More on Tree Models

QF607 Numerical Methods

Zhenke Guan
zhenkeguan@smu.edu.sg

Outline

- Pricing following options:
  - Option on Stock Paying a Continuous Dividend Yield
  - Option on Currencies
  - Barrier Knock-In Options
  - Asian Options
  - Spread Option
- Extensions
  - Binomial Tree Pricer for Multi-State Node Values
  - Adaptive Binomial Tree
  - Multi-dimensional Binomial Method
  - Trinomial Tree

Option on Stock Paying a Continuous Dividend Yield

- For options paying continuous dividend yield at rate of q, 1 unit of the stock at time $T = 0$ becomes e^{qT} units at time T.

- Our portfolio replicating an option payoff that holds δ shares of the stock and $V(S_0, 0) - \delta S_0$ units of bond worths at time T

$$\underbrace{\delta S_T e^{qT}}_{\text{value of stock}} + \underbrace{e^{rT}(V_0 - \delta S_0)}_{\text{value of bond}}$$

- To replicate the option payoff V at T, the equations to solve becomes

$$\begin{cases} \delta S_u e^{qT} + e^{rT}(V_0 - \delta S_0) = V_u \\ \delta S_d e^{qT} + e^{rT}(V_0 - \delta S_0) = V_d \end{cases} \tag{1}$$

The solution is

$$\begin{cases} \delta = e^{-qT} \frac{V_u - V_d}{S_u - S_d} \\ V_0 = e^{-rT} \left(\underbrace{\frac{S_0 e^{(r-q)T} - S_d}{S_u - S_d}}_{\text{risk neutral probabiliy} \quad p} V_u + \underbrace{\frac{S_u - S_0 e^{(r-q)T}}{S_u - S_d}}_{1-p} V_d \right) \end{cases} \tag{2}$$

- Black-Scholes model for stock with continuous dividend rate:

$$\frac{dS_t}{S_t} = (r-q)dt + \sigma dW_t, \quad S_T = S_0 e^{(r-q-\frac{1}{2}\sigma^2)T + \sigma W_T}$$

- Second moment to match

$$\mathbb{E}[S_T^2] = e^{2(r-q)T+\sigma^2 T} = pu^2 + (1-p)d^2, \quad p = \frac{e^{r-q}T - d}{u-d} \tag{3}$$

- CRR tree calibration (imposing $u = \frac{1}{d}$):

$$u = \frac{b + \sqrt{b^2-4}}{2} \tag{4}$$

$$d = \frac{b - \sqrt{b^2-4}}{2} = \frac{1}{u} \tag{5}$$

where $b = e^{(r-q)T+\sigma^2 T} + e^{-(r-q)T}$

Option on Currencies

- A foreign currency can be regarded as an asset providing a yield at the foreign risk-free rate of interest, r_f.

- Therefore p is set as

$$p = \frac{e^{(r_d - r_f)t} - d}{u - d} \tag{6}$$

  where we normally use r_d to represent domestic risk-free rate.

- Pricing derivatives on stock with dividend, or foreign exchange rate — just replace the growth rate of the asset by $r - q$

- Note that future cash flow are still discounted by r

Trees with Path Dependent Options

- We have priced path-dependent options, but only a small subset
- When we value the i-th time step for American Option and Barrier Knock-Out options, we are making assumptions about what happened until time step t_i
  - the American option is not exercised at time steps prior to t_i
  - the Barrier KO option has survived until time step t_i
- In other words, our tree is pricing
  - American option that is **not yet** exercised at each node
  - Barrier option that is **not yet** knocked out at each node
- How about other path-dependent options — Knock-In option, Asian Option?

Pricing Barrier Knock-In Options

- Knock-In options can be priced with KIKO parity:

$$\text{KO} + \text{KI} = \text{Underlying Payoff Value}$$

  — if the barrier is triggered, you obtain the underlying payoff from KO, otherwise you obtain the underlying payoff from KI.

- This is what we do in practice, to avoid arbitrage coming from numerical errors
- But what if there is no KIKO parity? How do we price KI's then?
- We need to have two states at each tree node
  - one node representing the barrier is not yet triggered
  - one node representing the barrier has been triggered

Pricing Barrier Knock-In Options

At time step i, for the j-th node $V_{i,j}$, we calculate two values

- $V_{i,j}[0]$: the value of the trade if up to $i-1$ step the barrier is not triggered

$$V_{i,j}[0] = \begin{cases} pV_{i+1,j}[0] + (1-p)V_{i+1,j+1}[0] & \text{if } S_{i,j} \text{ does not hit the barrier} \\ pV_{i+1,j}[1] + (1-p)V_{i+1,j+1}[1] & \text{if } S_{i,j} \text{ hits the barrier} \end{cases}$$

- $V_{i,j}[1]$: the value of the trade if up to $i-1$ step the barrier is triggered

$$V_{i,j}[1] = pV_{i+1,j}[1] + (1-p)V_{i+1,j+1}[1]$$

- The value of the Knock-In option at time 0 is then $V_{0,0}[0]$ if current stock price S_0 does not trigger the barrier, $V_{0,0}[1]$ otherwise.

We noticed that at each time step we are doing

$$\mathbf{V}_{i,j} = \begin{bmatrix} V_{i,j}[0] \\ V_{i,j}[1] \end{bmatrix} = f\left(\begin{bmatrix} \mathbb{E}_{\mathbb{Q}}[V_{i_+,j}[0]] \\ \mathbb{E}_{\mathbb{Q}}[V_{i_+,j}[1]] \end{bmatrix}\right) = f(\mathbb{E}_{\mathbb{Q}}[\mathbf{V}_{\mathbf{i_+,j}}]) \tag{7}$$

That is

$$\mathbf{V}_{i,j} = f(\mathbb{E}_{\mathbb{Q}}[\mathbf{V}_{i_+,j}])$$

Same as our previous simple cases, except that now **V** is a vector, representing the continuation value of the trade under certain assumption.

The function f is the `valueAtNode` function in our implementation.

One minor step of Generalization to our Binomial Pricer

Before extending our tree to deal with multiple states, we make a minor generalization to our `binomialPricer`:

```python
def binomialPricer(S, r, vol, trade, n, calib):
    T = trade.expiry
    t = T / n
    (u, d, p) = calib(r, vol, t)
    # set up the last time slice, there are n+1 nodes at the last time slice
    # NOTE: instead of asking for payoff function, we ask for valueAtNode, and
     feed a None to continuation value to indicate that we are at terminal
    vs = [trade.valueAtNode(T, S*u**(n-i)*d**i, None) for i in range(n+1)]
    # iterate backward
    for i in range(n - 1, -1, -1):
        # calculate the value of each node at time slide i (i+1 nodes)
        for j in range(i + 1):
            nodeS = S * u ** (i - j) * d ** j
            continuation = math.exp(-r * t) * (vs[j] * p + vs[j + 1] * (1 - p))
            vs[j] = trade.valueAtNode(t * i, nodeS, continuation)
    return vs[0]
```

So our requirement to the `trade` now are: `expiry` and `valueAtNode`.

On the tradeable side we extend valueAtNode to return payoff if continuation is None

```python
class EuropeanOption():
    def __init__(self, expiry, strike, payoffType):
        self.expiry = expiry
        self.strike = strike
        self.payoffType = payoffType
    def payoff(self, S):
        if self.payoffType == PayoffType.Call:
            return max(S - self.strike, 0)
        elif self.payoffType == PayoffType.Put:
            return max(self.strike - S, 0)
        else:
            raise Exception("payoffType not supported: ", self.payoffType)

    def valueAtNode(self, t, S, continuation):
        # return payoff if we are at terminal slice
        if continuation == None:
            return self.payoff(S)
        else:
            return continuation
```

Binomial Tree Pricer for Multi-State Node Values

- Now we can easily extend our `binomialPricer` to deal with multi-state node values
- Just ask `valueAtNode` to take a **vector** of continuation values and returns a **vector** of node values

```
def binomialPricerX(S, r, q, vol, trade, n, calib):
  T = trade.expiry
  t = T / n
  (u, d, p) = calib(r, q, vol, t)
  # set up the last time slice, n+1 nodes
  vs = [trade.valueAtNode(T, S*u**(n-i)*d**i, None) for i in range(n+1)]
  # we expect valueAtNode to return us a array of vector
  nStates = len(vs[0])  # getting the number of states for this trade
  for i in range(n - 1, -1, -1): # iterate backward
    # calculate the value of each node at time slide i
    for j in range(i + 1):
      nodeS, df = S*u**(i-j)*d**j, math.exp(-r*t)
      cont = [df*(vs[j][k]*p + vs[j+1][k]*(1-p)) for k in range(nStates)]
      vs[j] = trade.valueAtNode(t * i, nodeS, cont)
  return vs[0][0] # note that we assume the first state is the state as of
      now
```

- Definition and calculation of node values are delegated to tradeables.

Pricing Barrier Knock-In Option

```python
class KnockInOption():
    def __init__(self, downBarrier, upBarrier, barrierStart, barrierEnd,...):
        ...
    def valueAtNode(self, t, S, continuation):
        if continuation == None:
            notKnockedInTerminalValue = 0
            if self.triggerBarrier(t, S):  # if the trade is not knocked in,
                # it is still possible to knock in at the last time step
                notKnockedInTerminalValue = self.underlyingOption.payoff(S)
            # if the trade is knocked in already
            knockedInTerminalValue = self.underlyingOption.payoff(S)
            return [notKnockedInTerminalValue, knockedInTerminalValue]
        else:
            nodeValues = continuation
            # calculate state 0: if no hit at previous steps
            if self.triggerBarrier(t, S):
                nodeValues[0] = continuation[1]
            # otherwise just carrier the two continuation values
        return nodeValues
```

Note how we segregate the general tree framework and tradeable specific pricing logic, like what we did for the simple one-state case.

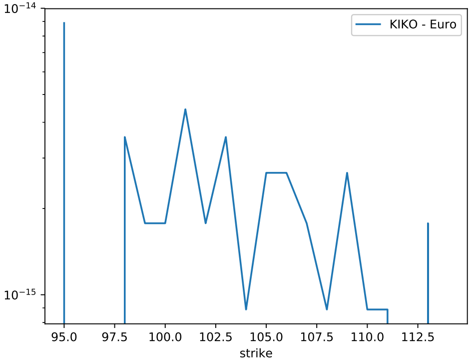
Testing KIKO Parity

```python
opt = EuropeanOption(1, 105, PayoffType.Call)
ki = KnockInOption(90, 120, 0, 1, opt)
ko = KnockOutOption(90, 120, 0, 1, opt)
S, r, vol = 100, 0.01, 0.2
kiPrice = binomialPricerX(S, r, vol, ki, 300, crrCalib)
koPrice = binomialPricer(S, r, vol, ko, 300, crrCalib)
euroPrice = binomialPricer(S, r, vol, opt, 300, crrCalib)
print("kiPrice = ", kiPrice)
print("koPrice = ", koPrice)
print("euroPrice = ", euroPrice)
print("KIKO = ", kiPrice + koPrice)
```

```
kiPrice =  6.001588670701864
koPrice =  0.294468481407765 5
euroPrice =  6.29605715210963 2
KIKO =  6.29605715210962 9
```

Testing KIKO Parity

```python
kis = [binomialPricerX(S, r, vol, KnockInOption(90, 120, 0, 1, EuropeanOption(1, k, PayoffType.Call)), 300, crrCalib) for k in range(95, 115)]
kos = [binomialPricer(S, r, vol, KnockOutOption(90, 120, 0, 1, EuropeanOption(1, k, PayoffType.Call)), 300, crrCalib) for k in range(95, 115)]
euros = [binomialPricer(S, r, vol, EuropeanOption(1, k, PayoffType.Call), 300, crrCalib) for k in range(95, 115)]
kikos = [abs(kis[i] + kos[i] - euros[i]) for i in range(len(kis))]
plt.plot(range(95, 115), kikos, label="KI+KO-Euro")
plt.legend(); plt.xlabel('strike'); plt.yscale('log')  # plot on log scale
plt.savefig('../figs/kiko.eps', format='eps')
plt.show()
```



Pricing Asian Options with Tree

- Knock-In Option gives us an idea about how tree prices products that require knowledge of the past — making **fassumptions** about the past and calculate continuation values based on the assumptions
- Pricing other path-dependent options with tree follow the same logic
- Asian option: it looks at the average $\bar{S}$ of the stock prices observed on a given list of fixing dates $T_i, \ i \in [1, n]$. An Asian call option with strike K's payoff is:

$$\max\left(\bar{S} - K, 0\right) \tag{8}$$

  - Arithmetic average: $\bar{S} = \frac{1}{n}\sum_{i=1}^{n} S_{T_i}$, more frequently seen, we illustrate using this variation
  - Geometric average: $\bar{S} = (\prod_{i=1}^{n} S_{T_i})^{\frac{1}{n}}$, has analytic solution under Black-Scholes model. Why ?

- To value an Asian option on a tree node, what is the information about the past we need to know? The **accumulated average** up to the time of tree node t

Pricing Asian Option with Tree

- Look at it from the last time step (last fixing date T_n): if we know $A = \frac{1}{n-1}\sum_{i=1}^{n-1} S_{T_i}$, the payoff becomes

$$
\begin{aligned}
&\max\left(\bar{S} - K, 0\right) \\
&= \max\left(\frac{1}{n}((n-1)A + S_{T_n}) - K, 0\right) \\
&= \frac{1}{n}\max(S_{T_n} - \underbrace{(nK - (n-1)A)}_{\bar{K}}, 0)
\end{aligned}
$$

  It is just a function of S_{T_n} if we know A.

- So, we sample a list of $[A_1, A_2, \ldots, A_m]$, as the possible value of our auxiliary variable A, then we can value the product at a tree node **assuming** the accumulated average in the past is A_i.
- Our `valueAtNode` function takes a vector of length m `continuationValues` and returns a vector of length m `nodeValues`.

Pricing Asian Option with Tree

- For the time steps between $n-1$ and n-th fixing date, node values are continuation values since A's do not change.
- At time step k that falls on a fixing date T_i, the logic is in `valueAtNode`:
  - We are aiming at calculating the value at node with stock price S, **assuming** the accumulated average is A, so the accumulated average after knowing S is

$$\hat{A} = (A \times i + S)/(i+1) \tag{9}$$

  - Therefore, we need to calculate the continuation value for accumulated average $\hat{A}$.
  - We have the continuation value for a sampled list $[A_1, A_2, \ldots, A_m]$, $\hat{A}$ do not fall exactly on one of them, but we can interpolate
    - Find the interval j that $\hat{A} \in [A_j, A_{j+1}]$, the node value at S, assuming the accumulated average is A is then

$$\frac{\hat{A} - A_j}{A_{j+1} - A_j} V_{j+1} + \frac{A_{j+1} - \hat{A}}{A_{j+1} - A_j} V_j \tag{10}$$

Asian Option — Implementation

```python
class AsianOption():
    def __init__(self, fixings, payoffFun, As, nT):
        self.fixings = fixings
        self.payoffFun = payoffFun
        self.expiry = fixings[-1]
        self.nFix = len(fixings)
        self.As, self.nT, self.dt = As, nT, self.expiry / nT
    def onFixingDate(self, t):
        # we say t is on a fixing date if there is a fixing date in (t-dt, t]
        return filter(lambda x: x > t - self.dt and x<=t, self.fixings)
    def valueAtNode(self, t, S, continuation):
        if continuation == None:
            return [self.payoffFun((a*float(self.nFix-1) + S)/self.nFix) for a in self.As]
        else:
            nodeValues = continuation
            if self.onFixingDate(t):
                i = len(list(filter(lambda x: x < t, self.fixings))) # number of previous fixings
                if i > 0:
                    Ahats = [(a*(i-1) + S)/i for a in self.As] # eq 9
                    nodeValues = [numpy.interp(a, self.As, continuation) for a in Ahats]  # eq 10
        return nodeValues
```

We managed to price Asian option without changing our tree pricer again.
Note that this implementation has a lot of room for improvements

- `As` and `nT` should not be part of `__init__`
- `onFixingDate` is called at each node, and it loops over all the fixing dates, it should be done only once

A better implementation would require a `setup` function and interaction with the tree's geometry

Additive Binomial Tree

It is also commonly seen that binomial trees use $X = \ln S$ as the state variable:

$$
\begin{tikzcd}
 & \boxed{X_u = X_0 + \underbrace{\ln u}_{=u_x}} \\
\boxed{X_0} \arrow[ur, "p"] \arrow[dr, "1-p"'] & \\
 & \boxed{X_d = X_0 + \underbrace{\ln d}_{=d_x}}
\end{tikzcd}
$$

$$t = 0 \qquad\qquad t = \Delta t$$

$$\begin{tikzcd} & \boxed{X_u = X_0 + u_x} \\ \boxed{X_0} \arrow[ur, "p"] \arrow[dr, "1-p"'] & \\ & \boxed{X_d = X_0 + d_x} \end{tikzcd}$$

The calibration formulas matching first and second moments of X are:

- First moment

$$\mathbb{E}[X] = X_0 + pu_x + (1-p)d_x = \overbrace{X_0 + (r - \frac{1}{2}\sigma^2)\Delta t}^{\text{BS solution: } X = X_0 + (r - \frac{1}{2}\sigma^2)\Delta t + \sigma W_{\Delta t}} \tag{11}$$

$$\Rightarrow \quad pu_x + (1-p)d_x = (r - \frac{1}{2}\sigma^2)\Delta t \tag{12}$$

$$\Rightarrow \quad p = \frac{(r - \frac{1}{2}\sigma^2)\Delta t - d_x}{u_x - d_x} \tag{13}$$

- Second moment

$$\mathbb{E}[X^2] = p(X_0 + u_x)^2 + (1-p)(X_0 + d_x)^2 \tag{14}$$

$$= X_0^2 + 2pX_0 u_x + 2(1-p)X_0 d_x + pu_x^2 + (1-p)d_x^2 \tag{15}$$

$$= X_0^2 + 2X_0 \underbrace{(pu_x + (1-p)d_x)}_{(r-\frac{1}{2}\sigma^2)\Delta t} + pu_x^2 + (1-p)d_x^2 \tag{16}$$

Black-Scholes model's second moment:

$$\mathbb{E}[X^2] = X_0^2 + 2X_0(r - \frac{1}{2}\sigma^2)\Delta t + (r - \frac{1}{2}\sigma^2)^2 \Delta t^2 + \sigma^2 \underbrace{\mathbb{E}[W_{\Delta t}^2]}_{=\Delta t} \tag{17}$$

- Equation for second moment:

$$pu_x^2 + (1-p)d_x^2 = (r - \frac{1}{2}\sigma^2)^2 \Delta t^2 + \sigma^2 \Delta t \tag{18}$$

- Knowning p from (13), we have two unknowns u_x, d_x and one equation (18).
- Impose the constraint that $u_x = -d_x = \Delta x$, (18) becomes

$$
\begin{aligned}
\Delta x^2 &= \sigma^2 \Delta t + ((r - \frac{1}{2}\sigma^2)\Delta t)^2 && (19) \\
\Rightarrow \Delta x &= \sqrt{\sigma^2 \Delta t + ((r - \frac{1}{2}\sigma^2)\Delta t)^2} && (20) \\
&\approx \sigma \sqrt{\Delta t} \quad \text{when} \quad \Delta t \to 0 && (21)
\end{aligned}
$$

And

$$
p = \frac{(r - \frac{1}{2}\sigma^2)\Delta t - d_x}{u_x - d_x} = \frac{1}{2} + \frac{r - \sigma^2/2}{2\sigma}\sqrt{\Delta t} \qquad (22)
$$

- Pricing algorithm is the same as multiplicative trees, parameters can be demonstrated to be converging to each other
- Bottom line — when Δt is small enough, all binomial model variations converges to Black-Scholes model

Multi-dimensional Binomial Method

Spread Option

A spread option gives the buyer the right to exchange a stock S_2 for S_1 at maturity T. Its payoff is $max(S_1(T) - S_2(T), 0)$

