# Dynamic hedging of a short position on a call

We consider the problem of hedging a short position on a European calls contract on a **Share XX** of maturity **T and strike K**. The contract is assumed to consist of a n number of calls, so that each is associated with exactly one share. In all the following it is assumed to take into account the transaction costs. This is because the purchase/sale of a number n of underlying XX, which is equal to S, incurs a fee of **φ(nS )**.

where φ is a positive function defined on axis [0, ∞) so that φ(0) = 0. This condition means that without a transaction there is no charge to be paid. A simple case often used is that of proportional charges, where φ(x) = cx, for a certain constant xe c > 0. It is also assumed that the purchase or sale of n calls generates a charge which is different from the underlying charge and is given by **ψ(nS)** where ψ is also a positive function satisfying the condition ψ(0) = 0.

The point for the call seller is to only dispose of a proportion of underlying the securities. The higher (or lower) the share price, the greater the need to strengthen (or reduce) the number of underlying securities The essence of the hedging problem is then to choose the quantity of securities to be held in such a way as to be able to face more or less serenely the maturity of the call..

The fundamental point in hedging an option position is to establish a position in a portfolio consisting of the underlying and some cash. The latter is constructed in such a way that it varies in the opposite direction to the position to be hedged. The loss incurred by the option will be offset by the gain brought by the hedging position

## The level of caching at each moment

At the initial point in time, the hedging position consists of **n_0** shares held at the unit price **S_0** and a cash portion **B_0** borrowed at an (annual) rate of **μ_0**

- $-B_0 = (\lambda_0 nC_0 + \psi(nC_0) + (n_0 S_0 + \varphi(n_0 S_0)) - \beta_0$
- $B_1 = (nC_0 + \lambda_0 nC_0) + (-nC_1 - \lambda_1 nC_1) + Su_1 + I_0 + \beta_1$

3. \

with, the cash portion B : \

- $I_0 = \delta_0 |B_0|(-\mu_0^{bor} I(B_0 < 0) + \mu_0^{len} I(0 < B_0))$

and what is due to the variation in the quantity of the underlying at any given time:

- $Su_1 \equiv Su_1(n_0, n_1, S_1) = (n_0 - n_1)S_1 - \varphi|n_0 - n_1|S_1$

and similarly,

- $B_m = -(1 + \lambda_m)n_C m - C_{m-1} + Su_m + (\lambda_{m-1} - \lambda_m)nC_{m-1} + I_{m-1} + \beta_m$

and at maturity you get,

- $B_M = -nC_M - C_{M-1} + \lambda_{M-1} nC_{M-1} + S_M + I_{M-1}$

## Losses and profits

We agree (improperly) to consider as loss and prot the level of cash Bm formed by passing from the instant $t_{m-1}$ to $t_m$ , i.e.,

$$P_L \equiv P_L(t_{m-1}, t_m) * P_L = -(1 + \lambda_m)nC_m - C_{m-1} + \lambda_{m-1} - \lambda_m nC_{m-1} + S_m + I_{m-1} + \beta_m$$

Without hedging or margin calls, the profit and loss would be,

$$P_L^{unhedged}(t_0, t_M) = (nC_0 - \psi(nC_0) + I(K < S_M)((nK - \varphi nK) - (nS_M + \varphi(nS_M))$$

Taking into account the hedging we obtain by summing up the instantaneous P&Ls:

- $P_L = nC_0 - C_M - n(\sum_{m=1}^{M} \lambda_m (C_m - C_{m-1}) + n\sum_{m=1}^{M}(\lambda_{m-1} - \lambda_m)C_{m-1} + \sum_{m=1}^{M}(n_{m-1} - n_m)S_m - \varphi(|n_{m-1} - n_m|)S_m$

Thus the resulting gross profit and loss in the absence of hedging will be essentially offset by the result of the purchases and sales of the underlying that are set against the short position to be hedged.

By introducing the $\Delta_m$ sensitivity of the option calculated at time $t_m$ with respect to the underlying. We obtain a linear approximation of $C_m$ as a function of $S_m$ . The $P_L$ function can then be approximated as follows:

$$P_L = (-n \triangle_m + n_m)(S_{m+1} - S_m) - n\alpha_m + \lambda_m nC_m + I_m(n_m, \beta_m) - \varphi(n_m S_m) - \psi(n_m, S_m)$$

To neutralize the uncertainties resulting from the unknown price $S_{m+1}$ one should choose $n_m$ so that : $n \triangle_m = n_m$

# The position under consideration:

We consider a short position on a number of **10,000** calls on an XX share of **remaining maturity 1 year**. The exercise price of each call is **100**, and that the **spot price is also 100**. For simplicity, we assume an annual credit risk free rate of **2%** and also an (implied) volatility that is assumed to be constant at **35%** for all maturities.

For market friction, a proportional transaction fee of **0.5%** is applied for the underlying and **0.2%** for derivatives. For simplicity, we consider an initial deposit rate of 20% constant until the maturity of the derivative. It is assumed that there is a margin call every month (i.e. 12) until the expiry of the call portfolio under consideration.

We consider a number J = 20 of possible price paths for share XX.

Let's start by simulating the J trajectories using the Black&Scholes model:

