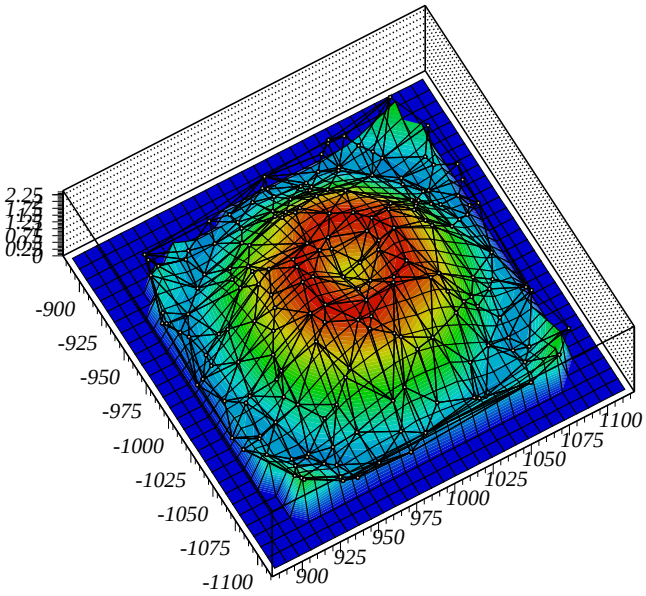
ROOT

© Е.М. Балдин*



Del Trirx ry rz 200

Эта статья была опубликована в сентябрьском номере русскоязычного журнала Linux Format (`http://www.linuxformat.ru`) за 2006 год. Статья размещена с разрешения редакции журнала на сайте `http://www.inp.nsk.su/~baldin/` и до февраля месяца все вопросы с размещением статьи в других местах следует решать с редакцией Linux Format, а после все вопросы следует решать со мной.

Текст, представленный здесь, не является точной копией статьи в журнале. Текущий текст в отличии от журнального варианта корректор не просматривал. Все вопросы по содержанию, а так же замечания и предложения следует задавать мне по электронной почте `mailto:E.M.Baldin@inp.nsk.su`.

Текст на текущий момент является просто *текстом*, а не книгой. Поэтому результирующая доводка в целях улучшения восприятия текста не проводилась.

*e-mail: E.M.Baldin@inp.nsk.su

Скрипт, который создал картинку, взят с `http://paw.web.cern.ch/paw/contributions/`. Автор скрипта Luke Jones.

Оглавление

1 Runing ROOT

Мало данные получить — надо ещё понять, а есть ли от них польза.

1.1 Введение

Даже если данных много — их надо как-то проанализировать. Это может сделать только человек. Компьютер в этом деле только помощник. Выбор инструмента очень важен. ROOT — хороший инструмент. У него был достойный предок и он мог бы быть гораздо лучше. Но здесь и сейчас надо анализировать данные, фиксируя недостатки, дабы исправить их в будущем. Это возможно, потому что ROOT — это свободный продукт.

Примерно через десять лет после возникновения PAW (Physics Analysis Workstation) стало скучно и лидер команды PAW Рене Бран (René Brun) с сотоварищами начал новый проект ROOT — An Object Oriented Data Analysis Framework[1].

Компьютеры стали много мощнее, но и поток данных увеличился. ROOT стал разрабатываться в рамках эксперимента NA49, где поток данных за один заход[2] мог превышает 10 Тб (грубо 1 Тб=10^3 Гб=10^6 Мб).

С начала 2006 года ROOT (`http://root.cern.ch/`) стал выпускаться под лицензией GNU, и, возможно, скоро попадёт во все основные дистрибутивы GNU/Linux.

1.2 Сравнение с PAW

PAW является предком ROOT, если уж не в смысле кода, то уж в смысле реализации идей заведомо. Поэтому полезно понять чем эти пакеты отличаются и в чём совпадают. Сравнительная таблица не претендует на фундаментальность, а просто отражает личные пристрастия автора.

Абсолютно вся функциональность, которая есть в PAW, присутствует и в ROOT. Для того чтобы файлы данными, сделанными для PAW, можно было проанализировать в ROOT, вместе с пакетом поставляется программа `h2root`:

```
> h2root <PAW rz-файл> <ROOT файл>
```

[1] Почему ROOT так называется? У меня есть только догадки: OO — видимо, Object Oriented, а сам ROOT от английского корень или источник (root). Предполагается, что ROOT — это база для разработок, а не просто система анализа, то есть корень всех ldots :) .

[2] Таких заходов было чуть меньше 5 тысяч.

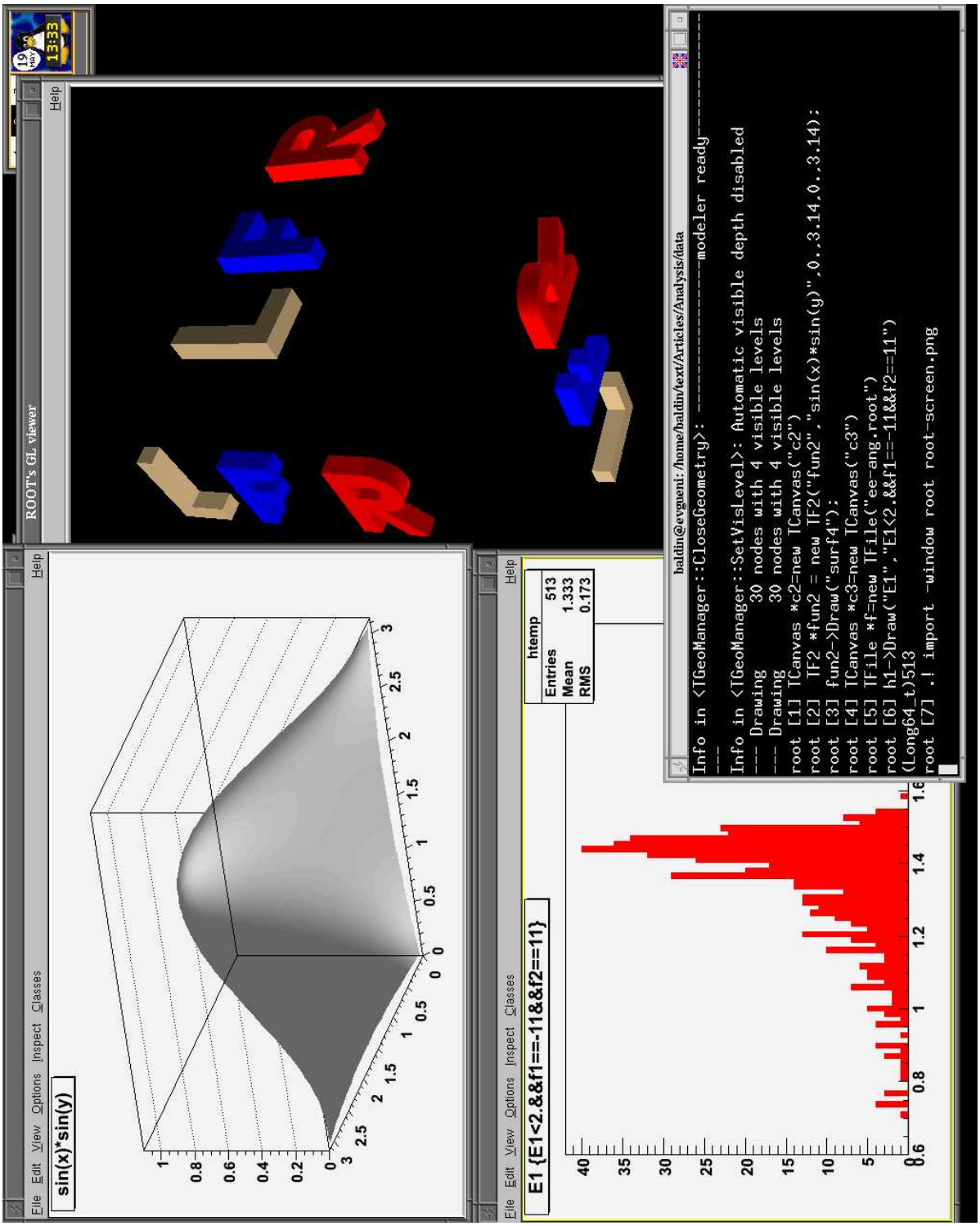


Рис. 1.1. root в действии — просто демонстрация

Признак	PAW	ROOT
Авторы	Рене Бран и др.	Рене Бран и др.
Возраст	20 лет	чуть больше 10 лет
GNU	начиная с 2000 года	с начала 2006 года
Интерпретатор	FORTRAN (COMIS)	C++ (CINT)
Командный процессор	KUIP	C++ (CINT)
Ускорение набора команд	сокращение команд	TAB-complition
Кириллица?	никак	аналогично
Состояние	матёрая, но немного устаревшая система	надо повзрослеть, хотя пора бы уже
Система помощи	Подробная официальная документация в почти 500 страниц, help в командной строке	Пухлое руководство пользователя, автодокументация по исходникам, но нет help — Аааaa.
Что бы сказал Брукс?	старая школа	все признаки «второй системы»
Что есть?	то, что нужно для анализа данных	это, плюс много чего ещё лишнего и не очень

Таблица 1.1. Сравнение PAW и ROOT (IMHO)

Почему PAW? Если в вашем проекте PAW уже используется особых причин для смены инструмента нет. Для стандартных операций анализа PAW использовать значительно проще чем ROOT. Это плата за попытку объять необъятное.

Почему ROOT?[3] C++ популярнее FORTRAN и KUIP. C++ привычнее и с помощью него проще делать задачи, которые являются вспомогательными к анализу — для всего используется один инструмент. ROOT активно поддерживается и развивается. У ROOT есть довольно мощное сообщество. На сайте `http://root.cern.ch` можно найти ответ почти на все вопросы, касающиеся пакета, в RootTalk (там же) можно задать вопрос любой сложности, на который ответят с очень большой вероятностью.

1.3 Запускаем ROOT

Так как ROOT получил лицензию LGPL совсем недавно, то, скорее всего, в вашем настольном дистрибутиве его нет. Поэтому запуск придётся отложить «на потом» после сборки и установки.

Брать исходники лучше всего с основного сайта: `http://root.cern.ch`. После распаковки дерева пакетов следует внимательно изучить инструкцию `README/INSTALL`.

[3] Я слышал такой вариант ответа: «потому что в отличии от PAW в графическом окне ROOT можно мышкой подправить экспериментальные данные» — очень надеюсь, что это была шутка.

Сборка стандартная:

```
> ./configure --prefix=/usr/local ; make ; make install
```

`make install` необходимо делать под пользователем root.

Можно попробовать собрать `rpm` или `deb` пакет. Собрать deb-пакет под Debian 3.1 (Sarge) без дополнительных телодвижений не удаётся, так как отсутствует пакет, на который указывают зависимости. По видимому, разработка ведётся для тестовой или нестабильной ветки дистрибутива.

После установки перед запуском необходимо установить переменные окружения. Для `bash`, это будет выглядеть примерно так:

```
> export ROOTSYS=/usr/local/
> export PATH=$PATH:$ROOTSYS/bin
> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ROOTSYS/lib/root
```

Установка `LD_LIBRARY_PATH` необходима из-за того, что почти весь функционал ROOT вынесен в разделяемые библиотеки, которые подгружаются во время работы программы. Аналогично можно оформить и свою библиотеку, расширив, таким образом, возможности ROOT.

Всё. Теперь открываем терминал и запускаем ROOT:

```
> root
   ******************************************
   *                                        *
   *        W E L C O M E  to  R O O T      *
   *                                        *
   *   Version   5.11/02     19 April 2006  *
   *                                        *
   *  You are welcome to visit our Web site *
   *          http://root.cern.ch           *
   *                                        *
   ******************************************

FreeType Engine v2.1.9 used to render TrueType fonts.
Compiled on 19 May 2006 for linux with thread support.

CINT/ROOT C/C++ Interpreter version 5.16.11, April 14, 2006
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```

Получив приглашение можно приступать к работе. Сказать «Hello World» из ROOT можно следующим образом:

```
root [0] cout << "Hello World" << endl;
Hello World
```

При запуска ROOT считывается файл настроек `.rootrc` сначала в текущей директории, если нет, то в домашней, а затем берётся системный файл `/etc/root/system.rootrc`. От версии к версии эта последовательность может меняться[4].

По умолчанию есть ещё три файла, которые могут управлять поведением программы:

- `rootlogon.C` — выполняется при запуске,
- `rootalias.C` — загружается при запуске, но не выполняется,
- `rootlogoff.C` — выполняется при завершении сеанса.

`root` можно запускать и не в интерактивном режиме. Для этого при запуске следует указать опцию `-b`. Полный список поддерживаемых опций можно получить при указании ключа `-h`.

Выйти из ROOT можно с помощью команды «`.q`». Если в процессе анализа удалось зациклить программу, то желание выйти можно усилить с помощью команд «`.qqq`», «`.qqqqq`» или «`.qqqqqqq`»[5]. `^C` так же может помочь в непредвиденных ситуациях.

1.4 «Командная логика»

В качестве командного процессора используется интерпретатор C++ CINT. Это означает, что интерактивная работа очень похожа на написание обычной программы. Знание языка C/C++ при «общении» с ROOT является обязательным. Как и для PAW напишем программу по вычислению чисел Фибоначчи:

```
root [0] {
end with '}', '@':abort > int a=0,b=1;
end with '}', '@':abort > cout << a << "␣" << b << "␣";
end with '}', '@':abort > for (int i=2;i<=10;i++) {
end with '}', '@':abort >  int x=a; a=b; b=x+b;
end with '}', '@':abort >  cout << b << "␣";
end with '}', '@':abort > }
end with '}', '@':abort > cout << endl;
end with '}', '@':abort > }
0 1 1 2 3 5 8 13 21 34 55
```

Команды группируются с помощью фигурных скобок. Этот же код можно сохранить в файл `fibonacci.cxx` и выполнить его как скрипт:

```
root [1] .x fibonacci.cxx
0 1 1 2 3 5 8 13 21 34 55
```

[4] В руководстве пользователя в этом месте присутствуют ошибки.

[5] Чем больше `q`, тем «сильнее» желание.

В случае C++ окончание команды отмечается «;». Если «;» опустить, то из ROOT получится неплохой калькулятор:

```
root [2] 2*sqrt(5)*sin(2*3.14*75/180)/3.14**2
(const double)2.27312089125660893e-01
root [3] 2**10
(const int)1024
root [4] 2.**1023
(const double)8.98846567431157954e+307
```

Все вспомогательные команды ROOT начинаются с точки («.»). Для выполнения команд оболочки используется команда «.!» за которой следуют shell-инструкции:

```
root [5] .! ls *.cxx
fibonacci.cxx
```

Полный список вспомогательных команд можно получить с помощью инструкции «.?».

Все необходимые для анализа объекты представлены в виде классов. Класс `TFile` соответствует файлу в который можно сохранять ROOT-структуры. Объект `TTree` представляет из себя более изощрённую реализацию идеи `ntuple`:

```
root [6] TFile *f=new TFile("ee-ang.root")
root [7] TTree *tree;
root [8] tree= (TTree *) f->Get("h1");
root [9] tree->Draw(«TAB»
void Draw(Option_t* opt)
Long64_t Draw(const char* varexp, const TCut& selection, Option_t*
option = "", Long64_t nentries = 1000000000, Long64_t firstentry = 0)
Long64_t Draw(const char* varexp, const char* selection, Option_t*
option = "", Long64_t nentries = 1000000000, Long64_t firstentry = 0)
root [10] tree->Draw("E1","E1<2.&&f1==-11&&f2==11")
```

В строке `[9]` после скобки была нажата клавиша «TAB», что привело к выводу подсказки по возможным командам. Недостаток команды `help` восполняется автоматически создаваемой подсказкой.

1.5 Графический интерфейс

Графическое окно в ROOT называется «канвой» (объект `TCanvas`). Можно открыть сколько угодно таких окон

```
//Создаём новую канву E1.
root [11] TCanvas *E1=new TCanvas("E1")
//Создаём новую канву cfunc.
root [12] TCanvas *cfunc=new TCanvas("func")
//Переходим в канву E1.
```

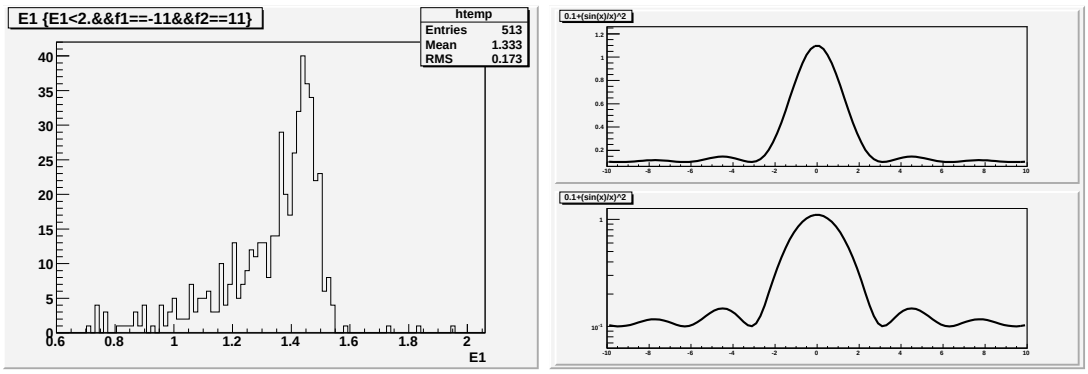


Рис. 1.2. Примеры графического представления гистограммы (канва E1) и функции (канва cfunc)

```
root [13] E1->cd();
//Рисуем гистограмму по параметру E1 с условием.
root [14] tree->Draw("E1","E1<2.&&f1==-11&&f2==11")
//Переходим в канву cfunc.
root [15] cfunc->cd()
//Делим канву cfunc на две части по Y.
root [16] cfunc->Divide(1,2)
//Создаём функцию.
root [17] TF1 f1("difr","0.1+(sin(x)/x)**2",-10,10)
//Переходим в верхнюю половину канвы cfunc.
root [18] cfunc->cd(1)
//Отображаем функцию.
root [19] f1->Draw()
//Переходим в нижнюю половину канвы cfunc.
root [20] cfunc->cd(2)
root [21] f1->Draw()
//Устанавливаем для нижней половины канвы cfunc
//логарифмический масштаб для оси Y.
root [22] cfunc->cd(2)->SetLogy()
//Из канвы cfunc создаём векторный eps-файл.
root [23] cfunc->Print("root-cfunc.eps")
//Из канвы E1 создаём растровый png-файл.
root [24] E1->Print("root-E1.png")
```

В отличии от своего предка PAW ROOT позволяет интерактивно менять параметры картинки с помощью по правому клику выпадающих меню. В зависимости от того на какой объект направлен указатель мыши тип меню меняется. Так же с помощью левой кнопки можно изменять интерактивно масштаб графика. Для возврата в исходное состоянии в меню относящееся к выбранной оси следует выбрать команду `UnZoom`.

Не стоит этим увлекаться, так как кажущаяся простота интегрально увеличивает время, потраченное на создание картинок. В отличии от набранных команд, движение и клики мыши сохранить для повторного использования не возможно, точнее не осмысленно.

1.6 Базовые объекты

ROOT унаследовал все базовые объекты анализа, которые были в PAW. Но в отличии от PAW ROOT не ограничивается исключительно анализом. Примером такого подхода, например, служит включение в пакет операций для работы с матрицами (линейная алгебра) и базовых средств для манипуляции объектов OpenGL (отображение физических объёмов). ROOT претендует на нечто большее, чем быть просто пакетом анализа, но всё-же в этом разделе будут перечислены только те объекты, которые могут пригодиться для представления данных.

1.6.1 Гистограммы

Гистограмма является одним из основных объектов анализа. По сравнению с PAW в ROOT было добавлено больше типов гистограмм. Конструктор гистограмм имеет вид `TH1F`. Для двумерной гистограммы вместо 1 надо подставить 2, а для трёхмерной (да, такие тоже есть, правда не понятно как их смотреть) — 3. `F` означает что на один бин используется `Float_t`, аналогично возможны и другие типы переменных для хранения значения в бине.

```
//Создаём новую канву.
root [25] TCanvas *ch=new TCanvas("Hist Test","Hist")
//Создаём гистограмму в 100 бинов от -3. до 3.
root [26] TH1F *h = new TH1F("h","Hist Test",100,-3.,3.)
//Обычно гистограммы заполняются с помощью метода Fill.
root [27] h->Fill(«TAB»
Int_t Fill(Double_t x)
Int_t Fill(Double_t x, Double_t w)
Int_t Fill(const char* name, Double_t w)
//Но мы сейчас идём другим путём:
//  а) создаём функцию G,
root [28] TF1 *func = new TF1("G","exp(-x**2)",-3,3)
//  б) заполняем гистограмму случайным образом
//     по форме функции G.
root [29] h->FillRandom("G",1000)
//Меняем цвет гистограммы.
root [30] h->SetFillColor(45)
//Подгоняем гистограмму распределением Гаусса
root [31] h->Fit("gaus")
  ...
```

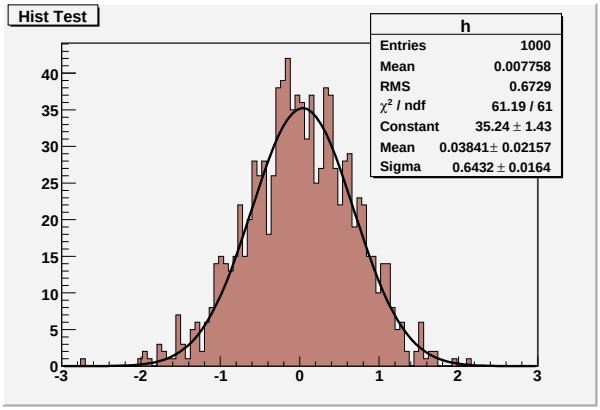


Рис. 1.3. Пример подогнанной гистограммы.

```
//Сохраняем полученную картинку.
root [32] ch->Print("root-histexample.eps")
```

Подгонкой «заведует» всё тот-же Minuit, что был и в PAW, правда переписанный на C++. Алгоритмы не поменялись.

1.6.2 Деревья

Деревья (tree) в ROOT — это логичное развитие идеи ntuple. ntuple, по сути дела, был таблицей со столбцами переменных типа `float`. В случае деревьев этого ограничения не существует, и в дереве можно сохранять любые объекты.

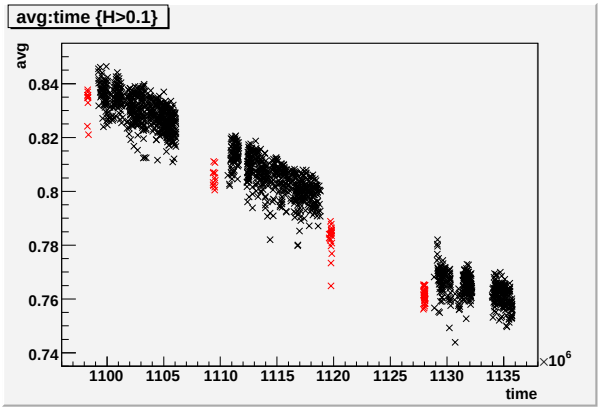


Рис. 1.4. Графическое представление дерева lkravg. Видно, что красные точки в среднем ниже чёрных, что и требовалось доказать.

```
//Создаём файл на диске.
root [33] TFile *f = new TFile("lkravg.root","RECREATE")
//Заводим новое дерево
root [34] TTree *lkravg = new TTree("lkravg","LKr degrad")
```

```
//Считаем файл lkravg.dat - тот самый, что "мучили" в
//статье про PAW
root [35] Long64_t nlines = lkravg->ReadFile("lkravg.dat",
                                    //список переменных
                                    "time:run:avg:avg_er:P:H")
root [36] cout << "Number of lines: " << nlines << endl
//Рисуем картинку: чёрные маркеры - есть магнитное поле,
//красные маркеры - нет магнитного поля.
root [37] lkravg->SetMarkerStyle(5)
root [38] lkravg->Draw("avg:time","H>0.1")
root [39] lkravg->SetMarkerColor(kRed)
root [40] lkravg->Draw("avg:time","H<=0.1","same")
//Пишем дерево в файл и закрываем файл.
root [41] lkravg->Write();
root [42] f->Close();
//Теперь этот файл можно открыть
root [43] TFile *f2 = new TFile("lkravg2.root")
//и посмотреть что в нём есть - дерево сохранилось.
root [44] .ls
TFile**         lkravg2.root
 TFile*         lkravg2.root
  KEY: TTree    lkravg;1        LKr degrad
```

В ROOT есть множество способов как создать и заполнить дерево. Подробности лучше посмотреть в пользовательской документации.

1.6.3 Функции

Как и в PAW в ROOT есть мощная поддержка функций как объектов. С помощью метода `Fit` можно подогнать гистограмму или график. Но до этого следует определить функцию, например, так:

```
//Файл mandel.cxx Множество Мандельброта
Double_t mandel(Double_t *XP,Double_t *par) {
  const Int_t nmax=30;
  Double_t xx=0.,yy=0.,tt,x,y;
  x=XP[0];y=XP[1];
  for (Int_t n=1;n<nmax;n++) {
    tt=xx*xx-yy*yy+x;
    yy=2.*xx*yy+y;
    xx=tt;
    if (xx*xx+yy*yy>4.) break;
  }
  return Double_t(n)/Double_t(nmax);
}
```

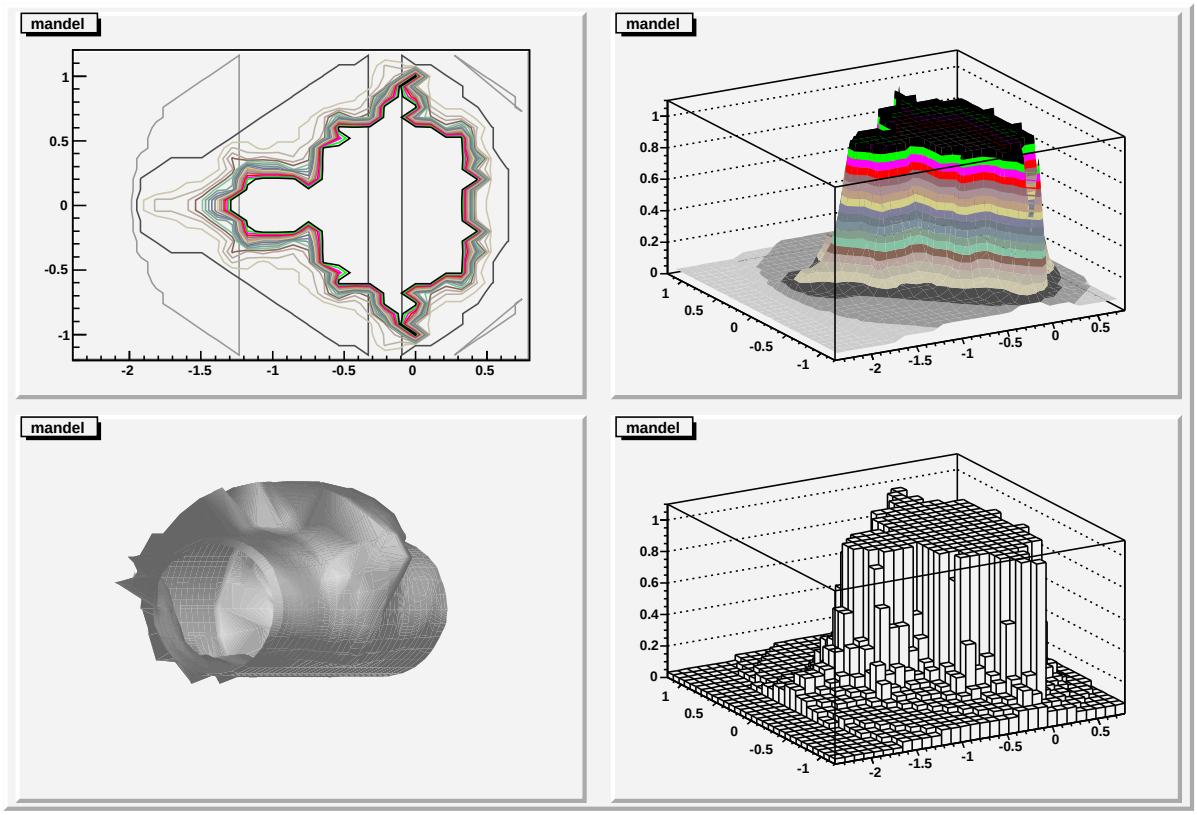


Рис. 1.5. Примеры графического представления двумерной функции или гистограммы. Множество Мандельброта

Текст функции следует сохранить в файле `mandel.cxx`. После с ним можно работать из ROOT:

```
//Загружаем описание функции mandel.cxx.
//Теперь можно обращаться к функции.
root [45] .L mandel.cxx
root [46] TCanvas *cm=new TCanvas("mandelbrot","Mandelbrot")
//Создаём объект «двумерная функция» TF2
root [47] TF2 *Mandelbrot=new
                  TF2("Mandelbrot",mandel,-2.4,.8,-1.2,1.2,0)
root [48] cm->Divide(2,2)
root [49] cm->cd(1)
root [50] Mandelbrot->SetNpx(«TAB»
void SetNpx(Int_t npx = 100)      // *MENU*
//Увеличиваем число шагов отображения.
//Как и в PAW функции отображаются через гистограммы.
root [51] Mandelbrot->SetNpx(200)
root [52] Mandelbrot->SetNpy(200)
//Контурное графическое представление.
```

```
root [53] Mandelbrot->Draw("cont")
root [54] cm->cd(2)
//Графическое представление в виде поверхность.
root [55] Mandelbrot->Draw("surf2")
root [56] cm->cd(3)
//Множество Мандельброта в цилиндрических координатах.
root [57] Mandelbrot->Draw("surf4cyl")
root [58] cm->cd(4)
//Графическое представление в стиле LEGO.
root [59] Mandelbrot->Draw("lego")
root [60] cm->Print("root-mandel.eps")
```

1.6.4 Графики

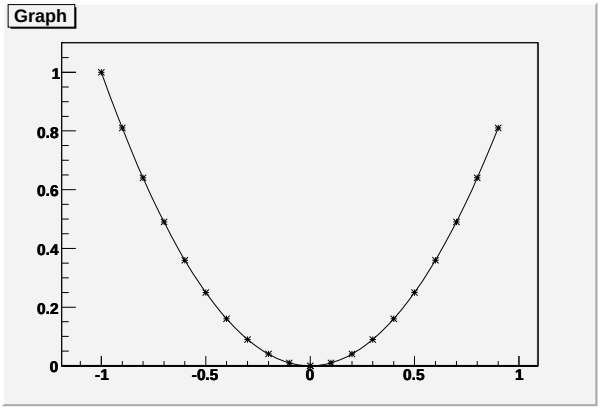


Рис. 1.6. Пример `TGraph`

Следует обратить внимание на ещё один полезный класс объектов — `TGraph`. `TGraph` — это графический объект, создаваемый из двух массивов одинаковой длины для оси абсцисс и оси ординат, соответственно.

```
root [61] Int_t n = 20;
root [62] Double_t x[n], y[n];
root [63] for (Int_t i=0;i<n;i++) {
end with '}', '@':abort > x[i]=(i-10)*0.1;
end with '}', '@':abort > y[i]=x[i]**2;
end with '}', '@':abort > }
//Создаём график.
root [64] TGraph *gr = new TGraph(n,x,y);
root [65] gr->Draw("ACP*")
```

Если добавить ещё два массива ошибок, то это уже будет `TGraphError`. Графики можно подгонять точно так же, как и гистограммы с помощью метода `Fit`.

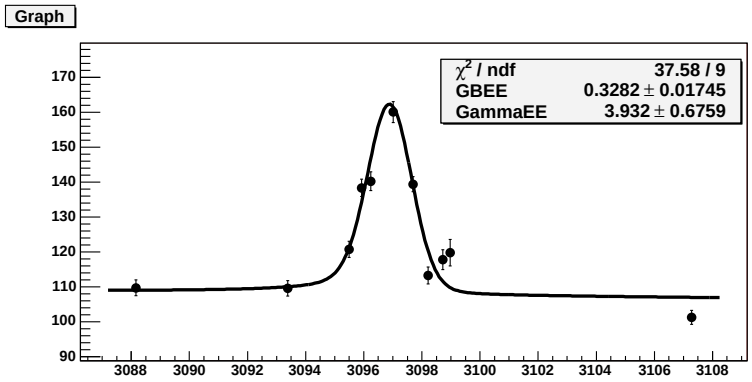


Рис. 1.7. Примеры TGraphError и подгоночной функции. χ^2 большой, поэтому данные надо «почистить».

1.7 Интерпретатор C++ (CINT)

Интерпретатор C++ или CINT, который используется в ROOT был независимым проектом. Сейчас он является составной частью ROOT, но его можно использовать и отдельно. Домашняя страничка cint расположена там же, где и ROOT: `http://root.cern.ch/root/Cint.html`.

CINT охватывает примерно 95% конструкций ANSI C и 85% от C++. Следует понимать, что полное соответствие стандартам никогда не было основной целью CINT. Не следует писать больших программ опираясь на интерпретатор, так как скорость выполнения команд уступает компилируемой версии программы примерно в десять раз. А где один порядок, там и два. Но для небольших скриптов для целей автоматизации анализа CINT вполне подходит, но для серьёзных целей надо писать обычные программы. Благо, абсолютно всё, что доступно в ROOT интерактивно, доступно и через библиотечные вызовы. Так уж ROOT сделан.

Для внешних CINT-скриптов есть две полезные команды:

```
//Выполняем скрипт script.cxx
root [66] .x script.cxx
//Загружаем функции, описанные в lib.cxx
root [67] .L lib.cxx
```

Одной из отличительных особенностей ROOT является возможность делать функции из внешних библиотек доступные для выполнения в скриптах CINT или интерактивно. Ниже будет приведён пример как подключить пользовательскую C-библиотеку.

Допустим существует C-библиотека в которой есть функции `myfunc2(char*)` и `myfunc1()`, и которые необходимо экспортировать в среду ROOT. Для этого нужно создать заголовочный файл `myfile.h` примерно следующего вида:

```
/*Файл myfile.h*/
#ifdef __cplusplus
```

```
extern "C" {
#endif
extern void myfunc1();
extern int myfunc2(char *);
#ifdef __cplusplus
}
#endif
```

Пока всё как обычно. Для того чтобы экспортировать функции в ROOT, необходимо создать ещё один заголовочный файл `myfileLinkDef.h` (к `myfile` добавляется `LinkDef`):

```
/*Файл myfileLinkDef.h */
#ifdef __CINT__
#pragma link C++ function myfunc1();
#pragma link C++ function myfunc2(char*);
#endif
```

Так же можно экспортировать и структуры, подставив вместо слова `function` слово `struct`. После создания описанных заголовочных файлов необходимо сгенерить «словарик»:

```
> rootcint -f myfileDict.cxx -c myfile.h myfileLinkDef.h
```

В результата будут созданы файлы `myfileDict.h` и `myfileDict.cxx`.

Далее нужно собрать саму библиотечку. Пусть для простоты вся библиотека представляет из себя один C-файл `myfile.c`:

```
# Компилируем myfile.c.
> gcc -c -fPIC myfile.c
# Компилируем словарик.
> g++ -c -fPIC `root-config --cflags` myfileDict.cxx
# Создаём разделяемую библиотеку.
> g++ -shared -o myfile.so myfile.o myfileDict.o
```

Теперь эту вновь созданную библиотеку можно загрузить в ROOT для интерактивной работы:

```
root [68] gSystem->Load("myfile")
root [69] myfunc1()
root [70] Int_t icount=myfunc2("string")
```

Это далеко не единственный способ подключить пользовательскую библиотеку к ROOT. Для компиляции скриптов можно использовать подсистему ACLiC.

P.S. Кроме CINT в среде ROOT можно использовать скрипты, написанные на Python или Ruby. И наоборот: из этих языков можно общаться с библиотеками ROOT. К сожалению описание этих механизмов выходит за рамки этой статьи.

1.8 Заключение

В этой статье ROOT не описан — здесь только собраны какие-то штрихи к портрету. Для более подробного знакомства с этим программным продуктом настоятельно рекомендуется посетить `http://root.cern.ch`.

ROOT это не просто инструмент анализа — это среда для генерации таких инструментов. Он, возможно, неуклюж и избыточен, но гибок и очень легко расширяем. Это не идеал, но идеал, скорее всего, будет на него похож.