Project 8

By - Ameya Shete

In [1]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.stats import ncx2
from scipy.stats import norm
from scipy.stats import multivariate_normal as mvn
from scipy import optimize
from scipy.optimize import fsolve
from sympy.solvers import solve
from sympy import Symbol
from sympy import exp
```

Question 1) Vasicek Model

In [2]:
```
np.random.seed(12)
r0, sigma, kappa, r_mean, T = 0.05, 0.1, 0.82, 0.05, 0.5
delta_t = 1/252
```

a) Monte Carlo Simulation for the price of a Pure Discount Bond

In [13]:
```
delta_t = 1/252
N, n = 1000, int(T/delta_t)
z = np.random.normal(0, 1, size=(N, n))
r = np.zeros((N, n+1))
r[:, 0] = r0
for i in range(n):
    r[:, i+1] = r[:, i] + kappa*(r_mean - r[:, i])*delta_t + sigma*np.

price, face_value = 0, 1000
for i in range(N):
    summation = np.sum(r[i, :])*delta_t
    price += face_value*np.exp(-summation)
bond_price_a = price/N
bond_price_a
```

Out[13]: 987.3882752178886

b) Monte Carlo Simulation for the price of a Coupon Paying Bond

In [4]:
```python
# Generate interest rate dynamics
cash_flow_payments = np.arange(0.5, 4.5, 0.5)
coupon, face_value = 30, 1000
cash_flows = [coupon]*len(cash_flow_payments)
cash_flows[-1] = cash_flows[-1] + face_value
T = 4
delta_t = 1/252
N, n = 1000, int(T/delta_t)
z = np.random.normal(0, 1, size=(N, n))
r = np.zeros((N, n+1))
r[:, 0] = r0
for i in range(n):
    r[:, i+1] = r[:, i] + kappa*(r_mean - r[:, i])*delta_t + sigma*np.
r = r[:, 1:]

# Bond price calculation
price, buckets = 0, int(T/delta_t/len(cash_flows))
for i in range(N):
    for j in range(len(cash_flows)):
        summation = np.sum(r[i, :(buckets*j) + 1])*delta_t
        price += cash_flows[j] * np.exp(-summation)

bond_price = price/N
bond_price
```

Out[4]: 1084.1746381137282

c) Pricing a Call Option on a Pure Discount Bond

In [5]:
```
option_maturity, bond_maturity = 3/12, 0.5
delta_t = 1/252
N, n = 1000, int(option_maturity/delta_t)
z = np.random.normal(0, 1, (N, n+1))
r = np.zeros((N, n+1))
r[:, 0] = r0
for i in range(n):
    r[:, i+1] = r[:, i] + kappa*(r_mean - r[:, i])*delta_t + sigma*np.
r = r[:, 1:]

payoff, K, face_value = 0, 980, 1000
for i in range(N):
    s_t = bond_maturity - option_maturity
    disc_rate = np.exp(np.sum(r[i, :])*-delta_t)
    B = (1/kappa)*(1 - np.exp(-kappa*s_t))

    first_term = (r_mean - (sigma**2)/(2*kappa**2))
    second_term = B*s_t
    third_term = (sigma**2)/(4*kappa) * B**2
    A = np.exp(first_term*second_term - third_term)

    zcb_price = A*face_value*np.exp(-B*r[i][-1])
    payoff += disc_rate*np.maximum(zcb_price - K, 0)

option_price = payoff/N
option_price
```

Out[5]: 11.938755580712284

d) Pricing a Call Option on a Coupon Paying Bond using Monte Carlo

In [6]:
```
option_maturity, bond_maturity = 3/12, 4
delta_t = 1/252
N, n = 1000, int(option_maturity/delta_t)
z = np.random.normal(0, 1, (N, n))
r = np.zeros((N, n+1))
coupon, num_coupon_payments, face_value = 30, 8, 1000
cash_flows = [coupon]*num_coupon_payments
cash_flows[-1] = cash_flows[-1] + face_value

r[:, 0] = r0
for i in range(n):
    r[:, i+1] = r[:, i] + kappa*(r_mean - r[:, i])*delta_t + sigma*np.
r = r[:, 1:]

r_i = []
for i in range(N):
    summation = np.sum(r[i, :])*-delta_t
    r_i.append(np.exp(summation))

face_value, K = 1000, 980
n_next = int((bond_maturity - option_maturity)/delta_t)
total_payoff = 0
for i in range(N):
    disc_rate = r_i[i]

    r_m = np.zeros((N, n_next+1))
    r_m[:, 0] = r[i, -1]
    z_next = np.random.normal(0, 1, (N, n_next))
    for j in range(n_next):
        r_m[:, j+1] = r_m[:, j] + kappa*(r_mean - r_m[:, j])*delta_t +
    r_m = r_m[:, 1:]

    buckets = int(((bond_maturity - option_maturity)/delta_t)/len(cash
    sum_branched_out_paths = 0
    for j in range(N):
        sum_per_path = 0
        for k in range(len(cash_flows)):
            sum_per_path += cash_flows[k]*np.exp(np.sum(r_m[j, :(bucke
        sum_branched_out_paths += sum_per_path
    average_all_paths = sum_branched_out_paths/N
    payoff = disc_rate*np.maximum(0, average_all_paths - K)

    total_payoff += payoff

option_price = total_payoff/N
option_price
```

Out[6]: 102.39681684764682

e) Price of a Call option on Coupon Paying Bond using the explicit method

In [9]:
```
r0, sigma, kappa, r_mean, T = 0.05, 0.1, 0.82, 0.05, 0.25
delta_t = 1/252
coupon, face_value = 30, 1000
time = np.arange(0.5, 4.5, 0.5)
coupons = [30]*len(time)
coupons[-1] = coupons[-1] + face_value
```

In [10]:
```
def calculate_A(T, Ti, r_bar, sigma, kappa):
    B = calculate_B(T, Ti, kappa)
    A = np.exp((r_bar - (sigma**2)/(2*kappa**2)) * (B - (Ti - T)) - (s
    return A

def calculate_B(T, Ti, kappa):
    B = 1 / kappa * (1 - np.exp(-kappa * (Ti - T)))
    return B

def calculate_bond_price(T, coupon_periods, r_star, r_bar, sigma, kapp
    bond_prices = np.zeros_like(coupon_periods)
    for i, Ti in enumerate(coupon_periods):
        A = calculate_A(T, Ti, r_bar, sigma, kappa)
        B = calculate_B(T, Ti, kappa)
        bond_prices[i] = A * np.exp(-B * r_star)
    return bond_prices

def objective_function(r_star, coupon_payments, coupon_periods, T, r_b
    bond_prices = calculate_bond_price(T, coupon_periods, r_star, r_ba
    return np.sum(coupon_payments * bond_prices) - K


def solve_for_r_star(coupon_payments, coupon_periods, T, r_bar, sigma,
    r_star_initial_guess = 0.05
    r_star = fsolve(objective_function, r_star_initial_guess, args=(co

    return r_star[0]
output = solve_for_r_star(coupons, time, T, r_mean, sigma, kappa, K)
```

In [16]:

```
def K(T, Ti, r_star, sigma, kappa, r_bar):
    A = calculate_A(T, Ti, r_bar, sigma, kappa)
    B = calculate_B(T, Ti, kappa)
    bond_prices = A * np.exp(-B * r_star)
    return bond_prices
strikes = [K(T, i, output, sigma, kappa, r_mean) for i in time]

# Generate interest rate dynamics
cash_flow_payments = np.arange(0.5, 4.5, 0.5)
coupon, face_value = 30, 1000
cash_flows = [coupon]*len(cash_flow_payments)
cash_flows[-1] = cash_flows[-1] + face_value
T = 4
delta_t = 1/252
N, n = 1000, int(T/delta_t)
z = np.random.normal(0, 1, size=(N, n))
r = np.zeros((N, n+1))
r[:, 0] = r0
for i in range(n):
    r[:, i+1] = r[:, i] + kappa*(r_mean - r[:, i])*delta_t + sigma*np.
r = r[:, 1:]

# Bond price calculation
price, buckets = [], int(T/delta_t/len(cash_flows))
for i in range(N):
    for j in range(len(cash_flows)):
        summation = np.sum(r[i, :(buckets*j) + 1])*delta_t
        price += cash_flows[j] * np.exp(-summation)
```

Question 2) CIR Model

In [125]:

```
r0, sigma, kappa, r_mean = 0.05, 0.12, 0.92, 5.5/100
```

a) Pricing a Call Option on a Zero Coupon Bond using Monte Carlo

In [104]:
```
option_maturity, bond_maturity = 0.5, 1
delta_t = 1/252
N, n = 1000, int(option_maturity/delta_t)
z = np.random.normal(0, 1, (N, n))
r = np.zeros((N, n+1))
r[:, 0] = r0
for i in range(n):
    r[:, i+1] = r[:, i] + kappa*(r_mean - r[:, i])*delta_t + sigma*np.
r = r[:, 1:]

r_i = []
for i in range(N):
    summation = np.sum(r[i, :])*-delta_t
    r_i.append(np.exp(summation))

option_price = 0
face_value, K = 1000, 980
n_next = int((bond_maturity - option_maturity)/delta_t)
total_payoff = 0
for i in range(N):
    disc_rate = r_i[i]

    r_m = np.zeros((N, n_next+1))
    r_m[:, 0] = r[i, -1]

    z_next = np.random.normal(0, 1, (N, n))
    for i in range(n_next):
        r_m[:, i+1] = r_m[:, i] + kappa*(r_mean - r_m[:, i])*delta_t +
    r_m = r_m[:, 1:]

    sum_r_all_paths, payoff = 0, 0
    for i in range(N):
        sum_r_all_paths += np.exp(np.sum(r_m[i, :])*-delta_t)*face_val
    average_all_paths = sum_r_all_paths/N
    payoff = disc_rate*np.maximum(0, average_all_paths - K)
    total_payoff += payoff

option_price = total_payoff/N
option_price
```

Out[104]: 0.46928708560163557

b)

In [ ]:

```

```

c)

In [138]:

```
option_maturity, bond_maturity = 0.5, 1
delta_t = 1/252
K = 980
N, n = 1000, int(bond_maturity/delta_t)
z = np.random.normal(0, 1, (N, n))
r = np.zeros((N, n+1))
r[:, 0] = r0
for i in range(n):
    r[:, i+1] = r[:, i] + kappa*(r_mean - r[:, i])*delta_t + sigma*np.
r = r[:, 1:]

# Price of bond that matures at t = S
sum_r_all_paths = 0
for i in range(N):
    sum_r_all_paths += np.exp(np.sum(r[i, :])*-delta_t)*face_value
zcb_S = sum_r_all_paths/N
sum_r_all_paths = 0

# Price of bond that matures at t = T
T = int((bond_maturity-option_maturity)/delta_t)
for i in range(N):
    sum_r_all_paths += np.exp(np.sum(r[i, :T+1])*-delta_t)
zcb_T = (sum_r_all_paths/N)
sum_r_all_paths = 0

# Explicit method for price of option
s_t = bond_maturity - option_maturity
theta = np.sqrt(kappa**2 + 2*sigma**2)
phi = (2 * theta)/(sigma**2 * (np.exp(theta*s_t)-1))
psi = (kappa + theta)/sigma**2

h1 = np.sqrt(kappa**2 + 2*sigma**2)
h2 = (kappa + h1)/2
h3 = (2*kappa*r_mean)/sigma**2

A = ((h1*np.exp(h2*s_t))/(h2*(np.exp(h1*s_t)-1)+h1))**h3
B = (np.exp(h1*s_t)-1)/(h2*(np.exp(h1*s_t)-1)+h1)

K = K/face_value
```

```
r_star = np.log(A/K)/B
r_t_S = np.mean(r[:, -1])
r_t_T = np.mean(r[:, :T+1])

first_P = 2*r_star*(phi + psi + B)
sec_P = (4*kappa*r_mean)/sigma**2
third_P = (2*(phi**2)*r0*np.exp(theta*s_t))/(phi + psi + B)

first_K = 2*r_star*(phi + psi)
sec_K = (4*kappa*r_mean)/sigma**2
third_K = (2*(phi**2)*r0*np.exp(theta*s_t))/(phi + psi)

option_price = (zcb_S/face_value)*ncx2.cdf(first_P, sec_P, third_P) -
option_price = option_price*face_value
option_price
```

Out[138]: 0.4082322142135164

Quesion 3) G2++ model

In [176]:

```
x0, y0, phi_0, r0, rho = 0, 0, 0.03, 0.03, 0.7
a, b, sigma, eta, phi_t = 0.1, 0.3, 0.03, 0.08, 0.03
```

Monte Carlo

In [140]:

```
option_maturity, bond_maturity = 0.5, 1
delta_t = 1/252
N, n = 1000, int(option_maturity/delta_t)
z1 = np.random.normal(0, 1, (N, n+1))
z2 = np.random.normal(0, 1, (N, n+1))
for i in range(n):
    z2[:, i] = z1[:, i]*rho + z2[:, i]*np.sqrt(1 - rho**2)
r = np.zeros((N, n+1))
x = np.zeros((N, n+1))
y = np.zeros((N, n+1))
r[:, 0] = r0
x[:, 0] = x0
y[:, 0] = y0
for i in range(n):
    x[:, i+1] = x[:, i] + -a*x[:, i]*delta_t + sigma*np.sqrt(delta_t)*
    y[:, i+1] = y[:, i] + -b*y[:, i]*delta_t + eta*np.sqrt(delta_t)*z2
    r[:, i+1] = x[:, i+1] + y[:, i+1] + phi_t
r = r[:, 1:]

r_i = []
for i in range(N):
    summation = np.sum(r[i, :])*-delta_t
    r_i.append(np.exp(summation))
```

```
    r_i.append(np.exp(summation))

option_price = 0
face_value, K = 1000, 950
n_next = int((bond_maturity - option_maturity)/delta_t)
total_payoff = 0
for i in range(N):
    disc_rate = r_i[i]

    r_m = np.zeros((N, n_next+1))
    x_m = np.zeros((N, n_next+1))
    y_m = np.zeros((N, n_next+1))

    r_m[:, 0] = r[i, -1]
    x_m[:, 0] = x[i, -1]
    y_m[:, 0] = y[i, -1]

    z1_next = np.random.normal(0, 1, (N, n_next+1))
    z2_next = np.random.normal(0, 1, (N, n_next+1))
    for i in range(n):
        z2_next[:, i] = z1_next[:, i]*rho + z2_next[:, i]*np.sqrt(1 -

    for i in range(n_next):
        x_m[:, i+1] = x_m[:, i] + -a*x_m[:, i]*delta_t + sigma*np.sqrt
        y_m[:, i+1] = y_m[:, i] + -b*y_m[:, i]*delta_t + eta*np.sqrt(d
        r_m[:, i+1] = x_m[:, i+1] + y_m[:, i+1] + phi_t
    r_m = r_m[:, 1:]

    sum_r_all_paths, payoff = 0, 0
    for i in range(N):
        sum_r_all_paths += np.exp(np.sum(r_m[i, :])*-delta_t)*face_val

    average_all_paths = sum_r_all_paths/N
    payoff = disc_rate*np.maximum(0, K - average_all_paths)

    total_payoff += payoff

option_price = total_payoff/N
option_price
```

Out[140]: 1.4828836563498287

Explicit Method

In [201]: