Jets in Switzerland

Nihal Bahaa Anwer Razk (nihal@aims.ac.za)
نهال بهاء انور رزق
African Institute for Mathematical Sciences (AIMS)

Supervised by: Dr Samuel Meehan
University of Washington, United States of America
Co-Supervised by: Dr Sahal Yacoob
University of Cape Town, South Africa

18 May 2017

Submitted in partial fulfillment of a structured masters degree at AIMS South Africa

AIMS | African Institute for Mathematical Sciences SOUTH AFRICA

Abstract

This research is focused on the study of quarks and gluons by the use of hadronic jets at the Large Hadron Collider (LHC) in Geneva, Switzerland. We are using the Monte Carlo technique to generate random data and using it to simulate these hadronic jets by implementing fundamental conservation laws such as energy and momentum conservation. This simulation produces a spray of simulated particles from which a number of observables can be formed to study the strong interaction. These observables can then be studied both via this simulation and in real data from the LHC at CERN to constrain the parameters describing the types and strengths of these interactions. These constraints will better allow us to understand the collisions that occur, thereby being more sensitive to new physics. The Python simulation code is accessible via bitbucket here.

Declaration

I, the undersigned, hereby declare that the work contained in this research project is my original work, and that any work done by others or by myself previously has been acknowledged and referenced accordingly.

[illegible]

Nihal Bahaa Anwer Razk, 18 May 2017

Contents

1. Introduction

What is the world made of? What happened during and immediately after the big bang? What is Dark energy? To answer these questions we need to study particle physics, as well as to study and improve the Standard Model, looking for new physics and new particles like the Higgs boson. In order to do this it would seem as if we need a big microscope to look inside the atoms themselves, and even inside the protons in the atoms. To build such a microscope we need to generate very high energies, and for that we need a very big accelerator such as The Large Hadron Collider (LHC). The LHC accelerates, particles in a 27-Kilometer circular bath 100 metres under the ground, located between France and Switzerland (ATL, 2012).

When protons at the LHC are accelerated, their energy becomes 7000 times higher than their rest mass energy. When two protons collide at such high energy, the interaction occurs between the constituents of the protons. As we know, protons are not really fundamental particles but are made up of quarks and gluons. When beams of particles collide at energies up to seven trillion electron-volts inside the LHC ATLAS detector, a lot of particles are produced, which are then measured by ATLAS (Ellis et al., 2008).

The detector itself is a many-layered instrument designed to detect the tiniest energetic particles. Consisting of subsystems layered around the centre point of the collision to record the trajectory, momentum, and energy of particles, the detector can measure how much energy comes from the electromagnetically interacting particles and hadronically interacting particles. It is clear from the high density of particles appearing in the detector that the product of one particle appears in the detector like a spray of particles. This spray of particles takes a cone shape, known as a jet (Ellis, 2010).

Jets are reconstructed from energy depositions in calorimeter cells (Ellis et al., 2008). The reason we focus on studying jets is actually to study the properties of the produced particles from the collision. In fact, in high energy collisions, each product particle will radiate until it reaches its stable energy. In the end, a parton shower will be formed, and at the end of this stage the partons tend to bind together to form stable hadrons. These hadrons are what we detect in the detector. We can define the jet by the produced parton, one jet equals one parton. To find the jet we need to cluster those soft radiated particles together to reach to the hard produced one, we do this using clustering algorithms. In this research we will utilise sequential recombination algorithms (Cacciari et al., 2008b).

Such collisions, taking place at high energies, are governed by the laws of quantum mechanics, and that causes the random nature of the parton shower construction. This randomness motivates us to use the Monte Carlo technique, which it used to simulate processes that are based on the electromagnetic and strong interactions. The reason is that when we have a complicated system composed of many parts, it is easier to break it down and simulate each little "sub-part" separately, and string them together. In the case that there is a random nature to a given subpart, we need to use a Monte Carlo simulation, also because most of the time when we deal with quantum mechanics and quantum chromodynamics (QCD) an analytic solution does not exist. That is why Monte Carlo simulations are widely used in high energy physics (Ydri, 2016).

In our research we will use the Monte Carlo techniques to simulate and understand the physics behind jets. We do this by generating many random events. Our simulation need to incorporate some conditions to make them similar to the real events. From this generated events we will introduce a number of observables, which will provide an insight into the features of collision processes which occur in high energy-physics.

2. Jet formation

The collision of partons in the LHC is governed by the laws of quantum mechanics, which causes a degree of randomness in the produced particles that might produce Z bosons or photon. in other cases it can produce a Higgs boson. The product parton will spawn extra quarks and gluons, that is because it is a bare parton which has colour charge, and this not allowed in QCD, this process form in the end what we call as the parton shower. After the parton shower stage, the partons will pair up into pairs of [quark-antiquark] or [quark-quark-quark] and produce many stable hadrons, which are collimated in the same direction and detected at the same time, this stage is called the hadronization level, this two process, parton shower and hadronization, are well described by Figure 2.1. We will consider these detected stable hadrons as pseudo-jets. To find the jets we will need to cluster these pseudo-jets using clustering algorithms.

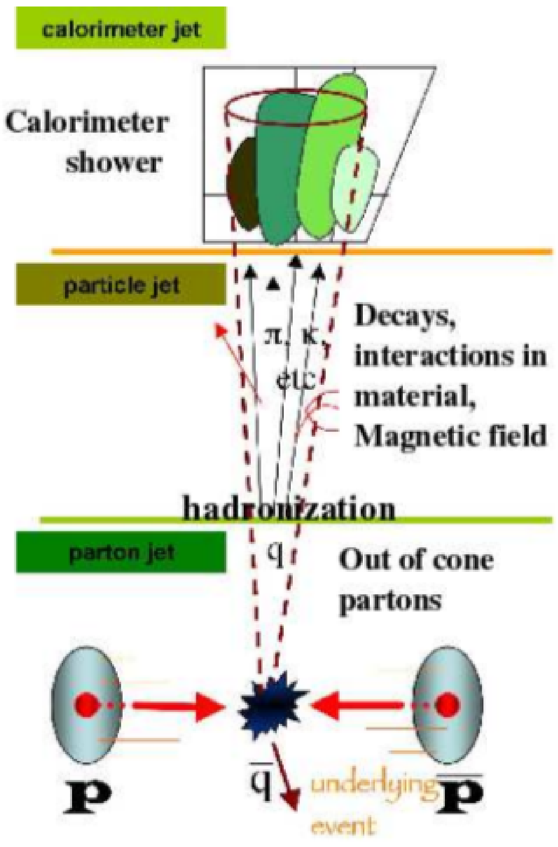


Figure 2.1: This figure shows jet production processes and measurements (Ellis et al., 2008)

2.1 The parton shower

After producing the high energy partons from the collision. They start to divide in to many partons, forming the parton shower. Parton shower processes can be shown by Feynman diagram, given an n leg

(M_n) Feynman diagram will division to (n+1) leg (M_{n+1}) Feynman diagram, as shown in Figure 2.2 and by Equation 2.1.1

$$|M_{n+1}|^2 \sim \frac{1}{\theta^2} C_A F(Z) \, |M_n|^2 \quad \text{(Ellis et al., 2003)} \tag{2.1.1}$$

where $Z = \frac{E_{rad}}{E_f}$ the ratio between the energy of the radiation parton to the energy of the final parton, θ is the angle between the two legs of the produced partons. The parton shower level is shown as the straight arrows, curly and wavy lines in Figure 2.3.

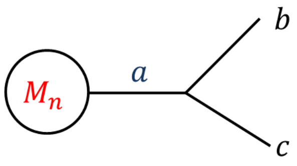


Figure 2.2: Feynman diagram

2.2 The Hadronization Stage

The idea after the hadronization is that when the partons created the parton shower it is not longer until the partons bind together in order to become more stable. This concept is similar to the heuristic model where electrons and protons bind together to form the hydrogen atom if they are placed approximately near to each other (Buckley et al., 2011). Hadronised states are represented in the figure 2.3 by the yellow circles. These final state hadrons, which we called pseudo-jets, are the detected particles.

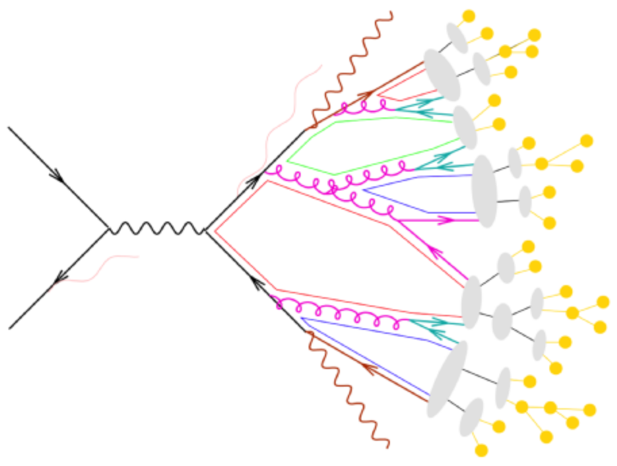


Figure 2.3: Diagram for electron–positron scattering
(Ellis et al., 2008)

3. Jet model

3.1 Toy Monte Carlo with Python

The Monte Carlo method is a method that solves problems by creating random events under some properties and sets up some conditions to make it approach the real event. Usually we use Monte Carlo methods in the problems that are hard to solve analytically, like cases in quantum mechanics and (QCD). In our research, we deal with partons (quarks and gluons), which are described by QCD. The random degree in this process allows us to use Monte Carlo methods to generate random events and study the properties of the hadronic collision.

3.1.1 Acceptance-rejection method. It is also called the Von-Neumann method and is commonly used when the analytical form of the function $F(x)$ is complex or not defined. However it is required to know the probability distribution function (PDF) for the system (Amsler et al.). To generate such values follow the PDF, we need to generate a pair of random numbers (x, y), whereby x is the value that we are interested in, and y is the value that will decide whether we will accept x or not.

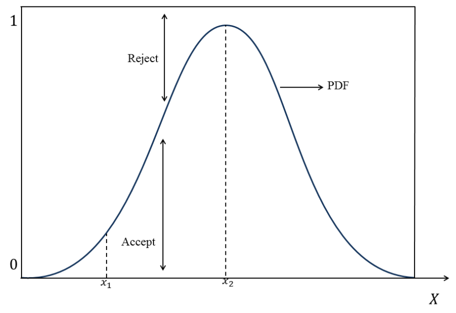


Figure 3.1: Plot of PDF $1/x$, with generated x values from accept and reject method

Whenever we generate an x value, we generate a y value as well, as the PDF is defined such that the total area is equal to unity, we will generate a y value between [0, Max(PDF)], then we compare it with the $PDF(x)$ if ($y > PDF(x)$) we will reject the x value if not we will accept the x value. Looking at the Figure 3.1, if we generate a value of $x = x_2$, then it is more likely to be accepted it because it has high probability, from the $PDF(x)$ it is bigger than the y value, but if we generate x value let's say at $x = x_1$ then it is more likely that the generated y value from the uniform distribution will be bigger than the PDF(x) and we will reject the x value. Under that condition, we make sure that the accepted x value follow the probability distribution function. Actually, the PDF that we will use in our Monte Carlo model, to construct the jet is $1/x$. Let's look at the algorithm to generate random values following this distribution.

```
Result: $x$ Values
$n = 1000$;
$x$=[ ];
$j$=0;
while $i < n$ do
    $x_{test}$=random nuber between [1,3];
    $y_{text}$=random number between [0,1];
    PDF(x) = $1/x_{\text{test}}$;
    if $y_{test} < PDF(x)$ then
        $x$value = $x_{\text{test}}$;
        append xvalue in $x$;
        $A$ +=1;
    end
    $G$ +=1;
end
```

Algorithm 1: Algorithm to generate x values, using accept-reject method, follow PDF $1/x$

The number of accepted x values = A, the number of generated x value = G, and the rejected x values = $G - A$. After drawing the x values we get Figure 3.2, which follows the probability distribution function $1/x$.

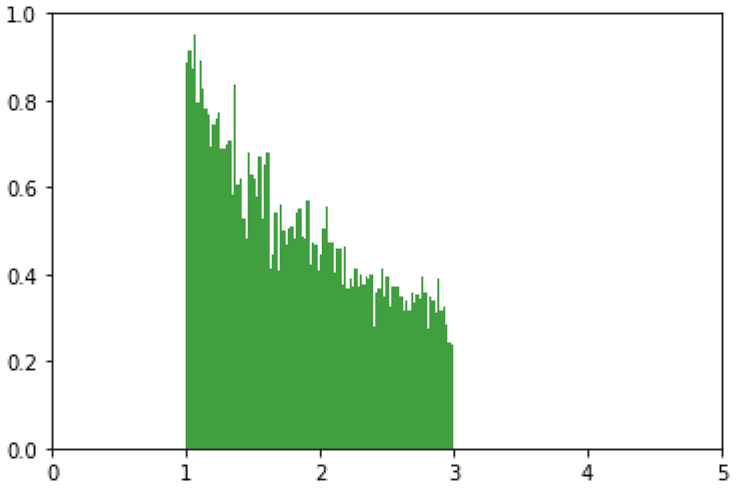


Figure 3.2: Python plot of x values using the accept-reject method

3.1.2 Inverse transform method. The inverse transform method is complementary to the previous method. It also generates random events under a certain probability distribution, but using a different means to do so.

The way this function works is to find the cumulative density function (CDF) by integrating the PDF then generating random values for y. Instead of comparing the y value to the probability distribution function, we will find x value directly by inverting the function $CDF(x) = y$. As stated earlier, for our current work we will use $PDF = 1/x$, then we find the CDF of this function, which is $\log(x)$, as shown below in Figure 3.3.

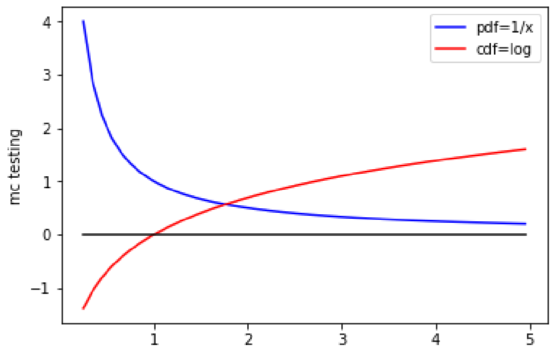


Figure 3.3: Plot of PDF $1/x$ and CDF $\log(x)$ together

Working through this algorithm we can generate n numbers of x values using the Inverse transform method. To generate random events we follow a probability distribution function PDF $= 1/x$.

```
Result: X Values
$n = 1000$;
$x_{\text{value}}$=[ ];
while $i < n$ do
    $y_{\text{text}}$ =random number;
    $x_{\text{test}} = \exp(y_{\text{test}})$;
    append $x_{\text{test}}$ in $x$ value;
    $A+ = 1$;
end
```

Algorithm 2: Algorithm to generate x values where PDF=$1/x$, using the inverse transform method

All the generated x values will be accepted values and it is equal to A Plotting the x values in the range [1-3] we get Figure 3.4, which gives the PDF of $1/x$.

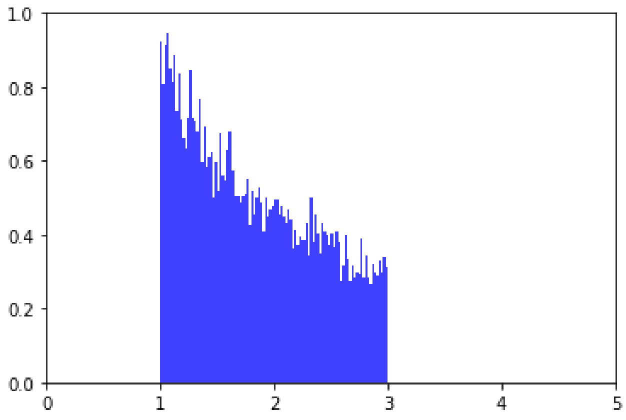


Figure 3.4: Python plot for x values using the inverse transform method

After generating random events using the accept-reject method and inverse transform method, which gives us the same result, we can see that the inverse transform method is more efficient than the accept-reject method. Using the inverse method we will decrease the cost of the algorithm because all the generated values of x will be accepted. But in the accept-reject method, we must go through the condition to accept or reject the values of x.

3.2 Parton shower simulation

in the same way that charged particles emit photons, the coloured partons emit gluons, and these gluons themselves can radiate, which in the end builds the parton shower.(Buckley et al., 2011)

As we mentioned above about the random nature of the jet, which came from the uncertainty of produced particles in the splitting process, and the way that any of these splitting processes can happen, g→gg,g→ qq and q→ qg, as well as there being no prediction about the shape of the jet, all of these reasons enable us to simulate the jet using Monte Carlo methods.

It is also important to define the Momentum-Energy 4-vector $P_\mu = \left(\frac{E}{c}, p_x, p_y, p_z\right)$, and how energy and momentum transform in special relativity. In our model we will set the speed of the light $c = 1$ and assume that all the emitted particles are massless. We will set an initial particle with initial Momentum-Energy 4-Vector $[E, p_x, p_y, p_z]$, which will split to two other particles. and we will try to simulate the one-two splitting by defining a number of parameters such as θ which is the angle of the radiation process as shown in figure 3.5, the Z parameter which is the fraction of the radiated energy, ϕ the azimuthal angle, which is in the range $[0 - 360]^\circ$, and the initial energy E_i.

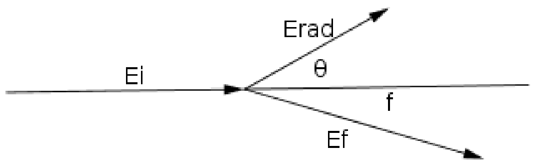


Figure 3.5: single one-to-two splitting

In this model we will use Monte Carlo inverse transfers method to generate θ and Z which follow the probability distribution functions $\frac{1}{\theta}$ and $\frac{1}{Z}$ respectively. We will limit the θ values from 0^o to 90°, and the Z value from 0.25 to 0.75. After that we can calculate the radiated energy as $E_{rad} = ZE_i$, and the final energy as $E_f = E_i - E_{rad}$. The Momentum-Energy 4-vector must be conserved, where the summation of the radiated particle 4-vector and the final particle 4-vector must together equal to the initial 4-vector $[Ei, p_{x_i}, p_{y_i}, p_{z_i}] = [(Ef + Er), (p_{x_r} + p_{x_f}), (p_{y_r} + p_{y_f}), (p_{z_r} + p_{z_f})]$. In our model, we set the initial particle and the radiated particle which usually the gluons, to be massless, but we found that the final particle can have mass. In this case we will deal with the basic Energy equation of Special Relativity $|p| = \sqrt{E^2 - m_0^2}$, where m_0 is the rest mass of the final particle.

For simplicity, in our model we simulate just two probabilities of the splitting process, that the quark will radiate gluons with $E_{radiation} = ZE_{final}$ and random θ, the other case that the gluon itself can split into two quarks which have equal energies, which means that each quark will have half of the gluon energy. This process will last until the typical scale of the transferred momentum becomes small, which on more more splitting for the parton is allowed. We can control that in the code by putting a critical energy. When the particles reach it, they will stop spitting. These final particles are considered the final state hadrons.

Here is a simple algorithm to the parton shower process.

```
Result: final state hadrons
$E_{critical}$=value;
$E_{initial}$=value;
initial four vector=$[E, px, py, pz]$;
while Particle-Energy $> E_{critical}$ do
    generate randome Z value;
    generate random $theta$;
    generate random $Phi$;
    $E_{radiation} = ZE_{inital}$;
    $E_{final} = E_{inital} - E_{radiation}$;
    make the split process with
    Radiated-particle 4-vector=$[E_r, p_{x_r}, p_{y_r}, p_{z_r}]$;
    Final-particle 4-vector=$[E_f, p_{x_f}, p_{y_f}, p_{z_f}]$;
end
```

Algorithm 3: Algorithm to describe the spliting process

In the end of this process we will have Energy-Momentum four vectors for the final states, which must have the summation of their energy equal to the energy of the initial parton. $\sum_{i=1}^{n} E_i = E_{initial}$. Also we must have the conservation in the Energy-Momentum four vector for each final state, which can be shown by verifying $E_i = (p_{x_i}^2 + p_{y_i}^2 + p_{z_i}^2)^{\frac{1}{2}}$.

After running the Python simulation code we get the parton shower simulation as shown in Figure 3.6.

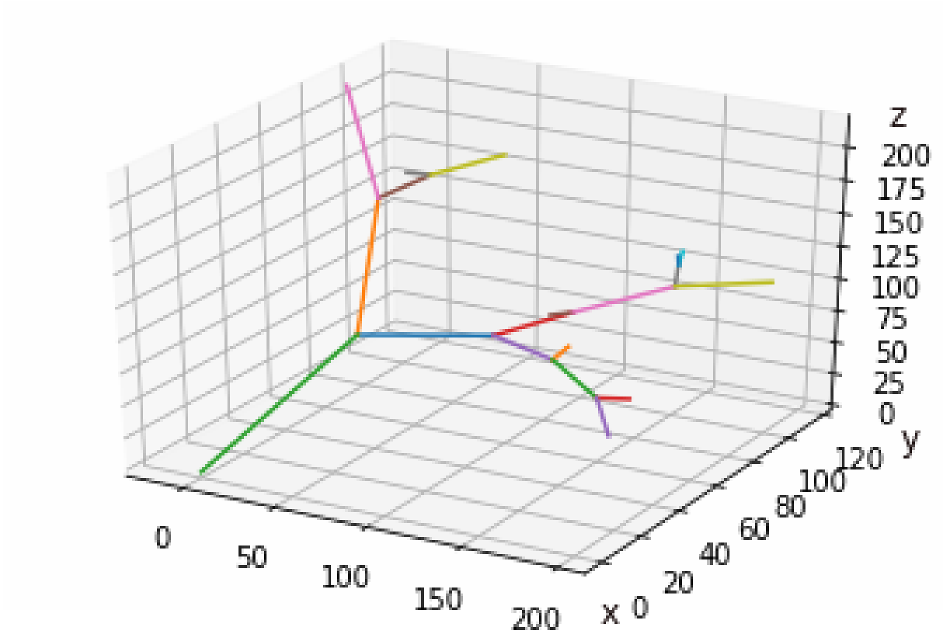


Figure 3.6: Simulation of a jet.

3.3 Jet clustering simulation

3.3.1 Jet clustering algorithms. Jet clustering algorithms are an efficient way to analyse the data, coming from the hadronization state, and define the jets. The production of one parton appears in the detector as a collection of particles. Then the definition of one parton is equal to one jet is no longer valid anymore. since there is ambiguity in what "one parton" means after the parton shower has gone through several splittings. Then to define the jets we need to cluster these pseudo-jets, in order to find the actual jets. We need to use clustering algorithms, which lead to jets whose shape is not affected by the noises in the environment, such as underlying events and pileup contamination. (Cacciari et al., 2008a). The recombination using k_t and Cambridge/Aachen algorithms gives us this feature. These algorithms are based on the concept of hadrons clustering which is the inverse of the parton shower concept. The parton shower takes the one parton and divides it into two partons. However, the clustering algorithm does the opposite. It takes two partons and clusters them to one parton, searching for the original parton. The selection process to cluster particles is controlled by comparing between the distances, which are calculated from these two Equations (Cacciari et al., 2008a).

$$d_{ij} = \min(k_{ti}^{2p}, k_{tj}^{2p}) \frac{\triangle_{ij}^2}{R^2}, \tag{3.3.1}$$

$$d_{iB} = k_{ti}^{2p}, \tag{3.3.2}$$

where

- R is the active area (radius of the jet cone).
- There is also the p parameter, which we actually use it as switch between the different algorithms. For $p > 0$ the behaviour of the jet algorithm becomes similar to the observed one from the k_t algorithm. In the case when $p = 0$ it represents the Cambridge/Aachen algorithm. Finally for $p < 0$, it corresponds to the anti-k_t clustering algorithm (Cacciari et al., 2008a).
- K_T: is the transverse momentum, which we will replace with p_T through our discussion. The transverse momentum p_T is the momentum in the perpendicular plane to the beam axis, which is the axis of the collision, and is described by Equation 3.3.3 (Cacciari et al., 2008a) and following Figure.

$$p_T = \sqrt{p_x^2 + p_y^2} \tag{3.3.3}$$

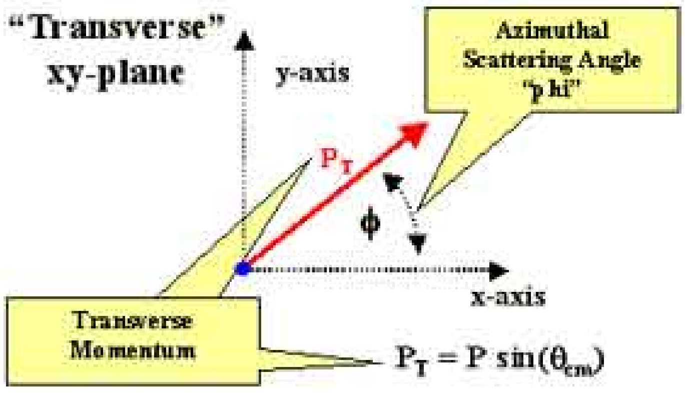


Figure 3.7: The transverse momentum (Win, 2012)

- $\triangle_{ij}$: is the angular distance, which is described by Equation 3.3.4.

$$\triangle_{ij}^2 = (y_i - yj)^2 + (\phi_i - \phi_j)^2, \tag{3.3.4}$$

where

  - y is the rapidity parameter which we use instead of the polar angle θ. It is common in particle experimental physics to define the pseudorapidity using the, rapidity using Equation 3.3.5.

$$y = \frac{1}{2} log \frac{E + p_z}{E - p_z} \tag{3.3.5}$$

    The following figure shows the values of the pseudorapidity η according to the position of the particle.

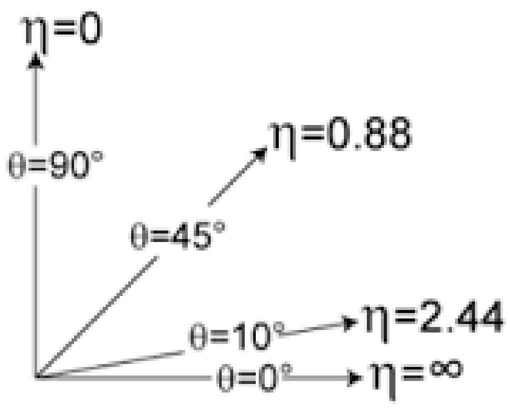


Figure 3.8: the pseudorapidity and θ angle (Rap, 2007)

    The zero angle is usually along the beam axis. As the polar angle approaches zero, at this angle, the pseudo-rapidity tends towards infinity, and the particles which are produced approaching this angle are considered as missing information, which can not be detected.

  - E is the energy of the produced particle.
  - p_z is the momentum along the beam axis.

3.3.2 Jet clustering technique. The parton shower simulation takes the initial four vector parton, and splits it in to many collimated final four vector particles, which represents the final state stable hadrons. This collimated spray of particles are not yet a jet. Even if we have all of the final hadrons originally from the same parton, there is still an ambiguity for the definition of the jet, and we can see that from the first splitting angle, in the jet simulation tree in Figure 3.6, where it seems like they come from two partons not just one. To avoid this ambiguity we should go through a clustering algorithm.

In our model, we take the final state hadrons, which are called pseudo-jets, and cluster them until we have groups of pseudo-jets that are distinct from each other. We will start looking for the jets by calculating the beam distances for each pseudo-jet, which are called d_{iB}, which is described in Equation 3.3.2, and calculate the distance between the pseudo jets themselves, which we called d_{ij}, Equation 3.3.1. After calculating the distances we will find the smallest distance, either d_{iB} or d_{ij}. If it belongs to the set of d_{iB}, then we consider pseudo-jet i as a jet and remove it from the cluster sequence. However, if the smallest distance is the d_{ij}, then we will cluster the particles i and j, and add the new clustering particle to our set of pseudo-jets in place of the previous two pseudo jets, i an j, and recalculate the distances again.

An example if we have 3 particles in the final state then we will have a set of six distances $[d_{12}$,d_{23},d_{31},d_{1B},d_{2B},$d_{3B}]$ three of them represent the distances between the particles, and the other three distances represent the beam distances. In our simulation we will work with three clustering algorithms, which are k_t, anti-k_t and Cambridge/Aachen algorithms, and we switch between these three algorithms by choosing the value of parameter p in these two Equations 3.3.1 and 3.3.2.

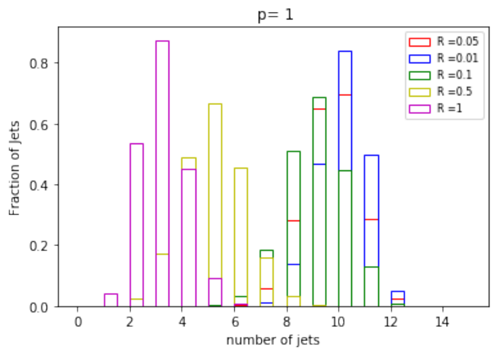
The next algorithm describes this clustering process.

```
Result: set of jets
Final-state=N final state particles;
$d_{ij}$ distances= $N(N-1)/2$ $d_{ij}$ distances ;
$d_iB$ distances=N $d_{ij}$ distances ;
set-of-distances=$d_iB$ distances+$d_{ij}$ distances
while number of final state particles $>$ 0 do
    if the smallest distans in set-of-distanse belonging to $d_{ij}$ not to $d_{iB}$ then
        delete i and j from Final-state set ;
        add the clustering particle to the set ;
        recalculate the distances again;
    else
        consider the particle i as a jet ;
        delete particle i from the set;
        recalculate the distances again;
    end
end
```

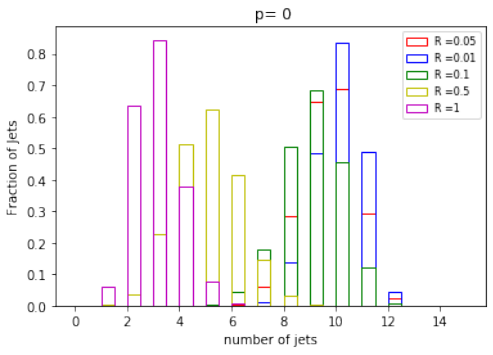
Algorithm 4: Algorithm to describe the clustering process

3.3.3 Number of jets. One observable which we can measure after the clustering process is the number of the jets, which is affected by the R parameter, and is not really affected by the p parameter. As we see from Figure 3.9 The highest number of jets appears when R is a small value, $R = 0.01$, and the smallest numbers of jets appear when R is a big value $R = 1$. We can explain this by looking at Equation 3.3.1. When R is small it makes the value of d_{ij} bigger, because it is in the denominator, then there will be more probability that d_{iB} is smaller than d_{ij}, which will lead to us have more jets. For big R values it makes the distance d_{ij} smaller, which means that more particles will cluster together until we find the jet.

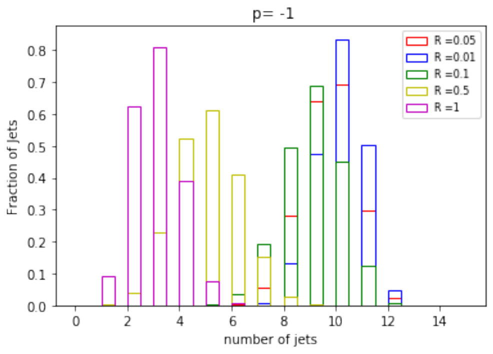
3.3.4 Number of constituents. After we get the number of jets, another observable can be introduced here that is the number of constituents. It has the opposite behaviour when comparing with the number of jets under the same value of the R parameter. If we have a small number of jets this means that each jet contains a big number of constituents, and if we have a big number of jets that means that each jet contains a small number of constituents. That can be seen in Figure 3.10, where if $R = 1$ then we will have a big number of constituents, and if $R = 0.01$ then we will have a small number of constituents.



(a) k_T algorithm

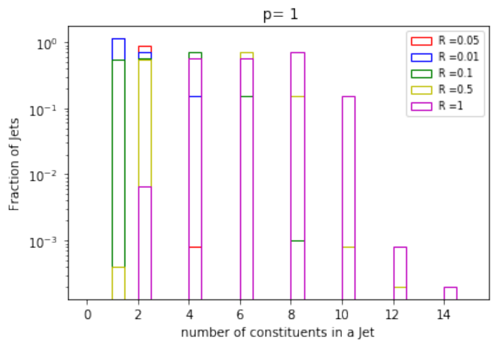


(b) Cambrige algorithm

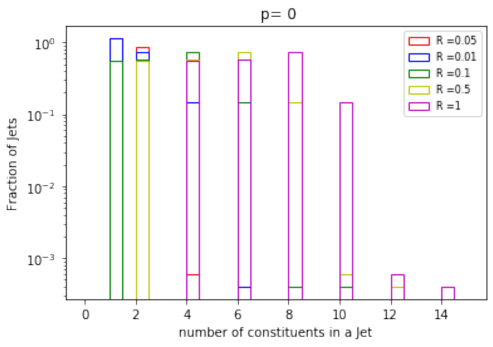


(c) anti $- k_T$ algorithm

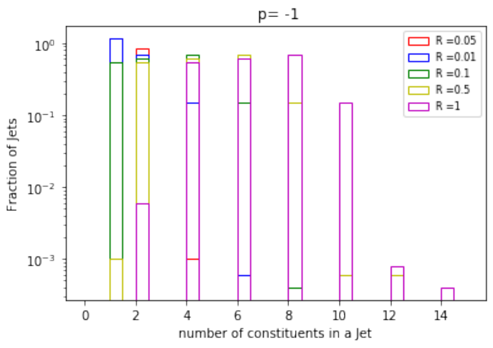
Figure 3.9: Python histograms represent the number of jets, after generating 10000 events and clustering them with different algorithms and different R values.



(a) k_T algorithm



(b) Cambrige algorithm



(c) anti $-\, k_T$ algorithm

Figure 3.10: Python histograms represent the number of constituents inside the jets, after generating 10000 events and clustering them with different algorithms and different R values.

3.4 Pythia/Fastjet

In high energy physics the tools to perform the simulation of collisions events, the parton shower, the hadronisation level, and the clustering algorithms are standardised. Which helps to make sure that all the people in the high energy community are using the same work environment and can share information and codes easily. In terms of the clustering algorithms, generally the used package is Fastjet, which is used in all the LHC experiments, which makes the algorithms for jet finding faster.

3.4.1 Pythia. Pythia is a program which is written in C++ and is an event generator for high-energy collisions. It consists of libraries which contain models for the initial-and final states, as well as multi partons interactions and particle decays. Currently it is one of the standard tools which is used in LHC studies (Sjöstrand et al., 2008). Pythia can simulate the interaction between pp, $\bar{p}p$, e^+e^+ and $\mu^+\mu^-$, and produce a set of events for any process we choose, and perform the parton shower and hadronisation levels to produce a set of final state particles that can then be used for analysis. Figure 3.11 shows the relation between the classes in Pythia, there are based on three different classes: ProcessLevel, PartonLevel and HadronLevel. The thick arrows show the carriage of different tasks from Pythia to the main classes. The narrow arrows show the flow of information between the classes through the Events, such as the event process which carries the information from the ProcessLevel to the PartonLevel, and the Event event which the information between PartonLevel to the HadronLevel.

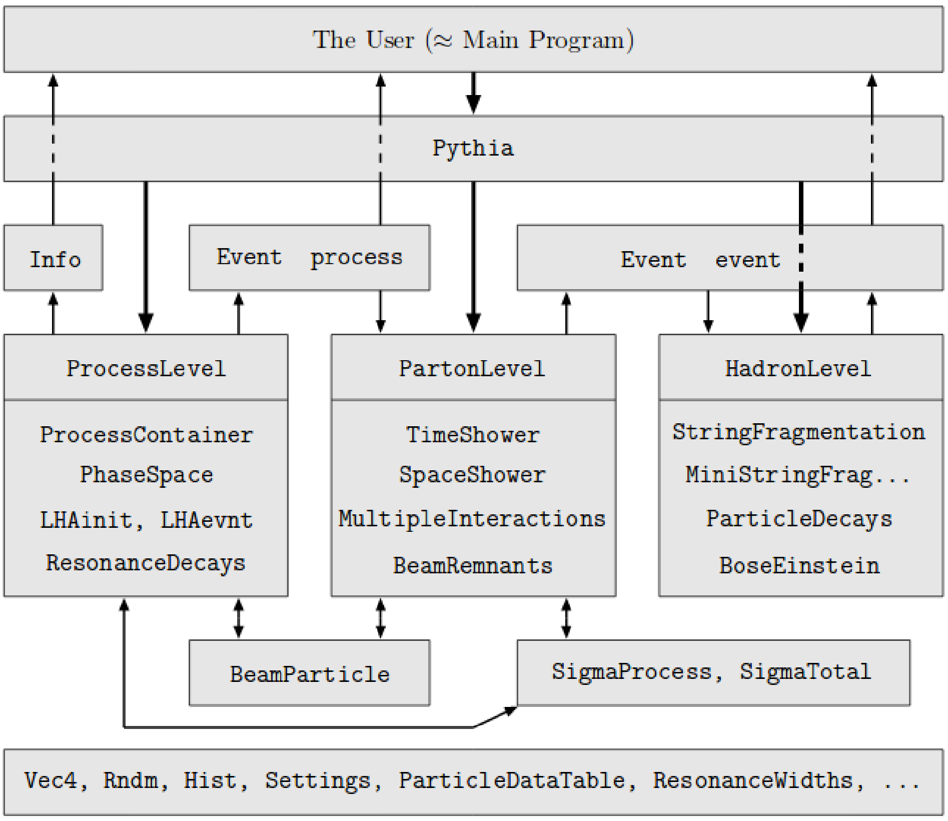


Figure 3.11: The relation between the main classes in Pythia 8 the thick arrows represent the physics tasks, and the narrow arrows represent the flow of information, between the classes (Sjöstrand et al., 2008).

In our project, after running Pythia8, We generate 10000 events and save it to the root output file a flat n-tuple that contains truth vectors of the relevant particles (W, Z) for truth tagging later along with vector<double> branches for all of the truth particles with final state particles (Sjöstrand et al., 2014).

