Pracownia Specjalistyczna

Wprowadzenie do pakietu ROOT
Część 3

•pliki binarne
•jak unikać powtórzeń w kodzie
•pliki ROOT - czytanie
•funkcje
•fitowanie funkcji

dr Katarzyna Rusiecka
Zakład Fizyki Hadronów IF UJ
katarzyna.rusiecka@uj.edu.pl

Pliki binarne

- Najprostsza definicja pliku binarnego mówi, że jest to plik, który nie jest plikiem tekstowym
- Pliki binarne mają zwykle mniejszy rozmiar i są szybciej odczytywane
- Nie można ich otworzyć i edytować w edytorach tekstowych
- Często używane do przechowywania dużej ilości danych, które nie muszą być modyfikowane

Otwieranie plików binarnych

```
fstrem myfile;
myfile.open(”logfile.dat”, ios::binary | ios::in);
```

Czytanie plików binarnych

```
float* x;
char * temp = new char[sizeof(float)];

while(!myfile.eof()){
   myfile.read(temp, sizeof(float));
   x = (float*)temp;
   cout << *x << endl;
}
```

```
float x;

while(!myfile.eof()){
   myfile.read((char*)&x, sizeof(float));
   cout << x << endl;
}
```

Pliki binarne

Zapisywanie do plików binarnych

```
fstrem myfile;
myfile.open(”logfile.dat”, ios::binary | ios::out);

float x;

myfile.write((char*)&x, sizeof(float));
```

Zadanie:

- Napisz krótkie makro, które będzie zapisywało jedną liczbę do pliku binarnego
- Sprawdź zawartość utworzonego pliku binarnego za pomocą programu hexdump:

```
$ hexdump –C plik.bin
```

- Przepisz plik Cs-137.dat z formatu tekstowego do binarnego. Sprawdź zawartość pliku.
- Zmodyfikuj makro WidmoCs.C tak, aby korzystało z danych binarnych zamiast tekstowych.

Jak unikać powtórzeń w kodzie?

Powtórzenia bloków kodu w makrach i programach:

- powstawanie trudnych do zlokalizowania i naprawienia błędów
- trudna edycja i modyfikowanie kodu
- długi i trudny do zinterpretowania kod
- mało elastyczny kod, trudny do przystosowania do nowych zastosowań

Sposoby na uniknięcie powtórzeń bloków kodu:

- stosowanie pętli (np. for)
- podział kodu na wyspecjalizowane funkcje
- w przypadku dłuższych i bardziej skomplikowanych projektów – wprowadzenie struktur i klas

Przykład: makro Histogramy.C (MS Teams, zakładka Pliki → Materiały z zajęć → ROOT_3)

Pliki ROOT

Konstruktor pliku ROOT - przypomnienie

```
TFile *file = new TFile(”filename.root”,”OPCJA”);
```

Zapisywanie obiektów do pliku ROOT - przypomnienie

```
TFile *file = new Tfile(”mojplik.root”,”RECREATE”);

hGauss->Write();
gEnergiaVsKanal->Write();
canvas->Wite();
file->Close();
```

Odczytywanie plików ROOT

```
TFile *file = new TFile(”mojplik.root”, ”READ”);

if(!file->IsOpen()){
  cout << ”Nie otwarto pliku!” << endl;
  return;
}

TH1F *hist = (TH1F*)file->Get(”hGauss”);
TGraph *gr = (TGraph*)file->Get(”gEnergiaVsKanal”);

hist->SetLineColor(kRed);
gr->SetMarkerStyle(8);
...
```

Opcja	Opis
NEW or CREATE	Stwórz nowy plik i otwórz go w celu zapisu do pliku. Jeśli plik już istnieje, nie zostanie otwarty.
UPDATE	Otwórz istniejący plik w celu zapisu do pliku. Jeśli plik nie istnieje, zostanie utworzony.
READ	Otwórz istniejący plik do odczytu (opcja domyślna).
RECREATE	Stwórz nowy plik. Jeśli plik już istnieje zostanie nadpisany.

