IMPERIAL COLLEGE LONDON, DEPARTMENT OF MATHEMATICS

Scientific Computing (M3SC)

Peter J. Schmid

February 6, 2017

1 LATTICE-BASED OPTION PRICING (COX-ROSS-RUBINSTEIN ALGORITHM)

Options are financial instruments that are based on a contract for a buyer to possibly exercise his/her right to buy or sell an underlying asset for a specified price (the strike price) at or before a specified future time (expiration date). This contract has to be bought, and the question in this section is how to fairly price the contract, given the specified time, specified price, and the current price and volatility of the underlying asset. When buying a contract to buy shares at a later time, we refer to the option as a **call** option; a contract to sell shares at a later time is referred to as a **put** option.
As an example, let's say you wish to buy a car in a month and you want to buy insurance against a possible price drop. You are interested in a contract that promises you a locked-in price on the car in a month from now; this contract should saveguard against potential financial loss. The insurance contract becomes invalid after the expiration date. The price of the insurance contract depends on many variables: (i) the price at which you wish to buy the car (the strike price; the lower the price, the more expensive the insurance), (ii) the duration of the contract (the longer the contract, the more expensive the contract), (iii) the volatility of the market (the more uncertain the market, the more expensive the contract), and (iv) the current interest rate on a bank account (since, alternatively, we could save the money, earn interest and thus compensate for price fluctuations in the price of the car). The question is then: how to fairly determine the price of the insurance contract? "Fair" will be taken as a condition such that alternative financial arrangements won't present any advantage over the insurance contract. An additional condition has to be taken into account: do you have to wait until the end of the contract before buying the car, or can you

buy the car at any time before the expiration date? This is the question about the exercise right of your contract.
The techniques to determine this fair price can be used in many other applications where we have to specify the value of an instrument based on (i) an estimate of risk inherent in the development of the associated asset and (ii) the ambient financial environment.

1.1 Background

The pricing of an option is based on an arbitrage argument. It states that a proper balance of shares of the underlying asset and a risk-free asset should present neither favorable nor unfavorable odds for profit when compared to the option contract. There are various methods of computing the value of option contracts based on this arbitrage argument. Among them is the famous Black-Scholes equation which implements a continuous model involving a partial differential equation to solve for the option price and its temporal evolution from the current time to the time of expiration. An alternative technique is based on the binomial model, proposed in an algorithmic way by Cox, Ross & Rubinstein.

1.2 The binomial model

The binomial model is a discrete-in-time, tree-based model that computes the value of an option contract in two steps. The first step propagates the price of the underlying asset forward in time, applying the probability for a rise or drop of the asset value for each time interval. This step gives the likely evolution of the price under a simple statistical model. Once at expiration date, the value of the option contract can be easily determined. From this distribution at the expiration date, the value of the contract is then traced back (using again an arbitrage argument) to determine the current value of the option contract.

1.3 The forward problem: establishing the binomial tree

The first step establishes a binomial tree for the underlying asset according to the following rule. The price rises by a relative amount u with a probability of p and falls by a relative amount d with probability $1-p$. The relative amount u is given by the volatility σ (or the standard deviation) of the underlying asset; using a random walk argument we have

$$u = \exp(\sigma\sqrt{\Delta t}) \tag{1.1}$$

over a time interval Δt. The drop d is simply $d = 1/u$. The volatility σ can be estimated from historical records, but the assumption of a constant volatility over the duration of the option contract puts a degree of non-realism on the outcome. We still have to determine the probability p of a rise or fall of the underlying asset. To this end, we invoke an arbitrage argument and suppose that the expected relative return (i.e., the return multiplied by its associated probability) is equivalent to the return of a risk-free asset with rate r over the same time interval. We have

