In [1]:
```python
# ch16_binomial_tree.ipynb
import numpy as np # library for numerical & math calculations
import matplotlib.pyplot as plt # library for graphing
import scipy.stats as sp # library with prob/stat functions
```

In [2]:
```python
# functions for exact value of European Call for comparison
def d1(S,K,T,t,r,sigma):
    d1 = (np.log(S/K) + (r+0.5*sigma**2)*(T-t))/(sigma*np.sqrt(T-t))
    return d1
def d2(S,K,T,t,r,sigma):
    d2 = (np.log(S/K) + (r-0.5*sigma**2)*(T-t))/(sigma*np.sqrt(T-t))
    return d2
def C_euro(S,K,T,t,r,sigma):
    C_euro = S*sp.norm.cdf(d1(S,K,T,t,r,sigma))-\
        K*np.exp(-r*(T-t))*sp.norm.cdf(d2(S,K,T,t,r,sigma))
    return C_euro
```

In [3]:
```python
# binomial tree method of option valuation

# stock paramters
S0 = 110
K = 120
T = 1
r = 0.05
sigma = 0.3

# binomial parameters for stock model
M = 170
dt = T/M
p = 0.5
nu = r - 0.5*sigma**2
u = np.exp(nu*dt+sigma*np.sqrt(dt))
d = np.exp(nu*dt-sigma*np.sqrt(dt))
print('p,u,d = ',p,u,d)

# stock model
from scipy.special import comb as nchoosek
S = np.zeros(M+1) # array of S values at t=T
prob = np.zeros(M+1)  #probability at t=T
for n in range(M+1): # n = 0,1,...,M
    S[n]=S0*d**(M-n)*u**(n)    # M-n down steps and n up steps
    prob[n] = nchoosek(M,n)*p**n*(1-p)**(M-n)
#print('S_T    = ',S)
#print(prob,sum(prob))

# compare probability distribution of S_T to asset model
#Z = (np.log(S/S0) - nu*T)/(sigma*np.sqrt(T))
#binwidth = np.log(u/d)/(sigma*np.sqrt(T))
#plt.plot(Z,prob/binwidth,'bo')
#xfine = np.linspace(-3,3) # fine array in x for plotting exact curve
#plt.plot(xfine,sp.norm.pdf(xfine),'r-')
#plt.xlabel('z')
#plt.ylabel('probability')
#plt.show()

# binomial tree for option values
V = np.zeros((M+1, M+1))*np.NaN # V(i,j) = option value at t_i, n down steps

# payout values at t=T
for n in range(M+1): # n = 0,1,...,M
    V[M,n] = max(S[n]-K,0)    # payout function at t=T
#print(V[M,:])

# propagate option value backwards using (16.3)
for i in range(M-1, -1, -1): # i = M-1,M-2,...,0
    for n in range(i+1):     # n = 0,1,...,i
        V[i,n]=np.exp(-r*dt)*(p*V[i+1,n+1]+(1-p)*V[i+1,n])
#print(V)
price = V[0,0]

# compare to exact option price
print('binomial option price      = ',price)
print('Black-Scholes option price = ',C_euro(S0,K,T,0,r,sigma))
print('error = ',price-C_euro(S0,K,T,0,r,sigma))
```

```
p,u,d =  0.5 1.0233057942680044 0.9772824808198374
binomial option price      =  11.395493972365943
Black-Scholes option price =  11.390187181129491
error =  0.005306791236451502
```

Note - Error ->0 but not monotonic in M (see text sec 16.4):

1. Error distinctly different for M even vs M odd.
2. error has decaying oscillations as M increases
3. overall, error can be bounded by const/M

In [ ]: