Noé's Matlab library - Financial Engineering course - LAB 1 :

Assignment 1 : Option price in *Matlab*.

European Option pricing with closed formula :

- *Formula used* : **Black's formula** for pricing futures (forward) options :
  $C(0,T) = B(0,T) \times (F_0 \mathcal{N}(d_1) - K\mathcal{N}(d_2))$, where : $d_1 = \frac{1}{\sigma\sqrt{T}} ln(\frac{F_0}{K}) + \frac{1}{2}\sigma\sqrt{T}$ and $d_2 = \frac{1}{\sigma\sqrt{T}} ln(\frac{F_0}{K}) - \frac{1}{2}\sigma\sqrt{T}$.
- *Inputs* : $F_0 = F(0,T)$ is the forward price of the underlying asset ; K is the strike price of the option ; B is the discount factor ; T is the time to maturity of the option ; σ is the annual volatility of the underlying asset ; $flag$ is the parameter to choose between call (1) or put (-1).
- *Ouput* : the price of the chosen european option (call or put).
- *Example* :

```
% F0=100; K=100; B=0.95; T=1; sigma=0.2; flag=1;
% optionPrice=EuropeanOptionClosed(F0,K,B,T,sigma,flag)

function optionPrice = EuropeanOptionClosed(F0, K, B, T, sigma, flag)
    %%
    % European option price with Closed formula.
    %%
    [call, put] = blkprice(F0, K, 0, T, sigma);
    % Choose the type of option :
    if flag == 1
       optionPrice = B * call;
    else
       optionPrice = B * put;
    end
end
```

European Option pricing with CRR tree method :

- *Method used* : the forward has a process that should respect the no-arbitrage theory : $F_0 = \mathbb{E}_0[F(1,1)] = \mathbb{E}_0[F_0 \times v] = F_0 \times \mathbb{E}[v]$ so in order to be a *Martingale* we need to have : $Pr(v = u) = q = \frac{1-u}{u-d}$. We use recombining tree, so we consider the random variable $\eta = log(v) = \pm\Delta x$ (v is a R.V than can go up u or down d in one timestep). So $u = exp(\Delta x) = 1/d$ $(d = exp(-\Delta x))$. **CRR** imposes : $Var(\eta) = (\Delta x)^2 + O((\Delta x)^4) \approx (\Delta x)^2$ and **Black** imposes : $Var(F(T,T)/F(0,T)) = \sigma^2\Delta t$. So finally : $\sigma^2\Delta t = (\Delta x)^2$ and we get : $u = e^{\sigma\sqrt{\Delta t}}$ and we *only have **one parameter left (σ).***
- *Tree structure implementation details :* see the comments in the code below.
- *Inputs* : same as for the closed formula method except for N which represents the number of timesteps in the binomial tree.
- *Ouput* : the CRR price of the chosen european option (call or put).

• *Example* :

```
% F0=100; K=100; B=0.95; T=1; sigma=0.2; N=100; flag=1;
% optionPrice=EuropeanOptionCRR(F0,K,B,T,sigma,N,flag)

function optionPrice = EuropeanOptionCRR(F0, K, B, T, sigma, N, flag)
    %%
    % European option price with CRR method
    %%
    % Calculate time step :
    dt = T/N;
    % Calculate up and down factors (cf explanation above) :
    u = exp(sigma * (dt^0.5));
    d = 1/u;
    % Calculate risk-neutral probability (cf explanation above) :
    p = (1 - d) / (u - d);
    % Initialize an array to store the prices :
    Price = [1 : N + 1];
    %
    % Initialize option prices AT MATURITY (we know the value : payoff).
    % Loop over each node in the binomial tree at time t = T
    for i = 1:N+1
        % Calculate and assign the asset price at each node using the up
        % factor u :
        % From u^(-2N) to u^(2N), so :
        % Tree(i) = (u^(N + 2 - 2 * i)); % Idx starts at 1 !
        %
        % Calculate and assign the option price at EACH NODE i :
        % C(t{N-1}, t{N}) = B(t{N-1}, t{N}) * E_t{N-1}[max(F0 * (u^(N + 2 -
2 * i)) - K, 0)]
        Price(i) = B * (p * max(F0 * (u^(N + 2 - 2 * i)) - K, 0) + (1 - p)
* max(F0 * (u^(N + 2 - 2 * (i + 1))) - K, 0));
    end
    %
    % Backward iteration to compute option prices at each node :
    for j = 1:N-1
        % Iterate over each node at the current time step
        for i = 1:N-j
            % Calculate the option price at each node using risk-neutral
probability (p)
            Price(i) = (B^(1/N)) * ((1 - p) * Price(i+1) + p * Price(i));
            % B^(1/N) = discount factor for each time step in CRR tree
        end
    end
    %
    % Final option price at t = 0 :
    call = Price(1);
    %
    % Calculate option price based on option type (call or put) :
```

```
    if flag == 1
       optionPrice = call;
    else
       optionPrice = call - B * (F0 - K); % Put-Call Parity
    end
end
```

