Exercise 34: (Hedging a long position)

The investor should use the opposite of the replicating strategy derived in the lecturer for the seller of the option. This means that rather than holding $1/2$ shares of stocks she should hold $-1/2$ shares of stocks (i.e., she short sells the stock). This generates an income of $£1/2 \cdot 4 = 2$. She should now invest this in the riskless asset.

If at time 1 the stock price goes up, she has an option worth $£3$ and has $£2 \cdot \frac{5}{4} = £2.5$ in the riskless asset. She must pay $£0.5 \cdot 8 = £4$ to cover the short position in the stock, which leaves her with $£1.5$ as desired.

If at time 1 the stock price goes down, she has an option worth $£0$ and has $£2 \cdot \frac{5}{4} = £2.5$ in the riskless asset. She must pay $£0.5 \cdot 2 = £1$ to cover the short position in the stock, which leaves her with $£1.5$ as desired.

Exercise 35: (Monte Carlo approximation of European put option)

In the following we will be using the method choice available in numpy.random to generate a sample from a discrete probability distribution. See here:
https://numpy.org/doc/stable/reference/random/generated/numpy.random.choice.html

In [1]:
```
import numpy as np
from scipy.special import comb
```

In [2]:
```
rng = np.random.default_rng(12345)
```

In [3]:
```
def generateriskneutralstocks(S0, u, d, r, samplesize):
    ptilde = (1 + r - d) / (u - d)
    stocks =  rng.choice(np.array([S0 * u, S0 * d]), size = samplesize, replace = True, p=np.
    return(stocks)

def putpriceMC(S0, u, d, r, samplesize, K):
    mystocks = generateriskneutralstocks(S0, u, d, r, samplesize)
    payoffs= np.maximum(K - mystocks, 0) / (1 + r)
    return(payoffs.mean())

print('MC price of put in binomial model is: {:.4f}'.format(putpriceMC(S0=4, u=2, d=0.5, r=0.
```

```
MC price of put in binomial model is: 1.1822
```

Exercise 36: (Multi-period binomial model)

In [4]:
```
def europeanput_binomial(S0, K, r, N, u, d):
    ptilde = (1 + r - d) / (u - d)
    myprice = 0
    for k in range(0, N + 1):
```

```
        stockprice = S0 * (u ** k) * (d ** (N-k))
        myprice = myprice + \
            comb(N, k) * (ptilde ** k) * ((1-ptilde) ** (N-k)) *\
                np.maximum(K - stockprice, 0)
    result = myprice / ((1+r) ** N)
    return result

print('Analytical price of put in binomial model is: {:.4f}'.format(
      europeanput_binomial(S0=4, u=2, d=0.5, r=0.25, N=1, K=5)))
```

```
Analytical price of put in binomial model is: 1.2000
```

In [5]:

```
def europeancall_binomial(S0, K, r, N, u, d):
    ptilde = (1 + r - d) / (u - d)
    myprice = 0
    for k in range(0, N + 1):
        stockprice = S0 * (u ** k) * (d ** (N-k))
        myprice = myprice + \
            comb(N, k) * (ptilde ** k) * ((1-ptilde) ** (N-k)) *\
                np.maximum(stockprice - K, 0)
    result = myprice / ((1+r) ** N)
    return result

print('Analytical price of call in binomial model is: {:.4f}'.format(
      europeancall_binomial(S0=4, u=2, d=0.5, r=0.25, N=1, K=5)))
```

```
Analytical price of call in binomial model is: 1.2000
```

The put-call parity in the binomial model is given by $C_0 - P_0 = S_0 - K/(1+r)^N$ where C_0, P_0 denote the price of the European call and put option, respectively. Let us use this to check our implementation:

In [6]:

```
S0 = 100
K = 100
u = 3
d = 1 / 3
r = 2 / 3
N = 5

test = S0 - K / (1 + r) ** N

call = europeancall_binomial(S0=100, u=3, d=1/3, r=2/3, N=5, K=100)
put = europeanput_binomial(S0=100, u=3, d=1/3, r=2/3, N=5, K=100)

print('call - put = {:.4f}'.format(call - put))
print('S0 - K/(1+r)^N= {:.4f}'.format(test))
```

```
call - put = 92.2240
S0 - K/(1+r)^N= 92.2240
```

In [ ]: