Python for Finance

Simulation & Option Pricing

Andras Niedermayer

DAUPHINE
UNIVERSITÉ PARIS

Outline

Basic issue

1. We work with random numbers generated by functions from the `numpy.random` sublibrary.

```
import numpy as np
import numpy.random as npr
```

2. Easiest function: `npr.rand(x)` returns an array of size x from the Uniform $[0, 1]$ distribution.
3. You can extend the array to multiple dimensions: `npr.rand(x,y,z,...)`
4. To get random numbers between two different real numbers, a and b, you can transform `npr.rand(x)` as: `a+(b-a)*npr.rand(x,y,z,...)`
5. The transformation works perfectly well with multi-dimensional arrays!

Other random number functions

- `npr.randn`: standard normal random numbers.
- `npr.randint(low,high)`: random *integers* from "low" (inclusive) to "high" (exclusive)
- `npr.random_integers(low,high)`: random *integers* from "low" (inclusive) to "high" (inclusive)
- `npr.choice(v, size=x, replace=True or False)`: random sample from given vector *v*, of size *x*, with or without replacement.
- `npr.random`, `npr.ranf`, `npr.sample` all generate random floats in $[0,1)$ (different seed methods).
  You need to specify size explicitly, i.e., `size=?`.

Applications

Standard normals

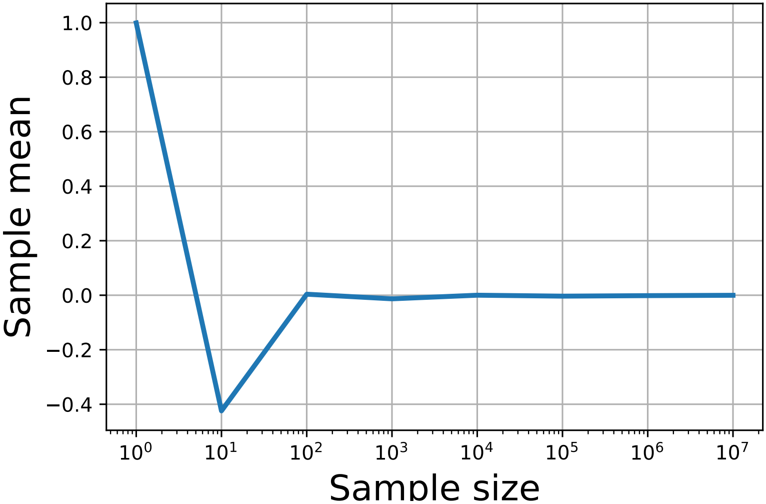
Draw from the standard normal distribution eight samples. The first sample has one observation, the second 10 observations, the third 10^2 observations, the eighth one, 10^7. Plot the means and standard deviation of each sample against the sample decade (log-10 scale). What do you observe? How do you interpret the observation?

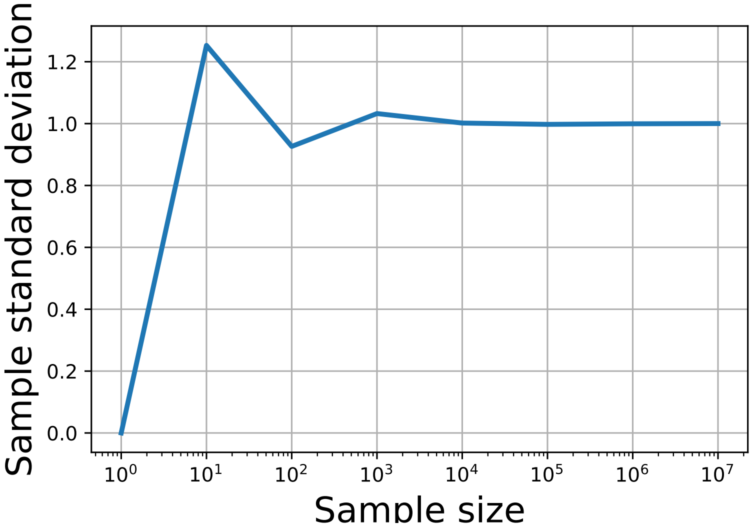
Sampling from a population

There are nine students in a class. One of them is 23, two are 24, three are 25, one is 26, and two are 27. Draw a 10,000-large sample from their age distribution. Plot a histogram of the empirical frequencies. What do you expect the height of the third column to be relative to the fifth?

