FIN 503: Advanced derivatives **(Due: 17/11/20)**

Assignment #9

Professor: Elena Perazzi *Assistant:* Marc-Aurèle Divernois

Students: Hien Lê, Francesco Maizza, Anita Mezzetti

Method 1: analytical formula

The Price of Zero Coupon Bond Put using the analytical formula and the give parameters is:

$$Price_{XBP} = 0.00748$$

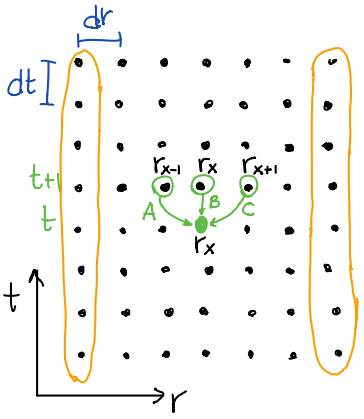
SECOND POINT

$$\frac{\partial V}{\partial t} + \frac{\partial V}{\partial r} k(\theta - r) + \frac{1}{2} \frac{\partial^2 V}{\partial r^2} \sigma r^2 = rV \qquad V := V(t,r) \quad r := r(t)$$

These quantities need to be approximated using the finite difference method.

First, we need to create a grid in which each step is dt or dr, according to the axis. Note that smaller steps lead to a better approximation.
We use a fixed step dt, while dr changes.

Grid



$$\begin{cases} r = 0, dr, 2dr, \dots, M\,dr = r_{max} \\ t = 0, dt, 2dt, \dots, T_2 \end{cases}$$

We need to set the boundary conditions we will use as starting points. As a matter of fact, in our code we set

$V(i,0)$
$V(0, min)$ } depend on the relation between r grid and r_0
$V(0, max)$ }

$$\frac{dV_{ij}}{dt} \approx \frac{V_{ij} - V_{i,j+1}}{dt \cdot t} \qquad \frac{dV_{ij}}{dr} \approx \frac{V_{i+1,j} - V_{i-1,j}}{2(dr \cdot r)} \qquad \frac{d^2V_{ij}}{dr^2} \approx \frac{V_{i+1,j} - 2V_{ij} + V_{i-1,j}}{(dr \cdot r)^2}$$

If we rewrite the PDE:

$$\frac{V_{i,N} - V_{i,N-1}}{dt} + k(\theta - r) \frac{V_{i+1,N} - V_{i-1,N}}{2dr} + \frac{1}{2} \sigma r^2 \frac{V_{i+1,N} - 2V_{i,N} + V_{i-1,N}}{dr^2} = r V_{iN}$$

All the terms depend on N, apart from $V_{i,N-1}$. We need to find it:

$$V_{\substack{i,N-1 \\ r \uparrow \quad \uparrow time}} = dt\,k(\theta - r) \frac{V_{i+1,N} - V_{i-1,N}}{2dr} + \frac{dt}{2} \sigma r^2 \frac{V_{i+1,N} - 2V_{i,N} + V_{i-1,N}}{dr^2} + (1 - dt\,r) V_{iN} =$$

$$= \underbrace{\left[-\frac{dt}{2dr}(\theta - r) + \frac{dt}{2dr^2} \sigma r^2 \right]}_{A} V_{i-1,N} + \underbrace{\left[1 - rdt - \frac{dt}{dr^2} \sigma r^2 \right]}_{B} V_{i,N} + \underbrace{\left[\frac{dt}{2dr} k(\theta - r) + \frac{dt}{2dr^2} \sigma r^2 \right]}_{C} V_{i+1,N}$$

