Lect3

December 17, 2019

1 Macro e file .mac .sh , Ntuple e Trees

In questo notebook parleremo brevemente di come si possono creare e usare le macro in ROOT e dei tipi di file correlati al framework.

In particolare vedremo come usando i file .mac e .sh si possono "meccanizzare" ovvero usare ripetutamente le macro create precedentemente, vedremo alcuni esempi di macro (che eseguiremo sul terminale non sul notebook) e come è possibile eseguirle ripetutamente da un unico file.

Parleremo poi di due classi ROOT più avanzate: - TTree - TNtuple

Solitamente derivanti dall'acquisizione dati.

!!! importante

Nel file seguente alcune istruzioni (import ROOT e %%cpp) sono usate solo e soltanto per far funzionare il codice sul notebook, non vanno usate nel framework!

1.1 Macro e file .mac e .sh

Il framework di ROOT permette di caricare ed eseguire macro, tuttavia cosa è una macro?

Le librerie e funzionalità di ROOT possono essere sostanzialmente usate in due modi: si possono integrare in un programma compilabile in C++ (complicato da integrare e necessita di alcuni flag in compilazione) di cui non parleremo oppure si possono scrivere dei codici in formalismo C++ che vengono poi eseguiti direttamente all'interno del framework, questa seconda opzione è detta macro.

Facciamo un esempio: il file TH1toTxt.cc è un file c++ in cui è presente solamente un omonima funzione, esso non è eseguibile e compilabile con gcc (il classico compilatore C++) ma può essere eseguito comodamente all'interno del framework, analizziamolo meglio

[1]:
```
import ROOT
```

```
Welcome to JupyROOT 6.14/06
```

[2]:
```
#per prima cosa vediamo cosa si trova nel file
#!!!!questo non è root sto eseguendo nella shell
!cat TH1toTxt.cc
```

```
void TH1toTxt (TH1F *h, string outfile) {
        ofstream out (outfile);
```

```
        int n=h->GetNbinsX();
        for (int i=0; i<n; i++)
out<<h->GetBinCenter(i)<<"\t"<<h->GetBinContent(i)<<endl;
        cout<<"converted TH1 "<<h->GetName()<<endl;
}
```

Sostanzialmente questo codice chiede un istogramma e il nome di un file di output in cui creare le coppie di punti corrispondenti ai suoi bin (così da poterne fare poi un TGraph).

Il codice può essere comodamente eseguito all'interno del framework in due modi: 1. con l'instruzione .x si esegue direttamente la macro dando i paramentri di input a patto che il nome del file e quello della funzione principale (quella che viene eseguita, simile al main in cpp) coincidano 2. con l'istruzione .L si caricano tutte le funzioni presenti nel file e queste diventano poi eseguibili liberamente all'interno del framework

Nel notebook non è possibile eseguire questi comandi (non sono cp nativi) quindi faremo un esempio direttamente da terminale, in ogni caso il codice che bisogna usare sarà:

- root -l 100mSimK12_trigger.root -> apro il file root
- .x TH1toTxt.cc (hAll, "hAllout.txt") ->eseguo la macro su hAll

oppure nel secondo caso - root -l 100mSimK12_trigger.root -> apro il file root - .L TH1toTxt.cc ->carico la macro - TH1toTxt(hAll, "hAllout.txt") ->eseguo la macro su hAll

Il secondo approccio risulta utile nel caso in un unico file vi siano molte macro che si vogliono eseguire indipendentemente.

In caso si voglia usare la stessa macro ripetutamente è possibile scrivere un file che contiene un'esecuzione "ripetuta" della stessa, questo si può fare con i file .mac se si vuole agire all'interno del framework o con i file .sh se si vuole agire da terminale.

Supponiamo per esempio di voler eseguire ripetutamente la macro di prima sui TH1F hSim, hAll, hRec ed hSel allora è possibile "meccanizzare" la cosa in due modi come vedremo nei due file TH1conv.mac e Txtconv.sh

[3]:
```
!cat TH1conv.mac
```

```
.x TH1toTxt.cc (hSim, "hSimout.txt")
.x TH1toTxt.cc (hAll, "hAllout.txt")
.x TH1toTxt.cc (hSel, "hSelout.txt")
.x TH1toTxt.cc (hRec, "hRecout.txt")
```

In questo file vi è semplicemente l'istruzione di prima ripetuta 4 volte questo file va eseguito sempre con .x all'interno del framework di root quindi si può eseguire anche all'interno di un file già aperto.

In alternativa se all'interno della macro vi sono le istruzioni necessarie per ricevere autonomamente i dati (ovvero se per eseguirla non è necessario essere già all'interno di uno specifico TFile allora è possibile eseguire la macro direttamente nella shell del terminale e la sua "meccanizzazione" può essere eseguita anche con un file .sh nel seguente modo: creiamo una macro che disegna e salva i file txt creati in precedenza:

[4]:
```
!cat TxttoTGraph.cc
```

```
void TxttoTGraph (string name) {
        TGraph * gr= new TGraph((name+".txt").c_str());
        TCanvas *c= new TCanvas("c", "c", 600, 600);
        c->cd();
        gr->Draw();
        c->Draw();
        c->Print((name+".png").c_str());
        cout<<"converted file "<<name<<endl;
}

//to execute in batch
//root -l -b 'TxttoTGraph.cc ("hAllout"); exit (0)'
```

Il file semplicemente crea un TGraph e poi salva il canvas in png, piccola nota è il fatto che il metodo print chiede un array di char quindi è necessario convertire con .c_str().

La macro in questo caso può essere eseguita dentro al framework come già visto semplicemente chiamando - .x TxttoTGraph.cc ("hAllout")

Oppure eseguita nella shell con l'istruzione - root -l -b 'TxttoTGraph.cc ("hAllout"); exit (0)' Dove exit zero serve per tornare poi al terminale e -b serve per eseguire in background (ovvero non mostrerà i canvas)

Per meccanizzare questa operazione ai quattro file creati prima basta scrivere il codice di esecuzione in un file .mac da eseguire nel framework (quindi con quattro copie della prima istruzione) oppure in modo da eseguirlo all'esterno in un file sh eseguibile come un normale file bash

[5]:
```
!cat Txtconv.sh
```

```
root -l -b 'TxttoTGraph.cc ("hAllout"); exit (0)'
root -l -b 'TxttoTGraph.cc ("hSimout"); exit (0)'
root -l -b 'TxttoTGraph.cc ("hRecout"); exit (0)'
root -l -b 'TxttoTGraph.cc ("hSelout"); exit (0)'
```

Questo file semplicemente esegue 4 volte l'struzione ed esce ogni volta in modo che sia nuovamente chiamabile da shell, per eseguirlo è sufficiente digitare - source Txtconv.sh

La cosa comoda è che volendo in questo secondo modo si possono aggiungere comandi nativi della shell, per esempio si può chiedere alla funzione di aprire i file .png creati, come in questo secondo file (analogo al primo ma con un'istruzione in più):

[6]:
```
!cat Txtconv2.sh
```

```
root -l -b 'TxttoTGraph.cc ("hAllout"); exit (0)'
root -l -b 'TxttoTGraph.cc ("hSimout"); exit (0)'
root -l -b 'TxttoTGraph.cc ("hRecout"); exit (0)'
root -l -b 'TxttoTGraph.cc ("hSelout"); exit (0)'

eog h*.png &
```

1.2 TNtuple e TTree

Le TNtuple sono per definizione "A simple TTree restricted to a list of float variables only."

