Лекции. Практические занятия

Солдатов Е.Ю.

2024 г.

ДЕРЕВЬЯ/НТЮПЛЫ

• Дерево (tree) в ROOT является аналогом ntuple (n-мерный кортеж) в PAW. Позволяет хранить большое количество данных одного типа или объектов одного класса.
• В ROOT деревья реализуются классом **TTree**.
• **TNtuple** также реализован в ROOT.

• Ntuple можно рассматривать как таблицу, каждая строка которой соответствует одному вхождению (событию), а столбцы —конкретным переменным

• Принципиальное отличие дерева от ntuple заключается в том, что содержимое ntuple ограничено данными типа *float*. Ветви дерева могут содержать переменные других типов, вплоть до объектов (а также массивы)
• Чаще всего каждому вхождению соответствует физическое событие (реальное или смоделированное). Однако существуют и другие варианты организации дерева (например, заполнение по трекам)
• Класс TNtuple — этоTTree, ограниченное значениями типа Float_t

Ref.manual: https://root.cern.ch/doc/master/classTNtuple.html

ДЕРЕВЬЯ/НТЮПЛЫ

• Данные хранятся в узлах (нодах), узлы организованы в структуру, которая может быть представлена в виде таблицы со столбцами.

• Для дерева столбцы такой таблицы называются ветвями (branch)
• Ветви реализуются классом **TBranch**

• Листы – элементарная единица данных в ветви. Лист содержит фактическое значение переменной.

• Каждая ветвь задаётся именем и типом данных.
I – целочисленный тип (Int_t)
F – тип с плавающей точкой (Float_t)
D – тип с плавающей точкой двойной точности (Double_t)
C – символьный тип (Char_t)
b – целочисленный тип байта
S – целочисленный тип short (Short_t)
O – логический тип (bool)
и т.д.

- Заметим, что в основном используются собственные типы!
- При записи на диск срабатывает алгоритм компрессии.

Ref.manual: https://root.cern.ch/doc/master/classTBranch.html

Деревья

- Создать объект класса **TTree**
  - *TTree *t1 = new TTree("t1", "Simple Tree")*
- Конструктору дерева передается два параметра
  “t1” – имя (идентификатор) дерева
  “Simple tree” – заголовок дерева
- Чтобы добавить к дереву ветвь **TTree::Branch**
  - *t1->Branch("px", &px, "px/F")*
- Три параметра, определяющие ветвь
  - Имя ветви
  - Адрес, по которому будет считываться значение переменной.
  - Напоминание: &—операция взятия адреса в C
  - Тип листа в формате имя/тип.

  Наиболее употребляемые типы были приведены на предыдущем слайде.
- Чтобы занести значения в дерево, используется метод **TTree::Fill**

Ref. manual: https://root.cern.ch/doc/master/classTTree.html

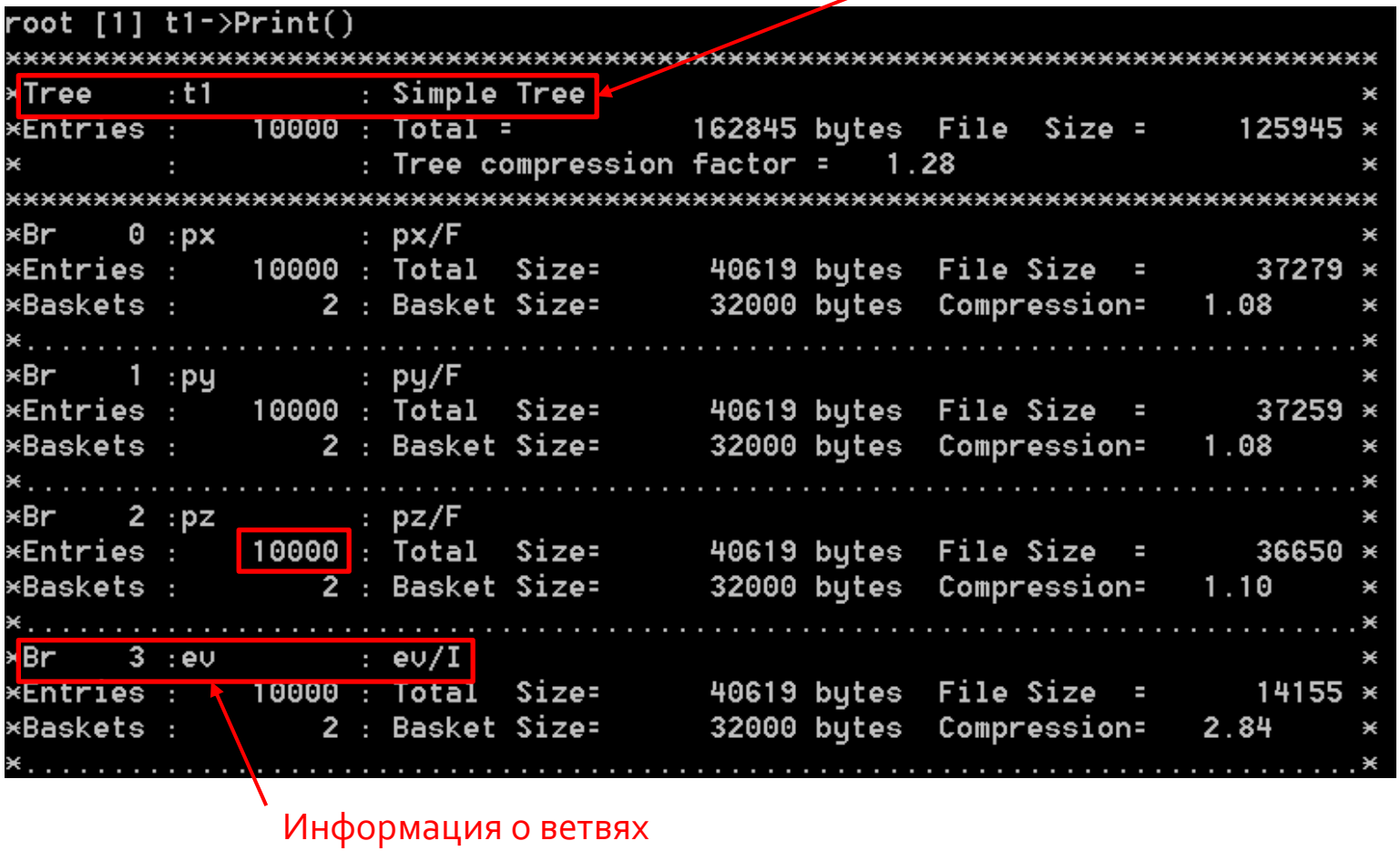
Деревья: создание и запись

- Скрипт, создающий простое дерево и записывающий его в файл **tree1.root**

```
{
TFile *f = new TFile("tree1.root", "recreate");    //создаем файл
TTree *t1 = new TTree("t1", "Simple Tree");    //создаем дерево
Float_t px, py, pz;   //определяем необходимые переменные
Int_t ev;
t1->Branch("px", &px, "px/F");   //создаем три ветви
t1->Branch("py", &py, "py/F");    //содержащие значения Float_t
t1->Branch("pz", &pz, "pz/F");
t1->Branch("ev", &ev, "ev/I");     //и одну со значениями Int_t
for (Int_t i=0; i<10000; i++) {   //заполнение дерева в цикле
gRandom->Rannor(px,py);
pz= px*px+ py*py;
ev= i;
t1->Fill();                //по команде Fill значения переменных
}                          //заносятся в дерево
f->Write();                //записываем дерево в файл
f->Close();          //закрываем файл
}
```

Деревья

- Вывести общую информацию о дереве:



Деревья

- Вывести все значения, записанные в i-ом вхождении (событии)
  t1->Show(i)

```
root [2] t1->Show(123)
=======> EVENT:123
 px                = 0.0741416
 py                = 0.155682
 pz                = 0.0297337
 ev                = 123
```

