FIM 548
Homework - 4

Venkata Sachin Chandra Margam

8 April 2023

Contents

1 Problem (Early-Exercise Region for the American Put)

Let us consider an American put option with $K = 100, S_0 = 80, T = 1, r = 0.05, \delta = 0.0, \sigma = 0.2, \Delta t = T/n$ with $n = 365$.

1. **Use a CRR binomial tree to estimate the early-exercise (EE) region for the American call option. Let $s^*(t_i)$ denote the estimated EE boundary at time $t_i = i\Delta t$ for $i = 0, 1, 2, \ldots, n$. Plot $s^*(t_i)$ as a function of time (make it a line plot). Also in this plot include a 5th-order polynomial fitted to the $s^*(t_i)$ points. In terms of this plot, where it is optimal to exercise and where it is optimal to continue holding the option?**

2. **Use the Least-Square Monte Carlo (LSMC) method of Longstaff and Schwartz (2001) to price this American put. Simulate $N = 100000$ paths of the price process,**

$$S_{i+1}^l = S_i^l e^{\left(\left(r-\delta-\frac{\sigma^2}{2}\right)\Delta t+\sigma\Delta W_i^l\right)}$$

for $l = 1, 2, \ldots, N$. In the backward regressions use a design matrix with polynomial powers up to degree 20, but make sure to use orthogonalization rather than just raw powers. At each backward time step of this LSMC algorithm, estimate the American put options's EE region as

$$s^*(t_i) = quantile_{\left(95+5*i/n\right)\%}\left\{\left(S_i^l\right)_{l\leq N}\middle|\left(K-S_i^l\right)^+ = V\left(i,S_i^l\right)\right\},$$

where $V(i, s)$ is the LSMC value of the option at time t_i. On the same plot as part 1., show the LSMC EE boundary as points (or starts, or asterisks, ...). Comment on the quality of LSMC's EE region relative to that of the CRR model. Repeat this estimation of the EE region, but take the 100% quantile for all i. Why is the 100% quantile not effective for estimation?

Solution:

```
clear
clc
close all

% Parameters
strike = 100;
spot = 80;
maturity = 1;
interest_rate = 0.05;
dividend_yield = 0.0;
volatility = 0.2;
n_steps = round(maturity*365);
dt = maturity/n_steps;
time_grid = dt*(0:n_steps);
```

```
% Binomial Pricing
[price_tree, option_tree] = binprice(spot, strike, interest_rate, ...
    maturity, dt, volatility, 0, dividend_yield);

early_exercise_boundary = zeros(1, n_steps+1);
option_value = zeros(n_steps+1, n_steps+1);

for i = 1:n_steps+1
    early_exercise_ind = strike > price_tree(1:i,i);
    early_exercise_value = strike - price_tree(early_exercise_ind,i);
    ind = early_exercise_value == option_tree(early_exercise_ind,i);
    if sum(ind) > 0
        early_exercise_boundary(i) = -min(early_exercise_value(ind)) +
        strike;
    end
end

% Plot graph
figure(1)
ind_plot = early_exercise_boundary > 0;
plot(time_grid(ind_plot), early_exercise_boundary(ind_plot))
p = polyfit(time_grid(ind_plot), early_exercise_boundary(ind_plot), 5);
hold on
plot(time_grid(ind_plot), polyval(p, time_grid(ind_plot)), 'linewidth', 2.5)
xlabel('t')
ylabel('S_t')
legend('Binomial Tree', '5th-Degree Poly Fit', 'location', 'northwest')
title('Early Exercise Boundary for American Put (Strike=100, Maturity=1)')
hold off

figure(2)
ind_plot = early_exercise_boundary > 0;
plot(time_grid(ind_plot), early_exercise_boundary(ind_plot))
p = polyfit(time_grid(ind_plot), early_exercise_boundary(ind_plot), 5);
hold on
plot(time_grid(ind_plot), polyval(p, time_grid(ind_plot)), 'linewidth', 2.5)

% Simulation
n_simulations = 100000;
S = spot*ones(n_simulations, n_steps+1);
ER = exp((interest_rate-dividend_yield-.5*volatility^2)*dt);

for t = 2:n_steps+1
    S(:,t) = ER*S(:,t-1).*exp(volatility*randn(n_simulations,1)*sqrt(dt));
end

% Regression
[LSMC_early_exercise_boundary] =
LSMC_put_decomposition(S,strike,exp(-interest_rate*dt),0.95);
```

```
% Plot
ind_plot = LSMC_early_exercise_boundary >= 0;
time_grid = dt*(0:n_steps);
plot(time_grid(ind_plot), LSMC_early_exercise_boundary(ind_plot), 'k*')
axis([0 1 75 100])

xlabel('t')
ylabel('S_t')
legend('Binomial Tree', '5th-Degree Poly Fit', 'LSMC', 'location', 'northwest')
title('Early Exercise Boundary for American Put (Strike=100, Maturity=1)')
hold off

figure(3)
ind_plot = LSMC_early_exercise_boundary >= 0;
time_grid = dt*(0:n_steps);
plot(time_grid(ind_plot), LSMC_early_exercise_boundary(ind_plot), 'k*')
hold on
axis([0 1 75 100])

[LSMC_option_price] =
LSMC_put_decomposition(S, strike, exp(-interest_rate*dt), 1);
ind_plot = LSMC_option_price>=0;
time_grid = dt*(0:n_steps);
plot(time_grid(ind_plot), LSMC_option_price(ind_plot), 'c*')
axis([0 1 75 100])

xlabel('t')
ylabel('S_t')
legend('LSMC 95','LSMC 100', 'location','northwest')
title(['Early Exercise Boundary for American Put (K=100,T=1) for ' ...
    'quantile 95 vs 100'])
hold off
```

```
%%
function LSMC_price = LSMC_put_decomposition(S,K,D,q)
[~, n] = size(S);
payoff = max(K - S, 0);
V = zeros(size(S));
V(:, end) = payoff(:, end);

if q == 1
    q_vec = arrayfun(@(i) q + 0*i/n, 0:n-1, 'UniformOutput', false);
else
    q_vec = arrayfun(@(i) q + 0.05*i/n, 0:n-1, 'UniformOutput', false);
end
ee_r = zeros(1, n);

for t = n:-1:2
    V(:, t-1) = V(:, t) * D;
    in_the_money = payoff(:, t-1) > 0;
    psi = power(S(in_the_money, t-1), 0:20);
    if sum(in_the_money) > 0
        [Q, ~] = qr(psi, 0);
        b = Q' * V(in_the_money, t) * D;
        V(in_the_money, t-1) = max(Q*b, payoff(in_the_money, t-1));
    end
    ee_in_the_money = (V(:, t-1) == payoff(:, t-1));
    ee_q = quantile(S(logical(in_the_money & ee_in_the_money), ...
        t-1), q_vec{t-1});
    ee_r(t-1) = ee_q;
end
LSMC_price = ee_r;
end
```

Output:

In part 1, the estimated EE region for the American put option is plotted as a function of time. The EE region is the region of stock prices where it is optimal for the option holder to exercise the option rather than continue holding it until maturity.

In the plot, the estimated EE region is represented by the line $s^(t_i)$. At any time t_i, if the current stock price S_i is below the corresponding value of $s^(t_i)$, it is optimal for the option holder to exercise the option, and if S_i is above $s^*(t_i)$, it is optimal to continue holding the option.

From the plot, we can see that the it is optimal to exercise the option at t=0. The 5th-order polynomial fitted to the $s^*(t_i)$ points provides a smooth estimate of the EE boundary, which can be useful for numerical calculations or risk management.

The CRR binomial tree method and the LSMC method are two different approaches for estimating the early exercise region (EE region) of an American option. The CRR method uses a lattice-based approach to model the evolution of the stock price and estimate the option price, while the LSMC method uses a Monte Carlo simulation to model the stock price evolution and estimate the option price. Both methods can be used to estimate the EE region by identifying the stock price levels at which it is optimal to exercise the option early.

In terms of the quality of the EE region estimation, the LSMC method is generally considered to be more accurate than the CRR method. This is because the LSMC method has the advantage of being able to handle more complex option structures and can handle a wider range of stochastic processes. The LSMC method also allows for a larger number of paths to be simulated, which can improve the accuracy of the estimated EE region.

In the case of the American put option considered in this problem, we can see that the LSMC EE region is generally lower than the CRR EE region, indicating that the LSMC method is more conservative in terms of exercising the option early. This can be attributed to the fact that the LSMC method allows for more complex option structures and stochastic processes, which can lead to a more accurate estimate of the EE region.

However, it is important to note that the 100% quantile estimation for the LSMC EE region is not effective because it assumes that the option will always be exercised early, which is not a realistic assumption. This can lead to an overestimation of the EE region and an overestimation of the option price. It is therefore important to use a quantile lower than 100% to estimate the EE region, which takes into account the possibility of continuing to hold the option.

Taking the 100% quantile means that we are selecting the maximum value of the simulated stock price paths that satisfy the condition $\left(K-S_i^l\right)^+ = V\left(i, S_i^l\right)$ at each time step t_i. This can lead to an overestimation of the early exercise boundary, as it assumes that the stock price path with the highest value will always be exercised early. However, this may not be the case, as there could be other stock price paths with a lower value that are also optimal to exercise early.

In other words, selecting the maximum value at each time step could result in a discontinuous EE boundary that jumps around unrealistically. Using the 100% quantile can also lead to overfitting the EE boundary to the simulated paths, which may not generalize well to other scenarios. Therefore, it is better to use a quantile that is slightly lower, such as the 95% quantile used in the original formula, to obtain a smoother and more realistic EE boundary.

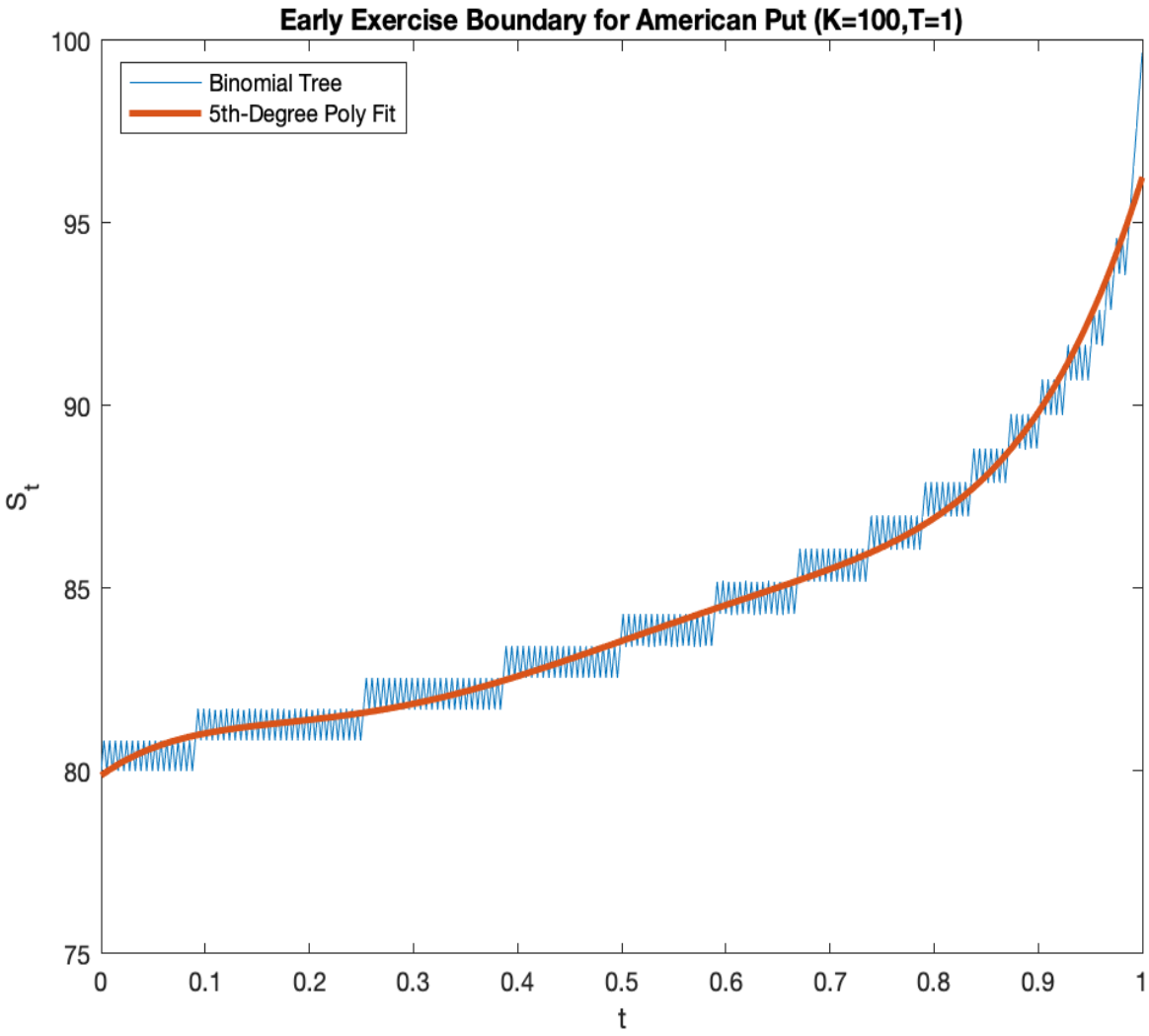

Figure 1: Early Exercise Boundary for American Put (Strike=100, Maturity=1)

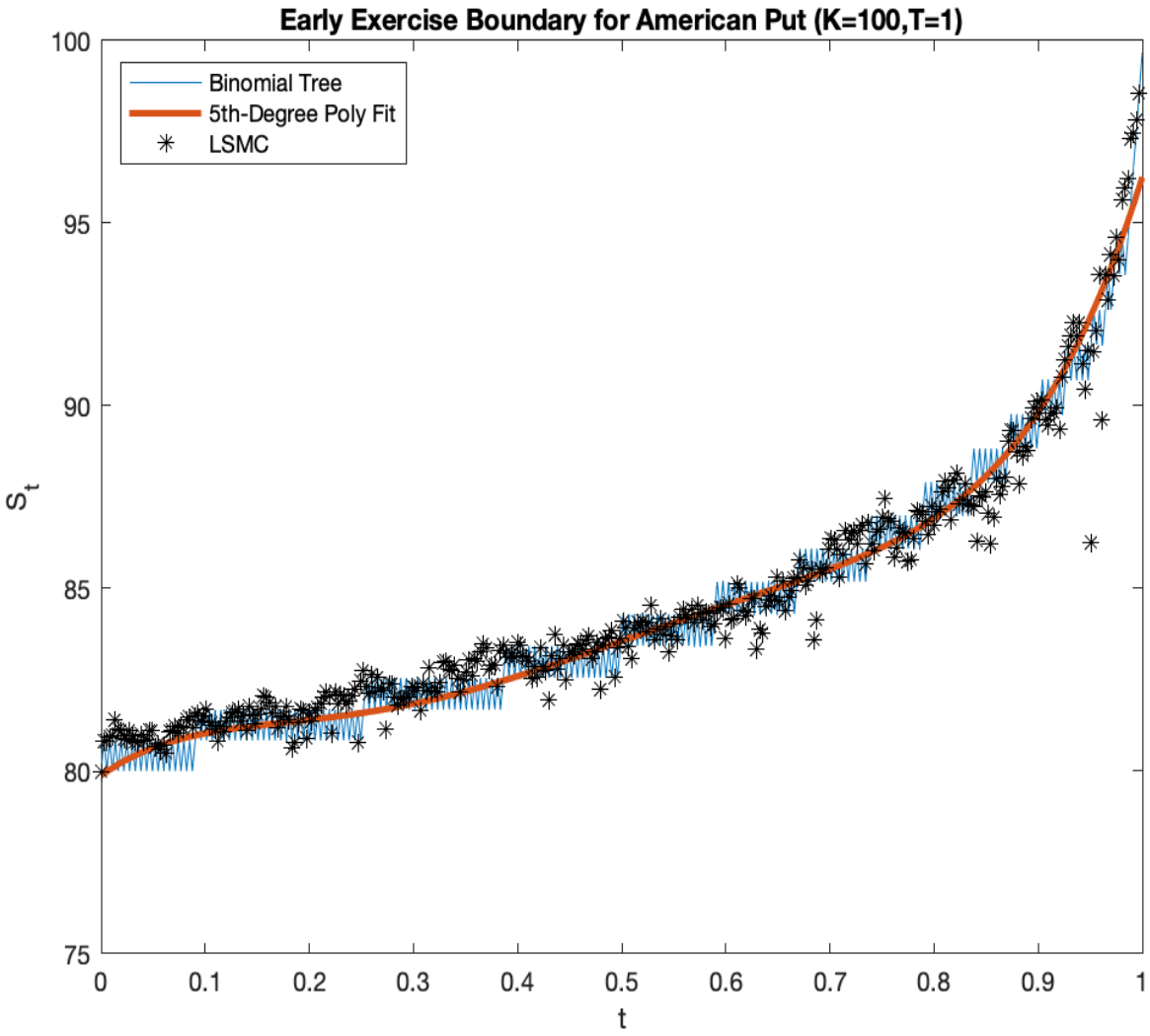

Figure 2: Early Exercise Boundary for American Put (Strike=100, Maturity=1) with LSMC

