CS 676 Assignment 3

Ying Kit Hui (Graduate student)

March 18, 2022

Listings

Precautions: Note that some of the discussions are in the comments of the code. They are usually under a separate section inside the codes. This is done because 1) avoid too much replication and 2) lack of time to edit it and put it in latex format. Please let me know if you strongly prefer them to be presented in latex.

1 Q1

1.1 Q1a

Given r_n at t_n, the Euler-Maruyama formula for computing the interest rate r_{n+1} at t_{n+1} is:

$$r_{n+1} = r_n + a(b - r_n)\Delta t + \sigma\sqrt{r_n}\sqrt{\Delta t}\,\phi_t, \tag{1}$$

where $\phi_t \sim \mathcal{N}(0, 1)$.

Assume $r_n = b$, then equation (1) becomes:

$$r_{n+1} = b + \sigma\sqrt{b}\sqrt{\Delta t}\,\phi_t \tag{2}$$

Noting that $a, b, \sigma > 0$, thus, $r_{n+1} < 0$ if and only if $b + \sigma\sqrt{b}\sqrt{\Delta t}\,\phi_t < 0$ if and only if $\phi_t < \frac{-b}{\sigma\sqrt{b}\sqrt{\Delta t}} = \frac{-\sqrt{b}}{\sigma\sqrt{\Delta t}}$. Thus the required condition on the standard normal sample ϕ_t so that $r_{n+1} < 0$ is

$$\phi_t < \frac{-\sqrt{b}}{\sigma\sqrt{\Delta t}}. \tag{3}$$

Note that generally interest rate r is positive or at least nonnegative. It is very rare for the market to have negative interest rate, it is often the result of some extreme monetary policy trying to tackle deflation and encourage people to spend and invest. Moreover, nonnegative interest rate is often an assumption in some mathematical models for finance. Thus, we want to make sure the computed interest rate r_{n+1} is nonnegative.

Moreover, the constant b in the mean-reverting process for interest rate is the 'long term mean level' of the interest rate. So the assumption that $r_n = b$ actually consider the case that when the interest rate just reverted to the mean b, what is the condition so that the next computed interest rate r_{n+1} is less than 0. This condition is just $\phi_t < \frac{-\sqrt{b}}{\sigma\sqrt{\Delta t}}$. Note that as $\Delta t \to 0$, $\frac{-\sqrt{b}}{\sigma\sqrt{\Delta t}} \to -\infty$ and so it is increasingly unlikely to have $r_{n+1} < 0$.

1.2 Q1b

Given r_n at t_n, the Milstein method for computing r_{n+1} is:

$$\begin{aligned} r_{n+1} &= r_n + a(b - r_n)\Delta t + \sigma\sqrt{r_n}\Delta Z(t_n) + \frac{1}{2}\sigma\sqrt{r_n}\frac{\sigma}{2\sqrt{r_n}}\left((\Delta Z(t_n))^2 - \Delta t\right) \\ &= r_n + a(b - r_n)\Delta t + \sigma\sqrt{r_n}\phi_t + \frac{\sigma^2}{4}(\phi_t^2 - \Delta t), \end{aligned} \tag{4}$$

where $\phi_t \sim \mathcal{N}(0,1)$ since $\Delta Z(t_n) = Z(t_n + \Delta t) - Z(t_n) \sim \mathcal{N}(0,1)$.

Assuming $r_n = 0$, the Milstein method (4) becomes:

$$r_{n+1} = ab\Delta t + \frac{\sigma^2}{4}(\phi_t^2 - \Delta t) \tag{5}$$

Hence, noting that $a, b, \sigma > 0$, $r_{n+1} > 0$ if and only if $ab\Delta t + \frac{\sigma^2}{4}(\phi_t^2 - \Delta t) > 0$ if and only if $\phi_t^2 > \frac{-4ab\Delta t}{\sigma^2} + \Delta t = \Delta t(\frac{-4ab}{\sigma^2} + 1)$. Note that $\Delta t > 0$ and $\phi_t^2 \geq 0$. Hence, a sufficient condition for $\phi_t > \Delta t(\frac{-4ab}{\sigma^2} + 1)$ is $\frac{-4ab}{\sigma^2} + 1 < 0$. After some algebraic manipulation, an inequality using parameters a, b, σ which guarantees that r_{n+1} is always positive is given by $\frac{\sigma^2}{4} < ab$. [1]

[1] This mean-reverting process for interest rate is called CIR model. There is one condition: $\frac{\sigma^2}{4} \leq ab$ as stated in Wikipedia that it will preclude $r = 0$. Is it related to the condition we derived here?

2 Q2

Use the notation as in the question. Denote the bond value by B. Note that in the following pseudo code, vectorized approach is employed. Also, we assume operations between scalars and vectors will be done by broadcasting the scalars appropriately.

Note that we assume the existence of a function which returns a random variable $\phi \sim \mathcal{N}(0,1)$. For simplicity, by $randn(M,1)$ we mean it generates M random variables that are $\mathcal{N}(0,1)$, stored in a row vector. Moreover, we make an additional assumption that whenever we call $randn(M,1)$, the resulting random variables are independent among themselves and also independent of all previously generated random variables.

Lastly, we also uses $ones(M,1)$ to mean it generates a M-by-1 row vector with each entry being 1.

Procedure 1 Monte Carlo method for bond value

Input: $M, N, T, a, b, \sigma, r_0$ ▷ scalars inputs
Output: B ▷ bond value obtained via MC

$\Delta t = T/N$
$r_0 = ones(M,1) \cdot r_0$
integral $= 0$
for $j = 1, \ldots, N$ **do**
  $\phi = randn(M,1)$
  ▷ M independent $\mathcal{N}(0,1)$
  $r_j = r_{j-1} + a(b - r_{j-1})\Delta t + \sigma\sqrt{r_{j-1}}\phi + \frac{\sigma^2}{4}(\phi^2 - \Delta t)$
  ▷ component-wise
  integral $= \Delta t(r_j + r_{j-1})/2 +$ integral
  ▷ component-wise, trapezoidal rule
end for
$B = \text{mean}(\exp(-\text{integral}))$

---

3 Q3

3.1 Q3a

Listing 1: binomialDeltaStraddle.m

```
% Q3a
function [V0,S, delta] = binomialDeltaStraddle(S0,r,sigma, T,N,K)
% V0 option price at time 0
% S, matrix of underlying prices, the i-th column represent t_{i+1}, only
% store until t_{N-1}
% delta, matrix of delta, same storage as S

dt = T/N;
% up and down ratio in bin. model
u = exp(sigma * sqrt(dt));
d = exp(-sigma * sqrt(dt));
% risk neutral probability of having an up
q = (exp(r * dt) - d) / (u-d);

S = zeros(N,N);
delta = zeros(N,N);

% the stock values at final time T
%          values are arranged in ascending (from top to bottom) order
Svec = S0*d.^([N:-1:0]') .* u.^([0:N]');

% final payoff
W = max(Svec - K,0) + max(K - Svec,0);

% fill in S and delta, column by column
for i = N:-1:1
    Svec = Svec(2:i+1) ./ u;
    S(1:i,i) = Svec;
    delta(1:i,i) = (W(2:end) - W(1:end-1)) ./ ((u-d)*Svec);
    % obtain option values at (i-1)th timestep, by risk neutral valuation
    W = exp(-r *dt) *(q* W(2:i+1) + (1-q)* W(1:i));
end

V0 =W;
end
```

3.2 Q3b

Listing 2: interpDelta.m

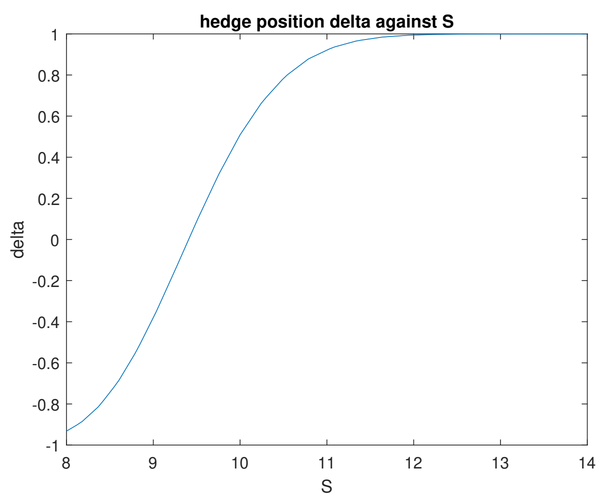
```
% Q3b
function delta = interpDelta(delta_n,S_n, S)
% input: column vectors
%       delta_n, S_n are layers of delta and S at a time
% output: column vectors
%       delta: interpolated delta

delta = zeros(size(S));
ub = max(S_n);
lb = min(S_n);

idx = lb <= S & S <= ub;
% normal linear interpolation for S that is in range
delta(idx) = interp1(S_n, delta_n, S(idx));

I = find(~idx);
% disp(I)

% note that the index array better to be a row vector, otherwise we have
% different behavior

% use S is out of range, use nearest S_n's delta
for i = I'
    [~, idx2] = min(abs(S_n - S(i)));
    delta(i) = delta_n(idx2);
end

end
```

3.3 Q3c

Listing 3: Q3c.m

```
%Q3c

sigma = 0.2;
r = 0.03;
mu = 0.15;
T = 1;
S0 = 10;
K = 0.95 * S0;
```

```
N = 250;

% query points
S_query = linspace(0.8*S0, 1.4*S0, 100);

[V0,S, delta] = binomialDeltaStraddle(S0,r,sigma, T,N,K);

n = 0.8*N;
delta_n = delta(1:n+1,n+1);
S_n = S(1:n+1, n+1);

delta_query = interpDelta(delta_n,S_n, S_query);

plot(S_query, delta_query)
title('hedge position delta against S')
xlabel('S')
ylabel('delta')
```



3.4 Q3d

Listing 4: Q3d_mu.m

```
%% Q3d

% Should compare no hedging, daily, weekly, monthly hedge effectiveness
clearvars
close all
```

```
rng('default') % reproducibility

% parameter initialization

M = 50000; % as suggested on Piazza, suggested 2000
sigma = 0.2;
r = 0.03;
mu = 0.15;
T = 1;
S0 = 10;
K = 0.95 * S0;
N = 250;
dt = T/N;

[V0,S, delta_bin] = binomialDeltaStraddle(S0,r,sigma, T,N,K);

%% no hedging

% Initial portfolio
S_MC = S0 * ones(M,1);
B = (V0-delta_bin(1,1)*S0)*ones(M,1);
delta_hedge_old = delta_bin(1,1)*ones(M,1);
delta_hedge_new = delta_hedge_old;

% at t_N
sample = randn(M,1);
% calculate S(t_{n+1}), see (8) in assignment 2
S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * T + sigma.* sample .* sqrt(
    T));
W = max(S_MC -K, 0) + max(K- S_MC, 0);
hedgingerror = -W+ delta_hedge_new .* S_MC + B.*exp(r*T);

PL_nohedging = exp(-r*T) * hedgingerror ./ V0;

PL_nohedging_mean = mean(PL_nohedging)

%% daily hedge
% Initial portfolio
S_MC = S0 * ones(M,1);
B = (V0-delta_bin(1,1)*S0)*ones(M,1);
delta_hedge_old = delta_bin(1,1)*ones(M,1);
delta_hedge_new = delta_hedge_old;
for n = 1:N-1
    % at time t_n
    % obtain phi_n for each path
```

```
    sample = randn(M,1);
    % calculate S(t_{n+1}), see (8) in assignment 2
    S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt + sigma.* sample .* sqrt(dt));
    delta_hedge_new = interpDelta(delta_bin(1:n+1, n+1), S(1:n+1, n+1), S_MC);
    B = B.*exp(r * dt) + (delta_hedge_old - delta_hedge_new).* S_MC;
    delta_hedge_old = delta_hedge_new;
end

% at t_N
sample = randn(M,1);
% calculate S(t_{n+1}), see (8) in assignment 2
S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt + sigma.* sample .* sqrt(dt));
W = max(S_MC -K, 0) + max(K- S_MC, 0);
hedgingerror = -W+ delta_hedge_new .* S_MC + B.*exp(r*dt);

PL_daily = exp(-r*T) * hedgingerror ./ V0;

% disp('daily hedge, mean PL')
PL_mean_daily = mean(PL_daily)

%% weekly hedge
% Initial portfolio
S_MC = S0 * ones(M,1);
B = (V0-delta_bin(1,1)*S0)*ones(M,1);
delta_hedge_old = delta_bin(1,1)*ones(M,1);
delta_hedge_new = delta_hedge_old;
dt_week = 5*dt;
for n = 5:5:N-5
    % at time t_n
    % obtain phi_n for each path
    sample = randn(M,1);
    % calculate S(t_{n+1}), see (8) in assignment 2
    S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt_week + sigma.* sample .* sqrt(dt_week));
    delta_hedge_new = interpDelta(delta_bin(1:n+1, n+1), S(1:n+1, n+1), S_MC);
    B = B.*exp(r * dt_week) + (delta_hedge_old - delta_hedge_new).* S_MC;
    delta_hedge_old = delta_hedge_new;
end
% after the for loop, we get t_{N-5} hedging position and bond value
```

```matlab

% at t_N
sample = randn(M,1);
% calculate S(t_{n+1}), see (8) in assignment 2
S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt_week + sigma.* sample .* sqrt(dt_week));
W = max(S_MC -K, 0) + max(K- S_MC, 0);
hedgingerror = -W+ delta_hedge_new .* S_MC + B.*exp(r*dt_week);

PL_weekly = exp(-r*T) * hedgingerror ./ V0;

PL_weekly_mean = mean(PL_weekly)

%% monthly hedge
% Initial portfolio
S_MC = S0 * ones(M,1);
B = (V0-delta_bin(1,1)*S0)*ones(M,1);
delta_hedge_old = delta_bin(1,1)*ones(M,1);
delta_hedge_new = delta_hedge_old;
dt_monthly = 20*dt;
for n = 20:20:N
    % at time t_n
    % obtain phi_n for each path
    sample = randn(M,1);
    % calculate S(t_{n+1}), see (8) in assignment 2
    S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt_monthly + sigma.* sample .* sqrt(dt_monthly));
    delta_hedge_new = interpDelta(delta_bin(1:n+1, n+1), S(1:n+1, n+1), S_MC);
    B = B.*exp(r * dt_monthly) + (delta_hedge_old - delta_hedge_new).* S_MC;
    delta_hedge_old = delta_hedge_new;

end
% note that last entry of 20:20:N is 240
% to liquidate the porfolio, we need to use dt_10 = 10*dt

% at t_N, N = 250
dt_10 = 10*dt;
sample = randn(M,1);
% calculate S(t_{n+1}), see (8) in assignment 2
S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt_10 + sigma.* sample .* sqrt(dt_10));
W = max(S_MC -K, 0) + max(K- S_MC, 0);
hedgingerror = -W+ delta_hedge_new .* S_MC + B.*exp(r*dt_10);

```

```
PL_monthly = exp(-r*T) * hedgingerror ./ V0;

PL_monthly_mean = mean(PL_monthly)

%% histogram
PL = [PL_nohedging, PL_daily, PL_weekly, PL_monthly];
xlabelslist =["no hedging P&L" "daily hedging P&L" "weekly hedging P&L" "monthly hedging P&L"];
% to have string array, need to use " instead of '

% histograms with 50 bins
for i = 1:4
    figure(1)
    subplot(2,2,i)
    binranges= linspace(min(PL(:,i)), max(PL(:,i)), 51); % use at least 50 bins
    bincounts = histc(PL(:,i), binranges);
    bar(binranges,bincounts,'histc')
    title('histogram for relative hedging error P&L')
    ylabel('number of occurences')
    xlabel(xlabelslist(i))
end

% histogram with normal dist fit, 50 bins
for i = 1:4
    figure(2)
    subplot(2,2,i)
    histfit(PL(:,i), 50)
    title('histogram for relative hedging error P&L, with normal dist fit')
    ylabel('number of occurences')
    xlabel(xlabelslist(i))
end


%% Comment on your observations
% We observe that for daily, weekly and monthly hedging, the P&L
% distribution is very close to normal distrbution. This is clearly
% reflected in our plots of histogram with normal distribution fit.
%
% Moreover, compared to daily, weekly and monthly hedgings, no hedging P&L
% does not behave like a normal distribution. It is more like a skewed
% distribution, but still with mean of P&L being roughly 0. No hedging P&L
```

```
% has occurrences of extremre losses while daily, weekly and monthly
% hedging do not have. This demostrates the superiority of hedging.
%
% Among daily, weekly and monthly hedgings, we see that daily hedging
% performs the best in the sense that it has the most occurence of nearly 0
% P&L. This can be seen by observing the number of occurences of the bin
% covering 0. Note that we have a perfect hedge when the P&L is 0. In
% particular, the number of occurences for bin covering 0 for daily hedging
% is around 7000, while that for monthly hedging is only around 5500.
%
% Note that P&L is of course not zero along each scenario path (as shown in
% histograms that we have occurence of nonzero P&L) since our rebalancing
% is not continuous so there is time discretization error. Also our
% computed delta is just an approximation of the real delta, such computed
% delta is obtained via binomial model where we rely on the option values
% as computed by the binomial model. But we know that option values from
% binomial model also have time discretization error. So this another
% source of error.
% (This part I am unsure)
%
```

```
PL_nohedging_mean =

  -0.1424


PL_mean_daily =

  -0.0014


PL_weekly_mean =
```

```
  -0.0035


PL_monthly_mean =

  -0.0120
```

3.5 Q3e

Listing 5: Q3e.m

```
%% Q3e

%% Data preparation
clearvars
close all

rng('default') % reproducibility

% parameter initialization

M = 50000; % as suggested on Piazza, suggested 2000
sigma = 0.2;
r = 0.03;
mu = 0.15;
T = 1;
S0 = 10;
K = 0.95 * S0;
N = 250;
dt = T/N;

[V0,S, delta_bin] = binomialDeltaStraddle(S0,r,sigma, T,N,K);

% no hedging
% Initial portfolio
S_MC = S0 * ones(M,1);
B = (V0-delta_bin(1,1)*S0)*ones(M,1);
delta_hedge_old = delta_bin(1,1)*ones(M,1);
delta_hedge_new = delta_hedge_old;

% at t_N
sample = randn(M,1);
% calculate S(t_{n+1}), see (8) in assignment 2
S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * T + sigma.* sample .* sqrt(T));
W = max(S_MC -K, 0) + max(K- S_MC, 0);
hedgingerror = -W+ delta_hedge_new .* S_MC + B.*exp(r*T);

PL_nohedging = exp(-r*T) * hedgingerror ./ V0;

% daily hedge
```

```
% Initial portfolio
S_MC = S0 * ones(M,1);
B = (V0-delta_bin(1,1)*S0)*ones(M,1);
delta_hedge_old = delta_bin(1,1)*ones(M,1);
delta_hedge_new = delta_hedge_old;
for n = 1:N-1
    % at time t_n
    % obtain phi_n for each path
    sample = randn(M,1);
    % calculate S(t_{n+1}), see (8) in assignment 2
    S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt + sigma.* sample .* sqrt(dt)
        );
    delta_hedge_new = interpDelta(delta_bin(1:n+1, n+1), S(1:n+1, n+1),
        S_MC);
    B = B.*exp(r * dt) + (delta_hedge_old - delta_hedge_new).* S_MC;
    delta_hedge_old = delta_hedge_new;
end

% at t_N
sample = randn(M,1);
% calculate S(t_{n+1}), see (8) in assignment 2
S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt + sigma.* sample .* sqrt(dt));
W = max(S_MC -K, 0) + max(K- S_MC, 0);
hedgingerror = -W+ delta_hedge_new .* S_MC + B.*exp(r*dt);

PL_daily = exp(-r*T) * hedgingerror ./ V0;

% weekly hedge
% Initial portfolio
S_MC = S0 * ones(M,1);
B = (V0-delta_bin(1,1)*S0)*ones(M,1);
delta_hedge_old = delta_bin(1,1)*ones(M,1);
delta_hedge_new = delta_hedge_old;
dt_week = 5*dt;
for n = 5:5:N-5
    % at time t_n
    % obtain phi_n for each path
    sample = randn(M,1);
    % calculate S(t_{n+1}), see (8) in assignment 2
    S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt_week + sigma.* sample .*
        sqrt(dt_week));
    delta_hedge_new = interpDelta(delta_bin(1:n+1, n+1), S(1:n+1, n+1),
        S_MC);
    B = B.*exp(r * dt_week) + (delta_hedge_old - delta_hedge_new).* S_MC;
    delta_hedge_old = delta_hedge_new;
end
% after the for loop, we get t_{N-5} hedging position and bond value

% at t_N
sample = randn(M,1);
```

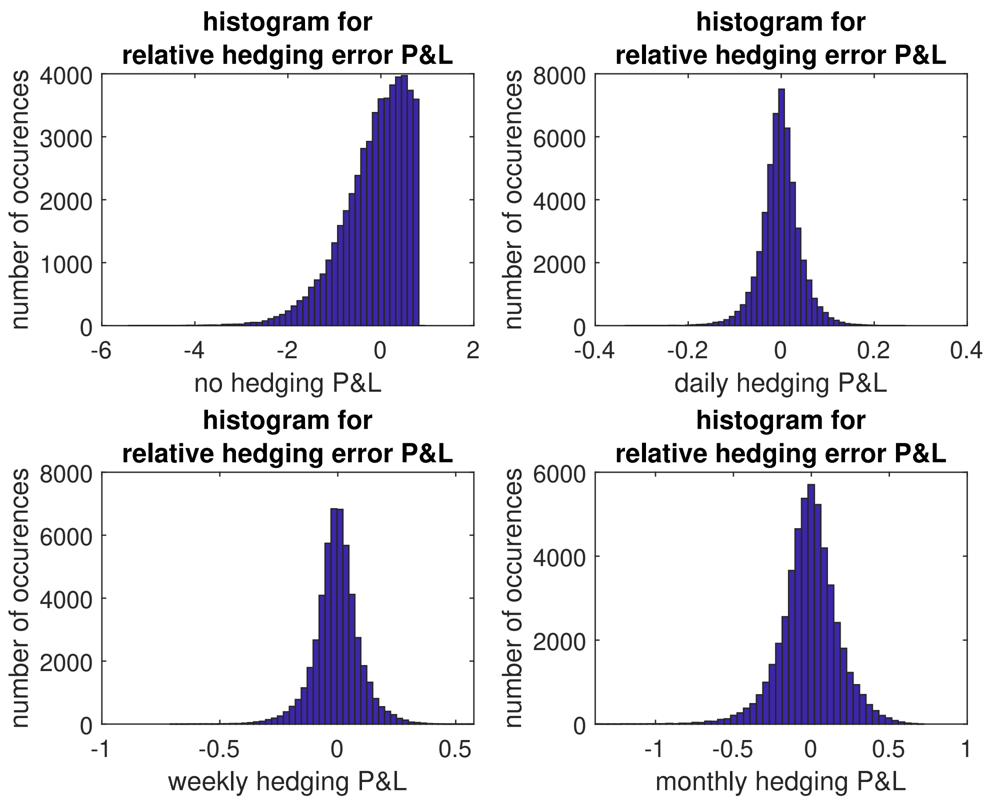
```
% calculate S(t_{n+1}), see (8) in assignment 2
S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt_week + sigma.* sample .* sqrt(dt_week));
W = max(S_MC -K, 0) + max(K- S_MC, 0);
hedgingerror = -W+ delta_hedge_new .* S_MC + B.*exp(r*dt_week);

PL_weekly = exp(-r*T) * hedgingerror ./ V0;

% monthly hedge
% Initial portfolio
S_MC = S0 * ones(M,1);
B = (V0-delta_bin(1,1)*S0)*ones(M,1);
delta_hedge_old = delta_bin(1,1)*ones(M,1);
delta_hedge_new = delta_hedge_old;
dt_monthly = 20*dt;
for n = 20:20:N
    % at time t_n
    % obtain phi_n for each path
    sample = randn(M,1);
    % calculate S(t_{n+1}), see (8) in assignment 2
    S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt_monthly + sigma.* sample .* sqrt(dt_monthly));
    delta_hedge_new = interpDelta(delta_bin(1:n+1, n+1), S(1:n+1, n+1), S_MC);
    B = B.*exp(r * dt_monthly) + (delta_hedge_old - delta_hedge_new).* S_MC;
    delta_hedge_old = delta_hedge_new;
end
% note that last entry of 20:20:N is 240
% to liquidate the porfolio, we need to use dt_10 = 10*dt

% at t_N, N = 250
dt_10 = 10*dt;
sample = randn(M,1);
% calculate S(t_{n+1}), see (8) in assignment 2
S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt_10 + sigma.* sample .* sqrt(dt_10));
W = max(S_MC -K, 0) + max(K- S_MC, 0);
hedgingerror = -W+ delta_hedge_new .* S_MC + B.*exp(r*dt_10);

PL_monthly = exp(-r*T) * hedgingerror ./ V0;

%% Data
PL = [PL_nohedging, PL_daily, PL_weekly, PL_monthly];

%% compute and report the performance measures of different rebalancing times
beta = 0.95;

PLperformance_table = zeros(4,4);
```

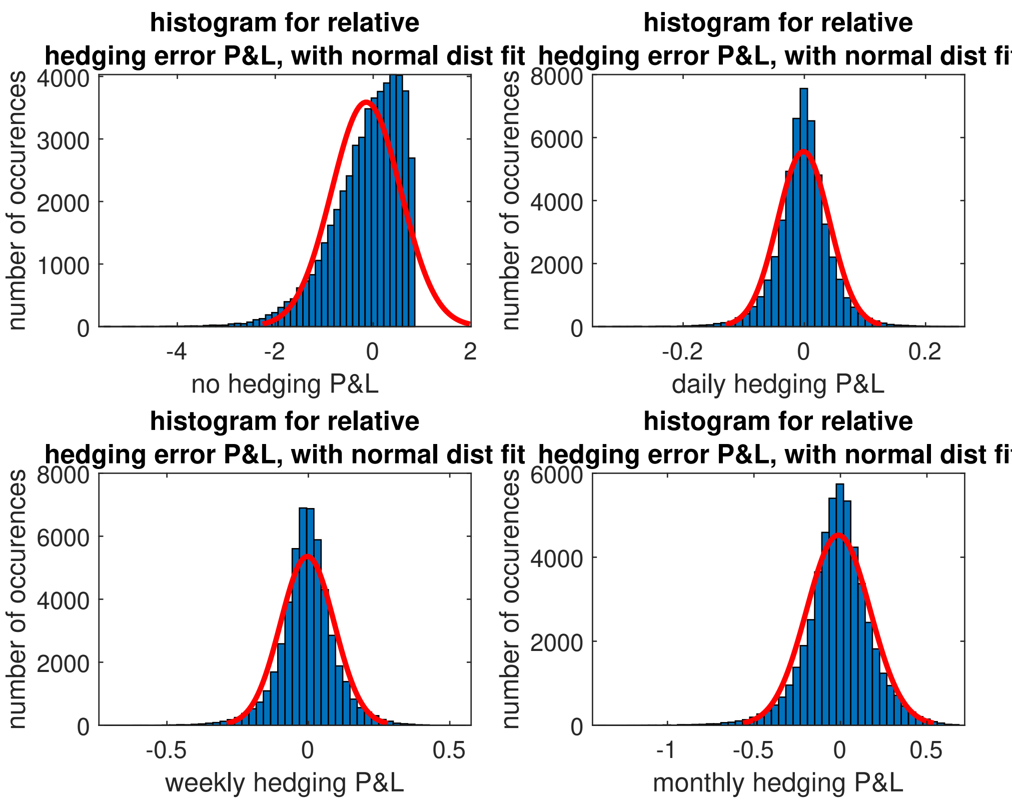
```matlab

for i = 1:4
    PLperformance_table(1,i) = mean(PL(:,i));
    PLperformance_table(2,i) = std(PL(:,i));
    [var,cvar] = dVaRCVaR(PL(:,i),beta);
    PLperformance_table(3,i) = var;
    PLperformance_table(4,i) = cvar;
end

PLperformance_table = array2table(PLperformance_table', 'VariableNames',...
    {'Mean','Standard deviation','VaR(95%)','CVaR(95%)'}, 'RowNames',...
    {'no hedging P&L', 'daily hedging P&L','weekly hedging P&L', 'monthly
        hedging P&L'})

%% Discussion
% Note that VaR describes the predicted minimum profit (maximum loss) with
% a specified probability conidence level over a certain period of time.
% That means that we want to have less VaR (in absolute value since our VaR
% is applied to P&L, not on loss). Similarly, CVaR is the is the average
% P&L, given P&L is less than VaR. In other words, CVaR si measuring the
% average amount of money lost given that we are in the worst (1—beta) = 5%
% scenarios. So we also want CVaR be small in absolute value.
%
% For mean, we want the mean be close to 0 since P&L being 0 means a
% perfect hedge. If the mean is close to 0, we want the standard deviation
% be small so that P&L along each path is close to 0.
%
% Rebalancing frequency: daily > weekly > monthly > no
%
% Based on the table above, we see that the means of no hedging, daily,
% weekly and monthly hedging are all close to 0, with the means of daily
% and weekly hedging particularly close to 0, being one order better than
% no hedging and monthly hedging. In fact the means (in absolute value) is
% a decreasing function of rebalancing frequency. We also see that the
% standard deviation is a decreasing function of rebalancing frequency.
% Similarly, VaR and CVaR (in absolute value) are decreasing functions of
% rebalancing frequency. Thus, the more frequent you rebalance, the better
% hedging preformance you get. In particular, among the rebalancing times
% we consider, daily hedging has the best performance.
```

```
PLperformance_table =

  4×4 table
```

	Mean	Standard deviation	VaR(95%)	CVaR(95%)
no hedging P&L	-0.14241	0.70586	-1.4808	-1.9854
daily hedging P&L	-0.0014149	0.042708	-0.070184	-0.10108

weekly hedging P&L	-0.0035004	0.094911	-0.15788	-0.22852
monthly hedging P&L	-0.012013	0.18374	-0.31453	-0.45074





histogram for relative
hedging error P&L, with normal dist fit
number of occurences
4000
3000
2000
1000
0
-4
-2
0
2
no hedging P&L
histogram for relative
hedging error P&L, with normal dist fit
number of occurences
8000
6000
4000
2000
0
-0.2
0
0.2
daily hedging P&L
histogram for relative
hedging error P&L, with normal dist fit
number of occurences
8000
6000
4000
2000
0
-0.5
0
0.5
weekly hedging P&L
histogram for relative
hedging error P&L, with normal dist fit
number of occurences
6000
4000
2000
0
-1
-0.5
0
0.5
monthly hedging P&L

3.6 Q3f

Let the discrete time of the lattice be denoted by $t_0 < t_1 < \ldots < t_{N-1} < t_N = T$. Suppose we are given a hedging rebalancing time t_n^{rb} that does not coincide with the discrete time t_n for any $0 \leq n \leq N$. Note that $t_n^{rb} \in (t_0, t_N)$. Then there is some t_n, (with $0 \leq n \leq N-1$) such that $t_n < t_n^{rb} < t_{n+1}$. Let $a = t_n^{rb} - t_n, b = t_{n+1} - t_n^{rb}$ and recall that $\Delta t = t_{n+1} - t_n$. In order to rebalance the hedge at t_n^{rb}, we can do (time) interpolation in the following way. Note that we consider only one single path. Of course this can then be applied to many paths. When we are doing hedging analysis, suppose we have already generated S_n at t_n by geometric Brownian motion based on real market price, and we have yet generated S_{n+1} at t_{n+1}. At t_n we have δ_n, B_n, the hedging position (i.e. units of underlying we hold) at t_n and the bond value (cash) we have at t_n. Then we can generate underlying price S_{rb} at t_n^{rb} by

$$S_{rb} = S_n \cdot e^{\left(\mu - \frac{1}{2}\sigma^2\right)a + \sigma\phi_1\sqrt{a}},$$

where $\phi_1 \sim \mathcal{N}(0,1)$ is a random variable that is independent of all random variables we have already used. Now we want to obtain the hedging position δ_{rb} at time t_n^{rb}. If we obtained δ_{rb}, the bond value B_{rb} at t_n^{rb} can be calculated using δ_{rb}, S_{rb} by

$$B_{rb} = B_n e^{ra} + (\delta_n - \delta_{rb})\, S_{rb}.$$

We obtain δ_{rb} by interpolating via time. That is we obtain

$$\delta'_{rb} = \textbf{interpDelta}(\delta^n, S^n, S_{rb})$$

$$\delta''_{rb} = \textbf{interpDelta}(\delta^{n+1}, S^{n+1}, S_{rb})$$

where δ^n, S^n are vectors of deltas and underlying prices at time t_n and δ^{n+1}, S^{n+1} are that at time t_{n+1}. Then we obtain δ_{rb} by

$$\delta_{rb} = \frac{b}{a+b}\delta'_{rb} + \frac{a}{a+b}\delta''_{rb}$$

This describes how we rebalance at t_n^{rb}.

And we can then generate S_{n+1} at t_{n+1} by

$$S_{n+1} = S_{rb} \cdot e^{\left(\mu - \frac{1}{2}\sigma^2\right)b + \sigma\phi_2\sqrt{b}},$$

where $\phi_2 \sim \mathcal{N}(0,1)$ is a random variable that is independent of all random variables we have already used. And obtain δ_{n+1} by the usual way (using $\textbf{interpDelta}(\delta^{n+1}, S^{n+1}, S_{n+1})$) and obtain B_{n+1} by

$$B_{n+1} = B_{rb}e^{rb} + (\delta_{rb} - \delta_{n+1})S_{n+1}.$$

This describes how we rebalance at t_{n+1}.

Any time that is before t_n^{rb} or after t_{n+1} is just the same procedure as employed in part d.

4 Q4

4.1 Q4a

For $0 < t < T$,

$$\delta_t = \begin{cases} 0, & \text{if } S_t \geq K \\ -1, & \text{if } S_t < K \end{cases}$$

Note that for completeness, I change the condition $S_t > K$ to $S_t \geq K$. But in fact $S_t = K$ is a measure 0 event. However, in Matlab implementation I am not sure whether it is really impossible to have $S_t = K$ so I will use $S_t \geq K$ at least for implementation.

4.2 Q4b

The expressions in terms of S_t, K are:

$$\delta_t S_t + B_t = \begin{cases} 0S_t + B_t = 0 + B_0 - K & \text{if } S_t \geq K \\ -1 \cdot S_t + B_t = -1 \cdot S_t + B_0 & \text{if } S_t < K, \end{cases}$$

where $B_0 = P_0 + S_0$ since we set the initial portfolio to have value 0.

Note that if the S_t goes from $S_t < K$ to $S_t \geq K$, we then buy a stock share. Note that initially $S_0 < K$ so that after the first time of such underlying price exceeding strike price, our δ_t changes from -1 to 0.

Now, if S_t goes from $S_t \geq K$ to $S_t < K$, assuming $\delta_t = 0$ before S_t goes below K, then since we sell one stock, we then have $\delta_t = -1$.

We see the pattern here, assuming $S_t < K$ and we are holding $\delta_t = -1$, then when $S_t \geq K$, we will have $\delta_t = 0$. Similarly, assuming $S_t \geq K$ and $\delta_t = 0$, then when $S_t < K$, we will have $\delta_t = -1$. This justifies the expressions in part a.

Now, return to $\delta_t + B_t$. Note that $r = 0$ so we don't have to worry about interest.

When $S_t \geq K$, since we assumed continuous hedging, so the time we bought one share will be exactly when $S_t = K$, and this will result in K dollars deduction in cash account so that $B_t = B_0 - K$, assuming that the cash account right before $S_t \geq K$ happens is just B_0.

When $S_t < K$, since we assumed continuous hedging, so the time we sell one share will be exactly when $S_t = K$, and this will result in K dollars addition in cash account so that $B_t = B_0 - K + K = B_0$, assuming that the cash account right before $S_t < K$ happens is $B_0 - K$.

These two paragraphs described what will happen for the portfolio since at initial time we do have $S_0 < K$ and $B_0 = B_0$.

4.3 Q4c

When $S_T \geq K$,

$$\begin{aligned} P\&L &= \frac{-P_T + \delta_T S_T + B_T}{P_0 + S_0} \\ &= \frac{0 + 0S_T + B_0 - K}{P_0 + S_0} \\ &= \frac{S_0 + P_0 - K}{P_0 + S_0} > 0, \end{aligned}$$

where $S_0 + P_0 - K > 0$ can be proved by an no-arbitrage argument.

When $S_T < K$,

$$\begin{aligned} P\&L &= \frac{-P_T + \delta_T S_T + B_T}{P_0 + S_0} \\ &= \frac{-(K - S_T) + -S_T + B_0}{P_0 + S_0} \\ &= \frac{B_0 - K}{P_0 + S_0} \\ &= \frac{S_0 + P_0 - K}{P_0 + S_0} > 0, \end{aligned}$$

Thus, since $-P_0 + \delta_0 S_0 + B_0 = 0$ but $-P_T + \delta_T S_T + B_T > 0$, we have an arbitrage.

4.4 Q4d and e

Listing 6: Q4d_and_e.m

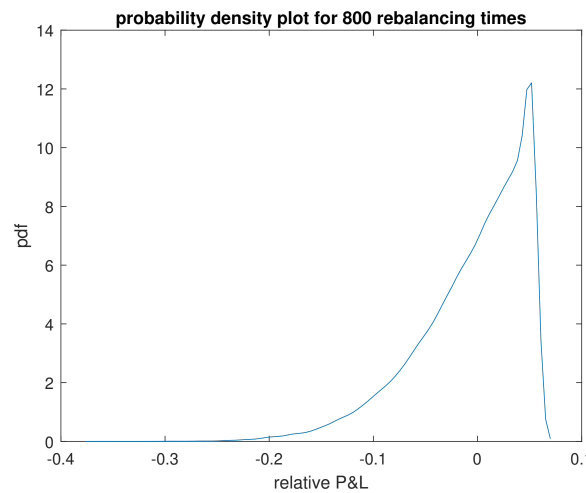
```
%% Q4d_and_e

close all
clearvars

rng('default')

% parameter initialization

M = 80000;
sigma = 0.2;
r = 0;
mu = 0.15;
T = 1;
S0 = 100;
K = 105;
Nlist = [100,200,400,800];

% [V0,S, delta_bin] = binomialDeltaStraddle(S0,r,sigma, T,Nlist(end),K);

[exactcall, exactput] = blsprice(S0, K, r, T, sigma)
PL = zeros(M,4);
for i = 1:length(Nlist)
    N = Nlist(i);
    % disp(N)
    % initial positions
    dt = T/N;
    % disp(dt)
    S_MC = S0 * ones(M,1);
    B = (exactput + S0)*ones(M,1);
    delta_hedge_old = -ones(M,1);
    delta_hedge_new = delta_hedge_old;
```

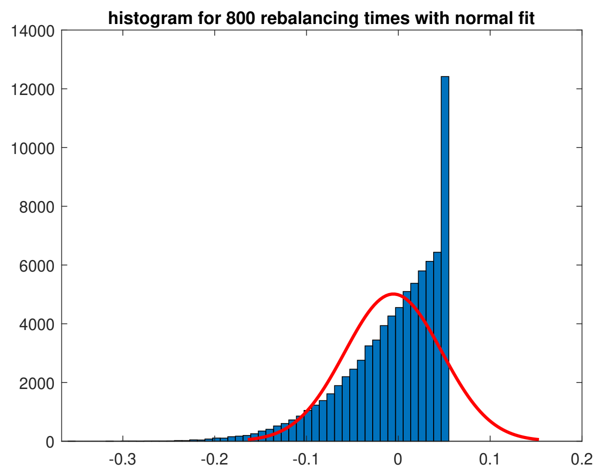
```
    % it seems like we also rebalance at the end t_N so we use 1:N instead of
    % 1:N-1
    for n = 1:N
        % logical values from t_{n-1}
        idx1 = S_MC < K; % delta_t = 1 if S_t >= K, although equality should not happen
        idx2 = ~idx1; % S_MC >= K

        % at time t_n
        % obtain phi_n for each path
        sample = randn(M,1);

        % calculate S(t_{n+1}), see (8) in assignment 2, using mu
        S_MC = S_MC.*exp((mu - 1/2 * sigma^2) * dt + sigma.* sample .* sqrt(dt));

        % update delta only when (S_n < K and S_{n+1} >= K) or (S_n >= K and
        % S_{n+1} < K), the former case update delta to 0, while the latter
        % update it to -1

        idx3 = idx1 & (S_MC >= K);
        idx4 = idx2 & (S_MC < K);

        delta_hedge_new(idx3) = 0; % 0 or 1???
        delta_hedge_new(idx4) = -1;

        B = B.*exp(r * dt) + (delta_hedge_old - delta_hedge_new).* S_MC;
        delta_hedge_old = delta_hedge_new;
    end
    W = max(K- S_MC,0);

    PL(:,i) = (-W + delta_hedge_new .* S_MC + B)./ (S0 + exactput);
end

beta = 0.95;

PLperformance_table = zeros(4,4);

for i = 1:4
    PLperformance_table(1,i) = mean(PL(:,i));
    PLperformance_table(2,i) = std(PL(:,i));
    [var,cvar] = dVaRCVaR(PL(:,i),beta);
    PLperformance_table(3,i) = var;
    PLperformance_table(4,i) = cvar;
end

PLperformance_table = array2table(PLperformance_table, 'RowNames',...
```

```
    {'Mean','Standard deviation','VaR(95%)','CVaR(95%)'}, 'VariableNames', ...
    {'100', '200','400', '800'})

% probability density plot
h = figure(1);
[f,xi] = ksdensity(PL(:,4));
plot(xi,f)
title('probability density plot for 800 rebalancing times')
xlabel('relative P&L')
ylabel('pdf')
saveas(h,'Q4fig1','epsc')

% histogram with normal fit
g = figure(2);
histfit(PL(:,4), 50)
title('histogram for 800 rebalancing times with normal fit')
saveas(g,'Q4fig2','epsc')

%% Discussion
% What do you observe about the mean and variance of the hedging error?
% I observe that the mean is actually less than 0, although being quite
% close to 0 (around -0.0005). The standard deviation is around 0.053,
% which is quite low. We see that for there are no substantial difference
% bewteen different rebalancing times. They all have similar (at least same
% order of magnitude) mean, standard deviation, VaR and CVaR.

%% Discussion, part e
% I have also tried to use some very high rebalancing times like 30000. But
% the mean of P&L is still around -0.0052306, which is roughly the same for
% rebalancing times 100,200,400,800. It seems to me that for each
% transcation, we incur some small losses since we are selling the stock
% with price < K and buying the stock with price > K, owing to the time
% discretization error so that we cannot exactly buy and sell the stock at
% price K. If we refine our timestepping, i.e. taking larger N, then the
% loss for each transaction is smaller but we also have to do more
% transactions. So that the total loss stays around the same. That might
% possibly explains why even we use a much higher rebalancing time, there
% is still no substantial improvment. This discussion also addresses part
% e.
```

```
exactcall =

    5.9056


exactput =

   10.9056
```

```
PLperformance_table =

  4×4 table

                             100           200            400            800
                          ----------    ---------    ----------     ----------

    Mean                  -0.0059175    -0.005792    -0.0057793     -0.0054221
    Standard deviation      0.053201     0.053139      0.053258       0.052841
    VaR(95%)                -0.10988     -0.10996      -0.11073       -0.10964
    CVaR(95%)               -0.14517     -0.14542      -0.14593       -0.14535
```





5 Q5

5.1 Q5a

$$
\begin{aligned}
\kappa = \mathbb{E}[J-1] &= \int_{-\infty}^{\infty} (e^y - 1) f(y) dy \\
&= \int_{-\infty}^{\infty} (e^y - 1) p_u \mu_u e^{-\mu_u y} 1_{y \geq 0} + (e^y - 1)(1 - p_u)\mu_d e^{\mu_d y} 1_{y<0} \\
&= p_u \mu_u \int_0^{\infty} e^{(-\mu_u + 1)y} - e^{-\mu_u y} dy + (1 - p_u)\mu_d \int_{-\infty}^{0} e^{(\mu_d+1)y} - e^{\mu_d y} dy \\
&= p_u \mu_u \left(\frac{1}{-\mu_u + 1} e^{(-\mu_u+1)y} + \frac{e^{-\mu_u y}}{\mu_u}) \right|_0^{\infty} + (1 - p_u)\mu_u \left(\frac{1}{\mu_d + 1} e^{(\mu_d+1)y} - \frac{e^{\mu_d y}}{\mu_d}) \right|_0^{\infty} \\
&= -p_u \mu_u \left(\frac{1}{-\mu_u + 1} + \frac{1}{\mu_u} \right) + (1 - p_u)\mu_d \left(\frac{1}{\mu_d + 1} - \frac{1}{\mu_d} \right) \\
&= \frac{p_u \mu_u}{\mu_u - 1} + \frac{(1 - p_u)\, \mu_d}{\mu_d + 1} - p_u - (1 - p_u) \\
&= \frac{p_u \mu_u}{\mu_u - 1} + \frac{(1 - p_u)\, \mu_d}{\mu_d + 1} - 1,
\end{aligned}
$$

where we use the fact that $\mu_u > 1$ and so $-\mu_u + 1 < 0$, and $\mu_d > 0$ in the fifth equality.

5.2 Q5b

Listing 7: Q5b.m

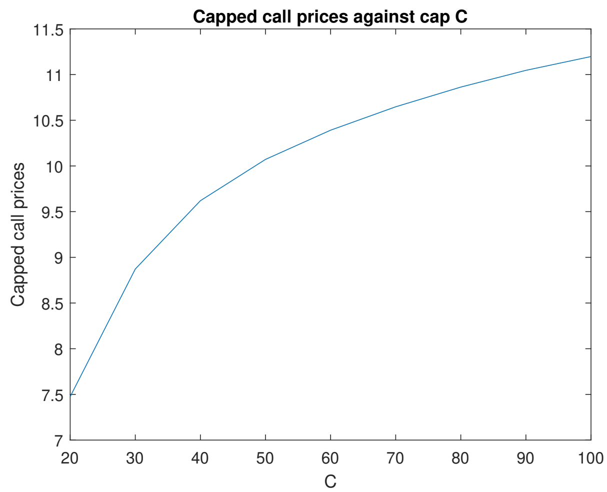
```
%% Q5b
% Note that we haven't used CappedCall function this time. Instead we
% basically implement the function in this file and compute the fair values
% of the capped call. The way we used here will be slightly faster since we
% only simulate once and the resulting paths will be used across all values
% of cap C. Of course, we could do it by calling CappedCall for each C.

% Parameters
close all

rng('default')

sigma = 0.15;
r = 0.05; % we use r, but should we consider it as compensated? I think so.
T = 1;
K = 95;
S0 = 95;
mu_u = 3.04;
mu_d = 3.08;
pu = 0.34;
lambda = 0.1;
% dt = 1/1000;
% use 1/1000 or N = 800?
```

```

N = 800;
dt = T/ N;
M = 25000;
Clist = 20:10:100;

% compensated drift E[J-1]
kappa = pu* mu_u / (mu_u - 1) + (1-pu) *mu_d /(mu_d +1) -1;
% compensated drift for X = log(S), risk neutral
drift = (r - sigma^2/ 2 - lambda* kappa);

% X = log(S)
X_old = log(S0) * ones(M,1 );
X_new = zeros(M,1);

jump_check = zeros(M,1);
jump_size = zeros(M,1);
jump_mask = zeros(M,1);
jump_up = zeros(M,1);
jump_down = zeros(M,1);

% apply Euler timestepping on X = log(S)
for i = 1:N %timestep loop
    % uniform distribution
    jump_check = rand(M,1);
    jump_mask = jump_check <= lambda *dt; % index for existence of jumping
    % resample, now for determining up or down jump
    jump_check = rand(M,1);
    jump_up = jump_check <= pu; % storing indices for up jump
    jump_down = ~jump_up;
    jump_size(jump_up) = exprnd(1/mu_u, sum(jump_up),1);
    jump_size(jump_down) = exprnd(1/mu_d, sum(jump_down),1);

    jump_size = jump_size .* jump_mask;

    X_new = X_old + drift* dt + sigma *sqrt(dt) * randn(M,1) + ...
        jump_size;
    X_old = X_new;
end

S = exp(X_new); % asset price values for each path

% do we have to generate different samples for different C? I think we
% don't. But of course we can easily do it.

% obtain MC capped call option values
V = zeros(length(Clist),1);
for i = 1:length(Clist)
    C = Clist(i);
    V(i,1) = mean(exp(-r*T) * min(max(S-K,0),C));
```

```matlab
end

V_mat = V;
V = array2table(V, 'RowNames', "C = " + string(Clist))

g = figure(1);
plot(Clist, V_mat)
title('Capped call prices against cap C')
xlabel('C')
ylabel('Capped call prices')
saveas(g, 'q5b','epsc')

% V_fun = CappedCall(S0, r, sigma, pu,mu_u,mu_d, lambda, K,T,Clist(1),M,N)
%
% for i = 1 : length(Clist)
%     V_fun_list(i) = CappedCall(S0, r, sigma, pu,mu_u,mu_d, lambda, K,T,Clist(i),M,N);
% end
%
% V_fun_list

%% Discussion
% How does the computed option value depend on the cap C ? Explain why your
% observation is reasonable.
%
% We see that the computed option value increases as the cap C increases.
% This is reasonable since the cap C is limiting the highest possible final
% payoff we can get from the capped call. If S_T is very large, let say S_T
% - K = 10000, then a capped call with C = 20 will only have final payoff
% 20 while a capped call with C = 100 will have final payoff 100 instead.
% In other words, a higher cap will allow us to earn more on some extreme
% in the money asset path so that the resulting option value is higher.
%
% Also note that the speed of increasing for option value as C increases is
% actually decreasing. This is reasonable since as the cap C increases, the
% number of asset price path that can exceed such cap becomes rarer (in
% fact very rare, the number of asset price path exceeding certain C,
% denoted by g(C), perhaps of order of an inverse of high order polynomial
% in C, or even inverse of an exponential function in C. I think this can
% be investigated further by examining brownian motion properties).
```

```
V =

  9×1 table

                V
              ------

    C = 20    7.4759
```

```
C = 30     8.8721
C = 40     9.6198
C = 50     10.072
C = 60     10.391
C = 70     10.648
C = 80     10.863
C = 90     11.046
C = 100    11.197
```



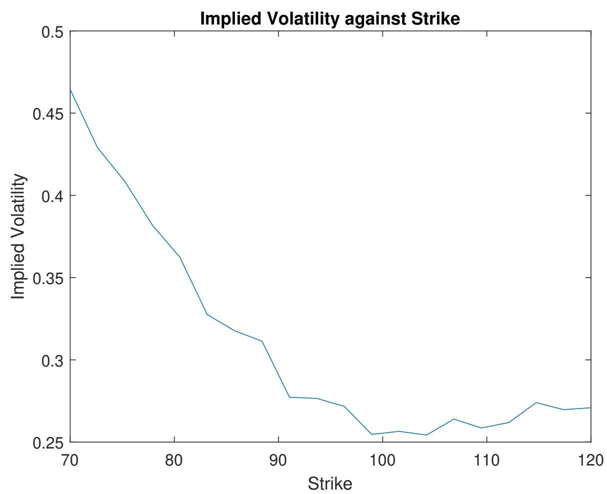
5.3 Q5c and d

Listing 8: Q5c_and_d.m

```
%% Q5c and d
clearvars
close all
% Parameters
rng('default')

C = Inf;
sigma = 0.15;
r = 0.05; % we use r, but should we consider it as compensated?
T = 1;
Klist = linspace(70,120,20);
S0 = 95;
mu_u = 3.04;
mu_d = 3.08;
pu = 0.34;
lambda = 0.1;
N = 800;
M = 25000;

V = zeros(length(Klist), 1);

for i = 1:length(V)
    K = Klist(i);
    V(i,1) = CappedCall(S0, r, sigma, pu,mu_u,mu_d, lambda, K,T,C,M,N);
end

V

IV = blsimpv(S0, Klist', r, T, V);
% note that default class for blsimpv is call

g = figure(1);
plot(Klist, IV')
title('Implied Volatility against Strike')
xlabel('Strike')
ylabel('Implied Volatility')
saveas(g, 'q5c','epsc')


%% Q5d
% We see a volatility skew, that is implied volatility decreases as the
% strike increases. We know that for equity options, the implied volatility
% against strike is often downward sloping. So this means that the assumed
% jump model does model such observed real life phenomenon. This may
% suggest that it is a good model to model the market underlying movement.
% Note that if we just use the geometric brownian motion with no jumps, we
```

```
% are just obtaining approximation of Black—Scholes price and the implied
% volatility plot will be close to a constant.
%
% There are also some oscillations when strike is large but I think it is
% due to the numerical approximation error (sampling error and time
% discretization error).
%
% Note also that the calculated implied volatilities are all larger than
% 0.25, which is again large than simga = 0.15. This means that our jump
% model "adds" more volatility (in BS sense) via allowing the underlying
% price to have jumps. This is also expected.
```

```
V =

   33.0580
   30.4673
   28.2215
   25.8250
   23.6206
   20.9767
   19.0789
   17.3534
   14.7537
   13.3346
   11.8622
   10.0269
    8.9879
    7.8927
    7.3412
    6.3101
    5.6838
    5.4365
    4.6737
    4.1623
```


Implied Volatility against Strike
Implied Volatility
Strike
0.5
0.45
0.4
0.35
0.3
0.25
70
80
90
100
110
120

6 Q6

Listing 9: Q6.m

```
%% Q6a
clearvars
close all

load('RawData.mat')

% change them all to column vectors
CSTest = CSTest';
CSTrain = CSTrain';
CVTest = CVTest';
CVTrain = CVTrain';

%% hedging error for training set
% assume the data is sorted with ascending time

dailyhedgingerror_training = CVTrain - DeltaTrain.* CSTrain;

g(1) = figure(1);
histogram(dailyhedgingerror_training,50)
title('histogram for daily hedging error, BS, training')

resulttable= zeros(1,4);
resulttable(1) = mean(dailyhedgingerror_training);
resulttable(2) = std(dailyhedgingerror_training);
beta = 0.95;
[var,cvar] = dVaRCVaR(dailyhedgingerror_training, beta);
resulttable(3) = var;
resulttable(4) = cvar;
resulttable_training_BS = array2table(resulttable, 'VariableNames',{'mean', ...
    'standard deviation', '95% VaR','95% CVaR'});

%% hedging error for testing set
% assume the data is sorted with ascending time

dailyhedgingerror_testing = CVTest - DeltaTest.* CSTest;

g(2) = figure(2);
histogram(dailyhedgingerror_testing,50)
title('histogram for daily hedging error, BS, testing')

resulttable= zeros(1,4);
resulttable(1) = mean(dailyhedgingerror_testing);
resulttable(2) = std(dailyhedgingerror_testing);
beta = 0.95;
[var,cvar] = dVaRCVaR(dailyhedgingerror_testing, beta);
```

```
resulttable(3) = var;
resulttable(4) = cvar;
resulttable_testing_BS = array2table(resulttable, 'VariableNames',{'mean', ...
    'standard deviation', '95% VaR','95% CVaR'});

%% Q6b 1
% we use OLS linear regression tot learn the parameters a,b,c
% Note that we use (4) \Delta f = \delta_{MV} \Delta S + \epsilon, in the
% paper and make use of (7) in the assignment.
% After some algebraic manipulation, we obtain the following form for
% linear regression
% \Delta f - \delta_{BS} \Delta S = \Delta S Vega/(S\sqrt{T}) a +
% \Delta S Vega \delta_{BS} /(S\sqrt{T}) b
% + \Delta S Vega \delta_{BS}^2 /(S\sqrt{T}) c + \epsilon
%
% Note that \Delta f - \delta_{BS}\Delta S is the BS hedging error
%
% Use the training data
inter = CSTrain.* VegaTrain./(STrain.* sqrt(TauTrain));
X = [ inter, inter.* DeltaTrain, inter.*DeltaTrain.*DeltaTrain];
mdl = fitlm(X, dailyhedgingerror_training,'Intercept', false)
a = mdl.Coefficients{1,1};
b = mdl.Coefficients{2,1};
c = mdl.Coefficients{3,1};

delta_MV_training = DeltaTrain + (VegaTrain./(STrain.* sqrt(TauTrain))).* ...
    (a + b * DeltaTrain + c * DeltaTrain.^2);

% alternatively, can use anova function to get sum of squared error
anova(mdl,'summary');

SSE_MV_training = sum((CVTrain- CSTrain .* delta_MV_training).^2);

Gain_training = 1 - SSE_MV_training ./ ( sum(dailyhedgingerror_training.^2) );

delta_MV_testing = DeltaTest + (VegaTest./(STest.* sqrt(TauTest))).*...
    (a + b * DeltaTest + c * DeltaTest.^2);

SSE_MV_testing = sum((CVTest- CSTest .* delta_MV_testing).^2);

Gain_testing = 1 - SSE_MV_testing ./ ( sum(dailyhedgingerror_testing.^2));

%% Q6b 2
hedgeerrorMV_training = CVTrain- CSTrain .* delta_MV_training;
g(3) = figure(3);
histogram(hedgeerrorMV_training,50)
title('histogram for daily hedging error, MV, training')
```

```

resulttable= zeros(1,4);
resulttable(1) = mean(hedgeerrorMV_training);
resulttable(2) = std(hedgeerrorMV_training);
beta = 0.95;
[var,cvar] = dVaRCVaR(hedgeerrorMV_training, beta);
resulttable(3) = var;
resulttable(4) = cvar;
resulttable_training_MV = array2table(resulttable, 'VariableNames',{'mean',
    ...
    'standard deviation', '95% VaR','95% CVaR'});


hedgeerrorMV_testing = CVTest- CSTest .* delta_MV_testing;
g(4) = figure(4);
histogram(hedgeerrorMV_testing,50)
title('histogram for daily hedging error, MV, testing')

resulttable= zeros(1,4);
resulttable(1) = mean(hedgeerrorMV_testing);
resulttable(2) = std(hedgeerrorMV_testing);
beta = 0.95;
[var,cvar] = dVaRCVaR(hedgeerrorMV_testing, beta);
resulttable(3) = var;
resulttable(4) = cvar;
resulttable_testing_MV = array2table(resulttable, 'VariableNames',{'mean',
    ...
    'standard deviation', '95% VaR','95% CVaR'});

% for easy comparsion
resulttable_training_BS;
resulttable_testing_BS;

%% Discussion Q6b 2
% We compare MV delta on training set against BS delta on training set and
% compare MV delta on testing set against BS delta on testing set. From the
% histograms, we see that the histograms for MV delta are more concentrated
% around 0 since the x-axis range of it is smaller. Moreover, by the table
% (mean, standard deviation, VaR, CVaR), we see that MV delta does give
% smaller standard deviation. This can be seen by comparing
% resulttable_training_BS vs resulttable_training_MV and comparing
% resulttable_testing_BS vs resulttable_testing_MV. We also see that by
% employing MV, we have improvments in VaR and CVaR, both in training and
% testing sets, compared to using BS delta.
%
% Now we compare MV delta on training set against MV delta on testing set.
% We see that the mean on training set is larger than that on testing set.
% Such phenomenon also exists on BS delta case. But in our case the
% difference between the mean is actually lower than that in BS delta case.
% More importantly, we observe that the standard deviation, VaR, CVaR (in
```

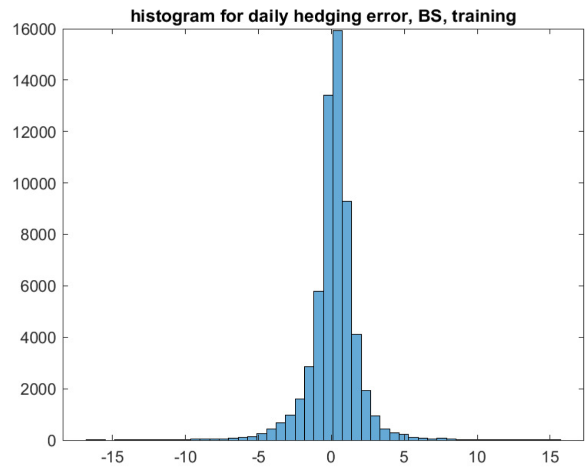
```
% absolute value) for training set is lower than that of testing set. That
% means hedging performance on training set is better. This makes sense
% since the coefficients a,b,c are fitted using training set so we expect
% better behaviour on training set. Keep in mind that we do have improvment
% on testing case, as compared to using BS delta, as reflected by
% Gain_testing being around 0.34.

%% Q6c

% design matrix
Xnew = [ones(length(STrain),1), STrain, DeltaTrain, DeltaTrain.^2, VegaTrain .* DeltaTrain, ...,
    VegaTrain.^2, DeltaTrain.^3].* CSTrain;
mdl_c = fitlm(Xnew, CVTrain, 'Intercept',false)

test_coeff=  regress(CVTrain, Xnew);

% Note that the p-values of c4 and c6 are pretty big, that means they might
% not be significant variables.

c0 = mdl_c.Coefficients{1,1};
c1 = mdl_c.Coefficients{2,1};
c2 = mdl_c.Coefficients{3,1};
c3 = mdl_c.Coefficients{4,1};
c4 = mdl_c.Coefficients{5,1};
c5 = mdl_c.Coefficients{6,1};
c6 = mdl_c.Coefficients{7,1};

% b 1

delta_c_train = c0 + c1* STrain + c2*DeltaTrain + c3 * DeltaTrain.^2 + ...
    c4 * VegaTrain .* DeltaTrain + c5 * VegaTrain.^2 + c6* DeltaTrain.^3;

SSV_c_train = sum((CVTrain- CSTrain .* delta_c_train).^2);

Gain_c_train = 1 - SSV_c_train ./ ( sum(dailyhedgingerror_training.^2));

delta_c_test = c0 + c1* STest + c2*DeltaTest + c3 * DeltaTest.^2 + ...
    c4 * VegaTest .* DeltaTest + c5 * VegaTest.^2 + c6* DeltaTest.^3;

SSE_c_testing = sum((CVTest- CSTest .* delta_c_test).^2);

Gain_c_test = 1 - SSE_c_testing ./ ( sum(dailyhedgingerror_testing.^2));

% b 2

hedgeerror_c_train = CVTrain- CSTrain .* delta_c_train;
g(5) = figure(5);
histogram(hedgeerror_c_train,50)
title('histogram for daily hedging error, part c, training')
```

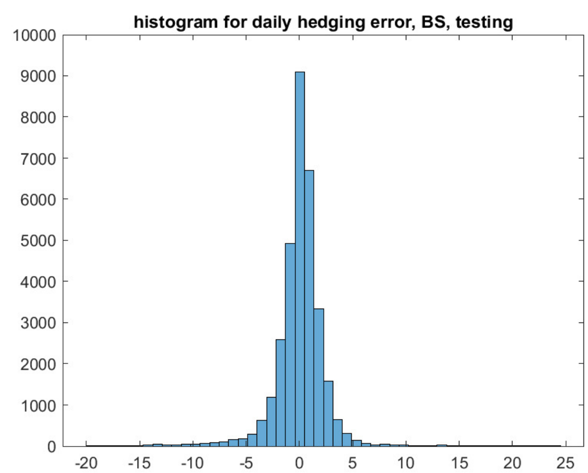
```

resulttable= zeros(1,4);
resulttable(1) = mean(hedgeerror_c_train);
resulttable(2) = std(hedgeerror_c_train);
beta = 0.95;
[var,cvar] = dVaRCVaR(hedgeerror_c_train, beta);
resulttable(3) = var;
resulttable(4) = cvar;
resulttable_train_c = array2table(resulttable, 'VariableNames',{'mean',...
    'standard deviation', '95% VaR','95% CVaR'});

hedgeerror_c_test = CVTest- CSTest .* delta_c_test;
g(6) = figure(6);
histogram(hedgeerror_c_test,50)
title('histogram for daily hedging error, part c, testing')

resulttable= zeros(1,4);
resulttable(1) = mean(hedgeerror_c_test);
resulttable(2) = std(hedgeerror_c_test);
beta = 0.95;
[var,cvar] = dVaRCVaR(hedgeerror_c_test, beta);
resulttable(3) = var;
resulttable(4) = cvar;
resulttable_test_c = array2table(resulttable, 'VariableNames',{'mean',...
    'standard deviation', '95% VaR','95% CVaR'});

%% Result
% we present all the results here cleanly
result = [resulttable_training_BS;...
    resulttable_testing_BS; resulttable_training_MV;...
    resulttable_testing_MV;resulttable_train_c; resulttable_test_c];
result.Properties.RowNames = {'BS,train','BS,test','MV,train','MS,test',...
    'part c,train','part c,test'};
result

Gain = zeros(1,4);
Gain(1) = Gain_training;
Gain(2) = Gain_testing;
Gain(3) = Gain_c_train;
Gain(4) = Gain_c_test;

Gain = array2table(Gain, 'VariableNames',{'MV,train','MV,test','part c, train',...
    'part c,test'})

for i =1:length(g)
    saveas(g(i),strcat('fig_Q6_',string(i)),'epsc')
end

%% Q6c Discussion
```

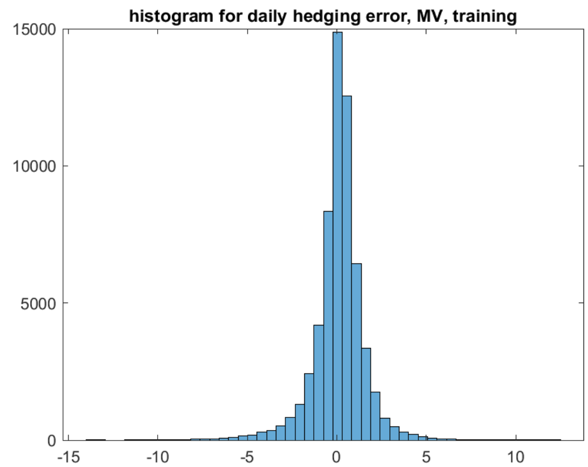
```
% Note that generally part c parametric form behaves quite well and achieve
% similar improvements as in MV delta. It is because the parametric form
% here is quite powerful, we included a lot of predictors and allow up to
% cubic delta. In a regression point of view, more predictors will give
% better training loss, while susceptible to worse generalization loss,
% especically in the case of adding irrelevant predictors. Observing the
% p-values of the predictors in part c, the p-value of square of vega is
% quite large meaning that it might not be relevant. In fact, the added
% complexity in this part c parametric form results into behaving better on
% training set while behaving worse on testing set, as compared to MV
% delta. This can be seen by observing the standard deviation and the Gain.
% Note that Gain on testing for part c is lower than that from MV delta.
% Note that Gain on training for part c is higher than that from MV delta.
% This observation is reasonable since our complex model will fit well on
% training set (overfitting) but not necessarily fit well on testing set.
% One reason for delta MV behaves better on the testing set is that it is
% supported by empirical observations of S&P 500 options on the
% relationship of delta_MV and delta_BS. Note in particular that there is
% no inverse of (S sqrt (T)) term in part c. That lack of domain knowledge
% might explain the worse behavior of part c on testing set and in
% generalization.
%
% But after all, generally the parametric form of part c captures quite a
% lot information on MV delta so that the resulting standard deviation and
% sum of square error than just using BS delta.
```

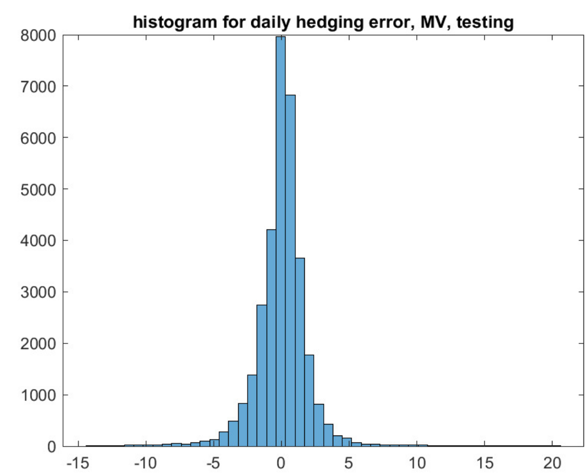
```
mdl =


Linear regression model:
    y ~ x1 + x2 + x3

Estimated Coefficients:
          Estimate       SE         tStat       pValue
          ________    _________    _______    __________

    x1    -0.27299    0.0070229    -38.871             0
    x2     0.36222     0.031966     11.332    9.8204e-30
    x3    -0.41284     0.033917    -12.172    4.8003e-34


Number of observations: 60128, Error degrees of freedom: 60125
Root Mean Squared Error: 1.45

mdl_c =


Linear regression model:
    y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7
```

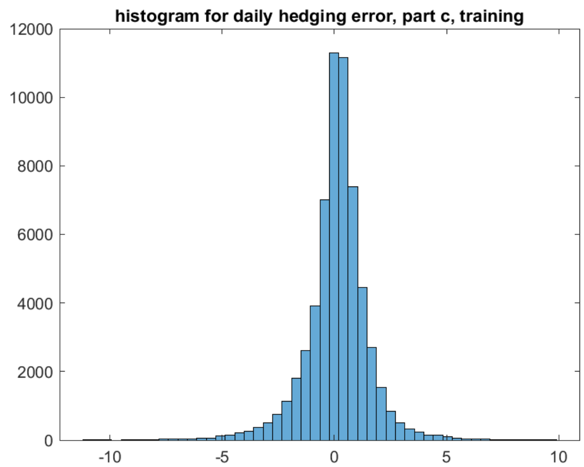
```
Estimated Coefficients:
                 Estimate          SE         tStat       pValue
               ___________    __________    _______    ___________

    x1             0.12593      0.003462     36.374     1.4326e-286
    x2         -0.00014369    2.8198e-06    -50.958               0
    x3             0.73666      0.019205     38.359     4.5928e-318
    x4             0.49336      0.049849     9.8971      4.4614e-23
    x5         -0.00017606    2.0208e-05    -8.7128      3.0394e-18
    x6          1.4767e-08     1.597e-08    0.92464         0.35516
    x7            -0.25453       0.03589    -7.0918      1.3379e-12


Number of observations: 60128, Error degrees of freedom: 60121
Root Mean Squared Error: 1.41

result =

  6×4 table

                       mean      standard deviation    95% VaR    95% CVaR
                     ________    __________________    _______    ________

    BS,train          0.17121          1.6783          -2.4483    -4.2003
    BS,test          0.071551          2.4473          -3.2446    -6.2405
    MV,train          0.11673          1.4425          -2.1993    -3.7314
    MS,test          0.052722          1.9859          -2.8954    -4.7596
    part c,train      0.12511          1.4034          -2.1806    -3.4717
    part c,test      0.066689          2.0314          -3.1757    -4.8612


Gain =

  1×4 table

    MV,train    MV,test    part c,train    part c,test
    ________    _______    ____________    ___________

    0.26409     0.34161      0.30251         0.31082
```

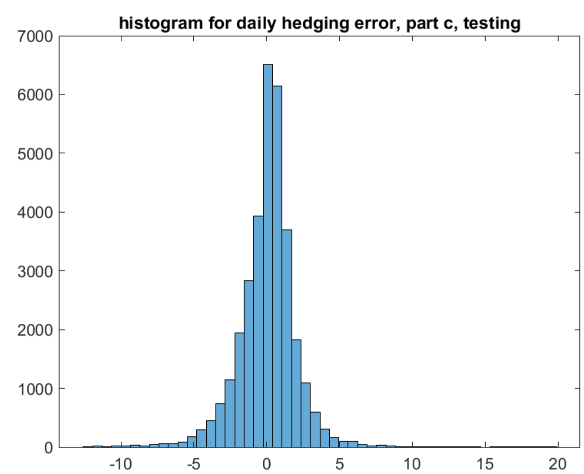

histogram for daily hedging error, BS, training
16000
14000
12000
10000
8000
6000
4000
2000
0
-15
-10
-5
0
5
10
15


histogram for daily hedging error, BS, testing
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
-20
-15
-10
-5
0
5
10
15
20
25


histogram for daily hedging error, MV, training
15000
10000
5000
0
-15
-10
-5
0
5
10


histogram for daily hedging error, MV, testing
8000
7000
6000
5000
4000
3000
2000
1000
0
-15
-10
-5
0
5
10
15
20


histogram for daily hedging error, part c, training
12000
10000
8000
6000
4000
2000
0
-10
-5
0
5
10


histogram for daily hedging error, part c, testing
7000
6000
5000
4000
3000
2000
1000
0
-10
-5
0
5
10
15
20