Python Programming for Finance

Marius A. Zoican, PhD.

DAUPHINE
UNIVERSITÉ PARIS

Lecture 6: Option pricing

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:

$$u = \exp\left(\sigma \Delta t\right) \tag{1}$$

$$d = u^{-1} \tag{2}$$

$$\begin{tikzcd}
 & & & u^3 S_0 \\
 & & u^2 S_0 \arrow[ur] \arrow[dr] & \\
 & u S_0 \arrow[ur] \arrow[dr] & & u S_0 \\
S_0 \arrow[ur] \arrow[dr] & & S_0 \arrow[ur] \arrow[dr] & \\
 & d S_0 \arrow[ur] \arrow[dr] & & d S_0 \\
 & & d^2 S_0 \arrow[ur] \arrow[dr] & \\
 & & & d^3 S_0
\end{tikzcd}$$

The Cox-Ross-Rubinstein model

Risk-neutral probabilities

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

$$p = \frac{\exp\left(r\Delta t\right) - d}{u - d}. \tag{3}$$

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

$$C_k = \exp\left(-r\Delta t\right)\left[pC^u_{k+1} + \left(1-p\right)C^d_{k+1}\right] \tag{4}$$

Background...

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

$$
\begin{aligned}
C^u_{k+1} &= \Delta u S + \exp\left(r\Delta t\right) B \\
C^d_{k+1} &= \Delta d S + \exp\left(r\Delta t\right) B
\end{aligned}
$$

Therefore, solving this system:

$$
\begin{aligned}
\Delta &= \frac{C^u_{k+1} - C^d_{k+1}}{\left(u-d\right) S} \\
B &= \frac{u C^d_{k+1} - d C^u_{k+1}}{\left(u-d\right)\exp\left(r\Delta t\right)}
\end{aligned}
$$

Background...(2)

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

$$C_k = \Delta S + B. \tag{5}$$

Replacing the previously found values for Δ and B,

$$C_k = \exp\left(-r\Delta t\right)\left[pC^u_{k+1} + \left(1-p\right)C^d_{k+1}\right] \tag{6}$$

Extensions

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

$$P_k = \exp\left(-r\Delta t\right)\left[pP^u_{k+1} + \left(1-p\right)P^d_{k+1}\right] \tag{7}$$

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`".

```python
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:

$$\text{OptionPayoff} = \texttt{typeEA} \times \max(S-K,0)\,. \tag{8}$$

```python
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).

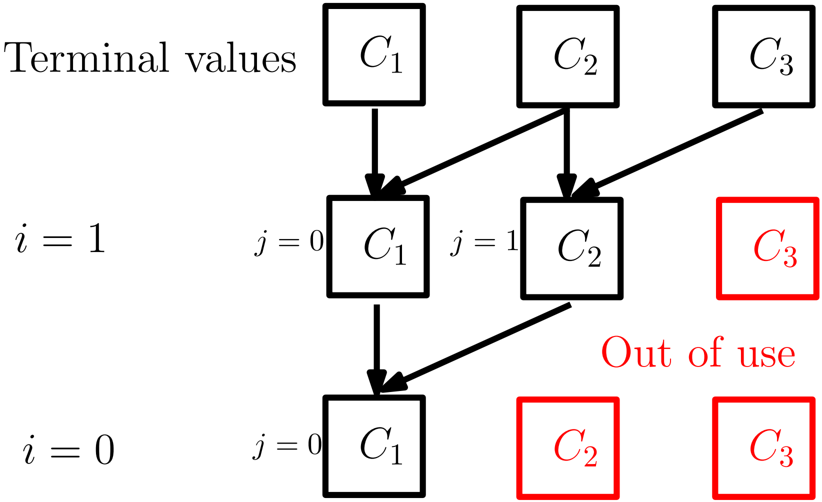
```
for i in range(0,N+1):
  ST[i]=S*u**(N-i)*d**i
  option[i]=max(typeCP*(ST[i]-K),0)
```

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)
```

Outline

Static Monte Carlo simulation

1. Simulate a large number (N) of terminal stock prices.
2. Compute the option value for each terminal stock price.
3. Take the average across all N.
4. The method is simple and intuitive. It requires two assumptions:
   - 4.1 The option is European (only exercised at maturity)
   - 4.2 The option payoff depends only on final stock prices.
5. Useful method for plain vanilla calls and puts.

Static Monte Carlo simulation

```python
def montecarlo(T,N,S,r,sigma,K,typeCP):
  if typeCP=="call":
    typeCP=1
  else:
    typeCP=-1

  z = npr.randn(N)
  ST = S*np.exp((r-0.5*sigma**2)*T
    +sigma*np.sqrt(T)*z)
  OptionT= np.maximum(typeCP*(ST - K), 0)
  option = np.exp(-r*T)*OptionT.mean()
  return option
```

Dynamic Monte Carlo simulation

Asian options

Let $A(0, T)$ be the average price of stock S between zero and T. An Asian call option pays off at T:

$$AC = \max\left(A(0, T) - K, 0\right). \tag{9}$$

An Asian put option pays off at T:

$$AP = \max\left(K - A(0, T), 0\right). \tag{10}$$

1. Asian option payoffs do not only depend on the final stock price.
2. To price them, we need to simulate *the entire path* of the stock price.

Dynamic Monte Carlo simulation

```
def montecarlo_dyn(T,N,steps,S,r,sigma,K,typeCP):
  if typeCP=="call":
    typeCP=1
  else:
    typeCP=-1
  dt=np.float(T)/steps
  SPath=np.zeros((steps+1,N))
  SPath[0]=S
  for t in range(1,steps+1):
   z = npr.randn(N)
   SPath[t]=SPath[t-1]*np.exp((r-0.5*sigma**2)*dt
      +sigma*np.sqrt(dt)*z)
    OptionT= np.maximum
       (typeCP*(SPath.mean(axis=0) - K), 0)
    option = np.exp(-r*T)*OptionT.mean()
    return option
```

Applications

- Monte Carlo simulation is a very powerful tool.
- How would you simulate an option on interest rates? on the VIX?
- How do you think the call (put) option value is if one allows for positive jumps? What about negative jumps?
- American options, however, need to consider the possibility of early exercise. Not as easy to implement.

Outline

Main idea

- American options can be exercised at any moment during their life.
- In practice, we only look at exercises at the specific steps in the tree (more like a Bermudan option).
- You need to account for the exercise possibility at any node in the tree.
- The value of a (call) option is:

$$V_0 = \sup_{\tau \in \{0, \Delta t, 2\Delta t, \ldots T\}} e^{-rT} \mathbb{E}\left[\max\left(S_\tau - K, 0\right)\right] \tag{11}$$

- That is, the value of the option is the *supremum* over all possible exercise dates.

Recursive framework

- The formula before can be cast into a recursive framework.
- At each potential exercise date t, the value of the option is the maximum of two choices:
  1. the value of immediate exercise
  2. the discounted expected value of the option at the next possible exercise moment.

$$V_t\left(S_t\right) = \max\left[\max\left(S_t - K, 0\right), e^{-r\Delta t}\mathbb{E}V_{t+\Delta t}\left(S_{t+\Delta t}\right)\right]$$

The issue

$$V_t\left(S_t\right) = \max\left[\max\left(S_t - K, 0\right), e^{-r\Delta t}\mathbb{E}V_{t+\Delta t}\left(S_{t+\Delta t}\right)\right]$$

- One can work backwards through this equation to get V_0.
- To get the option value at t, we need an *expectation* of the value at $t+1$.
- One solution would be to look at the value of the option on that particular path.
- However, that would imply perfect foresight, i.e., not really an expectation.

The issue

$$V_t\left(S_t\right) = \max\left[\max\left(S_t - K, 0\right), e^{-r\Delta t}\mathbb{E}V_{t+\Delta t}\left(S_{t+\Delta t}\right)\right]$$

- A **better** solution is to regress all continuation values, across **all paths** on functions of the stock price:

$$V_{t+1,i}\left(S_{t+1,i}\right)e^{-r\Delta t} = \beta_0 + \beta_1 S_{t,i} + \beta_2 S_{t,i}^2 + \ldots + \text{error}$$

- The fitted values of the regression are used as the expectation.
- You need to do as many regressions as steps in the tree.
- The number of observations is equal to the number of paths.
- Hence, the Least Squares Monte Carlo name of the approach.

Algorithm

- Define auxiliary variables.

```
if typeCP=="call":
  typeCP=1
else:
  typeCP=-1
dt = np.float(T) / steps
df = np.exp(-r * dt)
```

- Simulate the stock price paths.

```
SPth = np.zeros((steps + 1, N))
SPth[0] = S
z=npr.randn(steps,N)
for t in range(1, steps + 1):
 SPth[t]=SPth[t-1]*np.exp((r-0.5*sigma**2)*dt
      +sigma*np.sqrt(dt)*z[t-1])
```

Algorithm

- Compute immediate exercise values for **all** nodes in the tree.

```python
h = np.maximum(typeCP*(SPath - K), 0)
```

- Copy the exercise values into a **value** matrix.

```python
V = np.copy(h)
```

- For each level in the tree, regress discounted continuation values on stock prices polynomials:

```python
for t in range(steps-1, 0, -1):
 reg = np.polyfit(SPath[t], V[t + 1] * df, 7)
```

Algorithm

```
for t in range(steps - 1, 0, -1):
 reg = np.polyfit(SPath[t], V[t + 1] * df, 7)
 C = np.polyval(reg, SPath[t])
 V[t] = np.where(C > h[t], V[t + 1] * df, h[t])
option = df * 1 / N * np.sum(V[1])
return option
```

- Save the fitted continuation values of the regression (`C`)
- If the continuation value C is larger than the immediate exercise value h, update the value matrix with the (discounted) continuation value.
- If the continuation value C is smaller than the immediate exercise value h, update the value matrix with the immediate exercise value.
- The option value is the discounted average over all paths of the continuation values at the first possible exercise moment.

Outline

Discussion of the assignment
(if time permits).