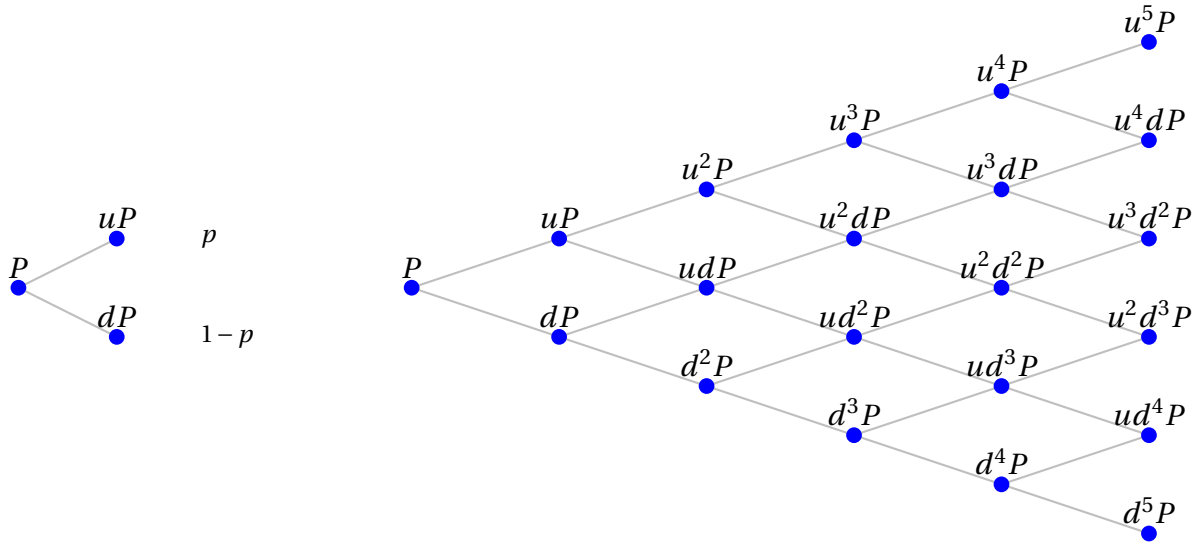

Figure 1.1: Binomial tree structure for computing the development of price P under a binomial statistical model: (left) over one day, (right) over five days, resulting in a tree structure.

$$pu + (1-p)d = \exp(r\Delta t) \tag{1.2a}$$

$$p = \frac{\exp(r\Delta t) - d}{u - d} \tag{1.2b}$$

We can continue this rise-drop procedure for each of the successive days to arrive at a binomial tree structure displayed in figure 1.1.
In the first step of the Cox-Ross-Rubinstein algorithm we build a binomal tree starting from the current date up to the expiration day of the option contract. The final level of the tree then represents the price distribution starting from the current price under a binomial probability model based on the asset's volatility. Figure 1.1(b) gives an example for a 5-level tree, spanning a time interval of five days until expiration.
Example: Given the current price of an underlying asset as $P = 100$, together with the (annualized) volatility of $\sigma = 0.3$ and the (annualized) rate of return of a risk-free asset (e.g. a Treasury note) of $r = 0.05$ we compute the forward binomial tree over five days. From the formula above, we have $u = 1.0191$ and $d = 0.9813$. The results for this example are displayed in Figure 1.2.

1.4 The backward problem: pricing the option contract

Once the price tree for the underlying asset is established, we determine, in a second step, the price of the option contract. To this end, we build a second tree (of identical structure) for the value of the option contract. From the final level of the tree in figure 1.1, representing the price distribution of the underlying asset at expiration, we can easily determine the

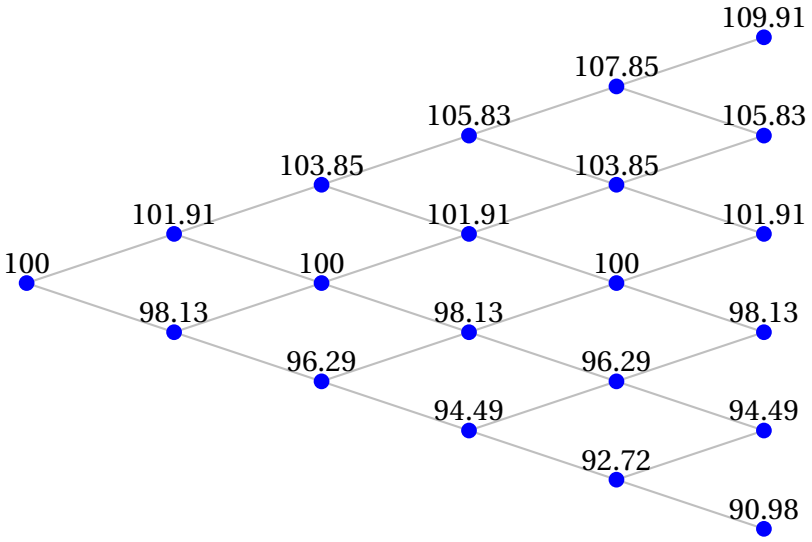

Figure 1.2: Binomial tree structure, for the example given in the text.

