CS 476: Assignment 1

Daniel Matheson; 20270871

Question 1:
The fair value of a European call option is given by

$$V_0 = e^{-rT}\mathbb{E}^q[V_T]$$

We are given that $r = 0$ and hence

$$V_0 = \mathbb{E}^q[V_T]$$

and

$$q^\star = \frac{e^{r(T-t)} - d}{u-d} = \frac{1-d}{u-d}$$

In this case we are dealing with a one period model; with one European call option with $K_1 = 94$ and $V_0^1 = 8$, and another with $K_2 = 92$ and $V_0^2 = 9$.
We must use this to find the fair-price of a European call with $K_3 = 100$, denoted V_0^3.
From this information we can infer that:

$$\begin{aligned}
V_0^1 &= \mathbb{E}^q[V_T] \\
&= \big(q^\star(uS_0 - K_1)^+ + (1-q^\star)(dS_0 - K_1)^+\big) \\
&= \big(q^\star(100u - 94)^+ + (1-q^\star)(100d - 94)^+\big) \\
&= \big(\frac{1-d}{u-d}(100u-94) + (1 - \frac{1-d}{u-d})(100d-94)^+\big) = 8 \\
\text{and } V_0^2 &= \big(\frac{1-d}{u-d}(100u-92) + (1 - \frac{1-d}{u-d})(100d-92)^+\big) = 9 \\
\text{and } V_0^3 &= \big(\frac{1-d}{u-d}(100u-100)^+ + (1 - \frac{1-d}{u-d})(100d-100)^+\big) \\
&= \big(\frac{1-d}{u-d}100(u-1)\big) \quad \text{since d} < 1 \text{ and so } dS_0 < 100 = K_3
\end{aligned}$$

We have a problem since there are $(\cdots)^+$ functions in our linear system, in the equations for V_0^1 and V_0^2; to remove these we will show that, in fact, $d \leq 0.92$, which removes the possibility of a positive value for $100d - K_i$ for both $i = 1,2$; making both $(\cdots)^+$ functions zero.
Assume $d > 0.92$; then buy one share of stock ABC at \$100 today, and short a European call option with strike \$92 for \$9 on stock ABC; the payoff at T is as follows:

$$\begin{cases} V_T^u = -100 + 92 + 9 = 1 > 0 & \text{with probability } p \\ V_T^d = -100 + 92 + 9 = 1 > 0 & \text{with probability } (1-p) \end{cases}$$

which represents an arbitrage opportunity, a contradiction ※ !
And hence $d \leq 0.92$.

We now have:

$$
\begin{aligned}
V_0^1 &= \Big(\frac{1-d}{u-d}(100u-94)\Big) = 8 \ \textbf{(1)}\\
V_0^2 &= \Big(\frac{1-d}{u-d}(100u-92)\Big) = 9 \ \textbf{(2)}\\
V_0^3 &= \Big(\frac{1-d}{u-d}(100u-100)\Big) = x\\
(1)-(2) &\implies \frac{1-d}{u-d}(-94-(-92)) = 8-9\\
&\implies \frac{1-d}{u-d} = \frac{1}{2}\\
&\implies 2-d = u \ \textbf{(3)}\\
\text{(3) into (1):} &\implies \frac{1-d}{2-2d}(100(2-d)-92) = 9\\
&\implies 100d^2 - 190d + 90 = 0\\
&\implies d = 0.9 \text{ or } 1\\
&\implies d = 0.9 \text{ since } d < 1\\
&\implies u = 2-d = 1.1
\end{aligned}
$$

Putting this into the equation for the value of the \$100 strike call option;

$$V_0^3 = \Big(\frac{1-d}{u-d}(100u-100)\Big) = \frac{0.1}{0.2}(110-100) = 5$$

Hence the European call option on ABC with strike \$100 has a fair-price of \$5.

Question 2:

$$
\begin{aligned}
u \geq\; & e^{r\Delta t} \geq d\\
\iff e^{\sigma\sqrt{\Delta t}+(r-\frac{\sigma^2}{2})\Delta t} \geq\; & e^{r\Delta t} \geq e^{-\sigma\sqrt{\Delta t}+(r-\frac{\sigma^2}{2})\Delta t}\\
\iff e^{\sigma\sqrt{\Delta t}}\cancel{e^{r\Delta t}}e^{-\frac{\sigma^2}{2}\Delta t} \geq\; & \cancel{e^{r\Delta t}} \geq e^{-\sigma\sqrt{\Delta t}}\cancel{e^{r\Delta t}}e^{-\frac{\sigma^2}{2}\Delta t}\\
\iff e^{\sigma\sqrt{\Delta t}}e^{-\frac{\sigma^2}{2}\Delta t} \geq\; & 1 \geq e^{-\sigma\sqrt{\Delta t}}e^{-\frac{\sigma^2}{2}\Delta t} \quad \text{take ln of both sides}\\
\iff \sigma\sqrt{\Delta t} - \frac{\sigma^2}{2}\Delta t \geq\; & 0 \geq -\sigma\sqrt{\Delta t} - \frac{\sigma^2}{2}\Delta t\\
\iff \sqrt{\Delta t}(\sigma - \frac{\sigma^2}{2}\sqrt{\Delta t}) \geq\; & 0 \geq \sqrt{\Delta t}(-\sigma + \frac{\sigma^2}{2}\sqrt{\Delta t}) \quad \text{assume } \Delta t \neq 0\\
\iff (\sigma - \frac{\sigma^2}{2}\sqrt{\Delta t}) \geq\; & 0 \geq (-\sigma + \frac{\sigma^2}{2}\sqrt{\Delta t})\\
\implies \sqrt{\Delta t} \leq\; & \frac{2}{\sigma}\\
\implies \Delta t \leq\; & \frac{4}{\sigma^2}
\end{aligned}
$$

Hence the solutions are $\Delta t = 0$ (trivial solution, which does not have any interpretation) and $\Delta t \leq \frac{4}{\sigma^2}$. We use the second, non-trivial, solution. Also note $\Delta t \leq 1$ since $T = 1$.

