

## Лекция 1. Введение в предмет курса. Команды ОС Linux

На современном уровне развития ускорительного эксперимента, все более его тесной связи с информационными технологиями, специалист в области физики высоких энергий<sup>1</sup> для успешной работы должен обладать серьезной, разносторонней подготовкой в методах обработки данных и компьютерного моделирования.

Цель настоящего курса состоит в том, чтобы познакомить слушателя с наиболее часто применяемыми в физике высоких энергий программами, показать общие приемы работы с ними и привести конкретные примеры, имитирующие реальные задачи, встречающиеся на практике.

Каждый из программных пакетов, обсуждаемых в настоящем курсе, имеет соответствующее руководство, объем которого зачастую составляет несколько сотен страниц. Данный курс не претендует на детальность, а скорее является «курсом молодого бойца», сжатым изложением наиболее важных сведений. По завершении курса студент должен получить базовые навыки обращения с рассмотренными программами, быть способным самостоятельно решать простейшие задачи и знать, каким образом получить необходимые для более профессиональной работы недостающие сведения.

Логика построения курса отвечает поставленным задачам. Начать обучение следует с базовых программных сред, предназначенных для обработки и хранения данных, графического представления информации<sup>2</sup>. В настоящее время *de facto* стандартом такой среды является мощный программный пакет ROOT, реализованный на C++. На изучение этого пакета отводится значительная часть курса. Однако ряд действующих экспериментов по сей день использует проверенную временем Fortran среду для обработки данных PAW. Поэтому начинается курс именно с изучения PAW. Разделы курса, посвященные PAW и ROOT во многом сходны: в них рассматривается работа с гистограммами, ntuples (для PAW) и TTree (для ROOT), сохранение информации в файл.

Чтобы изучать какие-либо процессы, необходимы физические генераторы (для ускорительной физики — генераторы столкновений). В курсе рассматривается наиболее распространенный генератор протон-протонных столкновений PYTHIA. Работа PYTHIA иллюстрируется на рождении  $Z^0$ - и Хиггс-бозона. Генераторы ядро-ядерных столкновений в рамках курса представлены популярными программами UrQMD и HIJING.

---

<sup>1</sup> А, вообще говоря, практически во всех областях физики

<sup>2</sup> Заметим, что эти задачи не продиктованы спецификой ускорительного эксперимента, следовательно, соответствующее программное обеспечение может применяться (и успешно применяется) в самых разных областях знания — от биологии до экономики

Чтобы изучать отклик экспериментальной установки на частицы, рожденные в столкновении, необходимо иметь программу, которая бы моделировала прохождение частиц через вещество детекторов. В настоящем курсе изучается программа GEANT<sup>3</sup>.

Завершает курс лекция, посвященная программному пакету AliROOT, предназначенному для всесторонней информационной поддержки эксперимента ALICE, подготовка которого к работе на ускорителе LHC близится к завершению. AliROOT, как следует из названия программы, базируется на пакете ROOT, и для своей работы использует генераторы столкновений (PYTHIA, HIJING, ...), а также программы для моделирования прохождения частиц через вещество. На примере AliROOT иллюстрируются принципы организации комплексного программного обеспечения для современного эксперимента.

Программные пакеты не могут работать сами по себе — для их функционирования необходима операционная среда (ОС). Программы, рассмотренные в рамках курса, реализованы под ОС Linux<sup>3</sup>.

Напомним наиболее важные консольные команды Linux.

#### 1. Работа с файлами и каталогами

посмотреть список файлов в директории

```
ls dirname -al
```

узнать текущую рабочую директорию

```
pwd
```

сменить директорию

```
cd dirname
```

создать директорию

```
mkdir dirname
```

удалить директорию

```
rmdir dirname
```

копировать/переместить файл с именем “file1” в файл с именем “file2”

```
cp file1 file2
```

```
mv file1 file2
```

скопировать рекурсивно содержимое директории

```
cp -r dirname .
```

копирование всех файлов с расширением \*.root из директории dirname в текущую директорию

```
cp dirname/*.root .
```

копирование ряда файлов file1, ..., fileN в директорию dirname

---

<sup>3</sup> Программный пакет ROOT можно установить и под ОС Windows

```
cp file1 ... fileN dirname
```

удалить пустую директорию

```
rm dirname
```

удалить файл

```
rm filename
```

удалить рекурсивно директорию со всеми содержащимися в ней файлами

```
rm -r dirname
```

Напоминаем, что в Linux **крайне затруднительно восстановить удаленный файл**, поэтому всегда необходимо четко понимать, какой конкретно файл подвергается удалению.

Шаблоны для обозначения директорий

..       родительский каталог (каталог на 1 уровень выше)

.         текущий каталог

~userN   домашний каталог пользователя с учетной записью userN

/         корневой каталог

## 2. Установка переменных окружения

для bash-оболочки:

```
export VARNAME=VALUE
```

Пример

```
export ROOTSYS=/home/cern/root-5.16.00
```

Чтобы вывести значение переменной окружения на экран

```
echo $ROOTSYS
```

## 3. Компилирование программ

Компилятор Fortran g77

Компилятор C++ g++

Компиляция программы

```
g++ prog.C -o prog
```

Подключение библиотек

```
-L/pathtolib -llibname
```

Немаловажную роль в работе программиста играет используемый текстовый редактор. В самом деле, хорошее знание редактора делает работу более эффективной, существенно сокращает время, затрачиваемое на рутинные задачи. Выбор текстового редактора во многом является делом вкуса и личных предпочтений пользователя. На взгляд авторов, наиболее подходящими инструментами являются текстовые редакторы

JOE <http://sourceforge.net/projects/joe-editor/> и Emacs <http://www.gnu.org/software/emacs/>. К достоинствам первого следует отнести простоту, небольшой размер и удобство пользования. Редактор Emacs отличается колоссальной функциональностью; его использование оправдано при создании больших программных пакетов. Оба редактора входят в состав большинства современных дистрибутивов Linux.

Рассмотрим более подробно работу в редакторе JOE. Чтобы запустить редактор и начать редактирование какого либо файла (создать его, если он не существует), например test.C, наберите в командной строке

```
$ joe test.C
```

Полагаем уместным привести здесь небольшой справочник команд, необходимых для эффективной работы в редакторе JOE. Комбинация клавиш, соединенных знаком "+", реализуется так: необходимо нажать первую клавишу, затем, удерживая ее, вторую. Запятая означает, что после нажатия комбинации, необходимо отпустить кнопки и нажать клавишу, которая указана после запятой.

<b>выполняемое действие</b>	<b>комбинация клавиш</b>
выйти из редактора без сохранения файла	Ctrl+C
сохранить файл	Ctrl+K, D
выйти и сохранить изменения	Ctrl+K, X
искать в тексте файла	Ctrl+K, F
перейти на строку с определенным номером	Ctrl+K, L
удалить строку целиком	Ctrl+Y
«срезать» строку с текущего положения курсора	Ctrl+J
перейти на страницу вверх	Ctrl+U
перейти на страницу вниз	Ctrl+V
перейти в начало файла	Ctrl+K,U
перейти в конец файла	Ctrl+K,V
пометить начало блока	Ctrl+K, B
пометить конец блока	Ctrl+K, K
удалить блок	Ctrl+K, Y
копировать блок	Ctrl+K, C
переместить блок	Ctrl+K, M
записать блок в файл	Ctrl+K, W
вставить содержимое из файла	Ctrl+K, R

## Лекция 2. PAW. Объекты PAW, основные команды

PAW или Physics Analysis Workstation — это интерактивная программа анализа и графического представления результатов, с возможностью автоматизации посредством написания скриптов.

PAW позволяет работать с большими объемами данных. Данные в основном представлены в виде списка не связанных между собой событий (ntuple). Данные можно представить в виде одномерных или двумерных гистограмм. В PAW реализована фильтрация событий по условию.

Пакет PAW был создан для целей физики элементарных частиц (ФЭЧ). История PAW берет начало в 1986 году в европейском центре ядерных исследований (CERN). В процессе разработки основной упор делался на работу с очень большим объемом данных.

По сути PAW является интерфейсом к набору библиотек CERNLIB. Все, что можно сделать в PAW интерактивно, также можно реализовать в скомпилированном программном коде. В PAW встроен интерпретатор языка программирования Fortran.

PAW не в явном виде основан на семействе фундаментальных объектов. Каждая команда PAW выполняет действие, которое или производит другой объект или производит «побочный эффект», такой, как вывод сообщения или графическое отображение, которое не сохраняется где-нибудь как структура данных. Для работы в PAW не требуется доскональных знаний всех подсистем и команд. Чтобы действовать эффективно, нужно изучить логику построения команд и основные стандартные приемы работы.

Для начала рассмотрим основные объекты при работе в PAW. PAW оперирует как векторами, так и гистограммами, ntuple'ами и cut'ами. Эти объекты являются для PAW базовыми.

**Вектора.** Вектора представляют собой одномерные и многомерные массивы чисел. Для работы с векторами предоставляется интерфейс к пакету для манипуляций с массивами SIGMA и интерпретатору Fortran COMIS. Вектора используются для анализа небольших объемов текстовых данных.

**Одномерные гистограммы.** Гистограмма — это базовый объект для анализа в PAW. Гистограмма представляет собой столбчатую диаграмму — один из видов графического представления эмпирических распределений. Чтобы создать (book) гистограмму достаточно определиться с пределами гистограммы и числом элементарных бинов (bin). При заполнении (fill) гистограммы в соответствующий записываемому числу бин

добавляется единица с определяемым пользователем весом (weight). В PAW любой гистограмме может приписываться подгоночная функция (fit) с результатами подгонки.

**Двумерные гистограммы.** Двумерная гистограмма является обобщением обычной одномерной гистограммы на плоскости. В отличие от одномерной гистограммы, при отображении двумерной возникает серьезная проблема выбора графического представления данных. Если есть возможность, то лучше все сводить к одномерным гистограммам.

**Ntuple.** Ntuples представляет собой очень удобный инструмент для того, чтобы анализировать статистические наборы данных. Ntuple можно рассматривать как обычную таблицу, где каждая строка соответствует одному событию, а каждый столбец соответствует конкретной переменной. Типичное рассматриваемое событие — это событие рассеивания или столкновения частиц, зарегистрированное в эксперименте физике высоких энергий, где переменными могут выступать угол рассеяния, число треков, выделяемая энергия в калориметрах и т. д. Стоит также заметить, что Ntuples полезны и для любого другого не физического анализа набора статистических данных. Интересующие свойства данных в Ntuple могут обычно выражаться как распределения переменных Ntuple или как корреляции между двумя или большим числом этих переменных. В Ntuple существует возможность строить новые распределения из подмножества данных при ограничениях (обычно используют «cuts») на некоторые из переменных.

**Cuts.** Ограничения — это булева (логическая) функция переменных Ntuple. Ограничения используются, чтобы отбирать событие из подмножества событий в Ntuple, заполняя при этом гистограммы.

**Masks.** Маска — это отдельные файлы, которые являются логически идентичными ряду логических переменных добавленных в конец структуры данных Ntuple. Mask создается, используя логический результат применения ограничений к набору событий. Маска нужна только для удобства работы.

**Styles (Стили).** Объект «стиль» представляет собой набор переменных, которые управляют видом графиков в PAW. Команды параметра OPTION выбирают специфические опции составления графика, типа логарифмического/линейного, bar-chart/scatter-plot, дисплей статистики и т.д. Команды значения параметра SET управляют обширным набором числовых параметров формата, используемых, чтобы управлять составлением графиков.

Стоит также отметить, что PAW — это система, включающая в себя различные инструменты и пакеты, которые могут также использоваться независимо от PAW. Некоторые из них даже имеют более долгую историю, чем PAW (HBOOK и HPLOT, SIGMA, COMIS, MINUIT). В данном курсе невозможно рассмотреть все пакеты, но некоторым из них мы уделим небольшое внимание.

Перейдем к работе с PAW.

Программа PAW запускается командой `raw`. При запуске PAW задает традиционный вопрос о выборе типа окна для отображения. На сегодня интерес представляет обычное X-окно (появляется по умолчанию, размеры которого заданы в файле `higz_windows.dat`), или буквенно-цифровой терминал. В последнем случае результаты выводятся с помощью текста, то есть для представления данных достаточно окна терминала.

Во время запуска PAW считывает и выполняет команды из файла `~/.rawlogon.kumac`. Получив после инициализации приглашение, можно приступить к работе.

Чтобы окончить сеанс достаточно команды `exit`. В случае если PAW удалось тем или иным образом «зациклить», то на помощь приходит прерывание `^C`.

Одной из самых полезных команд в PAW является команда `help`

```
PAW> help
1 : KUIP Command Processor commands
2 : MACRO Macro Processor commands
3 : VECTOR Vector Processor commands
4 : HISTOGRAM Manipulation of histograms
5 : FUNCTION Operations with Functions. Creation and plotting
6 : NTUPLE Ntuple creation and related operations
7 : GRAPHICS Interface to the graphics packages HPLOT and HIGZ
8 : PICTURE Creation and manipulation of HIGZ pictures
9 : ZEBRA Interfaces to the ZEBRA RZ, FZ and DZ packages
1 0 : FORTRAN Interface to MINUIT, COMIS, SIGMA and FORTRAN Input
/Output
1 1 : NETWORK To access files on remote computers
1 2 : MLP Multi-Layer Perceptron (MLP)
1 3 : OBSOLETE Obsolete commands
```

После ввода команды `help` выводится меню и предлагается выбрать интересующий вас пункт. Обычно, чтобы добраться до необходимой команды требуется пройти 2-4 уровня.

Одними из главных объектов в PAW являются следующие три: вектора, гистограммы и `ntuple`. Начнем изучение с векторов.

При работе с векторами рассмотрим несколько основных команд, более детально со многими командами можно ознакомиться в [1].

Вектора представляют собой аналог массивов в FORTRAN 77.

Для создания вектора используется команда

```
VECTOR/CREATE vname [ type values ],
```

где `VNAME` — имя вектора, а в круглых скобках указывается количество элементов данного вектора и его размерность. Вектора могут иметь размерность до 3. С помощью

`TYPE` — указывается тип данных `R(REAL)` или `I(INTEGER)` и `VALUES` — список вводимых значений.

Вектор заполняется сразу, если указан массив чисел после определения его типа.

Например,

```
VEC/CREATE V(10) R 1 2 3 4 5 66 77 88 99 111
```

Здесь происходит заполнение вектора массивом из десяти чисел.

Если приходится вводить много одинаковых элементов, то можно данную процедуру упростить, с помощью символа звездочка «\*». Для этого сначала указывается число повторяющихся символов, потом звездочка «\*», а затем само число, например:

```
VEC/CREATE Z(20) R 5*1 2 4*3 что эквивалентно
```

```
VEC/CREATE Z(20) R 1 1 1 1 1 2 3 3 3 3
```

В следующем примере показано заполнение только первых трех элементов вектора `W` с объявленным массивом из 20.

```
VEC/CREATE W(20) R 1 2 3
```

В дальнейшем с помощью команды `VECTOR/INPUT` векторные элементы могут быть дополнены или изменены.

Полный формат последней команды имеет вид

```
VECTOR/INPUT vname [ values ]
```

где `VNAME` — имя заполняемого вектора и `VALUES` — список вводимых значений.

`VEC/INPUT V(6:10) 1.1 2.22 3.333 4.4444 5.55555` означает, что заполняется вектор `V` с 6 по 10 элемент.

```
VEC/INPUT V 5*1 2 4*3 что эквивалентно VEC/INPUT V 1 1 1 1 1 2 3 3 3 3
```

С помощью команды `VECTOR/LIST` можно посмотреть список всех векторов созданных в данном сеансе работы.

Для того чтобы удалить вектора из памяти используется команда `VECTOR/DELETE VLIST`, где `VLIST` — список удаляемых векторов. Если после `VLIST` поставить звездочку, то это будет действовать как продолжение, то есть

`VEC/DEL AB*` — удаляет все вектора, которые начинаются с `AB`

`VEC/DEL *` — удаляет все имеющиеся вектора.

Для того чтобы прочитать данные из текстового файла и записать их в вектора, используют команду

`VECTOR/READ vlist fname [ format opt match ]`, где `vlist` — имя вектора, `fname` — файл из которого происходит считывание данных, `format` — формат данных, `opt` — опция, указывающая на состояние файла.

Формат считываемых данных может быть специфическим, то есть `FORMAT='F10.5,2X,F10.5'` или, если формат данных не определен, используется свободный формат.

Если при чтении данных из файла векторов не существует (заранее не созданы командой `VEC/CREATE`), то они будут созданы размером как читаемый файл.

Параметр `OPT` может принимать следующие значения:

'OC' - файл открыть, прочитать и затем закрыть (определено по умолчанию),

'O' - файл открыть и затем прочитать (остается открытым для следующих операций)

' ' файл прочитать (он уже открыт, он остается открытым)

'C' файл прочитать и затем закрыть (файл должен уже быть открытым).

В ходе работы часто возникает необходимость сохранять содержимое векторов. Для этих целей используется команда

`VECTOR/WRITE vlist fname [format chopt ]`, которая записывает информацию, указанную в векторе(ax) `vlist` в создаваемый `fname` текстовый файл. Если имя файла не определено, то содержимое записывается (выводится) на терминал. `format` – это формат записываемых данных, который определяется также как и для команды `vec/read`, `chopt` – это аналог параметра `opt` команды `vec/read`, где все то же самое, только вместо «прочитать» нужно подставить «записать».

Для отображения векторов на экране используется команда

`VECTOR/DRAW vname [ id chopt ]`

При этом рисование вектора `vname` интерпретируется как создание гистограммы. При этом дополнительно содержимое вектора сохраняется в гистограмме с идентификатором `ID`. С помощью параметра `chopt` можно указать различные способы рисования. Укажем только некоторые из них:

С — нарисовать кривую через точки, L — соединить бины с помощью линий, Р — нарисовать маркер в каждом бине, \* — нарисовать звездочку в каждом бине. Стоит также отметить, что данные параметры можно использовать вместе, указывая их без пробела.

Полный список этих параметров можно найти в руководстве пользователя или с помощью команды help.

Для рисования векторов может также использоваться команда

VECTOR/PLOT vname [ id chopt ].

Основное отличие данной команды от команды vector/draw состоит в том, что каждый элемент вектора vname заполняется в гистограмму, которая автоматически создается с размерностью в 100 каналов и затем рисуется. Если vname имеет вид vname1 %vname2, то будет рисоваться распределение vname1 в зависимости от vname2.

В PAW возможно некоторые упрощения при работе с командами. Полное название команды набирать долго, даже не смотря на то, что сохраняется история команд. Поэтому возможны следующие упрощения, если команда уникальна, то можно опускать корневые пункты меню. В случае команды LEGO, можно писать 2D\_PLOT/LEGO или просто LEGO. Также возможно сокращать имена и пути команд до тех пор, пока они остаются уникальными, например VECTROR/DRAW сокращается до VE/DR.

После определение базовых команд при работе с векторами рассмотрим несколько примеров иллюстрирующих их действия.

При этом стоит обратить внимание на команды SET и OPT. Эти команды позволяют менять параметры графического представления данных. Если запустить их без опций, то будет выведен список всех переменных, которые устанавливаются с помощью этих команд.

### **Пример 1.**

```
PAW> vector/create vect(10)
# создается вектор с массивом из 10 элементов
PAW> ve/inp 1 2 3 4 5 5 4 3 2 1
# происходит заполнение вектора данными значениями
PAW> zone 2 2
# делится графическое окно на четыре зоны
PAW> set htyp 4
# изменяется тип фона гистограммы (косые линии)
PAW> ve/draw vect
# рисуется вектор в первой зоне
PAW> ve/dra vect ! *l
```

```

# рисуется вектор во второй зоне, как линия и со звездочками в
каждом бине
PAW> set hwid 6
# изменяется ширина гистограммы
PAW> set hcol 2
# меняется цвет гистограммы на красный
PAW> ve/dra vect ! bs
# во второй зоне также рисуется вектор в виде столбцов
PAW> set hcol 4
# меняется цвет на синий
PAW> set hwid
# сбрасываются настройки ширины по умолчанию
PAW> set mtyp 4
# определяется тип маркера, кружочки
PAW> ve/dra vect ! pc
# рисуется вектор с кружочками в каждом бине и проводится кривая
PAW> set hcol
# сбрасываются значения цвета
PAW> ve/plo vect
# создается гистограмма из значений вектора

```

### **Лекция 3. Пакет HBOOK. Работа с гистограммами**

Как уже отмечалось, основными инструментами в PAW являются вектора, гистограммы и ntuple. В прошлой лекции рассматривались вектора, теперь рассмотрим основные идеи работы с гистограммами. Прежде всего, хотелось бы отметить, что для работы с гистограммами используется пакет HBOOK.

HBOOK – это пакет подпрограмм для обработки статистических распределений (гистограмм и Ntuples) в ФОРТРАНе. Данный пакет позволяет представлять полученные результаты графически, выводя информацию либо на принтер, либо на монитор с помощью пакета HPLOT. PAW, как уже говорилось, объединяет функциональные возможности данных пакетов HBOOK и HPLOT в интерактивную пользовательскую среду и предоставляет пользователю логическое рабочее окружение, где возможно чтение DST данных, их анализ и подготовка конечного результата данных. Данные пакеты доступны в CERN библиотеках.

В PAW, таким образом, возможно, создавать, заполнять и работать с гистограммами интерактивно, что с одной стороны удобно, но с другой стороны не очень практично, если

вы собираетесь заниматься с большим объемом данных. Для этих целей лучше создавать отдельные программы, где возможна различная работа с данными, запись их в гистограммы и сохранение их для последующей работы в формате HBOOK. Поэтому, прежде чем перейти к работе с гистограммами в PAW, кратко познакомимся с пакетом HBOOK.

Пакет HBOOK состоит из нескольких сотен подпрограмм написанных на ФОРТРАНе, которые дают возможность пользователю символически определять, заполнять и выводить одно- или двумерные плотности распределений в виде гистограмм, графиков, таблиц или обрабатывать их в виде Ntuples.

Пакет HBOOK состоит приблизительно из 120 подпрограмм, которые непосредственно доступны для пользовательской программы, через запросы типа:

```
CALL H. . . . . (P1, P2, . . . )
```

Стоит заметить, что это единственный интерфейс между пользовательской программой написанной на ФОРТРАНе и динамической структурой данных, управляемой HBOOK, которая может оставаться скрытой от пользователя.

Еще раз повторим, что основными элементами данных HBOOK являются гистограммы (одно — или двумерные) и Ntuple. Пользователь идентифицирует эти элементы данных, используя идентификатор в виде целого числа, который не должен быть равен нулю.

Пакет организован как часть библиотеки, из которой загружаются соответствующие внешние файлы. Таким образом, только те подпрограммы, которые фактически используются, будут загружены, что позволяет минимизировать используемую память. К сожалению, некоторые неиспользованные подпрограммы обычно присутствуют в программе. HBOOK использует пакет ZEBRA для управления структуры данных, чтобы управлять своей памятью. Рабочее пространство HBOOK — это массив, размещенный в общем заголовочном блоке /PAWC/. Фактическая длина общего блока определяется через оператор PARAMETER, как показано ниже:

```
PARAMETER (NWPASC = 50000)  
COMMON/PAWC/HMEMOR(NWPASC)
```

Кроме того, HBOOK должен информировать о пределе на память через вызов HLIMIT. Это соответствует

```
CALL HLIMIT(NWPASC)
```

Таким образом, при написании программы на ФОРТРАНе с использованием пакета HBOOK, в ней обязательно должны быть перечисленные выше обращения. Структура программы может выглядеть следующим образом:

```
PROGRAM HTEST
```

```

PARAMETER (NPPAWC=20000)
COMMON/PAWC/H(NPPAWC)
CALL HLIMIT(NPPAWC)
* инициализация гистограмм
. . .
* основное тело программы
. . .
* заполнение гистограмм
. . .
RETURN
END

```

Перед тем как заполнить гистограммы их необходимо объявить. Это выполняется с помощью вызова

```
CALL HBOOK1 (ID,CHTITL,NX,XMI,XMA,VMX)
```

где ID — это идентификатор гистограммы, CHTITL — имя гистограммы, NX — число каналов, XMI и XMA — соответственно нижний и верхний предел. Так же можно объявить о создании двумерной гистограммы

```
CALL HBOOK2 (ID,CHTITL,NX,XMI,XMA,NY,YMI,YMA,VMX)
```

где все аналогично, как и для одномерной гистограммы, только добавляются дополнительные параметры для второго канала.

Заполнение гистограмм происходит с помощью

```
CALL HFILL (ID,X,Y,WEIGHT)
```

где ID — это идентификатор заполняемой гистограммы, X и Y — заполняемые значения и WEIGHT — вес данного значения.

Аналогом данной команды являются:

```
CALL HF1 (ID,X,WEIGHT)
```

для одномерного случая и

```
CALL HF2 (ID,X,Y,WEIGHT)
```

для двумерного случая.

Для того чтобы сохранить созданные гистограммы необходимо в конце программы выполнить несколько команд.

С помощью команды

```
CALL HROPEN (LUN,CHTOP,CHFILE,CHOPT,LREC,ISTAT)
```

открывается доступ к HBOOK файлу на диске, где LUN — идентификатор связанный с файлом, CHTOP — имя директории связанной с идентификатором LUN, CHFILE — имя файла, CHOPT — параметры файла (например, N — создать новый файл), LREC — длина

файла в машинных словах (рекомендуется — 1024), ISTAT — код выполнения (если ISTAT равен нулю, то команда выполнена корректно).

Запись гистограмм выполняется командой

```
CALL HROUT (ID, ICYCLE*, CHOPT)
```

где ID — идентификатор записываемой гистограммы. Если необходимо записать все гистограммы, то ID полагают равным нулю.

Команда

```
CALL HREND (CHTOP)
```

закрывает директорию CHTOP, в которую производилась запись. Таким образом, если мы хотим записать гистограммы с идентификаторами 10 и 20 в файл work.hbook, то это будет выглядеть так:

```
CALL HROPEN(2, 'lun1', 'work.hbook', 'N', 1024, ISTAT)
CALL HROUT(0, ICYCLE, ' ')
CALL HREND('lun1')
CLOSE(2)
```

После того как мы научились создавать и сохранять гистограммы в фоновом режиме, перейдем в интерактивную среду (PAW), для работы с этими гистограммами. Для того чтобы посмотреть содержимое созданного файла в фоновом режиме, его, прежде всего, нужно загрузить. Для этого в PAW используется команда:

```
HISTOGRAM/FILE ID fname [lrecl chopt]
```

ID — идентификатор присваиваемый открываемому файлу, fname — имя открываемого файла, lrecl — длина файла и chopt — соответственно дополнительные опции. Обычно, lrecl полагают равным 0, это означает, что система сама определит правильную длину существующего файла. Так как в PAW можно сокращать названия команд до уникального набора символов, то данная команда может быть сокращена до hi/file. Таким образом, в PAW это будет выглядеть следующим образом:

```
PAW> hi/file 1 work.hbook
```

После работы с данными файла work.hbook, его можно закрыть

```
PAW> Close 1
```

С помощью команды HISTOGRAM/LIST можно посмотреть содержимое файла

```
PAW>hi/list
```

На экран выводится список:

- директорий, на который разбит этот файл;
- гистограмм, где первая цифра — это идентификатор гистограмм, вторая — размерность, третья — имя гистограммы.

Если в RAW открыть несколько файлов, то для того чтобы перемещаться из одного файла в другой используется команда

```
cd //lunID
```

где ID – это идентификатор, присваиваемый открываемому файлу.

Например,

```
hi/file 10 histos.hbook
```

```
hi/file 20 histos.hbook
```

```
cd //lun10
```

```
hi/list
```

```
cd //lun10
```

Для рисования гистограмм используется команда

```
HISTOGRAM/PLOT [ id chopt ]
```

где id – идентификатор гистограммы, которую нужно построить и chopt – параметры изображения гистограмм. Приведем, некоторые из них:

C — нарисовать кривую через бины

L — соединить бины с помощью линий

P — вывести маркеры в каждом бине

\* — вывести звездочку в каждом бине

S — наложить гистограмму поверх уже нарисованной

E — вывести гистограмму с ошибками и текущим маркером

HIST — вывести только контур гистограммы без ошибок.

Стоит также отметить, что данные параметры можно использовать вместе, указывая их без пробела.

Полный список этих параметров можно найти в руководстве пользователя или с помощью команды help.

Существует также возможность построить гистограмму в определенном диапазоне, например, либо hi/pl id(ic1:ic2), где ic1 и ic2 — начальные и конечные номера каналов (при этом они должны быть целыми), либо hi/pl id(x1:x2), где x1 и x2 — значения гистограммы.

Иногда возникает необходимость продублировать существующую гистограмму, для этой цели можно воспользоваться командой

```
HISTOGRAM/COPY id1 id2 [ title ]
```

которая копирует гистограмму с идентификатором id1 в гистограмму id2, при этом второй гистограмме можно присвоить новое имя title.

При работе с гистограммами возможно несколько арифметических действий, их можно складывать, вычитать, умножать и делить. Рассмотрим по порядку эти действия. Сложения двух гистограмм обеспечивается командой:

```
HISTOGRAM/OPERATIONS/ADD id1 id2 id3 [c1 c2]
```

В результате сложения гистограмм `id1` и `id2` с весами `c1` и `c2` соответственно, образуется новая гистограмма с идентификатором `id3`.

Синтаксис остальных команд аналогичен команде сложения.

Вычитание

```
HISTOGRAM/OPERATIONS/SUBTRACT id1 id2 id3 [ c1 c2 ]
```

умножение

```
HISTOGRAM/OPERATIONS/MULTIPLY id1 id2 id3 [ c1 c2 ]
```

и деление

```
HISTOGRAM/OPERATIONS/DIVIDE id1 id2 id3 [ c1 c2 ]
```

#### **Лекция 4. Работа с Ntuple**

Чтобы понять, для чего нужны Ntuples, рассмотрим следующей пример. Пусть имеется набор данных в формате DST, в которых содержится информация о 10000 событий.

В каждом событии содержится информация о нескольких переменных, (например, в количестве 200), для которых хотелось бы построить некоторые распределения согласно различным критериям отбора. Одна из возможностей состоит в том, чтобы создать и заполнить 200 гистограмм пособытийно, согласно критериям отбора. Если же условия отбора изменятся, придется создавать новые гистограммы.

Альтернативным решением является создание одного Ntuple. Ntuple можно рассматривать как обычную таблицу, где каждая строка соответствует одному событию, а каждый столбец соответствует конкретной переменной. Таким образом, информация о 200 переменных будет записана в колонки, а о каждом событии — в ряд.

Таким образом, единожды сохранив события в данном формате, мы упрощаем процедуру анализа данных. Теперь становится возможным интерактивный анализ данных в RAW, при этом можно накладывать, также интерактивно, различные условия для отбора событий. Обычно Ntuple создаются с помощью внешних программ (в стандартной документации к HBOOK очень подробно описывается, как это делается). Поэтому опустим способ создания файла в формате Ntuple, а перейдем к анализу имеющейся в нем информации.

Для того чтобы открыть файл в формате Ntuple используется стандартная команда HISTOGRAM/FILE, о которой было рассказано в прошлой лекции. Например,

```
PAW > hi/file 1 evgen.1.nt 0
```

открывает файл evgen.1.nt.

Используя команду HISTOGRAM/LIST, можно просмотреть какие директории содержит данный файл.

```
PAW > hi/list
==> Directory :
      999 (N)    HEPEVNT
      998 (N)    HEPinput
PAW >
```

Команда NTUPLE/PRINT idn выводит на монитор суммарную информацию об Ntuple с идентификатором idn.

```
PAW > nt/print 999
```

```
*****
* Ntuple ID = 999  Entries = 600    HEPEVNT
*****
* Var numb * Type * Packing *   Range   *   Block *   Name   *
*****
*      1 * I*4 * 16 * [-1,32000] * PARTICLE * itrac
*      2 * I*4 * 5 * [-1,11] * PARTICLE * istat
*      3 * I*4 * 21 * [-1000000,10] * PARTICLE * ipdg
*      4 * I*4 * 16 * [-1,32000] * PARTICLE * moth1
*      5 * I*4 * 16 * [-32000,1] * PARTICLE * moth2
*      6 * I*4 * 16 * [-1,32000] * PARTICLE * idau1
*      7 * I*4 * 16 * [-1,32000] * PARTICLE * idau2
*      8 * R*4 * * * * * PARTICLE * Pxyz(3)
*      9 * R*4 * * * * * PARTICLE * ener
*     10 * R*4 * 16 * [-1.,10.] * PARTICLE * mass
*     11 * R*4 * 16 * [-0.001,0.00] * PARTICLE * Vxyz(3)
*     12 * R*4 * 16 * [0.,0.001] * PARTICLE * Vtime
*****
* Block * Entries * Unpacked * Packed * Packing Factor *
*****
* PARTICLE * 600 * 64 * 40 * 1.600 *
* Total * --- * 64 * 40 * 1.600 *
*****
* Blocks = 1      Variables = 12      Columns = 16 *
*****
PAW >
```

С помощью команды

```
NTUPLE/SCAN idn [ uwfunc nevent ifirst option varlis ]
```

можно посмотреть подробное содержимое Ntuple с идентификатором `idn`, где отображаются численные значения параметров. Второй параметр `uwfunc` — накладываемое условие для вывода информации о количестве событий `nevent`, начиная с события `ifirst`.

Команда `SCAN` имеет несколько опций (`option`) вывода информации. По умолчанию вывод осуществляется в текстовом формате. В то время как опция «`option=S`» позволяет представить информацию о событии в графической форме в виде многомерного плота. При этом каждая переменная представлена на отдельной оси, где также указывается диапазон, в котором она меняется.

Для вывода информации из Ntuple в виде гистограмм используется команда

```
NTUPLE/PLOT idn [ uwfunc nevent ifirst nupd option idh ]
```

где параметры совпадают с параметрами для команды `NTUPLE/SCAN`.

При этом возможно построение как одномерной гистограммы

```
PAW> nt/plot 999.pxyz(1)
```

так и

```
PAW> nt/plot 999.pxyz(1)% pxyz(2)
```

двухмерной гистограммы.

В процессе построения гистограмм можно накладывать различные условия на параметры событий. Приведем несколько примеров иллюстрирующих это.

```
1) PAW> nt/plot 999.pxyz(1) ipdg=211
```

будет построена гистограмма значений `pxyz(1)` при условии, что параметр `ipdg=211`

```
2) PAW> nt/plot 999.pxyz(1) pxyz(1)<1.and.ipdg=211
```

будет построена гистограмма значений `pxyz(1)` при нескольких условиях, — необходимо чтобы параметр `ipdg` был равен 211 **и** при этом `pxyz(1)<1`. Если вместо оператора «`and`» подставить оператор «`or`», то условие отбора изменится и гистограмма будет строиться, если параметр `ipdg=211` **или** параметр `pxyz(1)<1`.

```
3) PAW > nt/plot 999.sqrt(pxyz(1)**2+pxyz(2)**2)
```

```
ipdg=211.and.sqrt(pxyz(1)**2+ pxyz(2)**2)<0.6
```

можно видеть, что при построении гистограмм используются математические выражения, как для условий отбора, так и для выводимых значений.

```
4) PAW> nt/plot 999.pxyz(1) pxyz(1)<1.and.ipdg=211 option=b
```

Возможно также накладывать опции рисования.

Приведем некоторые параметры (`option`) изображения гистограмм:

`C` — провести гладкую кривую через бины

`L` — соединить бины с помощью линий

P — рисовать маркер в каждом бине  
 \* — нарисовать звездочку в каждом бине  
 S — наложить гистограмму поверх уже нарисованной  
 E — нарисовать гистограмму с ошибками и текущим маркером  
 B — вывести гистограмму в виде столбцов  
 A — нарисовать гистограмму без подписей к осям.

Как можно заметить, что накладываемые ограничения бывают иногда громоздкими, что затрудняет работу. Поэтому в PAW предусмотрена команда

```
NTUPLE/CUTS cutid [ option fname wkid ],
```

позволяющая приписывать параметру cutid ограничения fname, которые в последствии могут быть неоднократно использованы при работе с данными. Параметр cutid имеет следующий формат \$nn, где nn — целые числа в интервале от 1 до 99.

Приведем пример:

```
PAW> nt/cut $5 ipdg=211
```

параметру \$5 присваивается условие, что переменная ipdg равна 211.

Таким образом, при

```
PAW> nt/plot 999.pxyz(1) $5
```

будет строиться гистограмма значений pxyz(1) при условии определенном параметром \$5, что идентично

```
PAW> nt/plot 999.pxyz(1) ipdg=211
```

В данной команде имеется несколько полезных опций (option). Например, опция «P»

```
PAW > cut $5 P
```

```
$5 = ipdg=211
```

позволяет вывести на экран определение ограничения. Опция «-»

```
PAW> nt/cut $5 -
```

приводит к сбросу определений параметра \$5.

Стоит заметить, что параметр \$0 означает все определенные параметры.

```
PAW > cut $1 pxyz(1)<1
```

```
PAW > cut $0 P
```

```
$1 = pxyz(1)<1
```

```
$5 = ipdg=211
```

При работе с данными в формате Ntuple возникает необходимость сохранять полученные распределения (гистограммы). Для этого в PAW сначала необходимо объявить гистограмму. В случае создания одномерной гистограммы используется команда

```
HISTOGRAM/CREATE/1DHISTO id title ncx xmin xmax [ valmax ]
```

где `id` — идентификатор гистограммы, `title` — название гистограммы, `ncx` — количество бинов, `xmin` — нижний предел, `xmax` — верхний предел. Для добавления информации в гистограмму из `Ntuple` применяется команда `NTUPLE/PLOT`, при этом указывается параметр `idh`, означающий идентификатор гистограммы, которая будет заполняться.

Полученное распределение в гистограмме можно профитировать какой-либо теоретической зависимостью. Для этого применяется команда

```
HISTOGRAM/FIT id func [ chopt np par step pmin pmax errpar ]
```

где `id` — идентификатор фитируемой гистограммы, `func` — функция, с помощью которой происходит фитирование, `chopt` — опции фитирования, `np` — количество параметров фитирования, `par` — начальные параметры фитирования (задаются в виде вектора), `pmin` — вектор с минимально допустимыми значениями, `pmax` — вектор с максимально допустимыми значениями и `errpar` — вектор для ошибок фитирования.

Функцию, с помощью которой происходит фитирование, можно задать изначально. Для этого нужно создать внешний файл, в котором будем описана данная функция. При этом нужно обратить внимание на то, что имя созданного внешнего файла и имя фитирующей функции должны быть идентичны. Также можно использовать предопределенные функции `PAW` для фитирования:

1)  $\text{par}(1) \cdot \exp(-0.5 \cdot ((x - \text{par}(2)) / \text{par}(3))^2)$  — распределение Гаусса, обозначается кратко буквой «G»

2)  $\exp(\text{par}(1) + \text{par}(2) \cdot x)$  — экспоненциальное распределение, «E»

3)  $\text{par}(1) + \text{par}(2) \cdot x + \text{par}(3) \cdot x^2 + \dots + \text{par}(n+1) \cdot x^n$  — полином степени  $n$ , «Pn» (например, P3 – полином 3 степени).

С помощью некоторых параметров опций фитирования (`chopt`) можно регулировать вывод информации, напечатанной в командной строке в результате фитирования, опция «Q» — подавляет вывод информации на печать, в то время как опция «V» выводит подробную информацию.

Для отображения информации о статистике на рисунке используется одна из опций команды `GRAPHICS/OPTION`, сокращено будет выглядеть так: `OPT STAT`. Для того чтобы отобразить информацию о фитировании — `OPT FIT`.

Приведем пример фитирования. Для начала создадим одномерную гистограмму

```
PAW> 1d 212 'Pz distribution of pions' 50 -1. 1
```

с идентификатором 212, имеющую 50 бинов, в диапазоне от -1 до 1. Далее заполним её данными из `Ntuple`

```
PAW> nt/plot 999.pxyz(1) ipdg=211 idh=212
```

Стоит отметить, что вместо `idh=212` можно писать просто `-212`. Нарисуем полученную гистограмму

```
PAW> hi/plot 212
```

Профитируем получившиеся распределения с помощью встроенной функцией Гаусса

```
PAW> hi/fit 212 G
```

Также можно задать предел фитирования

```
PAW> hi/fit 212(15:35) G
```

В PAW существует возможность сохранять полученные рисунки в формате PostScript. Для этого используется команда `GRAPHICS/METAFILE`. Для примера приведем 4 строчки, которые позволяют сохранить рисунок в файле с именем `pic.ps`.

```
PAW>fortran/file 66 pic.ps
```

открываем файл `pic.ps` с номером 66

```
PAW>graphics/meta 66 -111
```

инициализируем файл `pic.ps` в формате A4

```
PAW> hi/plot 212
```

рисуем гистограмму 212

```
PAW>fortran/close 66
```

закрываем файл `pic.ps`

При работе (анализе) с большим объемом информации лучше использовать скрипты (имеющие расширение `kumac`), т.е. файлы с определенной последовательностью команд PAW. Данные файлы запускаются в PAW командой `EXE`, например

```
PAW>exe work.kumac
```

Пример скрипта `work.kumac`:

```
fortran/file 66 pic.ps
graphics/meta 66 -111
hi/file 1 evgen.1.nt 0
ld 212 'Pz distribution of pions' 50 -1. 1.
nt/plot 999.pxyz(1) ipdg=211 idh=212
hi/plot 212
fortran/close 66
```

## Лекция 5. Начало работы с ROOT

Рассмотренный в предыдущих лекциях пакет программ PAW реализован на языке программирования Fortran. PAW хорошо зарекомендовал себя в задачах, типичных для обработки данных ускорительного эксперимента, таких, например, как работа с

гистограммами. Тем не менее, все возрастающая сложность экспериментальных установок и увеличение объема накапливаемых данных, требовала освоения научным сообществом новых, более эффективных инструментов обработки информации. Речь идет о языке C++, который к середине 90-х годов стал широко использоваться для создания коммерческих программных продуктов. Принципиальным отличием C++ от Fortran является поддержка объектно-ориентированного программирования (ООП).

Сотрудники Европейской организации ядерных исследований CERN Рене Бран и Фонс Ридмайкерс в 1994 г. занялись исследованием возможностей C++ применительно к задачам ядерного эксперимента. Результатом этой работы явилось создание программного пакета ROOT. К настоящему времени ROOT претерпел впечатляющую эволюцию и стал по сути дела стандартным программным обеспечением, используемым для хранения и обработки данных ускорительного эксперимента, практически вытеснив PAW.

Официальный Web-сайт проекта ROOT расположен по адресу <http://root.cern.ch>. На этом сайте содержится информация о текущем релизе.

Наилучшим справочным материалом по ROOT является блестяще написанное руководство пользователя, которое доступно по адресу <http://root.cern.ch/root/doc/RootDoc.html>.

Поскольку ROOT реализован на C++, то необходимо напомнить некоторые понятия этого языка, необходимые для практической работы.

В самом общем виде идея ООП заключается в том, чтобы моделировать окружающий мир как совокупность взаимодействующих объектов. В C++ такой подход реализуется с помощью классов.

Класс является типом данных, определяемым пользователем, и представляет собой модель реального объекта в виде данных и функций для работы с ними. Таким образом, класс — это тип данных, который объединяет переменные и функции. Функции класса называются функциями-членами или методами. Переменные класса называются данными-членами или полями. Принадлежность метода DoSomething классу MyClass обозначается так: MyClass::DoSomething.

Переменная соответствующего типа называется объектом, т.е. объект — это конкретный представитель, экземпляр данного класса.

При работе с объектами часто используются указатели. Указателем называется переменная, в которой хранится адрес памяти, по которому располагается другая переменная. Тип указателя — указатель на переменную данного типа.

Создание и определение указателя часто осуществляется с помощью операции `new`:

```
MyClass *pointer = new MyClass(...);
```

Здесь с помощью операции `new` выделяется объем динамической памяти, необходимой для размещения переменной типа `MyClass` (т.е. объекта) и адрес этого участка памяти записывается в соответствующий указатель — переменную `pointer`.

Обращение к методам класса через указатель производится с помощью операции `"->"`. Предположим, класс `MyClass` имеет метод `DoSomething(...)` (троеточие обозначает возможные параметры и опции этого метода). Тогда обращение к этому методу с помощью указателя `pointer` выглядит следующим образом:

```
pointer->DoSomething(...);
```

В случае работы непосредственно с объектом класса `MyClass`, обращение к методам производится через `"."`:

```
MyClass cl;  
  
cl.DoSomething(...);
```

Перейдем теперь собственно к ROOT. Полное название этого программного пакета “An Object Oriented Data Analysis Framework” — объектно-ориентированная среда для анализа данных. ROOT реализован как набор библиотек классов, обеспечивающих необходимую функциональность: гистограммы, функции, графики, деревья и т. д. В рамках настоящего курса будут рассмотрены классы ROOT, обеспечивающие работу с гистограммами, функциями, файлами, графиками, деревьями. Описание классов ROOT для различных версий пакета находится по адресу <http://root.cern.ch/root/Reference.html>.

Кроме того, в состав ROOT входит интерпретатор CINT, который воспринимает команды ROOT и выражения C/C++.

Существует два варианта работы с ROOT. Первый состоит в том, чтобы работать с программой `root.exe` в сессионном режиме. Второй вариант заключается во включении библиотек классов ROOT в собственные программы.

ROOT имеет принятые соглашения по наименованию различных типов переменных:

имена классов начинаются с <b>T</b>	TF1, TFile
переменные типа «не класс» заканчиваются на <b>_t</b>	Int_t
поля начинаются с <b>f</b>	fIntegral
методы начинаются с прописной	Fill(), Draw()
константы начинаются с <b>k</b>	kRed

глобальные переменные начинаются с **g**

**gStyle**

В ROOT используются predefined машинно-независимые типы переменных. К наиболее употребляемым типам относятся:

Char_t	знаковый символьный	1 байт
Int_t	знаковый целый	4 байта
Float_t	вещественный	4 байта
Double_t	вещественный	8 байт
Bool_t	логический (0 ложь, 1 истина)	

Для начала работы в ROOT наберите в командной строке

```
$ root
```

Эта команда запустит интерактивную сессию работы с ROOT (при условии, что программа установлена и правильно настроены соответствующие переменные окружения).

Для выхода из программы следует набрать

```
root[] .q
```

(далее root[] будет означать, что команды вводятся в командной строке ROOT).

Как уже обсуждалось, ROOT имеет встроенный командный интерпретатор CINT. CINT воспринимает три типа команд.

1. Собственно команды CINT начинаются с ". "

.?	вывести список всех команд
.L <filename>	загрузить файл filename
.x <filename>	загрузить и выполнить файл filename

2. C/C++ выражения в соответствии с синтаксисом языка

```
Int_t a = 8
a++
Int_t b=a*2
```

3. Команды SHELL начинаются с ". !"

```
.! pwd
```

Подобно командному интерпретатору ОС Linux, оболочка ROOT имеет встроенную историю команд, для навигации по которой следует использовать клавиши «вверх» и «вниз».

Приведем пример простейшей сессии работы в ROOT: создания гистограммы, заполнения ее гауссовым распределением и рисования гистограммы.

Сначала создадим объект класса TH1F и указатель h1 на этот объект:

```
root[] TH1F *h1 = new TH1F("h1", "Random gauss",100,-2,2)
```

Далее с помощью метода FillRandom заполним гистограмму значениями, распределенными по гауссу:

```
root[] h1->FillRandom("gaus", 1000)
```

Нарисуем гистограмму:

```
root[] h1->Draw()
```

Обратите внимание на то, что интерпретатор CINT позволяет опускать точки с запятой после команд.

Очевидно, работа в интерактивной сессии может быть полезна для небольших задач, поскольку каждый раз необходимые команды приходится набирать вручную (или отыскивать в истории команд). Для большей функциональности необходимо иметь возможность писать скрипты, т.е. последовательности команд, сохраненные в файле. В ROOT различается два типа скриптов: именованные и неименованные.

Неименованные скрипты представляют собой простую последовательность команд для CINT, заключенную в фигурные скобки. Пусть файл MyMacro.C содержит следующий код:

```
{  
    #include <iostream>  
    using namespace std;  
    for (Int_t i=0; i<10; i++) {  
        cout<<i<<endl;  
    }  
}
```

Запуск такого скрипта в интерактивной сессии осуществляется командой .x:

```
root[] .x MyMacro.C
```

По умолчанию ROOT будет запускать скрипт из текущей директории или из директории \$ROOTSYS/macros. В командной строке ROOT можно задать полное имя скрипта, который следует выполнить, например

```
root[] .x /home/user2/rootscripts/MyMacro.C
```

Теперь создадим скрипт, содержащий определение функции:

```
void drawhist() {  
    TH1F *h1 = new TH1F("h1", "simplest histo", 10, 0, 10);  
    h1->Fill(3, 3);  
    h1->Fill(4, 2);  
    h1->Fill(5, 1);  
    h1->Draw();  
}
```

Такой скрипт является именованным. Прежде чем выполнить функцию `drawhist()`, нужно загрузить скрипт в память:

```
root[] .L MyNamedMacro.C  
root[] main()
```

Обратите внимание, что в отличие от работы в режиме интерактивной сессии, в скриптах требуется ставить точки с запятой после каждой команды, в соответствии с синтаксисом C/C++.

Команды ROOT, которые будут приводиться далее в курсе, можно использовать как в режиме интерактивной сессии, так и в скриптах.

В заключение следует упомянуть о возможности преобразования данных HBOOK/PAW в ROOT-файлы. Для этого в составе ROOT существует специальная утилита `h2root`. Эта утилита автоматически преобразует гистограммы, `ntuples`, содержащиеся в HBOOK-файле, в соответствующие объекты ROOT и записывает их в новый файл.

Синтаксис утилиты `h2root` (в консольной строке ОС Linux)

```
$ h2root <hbook file> <root file>
```

где `hbook file` означает имя файла HBOOK, а `root file` имя выходного ROOT-файла.

## Лекция 6. Гистограммы (I)

Гистограммы в ROOT реализованы как иерархия классов, наследующих корневому классу TH1. ROOT поддерживает одномерные, а также двумерные и трехмерные гистограммы.

Начнем обсуждение с одномерных гистограмм. Наиболее употребляемым на практике классом является TH1F, это класс одномерных гистограмм, на содержимое бина (ячейки) которых отводится 4 байта (максимальная точность 7 знаков).

Общий синтаксис создания гистограммы

```
TH1F *h1 = new TH1F("HistName","Histogram title",Nbins,xmin,xmax);
```

Конструктору гистограммы передается 5 параметров.

1. HistName — имя гистограммы, оно не должно содержать пробелов.

2. Histogram title — заголовок гистограммы.

3. Nbins — целое число бинов в гистограмме.

4 и 5. xmin и xmax — левая и правая граница изменения величины по оси X.

Для внесения значений в гистограмму используется метод Fill(value), где value — значение, которое требуется занести. Этот метод увеличивает содержимое бина, к которому принадлежит значение value, на единицу. Иногда требуется внести значение с некоторым весом, это делается с помощью метода Fill(value, w), где w — вес. Вес может быть отрицательной величиной.

В качестве примера использования класса TH1F создадим одномерную гистограмму, имеющую диапазон по X от 0 до 10, разбитый на 20 бинов.

```
TH1F *h1 = new TH1F("h1", "My histo", 20,0,10);
```

Обратите внимание, что очень часто имя гистограммы совпадает с именем указателя, но делается это исключительно для удобства работы и не является необходимым требованием. Заголовок гистограммы обычно содержит более развернутую информацию.

Заполним гистограмму значениями, последовательно применяя метод Fill:

```
h1->Fill(5);
```

```
h1->Fill(2);
```

```
h1->Fill(5.6);
```

```
h1->Fill(7);
```

```
h1->Fill(6.1);
```

Для рисования гистограммы используется метод Draw():

```
h1->Draw();
```

В методе Draw() можно задавать различные опции рисования гистограмм. Например, если выполнить

```
h1->Draw("C")
```

то через точки, соответствующие значениям бинов будет проведена кривая. Опция "P" — нарисовать маркеры в каждом бине, "E" — нарисовать погрешности значений бинов. Опции можно совмещать, безо всяких пробелов, например,

```
h1->Draw("CP")
```

Предположим, мы создали некоторую другую гистограмму, указатель на которую h2, и хотим построить эту гистограмму в том же окне. Это часто бывает необходимо, чтобы визуально сравнить две гистограммы (например, сигнал и фон). Для этого в методе Draw нужно указать параметр "same".

```
h2->Draw("same")
```

Поле, на которое выводятся графические объекты в ROOT, называется canvas (определяется классом TCanvas).

Объект, например гистограмма, рисуется на текущем активном canvas. Если canvas не существует, то он создается автоматически и имеет по умолчанию имя c1.

Чтобы разделить canvas на несколько частей, можно воспользоваться методом Divide(k,l), где k и l, число разбиений по горизонтали и вертикали соответственно. Чтобы выбрать, на какой части canvas'а следует рисовать объект, можно воспользоваться методом cd(n), где n — номер части. Разделы нумеруются слева направо, сверху вниз.

Пример

```
TCanvas *MyC = new TCanvas ("MyC", "Test canvas", 1);
```

```
MyC->Divide(2,2)
```

```
MyC->cd(1)
```

```
h1->Draw()
```

Чтобы сохранить текущее изображение на canvas на диск, следует выбрать указателем мышки File menu/Save As и далее указать необходимый формат файла.

Сложение, деление и умножение гистограмм. Методы Add(), Divide() и Multiply() позволяют складывать, делить и умножать гистограммы.

Пусть у нас есть две гистограммы, указатели на которых h1 и h2.

Чтобы добавить к h1 гистограмму h2:

```
h1->Add(h2)
```

Можно добавить h2 к h1 с некоторым весом w:

```
h1->Add(h2,w)
```

Вычесть h2 из h1:

```
h1->Add(h2,-1)
```

Важно понимать, что при этом происходит побиновое сложение, следовательно, складывать (равно как умножать и делить) можно только гистограммы с одинаковым числом бинов.

Аналогично осуществляется побиновое деление и умножение гистограмм:

```
h1->Divide(h2)
```

```
h1->Multiply(h2)
```

ROOT позволяет также умножать и делить гистограммы на функцию. Для этого следует задать необходимую функцию. Работу с функциями одного аргумента в ROOT обеспечивает класс TF1. Создание функции происходит следующим образом:

```
root[] TF1 *f1 = new TF1("f1","formula",xmin,xmax);
```

Первый параметр задает имя функции. Параметр formula собственно задает функциональную зависимость, например, это может быть  $x*x*\exp(-x)$ ,  $\sin(x)$ , и т. д. Третий и четвертый параметр задают область определения функции.

Чтобы умножить или поделить гистограмму на функцию:

```
h1->Multiply(f1) или h1->Divide(f1)
```

**Прочие аспекты работы с гистограммами.** Чтобы создать идентичную копию какой-либо гистограммы (клонировать гистограмму), следует использовать метод Clone():

```
TH1F *h1_clone = (TH1F*)h1->Clone()
```

Однако мы получим две гистограммы с одинаковыми именами. Чтобы переименовать гистограмму, нужно сделать следующее

```
h1_clone->SetName("h1_clone")
```

Чтобы получить интегральное значение содержимого гистограммы

```
h1->Integral()
```

Чтобы нормировать гистограмму на величину norm.

```
Double_t scale = norm/h1->Integral()
```

```
h1->Scale(scale)
```

Чтобы сделать разбиение гистограммы более крупным

```
h1->Rebin(N);
```

По умолчанию этот метод сольет два соседних бина в один.

Чтобы дать заголовки осям X, Y, нужно сначала «взять» соответствующую ось, а затем использовать метод SetTitle:

```
h1->GetXaxis()->SetTitle("X axis title");
```

```
h1->GetYaxis()->SetTitle("Y axis title");
```

### **Фитирование гистограмм**

Фитирование гистограмм может осуществляться двумя способами: через графический интерфейс FitPanel и в командном режиме.

Чтобы воспользоваться FitPanel в интерактивном режиме следует кликнуть правой кнопкой мыши по нарисованной гистограмме, далее выбрать в появившемся меню пункт FitPanel. В верхней части открывшейся FitPanel можно выбрать желаемую для фита функцию: «polN», где N принимает значения от 0 до 9 соответствует полиномам порядка N, «gaus», «landau» — распределению Гаусса и Ландау соответственно; «expo» задает экспоненциальное распределение, «user» — определенное пользователем.

В нижней части панели находится горизонтальный ползунок, который позволяет варьировать диапазон фитирования.

Под ползунком кнопка Fit выполняет собственно фитирование, Reset выставляет все параметры в значения по умолчанию, Close закрывает окно FitPanel.

По нажатию кнопки Fit, программа выполняет фитирование гистограммы выбранной функцией с учетом заданных пользователем параметров и диапазона.

Вывод результатов фитирования происходит в окне, в котором была запущена интерактивная сессия ROOT. Эта распечатка дает весьма подробную информацию. Рассмотрим часть вывода наиболее важную с практической точки зрения.

Результаты фитирования представлены в виде таблицы значений. В полях NO. и NAME указывается номер параметра и его наименование, если такое имеется. В полях VALUE и ERROR приводятся значения найденных параметров функции, обеспечивающих правильную «подгонку» к значениям гистограммы и погрешности найденных значений.

Приведем пример вывода результатов фитирования гистограммы гауссовым распределением, которое имеет три параметра с именами Constant, Mean и Sigma.

NO.	NAME	VALUE	ERROR
1	Constant	1.39752e+03	8.54896e+00
2	Mean	6.14381e+01	8.84769e-02
3	Sigma	1.58783e+01	1.01910e-01

Для фитирования в режиме командной строки, или в скрипте, применяется метод `TH1F::Fit(f1,xmin,xmax)`. Существует возможность фитирования функцией, встроенной в ROOT, или использования собственных функций.

Например, чтобы фитировать гистограмму функцией Гаусса:

```
h1->Fit("gaus")
```

В ROOT имеются следующие встроенные функции:

1. `gaus` Функция Гаусса:  $f(x) = p_0 \cdot \exp(-0.5 \cdot ((x-p_1)/p_2)^2)$
2. `exp` Экспонента:  $f(x) = \exp(p_0 + p_1 \cdot x)$
3. `polN` Полином степени N:  $f(x) = p_0 + p_1 \cdot x + p_2 \cdot x^2 + \dots + p_N \cdot x^N$ .
4. `landau` Функция Ландау

В этих функциях  $p_0, \dots, p_N$  — параметры, которые предполагается отыскать в процессе фитирования.

Существует также возможность фитирования функцией, которую задает сам пользователь.

Пример создания функции  $\cos(x)$ , определенной в диапазоне  $(-3,5)$ :

```
TF1 *f1 = new TF1("f1", "cos(x)", -3, 5);
```

Нарисовать график функции очень просто:

```
f1->Draw();
```

Создать функцию, которая содержала бы свободные параметры, можно так:

```
TF1 *f1 = new TF1("f1", "[0]*sin(x)*exp(-[1]*x)", 0, 10);
```

Здесь `[0]`, `[1]` являются параметрами функции.

Чтобы теперь произвести фитирование гистограммы:

```
h1->Fit(f1);
```

Чтобы занести значения результатов фита в переменные для последующего использования, необходимо использовать соответствующие методы «взятия» параметров. Например, чтобы извлечь параметр  $\chi^2$ , характеризующий качество фита

```
Double_t chi2 = f1->GetChisquare()
```

найти параметр [1] и его погрешность:

```
Double_t par1 = f1->GetParameter(1)
```

```
Double_t par1_error = f1->GetParError(1)
```

## Лекция 7. Гистограммы (II). Графики

**Сохранение гистограмм на диск.** Чтобы иметь возможность записать гистограмму в файл на диске, нужно, прежде всего, создать этот файл. В ROOT файлы представлены классом TFile. ROOT-файлы имеют расширение ".root".

Создание файла (объекта класса TFile)

```
TFile f("histos.root", "new")
```

Первым параметром конструктора является имя файла (в данном случае `histos.root`).

Второй параметр является опцией создания(открытия) файла:

<code>new</code> или <code>create</code>	создать файл; если файл с таким именем уже существует, он не будет открыт;
<code>recreate</code>	создать файл; если файл с таким именем уже существует, он будет перезаписан;
<code>update</code>	открыть файл для записи; если файла с таким именем не существует, он будет создан;
<code>read</code>	открыть файл для чтения (по умолчанию).

Созданный (или открытый) таким образом файл становится текущей ROOT-директорией. Изначально, после запуска ROOT, текущей директорией является сама сессия ROOT. Общее правило заключается в том, что последний созданный файл есть текущая директория.

В ROOT имеется глобальная переменная `gDirectory`, указывающая на текущую директорию.

Чтобы показать текущую директорию

```
gDirectory->pwd() или .pwd
```

Показать содержимое директории

```
gDirectory->ls()   или   .ls
```

Сменить директорию позволяет команда `TFile::cd()`. Закрывать файл можно командой `TFile::Close()`.

Приведем пример, поясняющий работу с текущей ROOT-директорией.

```
root [] .pwd
Current directory: Rint:/
root [] TFile f1("file1.root","recreate")
root [] .pwd
Current directory: file1.root:/
root [] TFile f2("file2.root","recreate")
root [] .pwd
Current directory: file2.root:/
root [] f1.cd()
root [] .pwd
Current directory: file1.root:/
root [] f1.Close()
root [] .pwd
Current directory: Rint:/
```

Запись гистограммы (равно как и других объектов), осуществляется методом `TH1F::Write()`.

Пример сохранения гистограммы в файл `histos.root`:

```
TFile f("histos.root","recreate");
TH1F *h1 = TH1F h1("hgaus","histo from a gaussian",100,-3,3);
h1->FillRandom("gaus",10000);
h1->Write();
```

Можно записать сразу все объекты в текущей директорией командой `TFile::Write()`

```
f->Write();
```

Чтение объекта производится методом `TFile::Get("name")` этот метод возвращает указатель на объект с именем `name`.

Пример. Чтобы прочитать гистограмму, сохраненную ранее в файле `histos.root`:

```
TFile f("histos.root");
TH1F *h = (TH1F*)f.Get("hgaus");
```

Обратите внимание, что необходим `cast` к нужному типу указателя. Далее можно работать с гистограммой `hgaus` через указатель `h`.

**Двумерные гистограммы.**

Работа с двумерными гистограммами аналогична одномерному случаю. Основным классом является TH2F, отвечающей гистограмме с максимальной точностью в 7 знаков.

При создании двумерной гистограммы, в отличие от одномерной, необходимо указать число бинов как по оси X, так и по оси Y, а также соответствующие диапазоны изменения величин:

```
TH2F *h2 = new TH2F("h2","h2 title",NbinsX,xmin,xmax,NbinsY,ymin,ymax);
```

Методу Fill() также следует передавать два значения (x, y) и, опционально, вес:

```
h2->Fill(1.2,3.7)
```

```
h2->Fill(1.2,3.7,5)
```

Применение метода Draw() для графического отображения гистограммы ничем не отличается от одномерного случая:

```
h2->Draw()
```

По умолчанию, метод Draw() отображает двумерную гистограмму как «облако» точек пропорциональное содержимому каждой клетки в пространстве (x, y). Часто употребляемыми опциями метода Draw() являются "LEGO", "SURF1", "SURF2".

Параметр "LEGO" позволяет отобразить гистограмму в трехмерном виде, в котором содержимое каждой клетки отображается в виде «башни» соответствующей высоты.

Параметр "SURF1" отображает гистограмму в виде трехмерной «сетчатой» поверхности.

Если задать опцию "TEXT", то в каждой клетке двумерной гистограммы будет напечатано числовое значение ее содержимого.

Графики. Работа с графиками обеспечивается классом TGraph. Для создания графика нужно определить два массива, содержащих n значений абсцисс и ординат точек.

Создадим график, содержащий 20 точек, удовлетворяющих функции  $y = 10 \cdot \sin(x+0.2)$ .

```
Int_t n = 20;
Double_t x[n], y[n];
for (Int_t i=0; i<n; i++) {
    x[i] = i*0.1;
    y[i] = 10*sin(x[i]+0.2);
}
TGraph *gr1 = new TGraph (n, x, y);
```

Как гистограмма и функция, график рисуется с помощью метода Draw()

```
gr1->Draw("ACP")
```

В методе Draw( ) могут быть использованы различные опции, наиболее важные среди них:

- A     производится рисование координатных осей
- L     точки графика соединяются ломаной линией
- C     точки графика соединяются плавной кривой
- \*     каждая точка отмечается звездочкой
- P     в каждой точке отображается маркер текущего стиля.

Изменить стиль маркера можно командой gr1->SetMarkerStyle(N), где N обозначает номер стиля.

Опции независимы от регистра и могут указываться вместе без пробелов.

Пример скрипта, создающего график  $y(x)=10*\sin(x+0.2)$ , и обеспечивающего его рисование

```
{
  Int_t n = 20;
  Double_t x[n], y[n];
  // создаются массивы значений x и y
  for (Int_t i=0; i<n; i++) {
    x[i] = i*0.1;
    y[i] = 10*sin(x[i]+0.2);
  }
  // создается график
  TGraph *gr3 = new TGraph(n,x,y);
  TCanvas *c1 = new TCanvas ("c1","Graph Draw
Options",200,10,600,400);
  // задается стиль маркера и рисуется график с координатными осями,
  // точки графика соединены ломаной и каждая точка маркируется
  // в соответствии с заданным стилем
  gr3->SetMarkerStyle(21);
  gr3->Draw("APL");
}
```

Чтобы наложить два графика на одной картинке, следует в методе Draw для второго графика опустить параметр "A", т.е. не рисовать новых осей координат. Пример:

```
{
  Int_t n = 20;
  Double_t x[n], y[n], x1[n], y1[n];
  for (Int_t i=0; i<n; i++) {
```

```

    x[i] = i*0.5;
    y[i] = 5*cos(x[i]+0.2);
    x1[i] = i*0.5;
    y1[i] = 5*sin(x[i]+0.2);
}
TGraph *gr1 = new TGraph(n,x,y);
TGraph *gr2 = new TGraph(n,x1,y1);
TCanvas *c1 = new TCanvas("c1","Two Graphs",200,10,600,400);
gr1->SetLineColor(4);
gr1->Draw("AC*");
// совмещаем два графика, опустив опцию "A" у второго
gr2->SetLineWidth(3);
gr2->SetMarkerStyle(21);
gr2->SetLineColor(2);
gr2->Draw("CP");
}

```

Для создания графиков с погрешностями точек используется класс TGraphErrors.

Пример:

```

{
c1 = new TCanvas("c1","A Simple Graph with error
bars",200,10,700,500);
c1->SetFillColor(42);
c1->SetGrid();
c1->GetFrame()->SetFillColor(21);
c1->GetFrame()->SetBorderSize(12);
// создаются массивы значений
Int_t n = 10;
Float_t x[n] = {-.22,.05,.25,.35,.5,.61,.7,.85,.89,.95};
Float_t y[n] = {1,2.9,5.6,7.4,9,9.6,8.7,6.3,4.5,1};
// создаются массивы погрешностей
Float_t ex[n] = {.05,.1,.07,.07,.04,.05,.06,.07,.08,.05};
Float_t ey[n] = {.8,.7,.6,.5,.4,.4,.5,.6,.7,.8};
// создается и рисуется график
TGraphErrors *gr = new TGraphErrors(n,x,y,ex,ey);
gr->SetTitle("TGraphErrors Example");
gr->SetMarkerColor(4);
}

```

```
gr->SetMarkerStyle(21);
gr->Draw("ALP");
c1->Update();
}
```

В том случае, если погрешности точек несимметричны, используется класс `TGraphAsymmErrors` (подробнее см. Руководство пользователя)

## Лекция 8. Деревья

Наиболее важный раздел ROOT — работа с деревьями (соответствующий класс называется `TTree`). Дерево является обобщением `ntuple`, рассмотренного в разделе, посвященном работе в PAW, и дает более широкие возможности. Напомним, что `ntuple` можно представить как пронумерованную таблицу числовых значений. Каждая строка в такой таблице пронумерована и соответствует т.н. вхождению (часто каждое вхождение отвечает одному реальному или смоделированному событию). Дерево представляет собой аналогичную структуру, но принципиальное отличие от `ntuple` состоит в том, что содержимое дерева не ограничивается переменными типа `float`, а может включать другие типы данных, вплоть до классов. Столбцы таблицы, содержащие значения одного сорта, для дерева называются ветвями.

Создание дерева:

```
TTree t1("t1", "Simple Tree")
```

Для того чтобы добавить к дереву ветвь, используется метод `TTree::Branch`

```
t1.Branch("px", &px, "px/F")
```

Первый параметр метода это имя ветви. Вторым параметром задается адрес, по которому будет читаться переменная. (Напомним, что символ `&` в языке C означает операцию взятия адреса.) Третий параметр характеризует лист (значение) на дереве в формате имя/тип. Тип определяется символом. Наиболее употребительными являются `"F"` (`Float_t`), `"I"` (`Int_t`).

Таким образом, в примере выше создается ветвь с именем `"px"`, содержащая данные типа `Float_t`. Значения в эту ветвь будут заноситься из переменной `px`.

Чтобы занести значения переменных в дерево, используется метод `Fill()`.

Проиллюстрируем сказанное на примере скрипта, создающего простое дерево, и записывающего его в файл.

```
{
    //создается файл, в котором будет храниться дерево
    TFile f("tree1.root", "recreate");
    //создается собственно дерево
    TTree t1("t1", "Simple Tree");
    //создаются необходимые переменные
    Float_t px, py, pz;
    Int_t ev;
    //создаются три однотипные ветви
    t1.Branch("px", &px, "px/F");
    t1.Branch("py", &py, "py/F");
    t1.Branch("pz", &pz, "pz/F");
    // и одна ветвь, которая будет содержать целочисленные переменные
    (номер
    // события)
    t1.Branch("ev", &ev, "ev/I");
    //цикл, в котором происходит заполнение дерева
    for (Int_t i=0; i<10000; i++) {
        gRandom->Rannor(px,py); //присваивание случайных значений px и py
        pz = px*px + py*py;
        ev = i;
        t1.Fill(); //заполнение дерева
    }
    t1.Write(); //запись дерева в файл
}
```

После выполнения этого скрипта будет создан файл `tree1.root`, содержащий дерево `t1`. Это дерево будет состоять из 4 ветвей (по количеству вызовов метода `TTree::Branch`): `px`, `py`, `pz`, `ev`. Первые три ветви будут содержать нецелочисленные значения, а третья — целочисленные. Эти значения будут считываться из соответствующих одноименных переменных. Заполнение дерева происходит в цикле `for` (10000 вхождений в цикл). По аналогии с гистограммами, метод `Write()` позволяет записать объект — в нашем случае дерево — в файл.

Чтобы узнать общую информацию о дереве, можно использовать метод `Print()`

```
root [] t1.Print()
```

Этот метод выведет информацию о названии дерева, числе вхождений, размере, а также о количестве и содержимом ветвей.

Для того чтобы вывести значения, содержащиеся в конкретном вхождении, существует метод `Show(N)`, где `N` — номер интересующего нас вхождения.

Вывод этого метода для созданного дерева `t1` будет выглядеть следующим образом:

```
root [] t1->Show(151)
=====> EVENT:151
px          = -2.81273
py          = -0.944246
pz          = 8.80302
ev          = 151
```

Для просмотра содержимого дерева иногда бывает удобно воспользоваться `ROOT Object Browser`. Чтобы запустить его, следует в командной строке просто создать объект `TBrowser`:

```
root[] TBrowser b
```

Откроем в нем файл `tree1.root`. Если этот файл находится в памяти `ROOT`, то его можно найти в папке `ROOT Files`. В ином случае, можно загрузить этот файл в память, отыскав его во встроенном навигаторе по файловой системе и дважды кликнув по нему левой кнопкой мыши. После этого файл окажется в каталоге `ROOT Files`. Теперь, дважды кликнув на нем, в правой части `ROOT Object Browser` увидим содержимое этого файла.

В файле `tree1.root` содержится лишь дерево `t1`. Дважды кликнув на `t1`, увидим список ветвей. Если кликнуть дважды по значку ветви (в данном случае лист), то можно вывести содержимое этой ветви в виде гистограммы.

Рассмотрим далее процесс чтения данных из дерева.

Прежде всего, следует описать переменные, в которые будут считываться значения. Затем нужно указать дереву, что считывание будет осуществляться именно в эти переменные. Это делается с помощью метода `SetBranchAddress`. Этот метод имеет два параметра. Первым параметром является имя ветви дерева, вторым – адрес переменной, в которую необходимо осуществлять чтение.

Собственно считывание происходит с помощью команды `GetEntry(i)`, где `i` – номер вхождения. Этот метод считывает все значения, содержащиеся в `i`-ом вхождении, и присваивает их переменным, которые были указаны ранее в `SetBranchAddress`.

Чтобы узнать общее число вхождений, используется метод `GetEntries()`.

Следующий скрипт обеспечивает чтение дерева, созданного в предыдущем примере.

```

{
    //открывается файл, содержащий дерево
    TFile *f = new TFile("tree1.root");
    //создается указатель t1 на взятое из файла дерево с именем t1
    TTree *t1 = (TTree*)f->Get("t1");
    //создаются переменные, в которые будут считываться данные дерева
    Float_t px, py, pz;
    Int_t ev;
    //устанавливаются адреса переменных, в которые будет осуществляться
    //чтение ветвей дерева t1 с именами px, py, pz и ev
    t1->SetBranchAddress("px",&px);
    t1->SetBranchAddress("py",&py);
    t1->SetBranchAddress("pz",&pz);
    t1->SetBranchAddress("ev",&ev);
    //создается гистограмма
    TH2F *hpxpy = new TH2F("hpxpy","py vs px",30,-3,3,30,-3,3);
    //в переменную nentries вносится общее число вхождений в дерево t1
    Int_t nentries = (Int_t)t1->GetEntries();
    //цикл по вхождениям
    for (Int_t i=0; i<nentries; i++) {
        t1->GetEntry(i);
        hpxpy->Fill(px,py);
    }
    hpxpy->Draw();
}

```

По сравнению с предыдущим скриптом, в цикле `for` происходит чтение переменных по вызову `t1->GetEntry(i)`.

Простейший анализ содержимого дерева реализуется с помощью метода `Draw`.

`t1->Draw("px")` построит гистограмму значений `px`.

Построить двумерное распределение значений переменных можно следующим образом

`t1->Draw("px:pz")`

В методе `Draw` можно также указать определенные критерии, которым должны удовлетворять события (вхождения). Эти критериями должны являться выражениями языка C.

Например, можно указать следующую конструкцию:

`t1->Draw("pz", "px<0.5")`

в таком случае будет построена гистограмма значений  $p_z$ , для событий, удовлетворяющих поставленному условию, т.е. в которых значение  $p_x < 0.5$ .

Такая конструкция бывает очень полезна в следующем случае. Представим, что дерево заполнено событиями, и в каждом из событий выставляется значение `iscut`, отвечающее, удовлетворяет данное событие какому-либо критерию. Если событие «хорошее», то значение `iscut` есть 0, в противном случае 1. Если мы хотим построить значения каких-либо переменных только для тех событий, которые удовлетворяют этому критерию, то это можно сделать следующим образом:

```
t1->Draw("var1", "iscut==0");
```

Обратите внимание на использование `==`, поскольку производится сравнение значения `iscut` с нулем, а не операция присваивания. Использование в этом случае одного знака равенства является очень распространенной ошибкой.

Очевидно, событие может отбираться по целому ряду критериев, тогда их можно перечислить, используя операторы логических И и ИЛИ («&&» и «||» соответственно).

Например, построим гистограмму значений `var1`, для событий в которых `iscut1=0` и `iscut2=0`:

```
t1->Draw("var1", "iscut1==0 && iscut2==0");
```

В заключение необходимо пояснить, что не следует однозначно ассоциировать каждое вхождение в дерево с физическим событием. Нередко встречается другая организация дерева, в котором вхождения соответствуют, например, трекам частиц. Отметим также, что существует класс `TNtuple`, который представляет собой частный случай дерева, значения которого ограничиваются переменными типа `Float_t`, т.е. это полная аналогия `ntuple` в PAW.

## Лекция 9. Изучение генераторов физических процессов: PYTHIA

Программы PYTHIA, JETSET интенсивно используются для генерации событий в физике высоких энергий при описании процессов множественного рождения в столкновениях элементарных частиц. В частности это включает жесткие взаимодействия в столкновениях  $e^+e^-$ ,  $pp$  и  $ep$ , а также некоторые другие случаи. Программы предназначены для генерации полных событий, т.е. дают более детальную картину, чем мы наблюдаем в эксперименте, в рамках нашего понимания фундаментальной физики процессов. Обсуждаемые здесь программы Монте-Карло построены как ведомые системы, т.е. пользователь должен написать основную программу. Из нее различные программы вызываются для выполнения частных задач, после чего управление снова передается

основной программе. Некоторые из этих задач могут быть весьма тривиальными, и достаточно высокоуровневые программы могут производить большое число вызовов подпрограмм. Многие программы не предназначены для непосредственного вызова пользователем, а только через программы высокого уровня, типа LUEXEC, LUEEVT, PYINIT или PYEVT.

В основном это означает, что существует три пути, по которым вы взаимодействуете с программами. Во-первых, устанавливая переменные общих блоков, вы определяете, как программы должны выполнять частные задачи, т.е. какие процессы должны генерироваться, каковы предполагаемые массы частиц, какие константы связи должны использоваться, каковы сценарии фрагментации и так далее с сотнями опций и параметров. Во-вторых, при вызовах подпрограмм вы начинаете генерировать события по правилам, установленным выше. Обычно в подпрограмме существует несколько аргументов, которые связаны с деталями физической ситуации, например, с энергией в системе центра масс, предполагаемой в событии. В-третьих, вы также можете заглянуть в общий блок LUJETS для того, чтобы извлечь информацию о сгенерированном событии, или же вы можете вызвать различные функции и подпрограммы для дальнейшего анализа.

Работа с PYTHIA должна быть строго организована, так как необходимо вначале инициировать процедуру генерации и только затем генерировать события, и нельзя свободно изменять ключи и параметры в ходе выполнения программы. Поэтому точность полученных результатов напрямую зависит от заданной структуры программы. Таким образом работа с PYTHIA может быть разделена на три этапа.

1. Этап инициализации. Здесь определяются все основные характеристики будущего процесса генерации. В этом разделе содержится следующая информация.

Общие блоки, по меньшей мере, следующие, а возможно, и некоторые еще:

```
COMMON/LUJETS/N,K(4000,5),P(4000,5),V(4000,5)
COMMON/LUDAT1/MSTU(200),PARU(200),MSTJ(200),PARJ(200)
COMMON/PYSUBS/MSEL,MSUB(200),KFIN(2,-40:40),CKIN(200)
COMMON/PYPARS/MSTP(200),PARP(200),MSTI(200),PARI(200)
```

Выбор требуемых процессов. Некоторое количество фиксированных «меню» для подпроцессов может быть задано выбором различных значений MSEL, но при MSEL=0 это можно сделать, используя номера подпроцессов. Например, если нужно сгенерировать процессы 14, 18 и 29, это можно сделать так:

```
MSEL=0
MSUB(14)=1
MSUB(18)=1
MSUB(29)=1
```

Выбор кинематических ограничений в массиве SKIN. Например, для того, чтобы сгенерировать процесс жесткого рассеяния с  $5 \text{ ГэВ} < p_T < 10 \text{ ГэВ}$ , используются ограничения

SKIN(3)=5

SKIN(4)=10

К сожалению, излучение в начальном и конечном состоянии будет давать сдвиг кинематики жесткого рассеяния, делая эффекты от ограничений менее предсказуемыми. Следовательно, всегда нужно очень осторожно подходить к наложению ограничений, чтобы не обрезать никаких желаемых конфигураций событий.

Описание исходного физического сценария, например, для массы топ-кварка.

Выбор структурных функций, определяющие  $Q^2$  и прочие подробности процесса генерации.

Выключение частей генератора, не нужных для данного моделирования, например, фрагментации для исследований на партонном уровне.

Инициализация процедуры генерации события. Здесь устанавливается кинематика, обнаруженные максимумы дифференциальных сечений, использующиеся в последующем монте-карловском моделировании, а также выполняются другие подготовительные задачи. Инициализация выполняется PYINIT, которая должна вызываться только после того, как значения всех ключей и параметров выше устанавливаются на нужные значения. Должны быть определены система отсчета, пучки частиц и энергия взаимодействия в ГэВ.

CALL PYINIT ('CMS', 'p', 'pbar', 1800.)

Любой другой начальный материал, требующийся пользователю, например построение гистограмм.

2. Цикл генерации. Здесь генерируются и анализируются заданные события. Это включает в себя следующие действия:

Генерация следующего события при вызове

CALL PYEVNT

Печать нескольких событий для проверки того, что все работает, как запланировано, с помощью

CALL LULIST(1)

Анализ представляющих интерес свойств события, проводимый или при непосредственном считывании информации из общего блока LUJETS, или с использованием ряда сервисных подпрограмм в JETSET.

Сохранение событий на диске, или интерфейс для моделирования детектора.

3. Последний этап. Ставятся задачи.

Распечатка таблицы выведенных сечений, полученных в результате монте-карловского моделирования, с помощью команды

CALL PYSTAT(1)

Печать гистограмм и другой выходной информации, необходимой пользователю.

Полная информация о каждом новом сгенерированном событии сохраняется в общем блоке LUJETS, который, таким образом, формирует запись события. Здесь каждая струя или частица, которая появляется на некоторой стадии фрагментации или в цепочке распадов, будет занимать одну строку в массиве. Различные компоненты этой строки будут содержать информацию относительно того, является ли объект струей/частицей, историю рождения, настоящее состояние (фрагментировала/распалась или нет), данные об импульсе, энергии и массе, а также пространственно-временную позицию вершины рождения. Обратите внимание, что векторы  $K(I,3)$ - $K(I,5)$  и  $P, V$  могут иметь специальное значение для некоторых определенных приложений. Общий блок LUJETS увеличивался со временем и в настоящий момент может содержать 4000 входов. Большое количество входов обусловлено тем, что запись события содержит не только конечные частицы, но также все партоны и адроны промежуточного звена, которые впоследствии образовали ливни, фрагментировали или распались

COMMON/LUJETS/N, K(4000, 5), P(4000, 5), V(4000, 5)

N: число строк в массивах K, P, V, занятых текущим событием. N непрерывно обновляется, поскольку осуществляется определение первоначальной конфигурации и обработка фрагментации и распадов. В дальнейшем конкретное число партонов/частиц, установленное между 1 и N, обозначается I.

$K(I, 1)$ : код текущего состояния партона/частицы, сохраненного в строке. Основное правило заключается в том, что коды 1-10 соответствуют существующим в настоящее время партонам/частицам, в то время как большие значения кодов отвечают партонам/частицам, которые уже не существуют (распались или фрагментировали) или несут другой вид информации о событии.

= 0: свободная строка.

= 1: нераспавшаяся или нефрагментировавшая струя, во втором случае струя может быть единственной или последней струей в системе струй.

= 2: нефрагментировавшая струя, которая сопровождается несколькими струями в одной и той же синглетной по цвету системе.

= 3: нефрагментировавшая струя со специальной информацией о цветовом потоке, сохраненной в  $K(I, 4)$  и  $K(I, 5)$ , так что соседние по струне партоны могут следовать не друг за другом, а в свободном порядке в записи события.

= 4: частица, которая может распасться, но не в пределах разрешенного объема вокруг первичной вершины.

= 5: частица, которая должна распасться при следующем вызове LUEXES при данном расположении вершины (этот код устанавливается только пользователем).

= 11: распавшаяся частица или фрагментировавшая струя, вторая может быть единственной струей или последней из системы струй.

= 12: фрагментировавшая струя, которая сопровождается несколькими струями той же синглетной по цвету системы.

= 13: струя, которая была уничтожена в записи, когда использовалась специальная информация о цветовом потоке при перестройке системы струй.

= 14: партон, который совершил цепочку переходов в дальнейшие партоны.

= 15: частица, которая распалась при вмешательстве пользователя.

$K(I, 2)$ : KF — код партона/частицы.

$K(I, 3)$ : номер строки родительской частицы или струи, если известно, иначе 0. Обратите внимание, что отнесение частицы к данной струе в системе струй нефизично, и приведенное здесь значение нужно понимать только как определение способа, которым было сгенерировано событие.

$K(I, 4)$ : обычно номер строки первой дочерней частицы; равен 0 для нераспавшихся частиц или нефрагментировавших струй.

$K(I, 5)$ : обычно номер строки последней дочерней частицы; равен 0 для нераспавшихся частиц или нефрагментировавших струй.

$P(I, 1)$ :  $p_x$ , импульс для  $x$ -направления, в ГэВ/с.

$P(I, 2)$ :  $p_y$ , импульс для  $y$ -направления, в ГэВ/с.

$P(I, 3)$ :  $p_z$ , импульс для  $z$ -направления, в ГэВ/с.

$P(I, 4)$ :  $E$ , энергия, в ГэВ.

$P(I, 5)$ :  $m$ , масса, в ГэВ/с<sup>2</sup>.

$V(I, 1)$ :  $x$ -координата точки вершины рождения, в мм.

$V(I, 2)$ :  $y$ -координата точки вершины рождения, в мм

$V(I, 3)$ :  $z$ -координата точки вершины рождения, в мм

$V(I, 4)$ : время рождения, в мм/с

$V(I, 5)$ : время жизни частицы, в мм/с. Если распад частицы выключен то  $V(I, 5)=0$

## **Лекция 10. Пример работы с генератором PYTHIA: моделирование рождения Хиггс-бозона и $Z^0$ -бозона в $pp$ -столкновении**

Чтобы проиллюстрировать работу PYTHIA, представим себе, например, что требуется смоделировать рождение хиггсов с массой 300 ГэВ. При использовании PYTHIA программа, позволяющая получить что-нибудь подобное, выглядит следующим образом.

C... Общие блоки

```
COMMON/LUJETS/N,K(4000,5),P(4000,5),V(4000,5)
COMMON/LUDAT1/MSTU(200),PARU(200),MSTJ(200),PARJ(200)
COMMON/PYSUBS/MSEL,MSUB(200),KFIN(2,-40:40),CKIN(200)
COMMON/PYPARS/MSTP(200),PARP(200),MSTI(200),PARI(200)
COMMON/LUDAT2/KCH(500,3),PMAS(500,4),PARF(2000),VCKM(4,4)
COMMON/LUDAT3/MDCY(500,3),MDME(2000,2),BRAT(2000),KFDP(2000,5)
COMMON/PAWC/HBOOK(10000)
```

C... Число генерируемых событий. Ключи для соответствующих процессов.

```
NEV=1000
MSEL=0
MSUB(102)=1
MSUB(123)=1
MSUB(124)=1
```

C... Выбор масс  $t$  и  $H$  и кинематических ограничений на массы.

```
PMAS(6,1)=140
PMAS(25,1)=300
CKIN(1)=290
CKIN(2)=300
```

C... Только при моделировании жесткого процесса: выключены ненужные задачи.

```
MSTP(61)=0
MSTP(71)=0
MSTP(81)=0
MSTP(111)=0
```

C... Инициализация и печать парциальных ширин.

```
CALL PYINIT('CMS','p','p',16000.)
CALL PYSTAT(2)
```

C... Вызов печати гистограмм.

```
CALL HLIMIT(10000)
CALL HBOOK1(1,'Higgs mass', 50, 275., 325.,0.)
```

C... Генерация события. Рассмотрим сначала несколько событий.

```
DO 200 IEV=1,NEV
CALL PYEVNT
IF(IEV.LE.3) CALL LULIST(1)
```

С... Цикл по частицам для нахождения хиггсов и гистограммирования их масс.

```
DO 100 I=1,N
100 IF(K(I,2).EQ.25) HMASS=P(I,5)
CALL HF1(1,HMASS,1.)
200 CONTINUE
```

С... Печать сечений и гистограмм.

```
CALL PYSTAT(1)
CALL HISTDO
END
```

Процессы 102, 123, и 124 – три основных источника хиггсов ( $gg \rightarrow H$ ,  $ZZ \rightarrow H$  и  $WW \rightarrow H$ ). MSUB(ISUB)=1 — команда для включения процесса ISUB. Значение ключа MSEL=0 отключает любые процессы рождения частиц; далее можно сформировать «меню» из желаемых процессов, используя соответствующие значения MSEL. Команда PMAS устанавливает значения масс топ кварка и хиггса, а переменные CKIN – желаемый массовый диапазон хиггса. Хиггс с номинальной массой 300 ГэВ фактически имеет довольно широкое распределение по массе брейт-вигнеровского типа. Затем идут ключи MSTP, которые должны изменить процедуру генерации, в данном случае – выключать излучение в начальном и конечном состоянии, многократные взаимодействия для пучка струй и фрагментацию, так, чтобы остался только «партонный скелет» жесткого процесса. Вызов PYINIT инициализирует PYTHIA, далее находятся максимумы сечений, заново вычисляются свойства распада хиггса, которые зависят от массы хиггса, и т.д. Свойства распада могут быть распечатаны из PYSTAT(2).

В следующем примере представлена программа, моделирующая образование  $Z^0$  на LEP.

С-----

С... Предварительное декларирование параметров.

С... Все переменные в формате двойной точности.

```
IMPLICIT DOUBLE PRECISION(A-H, O-Z)
```

С... Три функции Pythia возвращают целые переменные, поэтому их надо определить. INTEGER PYK, PYCHGE, PYCOMP

С... Задаются общие блоки

```
COMMON/PYJETS/N,NPAD,K(4000,5),P(4000,5),V(4000,5)
COMMON/PYDAT1/MSTU(200),PARU(200),MSTJ(200),PARJ(200)
COMMON/PYDAT2/KCHG(500,4),PMAS(500,4),PARF(2000),VCKM(4,4)
COMMON/PYDAT3/MDCY(500,3),MDME(8000,2),BRAT(8000),KFDP(8000,5)
COMMON/PYSUBS/MSEL,MSELPD,MSUB(500),KFIN(2,-40:40),CKIN(200)
COMMON/PYPARS/MSTP(200),PARP(200),MSTI(200),PARI(200)
COMMON/PYMSSM/IMSS(0:99),RMSS(0:99)
```

```

C-----
C... Инициализация.
    ECM=91.2D0
    NEV=1000
    MSEL=0
    MSUB(1)=1
    DO 100 IDC=MDCY(23,2),MDCY(23,2)+MDCY(23,3)-1
        IF(IABS(KFDP(IDC,1)).GE.6) MDME(IDC,1)=MIN(0,MDME(IDC,1))
100 CONTINUE
    CALL PYINIT('CMS','e+','e-',ECM)
    CALL PYSTAT(2)
C... Задание гистограмм.
    CALL PYBOOK(1,'charged multiplicity ME',100,-0.5D0,99.5D0)
    CALL PYBOOK(2,'charged multiplicity PS',100,-0.5D0,99.5D0)
C-----
C... Цикл по событиям.
    DO 300 ICA=1,2
        IF(ICA.EQ.1) THEN
            MSTP(48)=1
            MSTJ(101)=2
        ELSE
            MSTP(48)=0
        ENDIF
C... Начало цикла.
        DO 200 IEV=1,NEV
            CALL PYEVNT
C... Печать первых нескольких событий.
            IF(IEV.LE.2) CALL PYLIST(1)
C... Заполнение гистограмм.
            CALL PYEDIT(3)
            CALL PYFILL(ICA,DBLE(N),1D0)
C... Конец цикла события.
        200 CONTINUE
C... Конец цикла моделирования.
    300 CONTINUE
C-----
C... Третий этап. Вывод данных и гистограмм.
    CALL PYSTAT(1)
    CALL PYHIST

```

END

## Лекция 11. Изучение генераторов физических процессов: UrQMD

Генератор взаимодействия UrQMD основан на транспортной модели UrQMD (Ultra Relativistic Quantum Molecular Dynamics[1], [2]), предназначенной для моделирования взаимодействий тяжелых ионов в диапазоне энергий от SIS до RHIC. В рамках этого генератора изучается широкий круг эффектов связанных с тяжелыми ионами: от мультифрагментации и коллективного движения до образования конечных частиц и корреляций между ними. Данный генератор работает на UNIX-совместимых платформах.

Для начала работы с UrQMD необходимо откомпилировать командный код программы, написанной на FORTRAN77. Для компиляции необходимо, чтобы на PC были установлены: компилятор FORTRAN77 и GNU-make. Компиляция инициализируется заданием команды make в той директории, где находятся источники программного кода генератора. После успешной компиляции создается двоичный файл `urqmd.TYPE`. Где TYPE это результат выполнения команды `uname`.

В отличие, от генератора PYTHIA, где и моделирование и обработка, полученных результатов осуществляется в одной программе, в генераторе UrQMD с начала производится генерирование событий с определенными параметрами, и уже затем отдельной программой производится обработка, полученной информации.

Что бы запустить генератор UrQMD необходимо определить рабочие параметры во входном файле. Входной файл становится доступен из UrQMD с помощью определения его имени, как переменной окружения `ftn09`. Выходные файлы определяются таким же образом, как переменные окружения `ftn13`, `ftn14`, `ftn15` или `ftn16`, в зависимости от формата выходных данных. Ниже приведен пример определения переменных окружения в программной оболочке `bash`.

```
$ export ftn09=inputfile
$ export ftn14=outputfile
$ export ftn15=collisionfile
```

Стандартный вид входного файла представлен на следующем примере:

```
# this is a sample input file for uqmd
# projectile
# Ap Zp
pro 197 79
# optional: special projectile: ityp, iso3
```

```

# PRO 101 2
# target
# At Zt
tar 197 79
# number of events
nev 1
# time to propagate and output time-interval (in fm/c)
tim 40 40
#
# incident beam energy in AGeV
ene 10.7
#
imp 3.0
#
# equation of state
eos 0 # CASCADE mode
# some options and parameters
cto 4 1 # output of initialization
ctp 1 1.d0 # scaling for decay width of Resonances
#
f15 # no output to file15
# end of file
xxx

```

Из примера видно, что описание каждой переменной состоит из трех частей: Первая — это описание значения данной переменной, начинающееся со знака # (данная часть может отсутствовать, она нужна только для удобства), затем следует символьная метка переменной и последним определяется значение вводимой переменной. Входной файл не имеет строго определенной структуры, однако настоятельно рекомендуется, чтобы среди входных параметров были определены: налетающее ядро, мишень, прицельный параметр и начальная энергия пучка. Ниже приведены параметры, которые чаще всего требуют определения.

```

# строка комментариев
xxx последняя строка в файле
pro Ap Zp определяется налетающее ядро
tar At Zt определяется мишень
nev nevents определяется количество событий для моделирования
ene ebeam определяется начальная кинетическая энергия пучка (в лаб. системе)

```

plb pbeam определяется начальный импульс пучка (в лаб. системе)

esm srt полная энергия в случае коллайдерного режима (ГэВ)

imp bmax определяется прицельный параметр ( $b_{min}=0$ )

rsd seed определяется генератор случайных чисел

stb ityp определяется частица, как стабильная

f14 запрещается вывод в файл 14

f15 запрещается вывод в файл 15

Генератор UrQMD может осуществлять вывод информации в различные виды выходных файлов. Стандартные выходные файлы это ftn13 и ftn14. Выходной файл записывается в текстовом формате. Стандартный файл содержит заголовочную информацию и информацию об образовавшихся частицах на определенном интервале времени (в конце моделирования взаимодействия). Файл 13, вдобавок, содержит координаты образования частиц в координатном и импульсном пространствах. Файл ftn15 содержит информацию обо всех столкновениях и распадах в пределах данного события. Файл ftn16 содержит информацию обо всех распавшихся частицах и обо всех стабильных частицах в конце данного события. Запись каждого события состоит из заголовка и тела события. Заголовки всех четырех перечисленных выходных файлов идентичны. Информация представленная в заголовке выходного файла содержит данные о версии используемого генератора, типах сталкивающихся частиц, прицельном параметре и номере события.

В первой строке тела стандартного выходного содержится информация о количестве строк, содержащих данные об образованных частицах. Так же в первой строке содержится время образования частиц в fm/c. В следующей строке находятся счетчики количества столкновений, резонансов, распадов на одно данное событие. Затем следуют строки содержащие информацию об индивидуальной частице. Формат данных и выводимая информация зависят от типа, выбранного выходного файла. Стандартный выходной файл обеспечивает всю необходимую информацию для проведения любого анализа взаимодействий тяжелых ионов в рамках модели UrQMD. Ниже приведен список данных об одной частице записываемых в выходной файл ftn14:

1 время образования частицы в fm/c

2 x координата в fm

3 y координата в fm

4 z координата в fm

5 энергия частицы в GeV

6 значение проекции импульса частицы на ось x в GeV

- 7 значение проекции импульса частицы на ось  $y$  в GeV
- 8 значение проекции импульса частицы на ось  $z$  в GeV
- 9 масса частицы в GeV
- 10 уникальный код частицы
- 11 изоспин частицы
- 12 заряд частицы
- 13 индекс частицы с которой происходило последнее взаимодействие
- 14 количество столкновений
- 15 информация о родительском процессе

Файл `ftn15` содержит информацию: о каждом акте взаимодействия между двумя участниками, распаде резонанса и разрыве струи которые происходят во время взаимодействия тяжелых ионов. Эта информация может быть использована для восстановления пространственно-временной эволюции события. Каждый акт (столкновения, распад или аннигиляция) описывается от 3 до N строками (три строки для распадов и аннигиляций, четыре для рассеяния, больше для распада струй) с индивидуальной информацией о частице. Заголовок события состоит из одной строки. Формат заголовка одинаков с начальными строками, описывающими двухчастичные взаимодействия и распады, которые следуют дальше в файле. Для того чтобы отличить начало события от начала описания столкновения или распада, первое число в заголовке события равно 0. Затем следует номер события, массы налетающей частицы и мишени, прицельный параметр, энергия двух частиц в с.ц.м., полное сечение взаимодействия тяжелых ионов, энергия и импульс (на частицу) в лабораторной системе координат. Заголовочная строка содержит вначале количество входящих и выходящих частиц, затем код процесса взаимодействия, номер столкновения в соответствующем событии, время столкновения в fm/c, полную энергию в с.ц.м., полное сечение, парциальное сечение для соответствующего канала и, наконец, барионную плотность в точке столкновения. Одной из задач, для которых используется файл `ftn15`, является возможность отследить траекторию одной частицы в процессе взаимодействия или возможность отследить изменение со временем энергии в одном акте взаимодействия.

[1]. *Microscopic Models for Ultrarelativistic Heavy Ion Collisions*. S. A. Bass, M. Belkacem, M. Bleicher, M. Brandstetter, L. Bravina, C. Ernst, L. Gerland, M. Hofmann, S. Hofmann, J. Konopka, G. Mao, L. Neise, S. Soff, C. Spieles, H. Weber, L. A. Winckelmann, H. Stöcker, W. Greiner, C. Hartnack, J. Aichelin and N. Amelin. *Prog. Part. Nucl. Phys.* **41** (1998) 225–370.

[2]. *Relativistic Hadron-Hadron Collisions and the Ultra-Relativistic Quantum Molecular Dynamics Model (UrQMD)* M. Bleicher, E. Zabrodin, C. Spieles, S.A. Bass, C. Ernst, S. Soff, H. Weber, H. Stöcker and W. Greiner. J. Phys. **G25** (1999), 1859–1896

## Лекция 12. Генератор столкновений HIJING

Релятивистская физика тяжелых ионов изучает ядро-ядерные столкновения высоких энергий с целью выяснения поведения ядерной материи при экстремальных условиях, таких как высокая плотность и температура ядерной среды. Основной целью исследований является получение в лабораторных условиях нового состояния ядерного вещества, кварк-глюонной плазмы (КГП), которая предсказывается в стандартной модели элементарных частиц (квантовой хромодинамике).

Одним из жестких пробников нагретого ядерного вещества является образование струй с большими поперечными импульсами. Источниками этих струй являются частицы, возникающие при адронизации кварков и глюонов. Кварки и глюоны до своей адронизации распространяются в ядерной среде, которая может включать как адроны, так и кварковую составляющую. При этом потери энергии кварка в процессе столкновения с частицами среды и при тормозном излучении зависят от свойств среды («jet quenching»). Таким образом, «jet quenching» также может служить сигнатурой кварк-глюонной плазмы.

Другим сигналом является образование мини-струй. Оценивается, что мини-струи дают вклад до 50% поперечной энергии при центральных столкновениях тяжелых ионов. Мини-струи при средних поперечных импульсах имеют более короткие длины пробега в партонной среде и вероятнее диссипируют в фоне, теряя память о начальных корреляциях и изменяя свои характеристики при взаимодействии со средой.

Для изучения образования струй (мини-струй) при релятивистских взаимодействиях ядер и явления «jet quenching», была разработана Монте-Карло программа HIJING (heavy ion jet interaction generator). Данная программа основана на модели пертурбативной КХД (используется для образования струй) и Лундовской струнной модели (для фрагментации струй).

Программа позволяет генерировать события до энергий 10 ТэВ/нуклон в системе центра масс, что включает область энергий ускорителя LHC.

Стоит заметить, что время генерирования события сильно зависит от энергии взаимодействия и типа частиц. Например,

pp взаимодействие при энергии  $\sqrt{s} = 200$  ГэВ  $\sim 700$  событий/мин

pp взаимодействие при энергии  $\sqrt{s} = 1.8$  ТэВ  $\sim 250$  событий/мин

AuAu взаимодействие при энергии  $\sqrt{s} = 200$  ГэВ/нуклон  $\sim 1$  событие/мин

PbPb взаимодействие при энергии  $\sqrt{s} = 6.4$  ТэВ/нуклон  $\sim 1$  событие/10мин

Пакет HIJING состоит из подпрограмм для физического моделирования и общих блоков для определения параметров и записи событий.

HIJING использует PYTHIA 5.3 для генерации кинематических переменных в жестких процессах и JETSET 7.2 для фрагментации струй. Поэтому HIJING использует такие же коды частиц как JETSET 7.2 и PYTHIA 5.3. Таким образом, пользователи могут использовать (модифицировать) подпрограммы JETSET 7.2 и изменять переменные параметров JETSET.72 и PYTHIA. Чтобы сэкономить время компилирования, JETSET 7.2 не был включен в основное тело программы HIJING. Вместо этого, PYTHIA 5.3 была немного модифицирована и вместе с JETSET7.2 объединена в HIPYSET. Две эти программы HIJING и HIPYSET, представляют собой пакет программ HIJING.

Пакет HIJING состоит из нескольких основных подпрограмм, базовых общих блоков и дополнительных подпрограмм.

Существует две главные подпрограммы, которые пользователи должны вызывать в своей программе – это HIJSET и HIJING.

HIJSET подпрограмма, которую вызывают перед подпрограммой HIJING, для инициализации основных входных параметров, таких как: тип частицы, энергии взаимодействия, системы взаимодействия.

HIJSET (EFRM, FRAME, PROJ, TARG, IAP, IZP, IAT, IZT)

EFRM – определяется энергия столкновения (ГэВ/нуклон) частиц в типе эксперимента указанной параметром FRAME

FRANE – указывается тип эксперимента

= CMS – эксперимент для сталкивающихся пучков в системе центра масс (ЦМ) с импульсом частиц пучка в направлении +z и импульсом частиц мишени в направлении -z

= LAB – эксперимент на фиксированной мишени, указывается импульс налетающей частицы (PROJ) в направлении +z

PROJ, TARG – определяются переменные для определения частиц пучка и мишени

= P – протон

= PBAR – антипротон

= N – нейтрон

= NBAR – антинейтрон

= PI+

= PI-

=A – ядро

IAP, IAT – масса налетающего ядра и мишени

IZP, IZT – заряд налетающего ядра и мишени

HIJING – это главная подпрограмма пакета HIJING, которая вызывается строго после HIJSET. Стоит отметить, в то время как подпрограмма HIJING может вызываться неоднократно, то подпрограмма HIJSET вызывается единожды.

HIJING (FRAME, BMIN, BMAX)

FRAME – указывается тип эксперимента, который определяется при вызове HIJSET

BMIN, BMAX – определяется нижний и верхний предел прицельного параметра.

Существует также несколько основных общих блоков, которые обеспечивают пользователя важной информацией о генерированных событиях. Общий блок HIMAIN1 содержит основную информацию о событиях, а общий блок HIMAIN2 – информацию об образованных частицах.

COMMON/HIMAIN1/NATT,EATT,JATT,NT,NP,N0,N01,N10,N11

NATT – полное число образованных стабильных и нераспавшихся частиц в текущем событии

EATT – полная энергия образованных частиц в системе центра масс, для проверки закона сохранения энергии

JATT – полное число упругих рассеяний в текущем событии

NT,NP – число участников налетающей частицы и мишени в текущем событии

COMMON/HIMAIN2/KATT(130000,4),PAT(130000,4)

KATT(I,1): (I=1,...,NATT) – код образовавшейся частицы. Совпадает с кодом частиц PUTHIA

KATT(I,2): (I=1,...,NATT) – статус кода для идентификации источника образования частицы

KATT(I,3): (I=1,...,NATT) – номер строки родительской частицы, если известно, иначе 0

KATT(I,4): (I=1,...,NATT) – статус числа частиц

PAT(I,1-4): (I=1,...,NATT) – 4-импульс ( $p_x$ ,  $p_y$ ,  $p_z$ ,  $E$ ) (ГэВ/с, ГэВ) образовавшийся частицы

Отметим, не вдаваясь в подробности, еще несколько блоков. Блок HIJJET1 содержит информацию об образованных партонах, которые связаны с валентными кварками и дикварками налетающего ядра или мишени для формирования системы струн для фрагментации. Блок HIJJET2 содержит информацию об образованных партонах, которые не связаны с валентными кварками и дикварками. Блок HISTRNG содержит информацию

о нуклонах налетающего ядра и мишени, и соответствующие конститuentные кварки и дикварки.

Общий блок HIPARNT содержит коды состояний и параметры, которые регулируют эффективность программы HIJING. Все они имеют разумные значения по умолчанию, так что неопытный пользователь поначалу может пренебречь ими и исследовать полный диапазон возможностей постепенно. Некоторые параметры просто переопределяют параметры JETSET 7.2 и PYTHIA 5.3. Поэтому для более детального изучения рекомендуем, обратится к руководству пользователя этих программ.

COMMON/HIPARNT/HIPR1(100),IHPR2(50),HINT1(100),IHNT2(50)

Параметры HIPR1(100) и IHPR2(50) – содержат входные параметры (ключи) для события, в то время как параметры GHINT1(100),IHNT2(50) содержат дополнительную информацию о текущем событии.

Так как общее число параметров достигает около 200, мы остановимся только на наиболее значимых параметрах, на наш взгляд.

Значения по умолчанию определяются с помощью D=...

HIPR1(8):(D=2.0 GeV) минимальный поперечный импульс жестких или полужестких рассеяний.

HIPR1(11):(D=2.0 GeV) минимальный поперечный импульс струи, которая будет взаимодействовать с существующей ядерной материи.

HIPR1(12):(D=1.0 fm) поперечное расстояние между струей и существующими нуклонами дальше которых они будут взаимодействовать и струи будут терять энергию.

HIPR1(29):(D=0.4 fm) минимальное расстояние между двумя нуклонами внутри ядра, когда координаты всех нуклонов в ядре инициализированы

IHPR2(2):(D=3) ключ для начального и конечного излучения в жестких процессах

=0: начальное и конечное излучение выключено;

=1: начальное излучение включено, а конечное - выключено;

=2: начальное излучение выключено, а конечное - включено;

=3: начальное и конечное излучение включено.

IHPR2(3):(D=0) ключ для жестких рассеяний с определенным поперечным импульсом  $P_T = \text{HIPR1}(10)$  на событие.

=1: обыкновенный жесткий процесс

=2: только прямое образование фотона

=3: образование тяжелых кварков (очарованных  $\text{IHPR2}(18)=0$  или прекрасных

$\text{IHPR2}(18)=1$ ). Для инклюзивного образования нужно положить  $\text{HIPR1}(10)=0.0$

IHPR2(4):(D=1) ключ для jet quenching в существующей ядерной среде

INPR2(6):(D=1) ключ для ядерных эффектов на партонной функции распределения  
INPR2(8):(D=10) максимально число образованных струй на нуклон-нуклонном взаимодействии. Если INPR2(8)=0, образование струй будет выключено

INPR2(9):(D=0) ключ гарантирует по крайней мере образование одной пары мини-струй на событие

INPR2(11):(D=1) выбирает модель образования барионов

=0 барион-антибарионная пара не образуется, начальные дикварки трактуются как точечные.

=1 дикварк-антидикварковая пара образуется, начальные дикварки трактуются как точечные.

=2 дикварк-антидикварковая пара образуется, с возможностью рассматривать дикварк как состояние "porcorn"

INPR2(12):(D=1) опция для выключения распада частиц таких как  $\pi^0$ ,  $K^0_S$ ,  $D^{+-}$ ,  $\Lambda$ ,  $\Sigma^{+-}$ ,  $X^0$ ,  $\Omega^-$ .

INPR2(13):(D=1) опция для включения single дифракционной реакции.

INPR2(14):(D=1) опция для включения упругих процессов

INPR2(18):(D=0) опция для включения образования В-кварков. По умолчанию включено образования очарованных кварков. Когда включается образование В-кварков, образование очарованных кварков автоматически выключается.

HINT1(72)-HINT1(75): параметры для распределения Вудса – Саксона (плотности ядерного вещества) налетающего ядра,  $\rho(x) = FNORM * (1 + W * (X/R)^2) / (1 + EXP((X-R)/D))$   
 $R = HINT1(72)$  – радиус ядра,  $D = HINT1(73)$  – толщина поверхности,  $W = HINT1(74)$  – принимает в расчет центральную неравномерность,  $FNORM = HINT1(75)$  – центральная плотность, находится из условий нормировки.

HINT1(76)-HINT1(79): параметры для распределения Вудса – Саксона ядра мишени,  
 $\rho(x) = FNORM * (1 + W * (X/R)^2) / (1 + EXP((X-R)/D))$       $R = HINT1(76)$ ,      $D = HINT1(77)$ ,  
 $W = HINT1(78)$ ,  $FNORM = HINT1(79)$ .

### **Лекция 13. Ознакомление с программой GEANT3, предназначенной для моделирования прохождения частиц через экспериментальную установку**

С увеличением масштабов и сложности экспериментов по физике высоких энергий работы по проведению модельных расчетов требуют все большего и большего внимания и становятся неотъемлемой частью:

разработки и оптимизации детекторов;

развития и тестирования программ по реконструкции и анализу данных;  
обработки экспериментальных данных.

Первая версия GEANT была написана в 1974 году и обладала способностью трассировать всего несколько частиц в событии через очень простую экспериментальную установку [1]. После значительного усовершенствования частей программы отвечающих за геометрию и трассировку новая версия GEANT 3.21 была выпущена в Марте 1994 года. С тех пор эта версия является основной рабочей версией, и значительных изменений в нее не вносилось.

GEANT является программой для моделирования прохождения элементарных частиц через вещество. Первоначально разработанная для экспериментов по физике высоких энергий, сегодня эта программа применяется в таких областях как медицина, биология и астрофизика.

Основные направления использования GEANT в физике высоких энергий:

- 1) трассировка частиц сквозь экспериментальную установку для моделирования отклика детектора;
- 2) графическое представление экспериментальной установки и траектории частиц.

Эти два направления объединены в интерактивной версии GEANT. Такой подход очень продуктивен так, как позволяет проводить прямые наблюдения того, что происходит с частицей во время прохождения через детектор и значительно облегчает отладку программы.

В соответствии с этими направлениями GEANT позволяет:

- 1) описывать экспериментальную установку как структуру геометрических объемов. Каждому объему присваивается соответствующий идентификатор (номер) среды ([GEOM]). Различные объемы могут обладать одинаковыми идентификаторами среды. Среда определяется так называемыми параметрами трассировки через среду (TRACKING MEDIUM parameters, которые включают в себя описание материала, заполняющего объем [CONS];

- 2) использовать события, полученные при моделировании каким либо генератором взаимодействий [KINE];

- 3) трассировать частицы сквозь различные области экспериментальной установки, принимая во внимание геометрические формы различных объемов и физические эффекты в соответствии с природой частиц с видами взаимодействий со средой и магнитными полями в среде [TRAK], [PHYS];

- 4) записывать траекторию частиц и отклик чувствительных элементов детекторов [TRAK], [HITS];

5) визуализировать детекторы и траекторию частиц [DRAW], [XINT].

Программа, созданная пользователем для моделирования содержит обязательные и необязательные подпрограммы, определяющие действие GEANT на различных этапах работы. Задачами пользователя являются:

- 1) создание пользовательских подпрограмм описывающих входные данные и экспериментальное окружение;
- 2) компоновка необходимых программных сегментов и утилит в выполняемую программу;
- 3) создание соответствующих потоков данных, контролирующих выполнение программы.

Основная программа в процессе своего выполнения проходит три этапа:

- 1) инициализация;
- 2) обработка события;
- 3) прекращение работы.

Пользователь на каждом из этих этапов может включить свой собственный код в соответствующие подпрограммы.

Инициализация управляется пользователем в подпрограмме UGINIT. Процедура инициализации состоит из следующих шагов, большинство из которых осуществляется вызовом соответствующих подпрограмм GEANT:

GINIT	инициализация общих блоков GEANT параметрами по умолчанию;
GFFGO	чтение данных для изменения параметров и установок по умолчанию или для обеспечения информацией о текущем событии;
GZINIT	инициализация менеджера по работе с банками памяти;
GDINIT	инициализация пакета рисования;
GPART	заполнение массива JPART свойствами частиц [CONS];
GMATE	заполнение массива JMATE характеристиками используемых материалов [CONS];

Затем следует пользовательский код, в котором определяется:

- 1) геометрия различных составляющих экспериментальной установки, которая хранятся в массивах данных JROTM и JVOLU;
- 2) параметры среды, через которую происходит трассировка [CONS], [TRAK], которые хранятся в массиве данных JTMED;
- 3) те элементы установки, которые необходимо рассматривать как чувствительные, дающие отклик, когда частица проходит сквозь них [HITS];

Обычно все это определяется в пользовательской подпрограмме UGEOM.

GGCLOS обработка всей пользовательской информации и подготовка к трассировки частиц;

GBHSTA создание стандартных гистограмм GEANT, если требуется пользователем;

GPHYSI вычисление таблиц энергетических потерь и сечений и запись их в массив JMATE. [CONS], [PHYS]

Обработка события начинается с вызова подпрограммы GRUN, которая должна выполняться для каждого события, передавая контроль выполнения программы следующим подпрограммам:

GTRIGI инициализация расчета события и создание заголовочного массива JHEAD;

GTRIG обработка одного события;

GTRIGC очистка памяти, отведенной для одного события;

Во время своего выполнения GTRIG вызывает следующие подпрограммы:

GUKINE генерирует или читает первоначальную кинематику и записывает ее в массивы JVERTX и JKINE;

GUTREV вызывает GTREVE которая производит следующие действия с вершиной: 1) перемещает все частицы соответствующие вершине из постоянного стека JKINE во временный JSTAK, 2) контролирует прохождение каждой частицы сквозь установку вызывая GUTRAK/GTRACK; каждая частица трассируется в свою очередь и после пересечения чувствительного объема пользователь может записать любую необходимую информацию в массив данных JHITS.

Массив JSTAK является массивом вида LIFO (Last In — First Out). Это значит, что вторичная частица, образованная при трассировке первичной будет трассироваться дальше, а трассировка первичной частицы начнется только после того, как будет закончена трассировка вторичной и всех частиц, рожденных вторичной частицей. Необходимо подчеркнуть, что трассировка вторичных частиц в GEANT по умолчанию не производится. Пользователь должен сам определить, трассировать вторичные частицы или нет через подпрограммы GSKING/GSKPHO. Массив данных JXYZ, содержащий пространственные координаты треков частиц может быть заполнен во время трассировки [TRAK].

GUDIGI моделирует отклик детектора на событие, используя информацию о хитах, записанную в массив JHITS во время трассировки частиц. Информация об отклике детектора хранится в массиве JDIGI [HITS].

GUOUT осуществляет все необходимые действия при окончании обработки события и записывает всю информацию.

Некоторые дополнительные подпрограммы вызываются во время обработки события:

1) адронное взаимодействие моделируется с помощью GHEISHA [2] или FLUKA [3,4,5]. В подпрограммах GUPHAD и GUHADR пользователь может выбрать, какой из методов будет использован для генерации адронного ливня. По умолчанию в GEANT используется GHEISA.

2) подпрограмма GUSWIM вызывается в том случае, если происходит трассировка заряженной частицы в магнитном поле.

Прекращение работы контролируется пользователем в подпрограмме GULAST. В простейшем случае она может содержать просто вызов подпрограммы GLAST, которая вычисляет и печатает некоторую статистическую информацию, например: время на одно событие, параметры использования памяти и т. д.

[1] R.Brun, M.Hansroul, and J.C.Lassalle. *GEANT User's Guide*, DD/EE/82 edition, 1982.

[2] H.C.Fesefeldt. Simulation of hadronic showers, physics and applications. Technical Report PITHA 85-02, III Physikalisches Institut, RWTH Aachen Physikzentrum, 5100 Aachen, Germany, September 1985.

[3] P.A.Aarnio et al. Fluka user's guide. Technical Report TIS-RP-190, CERN, 1987, 1990.

[4] A.Fass`o A.Ferrari J.Ranft P.R.Sala G.R.Stevenson and J.M.Zazula. FLUKA92. In *Proceedings of the Workshop on Simulating Accelerator Radiation Environments*, Santa Fe, USA, 11-15 January 1993.

[5] A.Fass`o A.Ferrari J.Ranft P.R.Sala G.R.Stevenson and J.M.Zazula. A Comparison of FLUKA Simulations with measurements of Fluence and Dose in Calorimeter Structures. *Nuclear Instruments & Methods A*, 332:459, 1993.

**Лекции 14, 15. Основные блоки данных, функций и переменных, использующихся в приложении GEANT3. Структура программы, использующей приложение GEANT3. Пример работы с GEANT3: моделирование адронного (электромагнитного) calorimetра. Методы визуализации информации в программном пакете GEANT3.**

Взаимодействие между программными элементами GEANT осуществляется посредством массивов данных переменных, сгруппированных в несколько общих блоков. К тому же

внутри каждого программного сегмента подпрограммы взаимодействуют друг с другом с помощью обмена аргументами.

В GEANT чтение вводимых данных осуществляется подпрограммой GFFGO, которая использует пакет FFREAD [1]. Ключевые слова, считываемые GFFGO можно разделить на следующие типы:

- 1) общее управление;
- 2) управление физическими процессами;
- 3) управление процедурой отладки и процедурами ввода вывода;
- 4) пользовательские приложения;
- 5) генерация события.

Стандартные вводимые данные описываются в GEANT следующим образом: KEY — ключевое слово описывающее данные, сокращенное до 4 символов, N — максимальное число вводимых переменных, T — тип переменных. До вызова GFFGO пользователь может определить свои вводимые данные, используя вызов подпрограммы FFKEY:

```
CALL FFKEY ('key', VAR(1), NVAR, 'type')
```

Эти данные будут распознаваться GEANT таким же образом, как и предопределенные в GEANT данные.

Кинематические переменные в GEANT всегда соответствуют так называемой Master Reference System (MARS). Эта система определяется как локальная система отсчета, в которой определен первый объем, который содержит все остальные объемы (элементы) детектора. MARS – это декартова система координат в которой  $z=[x;y]$ .

Экспериментальная установка описывается определением первичного объема, в котором будут размещены все остальные объемы детектора. В терминологии GEANT каждый объем содержащий внутри себя другие объемы, созданные добавлением или делением первичного объема, называется Материнским. Объемы внутри Материнского называются Дочерними. Дочерние объемы в свою очередь могут содержать внутри себя объемы, и следовательно, будут по отношению к ним Материнскими.

Каждый объем, определенный в GEANT, имеет систему отсчета, присвоенную этому объему. Если этот объем содержит другие, то система отсчета называется Mother Reference System (MRS). Каждый дочерний объем имеет свою Daughter Reference System (DRS).

Основными физическими единицами измерения в GEANT являются: сантиметр, секунда, килогаусс, ГэВ, ГэВ/с, ГэВ/с<sup>2</sup> и градус.

Ниже приведен костяк стандартной программы, использующей GEANT. Пользовательские подпрограммы, которые рекомендуется использовать при составлении основной программы, выделены **жирным шрифтом**.

```
PROGRAM GEXAMP
PARAMETER (NGBANK=50000, NHBOOK=5000)
COMMON/GCBANK/Q(NGBANK)
COMMON/PAWC /H(NHBOOK)
C--> Инициализация памяти HBOOK и GEANT
CALL GZEBRA( NGBANK)
CALL HLIMIT(-NHBOOK)
C--> Инициализация графики
CALL HPLINT(0)
CALL IGMETA(8,0)
C--> Инициализация GEANT
CALL UGINIT
C--> Начало обработки события
CALL GRUN
C--> Конец обработки события
CALL UGLAST
END
*-----
SUBROUTINE UGINIT
C--> Инициализация GEANT
CALL GINIT
C--> Чтение вводимых данных
OPEN(4, FILE='gcards.dat', STATUS='UNKNOWN')
CALL GFFGO
C--> Инициализация структуры данных
CALL GZINIT
C--> Инициализация графики
CALL GDINIT
IF(NRGET.GT.0) THEN
C--> Чтение структуры данных из файла
CALL GRFILE(1, 'mygeom.dat', 'I')
ELSE
C--> Инициализация таблиц частиц
CALL GPART
C--> Описание геометрии и материалов
CALL UGEOM
```

```

ENDIF
C--> Таблицы энергетических потерь и сечений
CALL GPHYSI
IF(NRSAVE .GT. 0) THEN
C--> Сохранение параметров структур данных
CALL GRFILE(2, 'mysave.dat', 'NO')
ENDIF
C--> Печать банков данных
CALL GPRINT('MATE', 0)
CALL GPRINT('TMED', 0)
CALL GPRINT('VOLU', 0)
C--> Запись гистограмм
END
*-----
SUBROUTINE UGEOM
C--> Определяются материалы геометрия и среда трассировки
C--> Записываются банки данных о геометрии.
CALL GGCLOS
END
*-----
SUBROUTINE UKINE
C--> Генерируется кинематика
C--> Карта данных KINE itype x y z px py pz
CALL GSVERT(PKINE, 0, 0, 0, 0, NVERT)
CALL GSKINE(PKINE(4), IKINE, NVERT, 0, 0, NT)
C--> Печатается кинематика
IF (IDEBUG.NE.0) THEN
CALL GPRINT('VERT', 0)
CALL GPRINT('KINE', 0)
END IF
END
*-----
SUBROUTINE GUSTEP
C--> Вызывается в конце каждого шага трассировки
C--> Отладка программы
CALL GDEBUG
C--> Запись созданных частиц
IF (NGKINE.GT.0) CALL GSKING (0)
END

```

\*-----

SUBROUTINE **UGLAST**

C--> Подпрограмма завершения Termination routine

C--> Печатаются гистограммы и статистика

CALL GLAST

C--> Закрывается файл HIGZ/GKS

CALL IGEND

END

В GEANT используется технология ZEBRA для хранения больших структур данных, это позволяет адаптировать размер данных программы с размером решаемой задачи. Вызов GZEBRA инициализирует общий банк данных /GCBANK/, в котором хранится структура данных GEANT. Эта операция должна быть выполнена до выполнения любых других операций в системе GEANT. Вызов HLIMIT инициализирует систему ZEBRA для использования общих блоков /PAWC/ пакета гистограммирования HBOOK. Вызов HLIMIT должен происходить после вызова GZEBRA и аргумент должен определять размерность общего банка /PAWC/ со знаком минус. Основная программа работает в фоновом режиме. Если есть необходимость осуществления моделирования в интерактивном режиме, необходимо подключить интерактивную программу GXINT до вызова пользовательского кода. В показанном примере используется графическое приложение. Часто для программ в фоновом режиме или для небольших тестов графика не нужна. Для того чтобы исключить графику из своей программы необходимо удалить вызов следующих подпрограмм IGINIT, IGMETA, IGEND, GDINIT и GDEBUG. С другой стороны, если пользователь хочет включить и отладку и графику необходимо добавить следующий код:

SUBROUTINE GDCXYZ

ENTRY IGSA

ENTRY GDTRAK

END

Пользовательский код для определения среды и геометрии должен быть определен в подпрограмме UGEOM. Также предопределенная структура данных может быть считана с диска, однако в любом случае рекомендуется произвести вызов подпрограммы GPHYSI для инициализации таблиц сечений.

Удобно сохранять входные данные в специальном входном файле. Это позволяет использовать стандартный способ ввода и в случае необходимости легко менять исходные данные, не компилируя основную программу. Пример стандартного файла gcards.dat приведен ниже:

READ 4  
TRIG 10  
STOP

В первой строке содержится инструкция FFREAD открыть и начать обрабатывать файл с логическим номером 4, во второй строке определяется (переопределяется) количество событий для обработки. Последняя строка завершает ввод данных из файла.

Одной из основных задач при создании программы моделирования на GEANT является описание экспериментальной установки. Это осуществляется описанием геометрии и среды (материала). Способ задания среды, заполняющей объем, состоит в задании двух блоков параметров. Первый из них определяет природу материала заполняющего объем и содержит такую информацию как: атомный номер, атомный вес, плотность и т. д. (подробнее см. описание подпрограммы GSMATE). Второй набор параметров определяет процесс трассировки частицы: магнитное поле, точность трассировки, максимальные энергетические потери на один шаг трассировки и т. д. (подробнее см. описание подпрограммы GSTMED). Каждой среде прохождения соответствует материал через определенный пользователем уникальный номер среды. Различным средам прохождения могут соответствовать, с определенными ограничениями, одинаковые материалы. Каждый объем заполняется средой прохождения с определенным номером среды. Различные объемы могут заполняться одинаковыми средами прохождения. Трассировка частицы сквозь экспериментальную установку требует доступа к данным описывающим: геометрию детектора, материал и среду прохождения, свойства частиц.

В GEANT есть стандартные материалы, которые уже определены и описаны в массиве данных JMATE. Следующая подпрограмма осуществляет запись стандартных материалов в массив данных:

**CALL GMATE**

В случае необходимости существует возможность определить свой материал и записать его в массив данных.

**CALL GSMATE** (IMATE, CHNAMA, A, Z, DENS, RADL, ABSL, UBUF, NWBUF)

IMATE – номер материала (INTEGER);

CHNAMA – имя материала (CHARACTER \*20);

A – атомный вес (REAL);

Z – заряд (REAL);

DENS – плотность в гр/см<sup>3</sup> (REAL);

RADL – радиационная длина в см (REAL);

ABSL – длина абсорбции в см (REAL);

UBUF – массив NWBUF пользовательских переменных (REAL);

NWBUF – количество переменных UBUF (INTEGER).

Существует возможность получить параметры материала:

CALL **GFIMATE** (IMATE, CHNAMA, A, Z, DENS, RADL, ABSL, UBUF, NWBUF)

Параметры вызываемой подпрограммы те же, что и в **GSMATE**. Подпрограмма:

CALL **GPMATE** (IMATE)

Осуществляет печать параметров материала. Как отмечалось выше, помимо определения материала, необходимо также задать и среду трассировки. Данная операция осуществляется вызовом следующей подпрограммы:

CALL **GSTMED** (ITMED, NATMED, NMAT, ISVOL, IFIELD, FIELDM, TMAXFD, STEMAX, DEEMAX, EPSIL, STMIN, UBUF, NWBUF)

ITMED – номер среды трассировки (INTEGER);

NATMED – имя среды трассировки (CHARACTER \*20);

NMAT – номер материала (INTEGER);

ISVOL – идентификатор того является ли среда регистрирующей (INTEGER):

<0 не регистрирующий объем;

>0 регистрирующий объем;

IFIELD – идентификатор магнитного поля (INTEGER):

=0 нет магнитного поля;

=1 сильно не однородное магнитное поле (определенное пользовательской функцией GUFLD): трассировка происходит с использованием метода Рунге-Куты в подпрограмме GRKUTA;

=2 неоднородное магнитное поле (определенное пользовательской функцией GUFLD): трассировка по спирали осуществляется подпрограммой GHELIX;

=3 однородное магнитное поле вдоль оси z величиной, определенной в FIELD, трассировка по спирали осуществляется подпрограммой GHELIX3;

FIELDM – максимальное значение магнитного поля в килогауссах (REAL);

TMAXFD – максимальное угловое отклонение в магнитном поле на один шаг трассировки, в градусах (REAL);

STEMAX – максимальное значение шага трассировки в см (REAL);

DEEMAX – максимальные относительные энергетические потери на один шаг трассировки  $0 < DEEMAX < 1$  (REAL);

EPSIL – точность пересечения границ в см (REAL);

STMIN – минимальное значение энергетических потерь, обусловленных множественным рассеянием, черенковским излучением или эффектами магнитного поля для максимального шага трассировки, в см (REAL);

UBUF – массив NWBUF пользовательских переменных (REAL);

NWBUF – количество переменных UBUF (INTEGER).

Существует возможность получить параметры материала:

CALL **GFTMED** (ITMED, NATMED, NMAT, ISVOL, IFIELD, FIELDM, TMAXFD, STEMAX, DEEMAX, EPSIL, STMIN, UBUF, NWBUF)

Параметры вызываемой подпрограммы те же, что и в **GFTMED**. Подпрограмма:

CALL **GPTMED** (ITMED)

осуществляет печать параметров среды трассировки.

Во время инициализации необходимо задать параметры частиц участвующих в моделировании. Эта процедура осуществляется вызовом подпрограммы:

CALL **GPART**

При выполнении которой записываются характеристики стандартных частиц в структуру данных JPART. Подпрограмма

CALL **GPIONS**

Производит запись характеристик ионов в структуру данных JPART. Пользователь может определить сам и другие частицы или перепределить некоторые свойства стандартных частиц:

CALL **GSPART** (IPART, CHNPART, ITRTYP, AMASS, CHARGE, TFILE, UB, NWB)

IPART – номер частицы (INTEGER);

CHNPART – имя частицы (CHARACTER \*20);

ITRTYP – тип подпрограммы трассировки (INTEGER);

1 трассировка подпрограммой GTGAMA;

2 трассировка подпрограммой GTELEC;

3 трассировка подпрограммой GTNEUT;

4 трассировка подпрограммой GTHADR;

5 трассировка подпрограммой GTMUON;

6 трассировка специальной частицы geantiono производится подпрограммой GTNINO;

7 трассировка тяжелых ионов производится подпрограммой GTCKOV;

8 трассировка фотонов производится подпрограммой GTHION.

AMASS – вес частицы в ГэВ (REAL);

CHARGE – заряд частицы (REAL);

TFILE время жизни частицы в секундах (REAL);

UB – массив NWB пользовательских переменных (REAL);

NWB – количество переменных UB (INTEGER).

Существует возможность получить параметры частицы:

CALL **GFPART** (IPART, CHNPART, ITRTYP, AMASS, CHARGE, TFILE, UB, NWB)

Параметры вызываемой подпрограммы те же, что и в **GSPART**. Подпрограмма:

CALL **GPPART** (IPART)

осуществляет печать параметров частицы.

Неотемлемой частью GEANT является возможность визуализировать информацию о проводимых модельных расчетах. Для этой цели предназначен программный пакет рисования, который выполняет следующие функции: визуализирует компоненты детектора, изображает логическое дерево этих компонентов, рисует траекторию частиц, показывает хиты в регистрирующих элементах детектора. Визуализация может происходить, как в интерактивном режиме работы, так и при вызове подпрограмм основной программы.

Лучший способ работы с графикой, это инициализация программного пакета HPLOT [2] подпрограммой HPLINT. GDINIT производит вызов программного пакета визуализации GEANT. Основными подпрограммами которого являются:

GDRAW – рисует проекцию детектора;

GDRVOL - рисует другую проекцию детектора;

GDRAWC – рисует разрез детектора вдоль одной из осей;

GDRAWX – рисует разрез детектора под любым углом;

GDXYZ – рисует треки по окончании обработки события;

GDCXYZ – рисует треки во время трассировки частицы;

GDPART – рисует имя частицы и номер трека в конце события;

GDAHIT – рисует один хит;

GDHITS – рисует хиты для детекторов измеряющих траекторию;

GDCHIT – рисует хиты для детекторов типа калориметер;

GDTREE – рисует геометрическое дерево созданных объемов детектора;

GDSPEC – рисует спецификацию указанного объема;

GDFSPC – рисует спецификацию нескольких указанных объемов.

Для задания геометрии детектора служит программный пакет геометрии. Его основными задачами являются: определение, на этапе инициализации, геометрии детектора, сквозь которую будет происходить трассировка частицы; осуществление взаимодействия, во время обработки события, между подпрограммами трассировки и информацией о среде трассировки частицы. Взаимодействие устанавливается

подпрограммами GMEDIA/GTMEDI, GTNEXT/GNEXT и GINVOL которые дают ответы на следующие вопросы: в каком объеме данная точка, каково расстояние до ближайшего объема вдоль траектории частицы, каково расстояние до ближайшего объема, данная точка все еще в текущем объеме. При задании каждого объема определяются следующие характеристики: имя, форма, локальная координатная система, свойства вещества заполняющего объем, свойства среда трассировки, набор параметров для визуализации объема. До того как объем не будет позиционирован в основной системе координат он не имеет никакого пространственного соответствия с другими объемами детектора. Первоначальный объем должен быть определен самым первым, он и содержит основную координатную систему. Данный объем является материнским для всех остальных объемов, определенных впоследствии.

Пользователь может определить объем, используя следующие подпрограммы: GSVOLU, GSPOS, GSPOSP, GSDVN, GSDVT, GSDVX. Рассмотрим подробнее основные.

CALL **GSVOLU** (CHNAME, CHSHAP, NMED, PAR, NPAR, IVOLU)

CHNAME – уникальное имя объема (CHARACTER\* 4);

CHSHAP – имя формы объема. В GEANT существует 16 форм которые используются для определения объема: коробка, три типа трапезоидов, два типа труб, два типа конусов, сфера, параллелепипед, поликонус, полигон, эллиптически рассеченная труба, гиперболическая труба, скрученный трапезоид, обрезанная труба.

NMED – среда трассировки для данного объема, если объем располагается внутри другого то его среда трассировки замещает материнскую (INTEGER)

PAR – массив содержащий параметры формы объема (REAL);

NPAR – количество параметров описывающих форму (INTEGER);

IVOLU – номер объема (INTEGER).

CALL **GPVOLU** (IVOLU)

Производит печать параметров объема. Подпрограмма GSPOS осуществляет позиционирование данного объема внутри материнского

CALL **GSPOS** (CHMANE, NR, CHMOTH, X, Y, Z, IROT, CHONLY)

CHMANE – имя позиционируемого объема (CHARACTER \*4);

NR – номер копии позиционируемого объема (INTEGER);

CHMOTH – имя объема в который происходит размещение объема CHMANE (CHARACTER \*4);

X – координата x позиционируемого объема в основной системе координат (REAL);

Y – координата y позиционируемого объема в основной системе координат

(REAL);

Z – координата z позиционируемого объема в основной системе координат

(REAL);

IROT – номер матрицы вращения, описывающей ориентацию объема относительно координатной системы материнского объема (INTEGER);

CHONLY – индекс показывающий размещен ли данный объем только в данном материнском объеме или он может находится в других материнских объемах. Возможные значения ONLY или MANY (CHARACTER \*4).

Часто при описании детектора возникает необходимость определять одинаковые по своим свойствам объемы отличающиеся только размерами, например при описании электро-магнитного калориметра. В этом случае удобно и логично присвоить всем объемам одинаковые имена и различные параметры позиционирования и размеров. Это можно сделать определяя объем с нулевыми параметрами в подпрограмме **GSVOLU** и присвоить параметры в подпрограмме **GSPOSP**. Эта подпрограмма в основном подобна **GSPOS**, за исключением того что объем с нулевыми параметрами может быть позиционирован только **GSPOSP**, где определяются параметры объема.

CALL **GSPOS** (CHMANE, NR, CHMOTH, X, Y, Z, IROT, CHONLY, PAR, NPAR)

PAR – массив содержащий параметры формы объема (REAL);

NPAR – количество параметров описывающих форму (INTEGER);

[1] R.Brun et al. *FFREAD User Guide and Reference Manual*, dd/us/71 edition, February 1987.

[2] R. Brun and H. Renshall. *HPLOT users guide*, Program Library Y251. CERN, 1990.

**Лекция 16. Изучение комплексного подхода к моделированию современного ускорительного эксперимента на примере программы AliROOT, включающей в себя генераторы столкновений, GEANT3, GEANT4 и обеспечивающей визуализацию информации**

Тяжелоионный эксперимент ALICE [1] (A Large Ion Collider Experiment) на строящемся коллайдере LHC (ЦЕРН, Женева) является крупнейшим в мире экспериментом в области релятивистской физики тяжелых ионов. Уникальность эксперимента состоит в изучении физических процессов, возникающих при столкновении тяжелых ионов свинец-свинец (Pb-Pb) при энергиях в системе центра масс 5,5 ТэВ/нуклон, что позволяет достичь, и даже в какой-то степени перекрыть, энергетическую шкалу, ныне достигаемую только в исследованиях космических лучей. В соответствии с физическими задачами, установка ALICE содержит набор необходимых субдетекторов,

расположенных как внутри большого магнита L3 (до 0,5 Тесла), так и вне его: центральный трекер на основе кремниевых детекторов, время-проекционная камера, времяпролетный детектор идентификации частиц, черенковский детектор идентификации частиц, детектор переходного излучения, электромагнитный калориметр (фотонный спектрометр) и ряд других, а также мюонное плечо, включающее в себя абсорберы адронов, фотонов и электронов, дипольный магнит и мюонную станцию.

Вся эта система субдетекторов работает как единый ансамбль и должна запускаться сигналами отбора (триггерами) полезных событий различного уровня. Одним из основных триггерных детекторов (субдетекторов) является детектор T0, проект которого разработан и в настоящее время реализуется МИФИ.

Детектор состоит из двух сборок черенковских счетчиков, по 12 счетчиков в каждой сборке. Черенковские счетчики основаны на российских фотоумножителях ФЭУ-187 (диаметр 30 мм, длина 45 мм) с сетчатыми динодами и кварцевых радиаторах диаметром 30 мм и длиной 30 мм. Одна из сборок (П — правая) расположена довольно близко к вершине события. Ее положение — 70 см от вершины ограничено положением мюонного абсорбера. На стороне, противоположной мюонному абсорберу, расстояние левой сборки (Л) от вершины событий составляет 3,5 м.

Для осуществления моделирования эксперимента ALICE была разработана программная оболочка AliRoot [2]. AliRoot выполняет две основные функции: 1) моделирование первичных взаимодействий протонов или тяжелых ионов 2) анализ и реконструкция данных, полученных при моделировании, и реальных данных, полученных в эксперименте. Разработка программных пакетов для ALICE вначале велась независимо для различных физических групп и детекторов, затем было принято решение о создании общего программного пакета, в рамках одной системы обработки. Такой системой стал ROOT — стандартный программный пакет для обработки данных в физике высоких энергий, реализованный на C++.

Для установки AliRoot на локальном PC необходимо, чтобы на этом PC был уже установлен ROOT и одна из программ трассировки частиц через экспериментальную установку: GEANT3, FLUKA [3], GEANT4 [4]. Эти программы производят подробный отклик детекторов «хит». Хиты затем переводятся в идеальный отклик электроники «дигиты» (digits). Также производится моделирование и реального отклика электроники «сдигиты» (sdigits). Следующим этапом реконструкции является перевод дигитов в формат выходных данных с электроники “raw” формат. На этом этапе данные моделирования и реальные данные должны стать идентичными по формату. Далее AliRoot будет обрабатывать данные в “raw” формате и «не заметит» разницы. На заключительном

этапе образуются ESD (event summary data) данные, содержащие физическую информацию.

Перед началом установки AliRoot пользователь должен определить следующие переменные окружения:

```
# ROOT
export ROOTSYS=/home/mydir/root
export PATH=$PATH:$ROOTSYS/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ROOTSYS/lib
# AliRoot
export ALICE=/home/mydir/alice
export ALICE_ROOT=$ALICE/AliRoot
export ALICE_TARGET=`root-config --arch`
export PATH= $PATH:$ALICE_ROOT/bin/tgt_${ALICE_TARGET}
export LD_LIBRARY_PATH= $LD_LIBRARY_PATH:$ALICE_ROOT/lib/tgt_${ALICE_TARGET}
# GEANT 3
export PLATFORM=`root-config --arch`
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ALICE/geant3/lib/tgt_${ALICE_TARGET}
```

ROOTSYS – это место где установлен ROOT;

ALICE – корневая директория, где установлены все программные пакеты, используемые в ALICE;

ALICE\_ROOT – директория где установлен AliRoot, поддиректория ALICE;

ALICE\_TARGET – имя платформы на которую устанавливается AliRoot;

PLATFORM – аналогично ALICE\_TARGET, но для GEANT3.

После того как переменные определены, сначала устанавливается GEANT3. Заходим в директорию \$ALICE/geant3 и делаем make. Затем устанавливаем AliRoot командой make в директории ALICE\_ROOT. Запуск AliRoot производится командой aliroot.

Как уже отмечалось, процесс моделирования в AliRoot начинается с моделирования первичного взаимодействия, затем идет реконструкция и анализ полученных данных. Моделирование взаимодействия производится встроенными в AliRoot генераторами PYTHIA, HIJING, ISAJET, FRITIOF. Для начала моделирования необходимо загрузить исходные параметры в AliRoot. Они загружаются из конфигурационного файла (макроса) Config.C. Config.C загружается по умолчанию из

директории, в которой был запущен AliRoot. Так же можно загрузить Config.C командой:

```
root[] gAlice->Init("path/Config.C");
```

Надо отметить, что все команды уже выполняются в среде AliRoot. Также в конфигурационном файле определяется величина магнитного поля,

```
AliMagFMaps* field = new AliMagFMaps("Maps","Maps", 2, 1., 10.,
smag);
field->SetL3ConstField(0); //Using const. field in the barrel
rl->CdGAFile();
gAlice->SetField(field);   есть ли размытие точки взаимодействия,
// Generator Configuration
gener->SetOrigin(0, 0, 0);   // vertex position
gener->SetSigma(0, 0, 5.3);  // Sigma in (X,Y,Z) (cm) on IP
position
```

```
    какие детекторы участвуют в моделировании события,
Int_t   iPIPE   = 1;
Int_t   iPMD    = 1;
Int_t   iHMPID  = 1;
Int_t   iSHIL   = 1;
Int_t   iT0     = 1;
Int_t   iTOF    = 1;
Int_t   iTPC    = 1;
```

описываются физические процессы, включенные для моделирования.

```
gMC->SetProcess("DCAY",1);
gMC->SetProcess("PAIR",1);
gMC->SetProcess("COMP",1);
gMC->SetProcess("PHOT",1);
gMC->SetProcess("PFIS",0);
gMC->SetProcess("DRAY",0);
gMC->SetProcess("ANNI",1);
```

Запуск моделирования взаимодействия производится командой:

```
root[1] gAlice->Run(10);
```

В данном примере осуществляется моделирование 10 взаимодействий. В результате выполнения этой команды создаются файлы, содержащие информацию о хитах. Однако конечной целью моделирования взаимодействия является информация об отклике детектора в формате raw или root. Класс AliSimulation является основным классом в

AliRoot, ответственным за моделирование. Ниже приведен пример простейшего макроса для запуска полной цепочки моделирования.

```
AliSimulation sim;  
  
sim.WriteRawFiles("TRD PMD");  
  
sim.ConvertRawFilesToDate("raw.date");  
  
sim.ConvertDateToRoot("raw.date", "raw.root");
```

Здесь выбраны детекторы TPC PMD для моделирования, выходные данные будут записаны в формате raw. Последняя строка преобразует данные из raw формата в root формат. Следующим этапом моделирования является реконструкция. Макрос для запуска реконструкции приведен ниже.

```
AliReconstruction rec;  
  
rec.SetInput("./");  
rec.Run();
```

Необходимо помнить, что реконструкция будет производиться, если есть raw данные. Ниже приведены два примера запуска реконструкции в зависимости от формата raw данных.

```
AliReconstruction rec;  
rec.Run("raw.date");
```

или

```
AliReconstruction rec;  
rec.Run("raw.root");
```

После реконструкции создаются ESD файлы, содержащие физическую информацию. На заключительном этапе следует анализ полученной информации. Он осуществляется в зависимости от поставленной задачи различными пользователями и рабочими физическими группами.

[1] <http://aliceinfo.cern.ch/>

[2] <http://aliceinfo.cern.ch/Offline>

[3] <http://aliceinfo.cern.ch/static/offlineold/fluka/Welcome.html>

[4] <http://geant4.web.cern.ch/geant4/>