Ćwiczenie 2
Pakiet ROOT – wprowadzenie cz. 2

1. Automatyczne tworzenie szkieletu programu

W poprzednim ćwiczeniu utworzyliśmy szkielet programu „ręcznie" – samodzielnie zadeklarowaliśmy zmienne, gałęzie, adresy oraz pętle. Metoda jest dobra w przypadku kilku zmiennych. Jednak gdy ich liczba idzie w dziesiątki (a czasem setki), to nie tylko jest to nużące, ale też łatwo się pomylić. W związku z tym pakiet ROOT udostępnia funkcję, która wygeneruje automatycznie szkielet kodu.

1. Ze strony praktyk należy ściągnąć i zapisać w katalogu `/media/ROOT_cw2` plik z danymi (`mc_sample1.root`): `wget http://atlas.ifj.edu.pl/praktyki/materialy/2016/mc_sample1.root` lub za pomocą przeglądarki.
2. Otworzyć plik (`root -l mc_sample1.root`) i wyświetlić jego zawartość (`.ls`).
3. Do tworzenia szkieletu kodu służy polecenie `MakeClass()`:
   `ntuple->MakeClass("cwiczenie2")`

Po wykonaniu polecenia, ROOT poinformuje o utworzenie dwóch plików: `cwiczenie2.h` and `cwiczenie2.C`. Wyjdźmy z ROOTa (`.q`) i przyjrzyjmy się strukturze plików.

1.1. Plik główny (cwiczenie2.C)

Plik zawiera:

- deklaracje bibliotek:

```
1  #define cwiczenie2_cxx
2  #include "cwiczenie2.h"
3  #include <TH2.h>
4  #include <TStyle.h>
5  #include <TCanvas.h>
```

- funkcję `Loop()` należącą do klasy `cwiczenie2`,
- w funkcji znajduje się pętla po przypadkach numerowanych zmienną `jentry`:

```
32     if (fChain == 0) return;
33
34     Long64_t nentries = fChain->GetEntriesFast();
35
36     Long64_t nbytes = 0, nb = 0;
37     for (Long64_t jentry=0; jentry<nentries;jentry++) {
38        Long64_t ientry = LoadTree(jentry);
39        if (ientry < 0) break;
40        nb = fChain->GetEntry(jentry);   nbytes += nb;
41        // if (Cut(ientry) < 0) continue;
42     }
```

1.2. Czyszczenie pliku głównego

Plik `cwiczenie2.C` zawiera sporo informacji, które nie będą nam na razie potrzebne. Można go więc wyczyścić zostawiając następujące linie:

```
1  #define cwiczenie2_cxx
2  #include "cwiczenie2.h"
3  #include <TH2.h>
4  #include <TStyle.h>
5  #include <TCanvas.h>
6
7  void cwiczenie2::Loop()
8  {
9    if (fChain == 0) return;
10
11   Long64_t nentries = fChain->GetEntries(); //zamiast GetEntriesFast()
12
13   for (Long64_t jentry=0; jentry<nentries; jentry++)
14   {
15     Long64_t ientry = LoadTree(jentry);
16     if (ientry < 0) break;
17     fChain->GetEntry(jentry);
18   }
19 }
```

1.3. Plik nagłówkowy (cwiczenie2.h)

Plik zawiera:

- deklaracje bibliotek:

```
8  #ifndef cwiczenie2_h
9  #define cwiczenie2_h
10
11 #include <TROOT.h>
12 #include <TChain.h>
13 #include <TFile.h>
```

- klasę `cwiczenie2`, w której zostały zadeklarowane:
  - zmienne:

```
25      Float_t         m;
26      Float_t         x1;
27      Float_t         x2;
28      Float_t         x3;
29      Float_t         x4;
30      Float_t         x5;
31      Float_t         x6;
32      Float_t         x7;
```

- gałęzie:

```
35      TBranch         *b_m;    //!
36      TBranch         *b_x1;   //!
37      TBranch         *b_x2;   //!
38      TBranch         *b_x3;   //!
39      TBranch         *b_x4;   //!
40      TBranch         *b_x5;   //!
41      TBranch         *b_x6;   //!
42      TBranch         *b_x7;   //!
```

- funkcje:

```
44      cwiczenie2(TTree *tree=0);
45      virtual ~cwiczenie2();
46      virtual Int_t     Cut(Long64_t entry);
47      virtual Int_t     GetEntry(Long64_t entry);
48      virtual Long64_t  LoadTree(Long64_t entry);
49      virtual void      Init(TTree *tree);
50      virtual void      Loop();
51      virtual Bool_t    Notify();
52      virtual void      Show(Long64_t entry = -1);
```

- definicje funkcji, które (za wyjątkiem `Loop()`) stanowią dalszą część pliku.

2. Stworzenie i modyfikacja kompilowalnego programu

Plik `cwiczenie2.C` jest tzw. makrem – można go uruchomić za pomocą programu ROOT. Makra nie można skompilować za pomocą `g++`. Zwykle nie jest to problemem, ale tracimy dostęp do np. debuggera C++. W tej części ćwiczenia zmodyfikujemy makro tak, aby stało się wykonywalnym programem.

4. Utworzyć kompilowalny program: dodać funkcję `main()`, utworzyć obiekt klasy `cwiczenie2`, wywołać funkcję `Loop()` **(P)**.
5. Zmodyfikować program tak, aby wyświetlił informację o ilości wszystkich przypadków w próbce **(P)**.
6. Wyświetlić na ekranie informację o wartości parametru m dla pierwszych dziesięciu przypadków **(P)**.
7. Wyświetlić na ekran wartość parametru m tylko dla przypadków, gdy $x1 > 3$ **(P)**.
8. Stworzyć plik `histogramy1.root` zawierający następujące histogramy **(P)**:
   - rozkład zmiennej m,
   - rozkład zmiennej $x1$,
   - rozkład zmiennej m pod warunkiem, że $x1 > 1$,
   - rozkład zmiennej $x1$ pod warunkiem, że $x1 > 1$.

   Zakresy należy dobrać tak, żeby wszystkie przypadki były wyraźnie widoczne. Można skorzystać z informacji z `TBrowser`.
9. Otworzyć plik `histogramy1.root` i przejrzeć histogramy używając `TBrowser`.

3. Fitowanie funkcji oraz histogramy dwuwymiarowe

Bardzo często plik z danymi jest duży – procesowanie go zajmuje sporo czasu. Zauważaliśmy także, że „upiększanie" wykresów wymaga zwykle kilku iteracji. Dlatego często „surowe" wykresy zapisuje się w osobnym pliku, który następnie jest odpowiednio obrabiany.

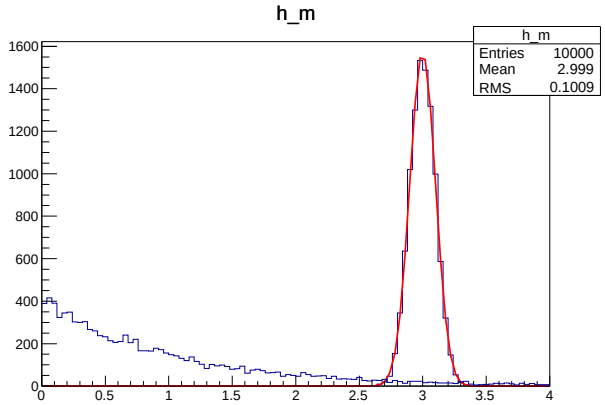
10. Utworzyć program `cw2_wykresy.cpp`, który wczyta histogramy z pliku `histogramy1.root` (**P**).
11. Utworzyć kanwę, narysować na niej wykres zmiennej `m` pobrany z pliku `histogramy1.root` oraz zapisać jako rysunek w formacie eps oraz jpg.
12. Do histogramu należy dofitować rozkład Gaussa (**P**). Ile wynosi znaleziona:
    - wartość stałej? .....
    - błąd wartości stałej? .....
    - średnia? .....
    - błąd średniej? .....
    - szerokość rozkładu ? .....
    - błąd szerokości? .....
13. Na pustej kanwie narysować dwuwymiarowy histogram zmiennych `x4` oraz `x5` a następnie zapisać go jako obraz w formacie eps oraz jpg (**P**).
14. Podstawowy sposób rysowania nie jest szczególnie ładny. Spróbować innych opcji (np. `surf`, `colz`, itd.).

4. Dołączenie informacji z drugiego pliku

W analizie często musimy porównywać dane z kilku różnych plików. Ponieważ mają zazwyczaj tę samą strukturę, dlatego wygodnie jest przygotować jeden kod do analizy i wczytywać do niego różne pliki z danymi.

15. Ze strony praktyk należy ściągnąć i zapisać w katalogu `ROOT_cw2` plik z danymi (`mc_sample2.root`).
16. Zmodyfikować program `cwiczenie2`, tak aby wczytywał on dane z pliku o nazwie podanej jako argument funkcji głównej `main`. W tym celu należy posłużyć się klasą `TChain`, która zastąpi `TTree` (**P**).
17. Stworzyć plik o nazwie `histogramy2.root` zawierający histogramy takie jak zdefiniowane w pkt. 8. Nazwa pliku powinna być podawana jako drugi argument funkcji `main`.
18. Zmodyfikować program `cw2_wykresy.cpp` tak, aby rysował wykresy o tej samej nazwie na jednej kanwie. Wykresy mają być pobrane z plików `histogramy1.root` oraz `histogramy2.root`.

Po poprawnym wykonaniu ćwiczenia połączony wykres zmiennej `m` powinien wyglądać następująco:



Podpowiedzi

4 Na końcu pliku `cwiczenie2.C` należy dodać:

```
1  int main(){
2    cwiczenie2 a; //tworzenie obiektu
3    a.Loop(); //wywolanie funkcji
4    return 0;
5  }
```

5 `cout << "Liczba przypadkow: \t" << nentries << endl;` Zadziała, jeżeli zmieniliśmy `GetEntriesFast()` na `GetEntries()`. Należy dodać (najlepiej do pliku `cwiczenie2.h`) bibliotekę `iostream` oraz polecenie `using namespace std`.

6 `if (jentry < 10) cout << "Przypadek nr: \t" << jentry << "\t m = " << m << endl;`

7 `if (x1 > 3) cout << "Przypadek nr: \t" << jentry << "\t x1 = " << x1 << "\t m = " << m << endl;`

8 W celu utworzenia pliku należy skorzystać z klasy `TFile` (zob. ćw. 1). Plik należy stworzyć z opcją `RECREATE`. Histogramy, po zadeklarowaniu i wypełnieniu, należy zapisać do pliku (`h_m->Write()`). Na koniec należy zamknąć plik (`f->Close()`).

10 Należy utworzyć obiekt (wskaźnik) klasy TH1F a następnie przypisać mu wartość z pliku: `f->GetObject("h_m", h_m);`

12 Wyszukać w Internecie *root gaus fit*. Wartości fitu zostaną wyświetlone w konsoli.

13 Klasa `TH2F`.

16 Zadanie ma kilka etapów:

a) dodać argumenty do funkcji głównej: `main(int argc, char** argv)`,

b) dodatkowo można dodać zabezpieczenie i wyświetlić nazwę argumentu:

```
1    if (argc!=2) {cout << "Zla liczba argumentow!" << endl; return 1;}
2    cout << argv[1] << endl;
```

c) utworzyć wskaźnik typu `TChain`, który wczyta drzewo o nazwie `ntuple`

```
1    TChain *chain = new TChain("ntuple");
2    chain->Add(argv[1]);
```

d) zmodyfikować funkcje w pliku `cwiczenie2.h` (zamienić `TTree` na `TChain`)

```
27  TChain          *fChain;   //!pointer to the analyzed TTree or TChain

50  cwiczenie2(TChain *tree=0);

55  virtual void     Init(TChain *tree);

64  cwiczenie2::cwiczenie2(TChain *tree) : fChain(0)

104 void cwiczenie2::Init(TChain *tree)
```