We start our process from T_2, the final T, for which we know the value of the Option:

$$(\text{Strike} - \text{Bond Price}(r.grid))^+$$

→ In the code we use the function maxx

Recursively, we find the values for all the other t. In particular, the value at t can be extracted by three values at $t+1$ (see ✓ for the graphical idea):

$$V(t, r_x) = A\,V(t+1, r_{x-1}) + B\,V(t+1, r_x) + C\,V(t+1, r_{x+1})$$

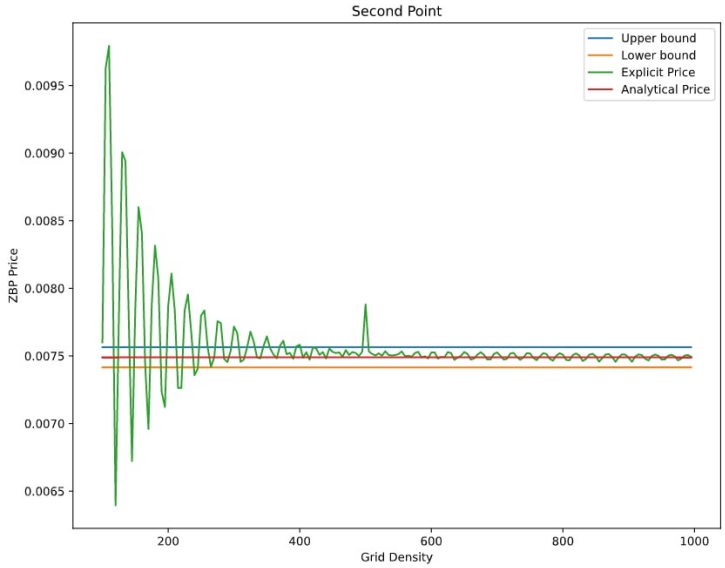
To make everything faster, we do not create a loop for each possible r_x, we work directly with vectors instead.

Of course, this equation does not hold for the first and the last column (see ——)
Therefore we set them manually:

$V(:, r_{max}) = 0$ ← maximum discounts ⇒ the values goes to zero

$V(:, 0) = k - bond\_price(0)$ ← no discount if $r=0$

We repeat the same process for different dr. So, at each step, we create a new grid and we follow the previous steps. As anticipated, increasing the density the price is better approximated. In particular, after some initial oscillations, the price becomes quite stable.



dr becomes smaller →

Point 3: Crank-Nicholoson method

See Matlab code for implementation. In this exercise we take i as index of r and j of time. We also chose $r_{min} = 0$ and $r_{max} = 1$.

We can try to rewrite the equation on slide 20 to get:

$$\begin{aligned}
&V_{ij+1}\left(\frac{1}{dt} - \frac{r_i}{2} - \frac{\sigma^2}{2dr^2}\right) - V_{ij}\left(\frac{1}{dt} + \frac{\sigma^2}{2dr^2} + \frac{r_i}{2}\right) \\
&+ V_{i+1,j+1}\left(\frac{k(\theta - r_i)}{4dr} + \frac{\sigma^2}{4dr^2}\right) - V_{i-1,j+1}\left(\frac{k(\theta - r_i)}{4dr} - \frac{\sigma^2}{4dr^2}\right) \\
&+ V_{i+1,j}\left(\frac{k(\theta - r_i)}{4dr} + \frac{\sigma^2}{4dr^2}\right) - V_{i-1,j}\left(\frac{k(\theta - r_i)}{4dr} - \frac{\sigma^2}{4dr^2}\right) \\
&= 0
\end{aligned}$$

Now, take a, b, c such that:

$$a = \frac{k(\theta - r_i)}{4dr} + \frac{\sigma^2}{4dr^2} \tag{0.1}$$

$$b = \frac{k(\theta - r_i)}{4dr} - \frac{\sigma^2}{4dr^2} \tag{0.2}$$

$$c = \frac{r_i}{2} + \frac{\sigma^2}{2dr^2} \tag{0.3}$$

And so, the equation becomes:

$$V_{i,j+1}(1/dt - c) - V_{i,j}(1/dt + c) + V_{i+1,j+1}a - V_{i-1,j+1}b + V_{i+1,j}a - V_{i-1,j}b = 0 \tag{0.4}$$

$$V_{i,j+1}(1/dt - c) - V_{i-1,j+1}b + V_{i+1,j+1}a = V_{i-1,j}b + V_{i,j}(1/dt + c) - V_{i+1,j}a \tag{0.5}$$

Now, note that the above can be written as

$$AV_{j+1} = BV_j - d$$

where A, B are tridiagonal matrices. We now only have to fill in these two matrices by a, b, c for each of the r_i in the r grid. The offset vector d is one where all elements are 0 except for the first and the last element (see slide 20). The first element is $r_1 \times (V_{1,j} + V_{1,j+1})$ and the last $r_{last} \times (V_{1,j} + V_{1,j+1})$ for each element j of the value matrix.

For the boundary conditions, we defined:

$$\begin{aligned}
V_{i,N} &= max(0, K - P(T_1, T_2, r_i(T_1)); \\
V_{0,j} &= K - P(T_1, T_2, 0); \\
V_{M,j} &= 0
\end{aligned}$$

Once the operations above have been carried out, we can calculate:

$$V_j = B^{-1}(AV_{j+1} + d)$$

To fasten computation, we did an LU decomposition, which is a method used for decomposing triangular matrices, on B.

Figure 1 shows the graph of the calculated price against the range of N_r (number of r intervals), while dt kept constant at 0.0001. We see that as N_r increases, price converges to around 0.0074 which is very close to the analytical price. Note that here we take $N_r \in (100, 1000)$ with a step-size = 10, which means that $\delta_r \in (0.01, 0.001)$ with the chosen r_{min}, r_{max}.

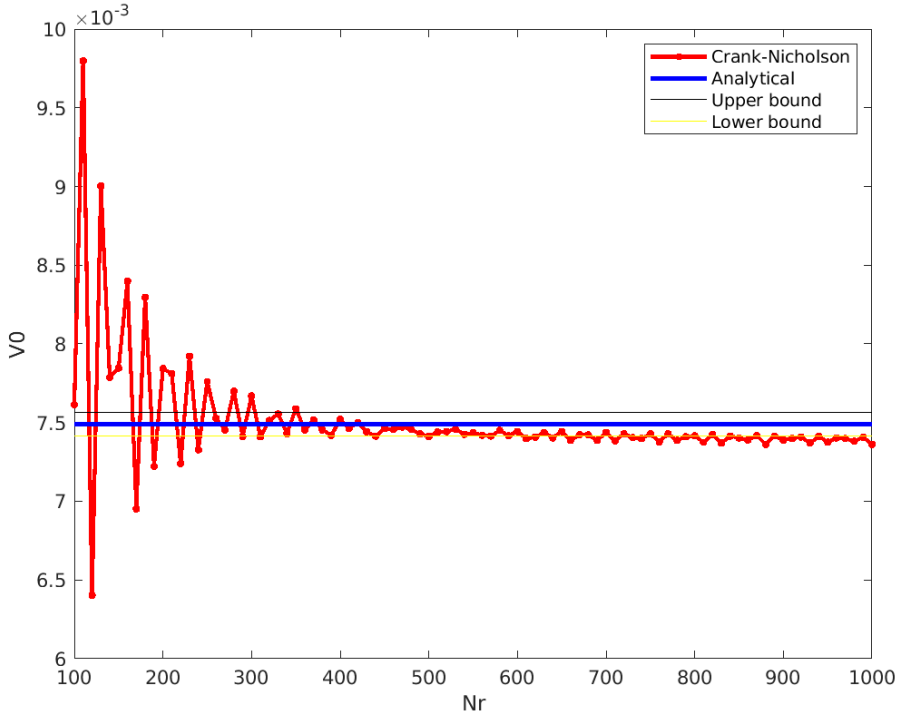


Figure 1: Crank-Nicholson prices against N_r

Figure 1 shows the graph of the calculated price against the range of N_t (number of t intervals), while dr was being kept constant at 0.05. The range in this picture is between 5 and 150 (step-size of 5), which means $dt \in (0.25/5, 0.25/150)$ ($T_1 = 0.25$). Here we see a similar picture, but with the range of fluctuation much smaller than before (the first value is around 0.00795).

We saw that for δ_r, with N_r greater or equal to ~ 400 ($dr = 0.04$) the prices stay more or less in the 1% of the analytical price. Whereas with δ_t, it is after $N_t \approx 20$ ($dt = 0.0125$) that the prices start to stay within the lower bound. The implicit method was not asked to be implemented as one of the three pricing methods, but judging from the fact that Crank-Nicholson is the average of explicit at $j+1$ and implicit at j, and from the results that we just obtained for the two latest methods, we can guess that the above-mentioned intervals will probably also work well for the implicit method.

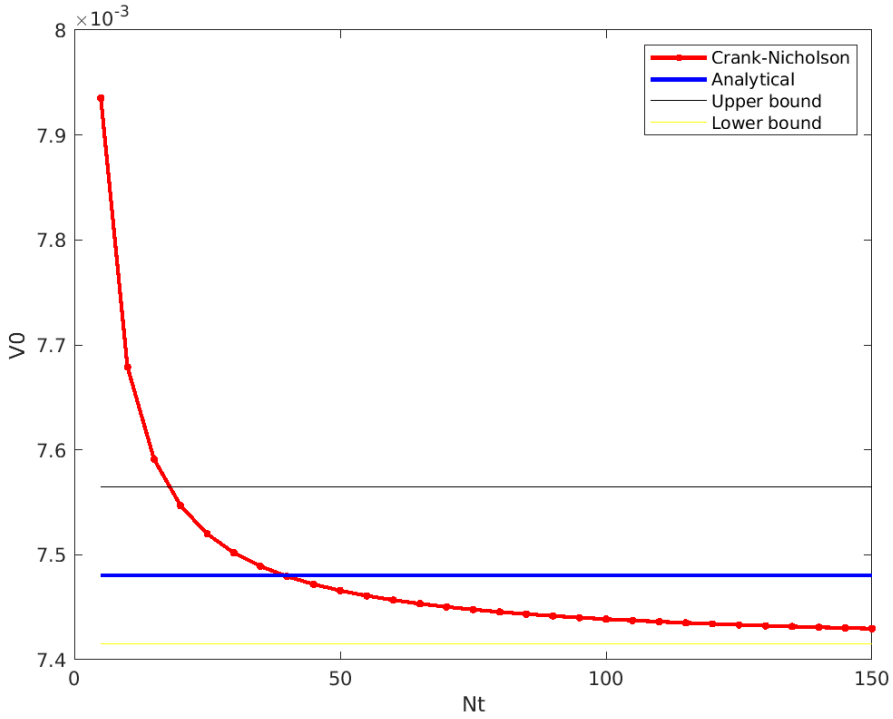


Figure 2: Crank-Nicholson prices against N_t

as9

November 16, 2020

[1]:
```
import numpy as np
import random as rd
import matplotlib.pyplot as plt
from scipy.stats import norm
```

0.1 Method 1: analytical formula

[2]:
```
def BB(t1, t2):
    return 1/k*(1-np.exp(-k*(t2-t1)))

def bond_price (T1, T2, r, theta, sigma, k): ## function for determing bond␣
 ↪price
    b = BB(T1,T2) #1/k*(1-np.exp(-k*(T2-T1)))
    A = np.exp((theta - sigma**2 / (2 * k**2)) * \
               (b-(T2-T1))-sigma**2/(4*k) * b**2)
    return A*np.exp(-b * r)
```

[3]:
```
strike = 0.805
k = 0.15
sigma_r = 0.01
theta = 0.05
r0 = 0.042
T1 = 0.25 # 3 months expiration date put option
T2 = 5.25

sigma_t = sigma_r*np.sqrt((1-np.exp(-2*k*T1))/(2*k))* BB(T1,T2)
h= 1 / sigma_t*np.log(bond_price(0,T2,r0,theta,sigma_r,k) /␣
 ↪(strike*bond_price(0,T1,r0,theta,sigma_r,k))) + sigma_t/2
```

[4]:
```
price_1 = strike * bond_price(0,T1,r0,theta,sigma_r,k) * \
            norm.cdf(-h+sigma_t) - bond_price(0,T2,r0,theta,sigma_r,k) * norm.
 ↪cdf(-h)
```

[5]:
```
print('Price of ZBP using the analytical formula is\n', price_1)
```

```
Price of ZBP using the analytical formula is
 0.0074897997038153585
```

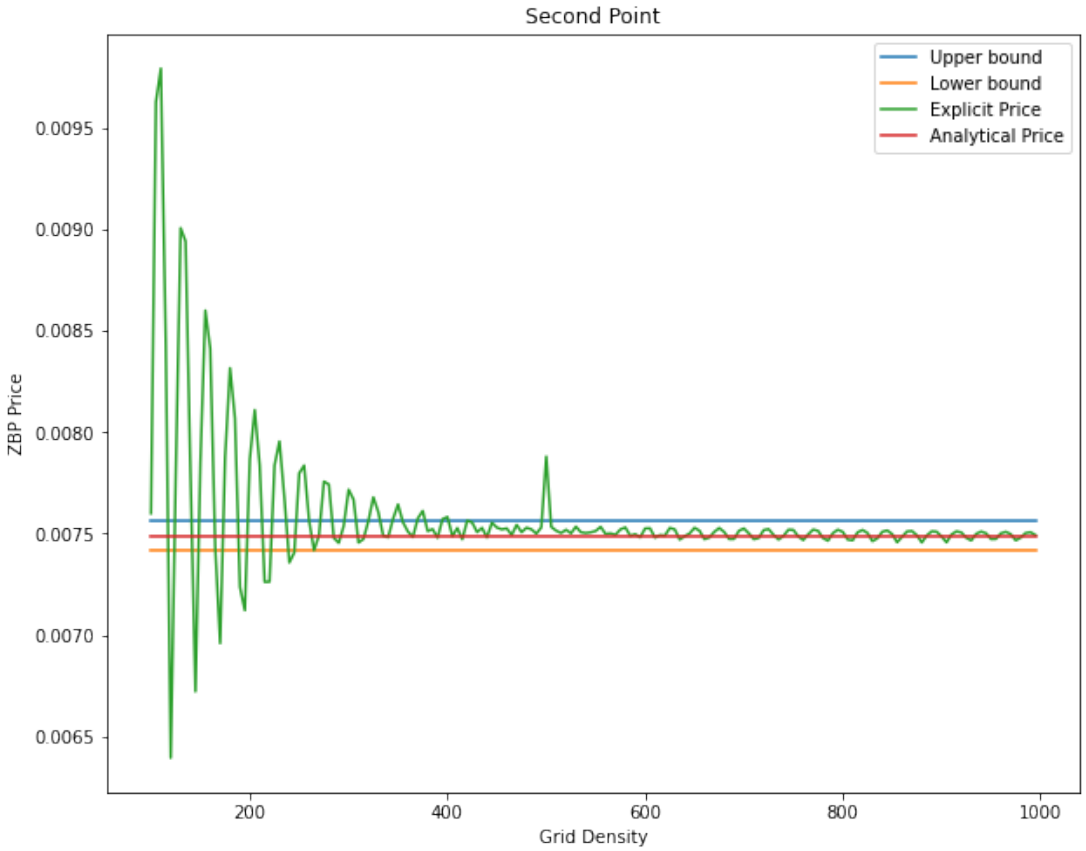
0.1.1 Method 2: Solving the PDE

[6]:
```
def maxx(a): # finctipn which compute max(0,a(i)) for each a(i) in a
    res = np.array([])
    for k in range(0,len(a)):
        if a[k] > 0:
            res = np.append(res,a[k])
        else:
            res = np.append(res,0)
    return res
```

[11]:
```
price_2 = []

r_density = np.arange(100,1000,5)

# r boundaries
[r_min, r_max] = [0, 1]

dt = 0.001 # time step

t_grid = np.arange(0,T1,dt)

for ri in r_density:

    dr = 1/ri

    r_grid = np.arange(r_min,r_max,dr)

    V = np.zeros((len(t_grid),len(r_grid)))

    # A B and C:
    A = dt * ( - k * (theta - r_grid[1:len(r_grid)-2]) / (2 * dr) \
            + sigma_r**2 / (2 * dr**2) )
    B = 1 - sigma_r**2 * dt / dr**2  -  dt * r_grid[1:len(r_grid)-2]
    C = A + dt/dr * k * (theta-r_grid[1:len(r_grid)-2])

    V[len(t_grid)-1,:] = maxx(strike - bond_price(T1, T2, r_grid, theta,␣
↪sigma_r, k))

    # set teh first and last column manually
    V[:, 0] = strike - bond_price(T1, T2, 0, theta, sigma_r, k)
    V[:, len(r_grid)-1] = 0

    for t in range(len(t_grid)-2,-1,-1):
        V[t, 1:len(r_grid)-2] = A * V[t+1, 0:len(r_grid)-3] + \
                                B * V[t+1, 1:len(r_grid)-2] + \
                                C * V[t+1, 2:len(r_grid)-1]
```

```
    r_grid_down = r_grid[np.where(r_grid < r0)[0][-1]]
    r_grid_up = r_grid[np.where(r_grid > r0)[0][0]]

    v_down = V[0, np.where(r_grid < r0)[0][-1]]
    v_up = V[0, np.where(r_grid > r0)[0][0]]

    price_2 += [v_down+(v_up-v_down)*(r0-r_grid_down)/(r_grid_up-r_grid_down)]
```

[12]:
```
plt.figure(figsize=(10,8))
plt.plot(r_density,len(r_density)*[1.01*price_1], label = 'Upper bound')
plt.plot(r_density,len(r_density)*[0.99*price_1], label = 'Lower bound')
plt.plot(r_density,price_2, label = 'Explicit Price')
plt.plot(r_density,len(r_density)*[price_1],label = 'Analytical Price')
plt.legend()
plt.xlabel("Grid Density")
plt.ylabel("ZBP Price")
plt.title('Second Point')
plt.show()
```



[ ]:

0.3 Method 3: Crank-Nicholson

```
%clear all
%close all
%clc
analytical_price = 0.0074897997038153585;
%% params
K = 0.805; % strike
k = 0.15; % kappa
sigma_r = 0.01;
theta = 0.05;
r0 = 0.042;

T1 = 3/12; % 3 months
T2 = 5 + 3/12; % 5 years 3 months

%% other self-defined params
Nrs = 100:10:1000;
Nts = 5:5:150;
r_min = 0;
r_max = 1;

%% different rs
Vs = zeros(length(Nrs),1);
for i=1:length(Nrs)
    Nr = Nrs(i);
    %disp(Nrs(i))
    dt = 0.0001;
    Nt = T1/dt;
    Vs(i) = FiniteDiffCN(Nr, Nt, r_max, r_min, K, k, sigma_r, theta,...
        r0, T1, T2);
end

%% different ts
Vs_T = zeros(length(Nts),1);
for i=1:length(Nts)
    Nt = Nts(i);
    %disp(Nrs(i))
    Nr = 500;
    Vs_T(i) = FiniteDiffCN(Nr, Nt, r_max, r_min, K, k, sigma_r, theta,...
        r0, T1, T2);
end

%% plot for Nr
figure(1)
plot(Nrs, Vs, 'r.-','LineWidth',2,'MarkerSize',10)
hold on
plot(Nrs, analytical_price*ones(length(Vs),1), 'b-', 'LineWidth',...
```

```
    2,'MarkerSize',10)
hold on
plot(Nrs, analytical_price*1.01*ones(length(Vs),1), 'k-', 'LineWidth',...
    0.75,'MarkerSize',5)
hold on
plot(Nrs, analytical_price*0.99*ones(length(Vs),1), 'y-', 'LineWidth',...
    0.75,'MarkerSize',5)
legend({'Crank-Nicholson', 'Analytical', 'Upper bound', 'Lower bound'})
xlabel("Nr")
ylabel("V0")

saveas(gcf,'CN_r.png')
hold off

%% plot for Nt
figure(2)
hold off
plot(Nts, Vs_T, 'r.-','LineWidth',2,'MarkerSize',10)
hold on
plot(Nts, 0.00748*ones(length(Vs_T),1), 'b-',...
    'LineWidth',2,'MarkerSize',10)
hold on
plot(Nts, analytical_price*1.01*ones(length(Vs_T),1), 'k-', 'LineWidth',...
    0.75,'MarkerSize',5)
hold on
plot(Nts, analytical_price*0.99*ones(length(Vs_T),1), 'y-', 'LineWidth',...
    0.75,'MarkerSize',5)

legend({'Crank-Nicholson', 'Analytical', 'Upper bound', 'Lower bound'})
xlabel("Nt")
ylabel("V0")

saveas(gcf,'CN_t.png')

%% Intervals where price is 1% from analytical
intervals_r = Nrs(abs());
intervals_t = Nts(abs(Vs_T - analytical_price) <= 0.01);

%% CN
function price = FiniteDiffCN(Nr, Nt, r_max, r_min, K, k, sigma_r,...
        theta, r0, T1, T2)

dr = (r_max-r_min)/Nr;
r_grid = r_min:dr:r_max;

dt = T1/Nt;
t_grid = 0:dt:T1;
```

```
M = round((r_max - r_min)/dr); % number of rates

N= length(t_grid); % number of time points

V = zeros(M+1, N+1);
%disp(size(V))
% boundary conditions
V(:, N+1) = max(0, K - bond_price(T1, T2, sigma_r, k, theta, r_grid))';
V(:,1) = (K- bond_price(T1, T2, sigma_r, k, theta, 0))';
V(M+1, :) = 0;

% see overleaf file for explanation
a = @(r) (k*(theta-r))/(4*dr) + (sigma_r^2)/(4*dr^2);
b = @(r) (k*(theta-r))/(4*dr) - (sigma_r^2)/(4*dr^2);
c = @(r) r/2 + sigma_r^2/(2*dr^2);

A = zeros(M-1, M-1);
B = zeros(M-1, M-1);

for i=2:M-1
    A(i-1,i) = a(r_grid(i));
    A(i,i-1) = -b(r_grid(i));
    B(i-1,i) = -a(r_grid(i));
    B(i,i-1) = b(r_grid(i));
end

for i=1:M-1
    A(i,i) = 1/dt - c(r_grid(i));
    B(i,i) = 1/dt + c(r_grid(i));
end

% fill V by using % A*V(j) = B*V(j+1) + d;
[L,U] = lu(B);

for i=N+1:-1:2
    d = zeros(size(A,2),1); % what should d be?
    d(1) = b(r_grid(2))*(V(1,i-1) + V(1,i));
    d(end) = a(r_grid(end))*(V(end,i-1)+ V(end,i));
    V(2:M,i-1) = U\(L\(A*V(2:M,i) + d));
end

price = interp1(r_grid,V(:,1),r0);
end

%% function of bond price
function price = bond_price(T1, T2, sigma, k, theta, r)
    B= 1/k * (1 - exp(-k*(T2-T1)));
    A = exp((theta - sigma^2/(2*k^2))*(B - T2 + T1) - sigma^2/(4*k)*B^2);
```

```
    price = A * exp(-B*r);
end
```