Lecture 7

Simulations and Event Generators

Exercise 1

Write a macro which performs a benchmark comparison between TRandom, TRandom2, TRandom3. Compare the performance of the Gaus method of these classes. Also judge the randomness for 200 mln events by making a fit through the simulated data.

Exercise 1: Solution

```
void doit(UInt_t rgNr = 1, Int_t nrEvents = 200000000)
{
  if (gRandom) delete gRandom;
  switch (rgNr)
  {
  case (2)
    gRandom = new TRandom2(0);
    break;
  case (3)
    gRandom = new TRandom3(0);
    break;
  default:
    gRandom = new TRandom(0);
    break;
  }

  TH1D* hist=new TH1D("hist","TRandom",500,-10,10);

  TStopwatch *st=new TStopwatch();

  st->Start();
  for (Int_t i=0; i<nrEvents; i++) hist->Fill(gRandom->Gaus(0,1));
  st->Stop();

  TF1* gs = new TF1("gs","gaus",-10,10);
  hist->Fit("gs");
  Double_t normchi2 = gs->GetChisquare()/gs->GetNDF();
  printf("%s : %.1fs %.2f mus/event %.4f\n",
    gRandom->GetName(), st->GetCpuTime(), st->GetCpuTime()/nrEvents, normchi2);
}
```

Exercise 1: Solution

Results for 200 mln events:

Random :	92.7 s	0.46 μs/event	2.0889
Random2:	135.6 s	0.68 μs/event	0.9439
Random3:	86.6 s	0.43 μs/event	0.9839

Exercise 1: Solution

Results for 200 mln events:

Random :	92.7 s	0.46 μs/event	2.0889
Random2:	135.6 s	0.68 μs/event	0.9439
Random3:	86.6 s	0.43 μs/event	0.9839

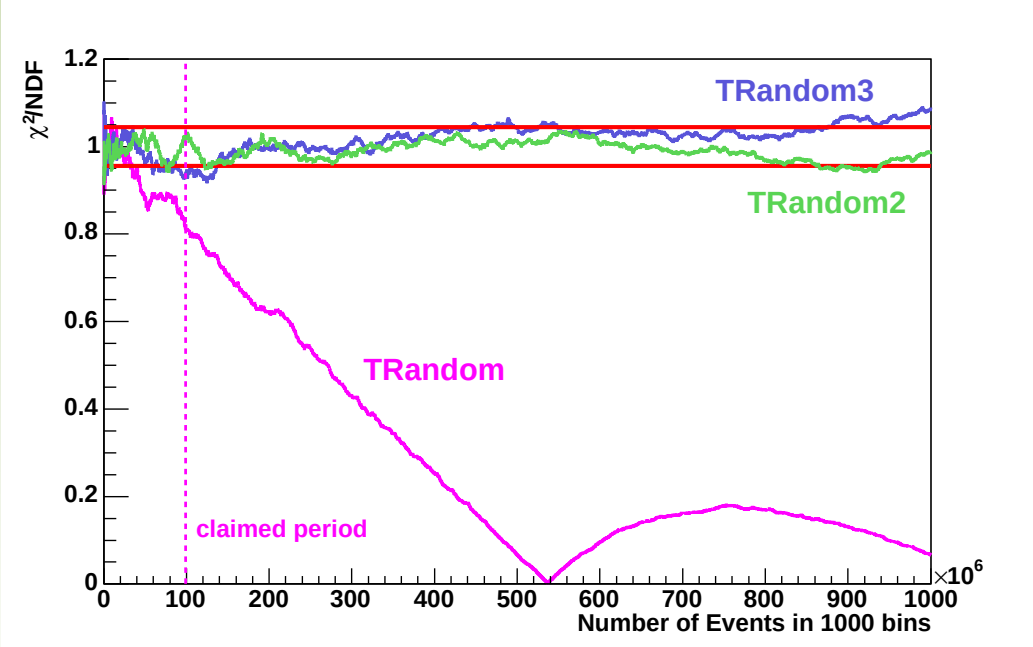
This seems to contradict Johan's findings last week, where he showed a small χ^2/NDF for TRandom. Here is why (unform distribution):

Exercise 1: Solution

Results for 200 mln events:

Random :	92.7 s	0.46 μs/event	2.0889
Random2:	135.6 s	0.68 μs/event	0.9439
Random3:	86.6 s	0.43 μs/event	0.9839

This seems to contradict Johan's findings last week, where he showed a small χ^2/NDF for TRandom. Here is why (unform distribution):



Exercise 1: Final Remarks

Be careful with random number generators

Always use TRandom3

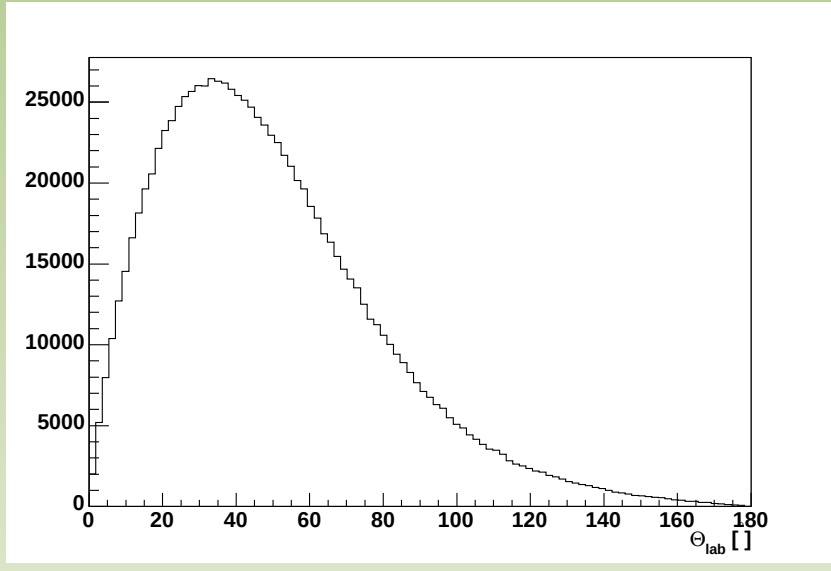
Exercise 2

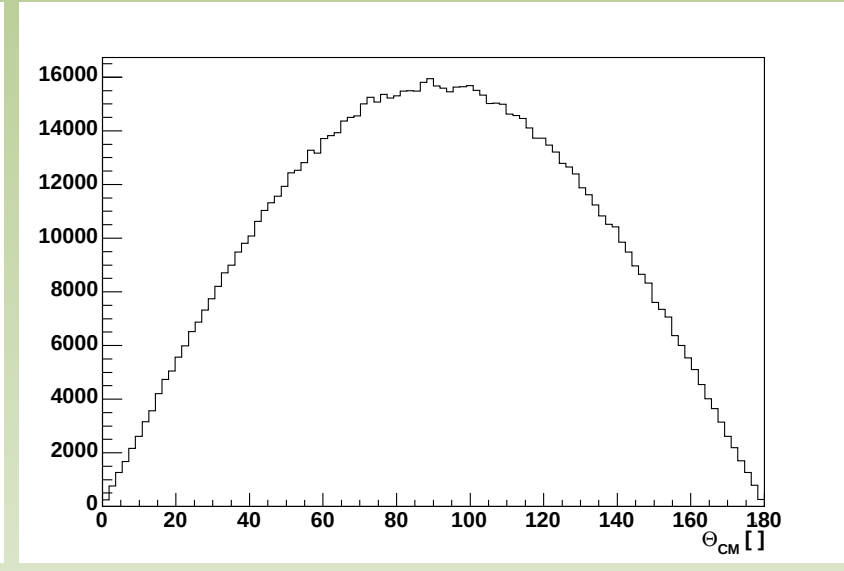
Write a macro which generates kinematically allowed events for the reaction p+d->p+p+n with an incident proton energy of 200 MeV and a deuteron at rest. Make a histogram of the scattering angle of the neutron in the lab. frame and in the center-of-mass frame.

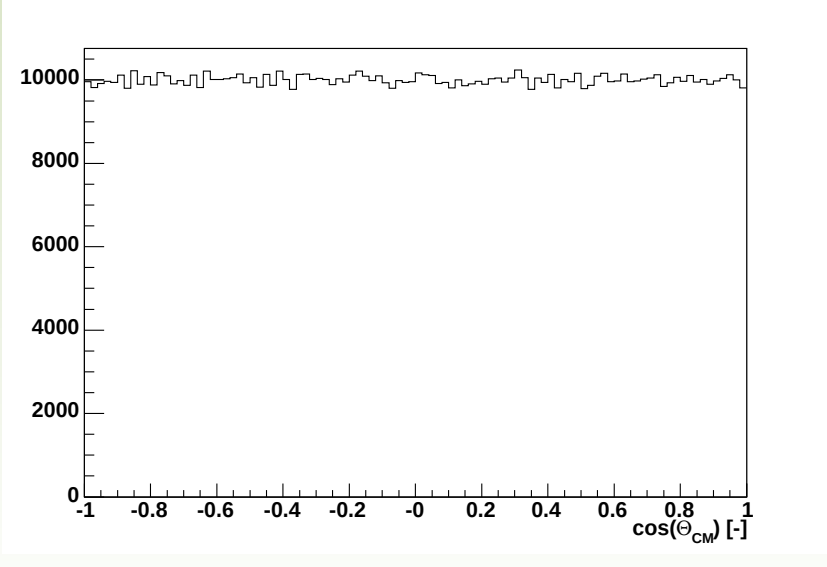
Exercise 2: Solution

```
static const Double_t md  = 1.87561282; // deuteron mass in GeV/c**2
static const Double_t mp  = 0.938272029; // proton mass in GeV/c**2
static const Double_t mn  = 0.939565360; // neutron mass in GeV/c**2

void zoem()
{
  // gSystem->Load("libPhysics");
  TLorentzVector target(0.0, 0.0, 0.0, md);
  Double_t pp = 0.2; // GeV/c
  TLorentzVector beam(0.0, 0.0, pp, sqrt(mp*mp+pp*pp));
  TLorentzVector W = beam + target;
  Double_t masses[3] = { mp, mp, mn};
  TGenPhaseSpace event;
  event.SetDecay(W, 3, masses);
  TH1D *h = new TH1D("his", "Theta_{lab}", 100, 0, 180);
  TH1D *k = new TH1D("kis", "Theta_{CM}", 100, 0, 180);
  TH1D *k2 = new TH1D("k2is", "cos(Theta_{CM})", 100, -1, 1);

  h->SetXTitle("#Theta_{lab} [^{#circ}]");
  k->SetXTitle("#Theta_{CM} [^{#circ}]");
  k2->SetXTitle("cos(#Theta_{CM}) [-]");

  for (Int_t n=0; n<1000000; n++)
  {
    event.Generate();
    TLorentzVector *pNeutron = event.GetDecay(2);
    h->Fill(pNeutron->Theta()*57.3);
    pNeutron->Boost(-W.BoostVector());
    k->Fill(pNeutron->Theta()*57.3);
    k2->Fill(cos(pNeutron->Theta()));
  }
  h->Draw();
}
```

Exercise 2: Solution







Lecture 8

Towards your own analysis code: ACLiC

Using ROOT for your analysis

With ROOT there are several ways to perform an analysis:

Method	complexity	speed	flexibility
By mouse	click-and-play	slowest	lowest
Line-by-line in CINT command line	“C++” and keyboard	slow	large
Via a macro interpreted by CINT	C++ and editor	medium	large
Using a shared library made by ACLiC	some ROOT insight	fastest	nearly maximum
With your own code linked with ROOT	programmer	fastest	maximum

This lecture, we’ll discuss the “shared library” option.

ACLiC

ACLiC — The Automatic Compiler of Libraries for CINT

What the ROOT developpers say about ACLiC:

" *Instead of having CINT interpret your script there is a way to have your scripts compiled, linked and dynamically loaded using the C++ compiler and linker. The advantage of this is that your scripts will run with the speed of compiled C++ and that you can use language constructs that are not fully supported by CINT. On the other hand, you cannot use any CINT shortcuts (see CINT extensions) and for small scripts, the overhead of the compile/link cycle might be larger than just executing the script in the interpreter.*

ACLiC will build a CINT dictionary and a shared library from your C++ script, using the compiler and the compiler options that were used to compile the ROOT executable. You do not have to write a makefile remembering the correct compiler options, and you do not have to exit ROOT. "

Before using ACLiC

Because your script is passed to the C++ compiler, you'll have to provide proper C++ source code. Some things to pay attention to:

No sloppy CINT-like constructs, like mixing objects and object-pointers,

`W->Theta()` $\neq$ `W.Theta()`

Also, you will have to properly terminate each line in your script

`W->Theta()` will not work

`W->Theta();` will work

Variables need to be declared before first use:

CINT: `Warning: Automatic variable z is allocated FILE:test.cxx LINE:3`

ACLiC: `test.cxx:3: 'z' undeclared (first use this function)`

Declaring all Classes

Also all classes you use need to be declared via their include-files.