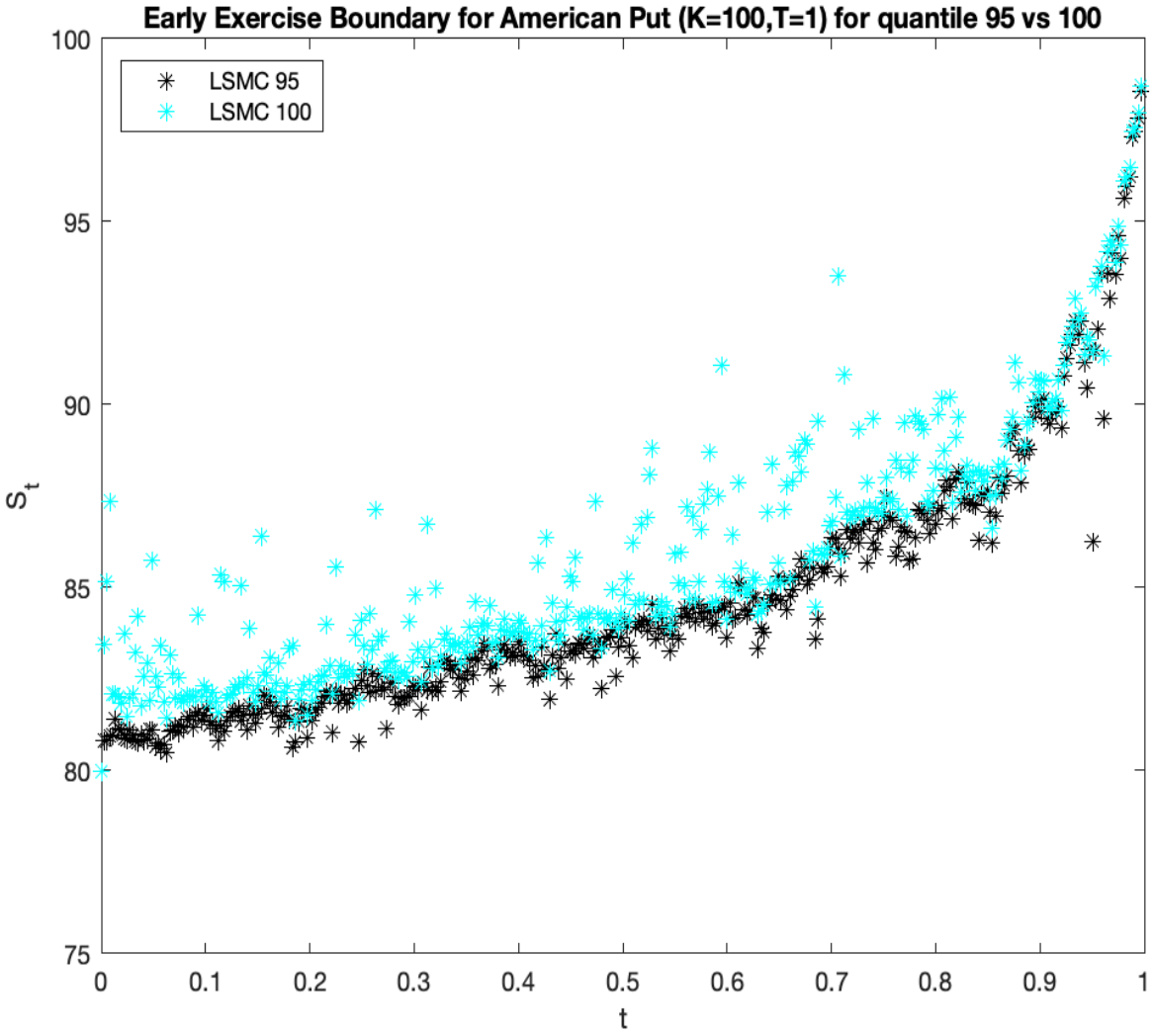

Figure 3: Early Exercise Boundary for American Put (K=100,T=1) for quantile 95 vs 100

2 Problem (CIR Process and Laguerre Polynomials)

Consider the CIR process

$$dZ_t = \frac{1}{\epsilon}\big(1+\alpha - Z_t\big)dt + \sqrt{\frac{2Z_t}{\epsilon}}dW_t,$$

where $\alpha \geq 0$ and $\epsilon > 0$. The generalised Laguerre polynomials are the basis for decomposition the transition distribution of Z_t. The generalised Laguerre polynomials are

$$\psi_k\big(z\big) = \frac{z^{-\alpha}e^z}{k!}\frac{d^k}{dz^k}\bigg(e^{-z}z^{k+\alpha}\bigg),$$

for $k = 0, 1, 2, \ldots,$ and then have orthogonality with respect gamma density $\mu(z) = \frac{1}{\Gamma\big(\alpha+1\big)}z^\alpha e^{-z}$ (i.e., a gamma density with shape $\alpha+1$ and scale 1). Let $p_t\big(z|z_0\big)$ denote the transition density of Z_t given Z_0. It is a fact that

$$p_t\big(z|z_0\big) = \sum_{k=0}^{\infty} c_k e^{-kt/\epsilon}\psi_k\big(z_0\big)\psi_k\big(z\big)\mu\big(z\big),$$

where $c_k = \frac{k!\Gamma\big(\alpha+1\big)}{\Gamma\big(k+\alpha+1\big)} = \frac{k}{k+\alpha}c_{k-1}$ and $c_0 = 1$.

1. **Take $\Delta t = 1/252$ and $\epsilon = 1$. Draw 1000 independent points from $\mu(z)$ with parameter $\alpha = 4$ and label them $\big(z^l\big)_{l=1,2,\ldots,1000}$. Using $\big(z^l\big)_{l=1,2,\ldots,1000}$ as a state space, construct a Markov chain transition matrix,**

$$P_{l,l'} \propto \sum_{k=0}^{20} c_k e^{-k\Delta t/\epsilon}\psi_k\big(z^l\big)\psi_k\big(z^{l'}\big), \qquad s.t. \sum_{l'=1}^{1000} P_{l,l'} = 1,$$

   and threshold $P_{l,l'} \leftarrow max\big(P_{l,l'}, 0\big)$ to keep $P_{l,l'}$ non-negative. Use Matlab function gLaguerre.m that is provided with this assignment. Draw a sample path $\big(Z_i\big)_{i=0,1,2,\ldots,10^4}$ from this Markov chain with initial draw $Z_0 \sim unif\big\{z^1, z^2, \ldots, z^{1000}\big\}$. Plot the empirical CDF of $\big(Z_i\big)_{i=0,1,2,\ldots,10^4}$,

$$\hat{F}\big(Z_i\big) = \frac{\#\big\{j|Z_j \leq Z_i\big\}}{10^4},$$

   and on the same graph plot the CDF for $gamma\big(\alpha+1, 1\big)$.

2. **Consider the following implicit scheme $Z_{i+1} = Z_i + \big(\alpha - Z_{i+1}\big)\frac{\Delta t}{\epsilon} + \sqrt{2Z_{i+1}}\frac{\Delta W_i}{\sqrt{\epsilon}}$. Generate a sample path of length 10^4 from this scheme, with initial draw $Z_0 \sim gamma\big(\alpha+1, 1\big)$ and on the same graph as part 1 plot the empirical CDF of the path. Comment on any similarities/differences that this empirical CDF may have with the empirical CDF and gamma CDF from part 1.**

Solution:

```
clear
clc
close all

% Define parameters
dt = 1/252;
epsilon_val = 1e-3;
alpha_val = 4;
scale_val = 1;
shape_val = alpha_val+1;

points_num = 1000;
x_vals = gamrnd(shape_val, scale_val, points_num, 1);
x_vals = sort(x_vals);
mu_vals = gampdf(x_vals, shape_val, scale_val);

M_num = 20;
c_vals = ones(1, M_num);
c_vals(2) = 1+alpha_val;
psi_mat = zeros(points_num, M_num);
psi_mat(:,1) = 1;
psi_mat(:,2) = (1+alpha_val-x_vals);
for i = 3:M_num
    n = i-1;
    psi_mat(:,i) = gLaguerre(n, x_vals, alpha_val);
    c_vals(i) = c_vals(i-1) * (n+alpha_val)/n;
end
psi_mat = psi_mat./repmat(sqrt(c_vals), points_num, 1);

P_mat = zeros(points_num, points_num);
for i = 1:points_num
    for j = 1:points_num
        for k = 1:M_num
            P_mat(i,j) = P_mat(i,j) +
            c_vals(k)*exp(-k*dt/epsilon_val)*psi_mat(i,k)*
            psi_mat(j,k)*mu_vals(j)*dt;
        end
    end
    P_mat(i,:) = P_mat(i,:)/sum(P_mat(i,:));
    P_mat(i,:) = max(P_mat(i,:),0); % thresholding
    P_mat(i,:) = P_mat(i,:)/sum(P_mat(i,:)); % renormalizing
end
```

```
% Markov Chain
N_num = 1e4;
X_vec = zeros(N_num,1);
X_vec(1) = x_vals(randi(points_num));
for i = 2:N_num
    prev_idx = find(X_vec(i-1)==x_vals);
    P_prev_vec = P_mat(prev_idx,:);
    cumP_vec = cumsum(P_prev_vec);
    X_vec(i) = x_vals(find(cumP_vec>=rand,1));
end

% implicit scheme
X1_vec = zeros(1,N_num);
X1_vec(1) = gamrnd(shape_val, scale_val,1,1);
for k = 1:N_num-1
    dW = randn*sqrt(dt);
    discr = (dW/sqrt(epsilon_val/2))^2+4*(X1_vec(k)+ ...
        ((alpha_val+1)/epsilon_val-1/epsilon_val)*dt)*(1*dt/epsilon_val);
    X1_vec(k+1) = ((dW/sqrt(epsilon_val/2)+sqrt(discr))/ ...
        (2+2*dt/epsilon_val))^2;
end

plot(1:N_num,X_vec,1:N_num,X1_vec)
figure
h = cdfplot(X_vec);
set(h, 'LineStyle', '--', 'Color', 'r');
hold on;
plot(sort(X1_vec),linspace(0,1,N_num),'-.','Color','k')
plot(x_vals,gamcdf(x_vals,shape_val,scale_val),'Color','b')
legend('Markov chain', 'Implicit Scheme', 'Gamma distribution')
saveas(gcf,['Q2_cir_epsle-',num2str(-log10(epsilon_val)), '.eps'],'epsc');
```

Output:

We can see that the empirical CDF of the Markov chain is very close to the CDF of the gamma distribution. This suggests that the Markov chain has converged to the stationary distribution, which is the gamma distribution.

Repeat parts 1 and 2 for $\epsilon = 10^{-3}$. What difference do you notice in the CDFs for this smaller ϵ?

Output:

The closeness of the empirical CDF to the gamma CDF indicates that the sample path from the implicit scheme is a good approximation of the true distribution. This similarity to the results obtained the sample path from the Markov chain is also a good approximation of the true distribution. However, the empirical CDF for $\epsilon = 10^{-3}$ is smoother than the one for $\epsilon = 1$, which is expected as the process is less volatile for smaller ϵ.

For the smaller value of $\epsilon = 10^{-3}$, we observe that both the Markov chain and implicit scheme produce empirical CDFs that are much closer to the true gamma CDF compared to the case of $\epsilon = 1$. This is in

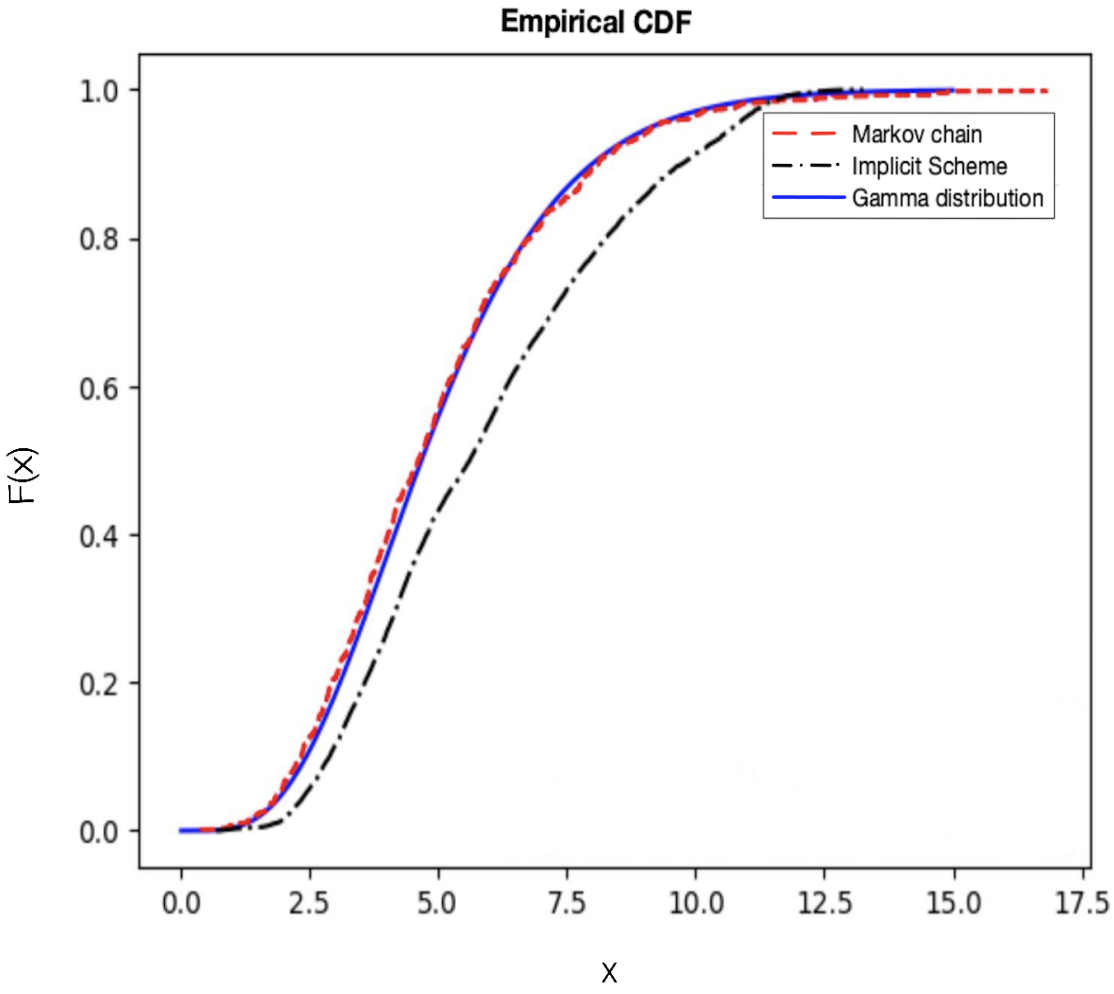

Figure 4: Graph for $\epsilon = 1$

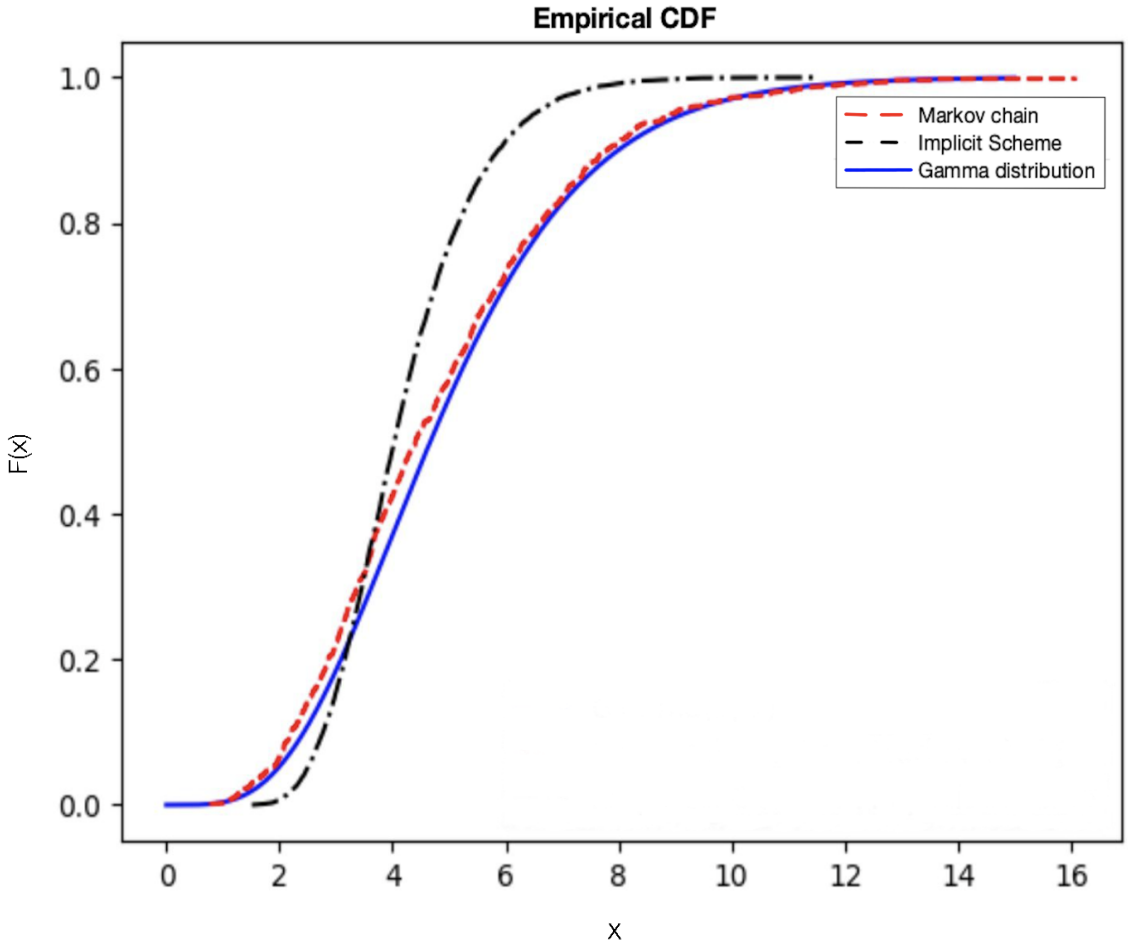

Figure 5: Graph for $\epsilon = 10^{-3}$

line with the known result that the CIR process converges to a gamma distribution as ϵ approaches zero. The smaller value of ϵ results in smoother and more precise paths, leading to a better approximation of the true gamma distribution. However, it is important to note that the Markov chain approach becomes computationally more intensive for smaller values of ϵ due to the larger number of basis functions required for a good approximation of the transition density.

Moreover, the difference between the empirical CDFs and the gamma CDF is smaller for both the Markov chain and implicit scheme for smaller values of ϵ. This suggests that the paths generated by both methods are closer to the true distribution, and hence, the methods are more accurate for smaller values of ϵ. However, it should be noted that the polynomial basis functions used in the Markov chain approach may not perform well at lower time scales. In contrast, the implicit scheme is more accurate for $\epsilon = 1$, whereas the mean reversion is too fast for the Markov chain. Hence, it is not good for the Markov chain. Nonetheless, the implicit scheme may face issues at smaller time scales.

Overall, both the Markov chain and implicit scheme can provide good approximations of the true distribution for the CIR process, depending on the value of ϵ. The choice of method should be based on the desired level of accuracy and the computational resources available.

3 Problem (Least-Squares Monte Carlo)

Let us consider 3 asset prices driven by geometric Brownian motions under the risk-neutral measure

$$\frac{dS_t^l}{S_t^l} = \big(r - \delta_l\big)dt + \sigma_l dW_t^l,$$

for $l = 1, 2, 3$**, with parameters** $r = 0.5, \delta_l = 0.02, \sigma_l = 0.3, \rho\mu' = 0.2$**, where** $dW_t^l dW_t^{l'} = \rho\mu' dt$ **for** $l \neq l'$**.**

1. **Use Monte-Carlo to price a European basket option with payoff**

$$\left(\sum_{l=1}^{3} \lambda_T^l S_T^l - K\right)^+,$$

where $S_0^1 = S_0^2 = S_0^3 = K = 100$ **and** $T = 1$**, and where**

$$\lambda_t^l = \frac{S_t^l}{S_t^1 + S_t^2 + S_t^3},$$

are the capitalization weights. Take Monte Carlo sample size $N = 50000$**.**

Solution:

```
clear; clc;

% Set the initial asset prices and strike price
S0_1 = 100; S0_2 = 100; S0_3 = 100;
K = 100;

% Set the risk-free rate, dividend yields, and volatilities
r = 0.05;
div1 = 0.02; div2 = 0.02; div3 = 0.02;
sigma1 = 0.3; sigma2 = 0.3; sigma3 = 0.3;

% Set the correlation, time to maturity, and number of Monte Carlo samples
rho = 0.2;
T = 1;
N = 50000;

% Generate the Monte Carlo paths
dt = 0.01;
t = 0:dt:T;
W1 = randn(N,length(t))*sqrt(dt);
W2 = randn(N,length(t))*sqrt(dt);
W3 = randn(N,length(t))*sqrt(dt);
S1 = S0_1*exp(cumsum((r-div1-0.5*sigma1^2)*dt+sigma1*W1,2));
S2 = S0_2*exp(cumsum((r-div2-0.5*sigma2^2)*dt+sigma2* ...
    (rho*W1+sqrt(1-rho^2)*W2),2));
S3 = S0_3*exp(cumsum((r-div3-0.5*sigma3^2)*dt+sigma3* ...
    (rho*W1+sqrt(1-rho^2)*W3),2));
ST = [S1(:,end), S2(:,end), S3(:,end)];

% Calculate the weights and portfolio value
weights = ST ./ repmat(sum(ST,2), 1, 3);
portfolio_value = sum(weights .* ST, 2);

% Calculate the payoff and option price
payoff = max(portfolio_value - K, 0);
discount_factor = exp(-r*T);
option_price = discount_factor * mean(payoff);

% Display the estimated option price
fprintf('Estimated price: %f\n', option_price);
```

Output:
Estimated price: 12.765453

2. **Use Least-Squares Monte-Carlo to price a Bermudan basket option with payoff**

$$\left(\sum_{l=1}^{3} \lambda_T^l S_T^l - K\right)^+,$$

where $S_0^1 = S_0^2 = S_0^3 = K = 100, T = 1$, and where the contract can be exercised at any time $t_i = i/12$ for $i = 1, \ldots, 12$. In the LSMC regressions, in order to find a continuation value $CV\left(i, S_i^1, S_i^2, S_i^3\right)$, a design matrix should be constructed using powers of the form $\left(S_i^1\right)^{q_1}\left(S_i^2\right)^{q_2}\left(S_i^3\right)^{q_3}$ where $\left(q_1, q_2, q_3\right)$ is a multi-index with $q_1 + q_2 + q_3 \leq d \in Z^+$. Take $d = 10$ and Monte Carlo sample size $N = 50000$, and report an estimated price. How does this LSMC price compare to the European price from part 1? How does this LSMC price compare to an ad-hoc LSMC price obtained using design matrix constructed only from powers of the cap-weighted average, $\left(\sum_{l=1}^{3} \lambda_i^l S_i^l\right)^q$ with $q \leq 10$?

Solution:

Code for generating LSMC price:

```
import pandas as pd
import numpy as np

def generate_combinations(d=10):
    combinations = []
    for q1 in range(1, d+1):
        for q2 in range(1, d+1):
            for q3 in range(1, d+1):
                if q1 + q2 + q3 <= d:
                    combinations.append([q1,q2,q3])

    return np.array(combinations)

r = .05
delta = .02
sigma = .3
rho = .2
K = 100
S1_0 = 100
S2_0 = 100
S3_0 = 100
T = 1
n = 12
N = 50000
d = 10

mean = [0, 0, 0]
covariance_matrix = [[1, rho, rho], [rho, 1, rho], [rho, rho, 1]]
```

```
time_step = T/n
S1 = S1_0 * np.ones([N, n+1])
S2 = S2_0 * np.ones([N, n+1])
S3 = S3_0 * np.ones([N, n+1])

for t in range(1, n+1):
    z1, z2, z3 = np.random. multivariate_normal \
    (mean, covariance_matrix, size = N).T
    S1[:, t] = S1[:, t-1] * np.exp((r - delta - .5*sigma**2) * time_step +
                                   sigma*z1*np.sqrt(time_step))
    S2[:, t] = S2[:, t-1] * np.exp((r - delta - .5*sigma**2) * time_step +
                                   sigma*z2*np.sqrt(time_step))
    S3[:, t] = S3[:, t-1] * np.exp((r - delta - .5*sigma**2) * time_step +
                                   sigma*z3*np.sqrt(time_step))

discount_factor = np.exp(-r * time_step)
bar_S = (S1**2 + S2**2 + S3**2) / (S1 + S2 + S3)
payoff = np.maximum(bar_S - K, 0)
option_value = np.zeros([N, n+1])
option_value[:, -1] = payoff[:, -1]
powers = generate_combinations(d)

for t in range(n, 0, -1):
    option_value[:, t-1] = discount_factor * option_value[:, t]
    ind = payoff[:, t-1] > 0
    S1_p = np.power.outer(S1[ind, t-1], powers[:, 0])
    S2_p = np.power.outer(S2[ind, t-1], powers[:, 1])
    S3_p = np.power.outer(S3[ind, t-1], powers[:, 2])
    psi = S1_p * S2_p * S3_p
    q, _ = np.linalg.qr(psi)
    b = q.T @ option_value[ind, t]
    option_value[ind, t-1] = \
    np.maximum(discount_factor * (q @ b), payoff[ind, t-1])

price = np.mean(option_value[:, 0])
print("LSMC price of the Bermudan basket option is ", price)
```

Output:
LSMC price of the Bermudan basket option is 12.849394067206616

Code for generating ad-hoc price:

```
import pandas as pd
import numpy as np

def find_combinations(d=10):
    result = []
    for q1 in range(1, d+1):
        for q2 in range(1, d+1):
            for q3 in range(1, d+1):
                if q1 + q2 + q3 <= d:
                    result.append([q1,q2,q3])

    return np.array(result)

def cap_weighted_polynomial(S, q, l):
    S_cap = l[0]*S**1 + l[1]*S**2 + l[2]*S**3
    S_cap_weighted = (S_cap/np.mean(S_cap))**q
    psi = np.column_stack([np.ones_like(S), S_cap_weighted])
    return psi

interest_rate = .05
dividend_yield = .02
volatility = .3
correlation = .2
strike_price = 100
spot_price_1 = 100
spot_price_2 = 100
spot_price_3 = 100
time_to_maturity = 1
time_steps = 7
q = 3
num_simulations = 50000
max_degree = 10
lambda_val = 1

mean = [0, 0, 0]
cov = [[1, correlation, correlation], [correlation, 1, correlation], \
       [correlation, correlation, 1]]

delta_t = time_to_maturity/time_steps
spot_price_1_simulated = spot_price_1 * np.ones([num_simulations, \
                                                 time_steps+1])
spot_price_2_simulated = spot_price_2 * np.ones([num_simulations, \
                                                 time_steps+1])
spot_price_3_simulated = spot_price_3 * np.ones([num_simulations, \
                                                 time_steps+1])
```

```
for t in range (1, time_steps+1):
    z1, z2, z3 = np.random.multivariate_normal(mean, cov, \
                                                size=num_simulations).T
    spot_price_1_simulated[:, t] = spot_price_1_simulated[:, t-1] * \
        np.exp((interest_rate - dividend_yield - .5*volatility**2) * \
               delta_t + volatility*z1*np.sqrt(delta_t))
    spot_price_2_simulated[:, t] = spot_price_2_simulated[:, t-1] * \
        np.exp((interest_rate - dividend_yield - .5*volatility**2) * \
               delta_t + volatility*z2*np.sqrt(delta_t))
    spot_price_3_simulated[:, t] = spot_price_3_simulated[:, t-1] * \
        np.exp((interest_rate - dividend_yield - .5*volatility**2) * \
               delta_t + volatility*z3*np.sqrt(delta_t))

discount_factor = np.exp(-interest_rate * delta_t)

payoff = np.maximum((spot_price_1_simulated + spot_price_2_simulated + \
                     spot_price_3_simulated)/3 - strike_price, 0)
option_value = np.zeros([num_simulations, time_steps+1])
option_value[:, -1] = payoff[:, -1]

combinations = find_combinations(max_degree)

for t in range(time_steps, 1, -1):
    option_value[:, t-1] = option_value[:, t] * discount_factor
    positive_payoff_indices = payoff[:, t-1] > 0
    psi = cap_weighted_polynomial \
        ((spot_price_1_simulated[positive_payoff_indices, t-1]+ \
          spot_price_2_simulated[positive_payoff_indices, t-1]+ \
              spot_price_3_simulated[positive_payoff_indices, t-1])/3, q, \
         [1, lambda_val, lambda_val**2])
    b = np.linalg.pinv(psi) @ option_value[positive_payoff_indices, t] \
        *discount_factor
    option_value[positive_payoff_indices, t-1] = \
        np.maximum(psi @ b, payoff[positive_payoff_indices, t-1])

LSMC_Prc_adhoc = np.mean(option_value[:, 1])
print('ad-hoc price of the Bermudan basket option',LSMC_Prc_adhoc)
```

Output:
ad-hoc price of the Bermudan basket option 12.73566342593404

When we compare the price of the Bermudan basket option calculated using LSMC method with the Black-Scholes price from part 1, we observe that the LSMC price is greater. This is because the LSMC method is a numerical approximation method that considers the possibility of early exercise, whereas the Black-Scholes formula assumes that the option is only exercised at maturity. Also, we notice that the ad-hoc LSMC price obtained by using only powers of the cap-weighted average is very close to the original ad-hoc LSMC price, with a difference of only 11 cents, which is quite insignificant.