Part II.

Reported in an Appendix below & illustrated in Figure 1 are the American option prices as computed by the American pricing algorithm using the option data & implied volatility estimates from Assignment 4 (also reported in the Appendix).

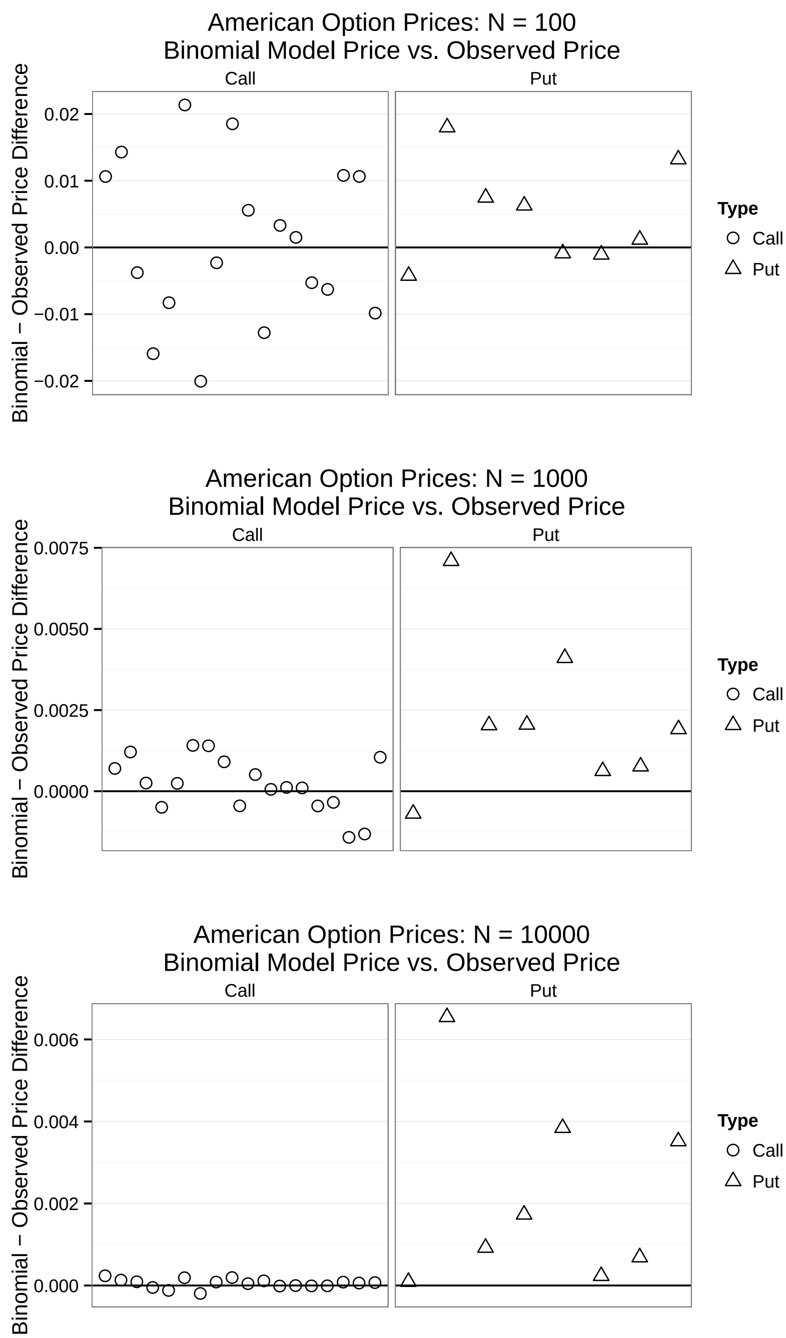

Figure 1: Price differences between the binomial model and observed market prices given options written on GOOGL over different values of $N = 10^2, 10^3, 10^4$. Implied volatilities for the binomial model computed with $N = 10^5$ and observed market prices.

For $N = 10^2$ we note that the American call option prices computed using the binomial model may be either greater or less than the observed market prices, while American put option binomial model prices are more restricted to being greater than the observed market prices. A pattern emerges when we increase N. We find that American call option prices computed by the binomial are approximately equal to the corresponding observed market price, and that American put option prices may be greater than or equal to their corresponding market prices.

Note that this pattern aligns with the result from theory stating that the early exercise feature of American call options contribute nothing to its value, that is,

$$C_0^{Am} = C_0^{Euro}$$

while this is not so for American put options. We find, perhaps more intuitively, that the early exercise feature of an American put does indeed make it more valuable, that is

$$P_0^{Am} \geq P_0^{Euro}$$

Appendix A Code Output: Data Tables[1]

N	spots	strike	tau	type	ask	vol	price	diff	absdiff
100	549.21	550	0.0493151	C	8.2	0.175079	8.21063	0.0106328	0.0106328
100	549.21	560	0.0493151	C	4.1	0.170382	4.1143	0.0142955	0.0142955
100	549.21	565	0.0493151	C	2.7	0.167716	2.69623	-0.0037745	0.0037745
100	549.21	570	0.0493151	C	1.8	0.16884	1.78408	-0.015917	0.015917
100	549.21	580	0.0493151	C	0.8	0.174525	0.791717	-0.00828282	0.00828282
100	549.21	500	0.0493151	P	0.4	0.236432	0.395757	-0.0042432	0.0042432
100	549.21	550	0.0493151	P	8.9	0.176016	8.918	0.0179967	0.0179967
100	553.95	540	0.0465753	C	19.4	0.232656	19.4214	0.0213601	0.0213601
100	553.95	545	0.0465753	C	13.9	0.182059	13.8799	-0.0200544	0.0200544
100	553.95	550	0.0465753	C	10.6	0.17687	10.5977	-0.00229908	0.00229908
100	553.95	560	0.0465753	C	5.5	0.169334	5.51853	0.0185308	0.0185308
100	553.95	565	0.0465753	C	3.8	0.168562	3.80557	0.00557056	0.00557056
100	553.95	570	0.0465753	C	2.45	0.165746	2.43723	-0.0127742	0.0127742
100	553.95	580	0.0465753	C	1	0.167287	1.0033	0.00330304	0.00330304
100	553.95	585	0.0465753	C	0.65	0.170599	0.651508	0.00150758	0.00150758
100	553.95	590	0.0465753	C	0.5	0.180324	0.494717	-0.00528268	0.00528268
100	553.95	600	0.0465753	C	0.2	0.185236	0.193716	-0.00628384	0.00628384
100	553.95	535	0.0465753	P	2.25	0.184271	2.25746	0.00746383	0.00746383
100	553.95	540	0.0465753	P	3.2	0.17915	3.20628	0.00627922	0.00627922
100	553.95	550	0.0465753	P	6.3	0.172108	6.29911	-0.000891419	0.000891419
100	555.29	550	0.0438356	C	10.9	0.171209	10.9108	0.0108001	0.0108001
100	555.29	560	0.0438356	C	5.6	0.164837	5.61064	0.0106399	0.0106399
100	555.29	570	0.0438356	C	2.4	0.160834	2.39016	-0.00984147	0.00984147
100	555.29	525	0.0438356	P	0.85	0.195245	0.848922	-0.00107817	0.00107817
100	555.29	535	0.0438356	P	1.8	0.182426	1.80116	0.00116351	0.00116351
100	555.29	550	0.0438356	P	5.6	0.173694	5.6132	0.0132002	0.0132002

N	spots	strike	tau	type	ask	vol	price	diff	absdiff
1000	549.21	550	0.0493151	C	8.2	0.175079	8.2007	0.000701657	0.000701657
1000	549.21	560	0.0493151	C	4.1	0.170382	4.10121	0.00120903	0.00120903
1000	549.21	565	0.0493151	C	2.7	0.167716	2.70025	0.000252437	0.000252437
1000	549.21	570	0.0493151	C	1.8	0.16884	1.79951	-0.000494547	0.000494547
1000	549.21	580	0.0493151	C	0.8	0.174525	0.800242	0.000242221	0.000242221
1000	549.21	500	0.0493151	P	0.4	0.236432	0.399309	-0.000690656	0.000690656
1000	549.21	550	0.0493151	P	8.9	0.176016	8.90709	0.00709223	0.00709223
1000	553.95	540	0.0465753	C	19.4	0.232656	19.4014	0.00141121	0.00141121
1000	553.95	545	0.0465753	C	13.9	0.182059	13.9014	0.00139959	0.00139959
1000	553.95	550	0.0465753	C	10.6	0.17687	10.6009	0.000907391	0.000907391

[1]Column information: N = number of steps for the binomial model; spots = market closing spot price; strike = contract strike price; tau = years to expiry; ask = market closing ask price; vol = implied volatility via the binomial model; price = American option price via the binomial model using the implied volatility; diff = difference between price and ask, absdiff = absolute difference between price and ask.

N	spots	strike	tau	type	ask	vol	price	diff	absdiff
1000	553.95	560	0.0465753	C	5.5	0.169334	5.49955	-0.000450127	0.000450127
1000	553.95	565	0.0465753	C	3.8	0.168562	3.80051	0.000513802	0.000513802
1000	553.95	570	0.0465753	C	2.45	0.165746	2.45006	5.85255e-05	5.85255e-05
1000	553.95	580	0.0465753	C	1	0.167287	1.00012	0.000118394	0.000118394
1000	553.95	585	0.0465753	C	0.65	0.170599	0.650103	0.000102743	0.000102743
1000	553.95	590	0.0465753	C	0.5	0.180324	0.499547	-0.000452898	0.000452898
1000	553.95	600	0.0465753	C	0.2	0.185236	0.199657	-0.000342843	0.000342843
1000	553.95	535	0.0465753	P	2.25	0.184271	2.25203	0.00203186	0.00203186
1000	553.95	540	0.0465753	P	3.2	0.17915	3.20205	0.00205032	0.00205032
1000	553.95	550	0.0465753	P	6.3	0.172108	6.3041	0.00410472	0.00410472
1000	555.29	550	0.0438356	C	10.9	0.171209	10.8986	-0.00142061	0.00142061
1000	555.29	560	0.0438356	C	5.6	0.164837	5.59868	-0.00131719	0.00131719
1000	555.29	570	0.0438356	C	2.4	0.160834	2.40105	0.0010471	0.0010471
1000	555.29	525	0.0438356	P	0.85	0.195245	0.850621	0.000621492	0.000621492
1000	555.29	535	0.0438356	P	1.8	0.182426	1.80076	0.000762419	0.000762419
1000	555.29	550	0.0438356	P	5.6	0.173694	5.60191	0.00191104	0.00191104

N	spots	strike	tau	type	ask	vol	price	diff	absdiff
10000	549.21	550	0.0493151	C	8.2	0.175079	8.20024	0.000236904	0.000236904
10000	549.21	560	0.0493151	C	4.1	0.170382	4.10013	0.000131087	0.000131087
10000	549.21	565	0.0493151	C	2.7	0.167716	2.70009	9.11665e-05	9.11665e-05
10000	549.21	570	0.0493151	C	1.8	0.16884	1.79995	-4.77606e-05	4.77606e-05
10000	549.21	580	0.0493151	C	0.8	0.174525	0.799881	-0.000119377	0.000119377
10000	549.21	500	0.0493151	P	0.4	0.236432	0.400088	8.78292e-05	8.78292e-05
10000	549.21	550	0.0493151	P	8.9	0.176016	8.90654	0.00654441	0.00654441
10000	553.95	540	0.0465753	C	19.4	0.232656	19.4002	0.000189486	0.000189486
10000	553.95	545	0.0465753	C	13.9	0.182059	13.8998	-0.00019353	0.00019353
10000	553.95	550	0.0465753	C	10.6	0.17687	10.6001	8.41612e-05	8.41612e-05
10000	553.95	560	0.0465753	C	5.5	0.169334	5.50019	0.000193259	0.000193259
10000	553.95	565	0.0465753	C	3.8	0.168562	3.80005	4.57036e-05	4.57036e-05
10000	553.95	570	0.0465753	C	2.45	0.165746	2.45011	0.000112992	0.000112992
10000	553.95	580	0.0465753	C	1	0.167287	0.999987	-1.31966e-05	1.31966e-05
10000	553.95	585	0.0465753	C	0.65	0.170599	0.649998	-1.77586e-06	1.77586e-06
10000	553.95	590	0.0465753	C	0.5	0.180324	0.499991	-9.10731e-06	9.10731e-06
10000	553.95	600	0.0465753	C	0.2	0.185236	0.199993	-7.34388e-06	7.34388e-06
10000	553.95	535	0.0465753	P	2.25	0.184271	2.25092	0.000920056	0.000920056
10000	553.95	540	0.0465753	P	3.2	0.17915	3.20173	0.00172862	0.00172862
10000	553.95	550	0.0465753	P	6.3	0.172108	6.30384	0.0038416	0.0038416
10000	555.29	550	0.0438356	C	10.9	0.171209	10.9001	8.25383e-05	8.25383e-05
10000	555.29	560	0.0438356	C	5.6	0.164837	5.60006	6.00218e-05	6.00218e-05
10000	555.29	570	0.0438356	C	2.4	0.160834	2.40007	7.15253e-05	7.15253e-05
10000	555.29	525	0.0438356	P	0.85	0.195245	0.850232	0.000231948	0.000231948
10000	555.29	535	0.0438356	P	1.8	0.182426	1.80069	0.000686623	0.000686623
10000	555.29	550	0.0438356	P	5.6	0.173694	5.60352	0.00351682	0.00351682

Appendix B Code

B.1 main.cpp

```
#include <iostream>
#include <fstream> // file stream
#include <sstream> // string stream
#include <cmath>
#include <ctime> // time code
#include <vector>

using namespace std;

// risk neutral probabilities
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;
}
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_am_r(int n, int h, int N, double S0, double K,
   double R, double sigma, double dt, char type) { // naive recursive implementation
   // american option price

   double u = u_fxn(R, sigma, dt);
   double d = d_fxn(R, sigma, dt);
   double S = price_asset(S0, n, h, u, d);
   double G_n = payoff(S, K, type);
```

```
  if (n == N) { // terminal node is simply the payoff
    return G_n;
  }
  else { // if we're not at the terminal node, use the recursive algorithm
    double Vu = price_option_am_r(n + 1, h + 1, N, S0, K, R, sigma, dt, type);
    double Vd = price_option_am_r(n + 1, h, N, S0, K, R, sigma, dt, type);
    return fmax( G_n, pow(1 + R, -dt) * ( p * Vu + q * Vd) );
  }
}
double price_option_am_i(int N, double S0, double K,
  double R, double sigma, double dt, char type) { // iterative implementation
  // american option price
  // 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

      S = price_asset(S0, i, j, u, d); // price S at i-th step given j heads

      prices[j] = fmax( payoff(S, K, type),
        pow(1 + R, -dt) * (p * prices[j + 1] + q * prices[j]) );

    }
  return prices[0];
}

int main() {
  int N = pow(10,5); // number of steps for the binomial model
  double R = 0.005; // riskless interest rate

  string infilepath = "../data/implied_vols_googl_NMAX.csv";
  // INPUT DATA COLUMN HEADERS
  // N, spots, strike, tau, type, ask, vol, time

  string outfilename = "../data/american_prices_googl_N" + to_string(N) + ".csv";
  double dt; char type;
  vector<double> spots, strikes, taus, asks, vols, prices, diffs, absdiffs;
  vector<string> types;

  // READ DATA
  /*
    Our approach is to loop over each line of the CSV and save each column
```

```
   to a corresponding vector. Later we will loop through each vector to compute
   the relevant price
*/
ifstream data( infilepath ); // open csv
string line; // each row of the csv prior to processing
vector<string> row; // we will separate the csv rows into their entries

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

   spots.push_back( stod(row.at(1)) );
   strikes.push_back( stod(row.at(2)) );
   taus.push_back( stod(row.at(3)) );
   types.push_back( row.at(4) );
   asks.push_back( stod(row.at(5)) );
   vols.push_back( stod(row.at(6)) );
}
data.close(); // close input csv

// COMPUTE AMERICAN OPTION PRICES
for (int i = 0; i < spots.size(); i++) {
   // compute time step value
   dt = taus.at(i) / N;

   // extract option type information
   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';

   // compute price
   clock_t t1 = clock();
   prices.push_back( price_option_am_i(N, spots.at(i), strikes.at(i),
      R, vols.at(i), dt, type) );
   clock_t t2 = clock();
   double elapsed = double(t2 - t1) / CLOCKS_PER_SEC;
   cout << "N = 10^" << log10(N) << "\t" << elapsed << endl;

   // difference between computed price and observed market price
   diffs.push_back( prices.at(i) - asks.at(i) );
   // absolute value
   absdiffs.push_back( fabs(diffs.at(i)) );
}

// WRITE DATA
ofstream outfile;
outfile.open( outfilename ); // initialize csv
string header = "N,spots,strike,tau,type,ask,vol,price,diff,absdiff";
outfile << header << endl; // header

cout << header << endl;
```

```
    for (int i = 0; i < spots.size(); i++) { // loop over every option in the data set
        // print to CSV
        outfile << N << "," << spots.at(i) << "," << strikes.at(i)
        << "," << taus.at(i) << "," << types.at(i) << "," << asks.at(i)
        << "," << vols.at(i) << "," << prices.at(i) << "," << diffs.at(i)
        << "," << absdiffs.at(i) << endl;

        // print to console
        cout << N << "," << spots.at(i) << "," << strikes.at(i)
        << "," << taus.at(i) << "," << types.at(i) << "," << asks.at(i)
        << "," << vols.at(i) << "," << prices.at(i) << "," << diffs.at(i)
        << "," << absdiffs.at(i) << endl;

    }

    outfile.close(); // close output csv
}
```