Create your own toy Monte Carlo

Tutorial created June, 2016

Idea by Dr. Samuel Meehan

Fiona Pons

Abstract

This is a short instructive tutorial intended for summer students to get familiar with the Monte Carlo method as used in High Energy Physics. At the end of this tutorial you should be able to generate a 3D model of a parton shower respecting some basic physics (like conservation of energy and momentum). Following this tutorial should help understand the concept of (not recreate) Monte Carlo simulations like PYTHIA used at CERN (so please excuse simplifications and incorrect physics). The assignments could be solved in any programming language but the instruction assumes that you use Python.

Contents

Step 1 : What is Monte Carlo?!??

Asked this question the first thing people (including me) do is google it. Here is the result:

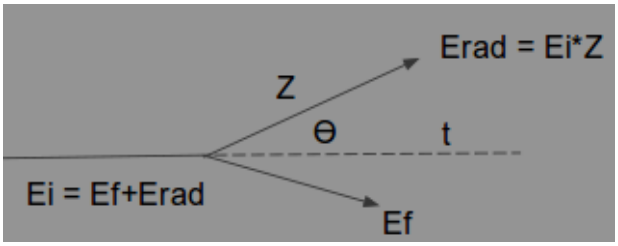
Monte Carlo officially refers to an administrative area of the Principality of Monaco, specifically the ward of Monte Carlo/Spelugues, where the Monte Carlo Casino is located. Informally ...
(beginning of the Wikipedia article "Monte Carlo")

What we are actually looking for is the "Monte Carlo method".
Like the casino it got its name from the Monte Carlo method attempts to profit from a lot of gambling. To be more precise: Monte Carlo samples many many random events (basically just draws and combines random numbers). Put together according to the given problem, statistics allows us to make a statement about the problem's solution ("Law of large numbers"). But why bother? Would not an analytical approach be better? Often yes. Sometimes the analytical solution is more complicated or there is none. Especially in quantum mechanics and QCD we only work with probability distributions. But when we measure, every wave function collapses to one definite value. So in order to get something comparable, a "model event", we need to pick a value for every variable in our process.(Such a variable could be the time after which a specific particle decays in a radioactive sample). What Monte Carlo does is generate a random value for every variable in the process according to that variables probability distribution. Then the output is the result/final state after inputting those random numbers. The output is called event and does not have much meaning on its own. More important is to generate a large number of these Monte Carlo events. A large number gives us statistics, averages, that will correspond to data if the underlying physics model used for Monte Carlo is correct.

Step 2 : Realize what this tutorial is about

The Monte Carlo method has a wider field of applications. This tutorial focuses on how it applies to High Energy Physics at CERN. This tutorial leads you step-by-step to creating a simple 3D model of a parton shower (not respecting much physics).
A high energy proton-proton collision will result in quarks flying away. Quarks have color charge. Therefore moving quark radiate. As the gluon is the mediator for the strong interaction (color) the quark will eventually loose part of his energy emitting a gluon.

Step 2 continued on next page...



The figure shows the vertice for such a radiation process. The gluon takes the fraction $Z = \frac{E_{rad}}{E_i}$ of the particles initial energy E_i. It is radiated at the angle θ to the quarks initial direction. The quark flies on with slightly less energy and slightly altered direction

The quark will travel on eventually radiating another gluon and so on until the energy limit at which hadronization starts. The gluon will decay into two quarks that will later radiate gluons that will decay into quarks and so on. The result is a shower. Ignoring particle types we have a shower of partons decaying into two partons according to the above vertice.
How to use this instruction?
- Boxes indicate assignments or important pictures
- when you reach in assignment you have already read all you need to solve it
→ do it immediately and do not read on as later text might refer to the solution
- To give you the chance to think about a problem part of the information is moved and referenced as hints
- hints also help you when you are stuck
- the hint page together with example solutions is at the end

Step 3 : The power of the pseudorandom number generator (and the Python manual)

The whole Monte Carlo principle is based on random numbers, on having a big number of random numbers. Therefore now is the time to get familiar with your computer's random generators (*Note: hints and example solutions are given in Python*). Your computer can generate random numbers following a Gaussian, Landau, exponential, uniform and probably some other distribution. Still for solving all the following problems you should limit yourself to the basic pseudorandom number generator, in Python: random.random(). That function gives you uniformly distributed numbers between 0 and 1. Why not use the nice preprogrammed ones like the Gaussian distributed random generator? In case the reason "it makes the assignments more interesting and challenging" is not enough for you, consider this: "What if your probability distribution has the shape of the New York skyline?" After this tutorial you will not be restricted to the standard distribution functions anymore. You will also love the Python manual. Google "python + whatever you want to do" and most of the time you will get a link to the Python manual and a function that does exactly that. Additionally Python offers a lot of different graphing options with example code. Normally it is easier to see what went wrong if you plot your solution instead of staring at a list of numbers.

Step 3 continued on next page...

Assignment 1

It is not very physical, but imagine your gluon has to take at least 25% of the particles energy and cannot have more than 75%. Within these limits it does not have a preferred energy.
Write a short program that gives you the gluon's energy as fraction of the total energy. Let the program generate many events and plot them in a histogram. (*If you are unfamiliar with python histograms check the python library "matplotlib" manual. A histogram is also a great way to check if your program is doing, what it is supposed to do. Think about how this histogram should look like.) If you get stuck, read hint 3.1*

Assignment 2

Now imagine that the angle theta at which the gluon is emitted can only be between 0 and $\frac{\pi}{2}$ and follows a Gaussian distribution (again not physical). Remember that you are only supposed to use the uniform random distribution. Now this is a lot harder to program than assignment 1. Think about it for a bit.

Read hint 3.2.1 .

Your random distribution should follow a Gaussian independent of how many random theta values you need. This means every new number should have a Gaussian probability distribution. Do not make a cut according to how many numbers of which value you already have.

Read hint 3.2.2 .

For uniformly distributed random numbers everything between the minimum and the maximum is equally probable. For Gaussian distributed numbers the probability of a number is given by evaluating the Gaussian at that number.

Read hint 3.2.3 .

The described method is called accept-and-reject.

Program the random generator for the gluon's emission angle using the accept-and-reject method. Again plot your result in a histogram and check if it looks Gaussian. *If the peak looks cut, read hint 3.2.4*

Step 4 : Get more familiar with accept-and-reject method

The accept-and-reject method is basic for Monte Carlo. It has more uses than just a random generator with a user-defined distribution function. Here is an example: By randomly picking points in a square we can calculate π (ok, not exactly but approximate it well).

Calculating π

Try to find an approach similar to step 3 assignment 2. Rereading hint 3.2.3 should help. What if you inscribe a circle in your square?

Write a program that calculates π with the accept-and-reject method. *If you get stuck read hint 4.1 . If you are still stuck read hint 4.2*

Calculate π several times and plot results in a histogram.

Calculating the quality of the π calculation

As this is an approximative calculation you should ask yourself about the quality of your approximation. You probably compared the results with the value of π you know. We want to use statistics to estimate the uncertainty. You already plotted a histogram with several π-values. One method to calculate and reduce the uncertainty is to calculate π several times and get mean and standard deviation σ. Alter the number of π-values you use and watch how σ changes. Another way to impact the uncertainty is to change the number of random points you use in the calculation. Again calculate several π values and get the mean and σ. Keep the number of π-values constant and alter the number of random points. Watch how σ changes.

Choose one of the described method and plot the uncertainty vs the number of trials/random points.

Step 5 : Combine into one random event

Before we had Z and θ seperately. The radiated particle in the vertice has both and gets both at the same time.

Combine random generators

Combine your programs. Generate many random events having Z and θ and plot into a histogram. *This means in one loop and both histograms on one canvas.*

Combine into one histogram

Play with 3D plotting and put Z and θ in one 3D histogram. *This is just to get familiar with 3D histograms. Here it is still relatively easy to see if the histogram is correct. Later you might want to make deductions from them and must be able to rely on their plotting correctness.*

Step 6 : Go more physical

Now that you are more familiar with generating random numbers following a user-defined distribution you can implement "more physical" distributions. The emission is co-linear (θ-distribution = $\frac{1}{x}$). The radiated particle only takes a small fraction of the original energy (Z-distribution = $\frac{1}{x}$).

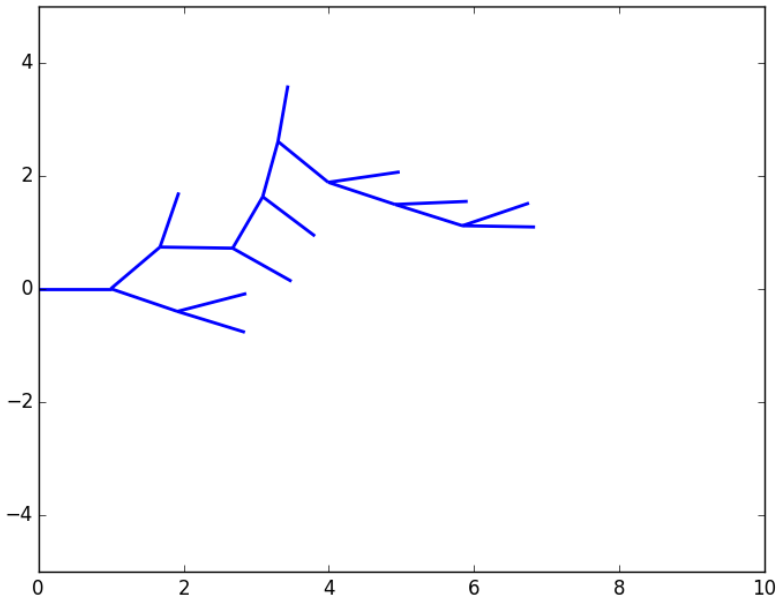
Update your program with the more physical distribution functions. Use $\frac{1}{x+const}$ adding a small constant in order to avoid the singularity at 0. Experiment with the constant so smaller angles and energies are notably preferred. (again histograms will help you to check)

Step 7 : String vertices together

This small description belongs to a part that is a lot of work. While doing this, keep in mind that this is the preparation for a 2D and later 3D picture.

Now string several vertices together to create a shower. Print the list of created and decayed particles together with their energies. *For useful advice read hints 7.1 and 7.2*

Step 8 : Finish parton shower in 2D



First 2D graphic

Complete everything you need for a 2D picture of your shower. (hint 8.1.1)

Create a 2D plot of your parton shower. *(just for plotting read hint 8.1.2)*

Conserve energy and display it in plot

To attempt a little more physical simulation energy should be conserved. Your initial particle starts with Z = 1 as it carries the total energy. Every energy from now on should be set in relation to that initial particles energy. So if you add up the final state particles energies the result should be 1. To keep track of how energy gets distributed among the partons invent a way to display the energy (value or at least high-or-low energy). If you feel uncreative read hint 8.2.1 .

Modify your simulation to conserve energy. Display every parton's energy in a way of your choosing.

Step 9 : But our world has more than 2 dimensions ...

If we stick closer to reality the emission will not only have an opening angle θ but also an azimuthal angle ϕ. There is no preferred ϕ which means all azimuthal angles are equally likely.

Implement the random generator for ϕ in your program. Think about how this gives you a 3D version of your shower. *Hint 9.1 and 9.2*

Step 10 : Improving the physics content further still being miles away from reality

A very fundamental concept of physics is the conservation of momentum, so it would be nice if our model respected that.Feel free to use the approximation that we are dealing with highly energetic particles having a large momentum compared to their mass (which we are). Careful to what extent you can apply this approximation. *If you are uncertain read hint 10.1*

Let momentum be conserved. (Check by adding up all final state momenta. And as you are already checking momentum conservation, also check if energy is still conserved.)

Did you just let a random number decide whether a particle is stable or not? That is a valid approach. But we now want to implement a better stability condition. Quarks start clustering, binding into hadrons, when they get less energetic. Therefore an energy limit would be a good stability condition.

Pick an energy limit (no matter which, just experiment how many final particles you get and want). Program that particles with energies below this limit stop decaying.

Step 11 : Tying up loose ends

Look, a 3D parton shower!

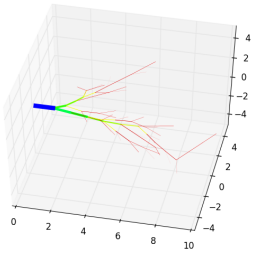
Congratulation! You have done a great job and almost reached the end of this tutorial. You have created a 3D model of a parton shower and now the only thing left to do is figure out how to plot it as a 3D graphic.

Visualize your 3D parton shower.

Save event into a file

Basically you were done one step earlier. You already displayed your model but Monte Carlo is not about getting nice pictures. You need a lot of events (showers). You want to do further computation, like applying a jet clustering algorithm or just see how your detector would "see" the jet. Therefore you need your information in a format readable for a different program. So a picture is not enough. The data should be stored in a file, usually some kind of text file. A table structure in your file would be nice too as it makes separating the different variables a lot easier. The actual structure and file type is up to you as long as you are able to write a program that can correctly extract all the data from your file.

Store your parton shower data in a file. The file needs to contain at least all the energies and momenta of all final state particles.



.

FINALLY DONE !!!
Congratulation and have fun with your toy Monte Carlo!

Hints

Hint 3.1 : uniform distribution between 0.25 and 0.75

Hint 3.2.1 : What if you can "throw away" random numbers?

Hint 3.2.2 : What if you can use two random numbers for one random theta?

Hint 3.2.3 : In the pair of random numbers you generate the first one is the actual value and the second one decides whether we are going to use this value or draw a new random number pair. The criteria to use (accept) random number 1 is that random number 2 is below the function value of number 1. This works because the function value is the probability that number 1 occurs. Number 1's probability is for example 0.2 . Our uniform random generator gives us number 1 with the same probability as every other number, but we only want number 1 in 20% of the cases. Now we have our second number. Every value between 0 and 1 is equally probable. The range from 0.2 to 1 is 4 times bigger than the interval 0 to 0.2. So only in 1 out of 5 cases number 2 will be below 0.2 and we will keep number 1. This means in 20% of the cases we will have number 1 as random value and in 80% we will draw a new value. Thus number 1's probability decreased from equally probable to .2 times as probable just like we wanted it.

Hint 3.2.4 : Make sure your second random number can go as high as your highest function value. Otherwise all x with f(x) above number 2's maximum will be equally probable.

Hint 4.1 : compare areas of square and circle

Hint 4.2 : Take a large number of points. If x and y values are both uniformly distributed random numbers between 0 and 1, every point in the unit square is equally probable. Then you check whether or not the point is in the inscribed circle (use quarter circle, that is easier). The fraction of all points that is in the circle is equal to the fraction of the total area the circle occupies.

Hint 7.1 : KEEP TRACK OF EVERYTHING e.g. use a structure called dictionary to store all ever created particles with their specific properties (e.g. energy)

Hint 7.2 : make sure you do not end up in an infinite loop → you need a stopping criteria (can be random for now)

Hint 8.1.1 : You need every particles trajectory. To string vertices together, retrieve the end point of the mother particle's trajectory and let the daughter particle start from there

Hint 8.1.2 : store trajectories as lines (list of start and end point) and use Python's LineCollection

Hint 8.2.1 : ideas: relate the line width to the particles energy or color partons with an energy above a certain threshold blue and all below red

Hint 9.1 : Use a uniform distribution from 0 to 2π for ϕ

Step 12 continued on next page...

Hint 9.2 : θ is the opening angle of the cone which means the angle compared to the original direction. ϕ rotates it around the original direction. Read about rotation matrices for help.

Hint 10.1 : $E^2 = p^2c^2 + m^2c^4$. So if m is small $E \approx p$ but you cannot use only that for conservation of momentum. Calculate it through. If you replaced p by E everywhere only parallel emissions of both particles would be possible.