## Part II.

### Question 1.

Presented in the figures below include the implied volatility plots for the given options written on Google stock. We note a decided 'smile' in shape of the implied volatilities as a function of strike price. That is, contrary to the modelling assumption of constant volatilities, we find variable values for the volatilities as computed by the binomial model.

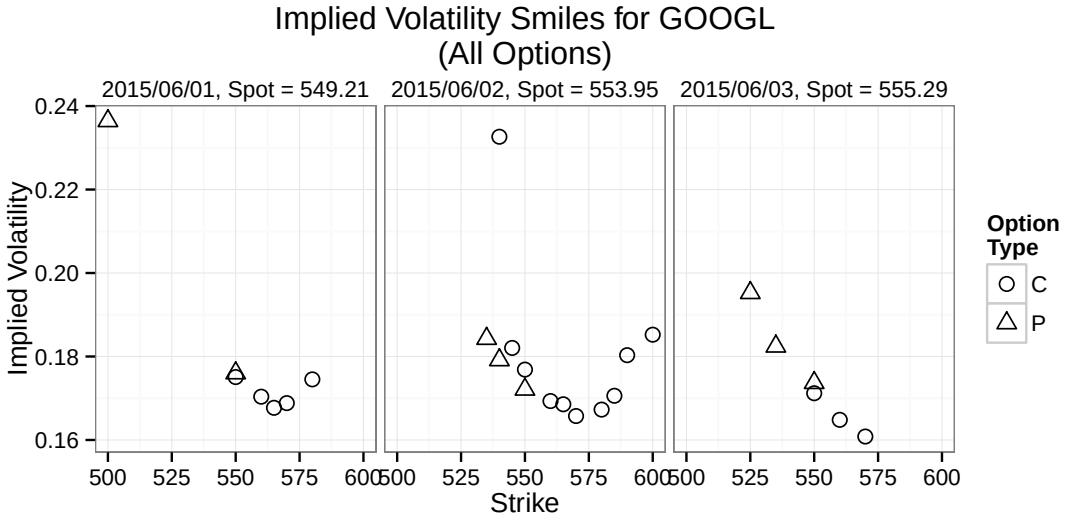


Figure 1: Volatility smile(s) for all options written on GOOGL.

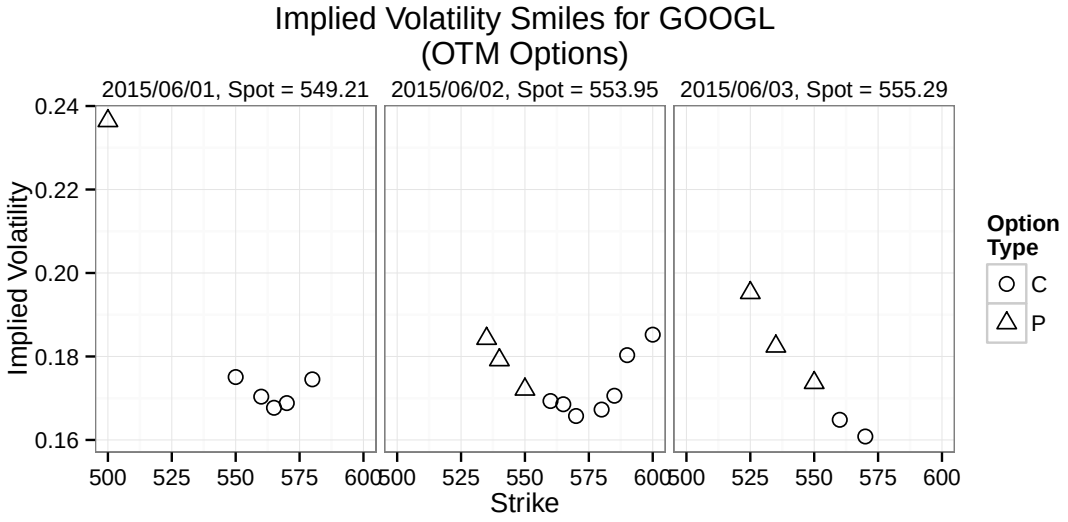


Figure 2: Volatility smile(s) for OTM options written on GOOGL.

If we were so daring as to examine the convergence of the binomial model we do indeed find a trend towards convergence relatively quickly. As illustrated by the figures below, given a four sampled options from the set of 26, by $N = 10^3$ we see little appreciable gain in volatility estimates.

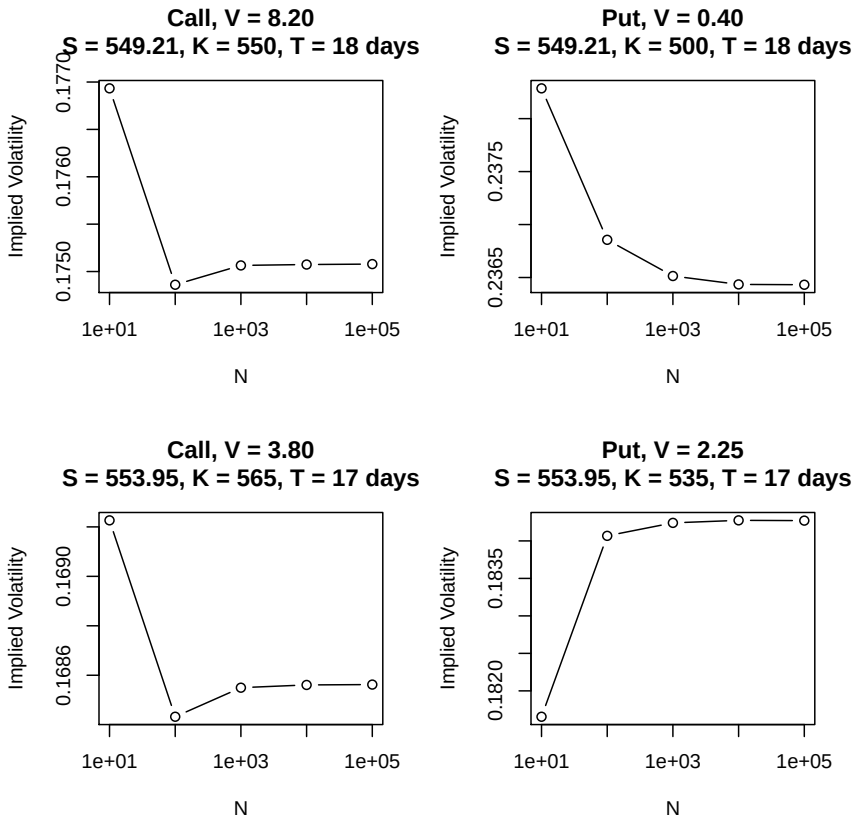


Figure 3: Implied volatility estimates given call option strike prices as a function of the $N$-steps of the binomial model. Qualitatively, convergence appears to occur relatively rapidly: By $N = 10^3$ little gains to volatility precision occurs.

However, examining the volatilities in question at such large intervals obfuscates some interesting behaviour of the binomial model. If we consider the model output while increasing the number of steps one at a time we find periodicity in our volatility estimates. For example, consider the figures below illustrating the first option in our list: A European call option with ask 8.20, spot price 549.21, strike price 550, with 18 days to expiry.

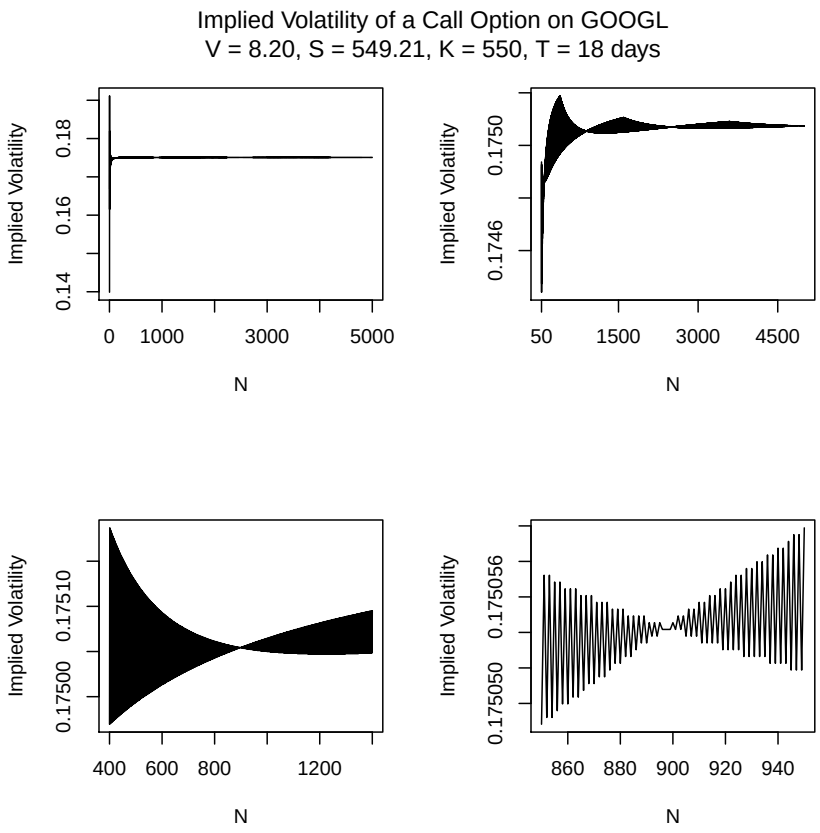


Figure 4: Periodic behaviour seen in the values of the implied volatility as computed for the first option in our data set by the bisection algorithm.

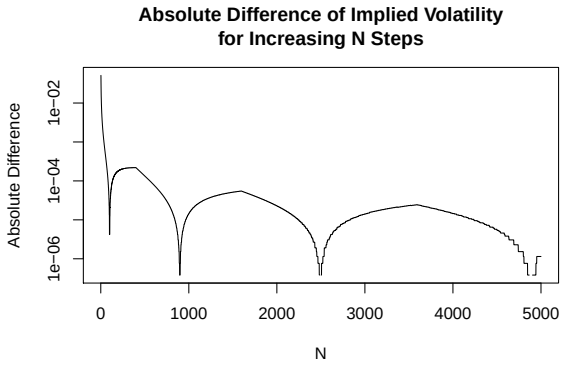


Figure 5: Increasing the value of $N$ leads to periodic behaviour in the additional precision for estimates of implied volatility, with a decreasing trend. Stepwise behaviour for large $N$ can be attributed to the cut-off of $|V_{obs} - V_{imp}| < 10^{-5}$ in the bisection and not the underlying model.

To be sure that such periodicity is not attributable to the bisection algorithm, or some source other than the binomial model, we should also examine the output of the pricing algorithm a single step at a time. As an example, we consider the first option in our list:

The same European call option with ask 8.20, spot price 549.21, strike price 550, with 18 days to expiry. To price an option we require a value for $\sigma$ and so for the sake of illustration we also assume a "true" value for the volatility parameter $\sigma$ to be 0.17507.

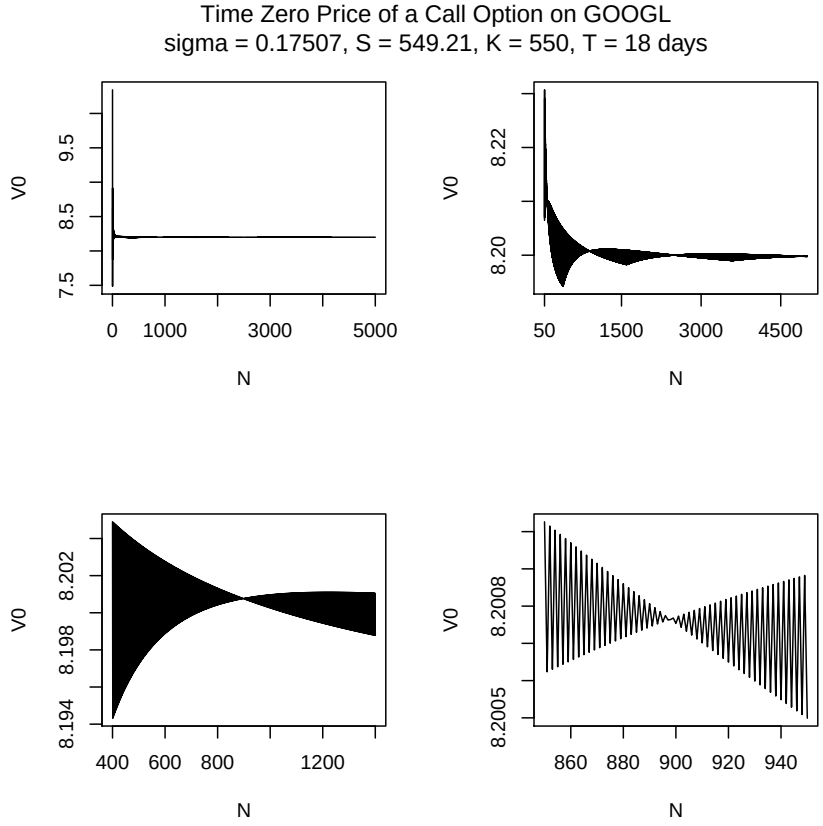


Figure 6: Periodic behaviour seen in the values of the prices as computed for the first option in our data set by the binomial model.

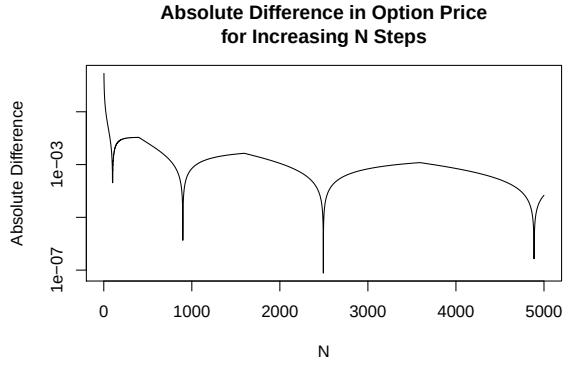


Figure 7: Increasing the value of $N$ leads to periodic behaviour in the additional precision for the prices given by the binomial model, with a decreasing trend.

# Appendix A Code Output: Data Table

| N | spots | strike | tau | type | ask | vol | time |
|---|---|---|---|---|---|---|---|
| 10 | 549.21 | 550 | 0.0493151 | C | 8.2 | 0.176933 | 3.9e-05 |
| 10 | 549.21 | 560 | 0.0493151 | C | 4.1 | 0.170029 | 3.7e-05 |
| 10 | 549.21 | 565 | 0.0493151 | C | 2.7 | 0.16742 | 4.7e-05 |
| 10 | 549.21 | 570 | 0.0493151 | C | 1.8 | 0.166173 | 3.4e-05 |
| 10 | 549.21 | 580 | 0.0493151 | C | 0.8 | 0.177043 | 3.5e-05 |
| 10 | 549.21 | 500 | 0.0493151 | P | 0.4 | 0.238286 | 3.5e-05 |
| 10 | 549.21 | 550 | 0.0493151 | P | 8.9 | 0.177883 | 4.3e-05 |
| 10 | 553.95 | 540 | 0.0465753 | C | 19.4 | 0.230424 | 3.6e-05 |
| 10 | 553.95 | 545 | 0.0465753 | C | 13.9 | 0.178366 | 3.2e-05 |
| 10 | 553.95 | 550 | 0.0465753 | C | 10.6 | 0.174327 | 3.4e-05 |
| 10 | 553.95 | 560 | 0.0465753 | C | 5.5 | 0.165306 | 3.3e-05 |
| 10 | 553.95 | 565 | 0.0465753 | C | 3.8 | 0.169226 | 3.8e-05 |
| 10 | 553.95 | 570 | 0.0465753 | C | 2.45 | 0.164458 | 3.7e-05 |
| 10 | 553.95 | 580 | 0.0465753 | C | 1 | 0.172824 | 3.7e-05 |
| 10 | 553.95 | 585 | 0.0465753 | C | 0.65 | 0.17106 | 3.2e-05 |
| 10 | 553.95 | 590 | 0.0465753 | C | 0.5 | 0.17981 | 3.6e-05 |
| 10 | 553.95 | 600 | 0.0465753 | C | 0.2 | 0.197235 | 3.1e-05 |
| 10 | 553.95 | 535 | 0.0465753 | P | 2.25 | 0.181655 | 4.4e-05 |
| 10 | 553.95 | 540 | 0.0465753 | P | 3.2 | 0.184056 | 3.4e-05 |
| 10 | 553.95 | 550 | 0.0465753 | P | 6.3 | 0.169496 | 3.8e-05 |
| 10 | 555.29 | 550 | 0.0438356 | C | 10.9 | 0.167392 | 3.2e-05 |
| 10 | 555.29 | 560 | 0.0438356 | C | 5.6 | 0.161185 | 3.2e-05 |
| 10 | 555.29 | 570 | 0.0438356 | C | 2.4 | 0.160159 | 3.6e-05 |
| 10 | 555.29 | 525 | 0.0438356 | P | 0.85 | 0.201013 | 2.9e-05 |
| 10 | 555.29 | 535 | 0.0438356 | P | 1.8 | 0.179182 | 3.5e-05 |
| 10 | 555.29 | 550 | 0.0438356 | P | 5.6 | 0.169864 | 3.4e-05 |
| 100 | 549.21 | 550 | 0.0493151 | C | 8.2 | 0.174861 | 0.000776 |
| 100 | 549.21 | 560 | 0.0493151 | C | 4.1 | 0.170056 | 0.000611 |
| 100 | 549.21 | 565 | 0.0493151 | C | 2.7 | 0.167812 | 0.000743 |
| 100 | 549.21 | 570 | 0.0493151 | C | 1.8 | 0.16941 | 0.000743 |
| 100 | 549.21 | 580 | 0.0493151 | C | 0.8 | 0.175021 | 0.000672 |
| 100 | 549.21 | 500 | 0.0493151 | P | 0.4 | 0.236856 | 0.000673 |
| 100 | 549.21 | 550 | 0.0493151 | P | 8.9 | 0.175797 | 0.000743 |
| 100 | 553.95 | 540 | 0.0465753 | C | 19.4 | 0.232136 | 0.000779 |
| 100 | 553.95 | 545 | 0.0465753 | C | 13.9 | 0.182533 | 0.000784 |
| 100 | 553.95 | 550 | 0.0465753 | C | 10.6 | 0.176918 | 0.000776 |
| 100 | 553.95 | 560 | 0.0465753 | C | 5.5 | 0.168933 | 0.000775 |
| 100 | 553.95 | 565 | 0.0465753 | C | 3.8 | 0.168432 | 0.000783 |
| 100 | 553.95 | 570 | 0.0465753 | C | 2.45 | 0.166134 | 0.000779 |
| 100 | 553.95 | 580 | 0.0465753 | C | 1 | 0.167133 | 0.000658 |
| 100 | 553.95 | 585 | 0.0465753 | C | 0.65 | 0.170506 | 0.000713 |
| 100 | 553.95 | 590 | 0.0465753 | C | 0.5 | 0.180766 | 0.000716 |
| 100 | 553.95 | 600 | 0.0465753 | C | 0.2 | 0.186349 | 0.000669 |
| 100 | 553.95 | 535 | 0.0465753 | P | 2.25 | 0.184068 | 0.000792 |

| N | spots | strike | tau | type | ask | vol | time |
|---|---|---|---|---|---|---|---|
| 100 | 553.95 | 540 | 0.0465753 | P | 3.2 | 0.179029 | 0.000764 |
| 100 | 553.95 | 550 | 0.0465753 | P | 6.3 | 0.172232 | 0.000791 |
| 100 | 555.29 | 550 | 0.0438356 | C | 10.9 | 0.170963 | 0.000748 |
| 100 | 555.29 | 560 | 0.0438356 | C | 5.6 | 0.1646 | 0.000774 |
| 100 | 555.29 | 570 | 0.0438356 | C | 2.4 | 0.161098 | 0.00074 |
| 100 | 555.29 | 525 | 0.0438356 | P | 0.85 | 0.195326 | 0.000718 |
| 100 | 555.29 | 535 | 0.0438356 | P | 1.8 | 0.182423 | 0.000703 |
| 100 | 555.29 | 550 | 0.0438356 | P | 5.6 | 0.173479 | 0.000707 |
| 1000 | 549.21 | 550 | 0.0493151 | C | 8.2 | 0.175064 | 0.050889 |
| 1000 | 549.21 | 560 | 0.0493151 | C | 4.1 | 0.170354 | 0.043909 |
| 1000 | 549.21 | 565 | 0.0493151 | C | 2.7 | 0.167709 | 0.043309 |
| 1000 | 549.21 | 570 | 0.0493151 | C | 1.8 | 0.168856 | 0.044952 |
| 1000 | 549.21 | 580 | 0.0493151 | C | 0.8 | 0.174512 | 0.03246 |
| 1000 | 549.21 | 500 | 0.0493151 | P | 0.4 | 0.236514 | 0.04254 |
| 1000 | 549.21 | 550 | 0.0493151 | P | 8.9 | 0.176002 | 0.047771 |
| 1000 | 553.95 | 540 | 0.0465753 | C | 19.4 | 0.232622 | 0.045503 |
| 1000 | 553.95 | 545 | 0.0465753 | C | 13.9 | 0.182027 | 0.044222 |
| 1000 | 553.95 | 550 | 0.0465753 | C | 10.6 | 0.176851 | 0.049084 |
| 1000 | 553.95 | 560 | 0.0465753 | C | 5.5 | 0.169344 | 0.047043 |
| 1000 | 553.95 | 565 | 0.0465753 | C | 3.8 | 0.16855 | 0.048993 |
| 1000 | 553.95 | 570 | 0.0465753 | C | 2.45 | 0.165744 | 0.042694 |
| 1000 | 553.95 | 580 | 0.0465753 | C | 1 | 0.167281 | 0.048292 |
| 1000 | 553.95 | 585 | 0.0465753 | C | 0.65 | 0.170593 | 0.037344 |
| 1000 | 553.95 | 590 | 0.0465753 | C | 0.5 | 0.180357 | 0.044509 |
| 1000 | 553.95 | 600 | 0.0465753 | C | 0.2 | 0.185284 | 0.041304 |
| 1000 | 553.95 | 535 | 0.0465753 | P | 2.25 | 0.184241 | 0.037318 |
| 1000 | 553.95 | 540 | 0.0465753 | P | 3.2 | 0.17914 | 0.043963 |
| 1000 | 553.95 | 550 | 0.0465753 | P | 6.3 | 0.172108 | 0.041886 |
| 1000 | 555.29 | 550 | 0.0438356 | C | 10.9 | 0.171241 | 0.047301 |
| 1000 | 555.29 | 560 | 0.0438356 | C | 5.6 | 0.164866 | 0.050523 |
| 1000 | 555.29 | 570 | 0.0438356 | C | 2.4 | 0.160804 | 0.046303 |
| 1000 | 555.29 | 525 | 0.0438356 | P | 0.85 | 0.195227 | 0.037083 |
| 1000 | 555.29 | 535 | 0.0438356 | P | 1.8 | 0.182427 | 0.047123 |
| 1000 | 555.29 | 550 | 0.0438356 | P | 5.6 | 0.173728 | 0.049172 |
| 10000 | 549.21 | 550 | 0.0493151 | C | 8.2 | 0.175074 | 4.71119 |
| 10000 | 549.21 | 560 | 0.0493151 | C | 4.1 | 0.170379 | 4.69071 |
| 10000 | 549.21 | 565 | 0.0493151 | C | 2.7 | 0.167714 | 4.89059 |
| 10000 | 549.21 | 570 | 0.0493151 | C | 1.8 | 0.168842 | 3.96676 |
| 10000 | 549.21 | 580 | 0.0493151 | C | 0.8 | 0.174532 | 4.45607 |
| 10000 | 549.21 | 500 | 0.0493151 | P | 0.4 | 0.236435 | 4.21464 |
| 10000 | 549.21 | 550 | 0.0493151 | P | 8.9 | 0.176011 | 4.88668 |
| 10000 | 553.95 | 540 | 0.0465753 | C | 19.4 | 0.232652 | 4.65932 |
| 10000 | 553.95 | 545 | 0.0465753 | C | 13.9 | 0.182063 | 4.41524 |
| 10000 | 553.95 | 550 | 0.0465753 | C | 10.6 | 0.176868 | 4.8393 |
| 10000 | 553.95 | 560 | 0.0465753 | C | 5.5 | 0.16933 | 4.01044 |
| 10000 | 553.95 | 565 | 0.0465753 | C | 3.8 | 0.168561 | 4.05048 |
| 10000 | 553.95 | 570 | 0.0465753 | C | 2.45 | 0.165742 | 4.47098 |

