CS 476: Assignment 2

Daniel Matheson; 20270871

Question 1:
Assume that Z_t is a standard Brownian motion. Apply Ito's Lemma to show that
(a) $X_t = (1 + Z_t)e^{\frac{t^2}{2}}$ satisfies $dX_t = tX_t dt + e^{\frac{t^2}{2}} dZ_t$.
(b) $X_t = Z_t^3 - 3tZ_t$ satisfies $dX_t = 3(Z_t^2 - t)dZ_t$.

Solution:
Ito's lemma: for Ito process S_t, i.e:

$$dS_t = a(S_t, t)dt + b(S_t, t)dZ_t$$

Let $\big(G(s,t) : \mathbb{R}^2 \to \mathbb{R}\big) \in C^2(\mathbb{R})$ then $Y_t = G(S_t, t)$ is also an Ito process which satisfies:

$$dY_t = \Big[\frac{dG}{dt} + a(S_t,t)\frac{dG}{dS} + \frac{1}{2}b(S_t,t)^2\frac{d^2G}{dS^2}\Big]dt + b(S_t,t)\frac{dG}{dS}dZ_t$$

(a) Since Z_t is the standard Brownian motion, it is an Ito process ($a = 0, b = 1 \ \forall \ t$)
Let $X_t = G_1(Z_t) = (1 + Z_t)e^{\frac{t^2}{2}}$. And note that the derivatives of G_1 are: (with $S = Z$)

$$\frac{dG}{dt} = t(1+S)e^{\frac{t^2}{2}}$$
$$\frac{dG}{dS} = e^{\frac{t^2}{2}}$$
$$\frac{d^2G}{dS^2} = 0$$

Therefore, applying Ito's lemma with $a = 0, b = 1$ and the above derivatives, we get that

$$\begin{aligned} dX_t &= \big[t\underbrace{(1+Z_t)e^{\frac{t^2}{2}}}_{=X_t} + 0 + \frac{1}{2}(0)\big]dt + e^{\frac{t^2}{2}}dZ_t \\ &= tX_t dt + e^{\frac{t^2}{2}}dZ_t \end{aligned}$$

(b) Similarly to part (a) we let $G_2(S_t, t) = S_t^3 - 3tS_t$ where again $a = 0, b = 1$.
Now the derivatives:

$$\frac{dG}{dt} = -3S_t$$
$$\frac{dG}{dS} = 3S_t^2 - 3t$$
$$\frac{d^2G}{dS^2} = 6S_t$$

Hence by Ito's lemma: (with $S_t = Z_t$)

$$\begin{aligned} dX_t &= \big[-3Z_t + 0 + \frac{1}{2}6Z_t\big]dt + (1)(3Z_t^2 - 3t)dZ_t \\ &= \big[-\cancel{3Z_t} + \cancel{3Z_t}\big]dt + 3(Z_t^2 - t)dZ_t \\ &= 3(Z_t^2 - t)dZ_t \end{aligned}$$

QED □

Question 2:
The variance in the stochastic volatility of the Heston model

$$dv = -\lambda(v - \bar{v})dt + \eta\sqrt{v}dZ$$

is a mean reverting Cox-Ingersoll-Ross (CIR) process.
Due to possible negative variance in the Euler-Maruyama MC simulation method for the variance, the accuracy of the option value can be poor. Show that the Milstein method for the SDE for the variance gives the following scheme:

$$v_{i+1} = \left(\sqrt{v_i} + \frac{\eta}{2}\sqrt{\Delta t}\phi_i\right)^2 - \lambda(v_i - \bar{v})\Delta t - \frac{\eta^2}{4}\Delta t$$

In addition, prove that $v_{i+1} > 0$ if $v_i = 0$ and $\frac{4\lambda\bar{v}}{\eta^2} > 1$.

Solution:

The Milstein method gives a method to approximate the solution of

$$dS_t = a(S_t, t)dt + b(S_t, t)dZ_t$$

as follows:

$$S_{t+1} = S_t + a(S_t, t)\Delta t + b(S_t, t)\Delta Z_t + \frac{1}{2}b(S_t, t)\frac{\partial b}{\partial S}(S_t, t)\left((\Delta Z_t)^2 - \Delta t\right)$$

with strong convergence rate $\gamma = 1$ and weak convergence rate $\beta = 1$
And hence, letting $S = v$ and $t = i$, we have:

$$\begin{aligned}
a = -\lambda(v - \bar{v}); \qquad b = \eta\sqrt{v}; \qquad \frac{\partial b}{\partial S} &= \frac{\eta}{2\sqrt{v}} \\
v_{i+1} &= v_i + a\Delta t + b\Delta Z_i + \frac{1}{2}b\frac{\partial b}{\partial S}\left((\Delta Z_i)^2 - \Delta t\right) \\
&= v_i - \lambda(v_i - \bar{v})\Delta t + \eta\sqrt{v_i}\Delta Z_i + \frac{1}{2}\eta\sqrt{v_i}\left(\frac{\eta}{2\sqrt{v_i}}\right)\left((\Delta Z_i)^2 - \Delta t\right) \\
&= v_i - \lambda(v_i - \bar{v})\Delta t + \eta\sqrt{v_i}\Delta Z_i + \frac{\eta^2}{4}\Big((\Delta Z_i)^2 - \Delta t\Big) \\
&= \Big(v_i + \eta\sqrt{v_i}\Delta Z_i + \frac{\eta^2}{4}(\Delta Z_i)^2\Big) - \lambda(v_i - \bar{v})\Delta t - \frac{\eta^2}{4}\Delta t \\
&\text{note that } \Delta Z_i = \sqrt{\Delta t}\phi_i \text{ where } \phi_i \sim \mathcal{N}(0, 1) \\
&= \Big(v_i + \eta\sqrt{v_i}(\sqrt{\Delta t}\phi_i) + \frac{\eta^2}{4}(\sqrt{\Delta t}\phi_i)^2\Big) - \lambda(v_i - \bar{v})\Delta t - \frac{\eta^2}{4}\Delta t \\
v_{i+1} &= \left(\sqrt{v_i} + \frac{\eta}{2}\sqrt{\Delta t}\phi_i\right)^2 - \lambda(v_i - \bar{v})\Delta t - \frac{\eta^2}{4}\Delta t \qquad \text{as desired}
\end{aligned}$$