Solution (1)

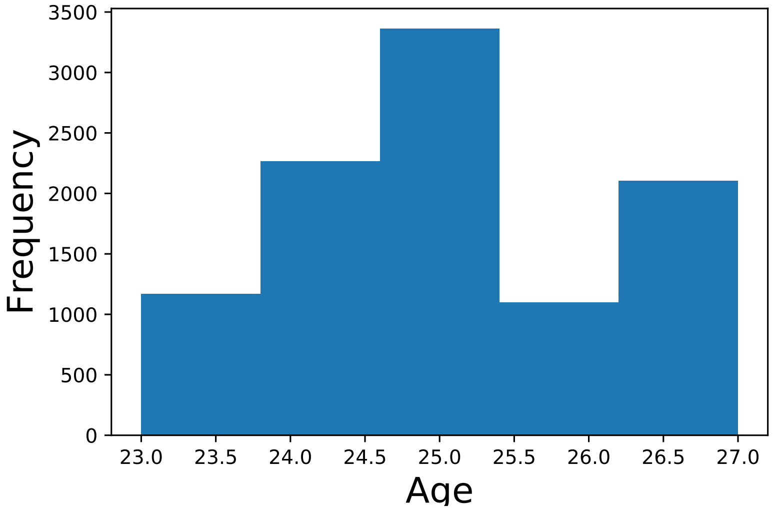
```
exponents=np.arange(0,8)
obs=np.zeros(8)
means=np.zeros(8)
stds=np.zeros(8)
for exponent in exponents:
    sample_size = 10**exponent
    obs[exponent]=sample_size
    random_sample = npr.randn(sample_size)
    means[exponent] = random_sample.mean()
    stds[exponent] = random_sample.std()
plt.semilogx(10**exponents, means,lw=2.5)
plt.xlabel("Sample␣size",fontsize=18)
plt.ylabel("Sample␣mean",fontsize=18)
plt.grid()
```


Sample mean
Sample size
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7


Sample standard deviation
0.0
0.2
0.4
0.6
0.8
1.0
1.2
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
Sample size

Solution (2)

```
ages=[23,24,24,25,25,25,26,27,27]
age_sample=npr.choice(ages, 10000)
plt.hist(age_sample, bins=5)
plt.xlabel("Age", fontsize=18)
plt.ylabel("Frequency", fontsize=18)
```



3500
3000
2500
2000
1500
1000
500
0
Frequency
23.0
23.5
24.0
24.5
25.0
25.5
26.0
26.5
27.0
Age

Yet some more distributions to draw from...

Function	Parameters	Distribution
`beta`	a, b, size	beta distribution
`binomial`	n, p, size	binomial distribution
`chisquare`	df, size	χ^2 distribution
`f`	dfnum, dfden, size	F-distribution
`lognormal`	mean, sigma, size	Log-normal distribution
`standard_t`	df, size	Student-t distribution

How to draw from any distribution (1/3)

We define an arbitrary cumulative distribution function:

$$F(x) = \frac{1}{1 + \exp(-x)} \tag{1}$$

We quickly see that $1 > F(x) > 0$, also F is non-decreasing. Further,

$$\lim_{x \to -\infty} F(x) = 0$$
$$\lim_{x \to +\infty} F(x) = 1$$

Therefore, F satisfies all properties of a CDF. We want do draw a 10,000-large sample from this distribution.

How to draw from any distribution (2/3)

First, we note that the CDF returns always a number between zero and one:

$$F(x) = \frac{1}{1 + \exp(-x)} = \mathbb{P}(X < x) = y \tag{2}$$

Moreover, the output of the CDF function is uniformly distributed:

$$\mathbb{P}(Y \leq y) = \mathbb{P}(F(X) < y) = \mathbb{P}\left(X < F^{-1}(y)\right) = F\left(F^{-1}(y)\right) = y$$

The algorithm becomes simple:

1. Draw a 10,000 sample from the uniform distribution (y values)
2. Compute x values as $F^{-1}(y)$:

$$x = -\log\left(\frac{1}{y} - 1\right) \tag{3}$$

How to draw from any distribution (3/3)

Thanks to vectorization, the solution is literally two short lines of code:

```
y=npr.rand(10000)
x=-np.log(1.0/y-1)
```

Outline

Simulate Black-Scholes stock prices

Let the stock price S_t follow a geometric brownian motion with drift r and volatility σ, i.e.,

$$dS_t = r \times dt + \sigma \times dz_t \tag{4}$$

The stock price at time T, S_T, is:

$$S_T = S_0 \exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}z\right), \tag{5}$$

where z is a standard normal variable.

Application

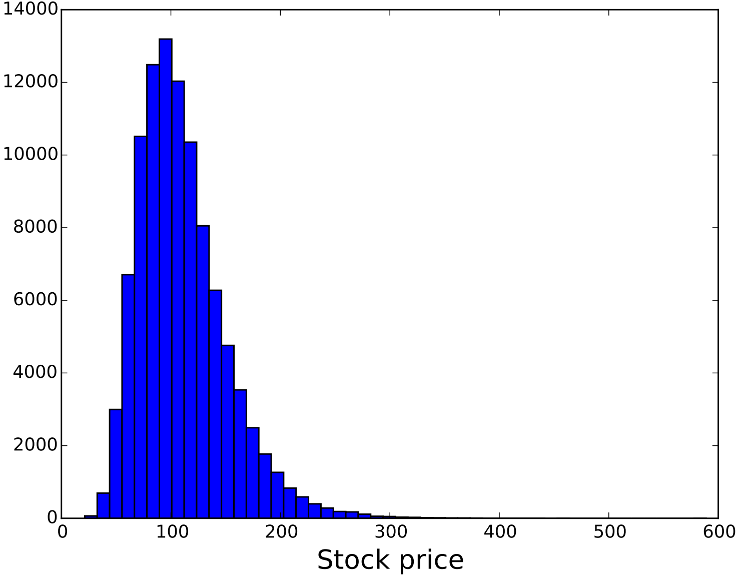
Let us simulate 100,000 possible stock prices in two-years time. The current stock price is 100. The risk-free rate is 5% per annum, and the annualised volatility is 25%.

1. Plot a histogram of the simulated stock prices. What distribution do they resemble?
2. What is the mean and variance of the simulated stock prices?
3. What happens with the mean and variance if we look at a three-years window?

Solution

```
S0=100
r=0.05
sigma=0.25
T=2
I=10**5
ST1=S0*np.exp((r-0.5*sigma**2)*T+
  sigma*np.sqrt(T)*npr.randn(I))

plt.clf()
plt.hist(ST1, bins=50)
plt.xlabel("Stock price", fontsize=18)
```


14000
12000
10000
8000
6000
4000
2000
0
0
100
200
300
400
500
600
Stock price

Outline

Again, the Black-Scholes model...

Consider the following dynamic for the stock price:

$$dS_t = rS_t \times dt + \sigma S_t \times dZ_t \tag{6}$$

In discrete time, from $t - \Delta t$ to t, we have (almost) the exact same expression as before:

$$S_t = S_{t-\Delta t} \exp\left(\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t} z\right), \tag{7}$$

Now though, we want a full **path** of stock prices, not just the end values.

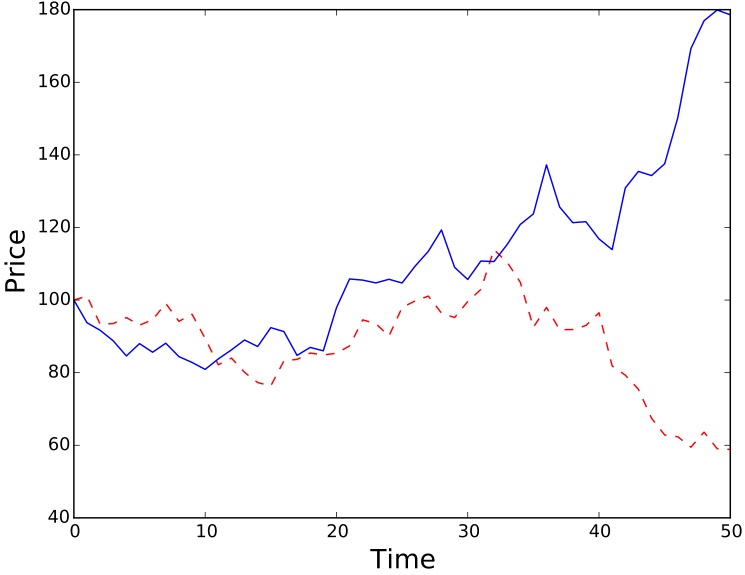
Application

Let us simulate 100,000 possible stock paths in two-years time, using 25 steps per year. The current stock price is 100. The risk-free rate is 5% per annum, and the annualised volatility is 25%.

1. Plot the first and the last path of the stock price.
2. How would the path look if the risk-free rate is 2%? or the volatility 50%?

Solution

```
S0=100
r=0.05
sigma=0.25
T=2
I=10**5
M=50
dt=np.float(T)/M
S=np.zeros((M+1,I))
S[0]=S0
for t in range(1,M+1):
        S[t]=S[t-1]*np.exp((r-0.5*sigma**2)*dt
          +sigma*np.sqrt(dt)*npr.randn(I))
```



180
160
140
120
100
80
60
40
Price
0
10
20
30
40
50
Time

The Merton (1976) model

Consider the following dynamic of the stock price, both including a Brownian motion and a Poisson jump process with intensity λ and mean jump size μ_J:

$$dS_t = (r - r_J) \times S_t dt + \sigma \times S_t dZ_t + J_t S_t dN_t. \tag{8}$$

1. $r_J = \lambda \left(\exp\left(\mu_J + \frac{\delta^2}{2} \right) - 1 \right)$
   is a drift correction to maintain the risk-neutral measure.
2. J_t is a jump at date t. The distribution of the jump is:

$$\log(1 + J_t) \approx \text{Normal}\left(\log(1 + \mu_J) - \frac{\delta^2}{2}, \delta^2 \right) \tag{9}$$

Discretization of the Merton model

$$S_t = S_{t-\Delta t} \left(e^{\left(r - r_J - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t} z_t^1} + \left(e^{\mu_J + \delta z_t^2} - 1 \right) y_t \right)$$

There are three sources of randomness, and thus we need to generate three sets of numbers:

1. The diffusion part of the stock price (Brownian motion): z_t^1.
2. The size of the jump, using normal variable: z_t^2.
3. The timing of the jump, using Poisson variable: y_t.

Application

Simulate the stock price from the Merton model with jumps, assuming the following parameters:

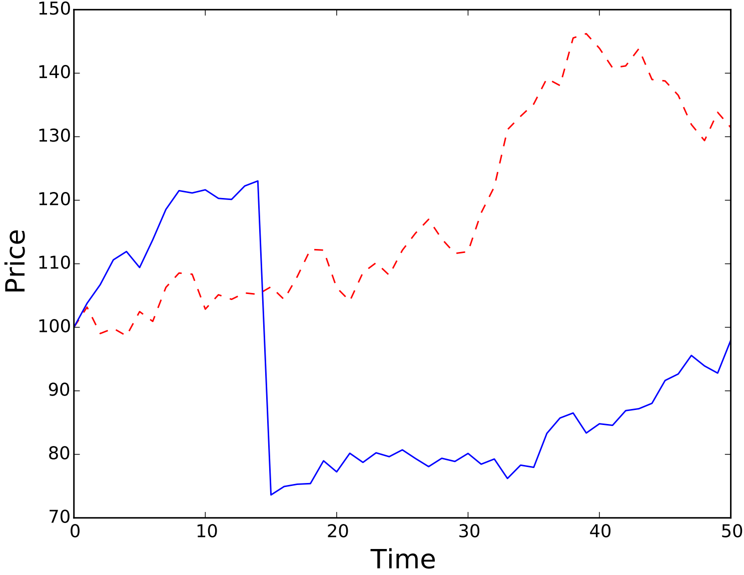
```
S0=100
r=0.05
sigma=0.2
lamb=0.75
mu=-0.6
delta=0.25
T=1.0
M=50
I=10**4
dt=T/M
```

Solution

```python
rj=lamb*(np.exp(mu+0.5*delta**2)-1)
S=np.zeros((M+1,I))
S[0]=S0

z1=npr.standard_normal((M+1,I))
z2=npr.standard_normal((M+1,I))
y=npr.poisson(lamb*dt, (M+1,I))

for t in range(1,M+1):
  S[t]=S[t-1]*(np.exp((r-rj-0.5*sigma**2)*dt
  +sigma*np.sqrt(dt)*z1[t])
  +(np.exp(mu+delta*z2[t])-1)*y[t])
  S[t]=np.maximum(S[t],0)
```


150
140
130
120
110
100
90
80
70
Price
0
10
20
30
40
50
Time

Outline

The Cox-Ross-Rubinstein model

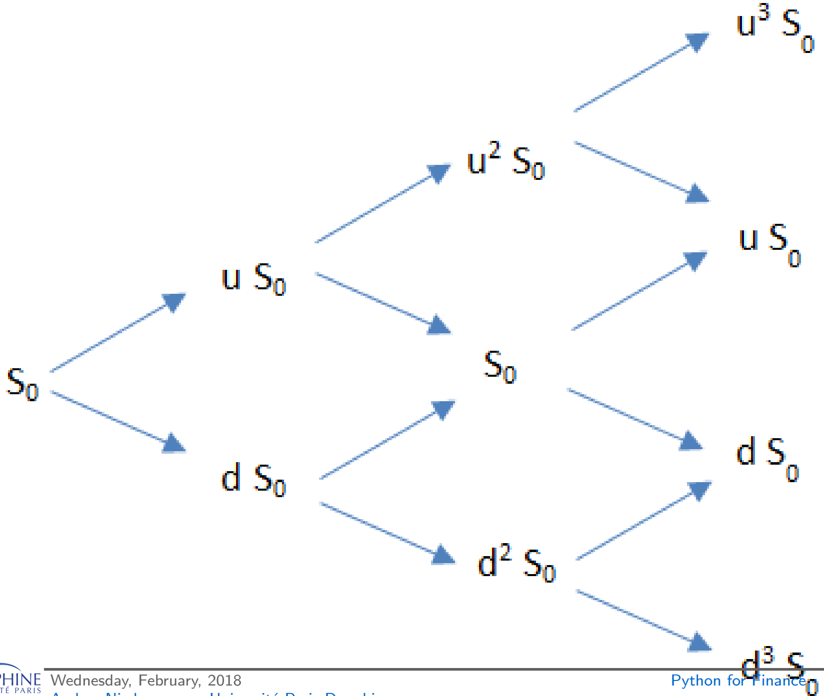
1. Consider a call (put) option on a stock maturing after time T.
2. The current stock price is S_0, strike price is K.
3. The volatility of the stock is σ, risk free rate r.
4. We want to compute the price of the option using a tree with N steps.

Up-and-down steps

- The length of a step is $\Delta t = \frac{T}{N}$.
- At each step the stock moves up by u, or down by d:

$$u = \exp\left(\sigma \Delta t\right) \tag{10}$$

$$d = u^{-1} \tag{11}$$


$u^3 S_0$
$u^2 S_0$
$u S_0$
$u S_0$
S_0
S_0
$d S_0$
$d S_0$
$d^2 S_0$
$d^3 S_0$

The Cox-Ross-Rubinstein model

Risk-neutral probabilities

The risk-neutral probability of an upward jump is p, where:

$$p = \frac{\exp(r\Delta t) - d}{u - d}. \tag{12}$$

The value of the option at step k is the discounted risk-neutral expectation at $t + 1$:

$$C_k = \exp(-r\Delta t)\left[pC_{k+1}^{u} + (1-p)\,C_{k+1}^{d}\right] \tag{13}$$

Background...

Remember that to price the option at step k we replicate the payoffs at step $k+1$ using Δ stocks and B bonds:

$$C_{k+1}^{u} = \Delta u S + \exp\left(r \Delta t\right) B$$
$$C_{k+1}^{d} = \Delta d S + \exp\left(r \Delta t\right) B$$

Therefore, solving this system:

$$\Delta = \frac{C_{k+1}^{u} - C_{k+1}^{d}}{\left(u - d\right) S}$$
$$B = \frac{u C_{k+1}^{d} - d C_{k+1}^{u}}{\left(u - d\right) \exp\left(r \Delta t\right)}$$

Background...(2)

The current value of the option is the current value of the stock and bond portfolio:

$$C_k = \Delta S + B. \tag{14}$$

Replacing the previously found values for Δ and B,

$$C_k = \exp\left(-r\Delta t\right)\left[pC_{k+1}^u + (1-p)\,C_{k+1}^d\right] \tag{15}$$

Extensions

1. Pricing put options is as simple as pricing call options. The recursive formula is the same:

$$P_k = \exp\left(-r\Delta t\right)\left[pP_{k+1}^u + (1-p)\,P_{k+1}^d\right] \tag{16}$$

2. For pricing American options, one compares the result from the recursive formula with the immediate execution payoff at each step.

Objective

Let us build a function `binomialtree` that prices European or American plain vanilla options.

Inputs

- T is the time to maturity (in years),
- N is the number of tree steps,
- S is the current stock price,
- r is the risk free rate (net, absolute terms, i.e., 0.03 for 3% p.a.),
- σ is annual volatility (absolute terms),
- K is the strike price,
- `typeEA` takes value ``European" or ``American",
- `typeCP` takes value ``call" or ``put".

```
def binomialtree(T,N,S,r,sigma,K,typeEA,typeCP):
```

Option payoff

1. The call option payoff is $\max(S-K,0)$ whereas the put option payoff is $\max(K-S,0)$.
2. We transform the `typeEA` variable from a string to a number: 1 for call options, -1 for put options.
3. Then we can write both the call **and** put option payoff together:

$$\text{OptionPayoff} = \texttt{typeEA} \times \max(S-K,0)\,. \tag{17}$$

```
if typeCP=="call":
    typeCP=1
else:
    typeCP=-1
```

Preliminary computations

1. The length of a step in the tree:

```
dt=np.float(T)/N
```

2. The upward and downward steps:

```
u=np.exp(sigma*np.sqrt(dt))
d=1/u
```

3. The risk-neutral probabilities:

```
p=(np.exp(r*dt)-d)/(u-d)
```

4. Initialise a vector of stock and a vector of option prices:

```
ST=np.zeros(N+1)
option=np.zeros(N+1)
```

Recursive solution: fill in the terminal values

1. The tree has N steps, hence $N+1$ terminal values for the stock and the option price.
2. Each possible terminal value is a combination of k downward steps and $N-k$ upward steps, where k varies from zero to N.
3. Terminal option prices are just computed using terminal stock prices (no expectation required).

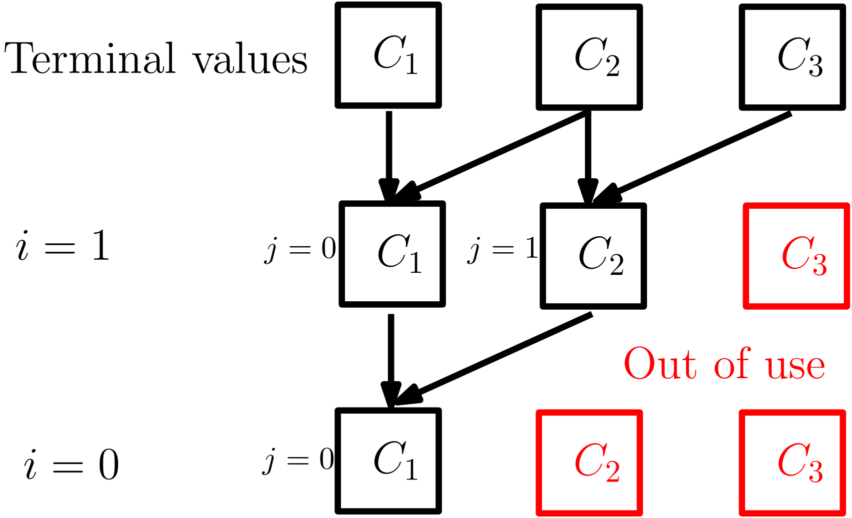
```
for i in range(0,N+1):
    ST[i]=S*u**(N-i)*d**i
    option[i]=max(typeCP*(ST[i]-K),0)
```

Moving backwards

Starting from the vector `option` of terminal values, loop:

1. Backwards over tree levels (i loop);
2. Forward over cells on a tree level (j loop)
3. At each new tree level, we overwrite the `option` vector with the discounted expected values of the option on the following level.

```
for i in range(N-1,-1,-1):
  for j in range(0,i+1):
    option[j]=
    np.exp(-r*dt)*(p*option[j]+(1-p)*option[j+1])
```


Terminal values
C_1
C_2
C_3
$i = 1$
$j = 0$
C_1
$j = 1$
C_2
C_3
Out of use
$i = 0$
$j = 0$
C_1
C_2
C_3

American options

For an American option, at each node j we:

1. Compute the stock price at that node.
2. Compare the European option value with the payoff on immediate exercise (using the stock price we just computed).
3. We keep the largest value of the two.

```
for i in range(N-1,-1,-1):
 for j in range(0,i+1):
    ....
   if typeEA=="American":
    ST[j]=np.exp(-r*dt)*(p*ST[j]+(1-p)*ST[j+1])
    option[j]=max(option[j], max(typeCP*(ST[j]-K)
```