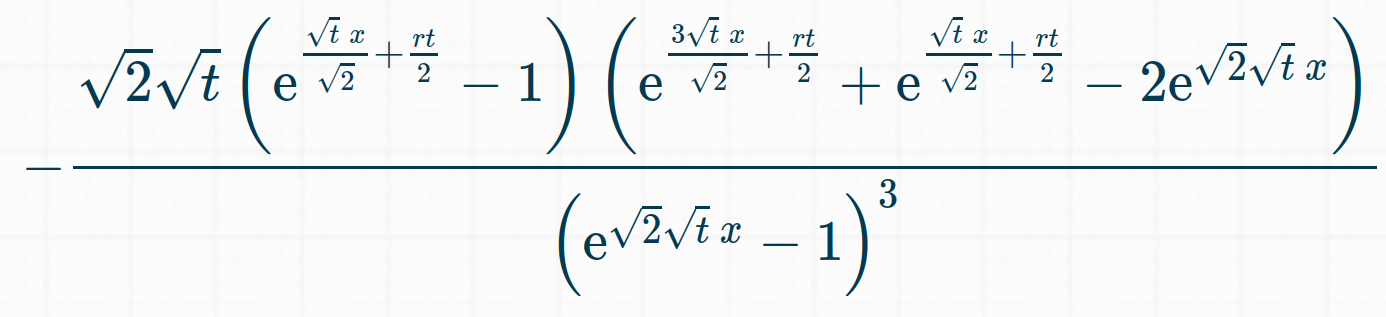
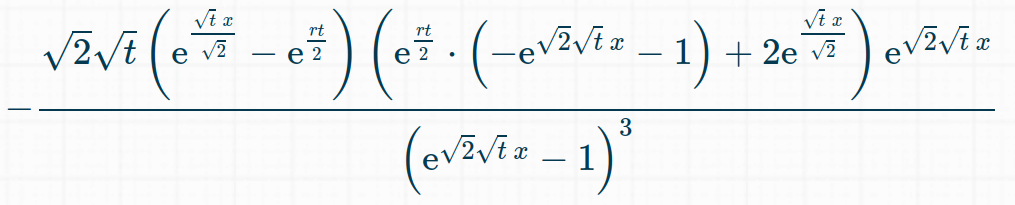
Function **price\_xyz:**

1. Calculated total number of days **without** using the datetime module of python.
   1. **get\_diff() function**: assigned day number to both the dates and returned the difference of two.
2. Delta\_t = total days / num\_steps.
3. Calculated pu, pd, ju, jd using formula given in problem statement
4. Building the **lattice of prices** as a nested list in python.
   1. First level = [ A\_t\_d ]
   2. Each level is dependent on its previous level.
   3. At = list of prices at level t.
   4. Using At-1,0 fill the first three nodes of At.
   5. Iterate over At-1 [1: ], and append only the **down value(At-1,j\*jd)** in At list.   
      [as ju\*At-1,j and At-1,j and are already filled from the previous values of At-1 list.]
5. Building **XYZ payoff list (X).**
   1. Last level of X is simply max(A\_t\_e - B, 0)
   2. Each level depends on the next level.
   3. Xt = list of payoffs at level t.
   4. For each subarray of size 3(a0, a1, a2) in Xt+1, a value is appended in Xt as   
      (a0\*pu + a1\*(1-pu-pd) + a2\*pd)\*e^(-r\*delta\_t)
6. In the end, the first level of XYZ will have only one node, i.e Xd.

Function **volrisk\_xyz**

1. Calculation for the function price\_xyz is done and the derivative is also calculated simultaneously.
2. Firstly, I calculated derivatives of pu, pd, ju and jd wrt sigma.





1. Building dA\_t along with A\_t.
   1. First level = [0] [as a\_t\_d is a constant]
   2. Using the multiplication rule differentiation to fill the dA\_t following the calculations of A\_t.  
      Eg A\_t, i = A\_t-1, j \* jd

dA\_t, i = dA\_t-1, j \* jd + djd \* A\_t-1, j

1. Building dX\_t along with X\_t.
   1. Last level is dependent on B.

If A\_num\_steps, i <= 0 then dX\_t, i = 0

Else dX\_t, i = dA\_num\_steps, i

* 1. Using the multiplication rule, each level is filled following its dependency on the next layer.

1. The first level of dX\_t will contain only one node, thus returning dXd\_t / dsigma