value of the option contract. We will consider a call option. In this case, the price C of the call option is given as $C = \max\{0, P - S\}$ where P is the price of the underlying asset and S represents the strike price of the option. This pricing dictates that the option contract at expiration is simply the difference of the current price P minus the strike price (at which we can buy the underlying asset). If the price P is lower than the strike price S, our contract is worthless and there is no point in exercising it, since we can buy the asset cheaper on the open market than by using our option contract.
From this known final price distribution at expiration, we follow the tree back to its root (the current date) by using an arbitrage argument at every node.
We invoke an arbitrage argument as follows. We price the call option such that over one period Δt the gain from the call option is matched by the gain of a mixed portfolio containing n shares of the risky underlying asset and an amount b of the risk-free asset; the variables n and b, i.e. the exact mixture of risky and risk-free assets, are yet unknown. The call price C at the start is thus expressed as $nS + b$. We then have over one period (see Figure 1.4)

$$C \equiv nS + b \quad \longrightarrow \quad unS + \exp(r\Delta t)b \equiv C_u \quad \text{with probability} \quad p \tag{1.3a}$$
$$C \equiv nS + b \quad \longrightarrow \quad dnS + \exp(r\Delta t)b \equiv C_d \quad \text{with probability} \quad 1 - p \tag{1.3b}$$

which, given the call option price from expiration (red box), allows us to determine the call-option prices C at previous time levels. The above equations, in matrix form, read

$$\begin{pmatrix} uS & \exp(r\Delta t) \\ dS & \exp(r\Delta t) \end{pmatrix} \begin{pmatrix} n \\ b \end{pmatrix} = \begin{pmatrix} C_u \\ C_d \end{pmatrix} \tag{1.4}$$

which yields the solution

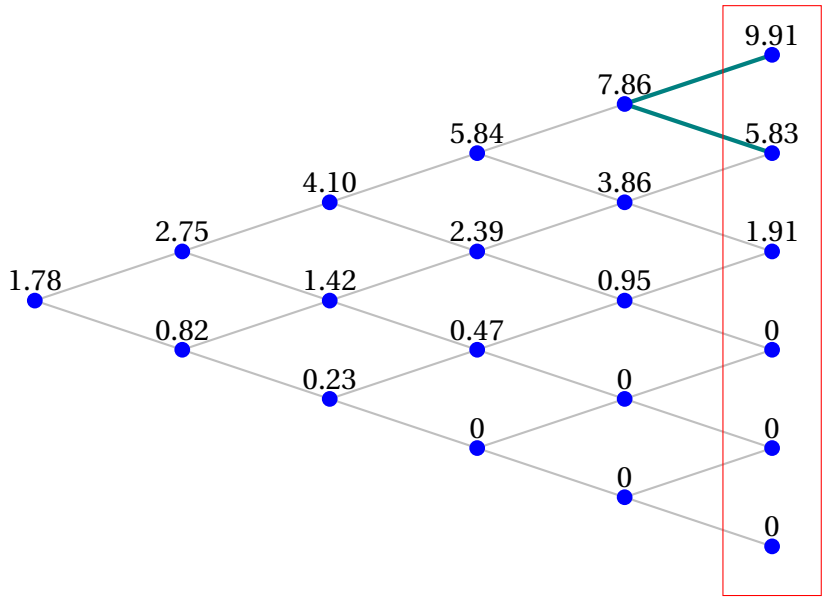

Figure 1.3: Binomial tree structure for the option price, starting at the right edge (red box) and back-tracking to its root by an arbitrage argument (see text).

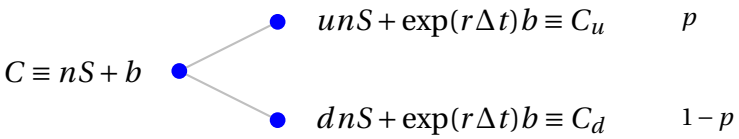

Figure 1.4: Recovery of the call-option value over one time period.

$$n = \frac{C_u - C_d}{(u-d)S}, \tag{1.5a}$$

$$b = \frac{uC_d - dC_u}{(u-d)\exp(r\Delta t)}, \tag{1.5b}$$

and from there the call value C at the earlier time is given as

$$C = \alpha C_u + \beta C_d \tag{1.6}$$

with

$$\alpha = \frac{1}{u-d}\left(1 - \frac{d}{\exp(r\Delta t)}\right) \qquad \beta = \frac{1}{u-d}\left(\frac{u}{\exp(r\Delta t)} - 1\right). \tag{1.7}$$

We then have the full Cox-Ross-Rubinstein algorithm. We first establish a price tree for the underlying asset, using the volatility of the market to determine the amount of relative gain or loss. Once we have the price distribution at expiration, we determine its matching option contract value (since only at expiration do we have information about option pricing). From this final option-price distribution, we recover the option price for the rest of the tree

in a progressive way (from right to left) using an arbitrage argument, until we arrive at the root of the tree (single left node), where we recover today's fair price of the option contract.

1.5 Adjustments for early exercise

The above algorithm assumes that the option contract can only be exercised at the date when it expires. This assumption has established our option value distribution at the right edge of the tree structure (see Figure 1.2 (red box)), where the value of the option at expiration is given by the value of the underlying asset at expiration minus the strike price of the option contract. From this distribution, we back-tracked the value of the option contract to the current date (the root of the tree). An option contract that can only be exercised at the end of its lifetime (at expiration) is known as a **European** option.
In contrast, option contracts that can be exercised *any time* before expiration are referred to as **American** options. Pricing American options follows the same principle with one important difference: when computing the option values during the backward-sweep through the binomial tree we have to compare the computed option value (i.e., the value computed above) to the amount that the current asset value exceeds (in the case of a call option) the strike price. If the former value is larger than the latter, it makes more sense to keep the option contract, and we accept the computed option value. In the reverse case (the difference between the asset and the strike price is larger than the computed value of the option contract), it would make sense to exercise the option right away and realize a risk-free gain. In this case, the fair value of the option contract is the excess of the asset above the strike price; we take this value in the tree, instead of the computed option value.

2 Lattice-based option pricing (Cox-Ross-Rubinstein algorithm)

The `python`-code below implements the Cox-Ross-Rubinstein algorithm, outline above, for European and American call and put options. The tree structure is represented as an array.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  def CoxRossRubinstein(volat,riskfree,days,Current, \
5                        Strike,optType):
6      # Cox-Ross-Rubinstein (CRR) algorithm for option
7      # pricing using a binomial tree structure and
8      # arbitrage arguments
9
10     # time step (approx. 250 trading days in a year)
11     dt = 1./250.
```

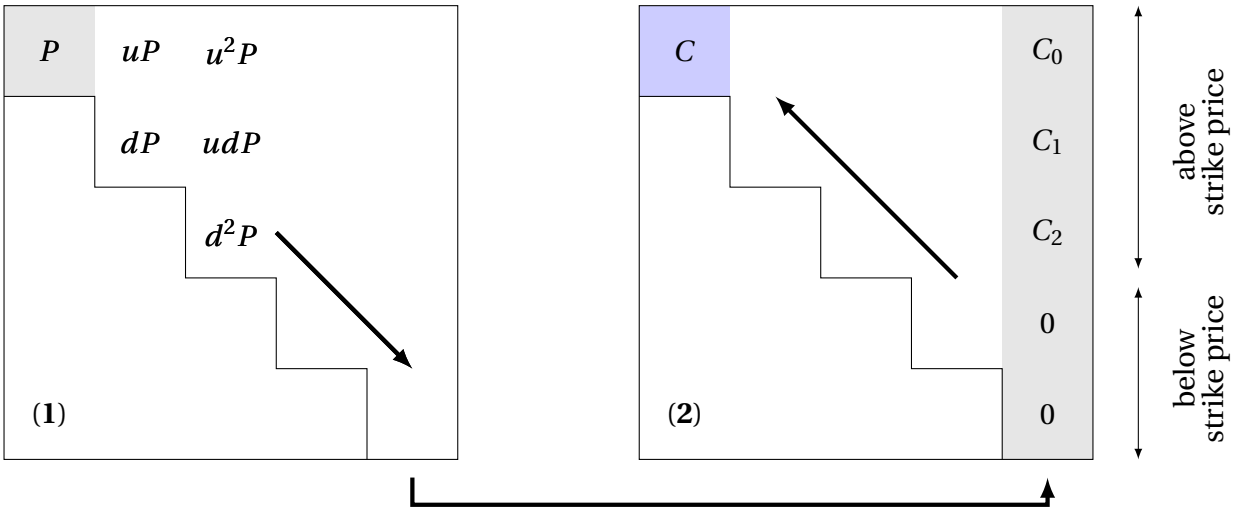

Figure 1.5: Cox-Ross-Rubinstein algorithm (demonstrated for pricing a call option). In the first step, the upper triangular part of the matrix, representing the binomial tree, is filled from the upper-left element (the current price) to the left-most column. From this last column the option price is determined and the results are transferred to the last column of a second matrix. The formula (1.6) is used to fill the columns from right to left, until we arrive at C (light blue box), the value of the option contract today.

```
12      # up/down factor (based on volatility)
13      u = np.exp(volat*np.sqrt(dt))
14      d = 1/u
15      # risk-free factor
16      Rf = np.exp(riskfree*dt)
17      alpha = (1. - d/Rf)/(u-d)
18      beta  = (u/Rf - 1.)/(u-d)
19
20      # mapping matrix for underlying asset
21      PP      = np.zeros((days+1,days+1))
22      PP[0,0] = u
23      PP     += np.diag(d*np.ones(days),-1)
24      # price vector at root of binomial tree
25      P       = np.zeros((days+1,1))
26      P[0]    = Current
27
28      # build underlying-asset binomial tree
29      SS = P.copy()
30      for i in range(0,days):
31          P  = np.matmul(PP,P)
32          SS = np.hstack((SS,P))
```

```
33
34      # evaluation for call options on expiration
35      tmp = SS[:,-1]-Strike
36      CCC = np.clip(tmp,0,max(tmp))
37      # evaluation for put options on expiration
38      tmp = Strike-SS[:,-1]
39      CCP = np.clip(tmp,0,max(tmp))
40
41      # reverse binomial tree for call option price
42      Cnew       = np.zeros((days+1,days+1))
43      Cnew[:,-1] = CCC
44      for it in range(days,-1,-1):
45          for i in range(0,it):
46              Cu = Cnew[i,it]
47              Cd = Cnew[i+1,it]
48              tmp = alpha*Cu + beta*Cd
49              if (optType == 'American'):
50                  tmp2 = SS[i,it]-Strike
51                  tmp  = max(tmp,tmp2)
52              Cnew[i,it-1] = tmp
53      CPrice = Cnew[0,0]
54
55      # reverse binomial tree for put option price
56      Cnew       = np.zeros((days+1,days+1))
57      Cnew[:,-1] = CCP
58      for it in range(days,-1,-1):
59        for i in range(0,it):
60            Cu = Cnew[i,it]
61            Cd = Cnew[i+1,it]
62            tmp = alpha*Cu + beta*Cd
63            if (optType == 'American'):
64                tmp2 = Strike-SS[i,it]
65                tmp  = max(tmp,tmp2)
66            Cnew[i,it-1] = tmp
67      PPrice = Cnew[0,0]
68
69      return PPrice,CPrice
```

```
1   if __name__ == '__main__':
2
3       import pandas as pd
4
```

```
    # read S&P 500 prices from file
    df    = pd.read_csv('SP500Prices.csv')
    AdjC  = df['AdjClose']
    Price = np.array(AdjC[::-1])

    # calculate logarithmic returns
    returns = []
    for i in range(0,len(Price)):
        r = np.log(Price[i]/Price[i-1])
        returns.append(r)

    # compute daily volatility
    volat_d = np.std(returns)
    # adjust to annualized volatility
    volat   = volat_d*np.sqrt(250)

