ACTSC 971 A21Q1(a)

Euler approximation:

1. Partition $[0,T]$ into M subintervals $0=t_0<t_1<\cdots<t_{M-1}<t_M=T$

Calculate $\Delta=\frac{T}{M}$ (step size)

2. Next, we want to generate a path. Repeat:

(i) Generate M independent random variates (twice) from the standard normal distribution, $\{Z_1,\cdots Z_M\}$ and $\{Z_1',\cdots Z_M'\}$

(ii) Obtain points on the trajectory of process $\{(S(t),Y(t))\}_{t\in[0,T]}$

Let $S(0)=s$, $Y(0)=y$. The paths be $\{(S(t_0),y(t_0)),(S(t_1),y(t_1)),\cdots\}$

Then the recursion is given by

$$\begin{cases} S(t_0)=s \\ y(t_0)=y \\ S(t_{i+1})=S(t_i)+rS(t_i)\Delta+S(t_i)\sqrt{y(t_i)}\sqrt{\Delta}\,Z_{i+1} \\ y(t_{i+1})=y(t_i)+k(\bar{y}-y(t_i))\Delta+\eta\sqrt{y(t_i)}\left[\rho\sqrt{\Delta}\,Z_{i+1}+\sqrt{1-\rho^2}\sqrt{\Delta}\,Z'_{i+1}\right] \end{cases}$$

for $i=0,1,\cdots M-1$

(Note $\bar{y}$ is a given parameter, not the average of path $y(t_i)$.

Q1

```
%rng('default') % For reproducibility
% evaluation
etas = [0.001, 0.01, 0.05, 0.1, 0.15, 0.2];
r = 0.2;
k = 2;
ybar = 0.01;
rho = -0.3;
s0 = 100;
y0 = 0.01;
T = 1;



% guess a K; try different M and N
K = 105;
M_ls = [100, 1000, 10000, 100000];
N_ls = [100, 1000, 10000, 10000];
% length of M = length of N
```

Apply the Euler's approximation. Put (option value, standard error) in a table

```
% table for option values: row for M and column for eta
vals = zeros(length(M_ls), length(etas));
% table for standard errors
errs = zeros(length(M_ls), length(etas));

for i = 1:length(M_ls)
    M = M_ls(i);
    N = N_ls(i);
    for j = 1:length(etas)
        eta = etas(j);
        [val, err] = EulerApprox(r, k, ybar, eta, rho, s0, y0, K, M, N, T);
        vals(i,j) = val;
        errs(i,j) = err;
    end
end
```

View option values

```
rowNames = {'M = 100, N = 100', '1000, 1000', '10000, 10000', '100000, 10000'};
colNames = {'eta = 0.001', '0.01', '0.05', '0.1', '0.15', '0.2'};
vals = array2table(vals, "RowNames",rowNames, "VariableNames", colNames)
```

```
vals = 4×6 table
```

	eta = 0.001	0.01	0.05	0.1	0.15	0.2
1 M = 100, N = 100	5.4873	5.5353	5.5375	5.5681	5.5813	5.5571
2 1000, 1000	4.7639	4.7565	4.7580	4.7644	4.7571	4.7650
3 10000, 10000	4.6761	4.6771	4.6760	4.6771	4.6767	4.6775
4 100000, 10000	4.6769	4.6770	4.6772	4.6769	4.6771	4.6769

Next, look at the standard errors

```
errs = array2table(errs, "RowNames",rowNames, "VariableNames", colNames)
```

```
errs = 4×6 table
```

	eta = 0.001	0.01	0.05	0.1	0.15	0.2
1 M = 100, N = 100	0.0535	0.0511	0.0574	0.0551	0.0551	0.0561
2 1000, 1000	0.0054	0.0051	0.0055	0.0052	0.0053	0.0055
3 10000, 10000	0.0005	0.0005	0.0005	0.0005	0.0005	0.0005
4 100000, 10000	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002

For a fixed η , the option payoff and error start to converge at M = 10000. We can pick M = 100000 and N = 10000. The M and N values are not too large to slow down the program; and they are also not too small to give inaccurate option prices and large standard errors.

Function

The function 'EulerApprox' is the Euler's approximation method to value the Asian option under Heston model.

Input:

- r: interest rate
- k, ybar, eta, rho: constants in Heston model
- s0: initial asset price
- y0: initial Y
- K: strike price
- M: number of simulations
- N: number of subintervals
- T: expiry time

Output:

a pair of

- val: time-0 option price (as the mean of payoffs over M simulations)

- stderr: standard error

```
function [val, stderr] = EulerApprox(r, k, ybar, eta, rho, s0, y0, K, M, N, T)
    % delta
    stepsize = T/N;

    % paths: a M * (N+1) matrix (i.e. M paths, each path has N+1 points)
    s_paths = zeros(M,N+1);
    y_paths = zeros(M,N+1);
    % generate M simulations

    % every path's initial values are s0 and y0
    s_paths(:,1) = s0;
    y_paths(:,1) = y0;

    for j = 1:N
       % generate N independent standard normal twice because we have
       % different brownian motion in dY(t)
       % In each simulation, we are going to generate a path of length N
       stdn_1 = normrnd(0,sqrt(stepsize),[M,1]);
       stdn_2 = normrnd(0,sqrt(stepsize),[M,1]);
       % forward recursion to get y and s
       y_paths(:,j+1) = y_paths(:,j) + k .* (ybar - y_paths(:,j)) .* stepsize + ...
                        eta .* sqrt(y_paths(:,j)) .* (rho * sqrt(stepsize) ...
                        .* stdn_1 + sqrt((1-rho^2) .* stepsize) .* stdn_2);
       s_paths(:,j+1) = s_paths(:,j) + r .* s_paths(:,j) .* stepsize + ...
                        s_paths(:, j) .* sqrt(y_paths(:,j) .* stepsize) .* stdn_1;
    end

    % next, let's calculate the payoff for every path
    payoffs = zeros(1,M);
    for i = 1:M
        % payoff function
        payoffs(i) = max(sum(s_paths(i,:) * stepsize) - K, 0);
    end

    % result is the mean of payoffs
    val = exp(-r*T) * mean(payoffs);

    % error
    stderr = std(exp(-r*T) * payoffs) / sqrt(M);

end
```