| N | spots | strike | tau | type | ask | vol | time |
|---|---|---|---|---|---|---|---|
| 10000 | 553.95 | 580 | 0.0465753 | C | 1 | 0.167287 | 4.70188 |
| 10000 | 553.95 | 585 | 0.0465753 | C | 0.65 | 0.170599 | 4.0263 |
| 10000 | 553.95 | 590 | 0.0465753 | C | 0.5 | 0.180325 | 4.20716 |
| 10000 | 553.95 | 600 | 0.0465753 | C | 0.2 | 0.185236 | 4.03739 |
| 10000 | 553.95 | 535 | 0.0465753 | P | 2.25 | 0.184274 | 4.27669 |
| 10000 | 553.95 | 540 | 0.0465753 | P | 3.2 | 0.179147 | 4.28248 |
| 10000 | 553.95 | 550 | 0.0465753 | P | 6.3 | 0.172112 | 4.72111 |
| 10000 | 555.29 | 550 | 0.0438356 | C | 10.9 | 0.171207 | 4.9435 |
| 10000 | 555.29 | 560 | 0.0438356 | C | 5.6 | 0.164836 | 4.92557 |
| 10000 | 555.29 | 570 | 0.0438356 | C | 2.4 | 0.160832 | 4.94991 |
| 10000 | 555.29 | 525 | 0.0438356 | P | 0.85 | 0.195248 | 4.25358 |
| 10000 | 555.29 | 535 | 0.0438356 | P | 1.8 | 0.182429 | 4.94104 |
| 10000 | 555.29 | 550 | 0.0438356 | P | 5.6 | 0.17369 | 4.6847 |
| 100000 | 549.21 | 550 | 0.0493151 | C | 8.2 | 0.175079 | 446.639 |
| 100000 | 549.21 | 560 | 0.0493151 | C | 4.1 | 0.170382 | 474.247 |
| 100000 | 549.21 | 565 | 0.0493151 | C | 2.7 | 0.167716 | 453.913 |
| 100000 | 549.21 | 570 | 0.0493151 | C | 1.8 | 0.16884 | 432.367 |
| 100000 | 549.21 | 580 | 0.0493151 | C | 0.8 | 0.174525 | 456.261 |
| 100000 | 549.21 | 500 | 0.0493151 | P | 0.4 | 0.236432 | 394.25 |
| 100000 | 549.21 | 550 | 0.0493151 | P | 8.9 | 0.176016 | 475.177 |
| 100000 | 553.95 | 540 | 0.0465753 | C | 19.4 | 0.232656 | 472.296 |
| 100000 | 553.95 | 545 | 0.0465753 | C | 13.9 | 0.182059 | 447.961 |
| 100000 | 553.95 | 550 | 0.0465753 | C | 10.6 | 0.17687 | 475.176 |
| 100000 | 553.95 | 560 | 0.0465753 | C | 5.5 | 0.169334 | 453.966 |
| 100000 | 553.95 | 565 | 0.0465753 | C | 3.8 | 0.168562 | 438.335 |
| 100000 | 553.95 | 570 | 0.0465753 | C | 2.45 | 0.165746 | 462.006 |
| 100000 | 553.95 | 580 | 0.0465753 | C | 1 | 0.167287 | 441.636 |
| 100000 | 553.95 | 585 | 0.0465753 | C | 0.65 | 0.170599 | 394.529 |
| 100000 | 553.95 | 590 | 0.0465753 | C | 0.5 | 0.180324 | 434.661 |
| 100000 | 553.95 | 600 | 0.0465753 | C | 0.2 | 0.185236 | 391.775 |
| 100000 | 553.95 | 535 | 0.0465753 | P | 2.25 | 0.184271 | 473.288 |
| 100000 | 553.95 | 540 | 0.0465753 | P | 3.2 | 0.17915 | 472.246 |
| 100000 | 553.95 | 550 | 0.0465753 | P | 6.3 | 0.172108 | 475.347 |
| 100000 | 555.29 | 550 | 0.0438356 | C | 10.9 | 0.171209 | 475.17 |
| 100000 | 555.29 | 560 | 0.0438356 | C | 5.6 | 0.164837 | 474.334 |
| 100000 | 555.29 | 570 | 0.0438356 | C | 2.4 | 0.160834 | 388.06 |
| 100000 | 555.29 | 525 | 0.0438356 | P | 0.85 | 0.195245 | 388.702 |
| 100000 | 555.29 | 535 | 0.0438356 | P | 1.8 | 0.182426 | 453.996 |
| 100000 | 555.29 | 550 | 0.0438356 | P | 5.6 | 0.173694 | 366.163 |

# Appendix B Code

## B.1 main.cpp

```
#include <iostream>
#include <iomanip> // more decimal precision
#include <fstream> // file stream
#include <sstream> // string stream
#include <cmath>
#include <ctime> // calculate time between dates
#include <time.h> // time code
#include <vector>

using namespace std;

const double p = 0.5; // global variable
const double q = 1 - p; // global variable


vector<string> splitstr(string str, char c) { // split string into vector by commas
   vector<string> out;
   stringstream ss(str);

   while (ss.good()) {
      string substr;
      getline(ss, substr, c);
      out.push_back(substr);
   }
   return out;
}
tm make_tm(string date) { // converts string to date/time struct tm
   // assumes YYYYMMDD date format
   int year, month, day;
   year = stoi( date.substr(0, 4) );
   month = stoi( date.substr(4, 2) );
   day = stoi( date.substr(6, 2) );
    tm tm = {0};
    tm.tm_year = year - 1900; // years count from 1900
    tm.tm_mon = month - 1;  // months count from Jan = 0
    tm.tm_mday = day;       // days count from 1
    return tm;
}
double u_fxn(double R, double sigma, double dt) { // up factor function
   return exp(sigma * sqrt(dt) + (R - 0.5 * pow(sigma, 2)) * dt);
}
double d_fxn(double R, double sigma, double dt) { // down factor function
   return exp(-sigma * sqrt(dt) + (R - 0.5 * pow(sigma, 2)) * dt);
}
double price_asset(double S0, int n, int h, double u, double d) { // asset price function
   return S0 * pow(u, h) * pow(d, n - h); // we should probably check if 0 <= h <= n...
}
double payoff(double S, double K, char type) {
   if (type == 'c') // payoff if call option
```

```
      return fmax(S - K, 0);
   else if (type == 'p') // payoff if put option
      return fmax(K - S, 0);
   else
      return -1; // do something obviously wrong if input is invalid
}
double price_option_r(int n, int h, int N, double S0, double K,
   double R, double sigma, double dt, char type) { // naive recursive implementation

   if (n == N) { // terminal node is simply the payoff
      double u = u_fxn(R, sigma, dt);
      double d = d_fxn(R, sigma, dt);
      double S = price_asset(S0, n, h, u, d);
      return payoff(S, K, type);
   }
   else { // if we're not at the terminal node, use the recursive algorithm
      return pow(1 + R, -dt) * ( p * price_option_r(n + 1, h + 1, N, S0, K, R, sigma, dt,
          type)
         + q * price_option_r(n + 1, h, N, S0, K, R, sigma, dt, type) );
   }
}
double price_option_i(int N, double S0, double K,
   double R, double sigma, double dt, char type) { // iterative implementation
   // assumes you want the time-zero price of the option
   double prices[N + 1], S;
   double u = u_fxn(R, sigma, dt);
   double d = d_fxn(R, sigma, dt);

   // first loop over all the terminal nodes for all possible H and T combinations
   // to compute V_N = payoff
   for (int i = 0; i < N + 1; i++) { // traverse upwards through the nodes
      S = price_asset(S0, N, i, u, d);
      prices[i] = payoff(S, K, type);
   }
   // use the recursive algorithm to price nodes prior to the terminal nodes
   for (int i = N - 1; i >= 0; i--) // traverse backwards through the tree
      for (int j = 0; j < i + 1; j++) // traverse upwards through the nodes
         prices[j] = p * prices[j + 1] + q * prices[j];
   return pow(1 + R, -dt * N) * prices[0];
}
double implied_vol(double V_obs, double S0, double K,
   int N, double R, double dt, char type, double eps = pow(10,-5)) { // bisection
   // bisection algorithm to solve for an implied volatility estimate
   const double ESCAPE_MIN = pow(10,-5); // value to quit if we go too low
   const double ESCAPE_MAX = 10; // value to quit if we go too high

   double V_test_lo, V_test_hi, V_test, sig_test;
   double sig_lo = 0.10;
   double sig_hi = 0.50;

   do {
      V_test_lo = price_option_i(N, S0, K, R, sig_lo, dt, type);
      V_test_hi = price_option_i(N, S0, K, R, sig_hi, dt, type);
```

```
      if ( fabs(V_test_lo - V_obs) < eps) // first guess for sig_lo was correct
         return sig_lo;
      else if ( fabs(V_test_hi - V_obs) < eps ) // first guess for sig_hi was correct
         return sig_hi;
      else if ( V_test_lo > V_obs ) { // sig_lo was originally set too high
         sig_hi = sig_lo;
         sig_lo = sig_lo/10;
      } else if ( V_test_hi < V_obs ) { // sig_hi was originally set too low
         sig_lo = sig_hi;
         sig_hi = 2 * sig_hi;
      } else { // sig_obs is somewhere in between sig_lo and sig_hi
         do {
            sig_test = (sig_lo + sig_hi)/2; // bisect!
            V_test = price_option_i(N, S0, K, R, sig_test, dt, type);

            if ( V_test > V_obs ) // price too high: contract right endpoint
               sig_hi = sig_test;
            else // price too low: contract left endpoint
               sig_lo = sig_test;
         } while ( fabs(V_test - V_obs) > eps );
         return sig_test;
      }
   } while ( (sig_lo > ESCAPE_MIN) & (sig_hi < ESCAPE_MAX) );
   return -1; // do something obviously wrong is we reach the escape values
}

int main() {
   string filepath = "google-opt2_old.csv";
   double S0[] = {549.21, 553.95, 555.29};
   int N[] = {pow(10,1),pow(10,2),pow(10,3),pow(10,4),pow(10,5)};
   double R = 0.005;
   double vol, dt; char type;
   vector<double> taus, strikes, asks, spots;
   vector<string> types;

   // READ DATA
   /*
      Our approach will be to loop over each line in the CSV and save each column
      to its own vector. Later we will loop over each vector value to compute
      the relevant implied vols.
   */
   ifstream data (filepath);
   string line; // each row of the csv prior to processing
   vector<string> row; // we will separate the csv rows into their entries
   struct tm tm1, tm2; // to store the issue date & expiry

   getline(data, line); // read header
   while (getline(data, line)) { // loop over each line in the CSV
      row = splitstr(line, ','); // split line on comma

      // get start & expiry dates for years to expiry
      tm1 = make_tm( row.at(0) );
      tm2 = make_tm( row.at(1) );
```

```
   taus.push_back( difftime(mktime(&tm2), mktime(&tm1))/(60 * 60 * 24 * 365) );
   types.push_back( row.at(2) ); // "C" call or "P" put
   strikes.push_back( stod(row.at(3)) );
   asks.push_back( stod(row.at(4)) );

   if (tm1.tm_mday == 1) { // select appropriate closing price
      spots.push_back( S0[0] );
   } else if (tm1.tm_mday == 2) {
      spots.push_back( S0[1] );
   } else if (tm1.tm_mday == 3) {
      spots.push_back( S0[2] );
   }
}
data.close(); // close input csv

// WRITE DATA
ofstream outfile;
outfile.open("implied_vols_googl_rounded.csv"); // initialized csv
outfile << "N,spots,strike,tau,type,ask,vol,time" << endl; // header

for (int j = 0; j < (sizeof(N)/sizeof(N[0])); j++) { // loop over all values of N
   clock_t t1, t2; // time code

   // COMPUTE IMPLIED VOLS
   for (int i = 0; i < taus.size(); i++) { // loop over every option in the data set
      dt = taus.at(i) / N[j];

      // check if call or put option
      if ( types.at(i) == "C" ) {
         type = 'c';
      } else if ( types.at(i) == "P" ){
         type = 'p';
      } else { // do something obviously wrong if input is unexpected
         type = 'z';
      }

      t1 = clock(); // start timer
      // compute implied vol here
      vol = implied_vol(asks.at(i), spots.at(i), strikes.at(i), N[j], R, dt, type);
      t2 = clock(); // end timer
      float diff = ((float)t2 - (float)t1); // compute time

      // print to CSV
      outfile << N[j] << "," << spots.at(i) << "," << strikes.at(i)
      << "," << taus.at(i) << "," << types.at(i) << "," << asks.at(i)
      << "," << vol << "," << diff/CLOCKS_PER_SEC << endl;

      // print to console
      cout << i << "\t" << "N: 10^" << log10(N[j]) << "\t vol: " << vol
      << "\t time: " << diff/CLOCKS_PER_SEC << endl;
   }
   cout << "-------------------------------------------------" << endl;
}
```

```
   outfile.close();
}
```