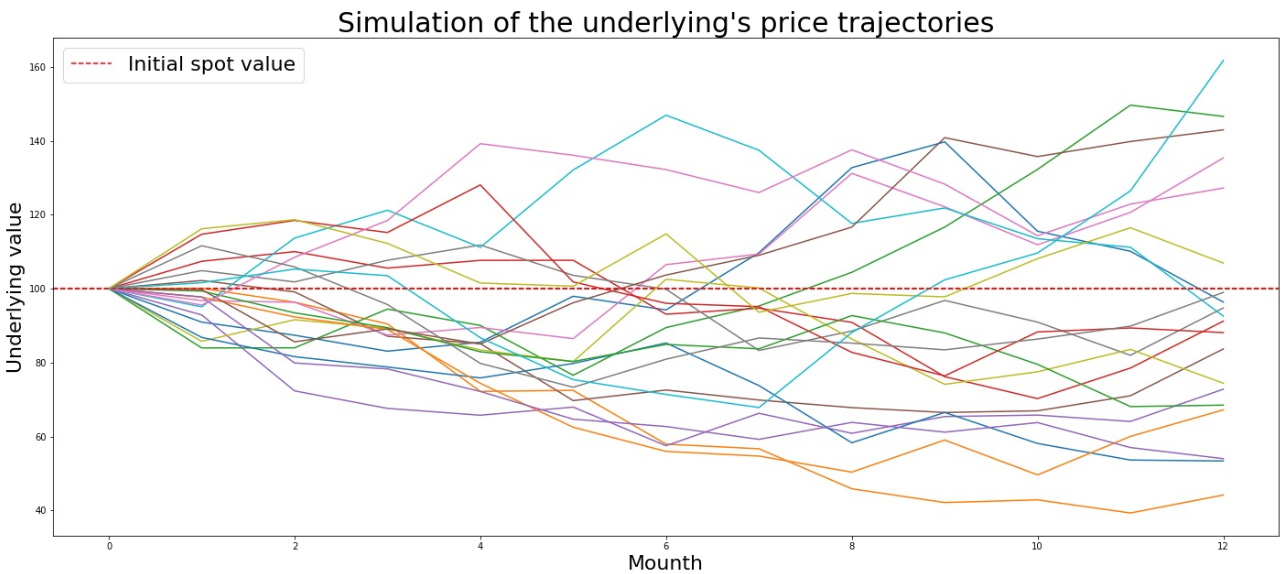
The theory of stochastic calculus allows us to see that in the classical Black Scholes context, the price of an XX share at an instant T, with t < T, is given by :

$$S_T(.) = S_t \exp((m - 2\sigma)\tau(t) + \sigma\sqrt{(\tau(t))}U(.))$$

This formula comes by introducing, using Ito's formula, the variable change $g(S) = log(S)$ in the Black&Scholes stochastic differential equation.

More details can be found here (http://www1.se.cuhk.edu.hk/~seem5670/Lecture/Lecture3.pdf).

```
import numpy as np
from pylab import *
import seaborn as sns
rcParams['figure.figsize'] = 24, 10

spot_init=100
int_rate=2/100
volatility=35/100
maturity=1
nb_step=12
step=\
    maturity/nb_step
J=20
nb_seed=20191008
strike = 100
n=10000

λ = 0.2


mat_d = (int_rate- 1/2 *volatility**2)*step *np.cumsum( np.ones((J,nb_step)),axis=1)

np.random.seed(nb_seed)

mat_shock = np.random.standard_normal((J,nb_step))
mat_spot_fut = spot_init*np.ones((J,nb_step))*np.exp(mat_d
                                                    + volatility * np.sqrt(step)*np.cumsum( mat_shock,axis=1 ))
vec_spot_init= spot_init*np.ones((J,1) )
mat_path_spot_fut= np.concatenate((vec_spot_init,np.round(mat_spot_fut,3)),axis=1)

axhline(y =100 , color='r', ls = "--")
plot(mat_path_spot_fut.T) # Simulation des trajectoires des cours du sous-jacent.
mat_path_spot_fut
plt.xlabel("Mounth", fontsize=22)
plt.ylabel("Underlying value", fontsize=22)
plt.title("Simulation of the underlying's price trajectories", fontsize=30 )
axhline(y =100 , color='r', ls = "--")
plt.legend(["Initial spot value"], fontsize=22 )
plt.show()
```




png

**1. Call values between 0 and M**

The theorem 7 in [1] (http://www1.se.cuhk.edu.hk/~seem5670/Lecture/Lecture3.pdf) provides the following calculations.

