2

$$\begin{aligned}
&T = 1, K = 100,\ S_0 = 100,\ r = 0.05,\ q = 0.04,\ \sigma = 0.2\\
&d_1 = \frac{\ln(S_0/K) + (r - q + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}\\
&d_2 = d_1 - \sigma\sqrt{T}\\
&c_{BS} = S_0 e^{-qT}\mathcal{N}_{cdf}(d_1) - Ke^{-rT}\mathcal{N}_{cdf}(d_2)
\end{aligned}$$

The price was calculated using the CRR binomial model with the number of steps $N$ increasing until the difference between the CRR price and the Black & Scholes price was less then $10^{-3}$. Figure 1 depicts how the model converges towards the value obtained using the Black-Scholes formula as the value of $N$ is sufficiently large. At $N = 1885$ the difference was within the required accuracy and the prices were calculated as:

$$c_{BS} = \$8.102643534,\ c_{CRR} = \$8.103643071$$

. The execution time was found to be of magnitude $\mathcal{O}(N^2)$.

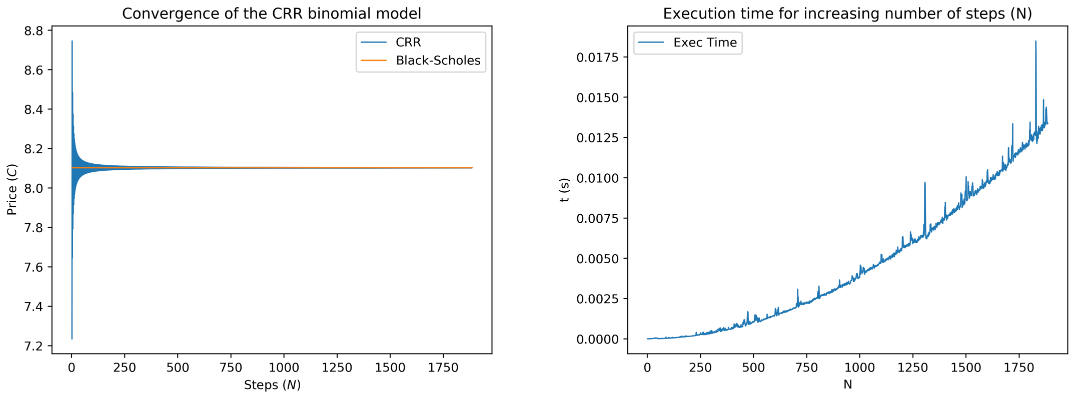



Figure 1: CRR binomial option price as a function of number of steps $N$. The convergence is easily observable as $N$ grows.

# Question 3

This question concerns pricing of an american put option with values $K = \$100.0$, $\sigma = 0.2$, $r = 5\%$. As the black & scholes model is unapplicable on american options, the absolute error between the price calculated with $N$ steps and $N - 1$ steps was used to make sure the required accuracy of $10^{-3}$ was met using the CRR binomial model. As indicated in the plot below, a maximum of approximately 2200 steps was needed to ensure the accuracy when time to maturity was set to 1 year. Therefore, $N = 2250$ was used in the subsequent calculations in this question. The time complexity for increasing number of steps was found to be of magnitude $\mathcal{O}(N^2)$.

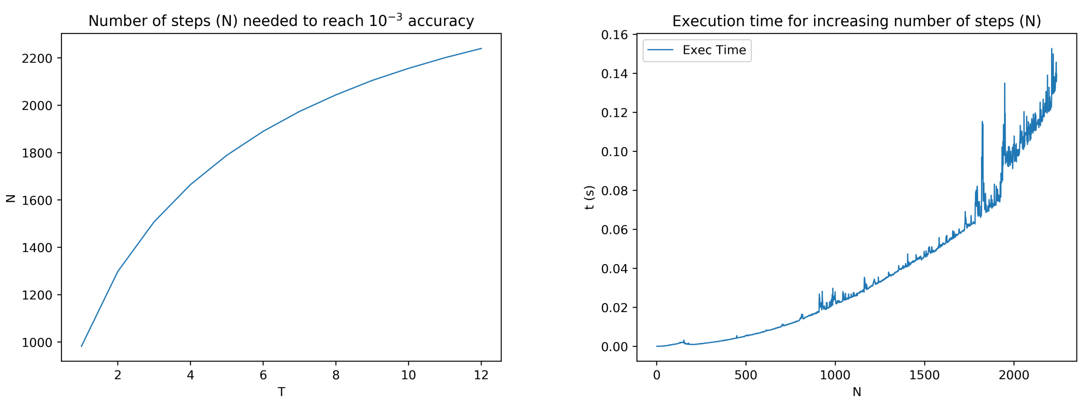



Figure 2: To the left the number of steps needed to ensure an accuracy of $10^{-3}$ is shown as a function of $T$. To the right, the execution time as a function of the number of steps is shown.

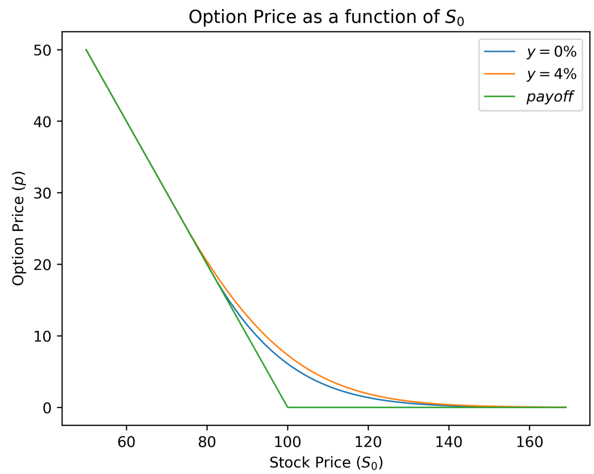



Figure 3: CRR binomial put price as a function of inital stock price $S_0$.

Figure 3 shows that when the initial stock price increases, the option price of the american put option also decreases. The strike price of the option is \$100, i.e gives the holder the right to sell a share of the underlying asset for \$100, and thus the option is less valuable for the holder the higher the initial stock price is and is thus prices lower. For low values of $S_0$ the price is equivalent to the the option intrinsic value due to the ability to exercise early.

When the dividend yield increases, the price of the put option is higher. This is due to the fact that the value of the underlying asset decreases with the value of the dividend after the dividend date. As such the put option, giving the right to sell the asset at the strike price, increases in value when the dividend is increased and should thus be priced higher. This does however not hold for small values of $S_0$ as the option then is exercised earlier and thus still is priced according to the intrinsic value.

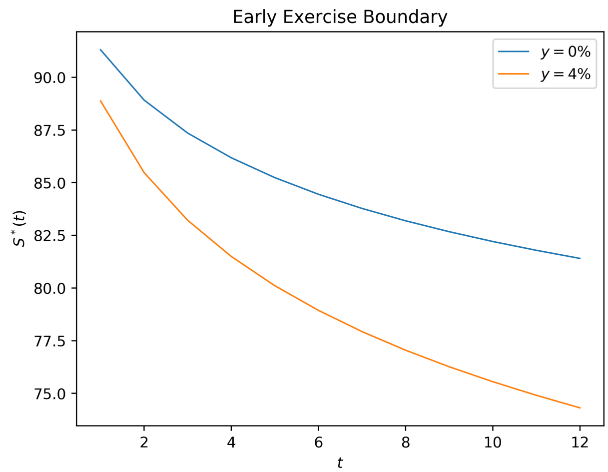



Figure 4: The critical price $S^*(t)$ as a function of time to maturity $t$ ranging from 1 month to 1 year.

| T | $S^*$ $(q = 0\%)$ | $S^*$ $(q = 4\%)$ |
|---|---|---|
| 1.0 | 91.3068 | 88.88 |
| 2.0 | 88.9181 | 85.4771 |
| 3.0 | 87.3533 | 83.2141 |
| 4.0 | 86.1824 | 81.4987 |
| 5.0 | 85.2383 | 80.1091 |
| 6.0 | 84.4505 | 78.9409 |
| 7.0 | 83.7774 | 77.9327 |
| 8.0 | 83.1916 | 77.0503 |
| 9.0 | 82.6741 | 76.2648 |
| 10.0 | 82.2084 | 75.5554 |
| 11.0 | 81.7902 | 74.9102 |
| 12.0 | 81.4064 | 74.3158 |

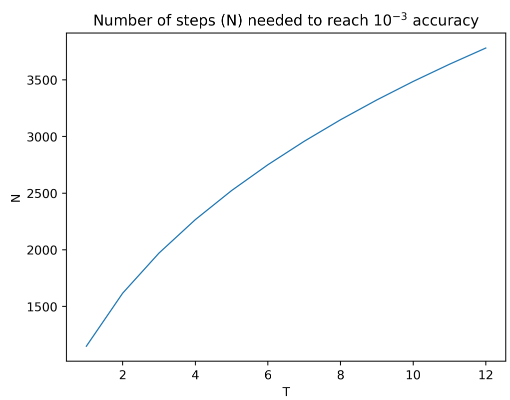
Table 1: asd

For time to maturity ranging from 1 to 12 months, the critical price $S^*$ was numerically approximated by calculating the difference between the option price and the intrinsic value of the option for decreasing values of $S_0$. This was done with a step size of 1 until an lower bound was found at which the difference exceeded 0.005. The interval was then updated and the step size was divided by 10. The process was repeated until the step size was 0.0001. At this point the approximation of $S^*$, i.e. the highest value of the stock price that provided a difference less than 0.005, was well within the required tolerance. The results are presented in the plot and table above. (See Appendix A.4 for more details on the algorithm used)

Figure 4 and table 1 show the critical prices $S^*$ on the early exercise boundary as a function of time to maturity t (1-12 months). As the yield is increased from 0% to 4% the corresponding critical prices decrease. The reason for this is that the increase in dividend yield decreases the future price of the underlying asset, and thus makes it more profitable to hold the put option as time progresses. Because of this the time value of the option increases and thus the intrinsic value of the option has to be higher in order for a early exercise to be optimal. The intrinsic value increases as the stock price decreases and therefore the critical prices on the early exercise boundary are lower.

## Question 4

This question concerns pricing of an american call option with values $K = \$100.0,\ \sigma = 0.2,\ r = 5\%$. As the black & scholes model is unapplicable on american options, the absolute error between the price calculated with $N$ steps and $N-1$ steps was used to make sure the required accuracy of $10^{-3}$ was met using the CRR binomial model. As indicated in the plot below, a maximum of approximately 3780 steps was needed to ensure the accuracy when time to maturity was set to 1 year. Therefore, $N = 3800$ was used in the subsequent calculations in this question. The time complexity for increasing number of steps was found to be of magnitude $\mathcal{O}(N^2)$.



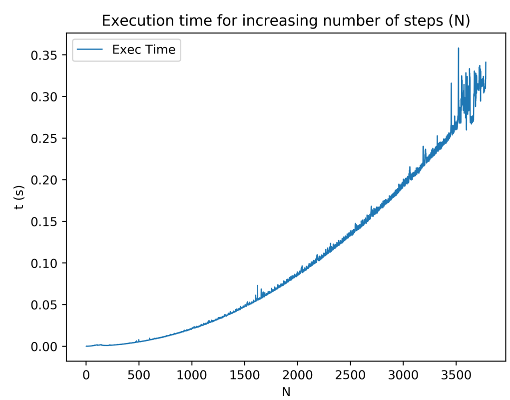




Figure 5: To the left the number of steps needed to ensure an accuracy of $10^-3$ is shown as a function of $T$. To the right, the execution time as a function of the number of steps is shown.

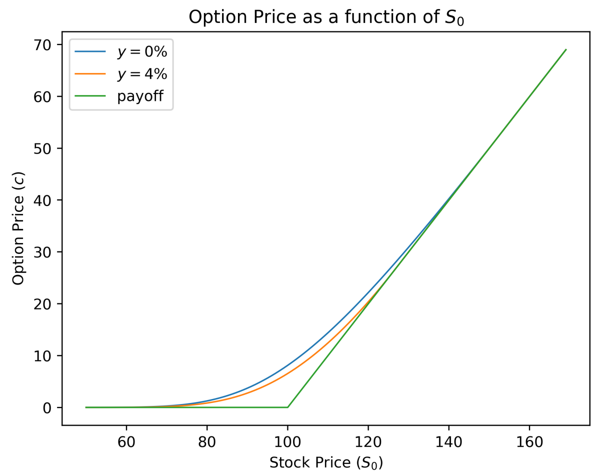



Figure 6: CRR binomial call price as a function of initial stock price $S_0$

Figure 6 shows that the price of the American call option increases as $S_0$ increases. The contract gives the holder the right to buy the underlying asset at strike price $K = \$100$, and thus the option increases in value as the inital price of the underlying asset $S_0$ increases. For high values of $S_0$, the option price converges towards the intrinsic value of the underlying option as the ability to exercise early comes into play.

When increasing the dividend yield (from 4% to 8%), the value of the option decreases. This is due to the fact that the value of the underlying option decreases with the dividend amount after the dividend date. When this occurs, the ability to buy the asset at the strike price is less valuable, and thus the option is priced lower. As for the put option in Question 3, the price difference goes towards zero for values of $S_0$ that enables early exercise (high values of $S_0$ in this case).

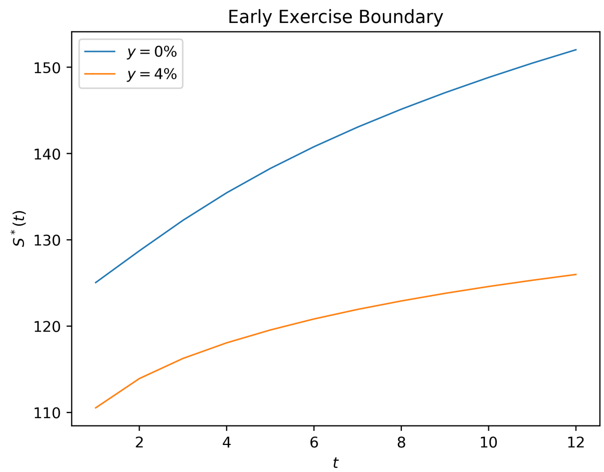



Figure 7: The critical price $S^*(t)$ as a function of time to maturity $t$ ranging from 1 month to 1 year.

| T | $S^*$ ($q = 4\%$) | $S^*$ ($q = 8\%$) |
|---|---|---|
| 1 | 125.0543 | 110.536 |
| 2 | 128.7319 | 113.928 |
| 3 | 132.2693 | 116.2616 |
| 4 | 135.4561 | 118.0681 |
| 5 | 138.2776 | 119.5583 |
| 6 | 140.7946 | 120.8348 |
| 7 | 143.0679 | 121.9429 |
| 8 | 145.1413 | 122.9212 |
| 9 | 147.0523 | 123.8013 |
| 10 | 148.8218 | 124.5968 |
| 11 | 150.4804 | 125.3194 |
| 12 | 152.032 | 125.9898 |

Table 2: asdasd

For time to maturity ranging from 1 to 12 months, the critical price $S^*$ was numerically approximated by calculating the difference between the option price and the intrinsic value of the option for increasing values of $S_0$. This was done with a step size of 1 until an upper bound was found at which the difference exceeded 0.005. The interval was then updated and the step size was divided by 10. The process was repeated until the step size was 0.0001. At this point the approximation of $S^*$, i.e. the lowest value of the stock price that provided a difference less than 0.005, was well within the required tolerance. The results are presented in the plot and table above. (See appendix A.4 for more details)

Figure 7 and table 7 show the critical prices for the call option as a function of time to maturity (1-12 months). When increasing the dividend yield to 8%, the call option decreases in value for the holder due to the fact that the underlying asset decreases in value (with the dividend amount). This in turn means that the time value of the call option decreases, and as such the early exercise boundary indicating when it is optimal to exercise the option before maturity moves to a lower level in terms of $S_0$. The reason for this is that a lower stock price and therefore return upon exercising the option is required for an early exercise to be optimal.

# A Code Appendix

## A.1 CRR Binomial Code

```
/*
  IE420 Project
  Julius Olson
  Binomial
*/

#include <iostream>
#include <cmath>
#include "binomial.h"
#include "options.h"

double risk_neutral_prob(double r, double delta, double u, double d) {
  return (exp(r * delta) - d) / (u-d);
}

/*
  Binomial: CRR pricing algorithm
  Input: option struct
  Output: option price and execution time
*/

binomial_res binomial(option &opt) {
  binomial_res res;
  clock_t start, end;

  start = clock();
  double delta, u, d, p;

  delta = opt.T / (double) opt.N;

  // Crr - Calculate u and d using volatility
  u = exp(opt.sigma * sqrt(delta));
  d = exp(-opt.sigma * sqrt(delta));
  p = risk_neutral_prob(opt.r-opt.q, delta, u, d);

  /*
    Payoff at maturity
  */
  double f[opt.N+1];
  for (int j=0; j<opt.N+1; j++) {
    f[j] = pow(u, j) * pow(d, opt.N-j) * opt.S0;
    f[j] = payoff(opt.option, opt.K, f[j]);
  }

  /*
    Backwards induction
  */
  for (int i=opt.N; i>0; --i) {
    for (int j=0; j<i; ++j) {
      f[j] = exp(-opt.r * delta) * ((p * f[j+1]) + (1-p) * f[j]);

      if (opt.exercise == A) {
        double early_exercise = pow(u, j) * pow(d, i - j - 1) * opt.S0;
        early_exercise = payoff(opt.option, opt.K, early_exercise);
        if (early_exercise > f[j]) {
```

```
          f[j] = early_exercise;
        }
      }
    }
  }
  end = clock();
  res.option_price = f[0];
  res.exec_time = (end-start) / (double) CLOCKS_PER_SEC;
  return res;
}
```

### A.2 Black & Scholes Code

```
/*
  IE420 Project
  Julius Olson
  Black & Scholes
*/

#include <cmath>
#include "binomial.h"


double n_cdf(double x) {
  return 0.5 * erfc(-x * sqrt(0.5));
}

double black_scholes(option &opt) {
  double d1, d2;

  d1 = (log(opt.S0 / opt.K) + (opt.r - opt.q + 0.5 * pow(opt.sigma, 2)) * opt.T) / (
    opt.sigma * sqrt(opt.T));
  d2 = d1 - opt.sigma * sqrt(opt.T);

  if (opt.option == C) {
    return opt.S0 * exp(-opt.q * opt.T) * n_cdf(d1) - opt.K * exp(-opt.r * opt.T) *
    n_cdf(d2);
  } else {
    return -opt.S0 * exp(-opt.q * opt.T) * n_cdf(d1) + opt.K * exp(-opt.r * opt.T) *
     n_cdf(d2);
  }
}
```

### A.3 Options Utils Code

```
/*
  IE420 Project
  Julius Olson
  Options utils
*/

#include "options.h"
#define MAX(x, y) (x > y ? x : y);

double payoff(option_type t, double K, double S) {
  if (t == P) {
    return MAX(K-S, 0);
  }
```

```
  return MAX(S-K, 0)
}
```

```
/*
  IE420 Project
  Julius Olson
  Options utils
*/

# ifndef OPTIONS_H

# define OPTIONS_H

enum exercise_type {
  A,
  E
};
enum option_type {
  P,
  C
};

struct option {
  double K;
  double S0;
  double sigma;
  double r;
  double q;
  int N;
  double T;
  option_type option;
  exercise_type exercise;
};

double payoff(option_type t, double K, double S);

#endif
```

## A.4 Critical Price Code

```
/*
  IE420 Project
  Julius Olson
  Critical Price
*/

#include <cmath>
#include "options.h"
#include "binomial.h"
#include "critical.h"
#include <iostream>
#include <iomanip>

using namespace std;

criticalRes criticalPrice(option &opt, double lowerbound, double higherbound) {
  double step, diff, price;
  criticalRes res;
```

```

/*
  For call options - start with low S0 increase until diff < 0.005.
  Then decrease step size and repeat until within accuracy.
  Finds lowest S0 that satisfies the condition that the difference between option
  price
  and intrinsic value is less than 0.005
*/
if (opt.option == C) {
  step = 1.0;
  opt.S0 = lowerbound;
  while (opt.S0 < higherbound) {
    price = binomial(opt).option_price;
    diff = abs(price - payoff(opt.option, opt.K, opt.S0));
    if (diff < 0.005) {
      if (step == 0.0001) {
        // Sufficient accuracy achieved => break
        break;
      }
      opt.S0 -= step;
      step /= 10.0;
    }
    else {
      opt.S0 += step;
    }
  }
  res.criticalPrice = opt.S0;
  res.diff = diff;
  return res;
}

/*
  For put options - Start with high S0 and decrease until diff < 0.005
  Then decrease step size and repeat until within accuracy.
  Finds highest S0 that satisfies the condition that the difference between option
   price
  and intrinsic value is less than 0.005
*/
if (opt.option == P) {
  step = 1.0;
  opt.S0 = higherbound;
  while (opt.S0 > lowerbound) {
    price = binomial(opt).option_price;
    diff = abs(price - payoff(opt.option, opt.K, opt.S0));
    if (diff < 0.005) {
      if (step == 0.0001) {
        // Sufficient accuracy achieved => break
        break;
      }
      opt.S0 += step;
      step /= 10.0;
    } else {
      opt.S0 -= step;
    }
  }
  res.criticalPrice = opt.S0;
  res.diff = diff;
  return res;
```

```
  }

  cout << "Wrong option type" << endl;
  return res;
}
```

```
/*
  IE420 Project
  Julius Olson
  Critical Price
*/
#ifndef CRITICAL_H

#define CRITICAL_H
#include "options.h"
struct criticalRes {
  double criticalPrice;
  double diff;
};
criticalRes criticalPrice(option &opt, double lowerbound, double higherbound);
#endif
```

### A.5 Main Code

```
/*
  IE420 Project
  Julius Olson
  Main
*/


#include <iostream>
#include <cmath>
#include <iomanip>
#include <fstream>
#include "binomial.h"
#include "black_scholes.h"
#include "critical.h"
using namespace std;

void q2 () {
  cout << "Running Q2..." << endl;
  /*
    Define option as struct
  */
  option opt;
  opt.K = 100.0;
  opt.T = 1.0;
  opt.S0 = 100.0;
  opt.sigma = 0.2;
  opt.r = 0.05;
  opt.q = 0.04;
  opt.N = 10;
  opt.exercise = E;
  opt.option = C;

  ofstream outfile, timeFile;
  binomial_res crr;
```

```
  double bs;

  /*
    Open file for results
  */
  outfile.open("outputs/q2.csv");
  outfile << "N,CRR,BS\n";
  timeFile.open("outputs/q2_time.csv");
  timeFile << "N,t" << endl;
  /*
    - Calculate price using black_scholes
    - Calculate price with CRR with different values for N
  */
  bs = black_scholes(opt);
  double diff = 1.0;
  opt.N = 2;
  while (diff > 1e-3) {
    if (opt.N % 100 == 0) {
      cout << opt.N << ", " << diff << endl;
    }
    crr = binomial(opt);
    outfile << setprecision(10) << opt.N << "," << crr.option_price << "," << bs << endl;
    timeFile << setprecision(10) << opt.N << "," << crr.exec_time << endl;
    diff = abs(crr.option_price - bs);
    ++opt.N;
  }
  outfile.close();
  timeFile.close();
}

/*
  Calculate the option price using the CRR binomial model as a function of initial stock price(S0)
*/
void priceFunctionOfS0 (option &opt, ofstream &outfile) {
  cout << "Calculating Price as function of S0 for q=" << opt.q << " and t=" << opt.T << endl;
  binomial_res crr;
  outfile << "S0,Price,intrinsic\n";
  for (int S0 = 50; S0 < 170; ++S0) {
    opt.S0 = double(S0);
    crr = binomial(opt);
    outfile << setprecision(10) << S0 << "," << crr.option_price << "," << payoff(opt.option, opt.K, S0) << "\n";
  }
}

/*
  Find the critical price for an option with maturities from 1 month to a year.
*/
void findCriticalPrices(option &opt, ofstream &outfile, double lowerbound, double higherbound) {
  outfile << "t,S_star,diff" << endl;
  for (int i = 1; i < 13; ++i) {
    opt.T = ((double)i) / 12.0;
    cout << "Calulating critical price for q=" << opt.q << " and t=" << opt.T << endl;
```

```
    criticalRes critical = criticalPrice(opt, lowerbound, higherbound);
    cout << "Critical: " << critical.criticalPrice << endl;
    outfile << i << "," << critical.criticalPrice << "," << critical.diff << endl;
  }
}

/*
  Find the number of steps (N) required for reaching a absolute error less than 1e-3
*/
void findNSatisfyingAccuracy(option &opt, ofstream &timefile, ofstream &outFile) {
  timefile << "N,t" << endl;
  outFile << "T,N" << endl;

  binomial_res crr;
  opt.N = 2;
  double diff;
  for (int i = 1; i<13; i++){
    opt.T = ((double) i) / 12;
    double old = 0.0;
    diff = 1.0;
    while (diff > 0.001) {
      crr = binomial(opt);
      diff = abs(crr.option_price - old);
      old = crr.option_price;
      timefile << opt.N << "," << crr.exec_time << endl;
      ++opt.N;
    }
    cout << "Diff: " << diff << " N: " << opt.N << " T: " << opt.T << "\n";
    outFile << i << "," << opt.N << endl;
  }
}

void q3() {
  cout << "Running Q3" << endl;
  option opt;
  opt.option = P;
  opt.K = 100.0;
  opt.sigma = 0.2;
  opt.r = 0.05;
  opt.q = 0.0;
  opt.S0 = 100.0;
  opt.exercise = A;

  ofstream timeFile;
  ofstream outfile;

  outfile.open("outputs/q3_n.csv");
  timeFile.open("outputs/q3_time.csv");
  findNSatisfyingAccuracy(opt, timeFile, outfile);
  timeFile.close();
  outfile.close();



  opt.N = 2250; // Chosen to maintain accuarcy.
  for (double q = 0.0; q <= 0.04; q+= 0.04) {
    if (q == 0.0) {
      outfile.open("outputs/q3_function_of_S0_0.csv");
```

```
    } else {
      outfile.open("outputs/q3_function_of_S0_4.csv");
    }

    opt.T = 1.0;
    opt.q = q;
    priceFunctionOfS0(opt, outfile);
    outfile.close();


    if (q == 0.0) {
      outfile.open("outputs/q3_critical_0.csv");
    } else {
      outfile.open("outputs/q3_critical_4.csv");
    }

    // Find critical prices using lower and upper bound as identified on the plot
    findCriticalPrices(opt, outfile, 60.0, 95.0);
    outfile.close();

  }
}

void q4() {
  cout << "Running Q4" << endl;

  option opt;
  ofstream outfile;

  opt.K = 100.0;
  opt.S0 = 100.0;
  opt.sigma = 0.2;
  opt.r = 0.05;
  opt.q = 0.04;
  opt.N = 3800;
  opt.option = C;
  opt.exercise = A;

  ofstream timeFile;
  outfile.open("outputs/q4_n.csv");
  timeFile.open("outputs/q4_time.csv");
  findNSatisfyingAccuracy(opt, timeFile, outfile);
  timeFile.close();
  outfile.close();

  opt.N = 2250;
  for (double q = 0.04; q<=0.08; q+=0.04) {
    opt.q = q;
    if (q == 0.04) {
      outfile.open("outputs/q4_function_of_S0_4.csv");
    } else {
      outfile.open("outputs/q4_function_of_S0_8.csv");
    }
    opt.T = 1.0;
    priceFunctionOfS0(opt, outfile);
    outfile.close();


```

```
    if (q == 0.04) {
      outfile.open("outputs/q4_critical_4.csv");
    } else {
      outfile.open("outputs/q4_critical_8.csv");
    }
    findCriticalPrices(opt, outfile, 100.0, 160.0);
    outfile.close();
  }

}

int main(int argc, char *argv[]) {
  if (argc < 2) {
    cout << "Please provide question number to run as argument" << endl;
    return 0;
  }

  int arg = atoi(argv[1]);

  switch (arg) {
  case 2:
    q2();
    break;
  case 3:
    q3();
    break;
  case 4:
    q4();
    break;
  }
  return 0;
}
```