Let $v_i = 0$ and $\frac{4\lambda\bar{v}}{\eta^2} > 1$.

$$v_{i+1} = \underbrace{\left(\sqrt{v_i} + \frac{\eta}{2}\sqrt{\Delta t}\phi_i\right)^2}_{>0 \text{ since } v_i>0} - \lambda(v_i - \bar{v})\Delta t - \frac{\eta^2}{4}\Delta t$$

$$\implies v_{i+1} > 0 \text{ if } \Big(- \lambda(v_i - \bar{v})\Delta t - \frac{\eta^2}{4}\Delta t\Big) > 0$$

$$\iff -\lambda(v_i - \bar{v})\Delta t - \frac{\eta^2}{4}\Delta t = \lambda\bar{v}\Delta t - \frac{\eta^2}{4}\Delta t > 0 \qquad \text{since } v_i = 0$$

$$\iff = \frac{4\lambda\bar{v}}{\eta^2}\Delta t - \Delta t > 0$$

$$\text{but note that } \frac{4\lambda\bar{v}}{\eta^2} > 1 \implies \frac{4\lambda\bar{v}}{\eta^2}\Delta t - \Delta t > \Delta t - \Delta t = 0$$

QED □

Question 3:
Consider a basket option on n stocks, $S^{(i)}, i = 1, 2, \ldots, n$, where the underlying prices are governed by the Heston stochastic volatility model:

$$\begin{aligned}
\frac{dS^{(i)}}{S^{(i)}} &= rdt + \sqrt{v^{(i)}}dZ^{(i)} \qquad i = 1, 2, \ldots, n \\
dv^{(i)} &= -\lambda^{(i)}(v - \bar{v}^i)dt + \eta\sqrt{v^{(i)}}dZ^{(n+i)} \qquad i = 1, 2, \ldots, n \\
\mathbf{E}(dZ^{(i)}dZ^{(n+i)}) &= \rho dt \qquad i = 1, 2, \ldots, n \\
\mathbf{E}(dZ^{(i)}dZ^{(j)}) &= \rho_{ij}dt \qquad i, j = 1, 2, \ldots, n
\end{aligned}$$

where $\lambda^{(i)}$ is the speed of reversion of the variance v to its long-term mean $\bar{v}$. Assume that returns of the stocks are independent.
Provide a pseudo-code for pricing a European basket option at $t = 0, S_0^{(i)}, i = 1, 2, \ldots, n$ with payoff $P(S^{(1)}(T), S^{(2)}(T), \ldots, S^{(n)}(T))$ at $t = T$ using a Monte Carlo with Euler approximation. Ensure that $S^{(i)}$ does not become negative using a simple correction $S = max(S, 0)$.
You may assume the existence of a function which returns the independent random variables $\phi \sim \mathcal{N}(0, 1)$. You may also assume that given a symmetric positive definite matrix $\mathcal{Q}$, there is a function which returns the Cholesky factors of $\mathcal{Q} = LL^T$.

Solution:
We will denote the function which returns the independent random variables $\phi \sim \mathcal{N}(0, 1)$ as $indnorm(n)$; with parameter n indicating the number of variables requested.
And we will denote the function which returns the Cholesky factors $\mathcal{Q} = LL^T$ as $chol(Q)$.

```
function MC_price = euro_basket_price( S_0, V_0, vbar, r, Q, ρ, T, Δt, λ, η, payoff, M )
  % Our function takes in the parameters:
  % S_0: the starting prices of the n assets
  % V_0: the volatility at time t = 0
  % vbar: the long-run average volatility (an n x 1 vector)
  % r: risk-free rate
  % Q: correlation matrix for the n assets
  % ρ: correlation of each stock to its volatility
  % T: expiry time
  % Δt: time increment
  % λ: a (n x 1) vector of 'reversion to mean' speeds for each asset
  % η: parameter in the SDE for volatility
  % payoff: the payoff function
  % M: the number of Monte Carlo simulations

  N = round(T/Δt) ; % find the number of time steps required
  n = length(S_0);       % number of assets
  indphi = zeros(2*n); % initializing for 2n independent standard normals
  phi = zeros(2*n);    % initializing for 2n correlated standard normals
  prices = S_0;          % initializing prices
  vol = V_0;             % initializing volatility
  % initializing the correlation matrix for both prices and volatility
  C = zeroes(2*n, 2*n);
  % setting the values for the correlation matrix
  C(1:n,1:n) = Q;
  C(n+1:, n+1:) = diag(zeros(n) + 1);
  C(1:n, n+1:) = diag(zeros(n) + rho);
  C(n+1:, 1:n) = diag(zeros(n) + rho);
  % i.e, C looks like 4 (n x n) matrices put together:
```