3.4.2 Fastjet. Fastjet provides libraries for the clustering algorithms such as k_t, anti-k_t and Cambridge/Aachen clustering algorithms. Using plug in facility, further jet algorithms would be included in the fastjet interface. Now the studies in the LHC, which is a high-noise environment, needs more tools than just jet finding algorithms. One of these useful tools is the area of the jet, which is used due to the correction of underline events and the pileup contamination (Amsler et al.). Fastjet provides libraries for the clustering algorithms such as k_t, anti-k_t and Cambridge/Aachen clustering algorithms. More tools which are provided by Fastjet, available online at (http://fastjet.fr).

The program also has new algorithms, which improve the speed of the events containing up to 10^4 particles. In our project, the next step after generating events from Pythia 8, will be Fastjet which will take as input the set of final state particles and put them through many of sequential recombination algorithms. Here are some of the kinematic access functions (Cacciari et al., 2011), that will be used in the project:

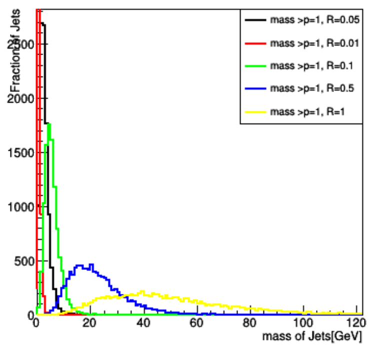
- double E() const, which returns the energy of the jet.
- double pt() const, which returns the transverse momentum of the jet.
- double m() const, which returns the mass of the jet.
- double eta() const, which returns the rapidity of the jet.
- double phi() const, which returns the phi angle between (0,.2π).

4. Jet's observables

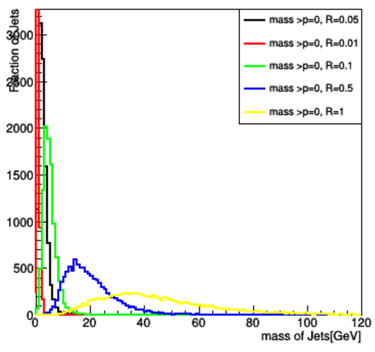
The main point of studying the jets is to study the proprieties of the produced partons from high energy collisions. To study the jets we need to study the number of observables, which will help us to define the jets more clearly. The basic parameters that we will change in our simulations, to study the properties of the jets, are the p and R parameters, which have been defined in Chapter 2.

4.1 Observables from Pythia/Fastjet model

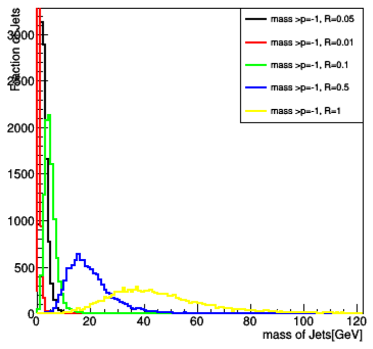
4.1.1 Mass. A useful observable to measure is the mass. After generating 10000 events using, Pythia8, we cluster those events with the k_t, anti-k_t and Cambridge/Aachen algorithms. Using the kinematic access function, double m() const in Fastjet, we get the mass of the jet. Plotting the mass data using the root histogram we get Figure 4.1. As we see from Figure 4.1athe largest the value of R, say $R = 1$, the corresponding mass values are high. For a small R value, say $R = 0.01$, we find this corresponds to small mass values. That is can be understood from our discussion about the number of constituents of the jet in Chapter 3, where for small values of R we have a small number of constituents inside the jets which leads to a small mass of the jet. For big R values, it is corresponding to the high number of constituents, which leads to a high mass value for the jet.



(a) k_T algorithm



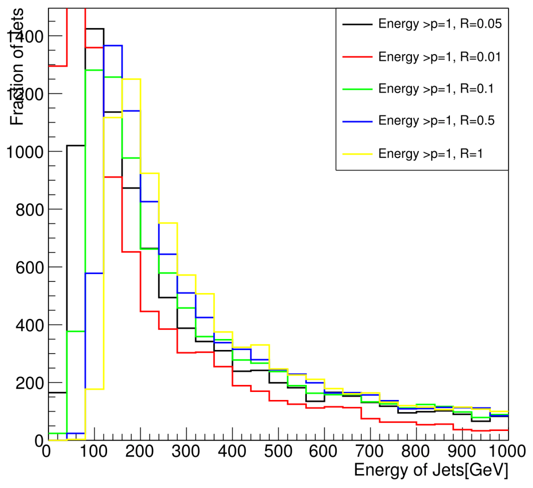
(b) Cambrige algorithm



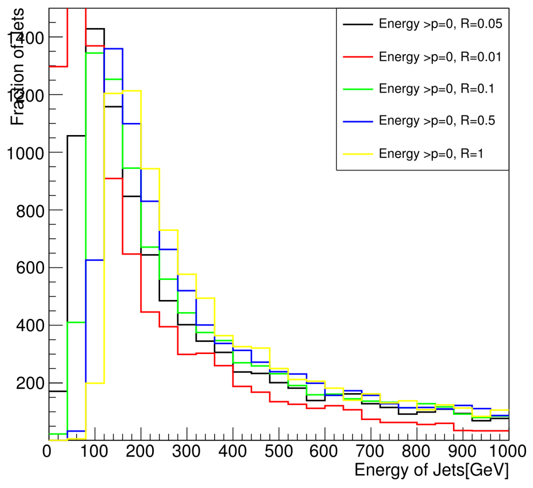
(c) anti $- k_T$ algorithm

Figure 4.1: Root histogram representation of the mass distribution of the jets from events were generated with Pythia8, they have been clustered with k_t, anti-k_t and Cambridge/Aachen , for different R values.

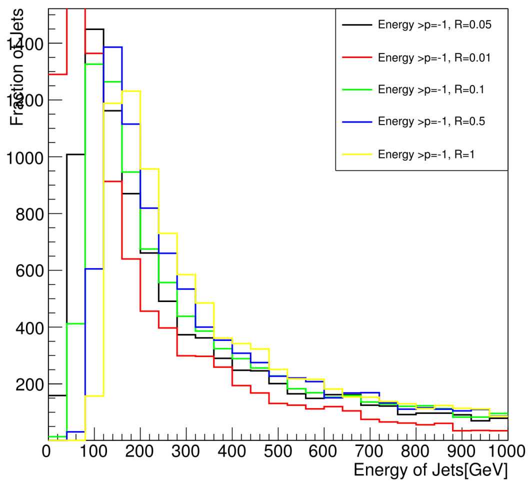
4.1.2 Energy. Another important observable is the energy of the jet, which is affected by the chosen R values. Similar to the discussion in Chapter 3, Section 3.1.3 the R value affects the number of constituents inside the jet, which of course affects the energy of the jet as well. If we decrease the number of constituents inside the jet we also decrease its energy, and if we increase it we increase its energy. This is described in Figure 4.2. In this figure we compare between the energy of the jet using the three clustering algorithms k_t, anti-k_t and Cambridge/Aachen algorithms, with different values of R. We notice that for a small value of R, $R = 0.01$, the energy of the jet is small, and for a big value of R, $R = 1$, the energy of the jet is big.



(a) k_T algorithm



(b) Cambrige algorithm



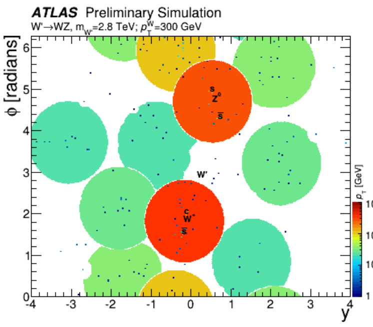
(c) anti $-$ k_T algorithm

Figure 4.2: Root histogram representing the Energy distribution of jets from events we have generated with Pythia8, which have been clustered with k_t, anti-k_T and Cambridge/Aachen algorithms using Fastjet, for different R values.

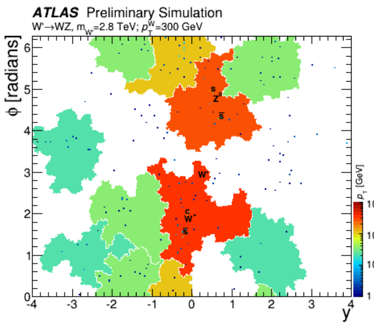
4.2 Observables from Python model

4.2.1 Jet shape. One of the most important properties of the jet is its shape. The shape of the jet changes according to the clustering algorithm that we use. The p parameter switches between the algorithms as follows, if $p > 0$ then we use the k_t-algorithm, if $p = 0$ we use Cambridge/Aachen algorithm and if $p < 0$ then we work with anti-k_t-algorithm.

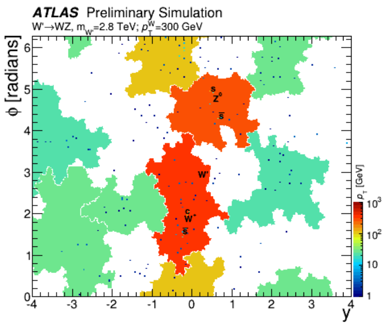
There is an ATLAS simulation of a di-jet event, which describes the shape of the jet under different clustering algorithms.



(a) anti-k_T algorithm



(b) Cambridge/Aachen algorithm



(c) k_T algorithm

Figure 4.3: ATLAS simulation of a di-jet event (Pythia8), which describe the shape of the jet that has been clustered by k_t, anti-k_T and Cambridge/Aachen algorithms with an active area of (R=1.0) (CER, 2012).

As we see the anti-k_t algorithm has the most regular shape, and this is because the cone shape of the jet in this algorithm is less affected by the noise in the environment (Cacciari et al., 2008b). We can show that in our model by introducing an observable which we called the variance of d_{1i}.

The radiated soft particles accumulate around the hard one. From Figure 4.3 we see that the anti-k_t algorithm has the most regular circular shape. We can then differentiate between these three algorithms by calculating the distances between the hard particle, which is placed in the centre, and each soft particle. If those distances approach each other, that means that they have a circlar shape. We can show that by introducing the variance observable. After the clustering process that we have done in Chapter 3, we choose the highest energy jet and choose from its constituents the highest energy on, then calculate the distance d_{1i}, which we defined in Equation 3.3.1, between the highest energy constituent 1 and the other constituents i. Calculating the variance between the distances and generating 10000 events, we then showing the log of the data in histograms, in Figure 4.4.

In Figure 4.4 we can see that the anti-k_t algorithm has negative log variance values, which means that the anti-k_t algorithm has variance values which are very small and approaching zero. Also it has the most regular circular shape when comparing with the k_t and Cambridge/Aachen algorithms.

4.2.2 Pseudo-mass. As the particles in our Python model are massless, then to describe the mass we will use the mass definition in general relativity (Jaramillo and Gourgoulhon, 2011), and we will call it the pseudo-mass. We describe the pseudo-mass in our model by equation 4.2.1

$$pseudo - mass(J) = E(j_1) \times E(j_2) \times \triangle R(j_1, j_2) \tag{4.2.1}$$

where $E(j_1)$ and $E(j_2)$ are the energies of the first splitting particles in the jet tree, and $\triangle R(j_1, j_2)$ is the angler distance between them.

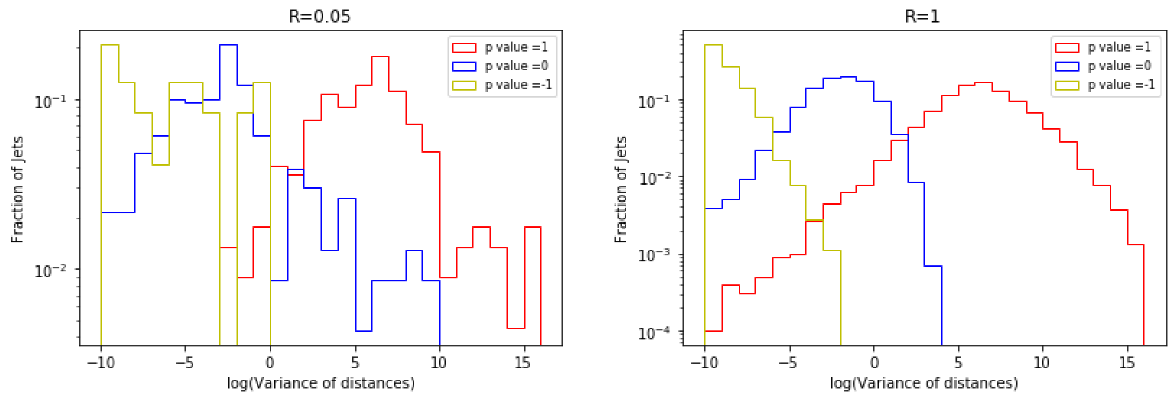


Figure 4.4: Python histogram shows the variance σ^2 in the distances between the hard particle and the soft particles in different two active area $R = 0.05$ and $R = 1$

After the clustering process that we have done in Chapter 3, we choose the highest energy jet and calculate the pseudo-mass for it by Equation 4.2.1. We generate 10000 events and each time we calculate the pseudo-mass from the generated events. Then we plot the histogram for the pseudo mass distribution with different values of R. As in sub-section 4.1.1 of the mass, we see that mass and pseudo-mass are affected by same way corresponding to the chosen R parameter values.

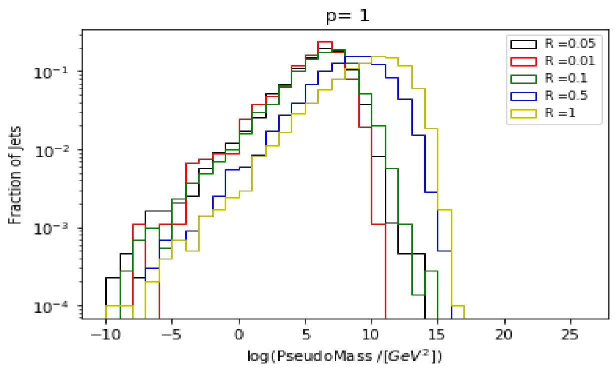


Figure 4.5: Python plot showing the pseudo-mass distribution in different active areas R (Ellis et al., 2008)

As the definition of pseudo-mass contains the angler distance, $\triangle R(j_1, j_2)$, between the first two splitting particles in the jet tree, then we can use the pseudo-mass also to determine the shape of the jet using different clustering algorithms. As we see from Figure 4.6, which compares the three algorithms in different active areas ($R = 1$ and $R = 0.1$). The pseudo-mass for k_t algorithm is bigger than the pseudo-mass for anti-k_t algorithm, which means that the angler distance $\triangle R(j_1, j_2)$ for the k_t algorithm is also bigger, and that describes the shape of the jet in these algorithms.

4.2.3 Energy. By the same discussion as in sub-section 4.2.3, we can explain how the parameter R affects the eEnergy of the jet. Using Python to generate 10000 events and clustering these events using different algorithms (p=0, p=1 and p=-1), in different active areas, we get Figure 4.7, which shows the energy distribution of the jets.

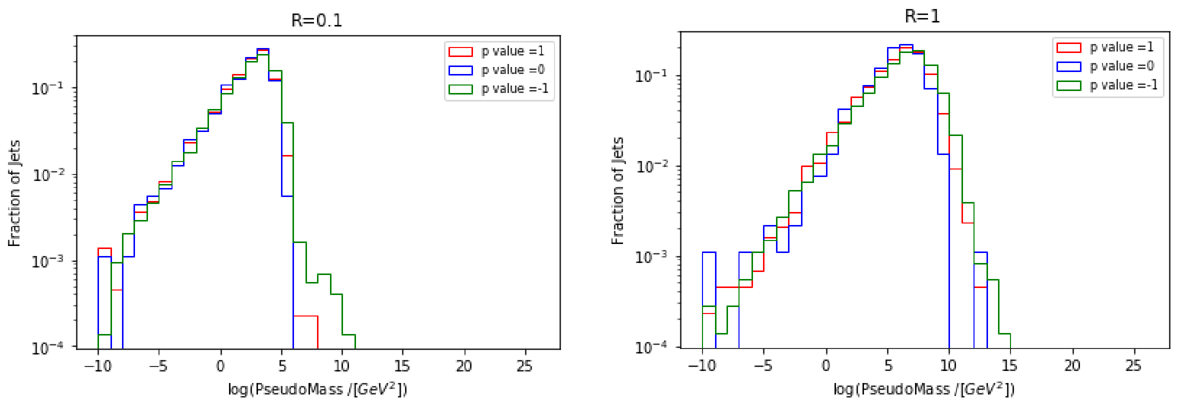
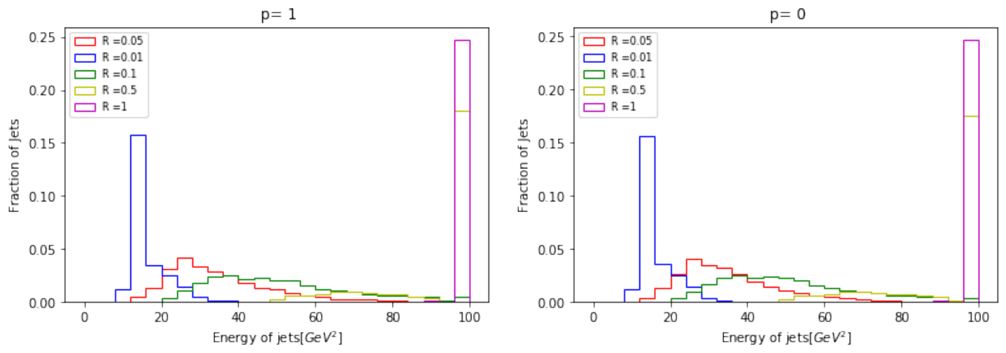
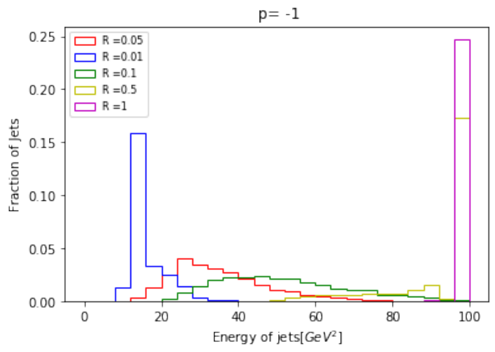


Figure 4.6: Python histograms of the log(pseudo-mass) with different clustering algorithms have been clustered with k_t, anti-k_T and Cambridge/Aachen algorithms, in two active areas $R = 0.1$ and $R = 1$



(a) k_T algorithm

(b) Cambrige algorithm



(c) anti $-$ k_T algorithm

Figure 4.7: Python histograms representing the energy distribution of the jets, which have been clustered with k_t, anti-k_T and Cambridge/Aachen algorithms, for different R values.

5. Conclusion

Through this project we have seen how computers can enable us to broaden our minds, as well as help us understand and simulate physics problems in order to find a solution which can be applied in real experiments. As for the case of jet simulations, the simulation gives us more understanding of the physics behind the jets in terms of the collision process in high energy physics, patron shower level, hadronisation level and detectors. It also enables us to generate events similar to the real one and study their features.

Furthermore, it helps us to study the real data from the LHC, since the production of one parton appears in the detector as a shower of particles, and using different clustering algorithms enables us to cluster these soft particles together and reach the hard initial particles that we are interested in produced from the collision process. Introducing a number of observables enables us to study the properties of these jets and know more about their shape, energy and mass, which of course provides an insight in understanding the collision process.

Acknowledgements

Firstly, I thank god that has given me power and patience to complete my research.

I thank my parents for encouraging me all the time and believe in me.

I thank my co-supervisor, Sahal yacoob, that has helped me and introduced me to his other students that have also been of assistance to me.

I thank my AIMS family 2016/2017, especially my friends to assist my through this research and courage me all the time.

Finally, I can't find words to thank my supervisor, Samuel Meehan, for all knowledge and guidance he has shared with me for my research project. He has shown diligence and perseverance in order to help me learn and understand the content of this research. His input has not only motivated me to do my research, but also to acquire to be a good future scientist.

References

Rapidity angle, 2007. URL https://en.wikipedia.org/wiki/Pseudorapidity#/media/File:Pseudorapidity2.png.

ATLAS experiment, 2012.

Jet 2012 Monte Carlo Event Displays, 2012. URL https://twiki.cern.ch/twiki/bin/view/AtlasPublic/JetEtmissApprovedBOOST2014EventDisplays.

Pseudo-Rapidity, Azimuthal Angle, and Transverse Momentum, 2012. URL https://www-cdf.fnal.gov/physics/new/qcd/ue_escan/etaphi.html.

C. Amsler et al. 33. monte carlo techniques.

A. Buckley, J. Butterworth, S. Gieseke, D. Grellscheid, S. Höche, H. Hoeth, F. Krauss, L. Lönnblad, E. Nurse, P. Richardson, et al. General-purpose event generators for lhc physics. *Physics Reports*, 504(5):145–233, 2011.

M. Cacciari, G. P. Salam, and G. Soyez. The Anti-k(t) jet clustering algorithm. *JHEP*, 04:063, 2008a. doi: 10.1088/1126-6708/2008/04/063.

M. Cacciari, G. P. Salam, and G. Soyez. The anti-kt jet clustering algorithm. *Journal of High Energy Physics*, 2008(04):063, 2008b.

M. Cacciari, G. P. Salam, and G. Soyez. Fastjet user manual. *arXiv preprint arXiv:1111.6097*, 2011.

M. Cacciari, G. P. Salam, and G. Soyez. FastJet User Manual. *Eur. Phys. J.*, C72:1896, 2012. doi: 10.1140/epjc/s10052-012-1896-2.

R. Ellis, W. Stirling, and B. Webber. Qcd and collider physics (cambridge monographs on particle physics, nuclear physics and cosmology), 2003.

S. Ellis. "What is a Jet?", 2010. URL https://twiki.cern.ch/twiki/bin/view/AtlasPublic/JetEtmissApprovedBOOST2014EventDisplays.

S. Ellis, J. Huston, K. Hatakeyama, P. Loch, and M. Toennesmann. Jets in hadron–hadron collisions. *Progress in Particle and Nuclear Physics*, 60(2):484–551, 2008.

here. jet simulation. bitbucket,Nihal, https://bitbucket.org/nihalbahaa/jet-simulation/overview, 2017.

J. L. Jaramillo and E. Gourgoulhon. Mass and Angular Momentum in General Relativity. *Fundam. Theor. Phys.*, 162:87–124, 2011. doi: 10.1007/978-90-481-3015-3_4. [,87(2010)].

T. Sjöstrand, S. Mrenna, and P. Skands. A brief introduction to pythia 8.1. *Computer Physics Communications*, 178(11):852–867, 2008.

T. Sjöstrand, R. Corke, and P. Skands. Pythia 8 worksheet. 2014.

B. Ydri. *Computational Physics: An Introduction to Monte Carlo Simulations of Matrix Field Theory*. WSP, 2016. URL https://inspirehep.net/record/1375035/files/arXiv:1506.02567.pdf.