Деревья: запись структуры

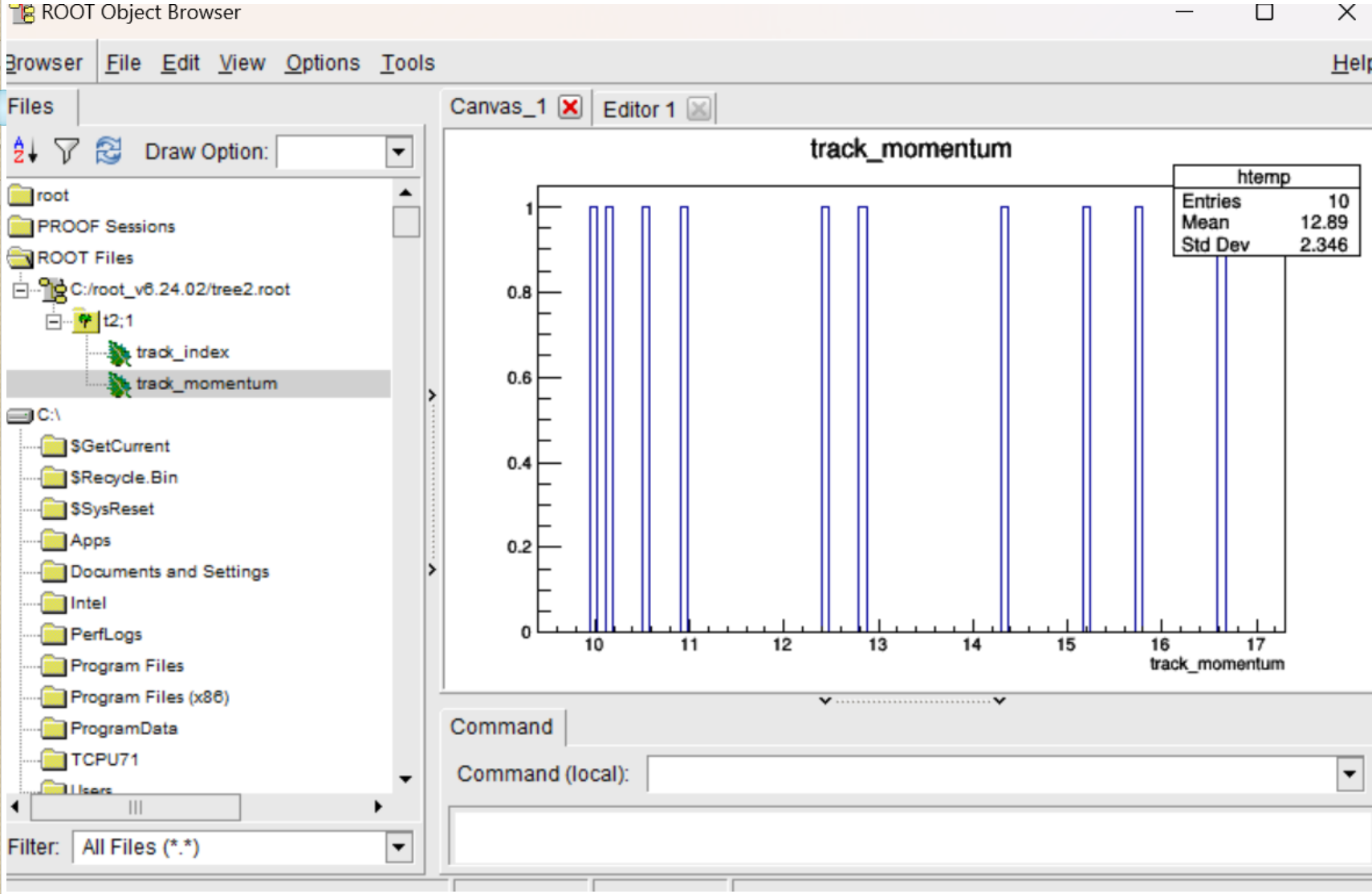
```
struct TrackData {
        Int_t trk_index;
        Float_t momentum;
};

void TreeWrite()
{
        TFile *f2 = new TFile("tree2.root", "recreate");    //создаем файл
        TTree *t2 = new TTree("t2", "Track Data Tree");    //создаем дерево
        TrackData data;
        TRandom3* rnd = new TRandom3();

//создаем ветви дерева, связанные со структурой
        t2->Branch("track_index", &data.trk_index, "track_index/I");
        t2->Branch("track_momentum", &data.momentum, "track_momentum/F");

        for (Int_t i=0; i<10; i++) {   //заполнение дерева в цикле
        data.momentum=10+i*rnd->Rndm();
        data.trk_index=i;
        t2->Fill();                    //по команде Fill значения переменных
        }                              //заносятся в дерево
        f2->Write();                   //записываем дерево в файл
        f2->Close();
}
```

Деревья: запись структуры



Принципы записи объектов в дерево такие же, как и для структуры.

Деревья: чтение

- Прежде всего, следует описать переменные, в которые будут считываться значения

- Затем указать адреса переменных, в которые будут считываться ветви с помощью метода TTree::SetBranchAddress
  - SetBranchAddress(“px”, &px)
- Два параметра метода
  - Имя ветви
  - Адрес переменной, в которую следует записывать считанные данные

- Общее число вхождений в дерево TTree::GetEntries()

- Чтение переменных происходит по команде TTree::GetEntry(i)
  - i – номер вхождения, которое необходимо считать

- Следующий скрипт иллюстрирует процесс чтения дерева

Деревья: чтение

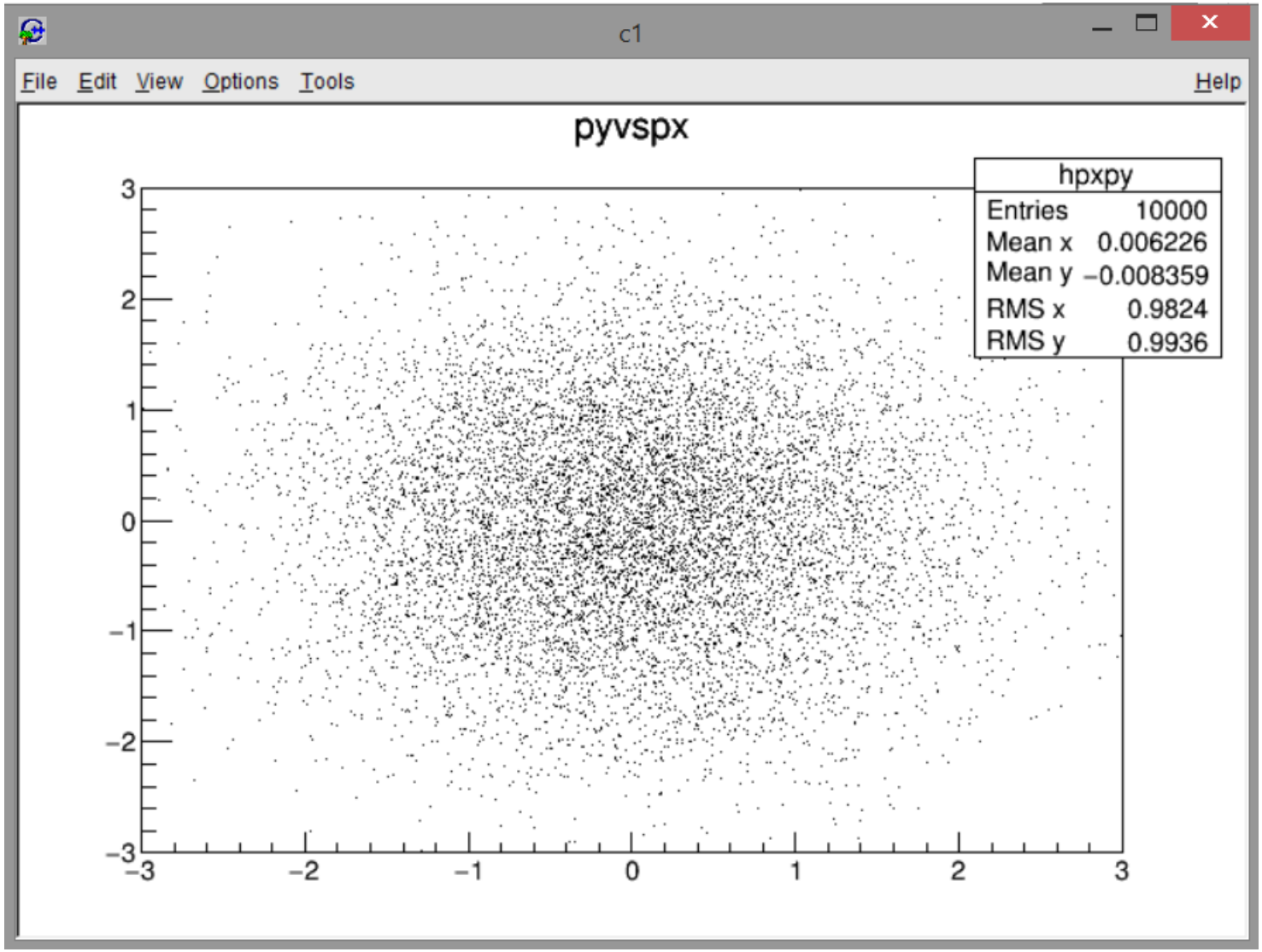
- Скрипт, считывающий данные дерева `t1`, сохраненного в файле `tree1.root`

```
{
TFile *f = new TFile("tree1.root");
TTree *t1 = (TTree*)f->Get("t1");
Float_t px, py, pz;
Int_t ev;
t1->SetBranchAddress("px",&px);
t1->SetBranchAddress("py",&py);
t1->SetBranchAddress("pz",&pz);
t1->SetBranchAddress("ev",&ev);

TH2F *hpxpy = new TH2F("hpxpy","py vs px",30,-3,3,30,-3,3);
Int_t nentries = (Int_t)t1->GetEntries();
for (Int_t i=0; i<nentries; i++) {
  t1->GetEntry(i);
  hpxpy->Fill(px,py);
}
hpxpy->Draw();
}
```

Деревья

- По выполнении скрипта будет нарисована двумерная гистограмма



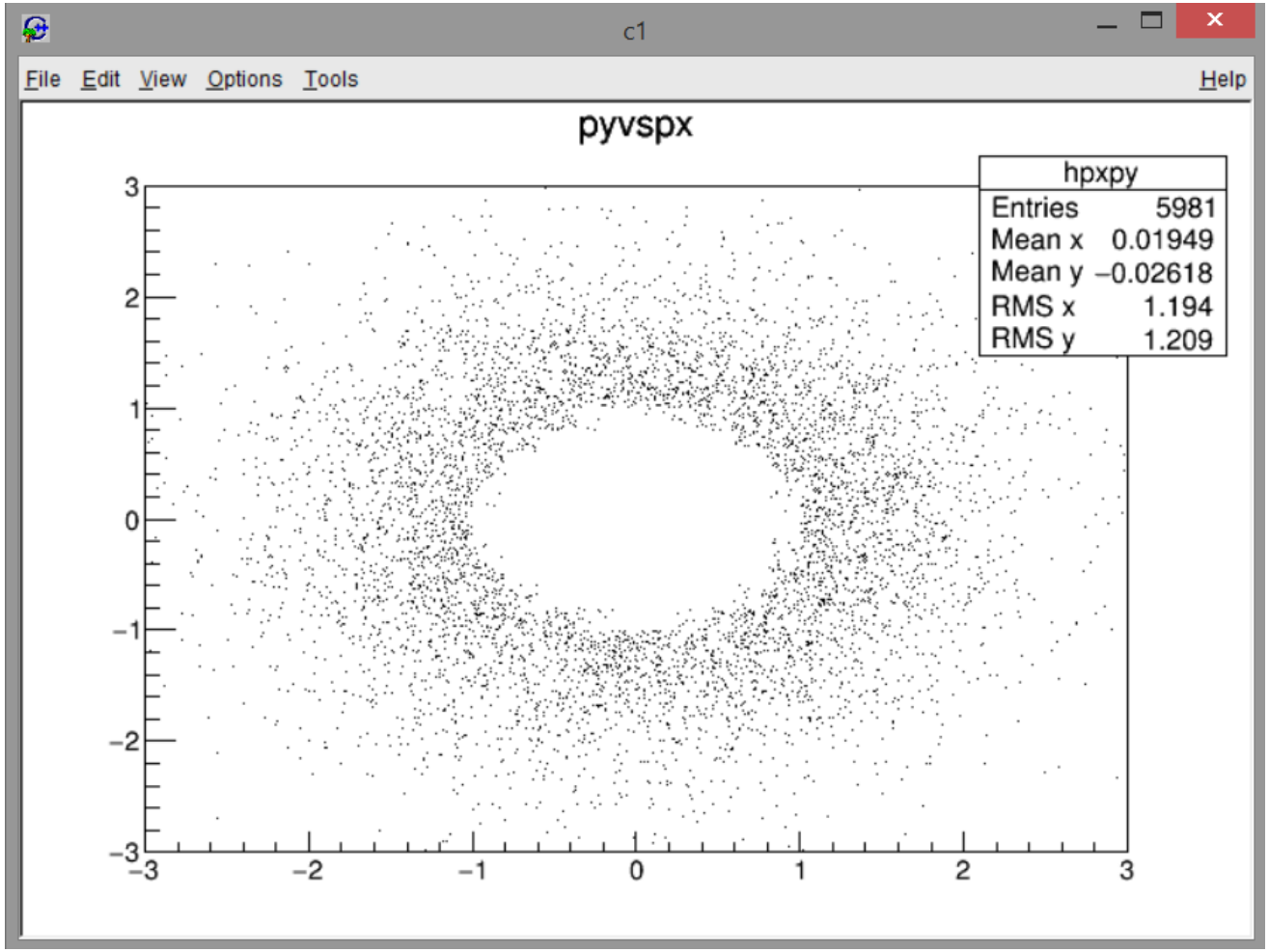
Деревья: анализ

• Анализ данных производится наложением условий, обусловленных физикой процесса, экспериментальными ограничениями и т.п.

Например, если одну строчку кода модернизировать:

```
if(pz> l ){ hpxpy->Fill(px,py); }
```

получится:



Деревья: чтение структуры

```
void TreeRead()
{
        TFile *f2 = new TFile("tree2.root");    //читаем файл
        TTree *t2 = dynamic_cast<TTree*>(f2->Get("t2"));    //читаем дерево
        TrackData data;

//связываем ветви дерева с с полями структуры
        TBranch *b_trk_index = t2->GetBranch("track_index");
        TBranch *b_trk_momentum = t2->GetBranch("track_momentum");
        b_trk_index->SetAddress(&data.trk_index);
        b_trk_momentum->SetAddress(&data.momentum);

        for (Int_t i=0; i<t2->GetEntries(); i++) {   //считывание дерева в цикле
        b_trk_index->GetEntry(i);
        b_trk_momentum->GetEntry(i);
        std::cout<<"Track index: "<< data.trk_index <<"; track momentum: "<<
data.momentum << " GeV."<<std::endl;
        }

        f2->Close();
}
```

Деревья: чтение структуры

```
root [0] .L read_struct.C
root [1] TreeRead()
Track index: 0; track momentum: 10 GeV.
Track index: 1; track momentum: 10.1629 GeV.
Track index: 2; track momentum: 10.5652 GeV.
Track index: 3; track momentum: 12.8416 GeV.
Track index: 4; track momentum: 10.9266 GeV.
Track index: 5; track momentum: 12.4249 GeV.
Track index: 6; track momentum: 15.7449 GeV.
Track index: 7; track momentum: 15.2101 GeV.
Track index: 8; track momentum: 14.3203 GeV.
Track index: 9; track momentum: 16.6596 GeV.
root [2]
```

Деревья: анализ в интерактивном режиме (1/2)

- Построить гистограмму из значений ветки px дерева tree
  - tree->Draw("px")

  Создаёт в текущей директории гистограмму с названием htemp, которую можно получить при помощи команды
  - TH1F *h = (TH1F*)gDirectory->Get ("htemp")

- Также можно строить многомерные гистограммы, разделяя имена веток двоеточиями
  - tree->Draw("px:pz")

- При помощи оператора >> можно указать название новой гистограммы, чтобы отличать её от других в директории, а также задать её биннинг в формате (nBins, min, max{, до 3 измерений})
  - tree->Draw("px:pz>>px_vx_pz(10, 0, 150, 20, 0, 300)")

    px pz

- Можно использовать не только имена веток, но и все доступные функции C++, в том числе и из библиотеки ROOT. В данном примере используется функция вычисления разности полярного угла TVector2::Phi_mpi_pi(x)
  - tree->Draw("TVector2::Phi_mpi_pi(photonPhi - metPhi)")

Деревья: анализ в интерактивном режиме (2/2)

Второй параметр в функции Draw() является весом, на который умножается каждое из вхождений в гистограмму. В качестве него можно использовать:

- Булевые выражения (0 и 1), для отбора событий. Например, построить распределение по px только для событий, где pz больше 150
  - tree->Draw("px", "pz > 150")
- Числа с плавающей точкой, для использования весов событий. Например, построить распределение по px, используя теоретически рассчитанный вес из ветки weight
  - tree->Draw("px", "weight")
- Их комбинацию.
  - tree->Draw("px", "weight * (pz > 150)")

Важно! Обращайте внимание на то, как рассчитывается выражение для второго параметра:

- weight * (pz > 150) – использовать значение веса weight и результат отбора (pz > 150)
- weight * pz > 150 – строить распределение только для событий, где значение (weight * pz) больше 150

Больше информации о функции Draw(): см. Reference Manual

Ещё о Draw()

- Этот дополнительный конструктор Draw() позволяет ещё некоторые вещи.

- Draw(varexp,selection,option,nentries,firstentry)

  varexp – выражение, состоящее из идентификаторов веток, px, px:py

  selection – вес или отбор

  option – опции рисования

  nentries – сколько вхождений нужно отрисовать

  firstentry – с какого вхождения в дерево нужно начать

Например, нужно нарисовать 1 вхождение под номером 123, тогда nentries=1, firstentry=123.

Предположим в ветке записан вектор. Как получить 5 элемент вектора в 123 вхождении?

```
output_tree->Draw("weight_vec[5]","","",1,123)
```

Работа с файлами в интерактивном режиме

- Убрать логотип при запуске ROOT (отключен по умолчанию с версии 6.20):
  - root -l
- Создать объект TFile _file0, открыв файл inputFile.root
  - root -l inputFile.root

При работе с ROOT в интерактивном режиме, можно использовать все объекты в текущей директории напрямую по имени, не создавая для них специальных переменных. Так, узнать содержимое файла inputFile.root можно при помощи двух команд:

- root -l inputFile.root
- _file0->ls()

Также это означает, что можно напрямую работать с деревьями и гистограммами, записанными в этот файл. Так, если файл содержит дерево output_tree и гистограмму cutFlow, то узнать содержимое этого дерева и нарисовать эту гистограмму можно при помощи команд:

- root -l inputFile.root
- output_tree->Print() (или output_tree->Show(0))
- cutFlow->Draw()

Работа с файлами в интерактивном режиме Нововведения.

Начиная с версии 6.06 при установке ROOT, помимо самой команды root, появляются новые команды для более удобной работы с файлами.

Наиболее интересной из них является команда rootbrowse, которая позволяет заменить часто используемую последовательность команд:

- root -l inputFile.root
- TBrowser b

командой

- rootbrowse inputFile.root

Обновление документации ROOT

Авторы программного пакета ROOT улучшили документацию, которая теперь доступна по ссылке https://root.cern/doc/master/index.html

Особое внимание стоит обратить на вводный курс с подробными слайдами и примерами, доступный по ссылке https://github.com/root-project/training/tree/master/BasicCourse
Он будет хорошим дополнением к данному, более подробно раскроет некоторые из затронутых тут тем, а также даст вводные знания по новым темам: pyroot и как перейти на него с C++, JSROOT, новый способ параллельной обработки содержащих большие деревья файлов TDataFrame и многим другим.

Также для ROOT было написано множество примеров, которые можно найти как в папке /tutorials/ внутри пакета, так и по ссылке https://root.cern/doc/master/group__Tutorials.html

Автоматическое создание скелета класса для анализа

• Предположим у вас есть файл с деревом данных (tree_data), которые необходимо анализировать.
Там может быть много ветвей.
• В ROOT есть возможность не писать самому скелет программы для анализа (подгружающий всё дерево).
Есть методы *MakeClass* и *MakeSelector*

• Открыв файл можно выполнить:

```
tree_data->MakeClass("my_analysis");
```

или

```
tree_data->MakeSelector("my_analysis");
```

• Создадутся заголовочный файл my_analysis.h и основной my_analysis.cxx
Чтение нужного вам дерева уже будет там прописано.
• Скелеты методов также уже будут созданы (инициализация, окончание, основное выполнение).

• Вам остаётся лишь написать код для анализа данных в соответствующем методе. Всё!