Midterm Exam - FE621

Waldyr Faustini

In [98]:
```python
import numpy as np
```

Problem 1

a) European put

We will be using the additive binomial tree adapted from HW2. In order to calculate delta (as will be asked in problem (b)) we also return delta, as per the following discussion:

We will **assume** that 1 option -- 1 share (instead of the more common convention of 100 shares). In this case, suppose the investor is long 1 put.

The hedging discussion we will use here applies to the American case below as well.

Suppose we start at $t = 0$ with a stock worth S_0. At time $t_1 = \Delta t$, we have two scenarios:

- UP scenario: option worth C_u, stock worth S_u
- DOWN scenario: option worth C_d, stock worth S_d

We want to to find an amount Δ such that a portfolio consisting of:

- One long position in the put
- Δ long positions in the underlying is worth the same for both UP and DOWN scenarios, i.e.

$$C_u + \Delta S_u = C_d + \Delta S_d$$

or

$$\Delta = -\frac{C_u - C_d}{S_u - S_d}$$

Note that these are the values *one time step before reaching* $t = 0$ in our model where we go back in time. This corresponds to $i = 1$ in the loop below.

In [63]:
```python
def euro_vanilla_pricer(N, T, S, K, r, sigma, option_type, calculate_delta=False):

    if option_type == 'call':
        payoff = lambda s, k: max(s-k, 0)
    elif option_type == 'put':
        payoff = lambda s, k: max(k-s, 0)

    # Trigeorgis 1992 coeffs
    dt = T/N
    nu = r - 0.5 * sigma**2
    dx_u = np.sqrt(sigma**2 * dt + nu**2 * dt**2)
    dx_d = - dx_u
    p_u = 0.5 * (1 + nu * dt/dx_u)
    p_d = 1 - p_u
    edxd = np.exp(dx_d)

    # one-period discount factor
    disc = np.exp(- r * dt)
    walk_up = np.exp(dx_u - dx_d)

    # initialize asset price at maturity, starting with the bottom one
    # we will use the convention that 0 denotes the lowest value
    ST = np.zeros(N+1)
    ST[0] = S * np.exp(N * dx_d)
    for i in range(1, N+1, 1):
        ST[i] = ST[i-1]*walk_up

    value = np.zeros(N+1)
    for j in range(N+1):
        value[j] = payoff(ST[j], K)

    # now we just propagate back - since these are European options,
    # value at each node is just the propagated discounted expectation
    for i in range(N-1, -1, -1):
        for j in range(i+1):
            ST[j] = ST[j]/edxd  # unnecessary for pricing, necessary for hedging
            value[j] = disc * (p_u * value[j+1] + p_d * value[j])

        if i == 1:
            delta = -(value[0] - value[1])/(ST[0] - ST[1])

    if calculate_delta:
        return value[0], delta
    else:
        return value[0]
```

In [84]:
```python
T = 9/12
N_periods = 3
K = 49
sigma = 0.3
r = 0.05
S0 = 50
```

In [85]:
```python
put_price = euro_vanilla_pricer(N=N_periods, T=T, K=K, S=S0, r=r, sigma=sigma, option_type='put')
print("Put price for given settings (European): %.2f" % put_price)
```

```
Put price for given settings (European): 4.15
```

In [95]:
```python
_, delta = euro_vanilla_pricer(N=N_periods, T=T, K=K, S=S0, r=r, sigma=sigma, option_type='put',
                               calculate_delta=True)
print("One should long %.2f" % delta + " units of stock to hedge the put option at t=0")
```

```
One should long 0.37 units of stock to hedge the put option at t=0
```

Sanity check: compare with Black-Scholes value for delta for a put option (sign will be inverted since we are always short delta when holding puts)

In [93]:
```python
import scipy.stats
N_prime = scipy.stats.norm.pdf

d1 = ((np.log(S0/K) + (r + sigma**2/2)*T))/(sigma*np.sqrt(T))

print("BSM value: %.2f" % -N_prime(-d1))
```

```
BSM value: -0.37
```

American put

We will be using the additive binomial tree adapted from HW2:

In [94]:
```python
def american_vanilla_pricer(N, T, S, K, r, sigma, option_type, calculate_delta=False):


    if option_type == 'call':
        payoff = lambda s, k: max(s-k, 0)
    elif option_type == 'put':
        payoff = lambda s, k: max(k-s, 0)

    # Trigeorgis 1992 coeffs
    dt = T/N
    nu = r - 0.5 * sigma**2
    dx_u = np.sqrt(sigma**2 * dt + nu**2 * dt**2)
    dx_d = - dx_u
    p_u = 0.5 * (1 + nu * dt/dx_u)
    p_d = 1 - p_u

    # one-period discount factor and other constants
    disc = np.exp(- r * dt)
    walk_up = np.exp(dx_u - dx_d)
    d_p_u = disc * p_u
    d_p_d = disc * p_d
    edxd = np.exp(dx_d)

    # initialize asset price at maturity, starting with the bottom one
    # we will use the convention that 0 denotes the lowest value
    ST = np.zeros(N+1)
    ST[0] = S * np.exp(N * dx_d)
    for i in range(1, N+1, 1):
        ST[i] = ST[i-1]*walk_up

    value = np.zeros(N+1)
    for j in range(N+1):
        value[j] = payoff(ST[j], K)

    # now we just propagate back, including the early stop condition
    for i in range(N-1, -1, -1):
        for j in range(i+1):

            value[j] = d_p_d * value[j] + d_p_u * value[j+1]

            # however, there might be early exercise
            ST[j] = ST[j]/edxd
            value[j] = max(value[j],  payoff(ST[j], K))

        if i == 1:
            delta = -(value[0] - value[1])/(ST[0] - ST[1])

    if calculate_delta:
        return value[0], delta
    else:
        return value[0]
```

In [96]:
```python
put_price = american_vanilla_pricer(N=N_periods, T=T, K=K, S=S0, r=r, sigma=sigma, option_type='put')
print("Put price for given settings (American): %.2f" % put_price)
```

```
Put price for given settings (American): 4.29
```

In [97]:
```python
_, delta = american_vanilla_pricer(N=N_periods, T=T, K=K, S=S0, r=r, sigma=sigma, option_type='put',
                               calculate_delta=True)
print("One should long %.2f" % delta + " units of stock to hedge the put option at t=0")
```

```
One should long 0.39 units of stock to hedge the put option at t=0
```

c) Adding a barrier

In HW2 we built a pricer for European barrier options in the trinomial setting. We will adapt that code to use a binomial tree.

In practice:

- We add the barrier condition in the first payoff calculation at expiry
- We check the barrier condition for all back-propagated values
- Since our tree is no longer trinomial, we need to discount spot values when going back in time

In [37]:
```python
def euro_barrier_pricer(N, T, S, K, r, sigma, H, option_type, barrier_type):

    if barrier_type == 'uo':
        condition = lambda s, h: s <= h
    elif barrier_type == 'do':
        condition = lambda s, h: s >= h

    if option_type == 'call':
        payoff = lambda s, k: max(s-k, 0)
    elif option_type == 'put':
        payoff = lambda s, k: max(k-s, 0)

    # Trigeorgis 1992 coeffs
    dt = T/N
    nu = r - 0.5 * sigma**2
    dx_u = np.sqrt(sigma**2 * dt + nu**2 * dt**2)
    dx_d = - dx_u
    p_u = 0.5 * (1 + nu * dt/dx_u)
    p_d = 1 - p_u
    edxd = np.exp(dx_d)

    # one-period discount factor
    disc = np.exp(- r * dt)
    walk_up = np.exp(dx_u - dx_d)

    # initialize asset price at maturity, starting with the bottom one
    # we will use the convention that 0 denotes the lowest value
    ST = np.zeros(N+1)
    ST[0] = S * np.exp(N * dx_d)
    for i in range(1, N+1, 1):
        ST[i] = ST[i-1]*walk_up

    value = np.zeros(N+1)
    for j in range(N+1):
        value[j] = payoff(ST[j], K) * condition(ST[j], H)  # adding barrier condition

    # now we just propagate back - since these are European options,
    # value at each node is just the propagated discounted expectation
    for i in range(N-1, -1, -1):
        for j in range(i+1):
            ST[j] = ST[j]/edxd  # adding "discounting" of spot prices to "go back" in time
            value[j] = disc * (p_u * value[j+1] + p_d * value[j]) * condition(ST[j], H) # adding barrier condition

    return value[0]
```

In [29]:
```python
put_price = euro_barrier_pricer(N=N_periods, T=T, K=K, S=S0, r=r, sigma=sigma, H=38, option_type='put', barrier_type='do')
print("Put price for given settings (Euro/Barrier): %.2f" % put_price)
```

```
Put price for given settings (Euro/Barrier): 1.42
```

Trinomial tree

a)

Again we adapt our code from HW2. In this case:

- We remove the barrier conditions (although a "hot fix" could have been to keep it and set an infinitely large barrier)
- We add the early exercise feature

In [42]:
```python
def american_pricer_trinom(N, T, S, K, r, sigma, option_type, div=0.0):

    if option_type == 'call':
        payoff = lambda s, k: max(s-k, 0)
    elif option_type == 'put':
        payoff = lambda s, k: max(k-s, 0)

    # coefficients (from trinomal algo)
    dt = T/N
    nu = r - div - 0.5 * sigma**2
    dx = sigma * np.sqrt(3* dt) # optimal value
    edx = np.exp(dx)
    disc = np.exp(-r * dt)

    # probabilities (from trinomial algo)
    aux = (sigma**2 * dt + nu**2 * dt**2)/dx**2
    pu = 0.5 * (aux+ nu * dt/dx)
    pm = 1.0 - aux
    pd = 0.5 * (aux - nu * dt/dx)

    # to be able to properly evolve spot values back in time,
    ST = np.zeros(2*N+1)
    ST[0] = S*np.exp(-N*dx)

    for j in range(1, 2*N+1):
        ST[j] = ST[j-1]*edx

    # here we use a full grid for C instead of a vector
    value = np.zeros((N+1, 2*N+1))

    for j in range(2*N+1):
        value[N, j] = payoff(ST[j], K)

    for i in range(N-1, -1, -1):
        for j in range(N-i, N+i+1):

            # since S(t)_j = S(t+delta t)_j, there is no need to bring S back in time
            value[i,j] = disc * (pu * value[i+1, j+1] + pm * value[i+1, j] + pd * value[i+1, j-1])

            # early exercise
            value[i,j] = max(value[i,j],  payoff(ST[j], K))

    return value[0,N]
```

In [43]:
```python
put_price = american_pricer_trinom(N=N_periods, T=T, K=K, S=S0, r=r, sigma=sigma, option_type='put')
print("Put price for given settings (American/trinomial): %.2f" % put_price)
```

```
Put price for given settings (American/trinomial): 3.59
```

This value is different from the one in the previous exercise - we suspect this to be due to numerical inaccuracies due to only using 3 time steps.

b)

Increasing the count for both models:

In [49]:
```python
put_price = american_pricer_trinom(N=200, T=T, K=K, S=S0, r=r, sigma=sigma, option_type='put')
print("Put price for given settings (American/trinomial): %.2f" % put_price)

put_price = american_vanilla_pricer(N=200, T=T, K=K, S=S0, r=r, sigma=sigma, option_type='put')
print("Put price for given settings (American/binomial): %.2f" % put_price)
```

```
Put price for given settings (American/trinomial): 3.90
Put price for given settings (American/binomial): 3.91
```

We see the values converge at ~ 3.90.

Problem 3

a) Explicit

For the PDE

$$\frac{\partial V}{\partial t} + 2\tan(S)\frac{\partial V}{\partial S} + S^3\frac{\partial^2 V}{\partial S^2} - rV0$$

let us call $\mu(S) = 2\tan(S)$ and $\sigma(S) = S^3$. The under the explicit scheme, same as before

$$\frac{\partial V}{\partial t}(i,j) = \frac{V_{i+1,j} - V_{i,j}}{\Delta t}$$
$$\frac{\partial V}{\partial S}(i,j) = \frac{V_{i+1,j+1} - V_{i+1,j-1}}{2\Delta S}$$
$$\frac{\partial^2 V}{\partial S^2}(i,j) = \frac{V_{i+1,j+1} - 2V_{i+1,j} + V_{i+1,j-1}}{\Delta S^2}$$

