MTH5530 Report

Andrew Buttigieg
Chrysovalantis Thomopoulos

Biased simulation schemes for stochastic volatility models are analysed. Explanations are provided in regards to the Feller condition and its violation. Methods to counter this possible violation are then described and we conduct numerical analysis n order to depict the bias, standard error, RMSE and run time for all of our schemes. Finally, we compare the $\omega = 1$ and $\omega = 0.3$ cases.

Contents

1 Introduction

In simulating CEV-SV stochastic volatility models and in particular the Heston model, problems have risen with accuracy and time loss have deeming specific simulations inefficient and inaccurate. In particular when seeking a Euler discretisation, negative values for volatility can cause the discretisation to fail when the Feller condition for the Heston volatility model is violated. This is described in Lord, Koekkoek, and Dijk 2010 where different fixes for this issue are discussed. Our aim is to compare the computational efficiency and estimation accuracy of four different fixes for this issue, the Absorption, Reflection, Partial Truncation and Full Truncation schemes and understand how the change in volatility changes our results.

2 Method and Results

Question 1

The Feller condition as presented below describes when parameters of a Heston stochastic volatility model do not give negative values for the volatility;

$$\omega^2 \leq 2\kappa\theta.$$

When the inequality holds, no negative values for volatility occur. The Heston stochastic volatility model is depicted below;

$$\begin{aligned} dS_t &= rS_t dt + \sqrt{V_t} S_t dW_t^1 \\ dV_t &= -\kappa(V_t - \theta)dt + \omega\sqrt{V_t} dW_t^2 \\ dW_t^1 dW_t^2 &= \rho dt, \end{aligned}$$

The V_t term has to be positive as to prevent imaginary values for the asset price. However in many cases, valid parameters of the Heston are given which violate the Feller condition, such as the parameters provided which we model in Question 3 where $\omega^2 = 1$ and $2\kappa\theta = 0.36$. When we discretize and use our modified Euler schemes of Full Truncation, Absorbtion etc. we see these as multiple solutions to keeping the volatility process positive.

Question 2

Based on Lord, Koekkoek, and Dijk 2010 we know that our Euler schemes can be unified like so:

$$\begin{aligned} \tilde{V}(t + \Delta t) &= f_1(\tilde{V}(t)) - \kappa\Delta t(f_2(\tilde{V}(t)) - \bar{V}) + \omega f_3(\tilde{V}(t))^{1/2}\Delta W_V(t) \\ V(t + \Delta t) &= f_3(\tilde{V}(t + \Delta t)) \end{aligned}$$

With $\tilde{V}(0) = V(0)$, $f_i(x) = x$ for $x \geq 0$ and $i = 1, 2, 3$ and $f_i(x) \geq 0$ for $x \in \mathbb{R}$ and $i = 1, 3$.
For the absorption method we take;

$$f_i(x) = x^+,\ i = 1, 2, 3$$

Which forces all negative values achieved by the volatility process to become zero - hence zero becomes akin to an absorbing state. For the reflection method we take;

$$f_i(x) = |x|,\ i = 1, 2, 3$$

Which reflects the negative values in regards to the origin, so the process does not reach zero at all. For the partial truncation we take;

$$f_1(x) = x,\ f_2(x) = x,\ f_3(x) = x^+$$

Which only takes the absolute value of the process after all intermediary calculations. Lastly for the full truncation we take;

$$f_1(x) = x,\ f_2(x) = x^+,\ f_3(x) = x^+$$

In each correction scheme bias is introduced creating drift in the variance which deviates the correction schemes behaviour away from the behaviour of the true process. As such, each scheme was developed iteratively to reduce the error from altering the original Heston model introduced by the previous scheme. The reflection method was determined to have the largest positive bias. Broadie and Kaya 2006 numerically determined that the absorption fix introduces bias in pricing a vanilla European option but less than that of the reflection fix. The partial and full truncation methods highlight successive attempts to further reduce this introduced bias. In all cases log asset price is calculated to ensure non negativity:

$$lnS(t+\Delta t) = lnS(t) + (\mu - \frac{1}{2}\lambda^2 S(t)^{\beta-1} V(t))\Delta t$$
$$+\lambda S(t)^{\beta-1}\sqrt{V(t)}\Delta W_S(t)$$

Question 3

We use the absorption, reflection, partial truncation and full truncation schemes to compare the bias, standard error, RMSE and run time of each.

Method	Paths	10,000	40,000	160,000
	Steps/year	20	40	80
Absorption scheme	Bias	2.0504	1.5689	1.2031
	Standard Error	0.5851	0.3403	0.1490
	RMSE	2.1322	1.6054	1.2123
	Run time	0.0439	0.2491	3.0844
Reflection scheme	Bias	4.4632	3.2121	2.3686
	Standard Error	0.7651	0.3630	0.1799
	RMSE	4.5283	3.2326	2.3755
	Run time	0.0447	0.2893	3.8939
Partial truncation scheme	Bias	0.4295	0.2312	0.0905
	Standard Error	0.5262	0.2917	0.1429
	RMSE	0.6792	0.3722	0.1692
	Run time	0.0594	0.3842	4.3039
Full truncation scheme	Bias	0.0461	0.0216	0.0020
	Standard error	0.5648	0.2808	0.1391
	RMSE	0.5667	0.2816	0.1391
	Run time	0.0635	0.3743	2.7354

As can be expected, bias, standard errors, and, RMSE improve as paths and steps per year increase whilst run time increases reflecting the additional calculations involved increasing paths and number of steps. The bias of the full truncation scheme is the lowest out of them all regardless of the paths and steps/year, as is the RMSE. This agrees with theory from Lord, Koekkoek, and Dijk 2010 as was discussed when presenting each scheme. The standard error is relatively uniform for all schemes.

Question 4

We use the full truncation scheme with the given $\Delta S = 0.01 S_0$, 100, 00 simulations and 50 time steps/year. We compare the delta and gamma curves that arise with the Black-Scholes model.

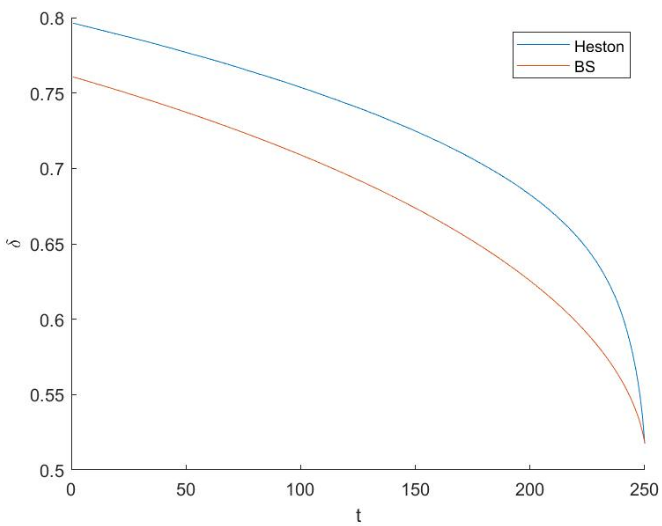

Figure 1: Delta Comparison

The value of Delta measures the option prices sensitivity to a change in initial stock value, whereas the value of Gamma measures the sensitivity of the value of Delta to the change in the initial stock value. A Delta value of 1 represents a $1 increase in in option price when the stock price in increased by 1. Hence we see that when modelling the option price with the Heston volatility model, the sensitivity to the initial change in stock price was consistently higher at all time steps than in the case when volatility is constant in the Black-Scholes model. The difference as expected peters out to 0 as time approaches maturity - where the value of the option is known. This is mirrored in the value of Gamma though we see the difference in Gamma between the two volatility models become smaller. The higher value of delta also indicates that under the Heston model, the option has a higher chance of expiring in the money than in the case of BS. However, the slightly higher Gamma indicates that this probability is slightly more likely to fluctuate.

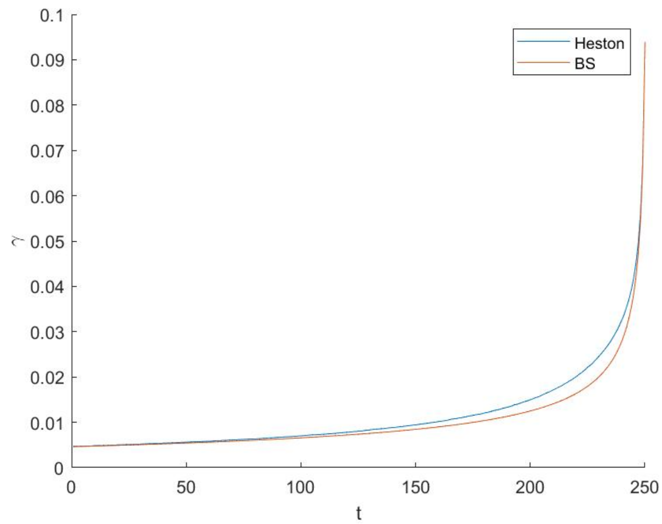

Figure 2: Gamma Comparison

Question 5

We now repeat the method we used before for $\omega = 0.3$

Method	Paths	10,000	40,000	160,000
	Steps/year	20	40	80
Absorption scheme	Bias	0.8418	0.8350	0.8811
	Standard Error	0.5958	0.3069	0.1574
	RMSE	1.0313	0.8896	0.8950
	Run time	0.0398	0.2437	2.5087
Reflection scheme	Bias	0.8814	0.8736	0.8731
	Standard Error	0.6475	0.2790	0.1550
	RMSE	1.0937	0.9171	0.8867
	Run time	0.0384	0.2419	3.3350
Partial truncation scheme	Bias	0.8752	0.8909	0.8940
	Standard Error	0.5959	0.2988	0.1514
	RMSE	1.0588	0.9397	0.9067
	Run time	0.0607	0.3707	3.9641
Full truncation scheme	Bias	0.8089	0.8872	0.8831
	Standard error	0.6841	0.3276	0.1668
	RMSE	1.0594	0.9457	0.8987
	Run time	0.0689	0.4223	4.6918

In comparing the two tables, it can be seen that standard error and runtime has remain relatively consistent across all schemes. There is however a marked increase in both bias and RMSE for both the partial and full truncation schemes while the values remain consistent for the absorption and reflection schemes. In setting $\omega = 0.3$ the Feller condition is no longer violated, that is

$$\omega^2 = 0.09 \leq 0.36 = 2\kappa\theta.$$

Hence it can be deduced that when the Feller is not violated, the partial and full truncation schemes distort results. This is in contrast to when the condition is violated where they prove to be the most accurate schemes. In this scenario however a correction scheme is not needed since the Feller condition will guarantee non negative values for volatility.

3 Conclusion

In this report we compare the effectiveness of four different corrections of the Heston stochastic volatility model in pricing an vanilla European call option when the Feller condition is violated. It was numerically determined that the bias and hence RMSE minimised most for the full truncation scheme followed by the partial truncation scheme with the reflection and absorption schemes performing markedly worse. This mirrors the results given in the paper. As expected the number of steps and paths increase the accuracy increased and error decrease while the run time increased. Hence the wisest choice in regards to accuracy whilst the Feller Condition does not hold is the Full Truncation scheme regardless of the paths and steps/year. In the case where the Feller condition does hold, the full truncation and partial truncation schemes perform the worst but in this scenario no correction scheme is needed in the first place as the Feller condition guarantees non-negative values for volatility in this scenario.

4 Bibliography

Broadie, Mark and Özgür Kaya (2006). "Exact Simulation of Stochastic Volatility and Other Affine Jump Diffusion Processes". English. In: *Operations research* 54.2. Copyright - Copyright Institute for Operations Research and the Management Sciences Mar/Apr 2006; Document feature - Equations; Graphs; Tables; ; Last updated - 2020-11-17; CODEN - OPREAI, pp. 217–231,402–403. URL: https://www.proquest.com/scholarly-journals/exact-simulation-stochastic-volatility-other/docview/219170321/se-2?accountid=12528.

Lord, Roger, Remmert Koekkoek, and Dick Van Dijk (2010). "A comparison of biased simulation schemes for stochastic volatility models". In: *Quantitative Finance* 10.2, pp. 177–194. DOI: 10.1080/14697680802392496. eprint: https://doi.org/10.1080/14697680802392496. URL: https://doi.org/10.1080/14697680802392496.

5 Appendix

Absorption

```
function [final_avg_price, RMSE, std_err, bias,
   runtime] = absorption(paths, steps_per_year, omega)
% Define function to calculate option price based on
   the Heston model
% modified by absorption method. Function of number of
    paths, steps
% per year and value of omega. Return option price,
   RMSE standard
% error bias and runtime.

    % Set problem parameters and initial conditions
    S0 = 100*ones(paths,1);
    K = 100;
    T = 5;
    r = 0.05;
    V0 = 0.09*ones(paths,1);
    theta = 0.09;
    kappa = 2;
    rho = -0.3;

    % Set step size for time and create vectors of
       volatilities/ log
    % stockprices to be populated.
    dt = 1/steps_per_year;
    M = 1/dt;
    [V,Vtil,logS] = deal(cell(1,T*(M)));

    % Define avg option price vector to be populated
    avg_option_price_vec = deal(zeros(1,100));

    % Define runtime vector to be populated
    timevec = zeros(1,100);

    % Repeat calculation over all time steps 100 times
        to calc avg
    for j = 1:100
        tic

        % Define each random time step using scaled
           standard normal random
        % variables and using Cholesky decomposition
           in two dimensions to
```

```
    % ensure the dWs is correlated
    dWv = sqrt(dt)*normrnd(0,1,[paths,T*(M)]);
    dWs = rho*dWv + sqrt(1-rho^2)*normrnd(0,1,[
        paths,T*(M)])*sqrt(dt);

    % Iterate through each timestep
    for i = 0:T*(M)-1

        % Input initial conditions into Heston
            model
        if i == 0
            Vtil{i+1} = f1(V0) - kappa.*dt.*(f2(V0
                )-theta) + omega.*sqrt(f3(V0)).*(
                dWv(:,i+1));
            V{i+1} = f3(Vtil{i+1});
            logS{i+1} = log(S0) + (r-(1/2).*V0).*
                dt + sqrt(V0).*dWs(:,i+1);

        % Iterate ith time step in discretised
            Heston model
        else
            Vtil{i+1} = f1(Vtil{i}) - kappa.*dt.*(
                f2(Vtil{i})-theta) + omega.*sqrt(f3
                (Vtil{i})).*(dWv(:,i+1));
            V{i+1} = f3(Vtil{i+1});
            logS{i+1} = logS{i} + (r-(1/2).*V{i})
                .*dt + sqrt(V{i}).*dWs(:,i+1);
        end
    end

    % Calculate vector of option price for each
        path using the final
    % value of the stock for each path
    option_price = pos(exp(logS{length(logS)})-K);

    % Calculate average over all paths of option
        prices
    avg_option_price_vec(j) = exp(-0.05*5)*mean(
        option_price)

    % Record runtime for this simulation
    timevec(j) = toc;
end

% Calculate average over 100 simulations of option
    price, std error,
```

```
    % bias, RMSE and runtime
    final_avg_price = mean(avg_option_price_vec)
    std_err = sqrt(mean((avg_option_price_vec-
       final_avg_price).^2))
    bias = abs(final_avg_price-34.9998)
    RMSE = sqrt(bias^2+std_err^2)
    runtime = mean(timevec)
end

% Define required f1, f2, and f3 for absorption method

function r = f1(x)
    r = max(x,0);
end

function r = f2(x)
    r = max(x,0);
end

function r = f3(x)
    r = max(x,0);
end

function r = pos(x)
    r = max(x,0);
end
```

Reflection

```
function [final_avg_price, RMSE, std_err, bias,
   runtime] = reflection(paths, steps_per_year, omega)
% Define function to calculate option price based on
   the Heston model
% modified by reflection method. Function of number of
    paths, steps
% per year and value of omega. Return option price,
   RMSE standard
% error bias and runtime.

    % Set problem parameters and initial conditions
    S0 = 100*ones(paths,1);
    K = 100;
    T = 5;
    r = 0.05;
    V0 = 0.09*ones(paths,1);
    theta = 0.09;
```

```
kappa = 2;
% omega = 1;
rho = -0.3;

% Set step size for time and create vectors of
   volatilities/ log
% stockprices to be populated.
dt = 1/steps_per_year;
M = 1/dt;
[V,Vtil,logS] = deal(cell(1,T*(M)));

% Define avg option price vector to be populated
avg_option_price_vec = deal(zeros(1,100));

% Define runtime vector to be populated
timevec = zeros(1,100);

% Repeat calculation over all time steps 100 times
    to calc avg
for j = 1:100
    tic

    % Define each random time step using scaled
       standard normal random
    % variables and using Cholesky decomposition
       in two dimensions to
    % ensure the dWs is correlated
    dWv = sqrt(dt)*normrnd(0,1,[paths,T*(M)]);
    dWs = rho*dWv + sqrt(1-rho^2)*normrnd(0,1,[
       paths,T*(M)])*sqrt(dt);

    % Iterate through each timestep
    for i = 0:T*(M)-1
        % Input initial conditions into Heston
           model
        if i == 0
            Vtil{i+1} = f1(V0) - kappa.*dt.*(f2(V0
               )-theta) + omega.*sqrt(f3(V0)).*(
               dWv(:,i+1));
            V{i+1} = f3(Vtil{i+1});
            logS{i+1} = log(S0) + (r-(1/2).*V0).*
               dt + sqrt(V0).*dWs(:,i+1);

        % Iterate ith time step in discretised
           Heston model
        else
```

```
                Vtil{i+1} = f1(Vtil{i}) - kappa.*dt.*(
                   f2(Vtil{i})-theta) + omega.*sqrt(f3
                   (Vtil{i})).*(dWv(:,i+1));
                V{i+1} = f3(Vtil{i+1});
                logS{i+1} = logS{i} + (r-(1/2).*V{i})
                   .*dt + sqrt(V{i}).*dWs(:,i+1);
            end
        end

        % Calculate vector of option price for each
           path using the final
        % value of the stock for each path
        option_price = pos(exp(logS{length(logS)})-K);

        % Calculate average over all paths of option
           prices
        avg_option_price_vec(j) = exp(-0.05*5)*mean(
           option_price)

        % Record runtime for this simulation
        timevec(j) = toc;
    end

    % Calculate average over 100 simulations of option
        price, std error,
    % bias, RMSE and runtime
    final_avg_price = mean(avg_option_price_vec)
    std_err = sqrt(mean((avg_option_price_vec-
       final_avg_price).^2))
    bias = abs(final_avg_price-34.9998)
    RMSE = sqrt(bias^2+std_err^2)
    runtime = mean(timevec)
end

% Define required f1, f2, and f3 for reflection method

function r = f1(x)
    r   = abs(x);
end

function r = f2(x)
    r = abs(x);
end

function r = f3(x)
    r = abs(x);
```

```
end

function r = pos(x)
    r = max(x,0);
end
```

Partial Truncation

```
function [final_avg_price, RMSE, std_err, bias,
   runtime] = partial_trunc(paths, steps_per_year,
   omega)
% Define function to calculate option price based on
   the Heston model
% modified by partial tuncation method. Function of
   number of paths, steps
% per year and value of omega. Return option price,
   RMSE standard
% error bias and runtime.

    % Set problem parameters and initial conditions
    S0 = 100*ones(paths,1);
    K = 100;
    T = 5;
    r = 0.05;
    V0 = 0.09*ones(paths,1);
    theta = 0.09;
    kappa = 2;
    % omega = 1;
    rho = -0.3;

    % Set step size for time and create vectors of
       volatilities/ log
    % stockprices to be populated.
    dt = 1/steps_per_year;
    M = 1/dt;
    [V,Vtil,logS] = deal(cell(1,T*(M)));

    % Define avg option price vector to be populated
    avg_option_price_vec = deal(zeros(1,100));

    % Define runtime vector to be populated
    timevec = zeros(1,100);

    % Repeat calculation over all time steps 100 times
        to calc avg
    for j = 1:100
```

```
tic

% Define each random time step using scaled
   standard normal random
% variables and using Cholesky decomposition
   in two dimensions to
% ensure the dWs is correlated
dWv = sqrt(dt)*normrnd(0,1,[paths,T*(M)]);
dWs = rho*dWv + sqrt(1-rho^2)*normrnd(0,1,[
   paths,T*(M)])*sqrt(dt);

% Iterate through each timestep
for i = 0:T*(M)-1

    % Input initial conditions into Heston
       model
    if i == 0
        Vtil{i+1} = f1(V0) - kappa.*dt.*(f2(V0
           )-theta) + omega.*sqrt(f3(V0)).*(
           dWv(:,i+1));
        V{i+1} = f3(Vtil{i+1});
        logS{i+1} = log(S0) + (r-(1/2).*V0).*
           dt + sqrt(V0).*dWs(:,i+1);

    % Iterate ith time step in discretised
       Heston model
    else
        Vtil{i+1} = f1(Vtil{i}) - kappa.*dt.*(
           f2(Vtil{i})-theta) + omega.*sqrt(f3
           (Vtil{i})).*(dWv(:,i+1));
        V{i+1} = f3(Vtil{i+1});
        logS{i+1} = logS{i} + (r-(1/2).*V{i})
           .*dt + sqrt(V{i}).*dWs(:,i+1);
    end
end

% Calculate vector of option price for each
   path using the final
% value of the stock for each path
option_price = pos(exp(logS{length(logS)})-K);

% Calculate average over all paths of option
   prices
avg_option_price_vec(j) = exp(-0.05*5)*mean(
   option_price)
```

```
        % Record runtime for this simulation
        timevec(j) = toc;
    end

    % Calculate average over 100 simulations of option
        price, std error,
    % bias, RMSE and runtime
    final_avg_price = mean(avg_option_price_vec)
    std_err = sqrt(mean((avg_option_price_vec-
       final_avg_price).^2))
    bias = abs(final_avg_price-34.9998)
    RMSE = sqrt(bias^2+std_err^2)
    runtime = mean(timevec)
end

% Define required f1, f2, and f3 for partial
   truncation method

function r = f1(x)
    r = x;
end

function r = f2(x)
    r = x;
end

function r = f3(x)
    r = max(x,0);
end

function r = pos(x)
    r = max(x,0);
end
```

Full Truncation

```
function [final_avg_price, RMSE, std_err, bias,
   runtime] = full_trunc(paths, steps_per_year, omega)
% Define function to calculate option price based on
   the Heston model
% modified by full truncation method. Function of
   number of paths, steps
% per year and value of omega. Return option price,
   RMSE standard
% error bias and runtime.
```

```
% Set problem parameters and initial conditions
S0 = 100*ones(paths,1);
K = 100;
T = 5;
r = 0.05;
V0 = 0.09*ones(paths,1);
theta = 0.09;
kappa = 2;
% omega = 1;
rho = -0.3;

% Set step size for time and create vectors of
   volatilities/ log
% stockprices to be populated.
dt = 1/steps_per_year;
M = 1/dt;
[V,Vtil,logS] = deal(cell(1,T*(M)));

% Define avg option price vector to be populated
avg_option_price_vec = deal(zeros(1,100));

% Define runtime vector to be populated
timevec = zeros(1,100);

% Repeat calculation over all time steps 100 times
    to calc avg
for j = 1:100
    tic

    % Define each random time step using scaled
       standard normal random
    % variables and using Cholesky decomposition
       in two dimensions to
    % ensure the dWs is correlated
    dWv = sqrt(dt)*normrnd(0,1,[paths,T*(M)]);
    dWs = rho*dWv + sqrt(1-rho^2)*normrnd(0,1,[
       paths,T*(M)])*sqrt(dt);

    % Iterate through each timestep
    for i = 0:T*(M)-1

        % Input initial conditions into Heston
           model
        if i == 0
            Vtil{i+1} = f1(V0) - kappa.*dt.*(f2(V0
               )-theta) + omega.*sqrt(f3(V0)).*(
```

```
                dWv(:,i+1));
            V{i+1} = f3(Vtil{i+1});
            logS{i+1} = log(S0) + (r-(1/2).*V0).*
                dt + sqrt(V0).*dWs(:,i+1);

        % Iterate ith time step in discretised
            Heston model
        else
            Vtil{i+1} = f1(Vtil{i}) - kappa.*dt.*(
                f2(Vtil{i})-theta) + omega.*sqrt(f3
                (Vtil{i})).*(dWv(:,i+1));
            V{i+1} = f3(Vtil{i+1});
            logS{i+1} = logS{i} + (r-(1/2).*V{i})
                .*dt + sqrt(V{i}).*dWs(:,i+1);
        end
    end

    % Calculate vector of option price for each
        path using the final
    % value of the stock for each path
    option_price = pos(exp(logS{length(logS)})-K);

    % Calculate average over all paths of option
        prices
    avg_option_price_vec(j) = exp(-0.05*5)*mean(
        option_price)

    % Record runtime for this simulation
    timevec(j) = toc;
end

% Calculate average over 100 simulations of option
     price, std error,
% bias, RMSE and runtime
final_avg_price = mean(avg_option_price_vec)
std_err = sqrt(mean((avg_option_price_vec-
    final_avg_price).^2))
bias = abs(final_avg_price-34.9998)
RMSE = sqrt(bias^2+std_err^2)
runtime = mean(timevec)
end

% Define required f1, f2, and f3 for full truncation
    method

function r = f1(x)
```

```
    r = x;
end

function r = f2(x)
    r = max(x,0);
end

function r = f3(x)
    r = max(x,0);
end

function r = pos(x)
    r = max(x,0);
end
```

Table Results

```
    % Generate scheme results as result_matrix that
       mirrors format given in
    % the assignment scheme. Omega is set to 1, for
       Question 5, change
    % third out of each function from 1 to 0.3

    % Call absorption function and return results for
       the three set of
    % paths and numbers of timesteps
    [final_avg_price_1abs, RMSE_1abs, std_err_1abs,
       bias_1abs, runtime_1abs] = absorption(10000,
       20, 1);
    [final_avg_price_2abs, RMSE_2abs, std_err_2abs,
       bias_2abs, runtime_2abs] = absorption(40000,
       40, 1);
    [final_avg_price_3abs, RMSE_3abs, std_err_3abs,
       bias_3abs, runtime_3abs] = absorption(160000,
       80, 1);

    % Call reflection function and return results for
       the three set of
    % paths and numbers of timesteps
    [final_avg_price_1ref, RMSE_1ref, std_err_1ref,
       bias_1ref, runtime_1ref] = reflection(10000,
       20, 1);
    [final_avg_price_2ref, RMSE_2ref, std_err_2ref,
       bias_2ref, runtime_2ref] = reflection(40000,
       40, 1);
    [final_avg_price_3ref, RMSE_3ref, std_err_3ref,
```

```
    bias_3ref, runtime_3ref] = reflection(160000,
    80, 1);

% Call partial truncation function and return
    results for the three set
% of paths and numbers of timesteps
[final_avg_price_1part, RMSE_1part, std_err_1part,
     bias_1part, runtime_1part] = partial_trunc
    (10000, 20, 1);
[final_avg_price_2part, RMSE_2part, std_err_2part,
     bias_2part, runtime_2part] = partial_trunc
    (40000, 40, 1);
[final_avg_price_3part, RMSE_3part, std_err_3part,
     bias_3part, runtime_3part] = partial_trunc
    (160000, 80, 1);

% Call full truncation function and return results
     for the three set
% of paths and numbers of timesteps
[final_avg_price_1full, RMSE_1full, std_err_1full,
     bias_1full, runtime_1full] = full_trunc(10000,
     20, 1);
[final_avg_price_2full, RMSE_2full, std_err_2full,
     bias_2full, runtime_2full] = full_trunc(40000,
     40, 1);
[final_avg_price_3full, RMSE_3full, std_err_3full,
     bias_3full, runtime_3full] = full_trunc
    (160000, 80, 1);

% Generate result_matrix from returned results
    that mirrors format
% given in the assignment specification
result_matrix = [bias_1abs, bias_2abs, bias_3abs
    ;...
                 std_err_1abs, std_err_2abs,
                    std_err_3abs;...
                 RMSE_1abs, RMSE_2abs, RMSE_3abs
                    ;...
                 runtime_1abs, runtime_2abs,
                    runtime_3abs;...
                 bias_1ref, bias_2ref, bias_3ref
                    ;...
                 std_err_1ref, std_err_2ref,
                    std_err_3ref;...
                 RMSE_1ref, RMSE_2ref, RMSE_3ref
                    ;...
```

```
                              runtime_1ref , runtime_2ref ,
                                 runtime_3ref ;...
                              bias_1part , bias_2part ,
                                 bias_3part ;...
                              std_err_1part , std_err_2part ,
                                 std_err_3part ;...
                              RMSE_1part , RMSE_2part ,
                                 RMSE_3part ;...
                              runtime_1part , runtime_2part ,
                                 runtime_3part ;...
                              bias_1full , bias_2full ,
                                 bias_3full ;...
                              std_err_1full , std_err_2full ,
                                 std_err_3full ;...
                              RMSE_1full , RMSE_2full ,
                                 RMSE_3full ;...
                              runtime_1full , runtime_2full ,
                                 runtime_3full ];
%     toc
```

Delta and Gamma Comparison

```
% Num of paths to simulate
paths = 100000;

% Set problem parameters and initial conditions
S0 = 100* ones (paths ,1);
S0u = 101* ones (paths ,1);
S0d = 99* ones (paths ,1);
K = 100;
T = 5;
r = 0.05;
V0 = 0.09* ones (paths ,1);
theta = 0.09;
kappa = 2;
omega = 1;
rho = -0.3;

% Set step size for time and create vectors of
   volatilities/ log
% stockprices to be populated.
dt = 1/50;
M = 1/dt;
[V,Vtil,logS,logSu,logSd] = deal(cell(1,T*(M)));

% Define avg option price vector to be populated
```

```
avg_option_price_vec = deal(zeros(1,100));

% Define delta and gamma vectors to be populated
delta_vecs = zeros(100,T*(M)-1);
gamma_vecs = zeros(100,T*(M)-1);

final_delta = zeros(1,T*(M)-1);
final_gamma = zeros(1,T*(M)-1);

tic
% Repeat calculation over all time steps 100 times to
   calc avg
for j = 1:100
    % Define each random time step using scaled
       standard normal random
    % variables and using Cholesky decomposition in
       two dimensions to
    % ensure the dWs is correlated
    dWv = sqrt(dt)*normrnd(0,1,[paths,T*(M)]);
    dWs = rho*dWv + sqrt(1-rho^2)*normrnd(0,1,[paths,T
       *(M)])*sqrt(dt);

    % Iterate through each timestep
    for i = 0:T*(M)-1
        % Input initial conditions into Heston model
           also calculate
        % values for initial higher and lower stock
           price
        if i == 0
            Vtil{i+1} = f1(V0) - kappa.*dt.*(f2(V0)-
               theta) + omega.*sqrt(f3(V0)).*(dWv(:,i
               +1));
            V{i+1} = f3(Vtil{i+1});
            logS{i+1} = log(S0) + (r-(1/2).*V0).*dt +
               sqrt(V0).*dWs(:,i+1);
            logSu{i+1} = log(S0u) + (r-(1/2).*V0).*dt
               + sqrt(V0).*dWs(:,i+1);
            logSd{i+1} = log(S0d) + (r-(1/2).*V0).*dt
               + sqrt(V0).*dWs(:,i+1);

        % Iterate ith time step in discretised Heston
           model also calculate
        % values for initial higher and lower stock
           price
        else
            Vtil{i+1} = f1(Vtil{i}) - kappa.*dt.*(f2(
```

```
                Vtil{i})-theta) + omega.*sqrt(f3(Vtil{i
                }).*(dWv(:,i+1));
            V{i+1} = f3(Vtil{i+1});
            logS{i+1} = logS{i} + (r-(1/2).*V{i}).*dt
                + sqrt(V{i}).*dWs(:,i+1);
            logSu{i+1} = logSu{i} + (r-(1/2).*V{i}).*
                dt + sqrt(V{i}).*dWs(:,i+1);
            logSd{i+1} = logSd{i} + (r-(1/2).*V{i}).*
                dt + sqrt(V{i}).*dWs(:,i+1);

        end
    end

    % Calculate delta and gamma at each time step
       using the finite
    % difference scheme
    for k=0:(M)*T-1
        F = exp(-0.05*(T-k*dt))*f2(exp(logS{(M)*T-k})-
           K);
        FU = exp(-0.05*(T-k*dt))*f2(exp(logSu{(M)*T-k
           })-K);
        FD = exp(-0.05*(T-k*dt))*f2(exp(logSd{(M)*T-k
           })-K);

        delta_vecs(j,(k+1)) = (1/2)*(mean(FU)-mean(FD)
           );
        gamma_vecs(j,(k+1)) = mean(FU + FD - 2*F);
    end
end

% Calculate average of delta and gamma at each time
   step by taking an
% average over the 100 simulations at each time step
for j = 1:M*T
    final_delta(j) = mean(delta_vecs(:,j));
    final_gamma(j) = mean(gamma_vecs(:,j));
end

toc

% Plot final values of delta for Heston model and
   compare with plotted
% delta for Black-Scholes
figure(1)
hold on
plot(final_delta)
```

```
plot(blsdelta(100,100,0.05,T-[0:dt:T-dt],0.3))
legend('Heston','BS')
hold off

% Plot final values of gamma for Heston model and
   compare with plotted
% gamma for Black-Scholes
figure(2)
hold on
plot(final_gamma)
plot(blsgamma(100,100,0.05,T-[0:dt:T-dt],0.3))
legend('Heston','BS')
hold off

% Define required f1, f2, and f3 for full truncation
   method

function r = f1(x)
    r = x;
end

function r = f2(x)
    r = max(x,0);
end

function r = f3(x)
    r = max(x,0);
end
```