This is what your script might look like:

```
#include "TH1.h"
#include "TRandom3.h"
 void test()
{
TH1D* h = new TH1D("h","h",1000,-10,10);
if (gRandom) delete gRandom;
gRandom = new TRandom3();
h->Fill(gRandom->Gaus(0,1));
}
```

Include-files for the ROOT classes can (typically) be found in $ROOTSYS/include ($ROOTSYS is where you installed ROOT).

Creating a shared library

Once your code is ready, simply load the script with a “+” behind the name:

```
root [0] .L test.cxx+
Info in <TUnixSystem::ACLiC>: creating shared library ./test_cxx.so
```

Now just proceed as before ....

... or start debugging ...

What happens when you call ACLiC

① ACLiC will call `rootcint` to create a CINT dictionary so that you can use all functions and classes in your script on the command line.

② ACLiC will pass the file, *e.g.* test.cxx, to the C++ compiler with all the flags used to compile ROOT (such as debugging or optimization options)

③ A shared library file is created by adding the proper file extension (platform dependent). In Linux a file test_cxx.so is made.

④ Also a file test_cxx.d is made, which lists the dependencies of your shared library (there are many!!!)

Some options/alternatives worth mentioning

ROOT[0] .L test.cxx+ will recompile if needed
ROOT[0] .L test.cxx++ will always recompile
ROOT[0] .x test.cxx++ will recompile and execute
ROOT[0] .x test.cxx++ (4000) recompile and execute with arg. 4000
ROOT[0] .x test.cxx+g will recompile with debug symbols (*c.f.* -g)
ROOT[0] .x test.cxx+O will recompile with optimizations (*c.f.* -O)
ROOT[0] .L test_cxx.so load previously compile code
gSystem->SetAclicMode(TSystem::kDebug) set default to debug
gSystem->SetAclicMode(TSystem::kOpt) set default to optimizations
gROOT->ProcessLine("test.cxx+") call ACLiC from another script
gROOT->LoadMacro("test.cxx+) the same
gSystem->AddIncludePath(" -I/tmp/lala ") add “/tmp/lala” to include path
gSystem->SetIncludePath(" -I/tmp/lala ") make “/tmp/lala” include path

WARNING: do NOT give your script a “.c” extension

Moving between Interpreter and Compiler

If you program properly, your scripts will always run with both

If you feel a need to use CINT limitations, you can do so using the variables `__CINT__` and `__MAKECINT__` to comment in/out pieces of code (see user manual).

My suggestion: don’t do it!! Or don’t ask for help ...

Exercises

(1) use ACLiC to compile the scripts you used for the exercises in the previous lecture. How much acceleration do you get?

(2) Complete the script below. Determine the speed (μs/event) for each of the 5 methods to fill a histogram when you run it in CINT or ACLiC mode. Comment on the difference ...

```
Double_t mygaus(Double_t* c, Double_t* par)
{
Double_t x = c[0];
return par[0]*exp(-0.5*(x-par[1])*(x-par[1])/par[2]/par[2]);
}

void compare()
{
TF1* gs1 = new TF1("gs1","gaus",-10,10); gs1->SetParameters(1,0,1);
TF1* gs2 = new TF1("gs2",mygaus,-10,10,3); gs2->SetParameters(1,0,1);

hist1->FillRandom(g1,nEvents); // method 1
hist2->FillRandom(g2,nEvents); // method 2
for(Int_t i=0;i<nEvents;i++) hist3->Fill(gRandom->Gaus(0,1); // method 3
for(Int_t i=0;i<nEvents;i++) hist4->Fill(gs1->GetRandom()); // method 4
for(Int_t i=0;i<nEvents;i++) hist5->Fill(gs2->GetRandom()); // method 5
}
```