In generale non vi è molta diffrenza tra le due classi, esse sostanzialmente permettono di immagazzinare una serie di dati mantenendoli nell'ordine di aquisizione, la differenza è che in un'Ntupla i dati salvati sono tutti quanti float mentre un albero permette di salvare tipi di dati più complessi (anche classi).

La funzionalità maggiore è la possibilità di plottare i dati in coppia in modo tale da cercare correlazioni e di mantenere in un unico oggetto tutti i dati su cui si vorrà poi fare analisi statistica (supponendo essi derivino da un'aquisizione comune).

In generale non ci addentreremo molto nel come creare e modificare Ntuple e TTre poichè questa cosa viene in generale fatti dal software di acquisizione ma piuttosto su come usare i dati contenuti.

Proviamo come esempio a creare un Ntupla e un TTre in cui caricare i dati del file LD30kAngle.txt, senza scendere nei dettagli dei costruttori vedimao direttamente la procedura "standard" per creare l'ntupla e il TTree, in questo caso le due strutture saranno sostanzialmente equivalenti essendo tutti i dati di tipo float però in caso avessimo input più complessi sarebbe stato necessario usare solamente i TTree oppure semplicare i dati.

[7]:
```
%%cpp
//per prima cosa ci servono i file di input
ifstream in_ntu ("LD30kAngle.txt");
//creiamo una Ntupla
//uso un puntatore (si fa anche in statico volendo)
//per creare una ntupla servono nome titolo e lista delle variabili
TNtuple *ntu=new TNtuple("ntu", "mass distance and error", "m:d:e");

//creo le variabili con cui fillarla
float m, d, e;
```

[8]:
```
%%cpp
while (in_ntu>>m>>d>>e){ //finche ricevo dati
    ntu->Fill(m,d,e); //uso fill per caricarla
}
```

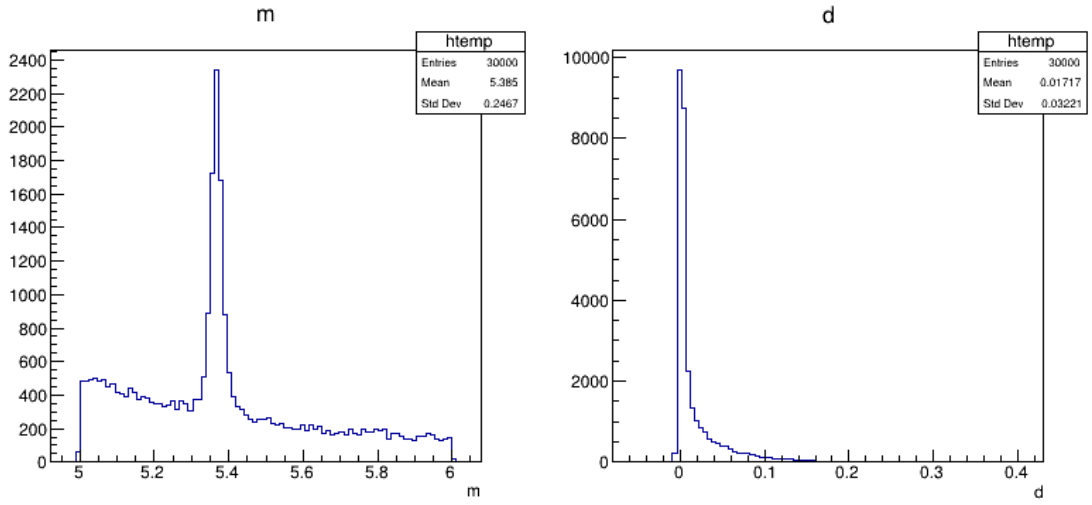
Per disegnare a schermo un'ntupla si è sufficiente usare Draw() specificando la variabile che si vuole plottare, esistono poi altre opzioni che vedremo ma intando proviamo a disegnare le prime due colonne (massa e distanza).

[9]:
```
%%cpp
//faccio un canvas
TCanvas *cntu= new TCanvas ("cntu", "cntu", 800, 400);
cntu->Divide(2,1);

cntu->cd(1);
//disegno la colonna della variabile m della ntupla
ntu->Draw("m");
```

```
cntu->cd(2);
//disegno la colonna d della ntupla
ntu->Draw("d");

cntu->Draw();
```



Quando si invoca Draw su un colonna della ntupla essa disegna automaticamente la colonna come unn TH1, tuttavia spesso il binning e il range lasciano molto a desiderare, vedremo come migliorare la cosa più avanti infatti.

Ora proviamo a fare una cosa analoga usando un TTre:

[10]:
```
%%cpp
//ora creaiamo un TTre
//per prima cosa bisogna dichiararlo
TTree *tree = new TTree ("tree", "mass and distance Tree");
//poi bisogna creare le variabili con cui lo si fillera (per referenza)
float mt, dt, et; //devo cambiare i nomi perche m,d,e sono gia definite
//ora creo input per albero
ifstream in_tree ("LD30kAngle.txt");

//per caricare i dati nel TTree bisogna creare le branch, vi sono molti modi␣
 ↪per farlo ma per i dati semplici
//e sufficiente fare cosi
//creo tre divisioni dell albero, chiamate branch,
//ogni branch e collegata per referenza a una delle variabili da caricare
//il terzo parametro specifica il formato con cui si carica la variabile /
 ↪F->float
//ogni branch sara quindi collegata per referenza a una variabile
```

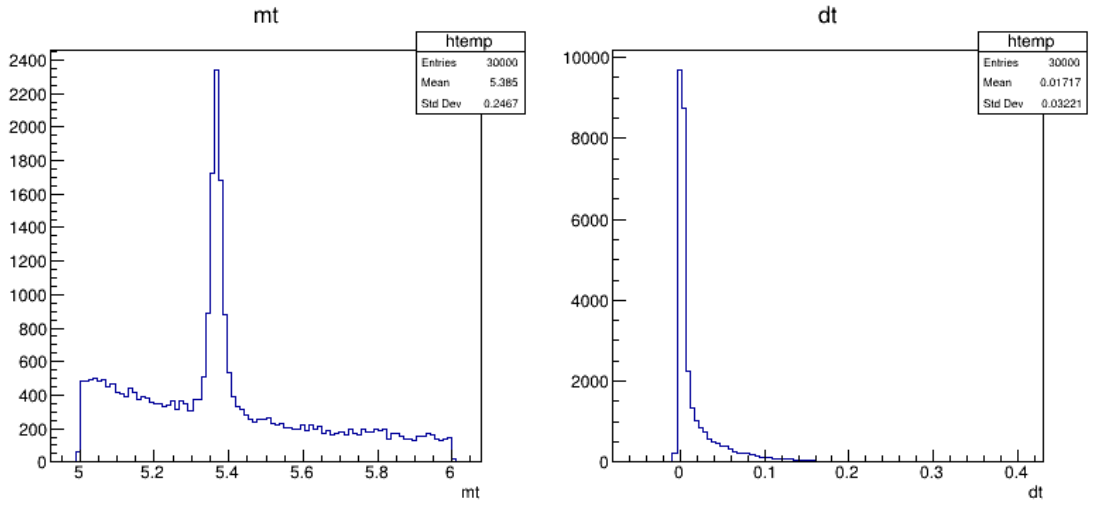
```
TBranch *m_branch = tree->Branch("mt", &mt, "mt/F");
TBranch *d_branch = tree->Branch("dt", &dt, "dt/F");
TBranch *e_branch = tree->Branch("et", &et, "et/F");
```

Ora che ho creato l'abero posso caricare i valori, per farlo a differenza dell'ntupla il metodo fill va chiamato senza parametro, questo perchè essendo definito per referenza alle variabili il metodo fill fa un loop su tutti i branch definiti e salva il valore della corrispondente variabile

[11]:
```
%%cpp
while (in_tree>>mt>>dt>>et) tree->Fill();
```

Proviamo ora a disegnare come abbimao fatto prima:

[12]:
```
%%cpp
TCanvas *ctree= new TCanvas ("ctree", "ctree", 800, 400);
ctree->Divide(2,1);
ctree->cd(1);
tree->Draw("mt");
ctree->cd(2);
tree->Draw("dt");
ctree->Draw();
```



1.3 Lavorare su TTree e TNtuple

Generalmente non si ha necessità di creare un albero e/o ntupla ma piuttosto di analizzare i dati presenti in uno di essi. Nel caso in cui si sia in possesso di un file contentte un'Ntupla o TTree è possibile vederne il contenuto (le branch) con il metodo print(), nei seguenti esempi useremo l'albero presente nel file dataD13Ntuple.root, per prima cosa spostiamoci nel file:

Osservazione Quello che faremo in seguito sarà eseguito su un albero ma essendo le Ntuple una classe che eredita da TTree tutto quanto dovrebbe essere ugualmente riproducibile su una Ntupla.

[13]:
```
%%cpp
TFile *file=new TFile("dataD13Ntuple.root", "READ"); //apro il file in lettura
file->cd();
```

Nel file è presente un TTree chiamato data, vediamo cosa contiene usando print()

[14]:
```
%%cpp
data->Print();
```

```
******************************************************************************
*Tree    :data      : data                                                   *
*Entries :    21014 : Total =         4650526 bytes  File  Size =    3636422 *
*        :          : Tree compression factor =   1.28                       *
******************************************************************************
*Br    0 :BsMass    : BsMass/D                                               *
*Entries :    21014 : Total  Size=     169047 bytes  File Size  =      92800 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.82     *
*...*
*Br    1 :JPsiMass  : JPsiMass/D                                             *
*Entries :    21014 : Total  Size=     169067 bytes  File Size  =      81846 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   2.06     *
*...*
*Br    2 :PhiMass   : PhiMass/D                                              *
*Entries :    21014 : Total  Size=     169057 bytes  File Size  =      82484 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   2.04     *
*...*
*Br    3 :distTrack : distTrack/D                                            *
*Entries :    21014 : Total  Size=     169077 bytes  File Size  =     161140 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.05     *
*...*
*Br    4 :distAngle : distAngle/D                                            *
*Entries :    21014 : Total  Size=     169077 bytes  File Size  =     160111 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.05     *
*...*
*Br    5 :distAllNoBs : distAllNoBs/D                                        *
*Entries :    21014 : Total  Size=     169097 bytes  File Size  =     159245 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.06     *
*...*
*Br    6 :distAll   : distAll/D                                              *
*Entries :    21014 : Total  Size=     169057 bytes  File Size  =     161043 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.05     *
*...*
*Br    7 :distRcNoBs : distRcNoBs/D                                          *
*Entries :    21014 : Total  Size=     169087 bytes  File Size  =     158660 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.06     *
```

```
*…*
*Br    8 :distRc    : distRc/D                                               *
*Entries :    21014 : Total  Size=     169047 bytes  File Size  =     161063 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.05     *
*…*
*Br    9 :eTrack    : eTrack/D                                               *
*Entries :    21014 : Total  Size=     169047 bytes  File Size  =     157967 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.07     *
*…*
*Br   10 :eAngle    : eAngle/D                                               *
*Entries :    21014 : Total  Size=     169047 bytes  File Size  =     158161 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.07     *
*…*
*Br   11 :eAllNoBs  : eAllNoBs/D                                             *
*Entries :    21014 : Total  Size=     169067 bytes  File Size  =     156511 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.08     *
*…*
*Br   12 :eAll      : eAll/D                                                 *
*Entries :    21014 : Total  Size=     169027 bytes  File Size  =     157980 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.07     *
*…*
*Br   13 :eRcNoBs   : eRcNoBs/D                                              *
*Entries :    21014 : Total  Size=     169057 bytes  File Size  =     155862 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.08     *
*…*
*Br   14 :eRc       : eRc/D                                                  *
*Entries :    21014 : Total  Size=     169017 bytes  File Size  =     157969 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.07     *
*…*
*Br   15 :BsPt      : BsPt/D                                                 *
*Entries :    21014 : Total  Size=     169027 bytes  File Size  =     158119 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.07     *
*…*
*Br   16 :JPsiPt    : JPsiPt/D                                               *
*Entries :    21014 : Total  Size=     169047 bytes  File Size  =     140369 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.20     *
*…*
*Br   17 :PhiPt     : PhiPt/D                                                *
*Entries :    21014 : Total  Size=     169037 bytes  File Size  =     158945 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.06     *
*…*
*Br   18 :K1Pt      : K1Pt/D                                                 *
*Entries :    21014 : Total  Size=     169027 bytes  File Size  =     142652 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.18     *
*…*
*Br   19 :K2Pt      : K2Pt/D                                                 *
*Entries :    21014 : Total  Size=     169027 bytes  File Size  =     142196 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.19     *
```

```
*…*
*Br   20 :mu1Pt     : mu1Pt/D                                              *
*Entries :    21014 : Total  Size=     169037 bytes  File Size  =     140046 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.20     *
*…*
*Br   21 :mu2Pt     : mu2Pt/D                                              *
*Entries :    21014 : Total  Size=     169037 bytes  File Size  =     139867 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.21     *
*…*
*Br   22 :mu1Eta    : mu1Eta/D                                             *
*Entries :    21014 : Total  Size=     169047 bytes  File Size  =      99660 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.69     *
*…*
*Br   23 :mu2Eta    : mu2Eta /D                                            *
*Entries :    21014 : Total  Size=     169050 bytes  File Size  =      99453 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.69     *
*…*
*Br   24 :prob      : prob/D                                               *
*Entries :    21014 : Total  Size=     169027 bytes  File Size  =     158863 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression=   1.06     *
*…*
*Br   25 :runNum    : runNum/I                                             *
*Entries :    21014 : Total  Size=      84764 bytes  File Size  =        750 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression= 112.37     *
*…*
*Br   26 :evNum     : evNum/I                                              *
*Entries :    21014 : Total  Size=      84757 bytes  File Size  =      68556 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   1.23     *
*…*
*Br   27 :trig1     : trig1/I                                              *
*Entries :    21014 : Total  Size=      84757 bytes  File Size  =       7536 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=  11.18     *
*…*
*Br   28 :trig2     : trig2/I                                              *
*Entries :    21014 : Total  Size=      84757 bytes  File Size  =       5089 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=  16.56     *
*…*
*Br   29 :trig12    : trig12/I                                             *
*Entries :    21014 : Total  Size=      84764 bytes  File Size  =       8352 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=  10.09     *
*…*
```

Come potete vedere vi sono 30 branches (br0->29) contenenti i risultati di 21000 eventi circa, la maggior parte sono double (/D) e alcune contengono interi (/I). Le altre informazioni non sono di nostro interesse (riguardano come viene gestita la memoria dei dati).

Ora che sappiamo cosa contiene il nostro albero proviamo a lavorarci un po', in seguito tralascieremo la parte estetica per contentrarci piuttosto su come usare i dati.

Una delle cose che è possibile fare e da una branch di un TTree ottenere un TH1, questo può essere sostanzialmente fatto in due modi, un modo più “formale” e uno più “furbo”.

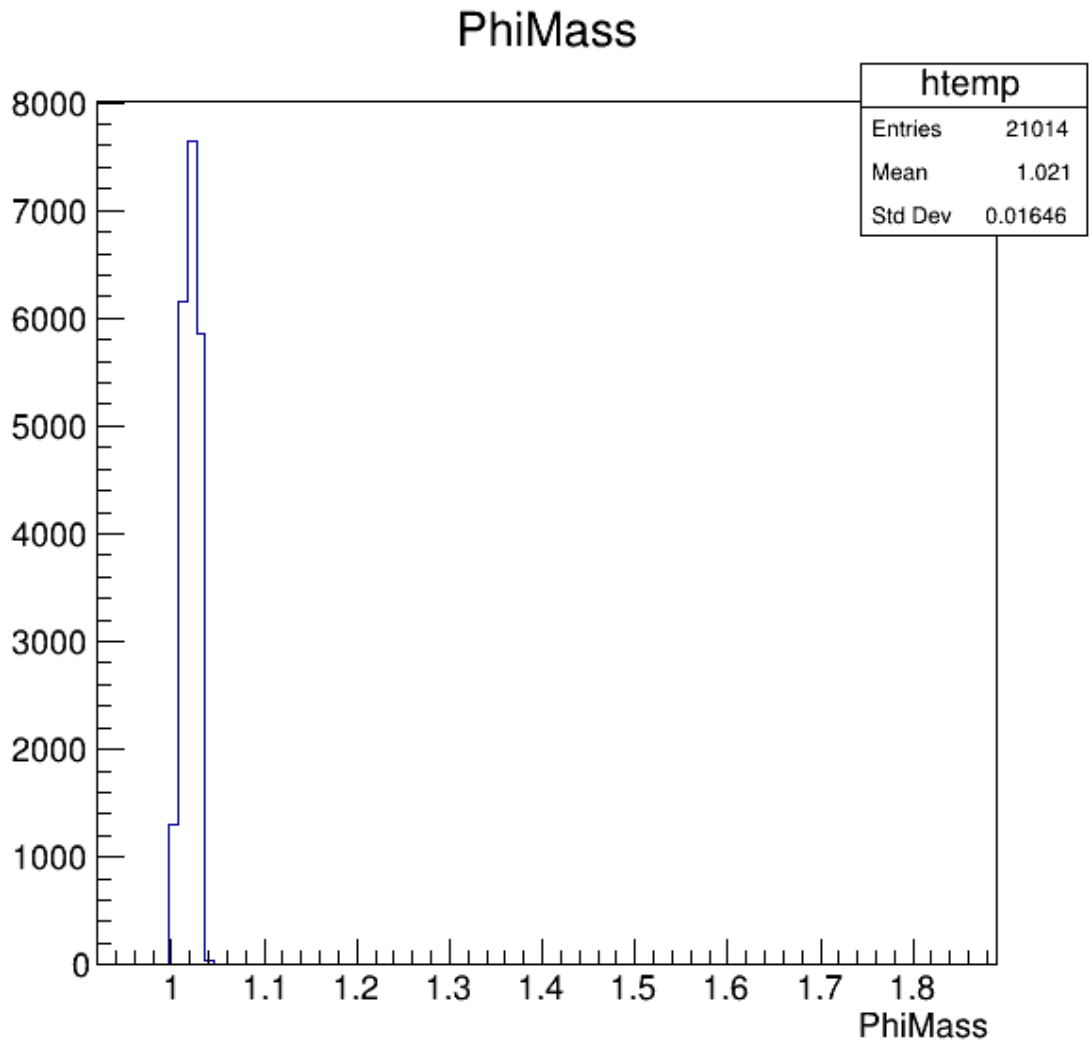
Supponiamo di voler ottenere un TH1F dalla branch PhiMass, è possibile farlo direttamente usando il metodo GetHistogram:

ATTENZIONE

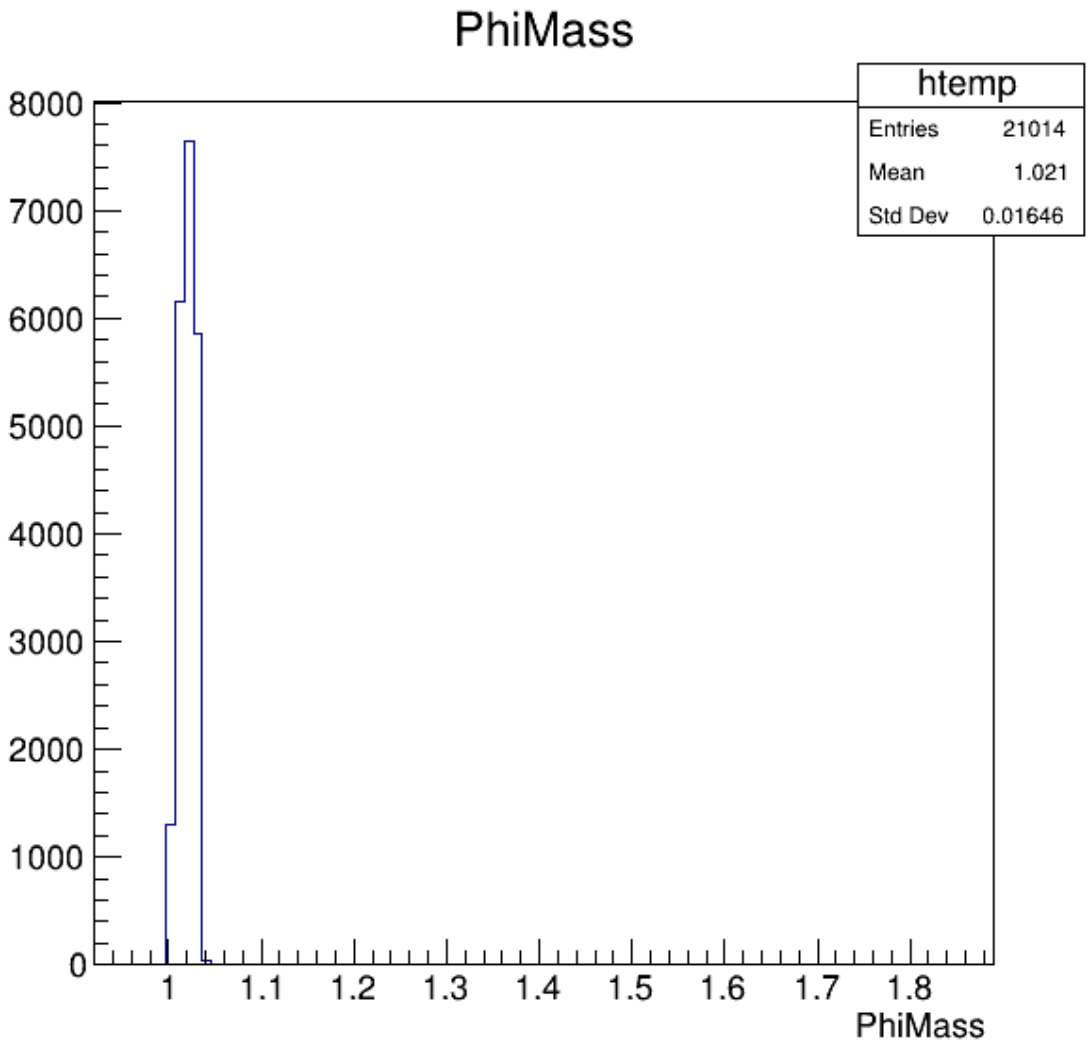
Il metodo ritorna in TH1 NON TH1F quindi è OBBLIGATORIO eseguire un cast (come abbiamo fatto per il metodo clone dei TH1F)

[15]:
```
%%cpp
TCanvas *ct=new TCanvas("ct", "ct", 600,600);
TCanvas *ct2=new TCanvas("ct2", "ct2", 600,600);
```

[16]:
```
%%cpp
ct->cd();
data->Draw("PhiMass");
TH1F *mPhi = (TH1F*)data->GetHistogram();
ct->Draw();
```



[17]:
```
%%cpp
ct2->cd();
mPhi->Draw();
ct2->Draw();
```



Sostanzialmente il metodo salva in un istogramma la branch che al momento è mostrata a schermo, la definizione è infatti:

TH1 *GetHistogram() { return GetPlayer()->GetHistogram(); }

Il problema di questo approccio tuttavia è che il binning e la rappresentazione sono spesso fatti automaticamente quindi non nel modo corretto per farvi poi un'analisi.

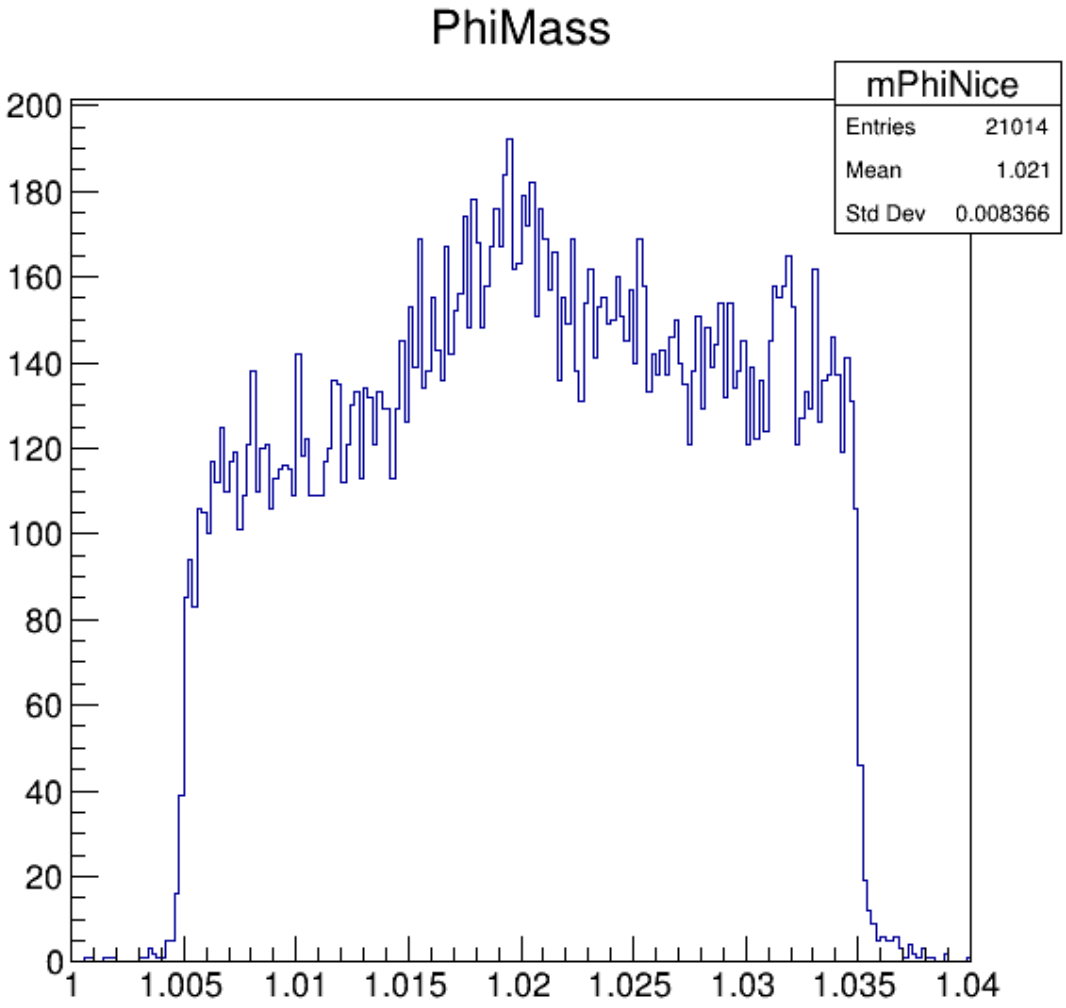
L'alternativa è usare il metodo Draw ridirigendo l'output (con l'operatore >>) a un istogramma con binning e range scelto da noi:

[18]:
```
%%cpp

//per prima cosa definiamo un TH1F con le caratteristiche che vogliamo noi
TH1F *mPhiNice = new TH1F ("mPhiNice", "PhiMass", 200, 1, 1.04);
//ora usiamo Draw ma ridirigiamo l output sul th1f che abbiamo creato
```

```
data->Draw("PhiMass>>mPhiNice");

//disegniamo il risultato
TCanvas * ct3= new TCanvas ("ct3", "ct3", 600, 600);
ct3->cd();
mPhiNice->Draw();
ct3->Draw();
```

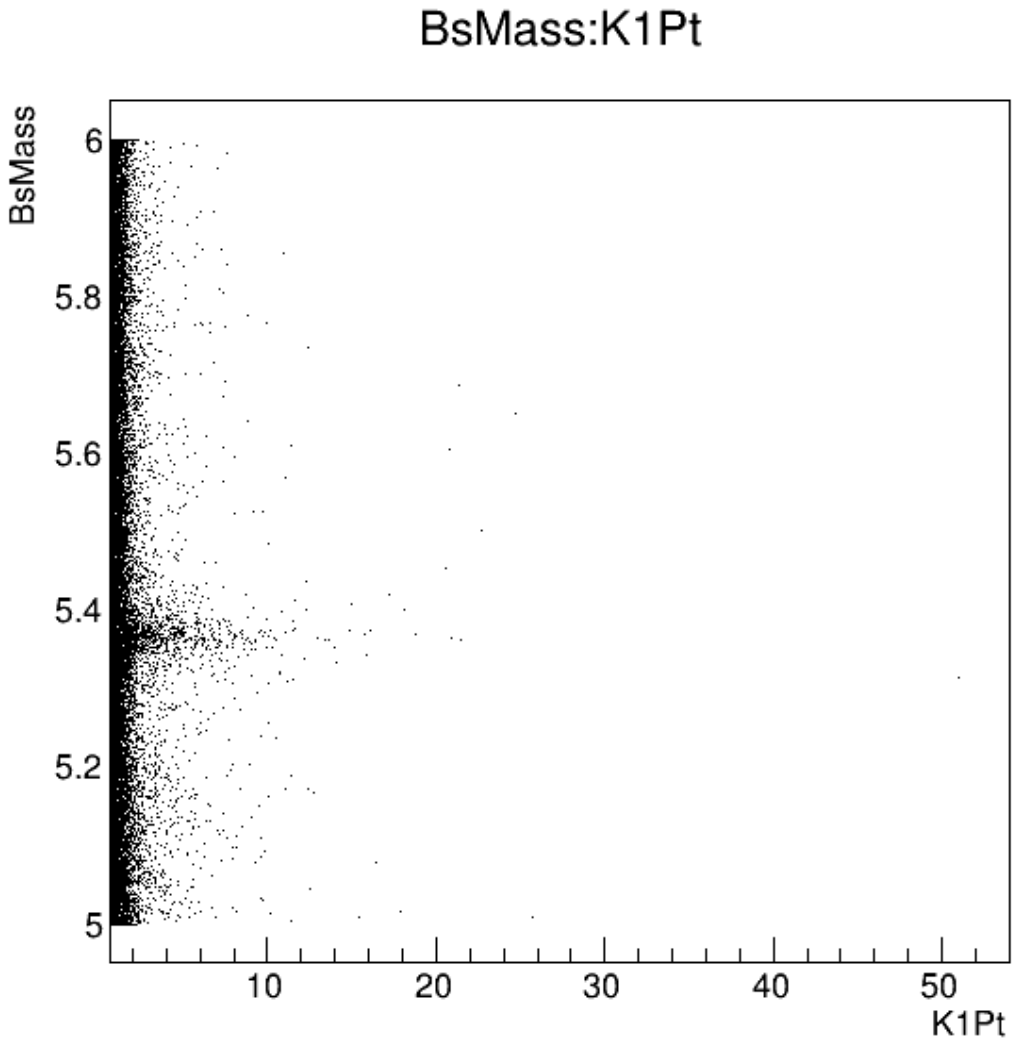


Il risultato ora è molto meglio, su questo istogramma potremo poi fare analisi dati, sul precedente no!