    # input to CRR-function
    Current  = Price[-1]
    days     = 100
    riskfree = 0.05
    Strike   = 2170

    optType  = 'American'   # or 'European'

    # compute put and call option values
    P,C = CoxRossRubinstein(volat,riskfree,days,Current,\
                            Strike,optType)

    # output
    print('Historical volatility = %0.5f' % volat)
    print('Current price         = %.2f' % Current)
    print('Strike price          = %4i' % Strike)
    print('Option type           = %s' % optType)
    print('days till expiration  = %3i\n' % days)
    print('SPX call option       = %.2f' % C)
    print('SPX  put option       = %.2f' % P)
```

3 Example 1

We compute the price distribution of a call option (versus the value of the underlying price) as it develops from today until expiration in 30 days.

```
if __name__ == '__main__':
```

```
2
3       # parameters
4       riskfree = 0.01
5       volat    = 0.3
6       optType  = 'European'
7       Current  = np.arange(45,65,1) # scan over prices
8
9       # loop over days until expiration
10      for i in range(30):
11          days   = 5*i
12          Strike = 55
13          CPrice = np.zeros(len(Current))
14          j = 0
15          for CC in Current:
16              P,C = CoxRossRubinstein(volat,riskfree,days,CC,\
17                                      Strike,optType)
18              CPrice[j] = C
19              j += 1
20
21          plt.plot(Current,CPrice)
22      plt.savefig('Call.png')
23      plt.draw()
24      plt.show()
```

4 EXAMPLE 2

We compute the price distribution (versus the value of the underlying price) of a portfolio consisting of one call and one put option for the same strike price, as it develops from today until expiration in 30 days. This option configuration guards against a quick rise in the market (when the call-option produces profit) and against a sharp drop in the market (when the put-option produces profit). Only when the market fails to move do we realize a loss.

```
1   if __name__ == '__main__':
2
3       # parameters
4       riskfree = 0.01
5       volat    = 0.3
6       optType  = 'European'
7       Current  = np.arange(45,65,1) # scan over prices
8
9       # loop over days until expiration
```

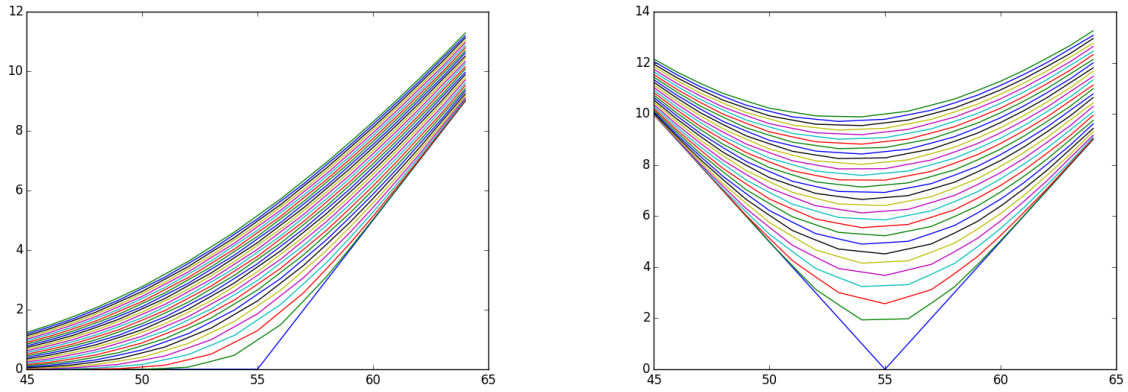

Figure 4.1: (a) Price of call option versus price of underlying asset for different expiration times. (b) Price of call-put combination (same strike price) versus price of underlying asset for different expiration times.

```
10      for i in range(30):
11          days   = 5*i
12          Strike = 55
13          CPrice = np.zeros(len(Current))
14          j = 0
15          for CC in Current:
16              P,C = CoxRossRubinstein(volat,riskfree,days,CC,\
17                                      Strike,optType)
18              CPrice[j] = C + P
19              j += 1
20
21          plt.plot(Current,CPrice)
22      plt.savefig('Straddle.png')
23      plt.draw()
24      plt.show()
```