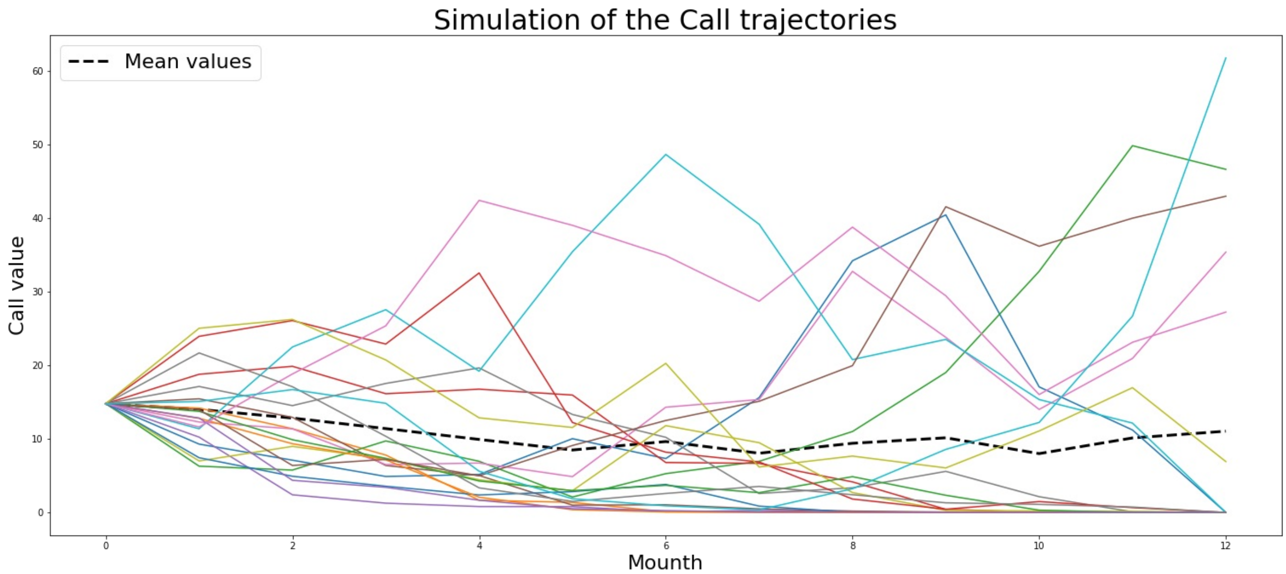
```python
import numpy as np
import scipy
from scipy import stats
from pylab import *
import pandas as pd

T = 12

def d1(S, K, T, r, σ, t):
    return (1/(σ*np.sqrt(T-t)))*(np.log(S/K) + (r + (1/2)*σ**2)*(T-t))


def d2(S, K, T, r, σ, t):
    return d1(S, K, T, r, σ, t) - σ * np.sqrt(T-t)

cdf = stats.norm(0,1).cdf

def call(S, K, T, r, σ, t):
    return S * cdf(d1(S, K, T, r, σ, t)) - K * np.exp(-r * (T-t)) * cdf(d2(S, K, T, r, σ, t))


Call = np.zeros((20,13))
np.set_printoptions(suppress=True)
for j in range(20):
    for i in range(T):
        Call[j][i] = np.round(call(mat_path_spot_fut[j][i],strike,maturity,int_rate,volatility,i/12),3)
        Call[j][T] = np.maximum(0,mat_path_spot_fut[j][T] - strike)


C = pd.DataFrame(data = Call)
C
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 14.767 | 9.279 | 7.098 | 4.885 | 5.154 | 10.015 | 7.317 | 15.587 | 34.167 | 40.402 | 17.046 | 11.198 | 0.000 |
| **1** | 14.767 | 14.161 | 11.370 | 7.790 | 1.665 | 1.365 | 0.100 | 0.036 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **2** | 14.767 | 6.282 | 5.757 | 9.707 | 6.944 | 2.095 | 5.269 | 6.939 | 10.986 | 18.998 | 32.735 | 49.794 | 46.582 |
| **3** | 14.767 | 18.758 | 19.842 | 16.120 | 16.737 | 15.940 | 6.779 | 6.667 | 4.150 | 0.420 | 0.028 | 0.026 | 0.000 |
| **4** | 14.767 | 10.225 | 2.395 | 1.261 | 0.800 | 0.795 | 0.092 | 0.271 | 0.040 | 0.038 | 0.006 | 0.000 | 0.000 |
| **5** | 14.767 | 15.426 | 12.915 | 6.376 | 5.018 | 0.990 | 1.043 | 0.488 | 0.188 | 0.051 | 0.009 | 0.001 | 0.000 |
| **6** | 14.767 | 12.276 | 11.352 | 6.502 | 6.708 | 4.865 | 14.290 | 15.327 | 32.732 | 23.774 | 13.988 | 20.910 | 35.343 |
| **7** | 14.767 | 17.103 | 14.504 | 17.516 | 19.624 | 13.303 | 10.201 | 2.574 | 3.344 | 5.578 | 2.139 | 0.088 | 0.000 |
| **8** | 14.767 | 6.992 | 8.969 | 7.282 | 4.431 | 2.951 | 11.786 | 9.472 | 2.704 | 0.285 | 0.197 | 0.145 | 0.000 |
| **9** | 14.767 | 11.363 | 22.463 | 27.528 | 19.174 | 35.391 | 48.606 | 39.127 | 20.753 | 23.493 | 15.330 | 12.129 | 0.000 |
| **10** | 14.767 | 7.429 | 4.915 | 3.558 | 2.373 | 2.825 | 3.803 | 0.859 | 0.020 | 0.052 | 0.000 | 0.000 | 0.000 |
| **11** | 14.767 | 12.781 | 9.329 | 7.142 | 2.070 | 0.368 | 0.066 | 0.022 | 0.001 | 0.005 | 0.000 | 0.000 | 0.000 |
| **12** | 14.767 | 13.728 | 9.876 | 7.346 | 4.270 | 2.962 | 3.683 | 2.688 | 4.859 | 2.331 | 0.308 | 0.000 | 0.000 |
| **13** | 14.767 | 23.909 | 26.047 | 22.862 | 32.512 | 12.212 | 8.182 | 6.857 | 1.823 | 0.435 | 1.474 | 0.657 | 0.000 |
| **14** | 14.767 | 12.820 | 4.364 | 3.416 | 1.661 | 0.510 | 0.246 | 0.067 | 0.080 | 0.011 | 0.003 | 0.000 | 0.000 |
| **15** | 14.767 | 13.948 | 6.377 | 7.200 | 4.978 | 9.089 | 12.484 | 15.090 | 19.931 | 41.506 | 36.141 | 39.951 | 42.933 |
| **16** | 14.767 | 11.612 | 18.800 | 25.338 | 42.373 | 38.982 | 34.870 | 28.680 | 38.734 | 29.414 | 16.006 | 23.101 | 27.200 |
| **17** | 14.767 | 21.657 | 17.068 | 10.340 | 3.339 | 1.500 | 2.578 | 3.534 | 2.405 | 1.310 | 1.094 | 0.732 | 0.000 |
| **18** | 14.767 | 25.007 | 26.201 | 20.705 | 12.847 | 11.535 | 20.242 | 6.166 | 7.652 | 6.044 | 11.061 | 16.937 | 6.905 |
| **19** | 14.767 | 15.064 | 16.680 | 14.819 | 5.599 | 1.862 | 0.899 | 0.353 | 3.219 | 8.562 | 12.217 | 26.646 | 61.697 |

```python
Call.mean(axis = 0)
```

array([14.767 , 13.991 , 12.8161 , 11.38465, 9.91385, 8.47775, 9.6268 , 8.0402 , 9.3894 , 10.13545, 7.9891 , 10.11575, 11.033 ])

```python
plot(Call.mean(axis = 0), ls = "--", color = "black", linewidth= 3)
plot(Call.T) # Simulation des trajectoires des cours du sous-jacent.
plt.xlabel("Mounth", fontsize=22)
plt.ylabel("Call value", fontsize=22)
plt.title("Simulation of the Call trajectories", fontsize=30 )
plt.legend(["Mean values"], fontsize=22 )
plt.show()
```




png

The call values are consistent. The higher the share price is above the strike, the more expensive the option is and vice versa.

**2. Deltas between 0 and M**

In the case of a call the Black&Scholes formula allows to obtain,

$$\triangle_{t,call} = N(d1)$$

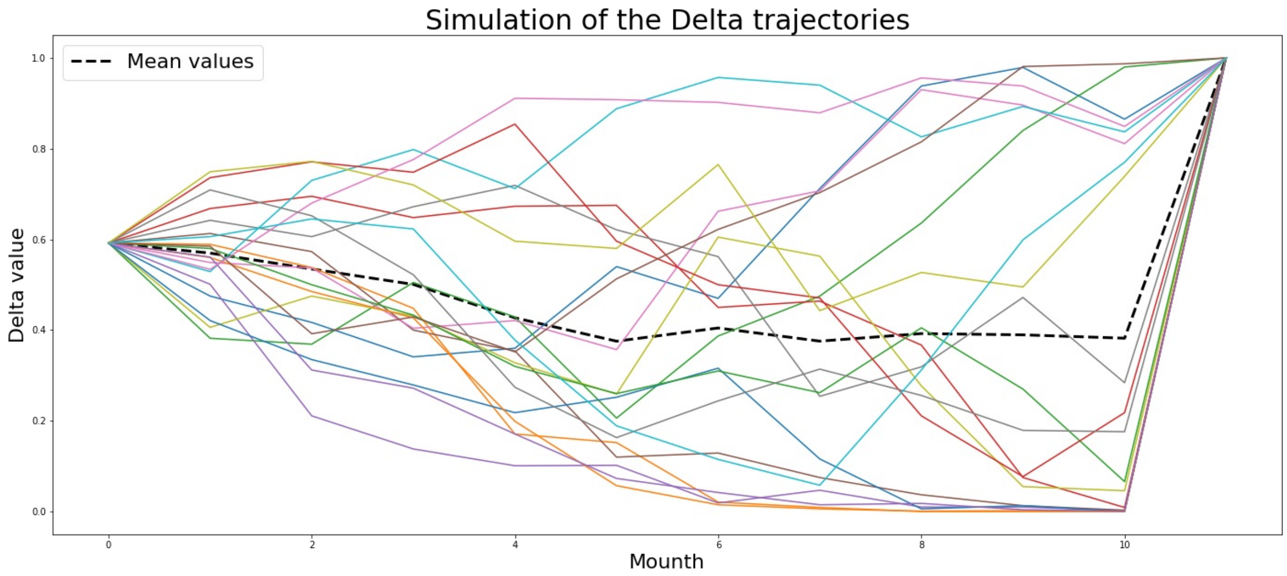
with,

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}(ln\frac{S_t}{K} + (r + \frac{1}{2}\sigma^2 2)(T-t))$$

```
Delta = np.zeros((20,12))

for j in range(20):
    for i in range(T-1):
        Delta[j][i] = np.round(cdf(d1(mat_path_spot_fut[j][i],strike,maturity,int_rate,volatility,i/12)),3)
        Delta[j][T-1]=1

D = pd.DataFrame(data = Delta)
D
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.592 | 0.475 | 0.417 | 0.341 | 0.360 | 0.540 | 0.470 | 0.712 | 0.938 | 0.979 | 0.865 | 1.0 |
| **1** | 0.592 | 0.589 | 0.538 | 0.448 | 0.171 | 0.152 | 0.021 | 0.009 | 0.000 | 0.000 | 0.000 | 1.0 |
| **2** | 0.592 | 0.382 | 0.369 | 0.505 | 0.429 | 0.206 | 0.387 | 0.475 | 0.636 | 0.840 | 0.980 | 1.0 |
| **3** | 0.592 | 0.668 | 0.695 | 0.648 | 0.673 | 0.675 | 0.450 | 0.464 | 0.367 | 0.075 | 0.009 | 1.0 |
| **4** | 0.592 | 0.501 | 0.211 | 0.138 | 0.101 | 0.102 | 0.019 | 0.047 | 0.010 | 0.010 | 0.002 | 1.0 |
| **5** | 0.592 | 0.613 | 0.573 | 0.400 | 0.354 | 0.120 | 0.129 | 0.075 | 0.037 | 0.013 | 0.003 | 1.0 |
| **6** | 0.592 | 0.549 | 0.537 | 0.404 | 0.421 | 0.357 | 0.662 | 0.707 | 0.930 | 0.896 | 0.811 | 1.0 |
| **7** | 0.592 | 0.642 | 0.606 | 0.672 | 0.719 | 0.621 | 0.562 | 0.254 | 0.319 | 0.472 | 0.284 | 1.0 |
| **8** | 0.592 | 0.406 | 0.475 | 0.431 | 0.328 | 0.259 | 0.605 | 0.563 | 0.277 | 0.055 | 0.046 | 1.0 |
| **9** | 0.592 | 0.529 | 0.730 | 0.798 | 0.712 | 0.888 | 0.957 | 0.940 | 0.826 | 0.893 | 0.837 | 1.0 |
| **10** | 0.592 | 0.421 | 0.335 | 0.279 | 0.218 | 0.252 | 0.316 | 0.116 | 0.006 | 0.013 | 0.000 | 1.0 |
| **11** | 0.592 | 0.560 | 0.485 | 0.427 | 0.199 | 0.057 | 0.015 | 0.006 | 0.001 | 0.002 | 0.000 | 1.0 |
| **12** | 0.592 | 0.580 | 0.500 | 0.433 | 0.320 | 0.260 | 0.310 | 0.262 | 0.405 | 0.270 | 0.066 | 1.0 |
| **13** | 0.592 | 0.736 | 0.771 | 0.748 | 0.854 | 0.596 | 0.500 | 0.471 | 0.211 | 0.077 | 0.218 | 1.0 |
| **14** | 0.592 | 0.561 | 0.312 | 0.272 | 0.171 | 0.073 | 0.042 | 0.015 | 0.018 | 0.004 | 0.001 | 1.0 |
| **15** | 0.592 | 0.585 | 0.392 | 0.429 | 0.352 | 0.513 | 0.622 | 0.703 | 0.815 | 0.981 | 0.987 | 1.0 |
| **16** | 0.592 | 0.534 | 0.680 | 0.776 | 0.911 | 0.908 | 0.902 | 0.879 | 0.956 | 0.938 | 0.849 | 1.0 |
| **17** | 0.592 | 0.709 | 0.652 | 0.522 | 0.274 | 0.163 | 0.244 | 0.314 | 0.256 | 0.179 | 0.176 | 1.0 |
| **18** | 0.592 | 0.749 | 0.772 | 0.720 | 0.596 | 0.580 | 0.765 | 0.443 | 0.527 | 0.495 | 0.739 | 1.0 |
| **19** | 0.592 | 0.606 | 0.645 | 0.623 | 0.378 | 0.189 | 0.115 | 0.058 | 0.312 | 0.599 | 0.770 | 1.0 |

```
plot(Delta.mean(axis = 0), ls = "--", color = "black", linewidth= 3)
plot(Delta.T) # Simulation des trajectoires des cours du sous-jacent.
plt.xlabel("Mounth", fontsize=22)
plt.ylabel("Delta value", fontsize=22)
plt.title("Simulation of the Delta trajectories", fontsize=30 )
plt.legend(["Mean values"], fontsize=22 )
plt.show()
```




png

The delta values are consistent. They are well within the range of 0 to 1. The following remarks can also be made: - the more the call is out of the money, the closer the delta is to 0. - the closer the call is to the currency, the closer the delta is to 0.5 - the closer the call is to the currency, the closer the delta is to 1

**3. The unhedged PnL**

Sans couverture ni appel de marge la perte et profit réalisé s'écrirait,

$$P_L^{unhedged}(t_0, t_M) = (nC_0 - \psi(nC_0) + I(K < S_M)((nK - \varphi nK) - (nS_M + \varphi(nS_M))$$

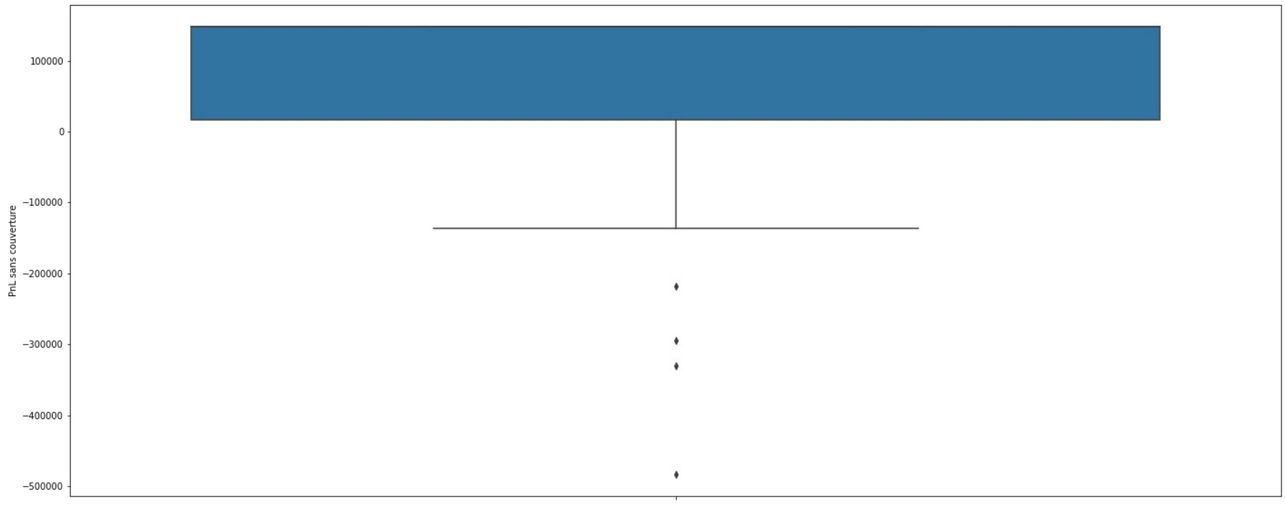
```
def psi(x):
  return 0.002*x

def phi(x):
  return 0.005*x

PnL = zeros((20,5))

for i in range (20):
  PnL[i][0]=i+1
  PnL[i][1]=spot_init
  PnL[i][2]=Call[i][12]
  PnL[i][3]=mat_path_spot_fut[i][12]
  PnL[i][4]=np.round_(n*Call[i][0]-psi(n*Call[i][0])+1*(strike<mat_path_spot_fut[i][12])*((n*strike-phi(n*strike))
-(n*mat_path_spot_fut[i][12]+phi(n*mat_path_spot_fut[i][12]))),3)
```

```
P1 = pd.DataFrame(data = PnL, columns = ["N° traj", "Spot init", "Call à maturité", "S à maturité", "PnL sans couv
erture" ])
P1
```

| | N° traj | Spot init | Call à maturité | S à maturité | PnL sans couverture |
|---|---|---|---|---|---|
| **0** | 1.0 | 100.0 | 0.000 | 96.361 | 147374.66 |
| **1** | 2.0 | 100.0 | 0.000 | 44.160 | 147374.66 |
| **2** | 3.0 | 100.0 | 46.582 | 146.582 | -330774.44 |
| **3** | 4.0 | 100.0 | 0.000 | 91.187 | 147374.66 |
| **4** | 5.0 | 100.0 | 0.000 | 72.788 | 147374.66 |
| **5** | 6.0 | 100.0 | 0.000 | 83.668 | 147374.66 |
| **6** | 7.0 | 100.0 | 35.343 | 135.343 | -217822.49 |
| **7** | 8.0 | 100.0 | 0.000 | 94.764 | 147374.66 |
| **8** | 9.0 | 100.0 | 0.000 | 74.408 | 147374.66 |
| **9** | 10.0 | 100.0 | 0.000 | 92.560 | 147374.66 |
| **10** | 11.0 | 100.0 | 0.000 | 53.372 | 147374.66 |
| **11** | 12.0 | 100.0 | 0.000 | 67.196 | 147374.66 |
| **12** | 13.0 | 100.0 | 0.000 | 68.480 | 147374.66 |
| **13** | 14.0 | 100.0 | 0.000 | 88.094 | 147374.66 |
| **14** | 15.0 | 100.0 | 0.000 | 53.993 | 147374.66 |
| **15** | 16.0 | 100.0 | 42.933 | 142.933 | -294101.99 |
| **16** | 17.0 | 100.0 | 27.200 | 127.200 | -135985.34 |
| **17** | 18.0 | 100.0 | 0.000 | 98.969 | 147374.66 |
| **18** | 19.0 | 100.0 | 6.905 | 106.905 | 67979.41 |
| **19** | 20.0 | 100.0 | 61.697 | 161.697 | -482680.19 |

```
sns.boxplot(y="PnL sans couverture", data=P1)
```

<matplotlib.axes._subplots.AxesSubplot at 0x119a37a90>



png

On peut remarquer que sans couverture et appels de marge, les pertes sont très élevées. Le gain est en revanche presque toujours identique puisque les profits se font sur la vente des Calls en 0.

**4. Number of shares to be held at each time in the case of the hedging**

To neutralize the uncertainties resulting from the unknown price $S_{m+1}$ one should choose $n_m$ so that: $|(n \triangle_m)| = n_m$.

```
N_actions = zeros((20,12))

N_actions= int_(n * Delta)

Na = pd.DataFrame(data = N_actions)
Na
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 5920 | 4750 | 4170 | 3410 | 3600 | 5400 | 4700 | 7120 | 9380 | 9790 | 8650 | 10000 |
| **1** | 5920 | 5890 | 5380 | 4480 | 1710 | 1520 | 210 | 90 | 0 | 0 | 0 | 10000 |
| **2** | 5920 | 3820 | 3690 | 5050 | 4290 | 2060 | 3870 | 4750 | 6360 | 8400 | 9800 | 10000 |
| **3** | 5920 | 6680 | 6949 | 6480 | 6730 | 6750 | 4500 | 4640 | 3670 | 750 | 90 | 10000 |
| **4** | 5920 | 5010 | 2110 | 1380 | 1010 | 1019 | 190 | 470 | 100 | 100 | 20 | 10000 |
| **5** | 5920 | 6130 | 5729 | 4000 | 3540 | 1200 | 1290 | 750 | 370 | 130 | 30 | 10000 |
| **6** | 5920 | 5490 | 5370 | 4040 | 4210 | 3570 | 6620 | 7070 | 9300 | 8960 | 8110 | 10000 |
| **7** | 5920 | 6420 | 6060 | 6720 | 7190 | 6210 | 5620 | 2540 | 3190 | 4720 | 2839 | 10000 |
| **8** | 5920 | 4060 | 4750 | 4310 | 3280 | 2590 | 6050 | 5629 | 2770 | 550 | 460 | 10000 |
| **9** | 5920 | 5290 | 7300 | 7980 | 7120 | 8880 | 9570 | 9400 | 8260 | 8930 | 8370 | 10000 |
| **10** | 5920 | 4210 | 3350 | 2790 | 2180 | 2520 | 3160 | 1160 | 60 | 130 | 0 | 10000 |
| **11** | 5920 | 5600 | 4850 | 4270 | 1990 | 570 | 150 | 60 | 10 | 20 | 0 | 10000 |
| **12** | 5920 | 5800 | 5000 | 4330 | 3200 | 2600 | 3100 | 2620 | 4050 | 2700 | 660 | 10000 |
| **13** | 5920 | 7360 | 7710 | 7480 | 8540 | 5960 | 5000 | 4710 | 2110 | 770 | 2180 | 10000 |
| **14** | 5920 | 5610 | 3120 | 2720 | 1710 | 730 | 420 | 150 | 180 | 40 | 10 | 10000 |
| **15** | 5920 | 5850 | 3920 | 4290 | 3520 | 5130 | 6220 | 7030 | 8149 | 9810 | 9870 | 10000 |
| **16** | 5920 | 5340 | 6800 | 7760 | 9110 | 9080 | 9020 | 8790 | 9560 | 9380 | 8490 | 10000 |
| **17** | 5920 | 7090 | 6520 | 5220 | 2740 | 1630 | 2440 | 3140 | 2560 | 1790 | 1760 | 10000 |
| **18** | 5920 | 7490 | 7720 | 7200 | 5960 | 5800 | 7650 | 4430 | 5270 | 4950 | 7390 | 10000 |
| **19** | 5920 | 6060 | 6450 | 6230 | 3780 | 1890 | 1150 | 580 | 3120 | 5990 | 7700 | 10000 |

**Question 5 : Amount of cash Bm to be held at each time**

The formulas used in this calculation are explained above.

```
M =12
S = mat_path_spot_fut;
s = np.zeros((20,13))
beta = np.zeros((20,13))
I = np.zeros((20,13))
B = np.zeros((20,13))

s[:,1] = (N_actions[:,0] -N_actions[:,1])*S[:,1] - phi(abs(N_actions[:,0] -N_actions[:,1])*S[:,1])
I[:,0] = Delta[:,0]*abs(B[:,0])*(- int_rate * (B[:,0]<0)+ int_rate * (B[:,0]>0))

B[:,0] = -(λ*n*Call[:,0] + psi(n*Call[:,0])) + (N_actions[:,0]*S[:,0]+ phi(N_actions[:,0]*S[:,0]) )-beta[:,0]
B[:,1] = -(1+λ)*n*(Call[:,1]-Call[:,0]) + s[:,1] + I[:,0] + beta[:,1]
```

