Anti-K_T algorithm

May 1, 2017

1 Formulae

$$\Delta_{ij} = \sqrt{(\eta_i - \eta_j)^2 + (\phi_i - \phi_j)^2} \tag{1}$$

$$\eta = \frac{1}{2} ln \left(\frac{|p| + p_l}{|p| - p_l} \right) \tag{2}$$

Where p_l is the momentum in the beam direction.

$$\phi = arctan \left(\frac{p_y}{p_x} \right) \tag{3}$$

$$p_T = \sqrt{p_x^2 + p_y^2} \tag{4}$$

$$d_{ij} = min \left(p_{Ti}^{2p}, p_{Tj}^{2p} \right) \times \frac{\Delta_{ij}}{R} \tag{5}$$

$$d_{iB} = p_{Ti}^{2p} \tag{6}$$

2 How the code works

The beginning of the code contains the old information from parton shower 3d code with few changes;
Now the Energy of the partons follows the exponential distribution $e^{(-\alpha E_{parton})}$

```
def E(alpha= 0.5):
    while True:
        x = np.random.uniform(0,10)
        y = np.random.uniform(-alpha,0)
        f = alpha*np.exp(-alpha*x)
        if y <=f:
            return x
```

and the function *parton3d()* returns a list of final state particles which are the corner stone of the algorithm. Also, since we have two partons, there is a function which creates and concatenates the two lists of final state particles from each parton shower.

```
def partons(E):
    J1 = parton3d(E)
    J2 = parton3d(-E)
    return J1 + J2
```

The first part is followed by the functions d_{ij} and d_{iB}, which follows eqns 5, 6 of the formulae above respectively.

The whole algorithm is included in the function $jetcluster(p, R, J)$. In the heart of the algorithm, there is heap queue which is an implementation of the priority queue algorithm. The interesting property of heap is that it gives the smallest element in a list, maintaining the list invariant, this is done with the method *heap.pop*.

Regarding storing of the data and processing it, the *class* idea is implemented, with the help of a function called *combine()* which performs the date processing. Where the class $Pseudojet$ works as a container of all the final state particles defined as Pseudojets in the class with the attributes momentum, index, jet, and exists .

```
class Pseudojet:
    instances  = []
    def __init__(self, v_momentum):
        self.momentum = v_momentum
        self.index = len(Pseudojet.instances)
        self.is_jet = False
        self.exists = True
        Pseudojet.instances.append(self)
```

The first attribute describes the momentum of the Pseudojet, the second attributes describes the location of the particle, and the third attribute is meant to judge the particle after it's being checked through the heap queue, it changes to $True$ under the condition of the algorithm. The last attribute describes the existence, where after the smallest distance turns to be d_{ij} i.e if we don't have a jet, then those pseudojets will be combined into one pseudojet. For each pseudojet, one of the existed attributes will change to False. The last two attributes are controlled by the function below.

```
def combine(J1, J2,p,R):
    J = J1.momentum + J2.momentum
    J1.exists = False
    J2.exists = False
    J  = Pseudojet(J)
    di_B = diB(J,p)
    heapq.heappush(H,(di_B,J.index,-1))
    for i in range(len(lis)-1):
        if lis[i].exists:
            di_j = dij(J,lis[i],p,R)
            heapq.heappush(H,(di_j,J.index,i))
```