European Option pricing with MC method :

- *Method used* : consider a derivative dependent on a single market value variable S, that provides a payoff at time T. Assuming constant interest rates, we can value the derivative as follows. **First**, sample a random path for S in a risk-neutral world. **Next**, calculate the payoff for the derivative. **Then**, repeat the first two steps to get many sample payoffs to get an estimate of the expected payoff in a risk-neutral world. **Next**, compute the mean (i.e an estimate of the expected payoff in the risk-neutral world). **Finally**, discount the expected payoff at the risk-free rate to get an estimate of the value of the derivative.
- *Inputs* : same as for the CRR method except for N which represents the number of simulations for the MC.
- *Ouput* : the MC price of the chosen european option (call or put).
- *Example* :

```
% F0=100; K=100; B=0.95; T=1; sigma=0.2; N=100; flag=1;
% optionPrice=EuropeanOptionMC(F0,K,B,T,sigma,N,flag)

function optionPrice = EuropeanOptionMC(F0,K,B,T,sigma,N,flag)
    %%
    % European option price with Monte Carlo method.
    %%
    sum = 0; % Initialisation of the average
    for i = 1:N % Simulation of N MC paths
        g = random('Normal', 0, 1);
        sum = sum + max(F0 * exp(-((sigma^2) / 2) * T + sigma * (T^(1/2)) *
g) - K, 0);
    end
    sum = sum / N; % Mean to get the Expected value
    call = B * sum; % Discount
    if flag == 1
       optionPrice = call;
    else
       optionPrice = call - B * (F0 - K); % Put-Call Parity
    end
end
```

Pricing with different methods :

Parameters :

```
%% Pricing parameters :
S0 = 1;
K = 1;
```

```
r = 0.03;
TTM = 1/12;
sigma = 0.18;
flag = -1; % flag:  1 call, -1 put
d = 0.04;

%% Quantity of interest :
B = exp(-r * TTM); % Discount factor
F0 = S0 * exp(-d * TTM) / B; % Forward in B&S Model

%% Simulation parameters :
pricingMode = 1; % 1 ClosedFormula, 2 CRR, 3 Monte Carlo
M = 100000; % M = simulations for MC, steps for CRR
```

Question a) Different pricing methods :

```
%% Pricing :
OptionPrice = EuropeanOptionPrice(F0, K, B, TTM, sigma, pricingMode, M,
flag);
disp(['Option Price with the given pricing mode : ', num2str(pricingMode),
' - 1 ClosedFormula, 2 CRR, 3 Monte Carlo - ', num2str(OptionPrice)]);
%OptionPriceClosed = EuropeanOptionPrice(F0, K, B, TTM, sigma, 1, M, flag);
%disp(['Option Price in closed formula : ', num2str(OptionPriceClosed)]);
%OptionPriceCRR = EuropeanOptionPrice(F0, K, B, TTM, sigma, 2, M, flag);
%disp(['Option Price in CRR : ', num2str(OptionPriceCRR)]);
%OptionPriceMC = EuropeanOptionPrice(F0, K, B, TTM, sigma, 3, M, flag);
%disp(['Option Price in MC : ', num2str(OptionPriceMC)]);
```

Question b) Error plots :

Function to plot the error of CRR method :