- To price spread options whose payoff depends on two assets, we need two-dimensional binomial model
- Assume both $S_1(t)$ and $S_2(t)$ follow Black-Scholes SDE

$$\frac{dS_1}{S_1} = (r - q_1)dt + \sigma_1 dW_1 \tag{23}$$

$$\frac{dS_2}{S_2} = (r - q_2)dt + \sigma_2 dW_2 \tag{24}$$

$$\rho dt = dW_1 dW_2 \tag{25}$$

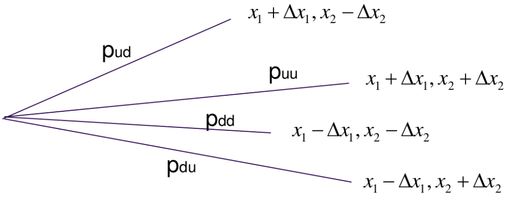
where ρ is the correlation between the Brownian motions of the two assets, q_1 and q_2 are dividend rates of the two assets respectively.

Multi-dimensional Binomial Model

- Using additive tree, we have

$$\begin{cases} dx_1 = (r - q_1 - \frac{1}{2}\sigma_1^2)dt + \sigma_1 dW_1 \\ dx_2 = (r - q_2 - \frac{1}{2}\sigma_2^2)dt + \sigma_2 dW_2 \end{cases} \tag{26}$$

- Choosing equal up and down jump sizes Δx_1 and Δx_2, each tree step looks like:



Multi-dimensional Binomial Model - Calibration

Denote $v_1 = r - q_1 - \frac{1}{2}\sigma_1^2$ and $v2 = r - q_2 - \frac{1}{2}\sigma_2^2$, to calibrate the model, we need to match the means, variances, and correlation:

$$
\begin{cases}
(p_{uu} + p_{ud})\Delta x_1 - (p_{du} + p_{dd})\Delta x_1 = v_1 \Delta t \\
(p_{uu} + p_{du})\Delta x_2 - (p_{ud} + p_{dd})\Delta x_2 = v_2 \Delta t \\
(p_{uu} + p_{ud})\Delta x_1^2 + (p_{du} + p_{dd})\Delta x_1^2 = \sigma_1^2 \Delta t + (v_1 \Delta t)^2 \approx \sigma_1^2 \Delta t \\
(p_{uu} + p_{du})\Delta x_1^2 + (p_{ud} + p_{dd})\Delta x_2^2 = \sigma_2^2 \Delta t + (v_2 \Delta t)^2 \approx \sigma_2^2 \Delta t \\
(p_{uu} - p_{ud} - p_{du} + p_{dd})\Delta x_1 \Delta x_2 = \rho \sigma_1 \sigma_2 \Delta t \\
p_{uu} + p_{ud} + p_{du} + p_{dd} = 1
\end{cases}
\tag{27}
$$

Multi-dimensional Binomial Model - Calibration

Solving the equation system, we have the model parameters:

$$
\begin{cases}
\Delta x_1 = \sigma_1 \sqrt{\Delta t} \\
\Delta x_2 = \sigma_2 \sqrt{\Delta t} \\
p_{uu} = \frac{1}{4} \frac{\Delta x_1 \Delta x_2 + \Delta x_2 v_1 \Delta t + \Delta x_1 v_2 \Delta t + \rho \sigma_1 \sigma_2 \Delta t}{\Delta x_1 \Delta x_2} \\
p_{ud} = \frac{1}{4} \frac{\Delta x_1 \Delta x_2 + \Delta x_2 v_1 \Delta t - \Delta x_1 v_2 \Delta t - \rho \sigma_1 \sigma_2 \Delta t}{\Delta x_1 \Delta x_2} \\
p_{du} = \frac{1}{4} \frac{\Delta x_1 \Delta x_2 - \Delta x_2 v_1 \Delta t + \Delta x_1 v_2 \Delta t - \rho \sigma_1 \sigma_2 \Delta t}{\Delta x_1 \Delta x_2} \\
p_{dd} = \frac{1}{4} \frac{\Delta x_1 \Delta x_2 - \Delta x_2 v_1 \Delta t - \Delta x_1 v_2 \Delta t + \rho \sigma_1 \sigma_2 \Delta t}{\Delta x_1 \Delta x_2}
\end{cases}
\tag{28}
$$

Multi-dimensional Binomial Model

- For every tree nodes (k, i, j), the assets' prices are calculated by

$$S_{1,k,i,j} = S_1 e^{(k-2i)\Delta x_1}, \quad S_{2,k,i,j} = S_2 e^{(k-2j)\Delta x_2} \tag{29}$$

  where k denotes the time step, and i, j represent the number of down movements for asset 1 and asset 2 from $t = 0$.

- At the end nodes of the tree, we calculate the payoff by

$$max(S_1 - S_2, 0) \tag{30}$$

- The value at each node, $V(k, i, j)$ is given by:

$$V_{k,i,j} = e^{-r\Delta t}[p_{uu}V_{k+1,i,j} + p_{ud}V_{k+1,i,j+1} + p_{du}V_{k+1,i+1,j} + p_{dd}V_{k+1,i+1,j+1}] \tag{31}$$

Analytic Solution for Spread Option — Margrabe Formula

- European style spread option can be priced analytically via a change of numeraire
- If we price it using S_2 as numeraire, the option payoff would be $[\frac{S_1(T)}{S_2(T)} - 1]^+$ in unit of S_2 shares.
- Under dollar measure, $\frac{S_1(T)}{S_2(T)}$'s volatility can be obtained using Itô lemma

$$\sigma = \sqrt{\sigma_1^2 + \sigma_2^2 - 2\rho\sigma_1\sigma_2}$$

- The risk-free interest rate becomes S_2's dividend yield q_2
- Now we have the diffusion of $\frac{S_1}{S_2}(t)$ under S_2 measure,

$$d\left(\frac{S_1}{S_2}\right) = \frac{S_1}{S_2}((q_2 - q_1)dt + \sigma dW) \tag{32}$$

Margrabe Formula: Change of Numeraire

- The option can be priced with Black-Scholes formula:

$$V(0) = e^{-q_1 T} \frac{S_1(0)}{S_2(0)} N(d_1) - e^{-q_2 T} N(d_2) \tag{33}$$

in unit of S_2, where

$$d_1 = \frac{\ln \frac{S_1(0)}{S_2(0)} + (q_2 - q_1 + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} \qquad d_2 = d_1 - \sigma\sqrt{T} \tag{34}$$

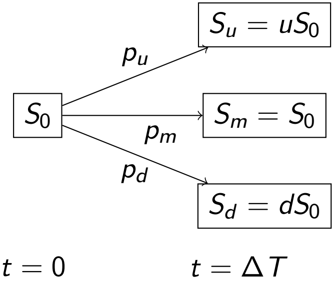
- So converting it to dollar amount is merely a multiplication of $S_2(0)$:

$$V_0 = e^{-q_1 T} S_1(0) N(d_1) - e^{-q_2 T} S_2(0) N(d_2) \tag{35}$$

- We can use this formula to assess the accuracy of our two-dimensional tree.

Trinomial Tree Model

- Developed by Phelim Boyle in 1986
- An extension of binomial tree model
- The asset price can go up by a factor of u or go down by a factor of d, or stay the same with probability of p_u, p_m and p_d respectively



- To calibrate trinomial tree model parameters

$$\begin{cases} p_u u + p_m + p_d d = e^{\mu \Delta t} & \rightarrow \text{match first moment} \\ p_u u^2 + p_m + p_d d^2 = e^{2\mu \Delta t + \sigma^2 t} & \rightarrow \text{match second moment} \\ u = \frac{1}{d} \\ p_u + p_m + p_d = 1 \end{cases} \tag{36}$$

- There are 5 unknows and 4 equations — countless solution
- Use $u = e^{\lambda \sigma \sqrt{\Delta t}}$, where $\lambda \geq 1$ is a tunable parameter, then

$$\begin{cases} p_u = \frac{1}{2\lambda^2} + \frac{\mu - \frac{\sigma^2}{2}}{2\lambda\sigma} \sqrt{\Delta t} \\ p_d = \frac{1}{2\lambda^2} - \frac{\mu - \frac{\sigma^2}{2}}{2\lambda\sigma} \sqrt{\Delta t} \end{cases} \tag{37}$$

Example

- Let $\lambda = \sqrt{3}$, we have

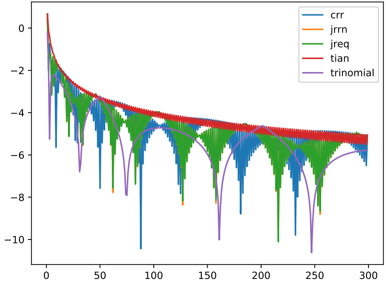
$$\begin{cases} u = e^{\sqrt{3\Delta t}}, \quad d = e^{-\sqrt{3\Delta t}} \\ p_u = \frac{1}{6} + \frac{\mu - \frac{\sigma^2}{2}}{\sqrt{12}\sigma}\sqrt{\Delta t} \\ p_d = \frac{1}{6} - \frac{\mu - \frac{\sigma^2}{2}}{\sqrt{12}\sigma}\sqrt{\Delta t} \\ p_m = \frac{2}{3} \end{cases} \tag{38}$$

Trinomial Tree Implementation

```
def trinomialPricer(S, r, q, vol, trade, n, lmda):
    t = trade.expiry / n
    u = math.exp(lmda * vol * math.sqrt(t))
    mu = r - q
    stdev = vol*math.sqrt(t)
    pu = 0.5/lmda/lmda + (mu-vol*vol/2)/2/lmda/stdev
    pd = 0.5/lmda/lmda - (mu-vol*vol/2)/2/lmda/stdev
    pm = 1 - pu - pd
    # set up the last time slice, there are 2n+1 nodes at the last time slice
    # counting from the top, the i-th node's stock price is S * u^(n - i),
    # i from 0 to n+1
    vs = [trade.payoff(S * u ** (n - i)) for i in range(2*n + 1)]
    # iterate backward
    for i in range(n - 1, -1, -1):
        # calculate the value of each node at time slide i, there are i nodes
        for j in range(2*i + 1):
            nodeS = S * u ** (i - j)
            continuation = math.exp(-r*t)*(vs[j]*pu + vs[j+1]*pm + vs[j+2]*pd)
            vs[j] = trade.valueAtNode(t * i, nodeS, continuation)
    return vs[0]
```

Trinomial Tree vs Binomial Tree

```python
opt = EuropeanOption(1, 105, PayoffType.Call)
S, r, vol, n = 100, 0.01, 0.2, 300
lmbda = math.sqrt(3)
bsprc = bsPrice(S, r, vol, opt.expiry, opt.strike, opt.payoffType)
crrErrs = [math.log(abs(binomialPricer(S, r, vol, opt, i, crrCalib) - bsprc)) for i in range(1, n)]
jrrnErrs = [math.log(abs(binomialPricer(S, r, vol, opt, i, jrrnCalib) - bsprc)) for i in range(1, n)]
jreqErrs = [math.log(abs(binomialPricer(S, r, vol, opt, i, jreqCalib) - bsprc)) for i in range(1, n)]
tianErrs = [math.log(abs(binomialPricer(S, r, vol, opt, i, tianCalib) - bsprc)) for i in range(1, n)]
triErrs = [math.log(abs(trinomialPricer(S, r, 0, vol, opt, i, lmbda) - bsprc)) for i in range(1, n)]
plt.plot(range(1, n), crrErrs, label="crr")
plt.plot(range(1, n), jrrnErrs, label="jrrn")
plt.plot(range(1, n), jreqErrs, label="jreq")
plt.plot(range(1, n), tianErrs, label="tian")
plt.plot(range(1, n), triErrs, label="trinomial")
```



Trinomial Tree vs Binomial Tree

- Trinomial tree is considered to produce more accurate results than binomial tree (marginal in my opinion)
- The extra node and parameter gives some freedom to position tree nodes at critical points of the payoff, for example
  - Discontinuity point at payoff
  - Barrier level, and with the center state, trinomial tree allows having node at barrier value at each time step