Funkcja - konstruktory

TF1 konstruktor – predefiniowana funkcja ROOT

```
TF1 *fun = new TF1(”name”,”function”,low,up)
```

`name` – nazwa funkcji
`function` – nazwa predefiniowanej funkcji ROOT, np. gaus, expo, pol2 itd...
`low` – dolny zakres wartości dla funkcji
`up` – górny zakres wartości dla funkcji

Przykład

```
TF1 *fun = new TF1(”fun”,”expo”,0,10)
```

TF1 konstruktor – prosta formuła funkcji

```
TF1 *fun = new TF1(”name”,”formula”,low,up)
```

`formula` – wzór funkcji

Przykłady

```
TF1 *fun = new TF1(”fun”,”3*x*sin(2*x)”,0,10)
```

```
TF1 *fun = new TF1(”fun”,”[0]*x*sin([1]*x)”,0,10,2)
```

- Parametry funkcji są zapisywane w nawiasach kwadratowych i numerowane od 0
- Ostatni argument w podanym przykładzie konstruktora to liczba parametrów funkcji

Funkcja - konstruktory

TF1 konstruktor – funkcja zdefiniowana przez użytkownika

```
TF1 *fun = new TF1(”name”,myFunction,low,up,npar)
```

`myFunction` – funkcja zdefiniowana przez użytkownika
`npar` – liczba parametrów funkcji

• “myFunction” musi być zaimplementowane jako niezależna funkcja (w sensie programistycznym) w makrze.

Przykład

```
Double_t myFunc(Double_t *x, Double_t *par){

  Double_t f = par[0]*x[0] + sin(x[0]*par[1]);

  return f;
}
.
.
.
Bool_t FitMyFunc(void){

  TF1 *fun = new TF1(”name”,myFunc,0,10,2);
.
.
.
}
```

Funkcje - przykłady

```
#include <iostream>
#include <fstream>
using namespace std;

void DrawFun(void){

  TCanvas *can = new TCanvas("can","can",800,800);
  can->Divide(2,2);

  TF1 *fun1 = new TF1("fun1","gaus",-10,10);
  fun1->SetParameter(0,2);
  fun1->SetParameter(1,0);
  fun1->SetParameter(2,1);
  can->cd(1);
  fun1->Draw();

  TF1 *fun2 = new TF1("fun2","gaus(0)+pol1(3)",-10,10);
  fun2->SetParameters(2,0,1,0.1,0.1);
  can->cd(2);
  fun2->Draw();

  TF1 *fun3 = new TF1("fun3","sin(x)/x",0,15);
  can->cd(3);
  fun3->Draw();

  TF1 *fun4 = new TF1("fun4","[0]+cos(x*[1])",0,15);
  fun4->SetParameters(2,0.7);
  can->cd(4);
  fun4->Draw();

  return;
}
```

Fitowanie funkcji

Fitowanie bezpośrednie i przez wskaźnik

```
hist->Fit(fun,"option")
TFitResultPtr fitRes = hist->Fit(fun,"option")
```

`fun` – wskaźnik do funkcji (do obiektu klasy TF1)
`option` – opcja; przy fitowaniu przez wskaźnik zawsze należy dodać opcję "S"

Parametry startowe fitu

```
fun->SetParameter(npar,x)
fun->SetParameters(x,y,z...)
fun->SetParLimits(npar,min,max)
fun->FixParameter(npar,x)
```

`npar` – numer parametru
`x,y,z...` – wartości parametrów
`min,max` – dolna i górna granica dla parametru

Dostęp do wyników fitu

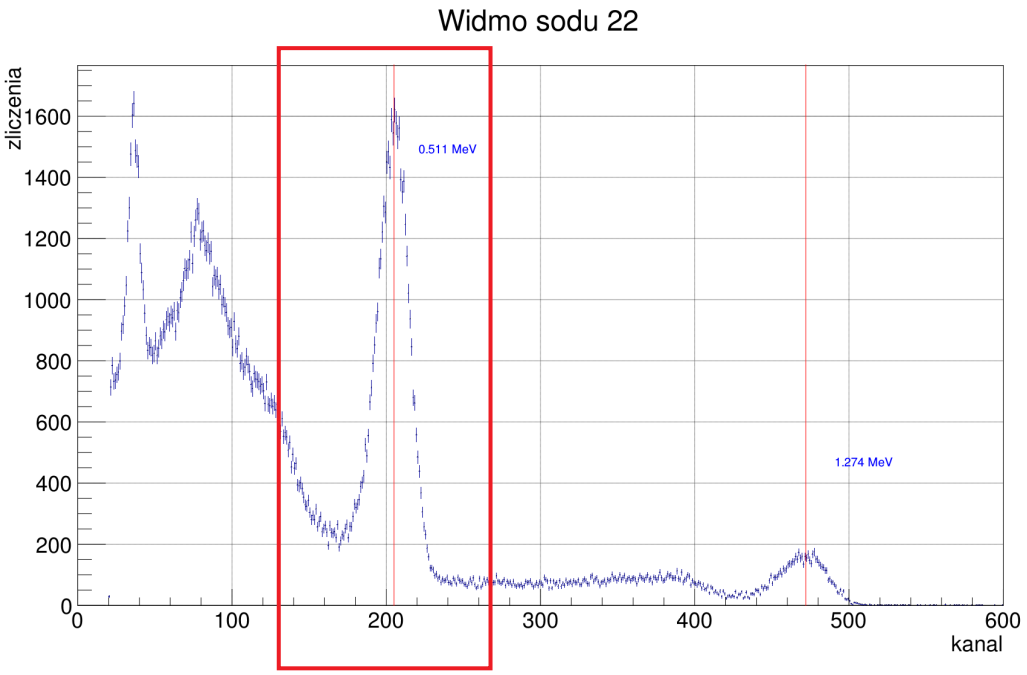
```
fun->GetParameter(n)
fun->GetParError(n)
fitRes->Chi2()
fitRes->Ndf()
fitRes->Edm()
```

Fitowanie funkcji - przykłady

```
void Fitowanie(void){

    TFile *file = new TFile("sample.root","READ");

    if(!file->IsOpen()){
      cout << "Could not open the file!" << endl;
      return;
    }

    TH1F *hist = (TH1F*)file->Get("histogram");

    TF1 *fun = new TF1("fun", "expo", 0, 10);
    fun->SetParameters(4,0.56);

    TFitResultPtr results = hist->Fit(fun, "S");

    cout << "Chi2 = " << results->Chi2() << endl;
    cout << "NDF = " << results->Ndf() << endl;
    cout << "EDM = " << results->Edm() << endl;

    hist->Draw();

    return;
}
```

Fitowanie funkcji - przykłady

```
Double_t MyFunc(Double_t *x, Double_t *par){

  if(x[0]>170 && x[0]<230)
    TF1::RejectPoint();
  Double_t f = TMath::Exp(par[0] + x[0]*par[1]);

  return f;
}

Bool_t Fitowanie(void){

  TH1F *hWidmoNa = new TH1F("hWidmoNa", "hWidmoNa", 1024, 0, 1024);

.
.
.

  TF1 *fun = new TF1("fun", MyFunc,
                     120, 280, 2);
  fun->SetParameters(2, 0.3);

  hWidmoNa->Fit(fun);

.
.
.

  return kTRUE;
}
```



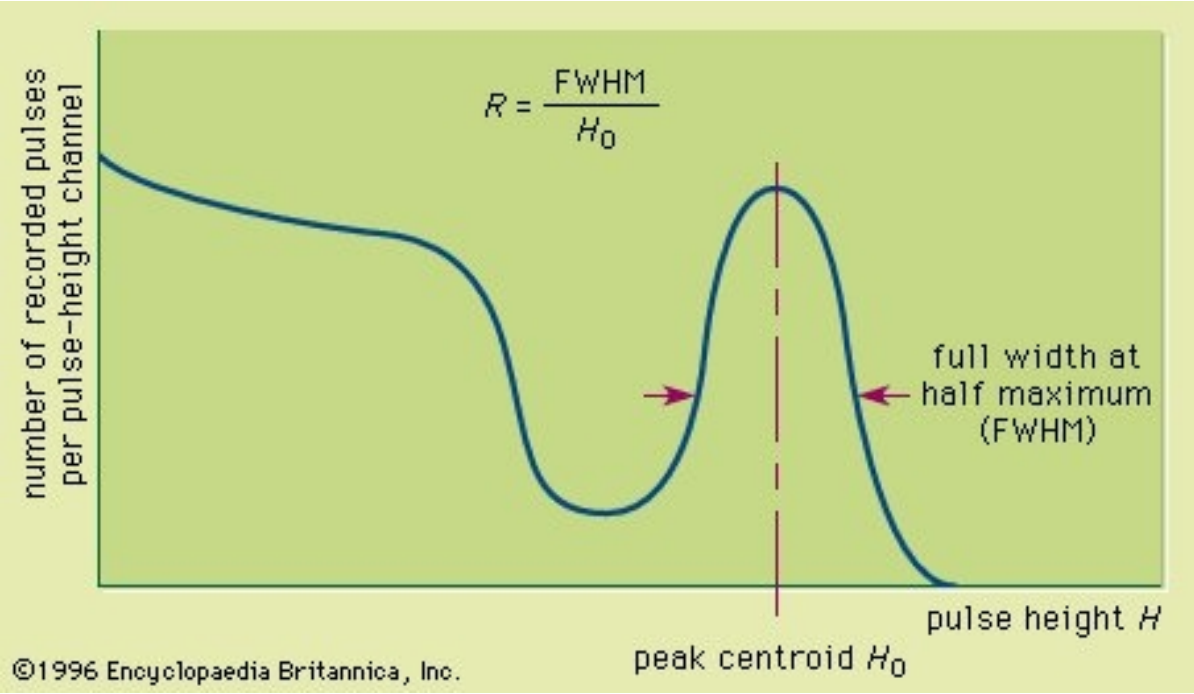
Zadanie

Wyznaczanie energetycznej zdolności rozdzielczej detektora

• Utwórz makro Fitowanie.C, w którym znajdzie się funkcja Fitowanie(). Niech argumentami funkcji będzie nazwa pliku ROOT, który będzie otwierany oraz nazwa histogramu, który będzie analizowany. Ustaw odpowiednie wartości domyślne argumentów (w naszym przypadku będą to plik i histogram utworzony w zadaniu z histogramami)
• Otwórz plik ROOT i sprawdź czy został poprawnie otwarty
• Pobierz z pliku ROOT histogram zawierający widmo cezu-137
• Utwórz funkcję, która opisze pik 662 keV oraz towarzyszące mu tło
• Ustal odpowiednie parametry startowe funkcji do fitu. Jak wyznaczyć takie parametry startowe?
• Dofituj funkcję do widma w wybranym zakresie (fit przez wskaźnik)
• Na podstawie uzyskanych parametrów fitu wyznacz energetyczną zdolność rozdzielczą i wypisz ją na ekran
• Zapisz wyniki fitu w pliku tekstowym (wartości parametrów i ich niepewności, Chi2, NDF, EDM). W pliku zapisz też wyznaczoną energetyczną zdolność rozdzielczą

Energetyczna zdolność rozdzielcza

Energetyczna zdolność rozdzielcza zdefiniowana jest jako stosunek szerokości połówkowej fotopiku do jego położenia. Typowa zdolność rozdzielcza detektorów scyntylacyjnych NaI(Tl) wynosi 8% dla fotonów o energii 1 MeV; dla detektora germanowego wynosi natomiast ok. 0,2%. Im lepsza (mniejsza) zdolność rozdzielcza, tym lepiej możemy rozróżnić cząstki lub kwanty o różnych energiach.



https://www.britannica.com/science/energy-resolution
http://ncbj.edu.pl/zasoby/wyklady/ld_stud_podypl/04.detekcja.pdf