- *Inputs* : $F_0 = F(0,T)$ is the forward price of the underlying asset ; K is the strike price of the option ; B is the discount factor ; T is the time to maturity of the option ; σ is the annual volatility of the underlying asset.
- *Outputs* : $M$is the number of steps for the CRR method, $errorCRR$ is the error of the CRR price.
- ***Hint 1*** : as CRR error, consider the difference in absolute value w.r.t the exact MATLAB value.
- *Example* :

```
% F0=100; K=100; B=0.95; T=1; sigma=0.2;
% [M, errorCRR] = PlotErrorCRR(F0, K, B, T, sigma)

function [M, errorCRR] = PlotErrorCRR(F0, K, B, T, sigma)
    %%
    % Plot the Error of CRR method.
    %%
    M = [1:9]; % Number of intervals in CRR
    for i = 1:9
```

```
        M(i) = 2^i; % Number of timesteps in the CRR tree from 2 to 2^9
    end
    %
    flag = 1; % EU Call Option
    valRef = EuropeanOptionClosed(F0, K, B, T, sigma, flag); % REF = MATLAB
closed formula value (cf above)
    %
    errorCRR = [1:9];
    for i = 1:9
        OptionPrice = EuropeanOptionCRR(F0, K, B, T, sigma, 2^i, flag);
        errorCRR(i) = abs(OptionPrice - valRef);
    end
    %
    L = [1:9];
    for i = 1:9
        L(i) = 1/M(i);
    end
    %
    loglog(M, errorCRR) % ErrorCRR vs 1/M in loglog scale
    hold on;
    xlabel('M (logscale)');
    ylabel('ErrorCRR (logscale)');
    title('ErrorCRR varying M (loglog)');
    % Add a horizontal line at y = 10^-4
    yline(10^-4, '--r', 'Criteria for selecting M');
    hold off;
    disp("CRRError FINISHED.")
end

% Plot Errors for CRR varing number of steps
figure;
[nCRR, errCRR] = PlotErrorCRR(F0, K, B, TTM, sigma);
```

Function to plot the error of the MC method :

- *Inputs* : same as just above.
- *Outputs* : $M$is the number of simulation for MC, $stdEstim$ is the standard deviation of the MC price.
- *Formula used* : as an estimator of σ^2, we use $\hat{s}_1 = \frac{1}{M}\sum_{i=1}^{M}(O_i - \mu)^2$, where O_i is the i-th simulated price.
- ***Hint 1*** : as MC error, consider an estimation of the unbiased standard deviation of the MC price.
- *Example* :

```
% F0=100; K=100; B=0.95; T=1; sigma=0.2;
% [M, stdEstim] = PlotErrorMC(F0, K, B, T, sigma)

function [M, stdEstim] = PlotErrorMC(F0, K, ~, T, sigma)
    %%
    % Plot the Error of MC method.
    %%
    % Initialize arrays for M and stdEstim
    M = [1:20];
    stdEstim = [1:20];
    %
    for i = 1:20
        M(i) = 2^(i); % Number of MC simulations from 2 to 2^20
    end
    %
    for N = 1:20
        % Initialize vector with size 2^N
        vect = [1:2^(N)];
        avg = 0;
        %
        for i = 1:2^(N)
            g = random('Normal', 0, 1);
            % i-th element : i-th simulated price
            vect(i) = max(F0 * exp(-((sigma^2) / 2) * T + sigma * (T^(1 /
2)) * g) - K, 0);
            avg = avg + vect(i);
        end
        % Calculate the final average -> Expected value of the option price
        avg = avg / (2^(N));
        % Initialize estimator of the standard deviation
        est = 0;
        % Loop over 2^N iterations
        for i = 1:2^(N)
            % Update the estimator
            est = est + (vect(i) - avg)^2;
        end
        % Unbiased estimator of the standard deviation of the option price
        stdEstim(N) = sqrt(est) / (2^(N));
    end
    %
    loglog(M, stdEstim)
    hold on;
    xlabel('N (logscale)');
    ylabel('ErrorMC (logscale)');
    title('ErrorMC varying N (loglog)');
    % Add a horizontal line at y = 10^-4
    yline(10^-4, '--r', 'Criteria for selecting N');
    hold off;
    disp("MCError FINISHED.")
end

% Plot Errors for MC varing number of simulations
figure;
[nMC, stdEstim] = PlotErrorMC(F0, K, B, TTM, sigma);
```

Question c) Error magnitude vs M and N for CRR and MC :

- *Hint* : in log-log plot, a straight line represents a power law relationship.

Numerical error for a Call is about 1/M for CRR :

```
% Perform linear regression (polyfit)
p = polyfit(log10(nCRR), log10(errCRR), 1);
% Generate the fitted curve using the coefficients obtained from polyfit
fitted_curve = polyval(p, log10(nCRR));
disp(['Linear regression SLOPE coefficient for ErrorCRR : ',
num2str(p(1))]);
%
%% Plot :
%
figure;
% Plot the data points
scatter(log10(nCRR), log10(errCRR), 'b', 'filled'); % Blue points
hold on;
% Plot the fitted curve
plot(log10(nCRR), fitted_curve, 'r'); % Red line
% Customize plot
xlabel('M (logscale)');
ylabel('ErrorCRR');
title('Linear Regression Fit for ErrorCRR');
legend('Data', 'Fitted curve with a straight line (poly of degree 1)');
hold off; % release the plot
```

Numerical error for a Call is about $1/\sqrt{M}$ for MC :

```
% Perform linear regression (polyfit)
q = polyfit(log10(nMC), log10(stdEstim), 1);
% Generate the fitted curve using the coefficients obtained from polyfit
fitted_curve = polyval(q, log10(nMC));
disp(['Linear regression SLOPE coefficient for ErrorMC : ', num2str(q(1))]);
%
% Plot :
%
figure;
% Plot the data points
scatter(log10(nMC), log10(stdEstim), 'b', 'filled'); % Blue points
hold on;
% Plot the fitted curve
plot(log10(nMC), fitted_curve, 'r'); % Red line
% Customize plot
xlabel('n');
ylabel('ErrorMC');
title('Linear Regression Fit for ErrorMC');
legend('Data', 'Fitted curve with a straight line (poly of degree 1)');
hold off; % release the plot
```

Question d) EU Call Option with KO barrier :

European option pricing with KO barrier with MC method :

- *Inputs* : same as for the classical option pricing functions (above) ; in addition, K_0 is the barrier price of the option.
- *Output* : the price of the European barrier call option.
- *Method* : same as for classical MC pricing of EU option, but we remove all the paths where the barrier is crossed (so we assign a zero in this case).

```
% F0=100; K=100; KO=1.2; B=0.95; T=1; sigma=0.2; N=10^6;
% optionPrice = EuropeanOptionKOMC(F0, K, KO, B, T, sigma, N)

function optionPrice = EuropeanOptionKOMC(F0, K, KO, B, T, sigma, N)
    %%
    % European barrier call option price up&in with MC method.
    %%
    sum = 0; % initialisation of the average
    %
    for i = 1:N % Simulation of N MC paths
        g = random('Normal',0,1);
        if exp(-((sigma^2) / 2) * T + sigma * (T^(1/2)) * g) < KO % Is the
barrier crossed ?
            sum = sum + max(F0 * exp(-((sigma^2)/2) * T + sigma * (T^(1/2))
* g) - K, 0);
        else
            sum = sum + 0; % Barrier is crossed : 0 value for this path
        end
    end
    %
    sum = sum / N; % Avg to get the expected value
    call = B * sum;
    optionPrice = call;
    %
    disp('KO BARRIER CALL OPTION PRICE FINISHED (MC METHOD).')
end
```

European option pricing with KO barrier with CRR method :

- *Inputs* : same as for the classical option pricing functions (above) ; in addition, K_0 is the barrier price of the option.
- *Output* : the price of the European barrier call option.
- *Method* : same as for classical CRR pricing of EU option, but the option value is put to zero if the value is above the barrier. So for all the computation part, it is explained in the classical CRR option pricing function above.

```
% F0=100; K=100; KO=90; B=0.95; T=1; sigma=0.2; N=100; flag=1;
% optionPrice=EuropeanOptionKOCRR(F0,K,KO,B,T,sigma,N,flag)

function optionPrice = EuropeanOptionKOCRR(F0, K, KO, B, T, sigma, N)
    %%
    % European barrier call option price up&in with CRR method.
    %%
    % Time step :
    dt = T/N;
    % u and d (up and down factors) :
    u = exp(sigma*((dt)^(1/2)));
    d = 1/u;
    % Risk-neutral probability :
    p = (1-d)/(u-d);
    % Initialize arrays to store prices :
    Price = [1:N+1];
    % Initialize option prices AT MATURITY.
    % Option value is zero if the value is above the barrier :
    for i = 1:N+1
        if u^(N+2-2*(i+1)) < KO % Is down branch below the barrier ?
            if u^(N+2-2*(i)) < KO % Is up branch below the barrier ?
                Price(i) = B * (p * max(F0 * (u^(N + 2 - 2 * i)) - K, 0) +
(1 - p) * max(F0 * (u^(N + 2 - 2 * (i + 1))) - K, 0));
                % No null value (both branches are below the barrier)
            else
                Price(i) = B * (p * 0 + (1 - p) * max(F0 * (u^(N + 2 - 2 *
(i + 1))) - K, 0));
                % Null value for the up branch since it is above the
barrier (KO the up branch)
            end
        else
            Price(i) = 0; % The option is worth 0 : both branches are above
the barrier.
        end
    end
    %
    % Backward iteration : same as classical CRR.
    for j = 1:N-1
        for i = 1:N-j
            Price(i) = (B^(1/N)) * ((1 - p) * Price(i + 1) + p * Price(i));
        end
    end
    %
    call = Price(1);
    optionPrice = call;
    %
    disp('KO BARRIER CALL OPTION PRICE FINISHED (CRR METHOD).');
end
```

Parameters and pricing with the two methods + the closed formula :

```
KO = 1.3; % Barrier

% Price with MC method :
N = 10^6; % Previously found N
OptionPriceKOMC = EuropeanOptionKOMC(F0,K,KO,B,TTM,sigma,N);
disp(['Barrier Call Option Price with MC : ', num2str(OptionPriceKOMC)]);

% Price with CRR method :
N = 100; % Previously found N
OptionPriceKOCRR = EuropeanOptionKOCRR(F0,K,KO,B,TTM,sigma,N);
disp(['Barrier Call Option Price with CRR : ', num2str(OptionPriceKOCRR)]);
% Price with closed formula :
OptionPriceKOBLACK = barrierbybls(intenvset('Rates',r-
d,'StartDates','03-Feb-2012','EndDates','03-Mar-2012','ValuationDate','03-
Feb-2012'),stockspec(sigma, S0),'call',K,'03-Feb-2012','03-Mar-2012',
'UO',KO);
disp(['Barrier Call Option Price with Black-Scholes : ',
num2str(OptionPriceKOBLACK)]);
```

Question e) KO Option Vega plot :

It works also with other greeks (cf below with the gamma).

```
x = linspace(0.70, 1.5, 1000); % underlying price in the range 0.70 Euro
and 1.5 Euro
Vega = zeros(1, 1000); % 1000 points (cf the line above)
KO_vega = 1.6; % WARNING : the fct does not accept KO Barrier <= Underlying
Price
%
for i = 1:1000
    Vega(i) = barriersensbyfd(intenvset('Rates', r-d, 'StartDates', '03-
Feb-2012', 'EndDates', '03-Mar-2012', 'ValuationDate', '03-Feb-2012'),
stockspec(sigma, x(i)), 'Call', K, '03-Feb-2012', '03-Mar-2012', 'UO',
KO_vega, 'OutSpec', 'vega');
    % Works also with other Greeks :
    % Gamma(i) = barriersensbyfd(intenvset('Rates',r-
d,'StartDates','03-Feb-2012','EndDates','03-Mar-2012','ValuationDate','03-
Feb-2012'),stockspec(sigma, x(i)),'Call',K,'03-Feb-2012','03-Mar-2012',
'UO',KO, 'OutSpec', 'gamma');
end
%
figure;
plot(x, Vega);
xlabel('Underlying Price');
ylabel('Vega');
title('Vega of Barrier Call Option');
```