We assume $\Delta t \leq \frac{4}{\sigma^2}$, then does there exist a unique risk neutral probability $q^\star \in [0,1]$ such that

$$S_t = e^{-r\Delta t}\mathbb{E}^q(S_{t+1})$$

where $\mathbb{E}^q(\cdot)$ is the expectation using $q^\star$ as the probability?
When we have Δt satisfying this condition, it means that $d \leq e^{r\Delta t} \leq u$, which implies that the model is *arbitrage-free*. By the *Fundamental Theorem of Asset Pricing* this means that $\exists$ unique $q^\star \in [0,1]$ that satisfies the condition above.

Question 3:
We wish to prove that

$$V_0^0 = e^{-rN\Delta t}\sum_{k=0}^{N}\binom{N}{k}(q^\star)^k(1-q^\star)^{N-k}V_k^N$$

by induction.
For $N=1$ we already know that $V_0^0 = e^{-r\Delta t}(q^\star(V_1^1 + (1-q^\star)V_0^1)$ from class.
Hence let $N=n$ and assume that

$$V_0^0 = e^{-rn\Delta t}\sum_{k=0}^{n}\binom{n}{k}(q^\star)^k(1-q^\star)^{n-k}V_k^n$$

And now we prove this holds for $N=n+1$;

$$\begin{aligned}
\text{We know } V_0^0 &= e^{-rn\Delta t}\sum_{k=0}^{n}\binom{n}{k}(q^\star)^k(1-q^\star)^{n-k}V_k^n \text{ and}\\
V_k^n &= e^{-r\Delta t}\big(q^\star V_{k+1}^{n+1} + (1-q^\star)V_k^{n+1}\big)\\
\implies V_0^0 &= e^{-rn\Delta t}\sum_{k=0}^{n}\binom{n}{k}(q^\star)^k(1-q^\star)^{n-k}e^{-r\Delta t}\Big[q^\star V_{k+1}^{n+1} + (1-q^\star)V_k^{n+1}\Big]\\
&= e^{-r(n+1)\Delta t}\Big[\underbrace{\sum_{k=0}^{n}}_{\text{let } j=k+1}\binom{n}{k}(q^\star)^{k+1}(1-q^\star)^{n-k}V_{k+1}^{n+1} + \sum_{k=0}^{n}\binom{n}{k}(q^\star)^k(1-q^\star)^{n+1-k}V_k^{n+1}\Big]\\
&= e^{-r(n+1)\Delta t}\Big[\underbrace{\sum_{j=1}^{n+1}}_{k=j}\binom{n}{j-1}(q^\star)^j(1-q^\star)^{n+1-j}V_j^{n+1} + \sum_{k=0}^{n}\binom{n}{k}(q^\star)^k(1-q^\star)^{n+1-k}V_k^{n+1}\Big]\\
&= e^{-r(n+1)\Delta t}\Big[\underbrace{\sum_{k=1}^{n+1}\binom{n}{k-1}(q^\star)^k(1-q^\star)^{n+1-k}V_k^{n+1}}_{(1)} + \underbrace{\sum_{k=0}^{n}\binom{n}{k}(q^\star)^k(1-q^\star)^{n+1-k}V_k^{n+1}}_{(2)}\Big]\\
&= e^{-r(n+1)\Delta t}\Big[\underbrace{(1-q^\star)^{n+1}V_0^{n+1}}_{k=0 \text{ term of } (2)} + \underbrace{(q^\star)^{n+1}V_{n+1}^{n+1}}_{k=n+1 \text{ term of } (1)}\\
&\qquad + \sum_{k=1}^{n}\left(\binom{n}{k-1} + \binom{n}{k}\right)(q^\star)^k(1-q^\star)^{n+1-k}V_k^{n+1}\Big]
\end{aligned}$$

$$= e^{-r(n+1)\Delta t}\Big[(1-q^\star)^{n+1}V_0^{n+1} + (q^\star)^{n+1}V_{n+1}^{n+1} + \sum_{k=1}^{n}\binom{n+1}{k}(q^\star)^k(1-q^\star)^{n+1-k}V_k^{n+1}\Big]$$

$$= e^{-r(n+1)\Delta t}\Big[\underbrace{\binom{n+1}{0}(q^\star)^0(1-q^\star)^{n+1}V_0^{n+1}}_{k=0 \text{ term}} + \underbrace{\binom{n+1}{n+1}(q^\star)^{n+1}(1-q^\star)^{n+1-n+1}V_{n+1}^{n+1}}_{k=(n+1) \text{ term}}$$

$$+ \sum_{k=1}^{n}\binom{n+1}{k}(q^\star)^k(1-q^\star)^{n+1-k}V_k^{n+1}\Big]$$

$$\implies V_0^0 = \sum_{k=0}^{n+1}\binom{n+1}{k}(q^\star)^k(1-q^\star)^{n+1-k}V_k^{n+1}$$

QED □

Question 4:
S_j^n denotes the underlying price at time t_n and node j; $0 \le n \le N$, $0 \le j \le n$.
Let $V_0^{\text{tree}}(S_j^n, K, t_n)$ be the fair value from a binomial tree model for an American straddle which has the payoff

$$\max(K - S_j^N, 0) + \max(S_j^N - K, 0)$$

Also assume volatility $\sigma > 0$, and risk-free rate $r > 0$. Prove by induction that $\forall\ \lambda > 0$

$$V_0^{\text{tree}}(\lambda S_j^n, \lambda K, t_n) = \lambda V_0^{\text{tree}}(S_j^n, K, t_n)$$

$\forall\ 0 \le n \le N, 0 \le j \le n$.
Solution:
We will make use of the fact that

$$\max(K - S_j^N, 0) + \max(S_j^N - K, 0) = |S_j^N - K|$$

We proceed by backward induction on $n = 0, \ldots, N$ for arbitrary $N > 1$ [1]
First, we show that the result holds for $n = N$, taking into account that at time t_N, the American option behaves as a European option; simply taking into account the payoff function. And so:

$$\begin{aligned} V_0^{\text{tree}}(\lambda S_j^N, \lambda K, t_N) &= |\lambda S_j^N - \lambda K| \\ &= |\lambda||S_j^N - K| \\ &= \lambda|S_j^N - K| \quad \text{since } \lambda > 0 \\ &= \lambda V_0^{\text{tree}}(S_j^N, K, t_N) \end{aligned}$$

Which holds $\forall\ j = 0, \ldots, n$ since the payoff function is the same $\forall\ j$.
Now assume the result holds for some $n = i < N$, and $\forall\ j = 0, \ldots, n$.

$$V_0^{\text{tree}}(\lambda S_j^i, \lambda K, t_i) = \lambda V_0^{\text{tree}}(S_j^i, K, t_i) \quad \textbf{(1)}$$

[1]Note that from our proof it is trivial to show that this result holds for $N = 1$

We now show that the result holds for $n = i - 1$, by using the assumption that V_0^{tree} is the fair-value, and hence uses the risk neutral probability measure $q^\star$ in recursively determining preceding node values; the fact that $\sigma > 0, r > 0$ are constant also confirms this assumption. Let $j \in \{0, \dots, i-1\}$, then;

$$\begin{aligned}
&\lambda V_0^{\text{tree}}(S_j^{i-1}, K, t_{i-1}) \\
&= \lambda \max\Big[\, \underbrace{|S_j^{i-1} - K|}_{\text{exercised value}}, \underbrace{e^{-r\Delta t}\big(q^\star V_0^{\text{tree}}(S_{j+1}^i, K, t_i) + (1-q^\star)(V_0^{\text{tree}}(S_j^i, K, t_i)\big)}_{\text{holding value}}\Big] \\
&= \max\Big[|\lambda S_j^{i-1} - \lambda K|, e^{-r\Delta t}(q^\star\{\lambda V_0^{\text{tree}}(S_{j+1}^i, K, t_i)\} + (1-q^\star)\{\lambda V_0^{\text{tree}}(S_j^i, K, t_i)\}\Big] \\
&\text{where } \lambda V_0^{\text{tree}}(S_j^i, K, t_i) = V_0^{\text{tree}}(\lambda S_j^i, \lambda K, t_i) \text{ by assumption (1) above (same for } j+1) \\
&= \max\Big[|\lambda S_j^{i-1} - \lambda K|, e^{-r\Delta t}(q^\star\ V_0^{\text{tree}}(\lambda S_{j+1}^i, \lambda K, t_i)\} + (1-q^\star)\{V_0^{\text{tree}}(\lambda S_j^i, \lambda K, t_i)\}\Big] \\
&= V_0^{\text{tree}}(\lambda S_j^{i-1}, \lambda K, t_{i-1}) \qquad \text{by definition}
\end{aligned}$$

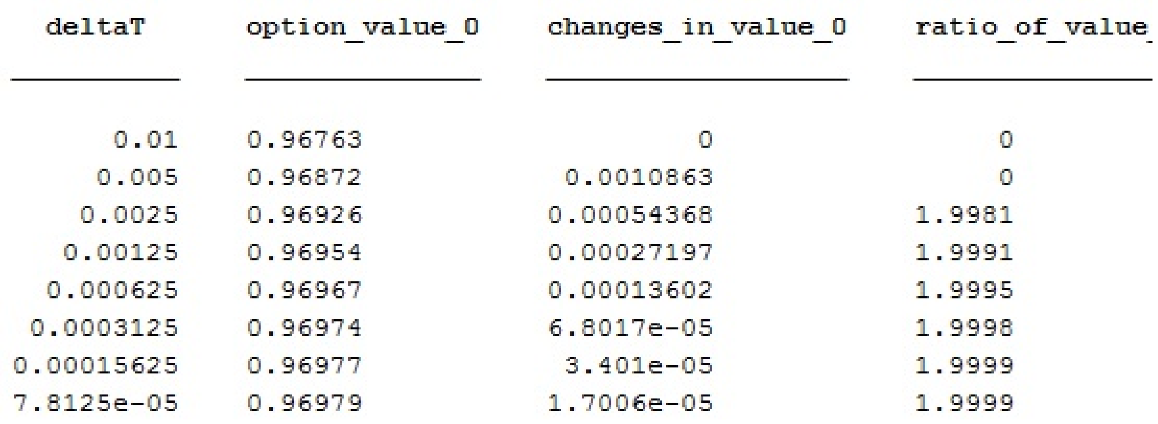
QED □

Question 5:
for $0 \le s < t$:

$$\begin{aligned}
\mathbb{E}(Z_t - Z_s) &= \mathbb{E}(Z_t) + \mathbb{E}(Z_s) \\
&= 0 + 0 = 0 \\
\mathbf{var}(Z_t - Z_s) &= \mathbf{var}(Z_t) + \mathbf{var}(Z_s) + 2\mathbf{Cov}(Z_t, -Z_s) \\
&= t + s - 2\mathbf{Cov}(Z_t, Z_s) \\
&= t + s - 2\mathbf{Cov}(Z(t-s) + Z(s), Z(s)) \\
&= t + s - 2\big[\underbrace{\mathbf{Cov}(Z(t-s), Z(s))}_{=0 \text{ since independent}} + \mathbf{Cov}(Z(s), Z(s))\big] \\
&= t + s - 2\mathbf{var}(Z_s) \\
&= t + s - 2s = t - s \qquad \square
\end{aligned}$$

MATLAB Section

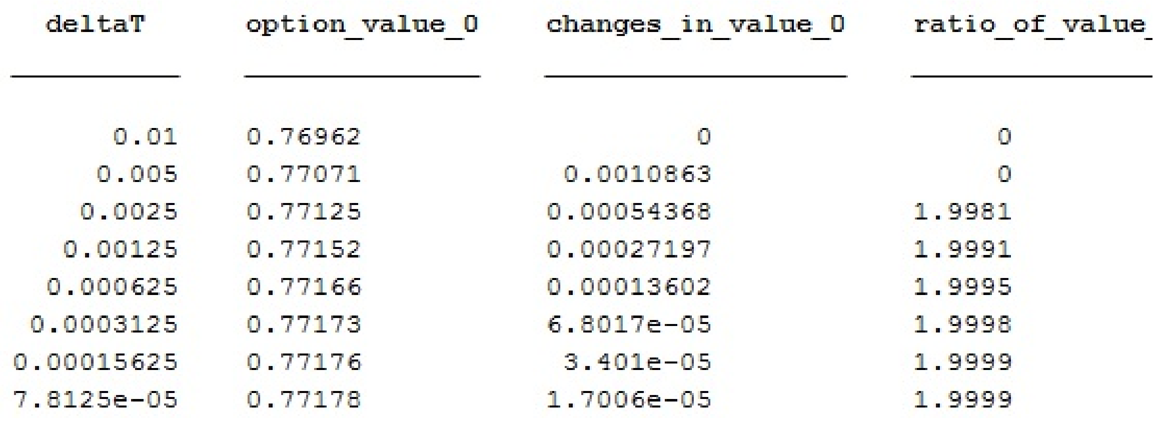
Question 6(a):



```
    deltaT       option_value_0    changes_in_value_0    ratio_of_value_
 ___________    ______________    __________________    _______________

       0.01     0.96763                    0                  0
      0.005     0.96872             0.0010863                 0
     0.0025     0.96926            0.00054368            1.9981
    0.00125     0.96954            0.00027197            1.9991
   0.000625     0.96967            0.00013602            1.9995
  0.0003125     0.96974            6.8017e-05            1.9998
 0.00015625     0.96977             3.401e-05            1.9999
 7.8125e-05     0.96979            1.7006e-05            1.9999
```

Figure 1: European Call Option Approximations (Actual Value = 0.9698)



```
    deltaT       option_value_0    changes_in_value_0    ratio_of_value_
 ___________    ______________    __________________    _______________

       0.01     0.76962                    0                  0
      0.005     0.77071             0.0010863                 0
     0.0025     0.77125            0.00054368            1.9981
    0.00125     0.77152            0.00027197            1.9991
   0.000625     0.77166            0.00013602            1.9995
  0.0003125     0.77173            6.8017e-05            1.9998
 0.00015625     0.77176             3.401e-05            1.9999
 7.8125e-05     0.77178            1.7006e-05            1.9999
```

Figure 2: European Put Option Approximations (Actual Value = 0.7718)

The price simulation is clearly affected by time discretization; as we decrease Δt we can see that we are getting more accurate values for the options, which converge to the *blsprice* function result in MATLAB. These values are in the title of the figures above; 0.9698 for the Call option and 0.7718 for the Put option.
From the assignment PDF, the theory says that $\exists\ \alpha \in \mathbb{R}$ such that

$$V_0^{\text{tree}}(\Delta t) = V_0^{\text{exact}} + \alpha\Delta t + O((\Delta t)^{3/2}) \quad \textbf{(1)}$$

As we can see in the 4th column; which is equivalent to the ratio

$$\lim_{\Delta t \to 0} \frac{V_0^{\text{tree}}(\frac{\Delta t}{2}) - V_0^{\text{tree}}(\Delta t)}{V_0^{\text{tree}}(\frac{\Delta t}{4}) - V_0^{\text{tree}}(\frac{\Delta t}{2})}$$

We have a convergence of the ratio to 2. This agrees with the theory, for some $\alpha < 0$ in (1) since the values in both tables converge to V_0^{exact} from below.

Code for 6(a):

```
function table_print = generate_table()
% This function simply outputs the table required to show convergence
% of the european options. The first column is deltaT, the second is the
% value of the option with that given deltaT, the third column is the
% change in value from one decrease in deltaT, and finally the last column
% is a ratio in the previous two changes, to see if they are converging.

nrows = 8;                          % constant
deltaT = zeros(nrows,1);            % initializing vectors
option_value_0 = zeros(nrows,1);     % "
changes_in_value_0 = zeros(nrows,1); % "
ratio_of_value_0 = zeros(nrows,1);   % "
option_type = 'call';               % call or put option

% this loop determines each option value at different values of
% deltaT, using the option_value function.
for j = 1:8
    deltaT(j) = 0.01/(2^(j-1));
    option_value_0(j) = option_value(deltaT(j), option_type, ...
         last_lattice_row(deltaT(j), rho));
end

% this loop simply computes the changes in the option values we computed
for j = 2:8
    changes_in_value_0(j) = option_value_0(j) - option_value_0(j-1);
end

% this loop computes the ratios of the changes in option values.
for j = 3:8
    ratio_of_value_0(j) = (option_value_0(j-1) - option_value_0(j-2)) / ...
                        (option_value_0(j) - option_value_0(j-1));
end

% outputs the table
table_print = table(deltaT, option_value_0, changes_in_value_0, ...
    ratio_of_value_0);
end
```

```
function value = option_value(deltaT, type, underlying_final)
% this function determines the value of a European option with time
% steps deltaT, the final value of the underlying asset, and the type
% of option which is either 'call' or 'put'

N = length(underlying_final); % set the length of vectors needed
payoff = zeros(N,1);          % initialize payoff vector
K = 10;                       % given strike price, constant for Q6
sigma = 0.22;                 % volatility

r = 0.02;                          % risk-free rate
u = exp(sigma * sqrt(deltaT));     % potential rise in stock price
d = exp(- sigma * sqrt(deltaT));   % potential fall in stock price
q = (exp(r * deltaT) - d)/(u - d); % risk-neutral probability measure

% this chunk of code creates the payoff vector, based on whether the
% option is a call or put.
if strcmp('call', type)
    payoff = max((underlying_final - K),0);
else
    payoff = max((K - underlying_final),0);
end

% This for loop has the purpose of iteratively discounting - using
% the risk neutral expectation - the payoff values until we get to the
% fair-price of the option V_0 (or in this case values(1)).
values = payoff;
for n = (N-1):-1:1
    for j = 1:n
        values(j) = exp(- r * deltaT) * (q * values(j+1) + ...
            (1-q) * values(j));
    end
end

value = values(1);

end
```

```
function ST_row = last_lattice_row(deltaT, rho)
% this function gives the last row of the lattice based on a time
% increment deltaT, and a divident payment rho (expressed as a fraction
% of the underlying asset value at the payment time); where the payment
% time is T/2.

T = 1; % Time to expiry, constant at 1 year
S_0 = 10; % Initial price of underlying
sigma = 0.22; % volatility
u = exp(sigma * sqrt(deltaT));   % potential gain in underlying asset
d = exp(- sigma * sqrt(deltaT)); % potential loss in underlying asset

N = T / deltaT;  % number of rows in the lattice
t = floor(N/2);  % dividend payment row

row = zeros(N+1, 1);  % initializing row
row(1) = S_0;         % set initial value to underlying initial

% This loop creates the final row of the lattice iteratively by
% multiplying the node at the top by u, and all of the other nodes by d.
% and if the node is in row t, it applies the dividend payment which
% reduces the value of the underlying asset at that point by (rho)S_t
for i = 1:N
   if i == t
       row = (1 - rho) * row;
   end
   row(i+1) = u * row(i);
   row(1:i) = d * row(1:i);
end

ST_row = row;

end
```

Question 6(b):

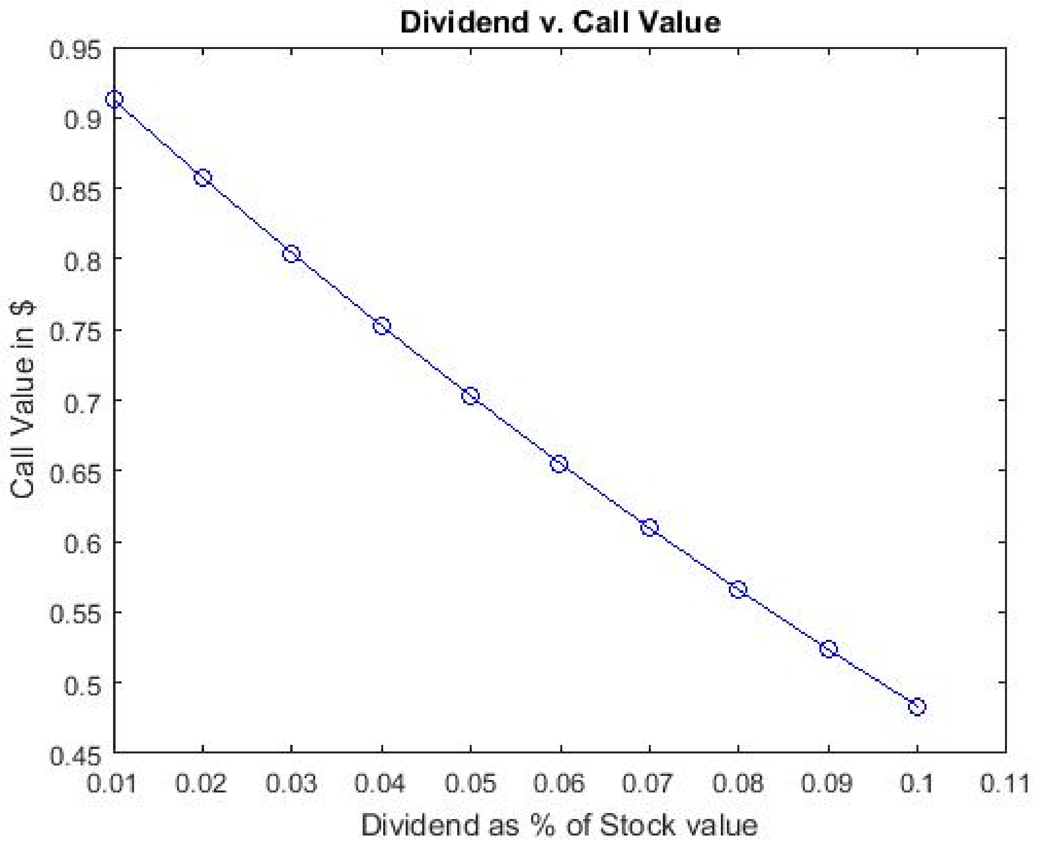


Figure 3: Dividend amount vs. Call value

It makes sense that the curve is decreasing; since we buy the option at time $t = 0$, when the initial stock price is S_0, knowing there will be a dividend payment at $t = \frac{T}{2}$ where T is the maturity date. Hence we should expect that as the amount of the dividend increases, the European call option should decrease in price, to reflect the fact that the value at expiry of the stock, S_T is expected to be lower; reducing possible gain from the call option for the buyer.

Code for 6(b):[2]

```
function rho_table = make_rho_table()
% This function plots the values of the European call option on S when
% there is a dividend payment of rho (as percentage of current price)
% at t = T/2, by varying the values of rho from 0.01 to 0.1 in increments
% of 0.01.

type = 'call';       % required for the option_value function
deltaT = 0.0005;     % small deltaT for a high number of time periods
rho = 0.01:0.01:0.1; % varying values of rho
prices = zeros(length(rho),1); % initializing a vector for option prices

for i = 1:length(rho)
    % fills in the option price values for each value of rho by calling
    % the option_value function
    prices(i) = option_value(deltaT, type, last_lattice_row(deltaT, rho(i)));
end

rho_table = { % output as a plot
    figure
    plot(rho, prices','b-o')
    title('Dividend v. Call Value')
    xlabel('Dividend as % of Stock value')
    ylabel('Call Value in $') };

end
```

[2]uses option_value code from 6(a) - see above

Question 7(a):

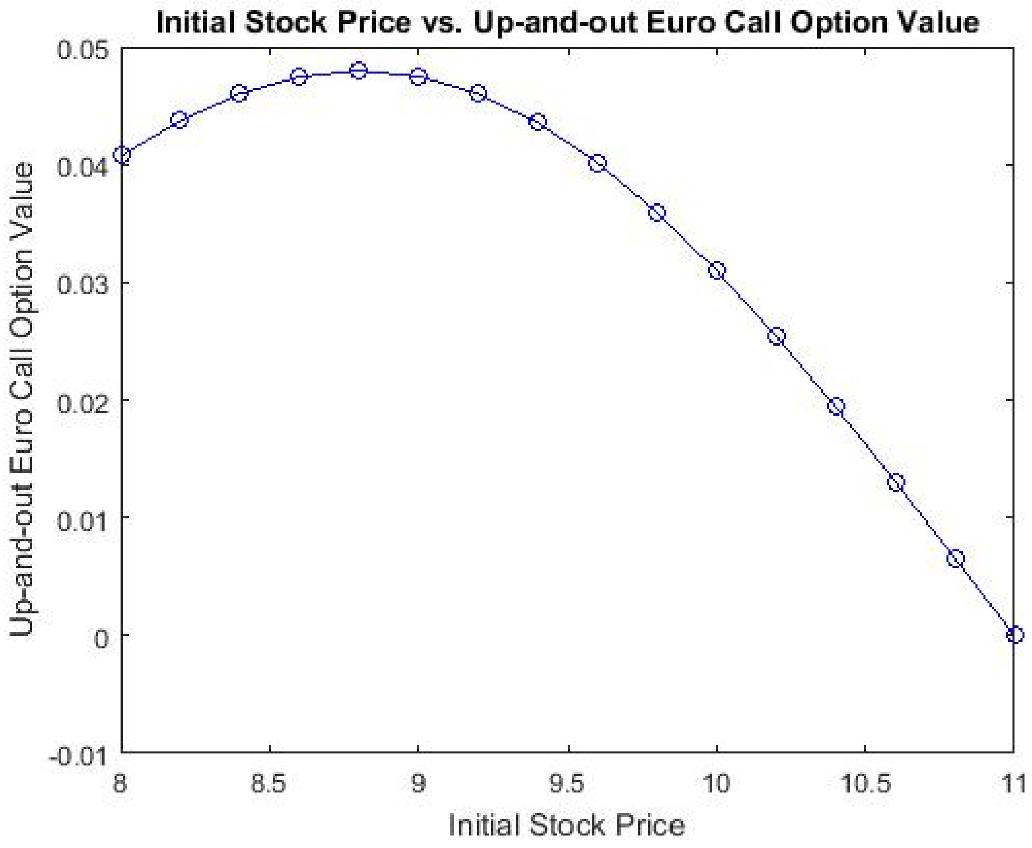


Figure 4: Up-and-out European Call Option value vs. Initial Value

The shape of the curve is entirely reasonable; it seems to be a smooth, continuous curve (if we were to reduce Δt); and it seems reasonable to assume that the higher the initial stock price, the lower the value of the option - with the value at $S_0 = 11$ being 0 - because the higher the value of S_0 that we start at, the more likely we are to cross the upper barrier of S_u, upon which the value of the option is 0.
Additionally, the fact that the option increases in value from about 8 to 9 also makes sense; the stock is no longer really in danger of crossing the barrier S_u with a large probability, but now it is more unlikely that our option will breach the strike price $K = 9.5$, lowering the value of the option as we go further to the left from K.

Code for 7(a):

```
function up_and_out_plot = plot_up_and_out()
% this function plots the up-and-out European call option values as a
% function of the initial stock prices.

K = 9.5;       % strike price on option
S_u = 11;      % upper barrier on S
sigma = 0.22;  % volatility
r = 0.02;      % risk free rate
T = 1;         % expiry
t = 0;         % time to be passed into the analytic formula is t = 0
S = 8:0.2:11;  % initial values of S; the underlying asset
values = zeros(length(S),1); % initializing vector for option values

for i = 1:length(S)
    % assign option values using analytic_soln function on each initial
    % value for S
    values(i) = analytic_soln(S(i), K, t, T, sigma, r, S_u);
end

up_and_out_plot = { % output plot
    figure
    plot(S, values','b-o')
    title('Initial Stock Price vs. Up-and-out Euro Call Option Value')
    xlabel('Initial Stock Price')
    ylabel('Up-and-out Euro Call Option Value') };

end
```

```
function option_value = analytic_soln(S, K, t, T, sigma, r, S_u)
% This function computes the value of the up-and-out European call option
% using the analytical formula given.
% S = underlying asset price at time t
% K = strike price
% t = time at which option is priced
% T = expirty time
% sigma = volatility
% r = risk free rate
% S_u = upper barrier on S

% Constants:
d_1 = (log(S/K) + (r + 1/2 * sigma^2) * (T-t))/(sigma * sqrt(T - t));
d_2 = (log(S/K) + (r - 1/2 * sigma^2) * (T-t))/(sigma * sqrt(T - t));
d_3 = (log(S/S_u) + (r + 1/2 * sigma^2) * (T-t))/(sigma * sqrt(T - t));
d_4 = (log(S/S_u) + (r - 1/2 * sigma^2) * (T-t))/(sigma * sqrt(T - t));
d_5 = (log(S/S_u) - (r - 1/2 * sigma^2) * (T-t))/(sigma * sqrt(T - t));
d_6 = (log(S/S_u) - (r + 1/2 * sigma^2) * (T-t))/(sigma * sqrt(T - t));
d_7 = (log((S*K)/(S_u)^2) - (r - 1/2 * sigma^2) * (T-t))/(sigma * sqrt(T - t));
d_8 = (log((S*K)/(S_u)^2) - (r + 1/2 * sigma^2) * (T-t))/(sigma * sqrt(T - t));

% Analytic Equation, self-explanatory. Just long.
value = S * (normcdf(d_1) - normcdf(d_3) - (S_u/S)^(1 + (2*r/sigma^2))* ...
    (normcdf(d_6) - normcdf(d_8))) - (K * exp(- r * (T - t))) * ...
    (normcdf(d_2) - normcdf(d_4) - (S_u/S)^(-1 + (2*r/sigma^2)) * ...
    (normcdf(d_5) - normcdf(d_7)));

option_value = value;

end
```

Question 7(b):

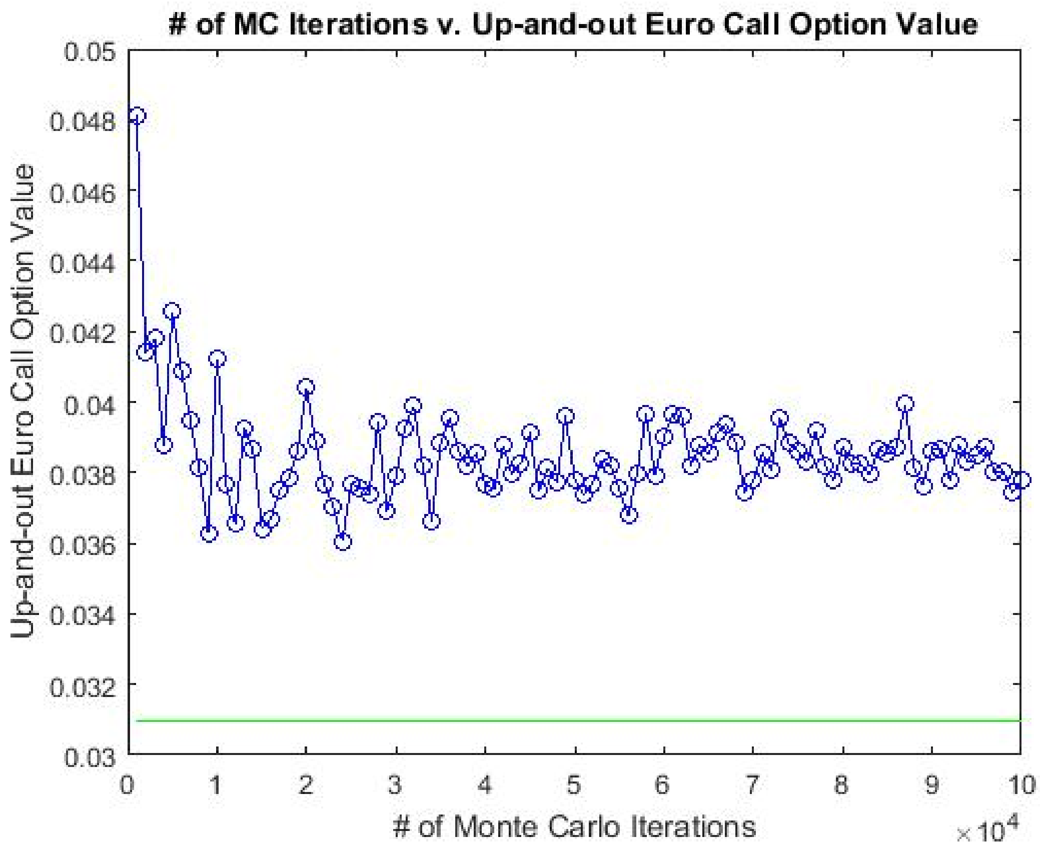


Figure 5: # of Monte Carlo iterations vs. Up-and-out Euro Call Option Value with $\Delta t = \frac{1}{250}$

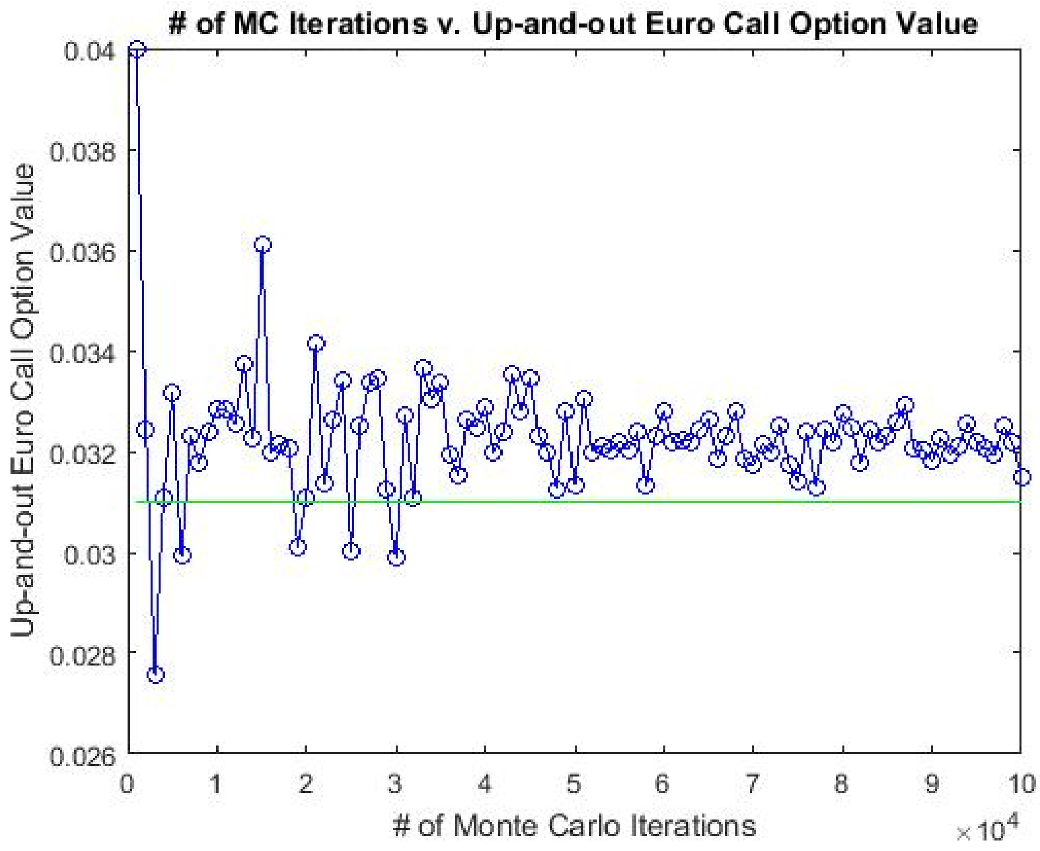


Figure 6: # of Monte Carlo iterations vs. Up-and-out Euro Call Option Value w/ $\Delta t = \frac{1}{8000}$

These results are expected; as we increase M (the number of Monte-Carlo iterations), the values of the option begin to converge to a certain value, γ. However, due to the nature of time discretization, there is a consistent error seen in the distance between the the analytical value (in green) and γ.
That is; the magnitude of time discretization changes the way in which S_T is computed by the formula

$$S(T_{n+1}) = S(t_n)e^{(r-\frac{1}{2}\sigma^2)\Delta t+\sigma\phi_n\sqrt{\Delta t}}$$

where $\{\phi_n\}$ are independent standard normals. The error seems to be consistently positive; i.e $\gamma > V_0^{\text{actual}}$.
With $\Delta t = \frac{1}{250}$ this error $(\gamma - V_0^{\text{actual}})$ is large, but when we decrease Δt to $\frac{1}{8000}$ we see that the gap between the two is much smaller. This suggests that, *theoretically speaking*, if we decreased $\Delta t \to 0$ we would see the Monte-Carlo approximation values $\gamma(\Delta t)$ converge to the analytical value $V_0^{\text{actual}})$ for large M; and if we could also let $M \to \infty$ then the approximation would be all the more accurate, as the graphs are showing less volatility in increasing M.

Code for 7(b):

```
function MC = MC_plot(deltaT)
% This function plots the results of the Monte-Carlo simulation for the
% up-and-out European call option, given a specified deltaT.

K = 9.5;        % strike price on option
S_u = 11;       % upper barrier on S
sigma = 0.22;   % volatility
r = 0.02;       % risk free rate
T = 1;          % expiry
S = 10;         % initial price on underlying asset

M = 1000:1000:100000; % range of values for the number of Monte-Carlo
                      % simulations that will be run

option_values = zeros(length(M),1); % initializing option_values

% The idea behind this loop is to range i from 1 to the number of different
% levels of M (100 such levels from 1,000 to 100,000).
% Then for every such level of M, we generate M paths (M values of S_T)
% to be used in the given formula for V_0 (equation 6 in the assignment
% pdf), which gives the option values.
for i = 1:length(M)
    paths = zeros(M(i),1);
    for j = 1:M(i)
        paths(j) = generate_path(S, T, sigma, r, deltaT, S_u);
    end
    option_values(i) = exp(-r * T) * sum(max(paths - K,0))/M(i);
end

% We create a vector for plotting the true solution of V_0 against
% the Monte-Carlo simulation's result
t = 0;
soln = zeros(length(option_values),1);
soln = soln + analytic_soln(S, K, t, T, sigma, r, S_u);

% Plot the Monte-Carlo option values against M (the number of simulations)
% together with the solid line which indicates the analytical solution.
MC = {
    figure
    plot(M, option_values,'b-o',M, soln,'g')
    title('# of MC Iterations v. Up-and-out Euro Call Option Value')
    xlabel('# of Monte Carlo Iterations')
    ylabel('Up-and-out Euro Call Option Value') };
end
```

```
function path = generate_path(S, T, sigma, r, deltaT, S_u)
% This function generates a random path of a binomial lattice with
% parameters as follows:
% S = initial price of underlying asset
% T = expiry
% sigma = volatilty
% r = risk-free rate
% deltaT = time increment
% S_u = upper barrier on the option (up-and-out call)

N = T/deltaT;  % determine the number of columns in the lattice

value = S;     % set the value to the initial value

% This loop uses the formula give in the assignment pdf for S(t+1)
% which is an iterative process; so we simply perform iterations on
% value N times
% at every step we check if the value is greater or equal to S_u, because
% if it is, we can simply return S_T (final price) = 0 and this will give
% a payoff value of 0 for the up-and-out euro call option when this
% function is called.
for i = 1:N
    value = value * exp((r - 1/2 * sigma^2) * deltaT ...
        + (sigma * randn * sqrt(deltaT)));
    if value >= S_u
        value = 0;
        break
    end
end

path = value;

end
```