$$\begin{bmatrix} \rho_{11} & \rho_{12} & \cdots & \rho_{1n} & \rho & 0 & \cdots & 0 \\ \rho_{21} & \rho_{22} & \cdots & \rho_{2n} & 0 & \rho & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \rho_{n1} & \rho_{n2} & \cdots & \rho_{nn} & 0 & 0 & \cdots & \rho \\ \rho & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & \rho & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \rho & 0 & 0 & \cdots & 1 \end{bmatrix}$$

```
  L = chol(C);                % Cholesky decomposition of C
  final_prices = zeros(M);    % initializing final price vector

  for i = 1:M                 % for M Monte-Carlo simulations
    for j = 1:N               % for N time intervals
      indphi = indnorm(2n);       % create 2n independent standard normals
      phi = (L * indphi);         % correlate the 2n standard normals
      % first solve for the next step of volatility using Euler's method,
    making sure not to have negative volatility; otherwise √vol yields an error
      vol = max(vol - λ*Δt*(vol - vbar) + η*√vol*√Δt*phi(n+1:), 0);
      % solve for next step of asset prices using vol and Euler's method,
    keeping the prices above 0 as required
      prices = max(prices + r * prices * Δt + prices*√vol*√Δt*phi(1:n), 0);
    end
    final_prices(i) = prices; % remember the final prices generated by random
    paths
```

```
    prices = S_0; % reset prices to starting prices in order to repeat MC algo
    vol = V_0;    % reset volatility in order to repeat MC algo
  end
  % Compute Monte-Carlo price of the European basket option using the formula
    from Assignment 1; i.e taking the discounted average of the payoffs from
    all of the random paths generated
  MC_price =  exp(-r * T) * (sum_{i=1}^{M} payoff(final_prices(i))) / M;
end
```

Question 4a

```
function [V_0, L] = binomialDelta(S_0, r, sigma, T, N, fpayoff)
% This function finds the fair-value of an option with payoff fpayoff on
% the underlying asset with initial price S_0, using a binomial lattice
% with N rows. It also returns a matrix which represents the underlying
% values and the corresponding deltas at every node of the lattice
% S_0 = initial underlying price
% r = risk-free rate
% sigma = volatility
% T = time to expiry
% N = number of timesteps/rows in lattice
% fpayoff = payoff function of option

% Initializing data needed for structured array L later
field_S = 'S';
field_delta = 'delta';
values_S = zeros(N+1, N+1);
values_delta = zeros(N, N);

deltaT = T/N;  % timestep

u = exp(sigma * sqrt(deltaT));   % rate at which stock price increases
d = exp(- sigma * sqrt(deltaT)); % rate at which stock price decreases
q = (exp(r * deltaT) - d)/(u-d); % risk neutral probability measure

S_row = zeros(N+1,1);  % initializing the row of prices
S_row(1) = S_0;        % first row is S_0
values_S(1,:) = S_row; % first row of matrix of prices is S_0

%%%%
% Psedo-code
%   for i = 1:N (timesteps)
%
```

$$S_j^{n+1} = \begin{cases} uS_j^n & \text{if } j = n \\ dS_j^n & \text{otherwise} \end{cases}$$

```
%       i.e: the top node gets multiplied by u and all others by d
%       Then assign this new generated row to the matrix of prices
%%%%
for i = 1:N
  S_row(i+1) = u * S_row(i);
  S_row(1:i) = d * S_row(1:i);

```

```
  values_S(i+1,:) = S_row;
end

% Now we do the option values:
option_values = fpayoff(S_row); % Last row of option values is payoff
option_matrix = zeros(N+1,N+1); % Initialize matrix of option values
option_matrix(N+1,:) = option_values; % Last row is payoff
%%%%
% Pseudo-code
%   for n = N:-1:1 (step backwards through lattice rows starting at second
%                       last row)
%       for j = 1:n (step through nodes in each row)
%           V_j^n = e^{-r*deltaT} (q*V_{j+1}^{n+1} + (1-q)*V_j^{n+1})
%       end
%   put the new generated option values into the matrix of option values
%%%%
for n = N:-1:1
  for j = 1:n
    option_values(j) = exp(- r * deltaT) * (q * option_values(j+1) + ...
      (1-q) * option_values(j));
  end
  option_values(n+1:N+1) = 0;
  option_matrix(n,:) = option_values;
end

% The fair-price for the option is then the first node's option value
V_0 = option_values(1);

% This is a vectorized equation to solve for the matrix of delta values, it
% represents the following equation:
%  delta_j^n = (V_{j+1}^{n+1} - V_j^{n+1}) / (S_{j+1}^{n+1} - S_j^{n+1})

values_delta = (option_matrix(2:end,2:end) - option_matrix(2:end,1:N)) ...
               ./ (values_S(2:end,2:end) - values_S(2:end,1:N));

% Some values of delta become "NaN" due to the fact that the matrices are
% sparse and has a lot of zero values.
values_delta(isnan(values_delta)) = 0;
% reduces the delta matrix to lower triangular (other values are
% irrelevant)
values_delta = tril(values_delta);

% return L
L = struct(field_S, values_S, field_delta, values_delta);

end
```

Question 4b

```
function delta = interpDelta(delta_n, S_n, S)
% This function returns linear interpolations for delta values given a
% vector of reference deltas (delta_n) and their prices(S_n); then uses
% these two vectors to interpolate the delta values for the given prices S

% find max/min of S_n
S_max = max(S_n);
S_min = min(S_n);
% set length of the vector to initialize the new deltas
n = length(S);
delta = zeros(n,1);

%%%
% Pseudo-code
%   delta = linear intepolations from prices S using delta_n and S_n as
%               reference points
%   whenever S > max(S_n), set delta for this S to be the highest delta_n
%   whenever S < min(S_n), set delta for this S to be the lowest delta_n
%%%
delta = interp1(S_n, delta_n, S);
delta(S > S_max) = delta_n(end);
delta(S < S_min) = delta_n(1);

end
```

Question 4c

```
function plot_out = plot_delta()
% This function plots hedging delta values  vs. various prices given for the
% underlying asset. It uses a european call option.
% The plot has two lines; which represent our simulated value for the
% deltas vs. the built-in MATLAB function's values (blsdelta)

sigma = 0.20;                   % volatility
r = 0.03;                       % risk-free rate
T = 1;                          % time to expiry
K = 90;                         % strike price
S_0 = 90;                       % initial underlying value
N = 250;                        % number of timesteps
fpayoff = @(x)max(x - K,0);     % payoff function
deltaT = T/N;                   % deltaT

% stock prices
S = linspace(80,140,100);

% get the European call option value, with associated prices and delta
% values from the binomial lattice
[V_0, L] = binomialDelta(S_0, r, sigma, T, N, fpayoff);

% extract deltas and prices from the second last column of the binomial
% lattice
delta_n = L.delta(end,:);
S_n = L.S(end-1,1:end-1);
```

```
28 % find the deltas for the values of S from 80 to 140 using interpolation
   interp_delta = interpDelta(delta_n, S_n, S);
30 % use built-in function to find the same deltas
   real_delta = blsdelta(S, K, r, deltaT, sigma, 0);
32
   % plot interp_delta vs. real_delta
34 plot_out = {
     figure
36   plot(S, interp_delta,'b',S, real_delta,'g')
     title('Interpolated Delta vs. Black Sholes Delta')
38   xlabel('Underlying Price')
     ylabel('Delta')
40   legend('Interpolated Delta', 'Black Sholes Delta', 'Location', 'southeast')
     ylim([-0.1,1.1])   };
42
   end
```