Ora sappiamo come disegnare e ottenere singoli istogrammi dai TTree, la potenzialità maggiore tuttavia è la possibilità di creare plot bidimensionali partendo dai singoli dati (ad esempio per cercare correlazioni) ma soprattutto la possibilità di usare il metodo Draw per "filtrare" i dati prima di salvarli in un istogramma, facciamo alcuni esempi:

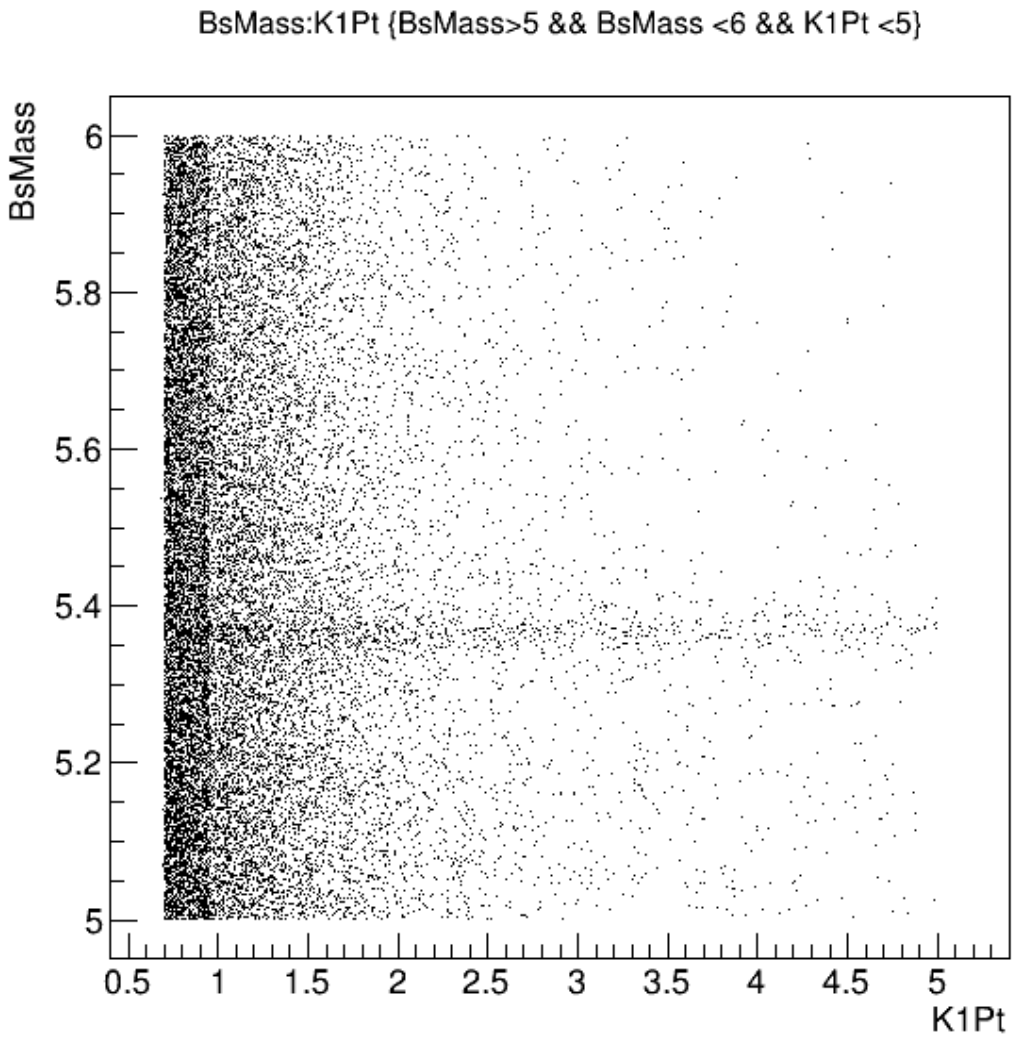
I dati nell'albero sono in verità eventi con un alto tasso di fondo, per eliminare il fondo dobbiamo scegliere alcuni "tagli" con cui filtrare i dati, per esempio proviamo a vedere se ci sono correlazioni tra la branch K1Pt e la branch BsMass, per fare ciò bisogna usare sempre il metodo Draw chiedendo di mostrare entrambe le branch:

[19]:
```
%%cpp
TCanvas *kp1bs= new TCanvas("kp1bs", "kp1bs", 600,600);
kp1bs->cd();
data->Draw("BsMass:K1Pt");
kp1bs->Draw();
```



Usando questo range tuttavia non si riesce molto a intuire molto bene la distribuzione dei dati, proviamo a restingere un po' il range, per fare ciò possiamo dare al metodo Draw delle istruzioni che filtrino i dati, ad esempio:

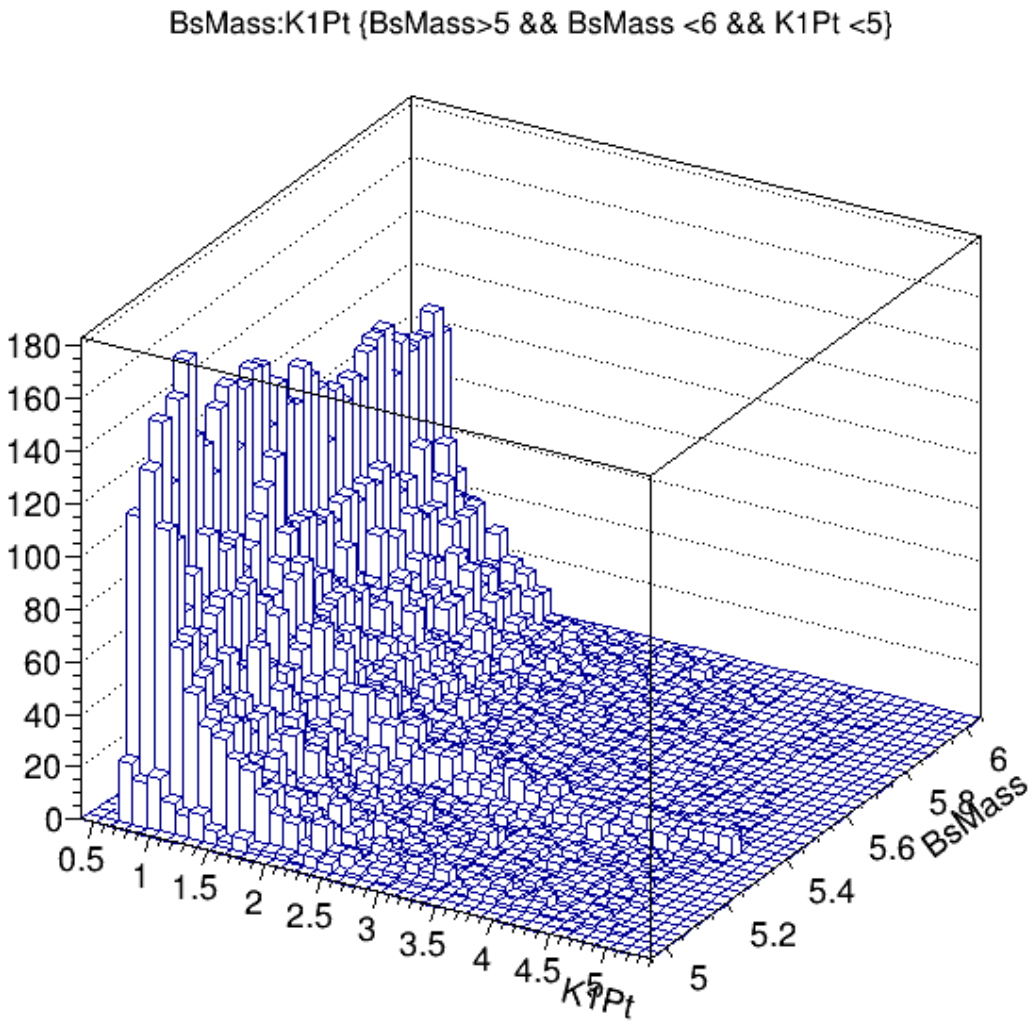
[20]:
```
%%cpp
TCanvas *kp1bs2= new TCanvas("kp1bs2", "kp1bs2", 600,600);
kp1bs2->cd();
//scegliamo di plottare la massa nel range [5,6] e k1pt nel solo con valori␣
 ↪sotto al 5
data->Draw("BsMass:K1Pt", "BsMass>5 && BsMass <6 && K1Pt <5");
kp1bs2->Draw();
```