We shall also write $S = j\Delta S$, such that

$$\mu_j = 2\tan(j\Delta S)$$

$$\sigma_j = (j\Delta S)^3$$

Putting everything together we get

$$\frac{V_{i+1,j} - V_{i,j}}{\Delta t} + \mu_j\frac{V_{i+1,j+1} - V_{i+1,j-1}}{2\Delta S} + \sigma_j\frac{V_{i+1,j+1} - 2V_{i+1,j} + V_{i+1,j-1}}{\Delta S^2}$$

$$\frac{V_{i,j}}{\Delta t} = \left[V_{i+1,j+1}\left(\frac{\mu_j}{2\Delta S} + \frac{\sigma_j}{\Delta S^2}\right) + V_{i+1,j}\left(\frac{1}{\Delta t} - \frac{2\sigma_j}{\Delta S^2}\right) + V_{i+1,j-1}\left(-\frac{\mu_j}{2\Delta S} + \frac{\sigma_j}{\Delta S^2}\right)\right]$$

$$\boxed{V_{i,j} = (p_u)_j V_{i+1,j+1} + (p_m)_j V_{i+1,j} + (p_d)_j V_{i+1,j-1}}$$

where, substituting the explicit formulae for μ_j and σ_j we have

$$\boxed{\begin{aligned}(p_u)_j &= \Delta t\left(\frac{2\tan(j\Delta S)}{2\Delta S} + \frac{(j\Delta S)^3}{\Delta S^2}\right)\\ (p_m)_j &= 1 - \Delta t\frac{(j\Delta S)^3}{\Delta S^2}\\ (p_d)_j &= \Delta t\left(-\frac{2\tan(j\Delta S)}{2\Delta S} + \frac{(j\Delta S)^3}{\Delta S^2}\right)\end{aligned}}$$

Notice how the coefficients are j-dependent since μ and σ are functions of j. Thus, for every iteration, they must be updated as well as the option values.

b) Implicit

To derive the implicit scheme all we do is formally replace $i+1 \to i$ for all "non-$\partial/\partial t$" terms in the PDE. Thus:

$$\frac{\partial V}{\partial t}(i,j) = \frac{V_{i+1,j} - V_{i,j}}{\Delta t}$$
$$\frac{\partial V}{\partial S}(i,j) = \frac{V_{i,j+1} - V_{i,j-1}}{2\Delta S}$$
$$\frac{\partial^2 V}{\partial S^2}(i,j) = \frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{\Delta S^2}$$

Putting everything together we get

$$\frac{V_{i+1,j} - V_{i,j}}{\Delta t} + \mu_j\frac{V_{i,j+1} - V_{i,j-1}}{2\Delta S} + \sigma_j\frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{\Delta S^2}$$

Isolating i terms in the LHS and $i+1$ terms in the RHS we get

$$V_{i,j}\left(\frac{1}{\Delta t} + \frac{2\sigma_j}{\Delta S^2}\right) + V_{i,j+1}\left(-\frac{\mu_j}{2\Delta S} - \frac{\sigma_j}{\Delta S^2}\right) + V_{i,j-1}\left(\frac{\mu_j}{2\Delta S} - \frac{\sigma_j}{\Delta S^2}\right) = \frac{V_{i+1,j}}{\Delta t}$$

From this we can conclude

$$\boxed{(p_u)_j V_{i,j+1} + (p_m)_j V_{i,j} + (p_d)_j V_{i,j-1} = V_{i+1,j}}$$

where

$$\boxed{\begin{aligned}(p_u)_j &= \Delta t\left(-\frac{\mu_j}{2\Delta S} - \frac{\sigma_j}{\Delta S^2}\right) = \Delta t\left(-\frac{2\tan(j\Delta S)}{2\Delta S} - \frac{(j\Delta S)^3}{\Delta S^2}\right)\\ (p_m)_j &= \Delta t\left(\frac{1}{\Delta t} + \frac{2\sigma_j}{\Delta S^2}\right) = 1 + \Delta t\frac{2(j\Delta S)^3}{\Delta S^2}\\ (p_d)_j &= \Delta t\left(\frac{\mu_j}{2\Delta S} - \frac{\sigma_j}{\Delta S^2}\right) = \Delta t\left(\frac{2\tan(j\Delta S)}{2\Delta S} - \frac{(j\Delta S)^3}{\Delta S^2}\right)\end{aligned}}$$

Problem 4

- (a): **TRUE**
- (e): **TRUE** since for log-return the expected value is $(r - \sigma^2/2)\Delta t < r\Delta t$.
- (h): **TRUE**
- Bonus: **TRUE** (see below)

To see why on the bonus question, we construct two independend normal variates X_1 and X_2 and consider the new variables

$$Z_1 = X_1$$
$$Z_2 = \rho X_1 + \sqrt{1-\rho^2}X_2$$

Then one can prove that Z_1 and Z_2 are also normal, with the same variance as X_1 or X_2, and with correlation ρ.

In [ ]: