Table of Contents

Taller 3

Convenciones utilizadas en esta guía

- Texto en azul es para ser reemplazado por algún un valor indicado
- Argumentos opcionales o pasos opcionales van en rosado.
- `Lineas de codigo de algun archivo mas en este fondo beige`
- Usaré como editor **gedit**. Si ustedes son familiares con algun otro, pueden usarlo.

Material

El material para este taller, se puede descargar del siguiente link:

- ejercicios-Parte3.tgz: Ejercicios Parte TC3

En esta carpeta, se encuentra el codigo necesario para la hacer los ejercicios. Tambien existe una carpeta **solucion** que contiene el codigo completo.

Ejercicio 1 Creación de un simple Histograma

El primer ejercicio consistirá en la creación de un Histograma con informacion de la cinematica de algunas particulas. Editemos el script que se trabajo al final del Taller 2 y que simula eventos de Supersimetria:

- Quitar el comentario en la linea siguiente del codigo y escribir el numero de bins 50 y el rango del histograma XMIN=0 y XMAX=100:

```
TH1F * h1 = new TH1F("hpt", "Muons Pt", );
```

- Esta linea define un Histograma 1D (clase TH1F).

- Bajar hasta donde está el ciclo de generación y buscar la linea con el comentario "// Particle Loop" y agregrar el siguiente codigo:

```
while( iev <maxEvts ) {

    pythia8->GenerateEvent();
    if (iev  <1) pythia8->EventListing();
    pythia8->ImportParticles(particles,"All");

    Int_t np = particles->GetEntriesFast();

    // Particle loop
```

Estudiemos que esta pasando aqui:

- El primer ciclo, es un ciclo sobre la generacion de eventos . Por ello es que la condicion es hacer este ciclo hasta que alcance el maximo numero de eventos que hemos especificado desde el principio del script. La generacion esta instruida por el comando **pythia8->GenerateEvent();**

- El segundo ciclo, aquel que hemos introducido, hace **por cada evento** un ciclo sobre el numero maximo de particulas que Pythia ha generado. El maximo numero de particulas se obtiene con el metodo **Int_t np = particles->GetEntriesFast();**

- Los muones tienen por código de identificación 13 y -13 (para el anti-mu). Volvemos a este comentario mas adelante

- Cada partícula está representada por un objeto de tipo TParticle y queremos graficar el momento transversal de cirtas particulas
    - ♦ Mirando en la documentación TParticle, como podríamos obtenerlo?

- Por el momento, el codigo introducido no hace aun lo que queremos. Recordemos, queremos hacer 3 cosas:

1. Entre la particulas, identificar aquellas que son muones -es decir con codigo 13 y -13
2. Un vez identificadas, tenemos calcular el momento transversal
3. Llenar el histograma que creamos para hacer

- Por lo tanto, debemos copiar el siguiente codigo - guiarse por los comentarios:

```
//1. Introducir una codicion para separar los muones


//Queremos graficar el momento Transverso de los muones Pt == Sqrt(px*px + py*py)
//2. Obtener aqui las componentes Px y Py del momento de las particulas



//3. Llenar el histograma h1 con el momento transversar
```

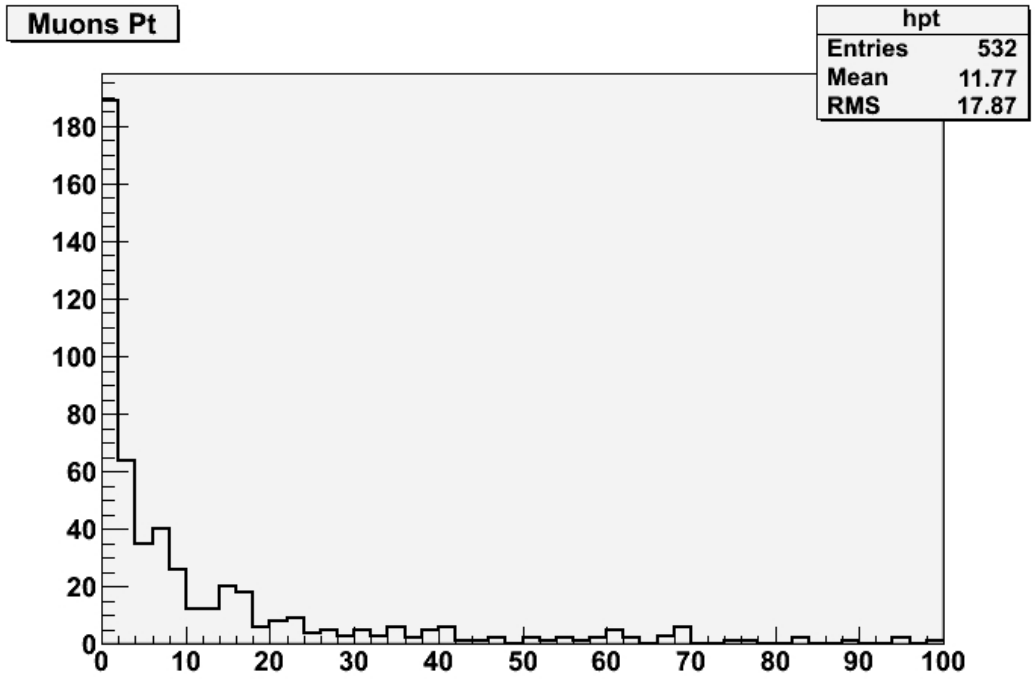
* Bien! Esto nos sirve para llenar el histograma. Ahora, si queremos visualizar el histograma debemos introducir al final de todo los ciclos lo siguiente:

```
// Ahora podemos dibujar el histograma y que lo muestre en pantalla
```

- Guarde las modificaciones y cierre. Ejecutar ahora el script desde la terminal:

La opción -l ("menos L minuscula") evita que salga la imagen de saludo de ROOT, lo que nos ahorra algunos milisegundos.

Show results... Hide results...



Ejercicio 2 PythiaAnalsis Una clase para hacer el análisis sobre datos simulados

Ahora lo que queremos es introducir una técnica para el análisis de datos simulados.

Hemos trabajadoe en el script **pythia8_susy.C**, el cual genera los eventos y los guarda en un archivo de formato de ROOT (manejado por la clase TFile). En el script, la creacion del archivo ocurre en la siguiente linea:

```
//Definir archivo de salida
TFile * outfile = new TFile("eventos_pythia8_SUSY.root","RECREATE");
```

Dentro de este archivo existe lo que se conoce como un TTree un árbol en el que cada rama corresponde a las variables que poseen las partículas y las hojas son los valores que toman por evento. Este árbol se define en la siguiente linea del código:

```
TTree*tree= new TTree("tree","Arbol con particulas segun Pythia8");
tree->Branch("particles",&particles);
```

¿Cómo hacer un analisis de los datos contenidos en este árbol? Lo haremos utilizando una clase llamada **PythiaAnalysis**. Esta clase es automáticamente generada por ROOT basada en la **estructura que tiene árbol** (ver nota Suplementaria No 1). La clase **PythiaAnalysis** se declara y define en dos archivos:

- **PythiaAnalysis.h**: declaración de la clase, sus variables y metodos
- **PythiaAnalysis.C**: definición del metodo **Loop**. Este metodo realiza un ciclo/loop sobre los eventos contenidos en el árbol

Miremos el archivo **PythiaAnalysis.h**

- El este archivo se declara la clase **PythiaAnalysis**. Contiene toda una serie de arreglos, los cuales definen el contenido por evento de cada particula simulada.

- Tambien contiene unos metodos (en el contexto de C/C++). Entre los mas importantes, se encuentran un **constructor** y otros metodos que se encargan de asociar y llenar cada uno de de los arreglos con los datos contenidos en el arbol.
- Para nosotros, el metodo que vamos a modificar es el llamado **Loop**, el cual hace un ciclo sobre los eventos generados. Este metodo se define en el archivo **PythiaAnalisis.C**. Editemos entonces este archivo:

Intentemos hacer de nuevo un histograma que nos cuente el numero de muones por evento. Nos concentraremos en implementar el método **Loop( )** que se encarga de hacer la iteración sobre los eventos almacenados.

- Para ello debemos adicionar un segundo ciclo, que nos itere sobre la particula i-esima, parecido a lo que hicimos en el ejercicio anterior:
- En este ciclo, preguntamos por el PDG_ID de la particula y comparamos
- Adicionamos una condición y un contador para muones
- Finalizado este loop podemos entonces graficar la distribución del número de muones por evento

Nota: la lectura del TTree nos hace perder los objetos partículas como tales (TParticle). Su información se encuentra ahora en forma de arreglos o **arrays** a la C/C++ (ver PythiaAnalysis.h).

- Uno tiene dos opciones:

1. leer y manipular los arreglos
2. leer los arreglos y crear de ellos objetos TParticle

Para este ejercicio, haremos una sencilla modificacion del metodo **Loop()** y utilizaremos los arreglos.

```
   //Adicionar un histograma para contar el numero de muones


  // Empieza loop sobre eventos
  for (Long64_t jentry=0; jentry


  }//cierra loop sobre eventos

  //Graficar el histograma del numero de muones por evento
```

Como correr este codigo? Un forma sencilla es la de cargar esta clase al inicio de la sesion en ROOT:

```
root [0]
(Int_t)0
```

Si todo sale bien, no deberia haber ningun mensaje de error. Si aparece alguno, hay que leer el mensaje. Posiblemente ha quedado algo mal escrito durante la modificacion realizada al metodo **Loop()**.

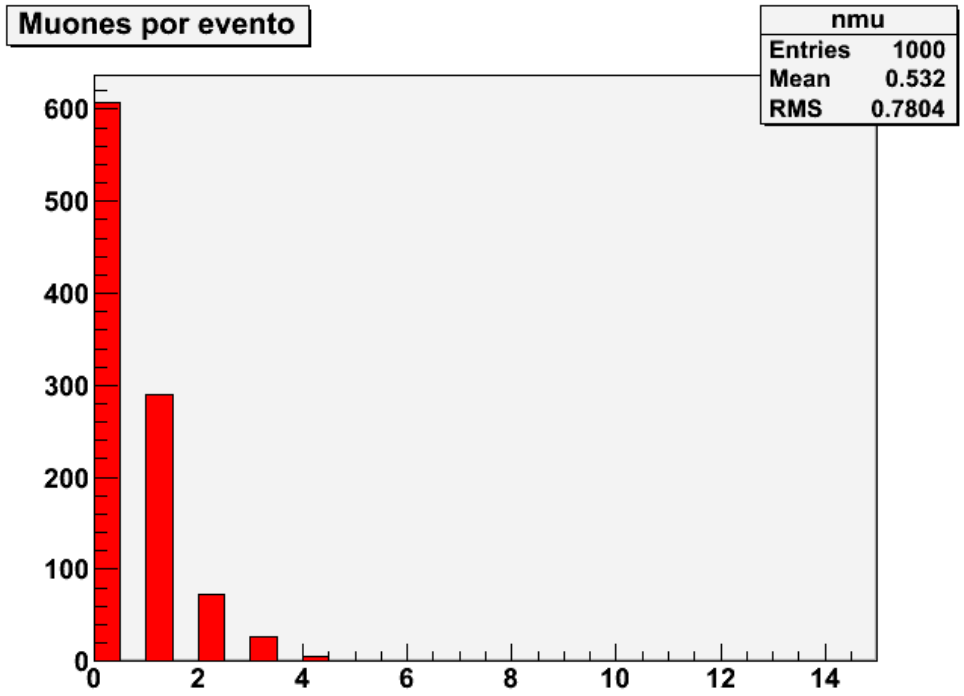
Ahora para correr este codigo, necesitamos dos lineas. La primera sera crear un apuntador a un objeto de la clase **PythiaAnalysis** y luego a este apuntador, hacer que corra el metodo **Loop()**. El constructor de un objeto **PythiaAnalysis** toma como argumento el nombre del archivo (va entre comillas) en donde se encuentran los datos simulados.

```
root [1]
```

```
root [2]
```

- En este ejemplo, **susy** es un apuntador a un objeto de tipo **PythiaAnalysis**. Una vez creado el objeto al que apunta, podemos llamar el metodo **Loop** y hacer el analisis sobre los eventos simulador.

Show results... Hide results...



Para hacer esto de una forma mas automatica, ver nota Suplementaria No 2.

Ejercicio 3 JetPythiaAnalysis Una clase para hacer el análisis sobre datos simulados incluyendo Jets

Ahora queremos introducir una técnica para el análisis de datos simulados usando los algoritmos de reconstrucción ded Jets que nos da el paquete FastJet (incluido en el Live-CD que ustedes tienen). Esto lo hemos implementado en una clase llamada **JetPythiaAnalysis** la cual posee las mismas caracteristicas de **PythiaAnalysis**, pero además incluye la posibilidad de reconstruir Jets.

- Abrir los archivos JetPythiaAnalysis.h y JetPythiaAnalysis.C : explorar
  - Al igual que en el caso anterior, en JetPythiaAnalysis.h se declara la clase (con sus variables y metodos)
  - JetPythiaAnalysis.C contiene la definicion del metodo **Loop()**
- Vamos a editar el archivo **JetPythiaAnalysis.C**

La idea es la de construir Jets a partir de partículas estables o cuyo tiempo de vida es suficiente para que alcancen a llegar a los distintos instrumentos de deteccion que se colocan en un detector de particulas. Por ejemplo, dentro de este tipo de particulas tendriamos muones, electrones, piones, kaones. En este ejercicio haremos la reconstruccion de Jets segun un algorithmo en especial, el Kt-jet y graficaremos la distribucion de momento transversal. Para ello los pasos a seguir son:

1. Definir el algoritmo para la reconstruccion de Jets
2. Definir el histograma
3. Hacer un ciclo sobre particulas por evento y seleccionar aquellas que sean **estables** y que **detectables**. Por **detectables** nos referimos a que particulas que podrian ser vistas en el detector y no particulas

que como neutrinos escaparian a la deteccion.
4. Llenar un contenedor con estas particulas
5. Correr el algorithmo de reconstruccion de Jets
6. Hacer un ciclo sobre los Jets resultantes, extraer su momento transversal y pasar el valor al histograma
7. Una vez terminado el ciclo sobre eventos, graficar

Editar el archivo JetPythiaAnalysis.C siguiendo los comentarios que alli se encuentran:

e introducir las siguientes lineas de codigo:

- Definir el algoritmo para reconstruir jets

```
//Definir el tipo de Algoritmo para reconstruir Jets y pasarle el parametro requerido
```

- Definir el histograma que vamos a llenar

```
//Definir un histograma para el momento transversal de los Jets
```

- En el ciclo de eventos, insertar un ciclo sobre particulas y colocar alli las condiciones necesarias para seleccionar las particulas estables y detectables y llenar un contenedor.

```
int np = 0;
 int max_part = particles_; //Maximo numero de particulas

// Definit un contenedor de Particulas para pasarle a FastJet


//Implementar aqui el analisis
 while ( np <max_part )  {

  //llenar aqui con las particulas estables: estas particulas serian aquellas que no tienen h

  //Este cierra la condicion

  ++np; // pasamos a la siguiente particula

}
```

- Ejecutar el algoritmo de reconstruccion de Jets y hacer un ciclo sobre el resultado. Extraer el momento transversal de cada Jet y llenar el histograma con este valor.

```
// Corremos ahora FastJet: hacer sobre las particulas la identificacion de Jets


// Objeto Cluster contiene los jets: podemos guardarlos en un contenedor de Jets


// Ahora podemos hacer un ciclo sobre los jets reconstruidos y extraer el momento transversal
// - llenar histograma



 //Limpiar el contenedor de jets y particulas en preparacion para el proximo evento
```

- Por fuera del ciclo sobre los eventos, graficar el histograma:

```
// Dibujar la distribucion de momento transversal
```

- Listo, esas serian todas las modificaciones a hacer al codigo. Para correr vamos a necesitar algunas cosas mas. Editar el archivo siguiente y seguir comentario:

```
//Para el ejercicio 3: needCompilation = true
bool needCompilation =
```

- Que ocurre? Resulta que para poder usar **FastJet** dentro de ROOT, necesitamos una opcion que cargue la libreria de FastJet y que **compile** el codigo que hemos realizado, a diferencia de los pasos anteriores. Esto lo podemos instruir para que se haga automaticamente al inicio de ROOT desde este archivo **rootlogon.C** (ver nota Suplementaria N2).

- Si todo ha quedado bien escrito, ROOT compila exitosamente las dos clases PythiaAnalysis y JetPythiaAnalysis.

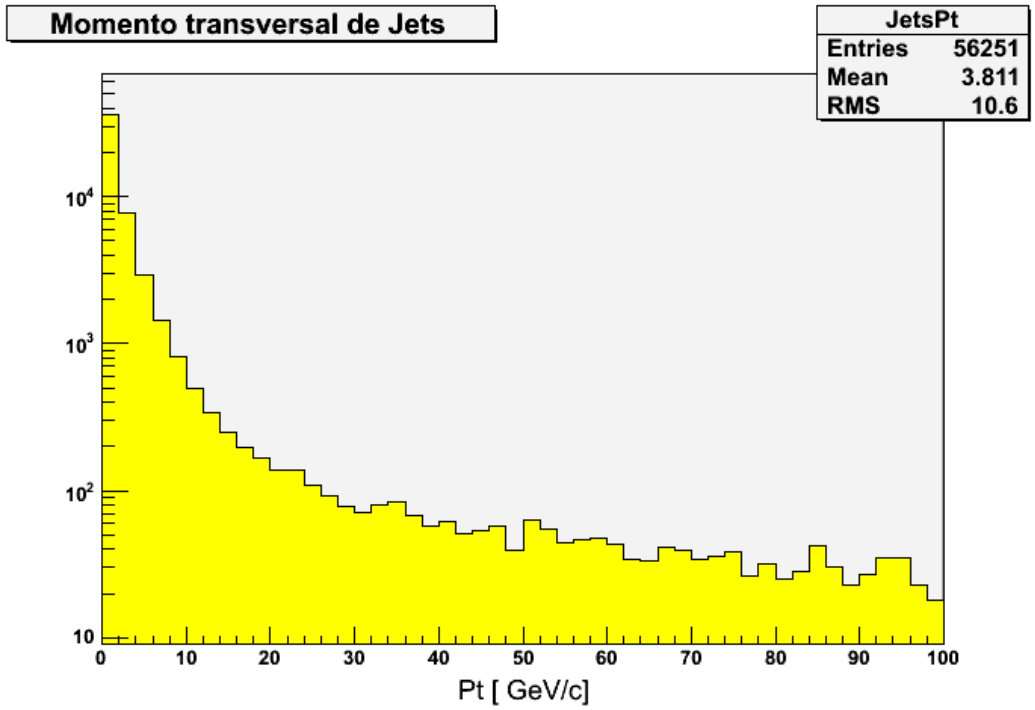
```
Info in
```

- Para correr JetPythiaAnalysis, tenemos que hacer dos pasos muy similares a los del ejercicio N2:

```
Info in
#--------------------------------------------------------------------------
#                      FastJet release 2.4.2
#            Written by M. Cacciari, G.P. Salam and G. Soyez
#                        http://www.fastjet.fr
#
# Longitudinally invariant Kt, anti-Kt, and inclusive Cambridge/Aachen
# clustering using fast geometric algorithms, with area measures and optional
# external jet-finder plugins.
# Please cite Phys. Lett. B641 (2006) [hep-ph/0512210] if you use this code.
#
# This package uses T.Chan's closest pair algorithm, Proc.13th ACM-SIAM
# Symp. Discr. Alg, p.472 (2002), S.Fortune's Voronoi algorithm and code .
#--------------------------------------------------------------------------
```

- El mensaje en pantalla es un mensaje de bienvenida de FastJet. El resultado de nuestro analisis sera algo como esto:

Show results... Hide results...



Eso sería todo! Buena suerte.

Nota Suplementaria N1

Estos son los pasos para crear una clase para hacer el analisis de un TTree:

```
root [0]
root [1]
root [2]
root [3]
Info in
```

Nota Suplementaria N2

Cada vez que uno inicia ROOT, el programa busca en la carpeta en donde se encuentra parado un script llamado **rootlogon.C**. Si no existe, ROOT inicia con los valores que trae por por defecto. Si existe, entonces ejecuta este script, dando la oportunidad de que el usuario introduzca cambios en las configuracion inicial. Nosotros podemos usar este script para que carge automaticamente las clases que necesitamos. Este seria un ejemplo del contenido de dicho script:

```
// Contenido del archivo rootlogon.C
void rootlogon()
{
gROOT->LoadMacro("PythiaAnalysis.C");
}
```

De esta forma nos evitamos este paso. Para ejecutar, el **Loop** sobre unos datos podemos guardar las dos lineas en un **macro**. Un **macro** almacena los comandos que dariamos a ROOT en una sesion interactiva, los cuales deben ir entre dos corchetes **{** y **}**. Al igual que en C/C++ cada comando debe ir separado por un **;**. Por ejemplo, uno podria almacenar todo en el siguiente macro:

```
// Contenido del archivo runAnalysis.C
{

PythiaAnalysis * susy = new PythiaAnalysis("eventos_pythia8_SUSY.root");
```

```
susy->Loop();

}
```

Este **macro** se puede ejecutar de dos formas:

- Desde linea de comando:

- Desde ROOT con el comando **.x** (punto x):

```
root [0]
```

-- AndresOsorio - 15-May-2012

This topic: Main > EPUniandes2011TC3
Topic revision: r11 - 2013-08-22 - AndresOsorio