L'effetto è gia migliore e in effetti possiamo vedere che all'intorno del 5.375 circa c'è una leggera zona più fitta, per migliorare ulteriormente possiamo usare anche le opzione del metodo Draw viste in precedenza, ad esempio l'opzione lego

[21]:
```
%%cpp
TCanvas *kp1bs3= new TCanvas("kp1bs3", "kp1bs3", 600,600);
```

```
kp1bs3->cd();
//questo serve per ruotare da terminale
gPad->SetPhi(-30);
//scegliamo di plottare la massa nel range [5,6] e k1pt nel solo con valori␣
 ↪sotto al 5
data->Draw("BsMass:K1Pt", "BsMass>5 && BsMass <6 && K1Pt <5", "lego");
kp1bs3->Draw();
```



In questo caso si vede poco ma è perchè abbiamo pochi eventi, nel caso poi si volesse questo plot può essere esportato in un TH2F nello stesso modo in cui si è esportato il TH1F di prima.

In conclusione Ntuple e TTree sono molto utili per cercare correlazioni e filtrare i dati, i quali possono poi essere raccolti in dei TH1 per l'analisi successiva.

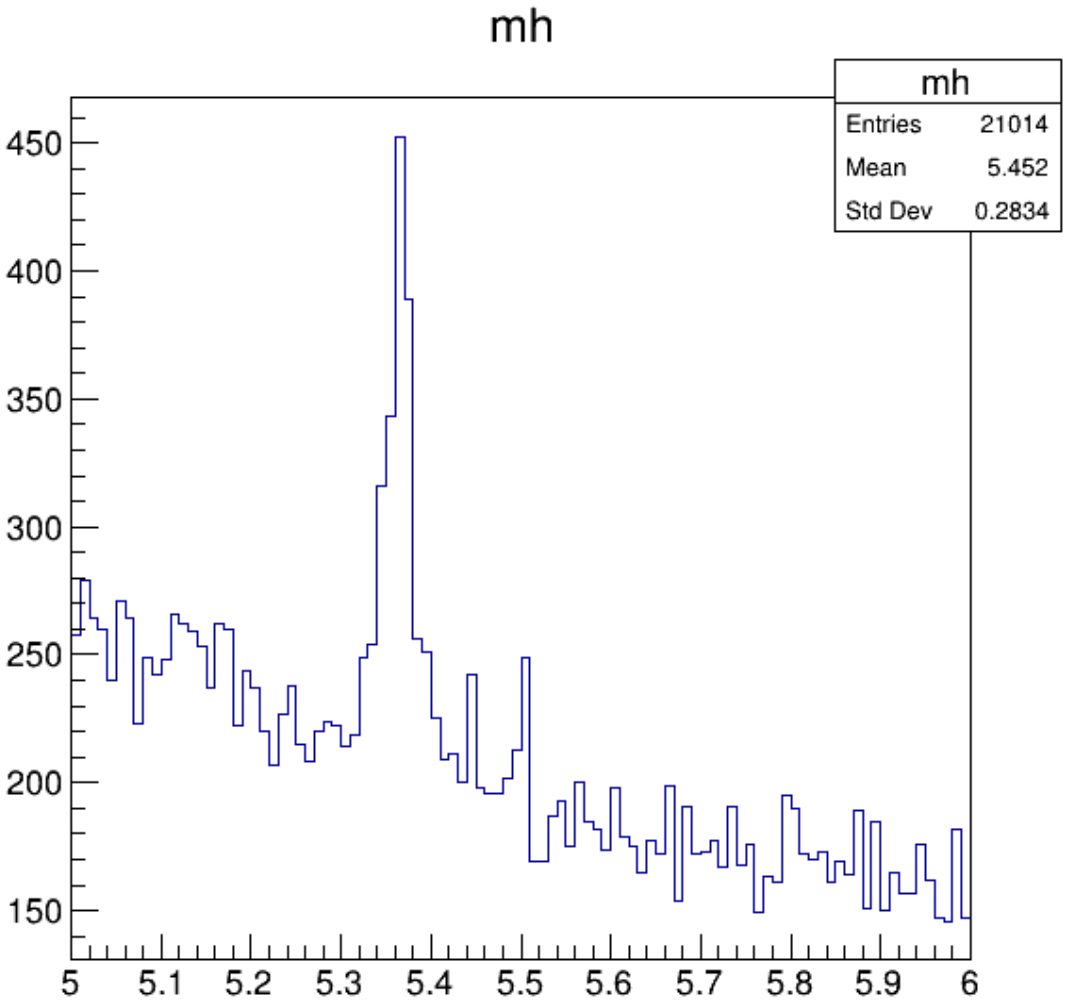
NOTA Non abbiamo fatto esempi espliciti poichè in generale non è una funzionalità usata ma

è possibile accedere al singolo evento di un TTree nel seguente modo, si deve usare il metodo GetVal(int) per avere un array (del tipo di dato della corrispondente branch) con il quale si possono vedere i singoli dati. Per poi farne un loop si può avere il loro numero con il metodo GetEntries(). Per esempio proviamo a ricevere tutti i dati di BsMass (la branch 0 del TTree)

[22]:
```
%%cpp
double *mar=data->GetVal(0);
cout<<mar[10];
```

```
5.48444
```

[23]:
```
%%cpp
TH1F *mh= new TH1F("mh", "mh", 100, 5, 6);
for (int i=0; i<data->GetEntries(); i++)mh->Fill(mar[i]);
TCanvas *mc=new TCanvas("mc", "mc", 600, 600);
mh->Draw();
mc->Draw();
```



[ ]: