MANUAL BÁSICO DE LA SIMULACIÓN PYTHIA DE ALTAS ENERGÍAS.

Y INTERFASE CON LA SIMULACIÓN RÁPIDA DEL DETECTOR

ELABORADO POR: JESÚS ALBERTO VELAZQUEZ CORRAL

JSALBVLZC.TS21@GMAIL.COM

Índice

Repositorio

En la siguiente dirección, encontrarás el repositorio donde se han depositado los scripts detallados que se desarrollarán en este manual.

1. Introducción

La exploración del mundo subatómico y la comprensión de fenómenos fundamentales requieren herramientas poderosas que vayan más allá de las limitaciones de la teoría cuántica y la experimentación directa. En este contexto, los generadores de eventos Monte Carlo (MC) emergen como herramientas esenciales en la física de partículas de alta energía. Estos programas de simulación, también conocidos como generadores de eventos, están diseñados meticulosamente para simular en detalle los estados finales de colisiones de partículas a altas energías. Desde la interacción fuerte hasta la formación de partículas estables, los generadores de eventos reproducen virtualmente cada fase del proceso con el fin de proporcionar predicciones detalladas y comparables con datos experimentales.

La generación de eventos en colisiones de altas energías implica diversas fases esenciales para simular con precisión los resultados experimentales. A continuación, se proporciona una descripción detallada de las principales etapas de este proceso:

1.Dispersión dura (Hard-Scattering)

La simulación de eventos generados por Montecarlo comienza con el cálculo de la sección eficaz de dispersión dura, un cálculo obtenido a partir de la teoría de perturbaciones. Este cálculo está dominado por la interacción fuerte entre partones, que involucra quarks y gluones, y se describe mediante la teoría cuántica de campos conocida como cromodinámica cuántica (QCD). La interacción fuerte da lugar a la producción de partículas de alta energía, como bosones W o Z, quarks top y otras partículas fundamentales.

2.Cascada de partones (Parton Shower)

Después de la colisión de hadrones, como protones, a altas energías, donde la interacción fuerte (QCD) predomina, se producen emisiones de radiación fuerte, similar a la radiación electromagnética. Este fenómeno se modela como una cascada de partones, donde partículas adicionales, mayoritariamente quarks y gluones, se generan sucesivamente. La emisión de estos partones adicionales es esencial en la simulación de eventos en Montecarlo, ya que contribuye significativamente a la distribución de energía-momento en el evento.

3.Hadronización

Los quarks y gluones generados en la cascada de partones se combinan y fragmentan para formar tanto hadrones estables como hadrones inestables, que son observables, como mesones, bariones y partículas resultantes del decaimiento de hadrones inestables. Este proceso se conoce como hadronización o fragmentación y está gobernado por la cromodinámica cuántica (QCD). Esta fase es de vital importancia, ya que se encarga de convertir los partones en partículas que son observables en los detectores experimentales.[1]

4.Evento subyacente (Underlying Event)

Es llamada de esta manera a la actividad adicional en una colisión de partículas que no está directamente relacionada con el proceso de dispersión dura. Esto incluye la radiación inicial y final del estado (ISR y FSR), los restos de los haces (remanentes) y posiblemente interacciones de partones múltiples (MPI). Es esencial generar esta información, ya que en un análisis de datos experimentales, puede afectar las mediciones de observables físicos y la precisión de las predicciones teóricas. Generar esta información con Monte Carlo nos ayuda a obtener estas incertidumbres de los datos experimentales.

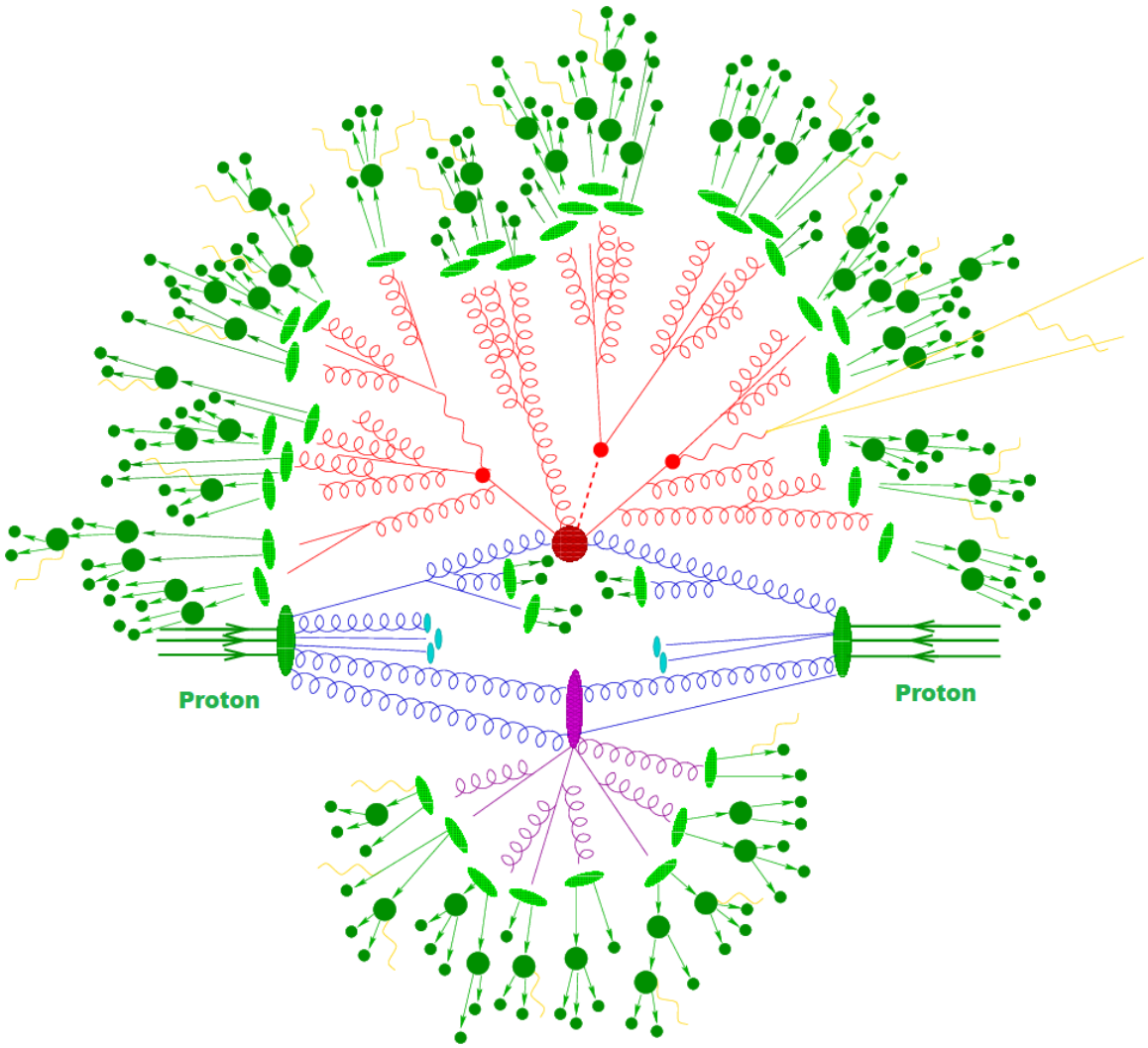

Figura 1: Esquema de una colisión hadrón-hadrón simulada por un generador de eventos de Montecarlo. La mancha roja en el centro representa la colisión fuerte, rodeada por una estructura en forma de árbol que representa la radiación de frenado simulada por las cascadas de partones. La mancha morada indica un evento secundario de dispersión fuerte. Las transiciones de partón a hadrón están representadas por manchas verdes claras, las manchas verdes oscuros indican decaimientos de hadrones, mientras que las líneas amarillas señalan la radiación de fotones.[2]

Estas fases permiten estudiar y comprender los procesos fundamentales que tienen lugar en colisiones de partículas de altas energías. Cada fase desempeña un papel crucial en la reproducción de los fenómenos observados en experimentos reales. Algunos de estos programas generadores de eventos se detienen a nivel de partón(Hard-Scatter), mientras que otros llegan al nivel de partículas estables individuales. Donde cada evento generado se representa mediante una lista de partículas en el estado final y sus momentos, a menudo con información de partículas producidas en etapas intermedias guardada. El mensaje clave es que las simulaciones de Montecarlo son nuestra mejor estimación de lo que puede suceder en el LHC, pero no podemos considerarlas como una verdad absoluta. Esencialmente, son una herramienta que nos proporciona predicciones teóricas que pueden compararse con datos experimentales.

La creación de este manual tiene como objetivo introducir al lector a uno de estos generadores de eventos, en particular a Pythia8. Este generador es altamente reconocido por su capacidad para simular todas las fases esenciales de la generación de eventos en colisiones de partículas de altas energías. Una de sus características distintivas es la implementación del modelo Lund durante la fase de hadronización, lo que lo convierte en una herramienta confiable y ampliamente utilizada en los estudios de física de partículas.

2. Primer acercamiento

Pythia es un programa diseñado para generar eventos que describen colisiones a altas energías. Este programa incorpora teorías y modelos que abordan diversos aspectos de la física de partículas, tales como distribuciones de partones, interacciones fuertes, hadronización, radiación de partones en estados iniciales y finales, interacciones de múltiples partones, fragmentación y decaimiento. Un componente esencial de Pythia es el modelo de Lund, que simula la formación de partículas hadrónicas a partir de quarks y gluones en colisiones de alta energía.[3]

Instalación

En esta primera parte, detallaremos el proceso de instalación de la última versión de Pythia8 hasta la fecha de publicación de este manual. La instalación se llevará a cabo en un sistema operativo UNIX, específicamente en la distribución de Ubuntu. Sin embargo, los pasos son los mismos para cualquier distribución. También realizaremos pruebas utilizando los ejemplos predeterminados proporcionados durante la instalación.

1. Para la instalación, es recomendable crear un directorio específico y entrar en él para llevar a cabo todos los pasos necesarios. En este directorio, instalaremos todo lo necesario según lo indicado en este manual.

```
$ mkdir pythia
$
$ cd pythia
$
```

2. Accede al sitio web oficial y descarga la última versión de Pythia8 en el directorio que creaste previamente. Puedes encontrar el enlace en la sección **Download and install**. El texto del enlace debería tener el formato Pythia8xxx.tgz, donde xxx representa la última versión de Pythia8. En el momento de redactar esta guía, la última versión disponible era pythia8310. Para descargarlo, utiliza el siguiente comando en la terminal:

```
$ wget https://pythia.org/download/pythia83/pythia8310.tgz
```

3. Así, para descomprimir y desplegar su contenido, utilizamos:

```
$ tar xvfz pythia8310.tgz
```

4. Hasta este punto, hemos descargado Pythia8. Ahora procederemos con su instalación básica. Lo siguiente sería utilizar **make** o **cmake** en la terminal dentro del directorio generado al descomprimir. Sin embargo, antes de hacer esto, debemos verificar qué compilador vamos a utilizar y asegurarnos de tenerlo instalado.

```
$ ./configure
```

5. En caso de que surja algún error, es posible que necesitemos actualizar los archivos y proceder con la instalación de la siguiente paquetería.(algunas opciones generales)

```
$ sudo apt-get update
$ sudo apt-install build-essential
```

6. Finalmente, para completar la instalación, procederemos con la compilación:

```
$ make -j4
```

Al usar el comando de compilación, empleamos la opción -j4 para indicar el número de núcleos utilizados en el proceso de compilación. Se recomienda usar la mitad de los núcleos disponibles en su máquina para optimizar el rendimiento del compilador.

2.1. Configuración estándar

La instalación de Pythia8 incluye diversos ejemplos que ilustran las distintas formas en que se puede utilizar este paquete. Estos ejemplos se encuentran dentro de la carpeta **/pythia8xxx /examples**. Puede encontrar una breve descripción de cada uno de estos archivos en el sitio web oficial. En esta guía, comenzaremos construyendo un ejemplo desde cero para explicar la configuración básica de Pythia.

1. Vamos a iniciar creando un script llamado mymain01.cc. Para ello, desde la terminal, accedemos a la carpeta /pythia8xx/examples y ejecutamos:

```
$ gedit mymain01.cc
$
```

2. Se abrirá la ventana del editor de texto (puede usar cualquier otro, como nano o vim). Lo primero que debemos hacer es declarar la función main, indispensable en C++, ya que siempre es la función llamada cuando el programa se ejecuta. Para hacer esto, escribimos en el editor de texto:

```
#include <iostream>

int main() {

}
```

3. En principio, después de guardar el script, estamos listos para ejecutarlo. Esto se debe a que el Makefile en la carpeta de ejemplos nos permite compilar archivos con la terminación mymainXX. Para lograrlo, primero compilamos el script (esto debe hacerse cada vez que se modifica el script) y luego lo ejecutamos:

```
$ make mymain00
$ ./mymain00
```

Por supuesto, al ejecutarlo, no ocurrirá nada, ya que nuestro script aún no realiza ninguna acción. Ahora avanzaremos con el flujo de un programa básico de Pythia. Para acceder a las funciones y procedimientos de Pythia8, primero debemos importarlos y luego indicar que estamos utilizando el espacio de nombres(namespace) de Pythia:

```
#include <iostream>
# include " Pythia8 / Pythia .h"
using namespace Pythia8 ;
int main (){

}
```

4. A continuación, procederemos a declarar un objeto de tipo Pythia, que funcionará como el generador de eventos y almacenará toda la información sobre las partículas producidas. Le daremos el nombre de pythia a este objeto.

```
#include <iostream>
# include " Pythia8 / Pythia .h"
using namespace Pythia8 ;
int main (){
     Pythia pythia ;
}
```

   A partir de este momento, esta configuración servirá como la base para todas las que trabajaremos.

2.2. Configuración de la simulación

Podemos comenzar con una configuración clásica, donde implementaremos la generación de pares de quarks top con las condiciones del Gran Colisionador de Hadrones (LHC), específicamente en su canal hadrónico, es decir, a través de interacciones fuertes.

Supongamos que se desea generar eventos con las siguientes condiciones:

1. Colisiones de beams de protones con la misma energía.
2. Energía total de centro de masa de los beams de 13 TeV = 13000GeV .
3. Se deben generar procesos del quark Top en interacción fuerte.

Por defecto, Pythia asume que los haces están formados por protones y que ambos tienen la misma energía, por lo que no es necesario indicarlo explícitamente. Sin embargo, para las otras dos condiciones, es necesario agregar líneas con el método **readString()**. Cada configuración en Pythia se realiza utilizando este método, como se muestra a continuación.

```
#include <iostream>
# include "Pythia8/Pythia.h"
using namespace Pythia8 ;

int main (){
     Pythia pythia ;
      pythia.readString("Beams:eCM = 13000.");
      pythia.readString("Top:gg2ttbar = on");
      pythia.init();

}
```

En este código, hemos agregado líneas que utilizan el método pythia.readString(**string**), el cual nos permite modificar la configuración predeterminada de la corrida que estamos simulando. El string que introducimos contiene la nueva configuración que queremos implementar. El primer string establece las condiciones de energía[1] para el LHC.

El segundo string especifica el proceso de generación de pares de quarks top mediante la fusión de gluones, $gg \rightarrow t\hat{t}$, este objeto se puede imaginar como interruptores que encienden o apagan procesos, de manera similar para el resto de procesos.

[1] En Pythia8, el string para configurar la energía del centro de masa se expresa en unidades de GeV; es esencial tener precaución con las unidades de energía.

Se puede encontrar información más detallada sobre las configuraciones utilizadas para los haces y los procesos del Top en la guía en línea de Pythia8. Una vez que estas configuraciones han sido establecidas utilizando el método **pythia.readString(string)**, el siguiente paso es inicializar la simulación mediante el método pythia.init(). Esto prepara la variable Pythia para la generación de eventos con la configuración deseada.

Ahora procedemos a crear un bucle que generará las colisiones de protones, formalmente llamadas eventos. Este bucle es conocido como el **Event Loop**. Primero, crearemos una variable que contendrá el número de eventos, y después de cada evento, se imprimirá el texto .ªnalysis here!". Donde luego, realizaremos nuestro primer análisis.

```
#include <iostream>
# include "Pythia8/Pythia.h"
using namespace Pythia8 ;

int main (){
     Pythia pythia ;
      pythia.readString("Beams:eCM = 13000.");
      pythia.readString("Top:gg2ttbar = on");
      pythia.init();

      int nEvents = 0; //número de eventos
        for ( int iEvent = 0; iEvent < nEvents ; ++ iEvent ) {
        if (! pythia . next ()) continue ;
        cout <<" Analysis here !" <<endl ;
        }
     }
```

Como se puede observar, dentro del Event Loop se ha incluido una condicional que utiliza el método **pythia.next()**. Este método es esencial para generar el próximo evento. Si, por alguna razón, la generación del evento falla[2], **pythia.next()** devolverá false. En este caso, la condición se activa y la instrucción continue se ejecuta, lo que lleva al Event Loop a omitir las líneas siguientes y pasar a la próxima iteración. Colocar esta línea en el script es crucial; de lo contrario, Pythia no producirá ningún evento.

Ahora expandiremos gradualmente el programa básico **mymain01** que se presentó anteriormente, en la dirección de una configuración de análisis más completa. En muchas ocasiones, deseamos combinar varios procesos. Para agregar el proceso $qq \to t\hat{t}$ al ejemplo anterior, simplemente incluimos una segunda llamada a pythia.readString:

```
pythia.readString("Top:qqbar2ttbar = on");
```

Para obtener estadísticas sobre la generación, las secciones transversales estimadas, incluso mensajes de problemas, agrega:

```
pythia.stat();
```

justo antes del final del programa. Durante la ejecución, es posible que recibas mensajes de problema. Estos vienen en tres tipos:

- Una advertencia (Warning)es un problema menor que se corrige automáticamente por el

[2]Existen diversas razones por las cuales podría ocurrir un fallo, como la verificación de la conservación de masa, momento o color.

programa, al menos aproximadamente.

- Un error es un problema más grande, que normalmente aún se corrige automáticamente por el programa, retrocediendo y volviendo a intentarlo.
- Un aborto es un problema tan importante que el evento actual no pudo completarse; en un caso raro, pythia.next() es falso y el evento debe omitirse.

Por lo tanto, el usuario solo necesita estar atento a los abortos. Durante la generación de eventos, un mensaje de problema se imprime solo la primera vez que ocurre (excepto en algunos casos especiales). El pythia.stat() mencionado anteriormente te dirá cuántas veces se encontró cada problema durante toda la ejecución.

2.3. Primer análisis

Una vez construido nuestro script mymain01 podemos partir de este como una base para realizar nuestro análisis, esto con la finalidad de extraer información de propiedades de cada i-ésima partícula producida en el evento, y accederemos mediante la rutina pythia.event[i].property(), donde property[3] debe ser reemplazada por la propiedad que se desee obtener.

```
for( int iEvent = 0; iEvent < nEvents ; ++ iEvent ) {
        if (!pythia.next())continue ;

        for ( int i = 0; i < pythia.event.size(); ++ i ){

                if ( pythia.event[i].isFinal()){

                cout<<" Particula final: "
                    << pythia.event[i].name()
                    <<" , id: "
                    << pythia.event[i].id()
                    <<" , su energia: "
                    << pythia.event[i].e() <<endl;
                                }
                }
        }
```

En este código se muestra cómo imprimir las nombres,id,y energías de las partículas finales de un evento, modificando el for anterior, y al nuevo archivo nombraremos mymain02.

1. La función del segundo bucle for es recorrer todas las partículas producidas en el evento actual mediante el índice ï", el cual varía desde 0 hasta el número de partículas en el evento, determinado por:
2. La variable pythia.event.size() , la cual es igual al número total de partículas presentes más uno (pues incluye el índice 0 de system).
3. Es dentro de este último bucle for donde se realiza el análisis de las propiedades de las partículas. En primer lugar, dado un índice i, la condicional if analiza si la partícula en cuestión es final o no. De activarse la condicional, se imprimen su nombre, id y su energía.

Tal que en la terminal se imprimirá lo siguiente:

```
~$ Particula final: K_L0 , id: 130 , su energia: 13.719
```

[3] Encontraremos un gran listado de propiedades a extraer en el manual en línea.

3. Aspectos prácticos a conocer

Como se habrá notado al ejecutar alguno de los scripts mymainXX, Pythia por defecto imprime información detallada sobre el proceso generado. Esta información es bastante extensa e incluye secciones como

- PYTHIA Process Initialization
- PYTHIA Multiparton Interactions Initialization
- PYTHIA Flag + Mode + Parm + Word + FVec + MVec + PVec + WVec Settings (changes only)
- PYTHIA Particle Data Table (changed only)
- PYTHIA Info Listing
- PYTHIA Event Listing (hard process)
- PYTHIA Event Listing (complete event)

Estaremos interesados en analizar las dos últimas, que se conocen como el Event Record.

La sección PYTHIA Event Listing (hard process) contiene información acerca de los primeros pasos de la simulación en el instante en que los beams interaccionan. La sección PYTHIA Event Listing (complete event) es la que es más utilizada para el análisis físico del evento, pues contiene información acerca de absolutamente todas las partículas producidas.

Compilar con interfaces de paquetes externos

Un aspecto esencial es saber cómo compilar Pythia8 junto con otros paquetes externos. Esto se debe a que el Makefile incluido en la carpeta de ejemplos solo tiene la capacidad de compilar scripts que tengan el formato mymainXX y estén destinados exclusivamente a Pythia. Por lo tanto, detallaré la estrategia a seguir cuando queramos compilar por nuestra cuenta dentro de la carpeta de ejemplos.

1. Tomando como base el siguiente comando para compilar exclusivamente pythia.

   ```
   ~$g++ mymain01.cc -o mymain01 -I../include -O2 -std=c++11 -pedantic
   -W -Wall -Wshadow -fPIC -pthread  -L../lib -Wl, -rpath,../lib
   -lpythia8 -ldl
   ```

   Este lo encontramos al momento de compilar los mymain0X que creamos previamente.

2. Añadiendo para compilar junto con el paquete hepmc2.

   ```
   ~$g++ mymain03.cc -o mymain03 -I../include -O2 -std=c++11 -pedantic
   -W -Wall -Wshadow -fPIC -pthread  -L../lib -Wl,-rpath,../lib
   -lpythia8 -ldl -I/miDirectorio/hepmc2.06.09/build/include
   -L/miDirectorio/hepmc2.06.09/build/lib -Wl,-rpath,/miDirectorio
   /hepmc2.06.09/build/lib -lHepMC \-DHEPMC2
   ```

   Esta estrategia se sigue una vez compilado el main42 que usa este paquete externo, que posteriormente utilizaremos para generar archivos de salida que serán de utilidad para la simulación de la respuesta de detector rápida.

3. Además, para compilar junto con el paquete que nos brindará un entorno de trabajo para las herramientas de análisis.

```
~$g++ mymain04.cc -o mymain04 -I../include -O2 -std=c++11 -pedantic
-W -Wall -Wshadow -fPIC -pthread  -L../lib -Wl, -rpath,../lib
-lpythia8 -ldl `root-config --cflags --glibs`
```

   Esta forma de compilación resulta especialmente útil cuando se pretende utilizar herramientas del entorno de trabajo Root, cuya implementación se detallará más adelante.

3.1. Identidad ID

Es importante señalar que por defecto, Pythia asume que los haces de partículas son protones. No obstante, es posible modificar esta configuración utilizando la siguiente sintaxis: **Beams:idA = 2212** para el haz izquierdo y **Beams:idB = 2212** para el haz derecho, donde el número 2212 corresponde al ID del protón según el Grupo de Datos de Partículas (PDG). También se puede ajustar estos valores para representar otras partículas.

Cuadro 1: Códigos PDG

Partícula	**Nombre**	**Código PDG**	**Partícula**	**Nombre**	**Código PDG**
Quark up	u	2	Muon neutrino	ν_μ	14
Quark down	d	1	Electron neutrino	ν_e	12
Quark charm	c	4	Photon	γ	22
Quark strange	s	3	Proton	p	2212
Quark top	t	6	Neutron	n	2112
Quark bottom	b	5	Higgs boson	H	25
Electron	e	11	Plomo (Pb)	-	1000822080
Positron	e^+	-11	Muon	μ	13

Estos códigos se ubican dentro del esquema de numeración de partículas de Monte Carlo, que es esencialmente una convención adoptada en la física de altas energías para asignar a cada tipo de partícula un código. Se debe denotar que el ID 90 está reservado para el sistema.

3.2. Archivos de salida

Es esencial comprender el formato (**Event Format**) en el cual se registran y almacenan los eventos simulados por este generador Monte Carlo. Esta información será especialmente útil para identificar las interacciones de partículas en experimentos de física de partículas. En esta sección, se proporcionarán aspectos relevantes de los dos formatos de salida más comunes utilizados en simuladores de Monte Carlo (MC) en el ámbito de la física de altas energías.

1.HepMC (Hep Monte Carlo)

- Es un formato estándar para almacenar información sobre eventos de física de partículas generados por simuladores Monte Carlo, a nivel de **Parton Shower**, es decir, contiene información que modela la radiación de partones (quarks y gluones) en eventos de dispersión profunda.
- Los archivos HepMC suelen tener la extensión .hepmc o .hepmc2.
- Los eventos en formato HepMC se someten a una simulación rápida o completa, con la finalidad de obtener información del detector.[4]

2.LHE (Les Houches Event format):

- Es un formato estándar acordado en talleres realizados en Les Houches, Francia. Este formato se utiliza para guardar eventos a nivel de **partones**, es decir, antes de la formación de hadrones.
- Los archivos LHE suelen tener la extensión .lhe.
- Posee una extension LHEF (Les Houches Event File), que cuenta con una estructura más organizada y legible para los datos de eventos generadoscon extesion .lhef. [1]

El establecimiento de un formato de salida consistente entre todos los generadores facilita la intercambio de información, datos entre diferentes simuladores. En muchos casos, los experimentos optan por utilizar sus propios formatos internos para almacenar datos de eventos, siendo estos formatos frecuentemente diseñados en base a estándares reconocidos como HepMC, LHE y LHEF. Esta elección asegura coherencia y compatibilidad con los generadores, facilitando la integración y el análisis de datos.

Interfase con Hepmc

El formato estándar HepMC es comúnmente utilizado en paquetes de análisis de datos, permitiéndonos comparar los resultados obtenidos en nuestras simulaciones con información real del experimento. Este formato es compatible, por ejemplo, con el entorno del paquete Delphes. Por lo tanto, implementaremos el mencionado paquete, proporcionando instrucciones para su instalación y configuración adecuada con Pythia. Asumiremos que trabajaremos desde la carpeta de ejemplos.

Recomendamos trabajar con versiones de HepMC2, una versión estable con la que podemos hacer interfase con las ultimas versiones de Pythia8, proporcionaré el sitio web oficial, donde podemos dar un vistazo mas detallado.

Actualmente, nuestro interés es descargar la última actualización. Para iniciar el proceso de instalación, necesitaremos descargar y compilar, de manera similar a como lo hicimos con Pythia8. Una vez realizado esto, seguiremos con los siguientes pasos en la terminal.

- Nos movemos al directorio principal de pythia82xx (cd .. si estas en ejemplos)
- Necesitamos configurar pythia con hepmc2

  ```
  ~$./configure --with-hepmc2=path
  ```

  donde path es dirección donde se realizo la instalación local, por ejemplo:

  ```
  ~$./configure --with-hepmc2=mi_directorio/hepmc2.06.09/build
  ```

- Despues de la configuración es necesario compilar de nuevo.

  ```
  ~$make
  ~$make install
  ```

Regresemos al subdirectorio de ejemplos. Ahora, también podemos utilizar los ejemplos **main41.cc** y **main42.cc** para probar el uso de la interfase con HepMC.

Sin embargo, decidí ampliar el procedimiento anterior para generar la producción de pares de quarks top. Al implementar esta extensión al archivo de salida, necesitaremos agregar los siguientes elementos:

```
#include "Pythia8Plugins/HepMC2.h"
```

esto antes del main(), luego dentro del main

```
     // Interface for conversion from Pythia8::Event to HepMC event.
HepMC::Pythia8ToHepMC ToHepMC;

// Specify file where HepMC events will be stored.
HepMC::IO_GenEvent ascii_io("filename.hepmc", std::ios::out);
```

y dentro del bucle for de manera similar

```
 //Inicio del loop
for (int iEvent = 0; iEvent < numero_evt; ++iEvent){

     if (! pythia . next ()) continue ;
    // HEP files
    HepMC::GenEvent* hepmcevt = new HepMC::GenEvent();
    ToHepMC.fill_next_event( pythia, hepmcevt );

    // Write the HepMC event to file. Done with it.
    ascii_io << hepmcevt;
    delete hepmcevt;

}
```

Nótese que el procedimiento anterior[4] se basa en la suposición de que estarás ejecutando tus programas principales desde el subdirectorio de ejemplos.

Como se mencionó en la sección anterior acerca de los formatos de salida, el formato estándar de registro de eventos HepMC es especialmente útil para comparaciones con análisis de datos experimentales. Para aquellos más ambiciosos, se describe aquí cómo configurar la interfaz de Pythia, asumiendo que ya has instalado HepMC. Un procedimiento similar es necesario para la interfaz con otras bibliotecas externas, por lo que los puntos mencionados anteriormente pueden tener una utilidad más general.

[4] Es importante destacar que el código completo se encuentra en el repositorio del manual.

4. Herramientas de análisis

En el ámbito del análisis en física de altas energías, el uso de ROOT se destaca como una herramienta fundamental y poderosa. ROOT, diseñado para el lenguaje de programación C++, se convierte en el entorno de trabajo preferido para investigadores y científicos. Aunque C++ puede considerarse complejo, la interfaz proporcionada por ROOT facilita significativamente su utilización, permitiendo a los investigadores centrarse en el análisis de datos en lugar de lidiar con la complejidad del código. Esta herramienta de código abierto se ha convertido en un estándar en la comunidad científica, ofreciendo capacidades avanzadas para el tratamiento de datos, tanto en simulaciones como en la adquisición de datos experimentales. Con sus características y funcionalidades especializadas, ROOT contribuye de manera significativa al proceso de análisis y visualización de datos en física de partículas.[5]

4.1. Ambiente de trabajo Root

Este se encuentra disponible en Linux, Mac y Windows. Para instalarlo, usualmente se presentan las siguientes opciones:

1. Uso de un gestor de paquetes, como conda, snap,etc.
2. Descargar una version pre-compilada

En todos los casos, asegúrate de utilizar siempre la versión más reciente de ROOT para obtener las últimas correcciones de errores, características y un soporte rápido.

Ambas son buenas opciones de instalación pero recomendaremos hacer una instalación a partir de un archivo fuente de ROOT, esto es especialmente útil si queremos tener sobre las opciones de instalación, en caso de querer usar la primera sugiero utilizar conda.

Los pasos para instalar son sencillos:

1. Instalar los prerrequisitos de acuerdo al sistema operativo.
2. Descargar del archivo fuente correspondiente al sistema operativo.
3. Compilar con las opciones que deseamos.
4. Ejecutar nuestro ambiente de ROOT.

Vamos a ilustrar como es este procedimiento en la terminal de cualquier distribución[5] de LINUX, después de ello, realizaremos los paso para conectar el ambiente de ROOT con el generador MC Pythia8.

1. El primer paso es actualizar los paquetes necesarios para que el ambiente funcione correctamente, o en dado caso de no tenerlos actualizarlos.

   ```
   ~$sudo apt-get install dpkg-dev cmake g++ gcc binutils libx11-dev
    libxpm-dev libxft-dev libxext-dev python3 libssl-dev
    python-is-python3
   ```

   Recomiendo además instalar los paquetes opcionales.

[5]Este procedimiento se esta considerando distribuciones Ubuntu

```
~$sudo apt-get install gfortran libpcre3-dev xlibmesa-glu-dev
libglew-dev libftgl-dev libmysqlclient-dev libfftw3-dev
libcfitsio-dev graphviz-dev libavahi-compat-libdnssd-dev
libldap2-dev python3-dev python-dev-is-python3 libxml2-dev
libkrb5-dev libgsl0-dev qtwebengine5-dev
```

2. El segundo paso es tener una carpeta donde vamos a instalar root, y una segunda donde vamos a compilar (usualmente compilamos en una directorio donde instalamos)

```
~$ mkdir root_install
~$ cd root_install
~$ mkdir root_build
```

Se acostumbra llamar build al directorio donde compilaremos.

3. Tercer paso es descargar un archivo fuente del sitio web de de root releases, y depositarlo en la carpeta de instalación.

```
~$ wget https://root.cern/download/root_v6,30.XX.source.tar.gz
```

Por ejemplo, descargando una versión al momento de creación de este manual.

4. Ahora procedemos a compilar en un directorio nuevo o en “build”

```
~$ cmake ../
~$ cmake ../ -Droofit=ON
~$ make
```

La primera línea se utiliza para compilar ROOT sin algún paquete externo. Por otro lado, en la segunda línea, si queremos compilar el paquete de herramientas para análisis Roofit, podemos usar esta opción. Si deseamos incluir más paquetes, está la opción -Dall=ON, que permite compilar todos los paquetes externos disponibles.

5. Por último, es necesario saber cómo activar nuestro ambiente, y puedes encontrar un atajo en el subdirectorio "bin"dentro de la carpeta donde realizamos la compilación.

```
~$  source bin/thisroot.sh
```

Un consejo que te será de gran utilidad es crear un archivo de comandos para ejecutarlo desde el directorio principal (home).

En este punto, con ROOT ya instalado, procederemos a realizar la configuración necesaria para su integración con Pythia8.

1. Similarmente a cuando conectamos la paquete Hepmc.

```
~$./configure --with-root=$ROOTSYS
```

2. Además es necesario compilar de nuevo,

```
~$ make -j8
~$ make install
```

Esto es especialmente útil porque desde un código de Pythia podemos obtener información de un histograma a nivel de propiedades de partículas extraídas con distintos métodos de ROOT.

4.2. Histogramas

Dentro de todo el arsenal de herramientas de analisis que contaremos en el ambiente de ROOT, la principal es la creación de histogramas de nuestras variables de interes.Por ejemplo:

- Pseudorapides
- Momento Transversal
- Ángulo azimutal

Estos a nivel partículas, sin considerar las implicaciones de la respuesta de detector, pensando en un análisis mas realista.

Podemos partir del mymain02, donde hicimos nuestro primer análisis de partículas generadas en un proceso del quark top; ahora actualizar este llamándolo ahora nuestro mymain04, y considerando los siguientes pasos.

1. Primero añadimos los headers de root a usar

   ```
   #include "TH1D.h"
   #include "TFile.h"
   #include "Pythia8/Pythia.h"

   using namespace Pythia8;
   ```

   El primero es para los histogramas que contengan doubles, y el segundo para crear archivos con la extensión de root.

2. Segundo dentro de nuestro main añadimos las lineas

   ```
   int main() {

    TFile *File = new TFile("output.root", "RECREATE");

    TH1D *hEta = new TH1D("hEta", ";Eta;Frecuencia", 50, -5.0, 5.0);
    TH1D *hPhi = new TH1D("hPhi", ";Phi;Frecuencia", 50, -3.2, 3.2);
    TH1D *hPt = new TH1D("hPt", ";pT;Frecuencia", 50, 0.0, 100.0);
   ```

   Estos intuitivamente conociendo los paquetes usados, podemos identificar el método para crear archivos root, y nuestros histogramas.

3. Tercero, debemos modificar el bucle for donde expresábamos el nombre, id y energía de las partículas finales, de manera que podamos almacenar nuestras variables de interés.

```
for (int i = 1; i < pythia.event.size(); ++i) {
     if (pythia.event[i].isFinal()) {
         // Llenar histogramas
         hEta->Fill(pythia.event[i].eta());
         hPhi->Fill(pythia.event[i].phi());
         hPt->Fill(pythia.event[i].pT());
     }
 }
```

En este caso ahora tomaremos como variables de interés, la pseudo rapidez, momento transversal, ángulo azimutal.

4. Por ultimo una vez cerrado el bucle for de eventos, deberemos dar la orden de escribir estos histogramas en el archivo con extensión root creado que llamamos output.

```
hEta->Write();
hPhi->Write();
hPt->Write();
File->Close();
```

Es importante cerrar el archivo al momento de terminar.

Como podemos ver las declaraciones de estos objetos de ROOT no son complicadas y no es necesario tener un profundo conocimiento del lenguaje de programación C++, es bastante intuitivo aplicar estas herramientas de analisis.

Para visualizar el contenido de los archivos ROOT, necesitaremos entrar al entorno de ROOT y ejecutar el comando root en la terminal.

```
~$  root
~$  new TBrowser
```

Procedemos a ejecutar el comando new TBrowser, el cual proporciona una interfaz de usuario para explorar y visualizar la estructura de los objetos almacenados en un archivo ROOT.

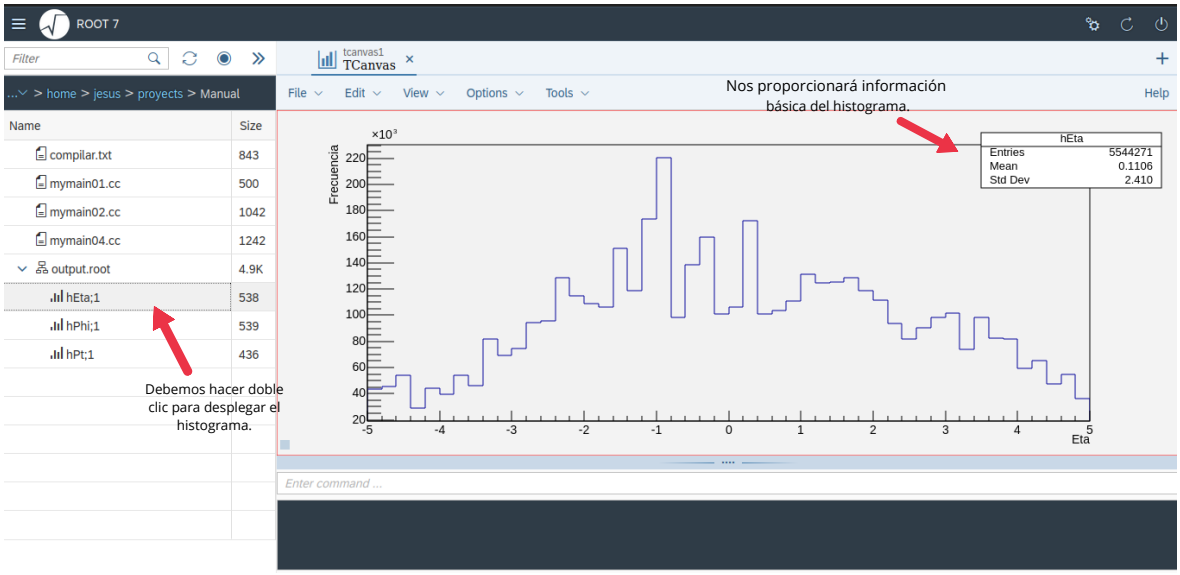

Figura 2: Visualización predeterminada de ultimas versiones.

5. Respuesta del detector

Es esencial realizar la simulación de la respuesta de detectores en experimentos de altas energías, para permitir la comparación entre la data adquirida y la información obtenida mediante simulaciones. La generación de eventos únicamente con Pythia nos proporciona información a nivel de partículas, pero para realizar comparaciones con los experimentos reales, es necesario reproducir la respuesta de los detectores correspondientes a los estudios en cuestión. Al implementar nuestros eventos generados en un simulador de detector, obtenemos información a nivel del detector, lo que nos habilita para llevar a cabo análisis detallados y estudios más precisos. Este paso es crucial para entender y validar los resultados experimentales.

5.1. Simulación rápida

Para obtener una respuesta rápida del detector, utilizaremos Delphes, un marco de trabajo en C++ que lleva a cabo una simulación eficiente y versátil de la respuesta de un detector. Esta simulación incluye un sistema de seguimiento integrado en un campo magnético, calorímetros electromagnéticos y hadronicos, un sistema de muones. Delphes se integra con formatos de archivo estándar, como Les Houches Event File o HepMC, y genera información clave, como leptones generados, fotones, energía perdida , colección de jets, etc. Estos observables son fundamentales para análisis específicos en experimentos de física de altas energías.[6]

Para comenzar con Delphes es necesario descargarlo.

```
~$ wget http://cp3.irmp.ucl.ac.be/downloads/Delphes-3.5.0.tar.gz
~$ tar -zxf Delphes-3.5.0.tar.gz
```

Posteriormente compilarlo

```
~$ cd Delphes-3.5.0
~$ make
```

Es bastante sencillo iniciar en este software, y para ejecutarlo usaremos.

```
~$  ./DelphesHepMC3 config_file output_file [input_file(s)]
```

Donde **config_file** es el archivo de configuración correspondiente al experimento a simular, este se encuentra en una extension Tcl, y podemos encontrarlos en el directorio de cards. El segundo **output_file** es archivo de salida que debemos indicar con extensión de root. Por ultimo es el archivo de entrada **input_file** el cual se debe encontrar en extensión de hepmc, para esto implementamos esta paquetería previamente en nuestro generador de eventos.

Por ejemplo, generaremos un archivo de salida en formato ROOT utilizando un archivo de entrada en formato HepMC. En este escenario, estaremos simulando la respuesta del experimento Compact Muon Solenoid, también conocido como CMS, de la siguiente manera.

```
~$  ./DelphesHepMC3 cards/delphes_card_CMS.tcl output.root input.hepmc
```

Podemos ver el contenido de esta salida usando TBrowser de root.

```
~$  root -l output.root
~$  TBrowser
```

Como se muestra en la figura siguiente, es posible obtener de manera rápida información clave simulada del experimento CMS. Esto proporciona una representación realista de cómo se verían los eventos de física de partículas en este detector. Es importante destacar que, aunque esta simulación rápida no alcanza un nivel de detalle profundo, omitiendo, por ejemplo, interacciones de partículas con la materia, presenta la ventaja de no requerir una gran cantidad de recursos computacionales y ofrece una configuración sencilla.

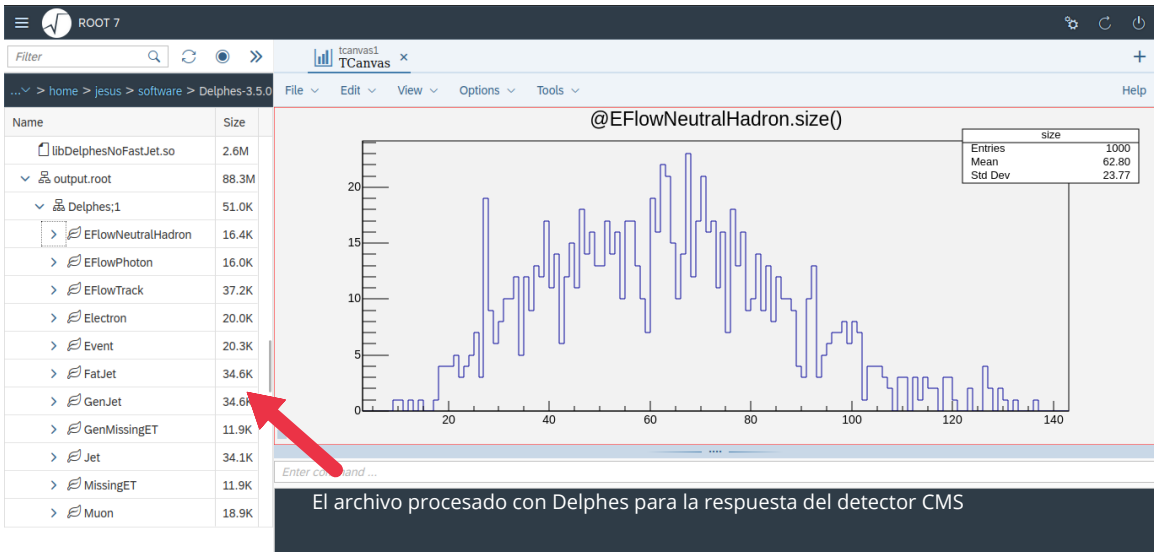

Figura 3: Visualización de información por la respuesta de detector.

5.2. Simulación completa

En la simulación rápida de Delphes, se cuenta con un repertorio de complejos experimentos que permite simular su respuesta. Sin embargo, es importante destacar que su debilidad radica en la incapacidad de simular completamente las interacciones de partículas a través de la materia. Para tareas que requieran una capacidad de simulación completa de la respuesta del detector, entra en juego el software Geant4. Este último dispone de un conjunto de herramientas diseñadas para simular la radiación a través de la materia y puede utilizarse en diversas aplicaciones, como la física de altas energías, la física médica, las aplicaciones espaciales y la física de radiaciones. Geant4 es especialmente útil para generar cantidades arbitrarias que se puedan comparar con datos experimentales o para prever el rendimiento de un detector, el código fuente de Geant4 está disponible bajo una licencia de código abierto[7]. Este incluye aplicaciones de ejemplo, al igual que Pythia con su carpeta de ejemplos, que demuestran configuraciones más simples y aplicaciones completas, mientras que otras sirven como punto de partida para construir su propia aplicación personalizada.

En el caso de la simulación de grandes experimentos como CMS, generar la respuesta del detector a nivel detallado es una tarea muy ambiciosa debido a los altos recursos computacionales que se requieren. Por esta razón, existe un grupo de trabajo dentro de la colaboración que se encarga de generar simulaciones y digitalizaciones. En este proceso, las partículas recién generadas por algún generador de eventos, como Pythia en nuestro caso, se someten a una simulación detallada basada en Geant4 del detector CMS, y se modela la respuesta de la electrónica del detector. Después de este paso, se obtiene una estructura con diferentes niveles de procesamiento de datos internos al experimento, siendo AOD y MINIAOD dos de ellos. Por esta razón, en este manual se presenta de manera divulgativa la utilización de Geant4 para simular experimentos como CMS, pero es importante destacar que también es útil para experimentos de menor escala en la realización de estudios locales.

Referencias

[1] Kar, Deepak. *Experimental Particle Physics*,IOP.(2019),doi:10.1088/2053-2563/ab1be6.

[2] A.Schaelicke, T.Gleisberg, S.Hoeche, S.Schumann, J.Winter, F.Krauss and G.Soff, *Event generator for particle production in high-energy collisions*,Prog. Part. Nucl. Phys. **53** (2004), 329-338.doi:10.1016/j.ppnp.2004.02.031 [arXiv:hep-ph/0311270[[hep-ph]].

[3] C.Bierlich, S.Chakraborty, N.Desai, L.Gellersen, I.Helenius, P.Ilten, L.Lönnblad, S.Mrenna, S.Prestel and C.T.Preuss.*A comprehensive guide to the physics and usage of PYTHIA 8.3*,SciPost Phys. Codeb. **2022**, doi:10.21468/SciPostPhysCodeb.8 [arXiv:2203.11601 [hep-ph]].

[4] A.Buckley, P.Ilten, D.Konstantinov, L. Lönnblad, J. Monk, W. Pokorski, T. Przedzinski and A. Verbytskyi, *The HepMC3 event record library for Monte Carlo event generators*,Comput. Phys. Commun. **260** (2021),doi:10.1016/j.cpc.2020.107310 [arXiv:1912.08005 [hep-ph]].

[5] R. Brun and F. Rademakers, *ROOT: An object oriented data analysis framework*," Nucl. Instrum. Meth. A **389** (1997), 81-86 doi:10.1016/S0168-9002(97)00048-X

[6] J. de Favereau *et al.* [DELPHES 3], *DELPHES 3, A modular framework for fast simulation of a generic collider experiment*," JHEP **02** (2014), 057 doi:10.1007/JHEP02(2014)057 [arXiv:1307.6346 [hep-ex]].

[7] S. Agostinelli *et al.* [GEANT4], *GEANT4–a simulation toolkit*," Nucl. Instrum. Meth. A **506** (2003), 250-303 doi:10.1016/S0168-9002(03)01368-8