```
for i in range(2,M-1):
  s[:,i] = (N_actions[:,i-1] -N_actions[:,i])*S[:,i] - phi(abs(N_actions[:,i-1] -N_actions[:,i])*S[:,i])
  I[:,i-1] = Delta[:,i-1]*abs(B[:,i-1])*(- int_rate * (B[:,i-1]<0)+ int_rate * (B[:,i-1]>0))
  B[:,i] = -(1+λ)*n*(Call[:,i-1]-Call[:,i-1]) + s[:,i] + I[:,i-1] + beta[:,i]
```

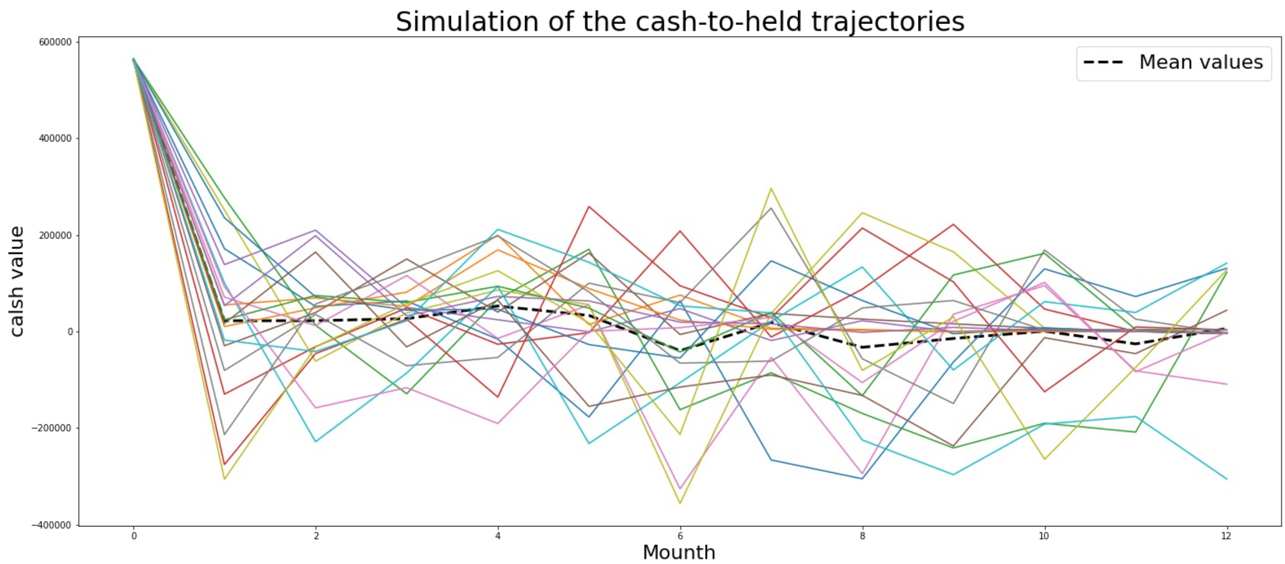
```
s[:,M-1] = 0
I[:,M-2] = Delta[:,M-2]*abs(B[:,M-2])*(- int_rate * (B[:,M-2]<0)+ int_rate * (B[:,M-2]>0))
B[:,M-1] = -(1+λ)*n*(Call[:,M-1]-Call[:,M-2]) + s[:,M-1] + I[:,M-2] + beta[:,M-1]
```

```
I[:,M-1] = Delta[:,M-1]*abs(B[:,M-1])*(- int_rate * (B[:,M-1]<0)+ int_rate * (B[:,M-1]>0))
s[:,M]= -phi(n*S[:,M])*(S[:,M]<= strike) - phi(n*strike)*(S[:,M]>= strike)
B[:,M] = -n*(Call[:,M]-Call[:,M-1]) + λ*n*Call[:,M-1] + s[:,M] + I[:,M-1]
```

```
Bm = pd.DataFrame(data = np.around(B,3))
Bm
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 565130.66 | 171708.667 | 52064.578 | 63254.021 | -15895.405 | -177278.862 | 63718.673 | -266246.192 | -305229.197 | -63288.661 | 129790.607 | 72421.377 | 13100 |
| **1** | 565130.66 | 10259.418 | 49001.992 | 81513.595 | 199707.855 | 14382.320 | 75557.795 | 6798.849 | 4107.002 | 0.000 | 0.000 | 0.000 | -2208 |
| **2** | 565130.66 | 277216.809 | 12980.749 | -129010.763 | 66783.727 | 170448.160 | -162019.052 | -85525.850 | -169761.186 | -241290.590 | -190137.874 | -208434.702 | 12253 |
| **3** | 565130.66 | -129911.899 | -31472.762 | 48814.231 | -26411.666 | -2519.427 | 208308.540 | -11450.498 | 87596.920 | 221979.239 | 46465.487 | 32.364 | -4246 |
| **4** | 565130.66 | 138589.520 | 210105.538 | 49987.179 | 24343.459 | -565.416 | 47423.885 | -18632.327 | 22387.358 | 4.477 | 5236.646 | 72.209 | -3637 |
| **5** | 565130.66 | -29474.355 | 39182.241 | 150297.110 | 40162.716 | 162573.847 | -6173.056 | 37517.702 | 25693.725 | 15896.109 | 6665.160 | 96.400 | -4169 |
| **6** | 565130.66 | 71295.472 | 12280.328 | 115813.854 | -14352.395 | 54943.886 | -326007.782 | -53770.333 | -294752.128 | 35834.189 | 95257.047 | -81518.931 | -1091 |
| **7** | 565130.66 | -80723.648 | 35429.701 | -70980.143 | -53756.563 | 100239.540 | 59842.545 | 255840.420 | -56527.980 | -149208.244 | 168779.351 | 25570.667 | -3170 |
| **8** | 565130.66 | 251980.869 | -61464.920 | 38504.181 | 85806.446 | 55657.841 | -356246.216 | 37649.714 | 246138.175 | 165065.184 | 7121.607 | 630.552 | -1967 |
| **9** | 565130.66 | 100446.391 | -228562.421 | -86171.242 | 93742.608 | -232293.516 | -106002.353 | 21216.385 | 133833.383 | -79813.876 | 61816.724 | 39446.812 | 14170 |
| **10** | 565130.66 | 235740.159 | 71798.928 | 44381.169 | 46276.307 | -27049.835 | -54992.286 | 146472.659 | 64149.167 | -4670.366 | 7515.185 | 0.000 | -2668 |
| **11** | 565130.66 | 54933.312 | 69521.739 | 52010.868 | 169196.252 | 88976.825 | 23488.371 | 4905.432 | 2505.750 | -593.262 | 987.215 | 0.000 | -3359 |
| **12** | 565130.66 | 24329.196 | 74666.031 | 60365.836 | 93738.129 | 48532.457 | -42410.384 | 39714.086 | -133025.763 | 117149.983 | 162015.919 | 3909.861 | -3345 |
| **13** | 565130.66 | -275712.312 | -45721.866 | 25653.683 | -136028.951 | 259012.540 | 94802.868 | 28393.462 | 214364.999 | 102752.741 | -124938.935 | 9259.266 | 3664. |
| **14** | 565130.66 | 53514.987 | 198517.042 | 32391.400 | 72702.446 | 63316.160 | 19430.097 | 15926.580 | -1918.852 | 8520.569 | 1903.828 | 36.038 | -2698 |
| **15** | 565130.66 | 16773.428 | 164595.492 | -31838.060 | 64855.709 | -155159.375 | -115158.332 | -90176.230 | -132433.891 | -237238.549 | -12838.416 | -45973.430 | 44162 |
| **16** | 565130.66 | 92999.597 | -158179.468 | -116409.610 | -190662.207 | 588.093 | 7902.303 | 28973.081 | -105922.887 | 20947.911 | 101613.119 | -83414.609 | -1456 |
| **17** | 565130.66 | -213896.629 | 57016.788 | 124519.807 | 198142.906 | 82060.784 | -65594.011 | -61274.856 | 48815.854 | 64177.493 | 2807.243 | 4353.881 | 3922. |
| **18** | 565130.66 | -306207.236 | -32012.007 | 57578.193 | 126075.865 | 17526.304 | -213240.454 | 296632.073 | -80689.315 | 30278.229 | -264780.613 | -74425.457 | 12770 |
| **19** | 565130.66 | -17858.135 | -41481.720 | 22120.160 | 211648.044 | 143354.038 | 53096.763 | 38592.474 | -224914.473 | -296659.954 | -191960.341 | -176104.189 | -3057 |

```
plot(Bm.values.mean(axis = 0), ls = "--", color = "black", linewidth= 3)
plot(Bm.values.T) # Simulation des trajectoires des cours du sous-jacent.
plt.xlabel("Mounth", fontsize=22)
plt.ylabel("calsh value", fontsize=22)
plt.title("Simulation of the cash-to-held trajectories", fontsize=30 )
plt.legend(["Mean values"], fontsize=22 )
plt.show()
```




png

**6. PnLs with delta hedging**

Taking into account the hedging, we obtain the sum of the instantaneous P&Ls:

- $P_L = nC_0 - C_M - n(\sum_{m=1}^{M} \lambda_m (C_m - C_{m-1}) + n \sum_{m=1}^{M} (\lambda_{m-1} - \lambda_m) C_{m-1} + \sum_{m=1}^{M} (n_{m-1} - n_m) S_m - \varphi(|n_{m-1} - n_m|) S_m$

```
PnL_2 = zeros((20,5))
alpha = np.zeros((20,13))
for i in range(M):
  alpha[:,i] = (Call[:,i+1]-Call[:,i]) + Delta[:,i]*(S[:,i+1] - S[:,i])
```

```python
for i in range (20):
  PnL_2[i][0]=i+1
  PnL_2[i][1]=spot_init
  PnL_2[i][2]=Call[i][12]
  PnL_2[i][3]=mat_path_spot_fut[i][12]
  PnL_2[i][4]=(-N_actions[i][M-1]*Delta[i][M-1] + int_(N_actions[i][M-1]*Delta[i][M-1]))*(S[i][M] - S[i][M-1]) - N_actions[i][M-1]*alpha[i][M-1]+λ*N_actions[i][M-1]*Call[i][M-1] + I[i][M-1] - phi(int_(N_actions[i][M-1]*Delta[i][M-1])*S[i][M-1])-psi(int_(N_actions[i][M-1]*Delta[i][M-1])*S[i][M-1])


P2 = pd.DataFrame(data = np.around(PnL_2,3) , columns = ["N° de traj", "Spot init", "Call à maturité", "S à maturité", "PnL avec couverture en Delta" ])
P2
```

| | N° de traj | Spot init | Call à maturité | S à maturité | PnL avec couverture en Delta |
|---|---|---|---|---|---|
| **0** | 1.0 | 100.0 | 0.000 | 96.361 | 265457.778 |
| **1** | 2.0 | 100.0 | 0.000 | 44.160 | -51341.070 |
| **2** | 3.0 | 100.0 | 46.582 | 146.582 | 147515.416 |
| **3** | 4.0 | 100.0 | 0.000 | 91.187 | -131962.983 |
| **4** | 5.0 | 100.0 | 0.000 | 72.788 | -91762.756 |
| **5** | 6.0 | 100.0 | 0.000 | 83.668 | -131506.982 |
| **6** | 7.0 | 100.0 | 35.343 | 135.343 | -259893.219 |
| **7** | 8.0 | 100.0 | 0.000 | 94.764 | -132199.857 |
| **8** | 9.0 | 100.0 | 0.000 | 74.408 | 87453.201 |
| **9** | 10.0 | 100.0 | 0.000 | 92.560 | 324774.196 |
| **10** | 11.0 | 100.0 | 0.000 | 53.372 | -1184.030 |
| **11** | 12.0 | 100.0 | 0.000 | 67.196 | -75941.540 |
| **12** | 13.0 | 100.0 | 0.000 | 68.480 | -8578.173 |
| **13** | 14.0 | 100.0 | 0.000 | 88.094 | 14374.685 |
| **14** | 15.0 | 100.0 | 0.000 | 53.993 | 26140.301 |
| **15** | 16.0 | 100.0 | 42.933 | 142.933 | 7877.721 |
| **16** | 17.0 | 100.0 | 27.200 | 127.200 | -48535.932 |
| **17** | 18.0 | 100.0 | 0.000 | 98.969 | -88747.442 |
| **18** | 19.0 | 100.0 | 6.905 | 106.905 | 220202.591 |
| **19** | 20.0 | 100.0 | 61.697 | 161.697 | -662141.024 |