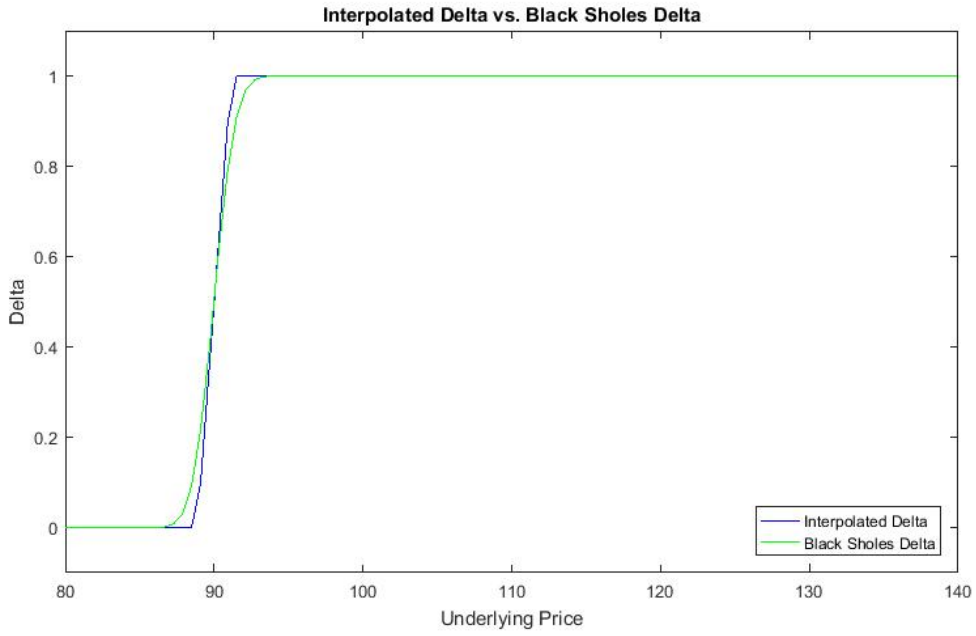

Figure 1: Interpolated δ vs. Black-Sholes δ

Question 5a

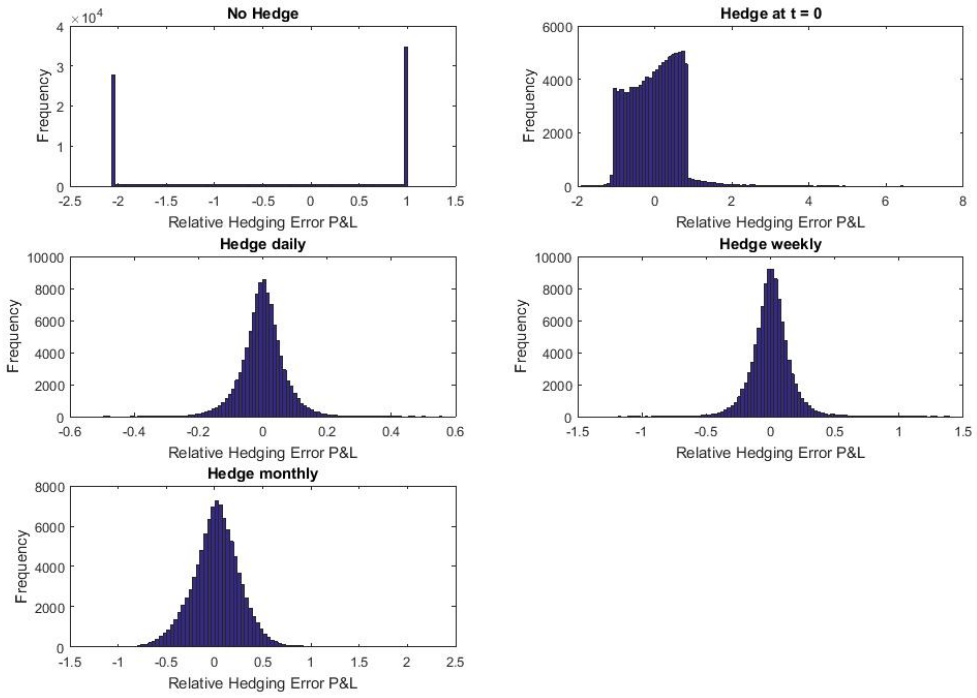

Figure 2: Relative Hedging Error for various rebalancing times

Comments

Note that, throughout Question 4, we were asked to use 10,000 Monte-Carlo simulations. These results were obtained using 100,000.

No Hedging: Clearly two values stand out dramatically: 1 and -2. This should be expected due to the nature of the option we are hedging. The value of the error is 1 in this case where we did not hedge at all, whenever the option expires as worthless (since we are the writer of the option). The error is -2 whenever the option is at its highest possible value of \$20.

Other methods: Clearly, as one should expect; we see that the higher the frequency of hedging, the less relative hedging error we encounter - this is exhibited by the lower spread (variance) of the data around 0 as we increase the frequency of rebalancing from monthly to weekly, then daily. One strange phenomenon that seems to appear is that in the case of hedging only once at $t = 0$, there are more positive large values for the error as opposed to the other histograms that appear to be essentially symmetric.

Code: The first function calculates the hedging errors.

```
function errors = hedging_error(S_0, sigma, r, mu, T, N, K, fpayoff, M)
% This function computes the relative hedging errors for a strategy using
% the following parameters:
% S_0 = the initial price of the underlying asset
% sigma = volatility
% r = risk-free rate
% mu = geometric brownian motion drift factor
% T = time to expiry
% N = number of hedges to be performed in 250 days; equally spaced
% K = strike price
% fpayoff = option payoff function
% M = number of Monte-Carlo simulations to be used

% initializing stock values
S = zeros(M,1) + S_0;

% creating a binomial lattice with 250 trading day nodes
[V_0, L] = binomialDelta(S_0, r, sigma, T, 250, fpayoff);

deltaT = T/N;                          % timestep
delta = zeros(M,1) + L.delta(1,1); % initializing delta values
B = V_0 - L.delta(1,1) * S_0;      % initializing cash account

%%%%
% Pseudo-code/explanation
% This code is solely for the "no hedging" scenario (number of hedges = 0)
%   We set  B_0 = V_0
%   then S_T =  S_0 * e^((mu - 1/2 sigma^2)T + sigma phi sqrt(T))  where  phi ~ N(0,1)
%   then  Pi_T = -V_T + B_0 e^(rT)
%   and we return  e^(-rT) Pi_T / V_0
%%%%
if N == 0
  B = V_0;
  S = S .* (exp((mu - 1/2 * sigma^2) * T + ...
  (sigma * randn(M,1) * sqrt(T))));
  portfolio_values = - fpayoff(S) + B * exp(r * T);
  errors = (exp(-r * T) * portfolio_values)/V_0;
  return
end

%%%%
% Pseudo-code
%   for i = 2:floor(N) (start at the second row, to the floor of the number
%                of hedges
%       generate new stock prices using formula:
%            S_{t+1} = S_t e^((mu - 1/2 sigma^2) Delta t + sigma phi sqrt(Delta t))  where  phi ~ N(0,1)


%       keep track of  delta_{t+1} and delta_t  for the
%       calculation of the new cash account balance
%
%       use the linear interpolation function interpDelta to find the new
%       values of deltal keeping in mind that the for loop index runs from
```

```
%       2 to floor(N) but there are 250 rows of L.delta and L.S; so we
%       reference row (or trading day) i*(250/N) at each step
%
%       update cash account using formula:
%        B_{t+1} = B_t e^{r\Delta t} + (\delta_{old} - \delta) S_t

%%%%
for i = 2:floor(N)
  S = S .* (exp((mu - 1/2 * sigma^2) * deltaT + ...
  (sigma * randn(M,1) * sqrt(deltaT))));
  delta_old = delta;
  delta = interpDelta(L.delta(i*(250/N),1:i*(250/N)), L.S(i*(250/N),1:i*(250/N
    )), S);
  B = B * exp(r * deltaT) + (delta_old - delta) .* S;
end

%%%%
% Pseudo-code/explanation
% The reason for this if statement is to account for the fact that the
%   binomial lattice we generated has 250 rows but there are times when
%   we selected hedging days that do not divide nicely into 250.
%       i.e: the monthly hedging every 20 days
%   therefore, we must not find  S_T or B_T using \Delta t  in
%   this case but rather  \Delta t|N - floor(N)|
%       i.e: for the 20 day hedging, there remain 10 days at the end
%       between the last hedge and the 250th trading day; so we must
%       multiply  \Delta t  by 0.5.
%
%   In both cases, once the appropriate  \Delta t  is found, we are
%   simply applying the formulas from the for loop above one more time in
%   order to get  S_T and B_T  for use in determining the value of
%    \Pi_T
%%%%
if N ~= floor(N)
  S = S .* (exp((mu - 1/2 * sigma^2) * deltaT*(abs(N - floor(N))) + ...
  (sigma * randn(M,1) * sqrt(deltaT*(abs(N - floor(N)))))));
  B = B * exp(r * deltaT*(abs(N - floor(N))));
else
  S = S .* (exp((mu - 1/2 * sigma^2) * deltaT + ...
  (sigma * randn(M,1) * sqrt(deltaT))));
  B = B * exp(r * deltaT);
end

% Find hedging portfolio values based on generated values
% \Pi_T = -V_T + \delta S_T + B_T
portfolio_values = - fpayoff(S) + delta .* S + B;

% return the relative errors
% errors = \frac{e^{-rT}\Pi_T}{V_0}
errors = (exp(- r * T) .* portfolio_values)/V_0;

end
```

And the second function creates the histograms:

```
function MC_hist = MC_delta_hedge()
% This function creates histograms of the relative hedging errors for no
% hedging, hedging once at t=0, and hedging daily/weekly/monthly.

sigma = 0.20;    % volatility
r = 0.03;        % risk-free rate
mu = 0.10;       % geom. brownian motion model for underlying asset drift
T = 1;           % time to expiry
K = 90;          % strike price
S_0 = 90;        % initial underlying price

fpayoff = @(x)min(max(x - K,0),20);    % option payoff

M = 100000;      % number of simulations (10 times the minimum asked)

n = [0,1,250,50,12.5];    % number of hedges we will run

errors = zeros(M,5);      % initializing data structure
% histogram labels
labels = {'No Hedge','Hedge at t = 0', 'Hedge daily', 'Hedge weekly', ...
  'Hedge monthly'};

%%%%
% Pseudo-code
%   for i = 1:5 (number of different histograms / number of hedges to run)
%       errors = find relative hedging errors with given number of hedges
%                    in 250 days
%       plot histogram with n(i) hedges
%%%%
for i = 1:5
  errors(:,i) = hedging_error(S_0, sigma, r, mu, T, n(i), K, fpayoff, M);
  subplot(3,2,i)
  hist(errors(:,i), 100)
  title(labels(i))
  xlabel('Relative Hedging Error P&L')
  ylabel('Frequency')
end
```

Question 5b

```
function [var,cvar] = dVaRCVaR(PL, beta)
% This function returns the VaR and CVaR of the vector of values PL which
% in this specific use indicates the relative P&L of our hedging strategy;
% with confidence (1 - beta)

% sort the P&L
sorted_PL = sort(PL);
% find the index at which VaR is located in a discrete distribution
var_index = floor(length(PL) * (1 - beta));
% VaR is the above index in sorted P&L
var = sorted_PL(var_index);

% CVaR is the discrete conditional expectation. Therefore we sum all values
% below and including VaR, and divide by the number of values
cvar = 1/var_index * (sum(sorted_PL(1:var_index)));

end
```

Question 5c

	VaR	CVaR	Mean	Std_Dev
No Hedging	-2.0644	-2.0644	-0.39738	1.3123
Hedge Once @ t=0	-0.96743	-1.0365	-0.0082142	0.61176
Daily Hedging	-0.10725	-0.14918	-0.00024758	0.065363
Weekly Hedging	-0.23635	-0.33373	0.0027894	0.15078
Monthly Hedging	-0.4061	-0.53733	0.019264	0.25289

As is clear from the table above, hedging performance is much better as we increase the frequency of our hedging portfolio rebalancing.
With respect to VaR: The value of VaR is highest for daily hedging, indicating that 95% of the errors are above this value. This is very good. As we decrease the frequency of hedging, VaR decreases.
With respect to CVaR: Again, the highest value is for daily hedging, meaning that the expected loss in the 5% worst case scenarios are low in magnitude. As we decrease the frequency of our hedging, the magnitude of loss increases.
With respect to the mean: The mean of daily hedging is nearly 0, and in the histogram it appeared to be approximately normally distributed. This is expected by Central Limit Theorem.
There is not a significant difference in means, they are all fairly close to 0, except for the case of no hedging whatsoever, where we have a negative mean. This is not good, and means that on average we make a loss.
With respect to standard deviation: The standard deviation is lowest for daily hedging, and increases as we decrease the frequency of rebalancing. This indicates that we are lowering our risk by hedging more often.

In practice, of course, one has to balance these requirements with transaction costs, liquidity issues, transaction execution analysis if the positions are very large etc.

Code:

```
function PL_table = PL_VaRCVaR_Table()
% This function creates a table that displays the VaR, CVaR, mean and
% standard deviation of P&L for no hedging, hedging once at t = 0, and
% hedging daily/weekly/monthly

beta = 0.95;     % (1-beta) is the VaR/CVaR confidence level
sigma = 0.20;    % volatility
r = 0.03;        % risk-free rate
mu = 0.10;       % geom. brownian motion model for underlying asset drift
T = 1;           % time to expiry
K = 90;          % strike price
S_0 = 90;        % initial underlying price

fpayoff = @(x)min(max(x - K,0),20);  % option payoff

M = 100000; % number of simulations (10x minimum)

n = [0, 1,250,50,12.5];  % number of hedges we will run
% initializing vectors
errors = zeros(M,5);
var = zeros(5,1);
cvar = zeros(5,1);
sta_dev = zeros(5,1);
means = zeros(5,1);

%%%%
% Pseudo-code
%   for i = 1:5 (number of different values for n(i); number of hedges)
%       errors = find relative hedging errors with given number of hedges
%                    in 250 days
%       find VaR, CVaR of errors
%       find standard deviation and mean of errors
%%%%
for i = 1:5
  errors(:,i) = hedging_error(S_0, sigma, r, mu, T, n(i), K, fpayoff, M);
  [var(i), cvar(i)] = dVaRCVaR(errors(:,i), beta);
  sta_dev(i) = std(errors(:,i));
  means(i) = mean(errors(:,i));
end

% row/column names for the table
columns = {'VaR', 'CVaR', 'Mean', 'Standard_Deviation'};
rows = {'No Hedging', 'Hedge Once @ t=0', 'Daily Hedging', ..
  'Weekly Hedging', 'Monthly Hedging'};
% show table
T = table(var,cvar,means,sta_dev, 'VariableNames', columns, 'RowNames', rows)

end
```

Question 6a

Timesteps	Num_Simulations	Euro_Call_Value
800	25000	6.5769
1600	1e+05	6.4607
3200	4e+05	6.5239

Code:
The first function does the majority of the work by finding the stock prices under the jump diffusion model, with jumps following the double exponential distribution rather than the log normal distribution.

```
function prices = jump_diff_prices(N, M, T, sigma, S_0, eta1, eta2, p_up,
    lambda, r)
% This code was taken from the supplementary notes, and modified to current
% needs. The purpose of the function is to generate M price paths according
% to the risk neutral jump diffusion model.
% N: represents the number of timesteps required.
% T: is the time to expiry
% sigma: is the volatility of the underlying asset
% eta1 and eta2: are the parameters for the double-exponential distribution
% of the jump size
% p_up: is the probability of a jump being an 'up-jump', given that a jump
% has occured
% lambda: is mean arrival time of a jump (Poisson process)
% r: is the risk free rate

deltaT = T/N; % timestep

% compensated drift E[J-1]
kappa = (p_up * eta1)/(eta1 - 1) + ((1-p_up)*eta2)/(eta2 + 1) - 1;

% compensated drift for X = log(S)
drift = (r - (sigma^2)/2.0 - lambda*kappa);

% X = log(S)
X_old(1:M,1) = log(S_0);
X_new(1:M,1) = zeros(M, 1);

% initializing variables
jump_chek = zeros(M, 1);
jump_size = zeros(M, 1);
jump_mask = zeros( M, 1);
jump_dir_chek = zeros(M,1);
jump_up = zeros(M,1);

%%%%
% Pseudo-code
%
% for (number of timesteps)
%   jump_check = M random number in [0,1]
%   jump_mask  = indices where jump_check <= jump probability
```

```
%                                                 (= lambda * deltaT)
%   jump_direction_check = M random numbers in [0,1]
%   jump_up = indices where jump_direction_check <= probability of a jump
%                         going up ( = p_up )
%   jump_size = f(y); the probability distribution given on the assignment
%    X_{t+1} = X_t + drift * Δt + σ√Δt * N(0,1) + jump
%   where X is the log of the stock price, and N(0,1) is the std. normal
%%%%
for i=1:N        % timestep loop
% check if jump happens
jump_chek(:,1) = rand(M,1);
jump_mask(:,1) = ( jump_chek(:,1) <= lambda * deltaT);

% jump direction
jump_dir_chek(:,1) = rand(M,1);
jump_up(:,1) = ( jump_dir_chek(:,1) <= p_up);

% jump size
jump_size(:,1) = p_up * exprnd(1/eta1,M,1) .* jump_up(:,1) + (1 - p_up) *
    exprnd(1/eta2,M,1) .* (1 - jump_up(:,1));
jump_size = jump_size.*jump_mask;

% update X
X_new(:,1) = X_old(:,1) + drift(:,1)*deltaT +sigma*sqrt(deltaT)*randn(M,1)
             + jump_size(:,1);
X_old(:,1) = X_new(:,1);
end

% S = exp(X)
S(:,1) = exp( X_new(:,1) );
% return S
prices = S(:,1);

end
```

The second function calculates the European call option value with an underlying asset that follows the jump diffusion model, using the function above

```
function value = euro_call_jumpdiff()
% This function outputs 3 European call option values for different
% timesteps and number of simulations, under the jump diffusion model.
% It uses the jump_diff_prices() function to accomplish most of the work.
% It takes the prices generated by that function and applies the
% Monte-Carlo option pricing formula (the discounted mean of the payoffs)
% The results are outputted in a table

sigma = 0.15;                % volatility
r = 0.05;                    % risk-free rate
T = 0.5;                     % time to expiry
S_0 = 100;                   % initial underlying price
K = 100;                     % strike price
fpayoff = @(x)max(x - K,0); % option payoff
eta1 = 3.0465;               % exponential distribution parameters used for
eta2 = 3.0775;                  % determining jump sizes
```

```
p_up = 0.3445;                  % probability that a jump is positive
lambda = 0.1;                   % Poisson process arrival rate of jumps

% Matrix representing the number of timesteps and simulations desired
NM_matrix = [ 800, 25000
              1600, 100000
              3200, 400000];

euro_call_values = zeros(3,1); % initializing

%%%%
% Pseudo-code
%   for i = 1:3
%       N = i^th row, first column of NM_matrix (i.e i^th timesteps value)
%       M = i^th row, second column of NM_matrix (i^th number of
%           simulations)
%       prices = jump diffusion model prices
%       option values = discounted mean payoff of prices
%%%%

for i = 1:3
  N = NM_matrix(i,1);
  M = NM_matrix(i,2);
   prices = jump_diff_prices(N, M, T, sigma, S_0, eta1, eta2, p_up, lambda,r);
   euro_call_values(i) = exp(- r * T) * mean(fpayoff(prices));
end

% returns the table
columns = {'Timesteps', 'Num_Simulations','Euro_Call_Value'};
T = table(NM_matrix(:,1), NM_matrix(:,2), euro_call_values, 'VariableNames',
    columns)

end
```

Question 6b

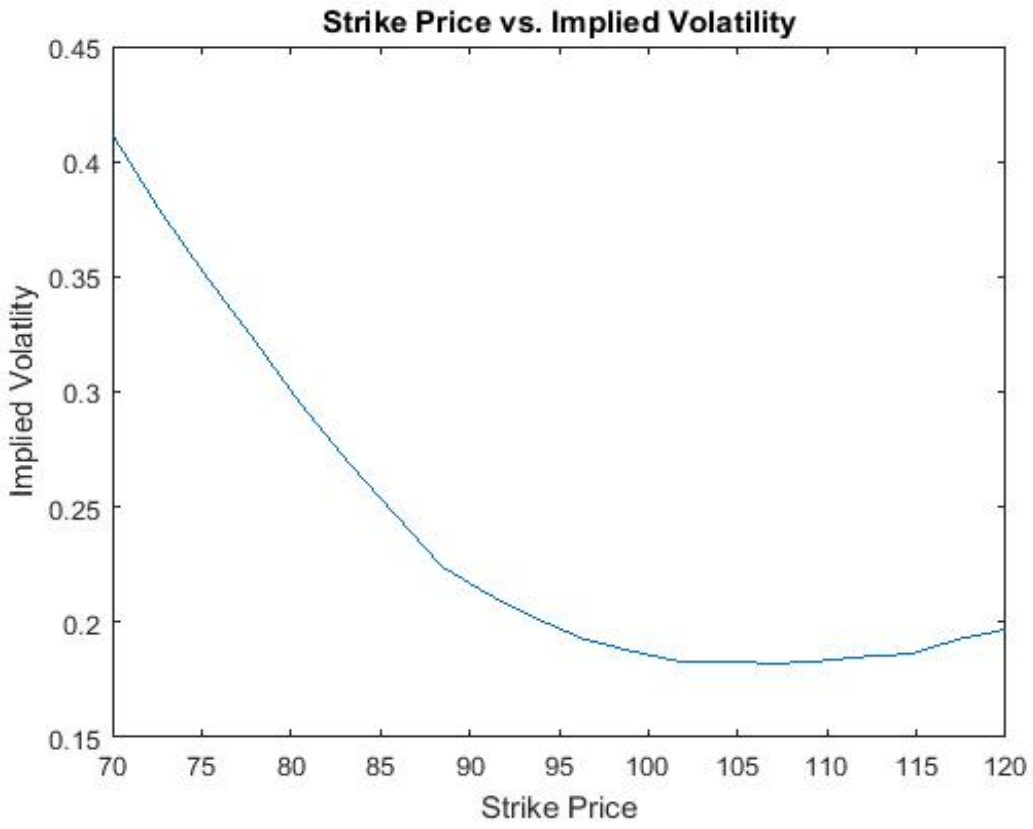

Figure 3: Volatility Smile; Strike Price vs. Black-Sholes Implied Volatility

The jump model exhibits a somewhat linear relationship between strike price and implied volatility for strike prices under 90. Afterwards, we begin to see diminishing marginal effect of strike price in lowering implied volatility.

If the Black-Sholes model were correct, this graph should simply show a linear relationship all the way across the board. However this is clearly not the case, and what this "*Volatility Smile*" (upward curve toward the right end of the graph) suggests is that there is a higher demand for options which are in-the-money and out-of-the-money, perhaps due to hedging activities. The reason we conclude there is such a demand is because a high demand for options would push the price up, which in turn increases the implied volatility of the underlying asset.[1] And clearly the lowest volatility is centered around $S_0 = \$100$; so the further the strike price gets from S_0, the larger the premium expected for the option; with the exception of extremely out-of-the-money call options below $\sim \$90$ strike price.

It is worth noting that the volatility smile was first seen after the 1987 stock market crash, and was not present before. This may be due to investor fears of another crash - meaning that they charge a higher premium on more risky options (deeper in/out the money) as a sort of insurance.

[1] see `http://www.investopedia.com/terms/v/volatilitysmile.asp`

Code

```
function implied_vol = implied_vol_graph()
% This function graphs the implied volatility of the option prices under
% the jump model with 3200 timesteps and 400,000 Monte-Carlo simulations
% against the strike price randing from 0.7*S_0 to 1.2*S_0 where S_0 is the
% initial price of the underlying.

N = 3200;       % timesteps
M = 400000;     % simulations


S_0 = 100;      % initial underlying price
K = linspace(0.7 * S_0, 1.2 * S_0, 20); % strike values
sigma = 0.15;  % volatility
r = 0.05;       % risk-free rate
T = 0.5;        % time to expiry
eta1 = 3.0465; % exponential distribution parameters used for
eta2 = 3.0775;      % determining jump sizes
p_up = 0.3445; % probability that a jump is positive
lambda = 0.1;  % Poisson process arrival rate of jumps

vol = zeros(20,1); % initialize

%%%%
% Pseudo-code
%   for i = 1:20
%       prices = jump diffusion model prices
%       payoff = european call option payoff with strikes K
%       value of option = discounted mean of Monte-Carlo path payoffs
%       volatility = black sholes implied volatility
%%%%
for i = 1:20
  prices = jump_diff_prices(N, M, T, sigma, S_0, eta1, eta2, p_up, lambda,r);
  payoff = max(prices - K(i),0);
  value = exp(- r * T) * mean(payoff);
  vol(i) = blsimpv(S_0, K(i), r, T, value);
end

% plot
implied_vol = {
  figure
  plot(K, vol)
  title('Strike Price vs. Implied Volatility')
  xlabel('Strike Price')
  ylabel('Implied Volatlity')   };

end
```