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/2017/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>
14 #include <vector>
```

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

```
29      vector<double>  *x;
30      vector<double>  *y;
31      vector<double>  *z;
32      vector<double>  *px;
33      vector<double>  *py;
34      vector<double>  *pz;
35      vector<double>  *e;
36      vector<int>     *id;
37      vector<double>  *charge;
38      vector<double>  *mu_like;
```

- gałęzie:

```
41      TBranch        *b_x;    //!
42      TBranch        *b_y;    //!
43      TBranch        *b_z;    //!
44      TBranch        *b_px;   //!
45      TBranch        *b_py;   //!
46      TBranch        *b_pz;   //!
47      TBranch        *b_e;    //!
48      TBranch        *b_id;   //!
49      TBranch        *b_charge;   //!
50      TBranch        *b_mu_like;   //!
```

- funkcje:

```
55      cwiczenie2(TTree *tree=0);
56      virtual ~cwiczenie2();
57      virtual Int_t     Cut(Long64_t entry);
58      virtual Int_t     GetEntry(Long64_t entry);
59      virtual Long64_t  LoadTree(Long64_t entry);
60      virtual void      Init(TTree *tree);
61      virtual void      Loop();
62      virtual Bool_t    Notify();
63      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)**. **Uwaga** – przy kompilacji może pojawić się błąd typu `error: 'vector' does not name a type`. W celu jego usunięcia wystarczy do pliku `cwiczenie2.h` dopisać `using std::vector;` (np. w linii nr 19).
5. Zmodyfikować program tak, aby wyświetlił informację o ilości wszystkich przypadków w próbce **(P)**.
6. Dla przypadku nr 1 wyświetlić rozmiar wszystkich zmiennych typu wektor **(P)**. Czy rozmiar wektorów jest taki sam czy różny? Czy jest to przypadek, czy też powinno tak być?
7. Dla przypadku nr 1 wyświetlić wartości wszystkich elementów wektora `z` **(P)**. Czy wyświetlone liczby są takie same?
8. Stworzyć plik `histogramy1.root` zawierający następujące histogramy **(P)**:
   i rozkład zmiennej `mu_like` dla wszystkich cząstek w próbce,
   ii rozkład zmiennej `mu_like` pod warunkiem, że $|id| = 13$,
   iii rozkład pędu poprzecznego $pT = \sqrt{px \cdot px + py \cdot py}$,
   iv rozkład zmiennej `pT` pod warunkiem, że `pT` $>$ `10`,
   v rozkład zmiennej `pT` pod warunkiem, że `mu_like` $>$ `0.9`.

   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`.
10. Jakie wnioski można wyciągnąć porównując rozkłady:
    - i oraz ii,
    - iii oraz v,

    przy założeniu, że zmienna `mu_like` opisuje prawdopodobieństwo, że dana cząstka jest mionem? W celu porównania wykresów należy je narysować w skali logarytmicznej.

3. Klasa TLorentzVector

W fizyce cząstek elementarnych bardzo często wykorzystujemy czterowektory. Dlatego do pakietu ROOT została dodana klasa `TLorentzVector` ułatwiająca wykonywanie działań na takich obiektach.

11. Z danego przypadku wybrać mion dodatni oraz ujemny[1] o największych pędach poprzecznych. Pęd poprzeczny należy policzyć używając `TLorentzVector` (**P**). Dodać rozkład ich pędów jako histogram w pliku `histogramy1.root`.
12. Dodać histogram pędu poprzecznego układu środka masy dwóch mionów o największym pędzie poprzecznym (i o przeciwnych znakach) do pliku `histogramy1.root`.
13. Dodać histogram rozkładu masy niezmienniczej mionów o największym pędzie poprzecznym do pliku `histogramy1.root`.

4. Fitowanie funkcji

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.

14. Utworzyć program `cw2_wykresy.cpp`, który wczyta histogramy z pliku `histogramy1.root` (**P**).
15. Utworzyć kanwę, narysować na niej wykres masy niezmienniczej układu dwóch mionów (patrz pkt 13) pobrany z pliku `histogramy1.root` oraz zapisać jako rysunek w formacie eps oraz jpg.
16. Do histogramu należy dofitować rozkład Gaussa (**P**). Ile wynosi znaleziona:
    - wartość stałej? .....
    - błąd wartości stałej? .....
    - średnia? .....
    - błąd średniej? .....
    - szerokość rozkładu ? .....
    - błąd szerokości? .....

5. Dołączenie informacji z drugiego pliku

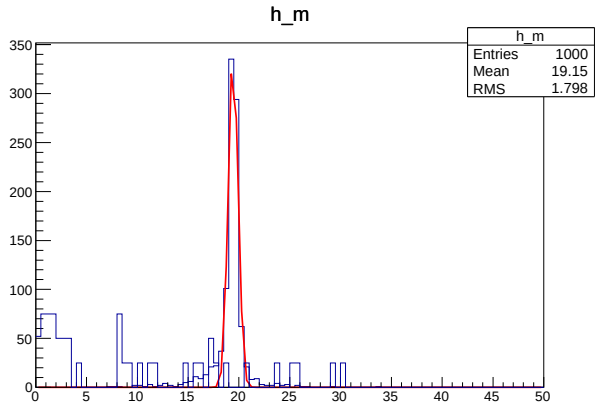
W analizie często musimy porównywać dane z kilku różnych plików. Ponieważ mają zazwyczaj tę samą strukturę, dlatego wygodnie jest przygotować jeden kod do analizy i wczytywać do niego różne pliki z danymi.

17. Ze strony praktyk należy ściągnąć i zapisać w katalogu `ROOT_cw2` plik z danymi (`mc_sample2.root`).

[1] Mion można rozpoznać po numerze identyfikacyjnym cząstki ($id_{\mu^-} = 13$, $id_{\mu^+} = -13$). Nadawaniem numerów zajmuje się grupa PDG. Więcej informacji można znaleźć na stronie (pdg.lbl.gov/2007/reviews/montecarlorpp.pdf).

18. 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**).
19. Stworzyć plik o nazwie `histogramy2.root` zawierający histogramy takie jak zdefiniowane w pkt. 8. Nazwa pliku powinna być podawana jako drugi argument funkcji `main`.
20. 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 Np. `x->size()` zwróci długość (rozmiar) wektora x.

7 `z->at(a)`, gdzie a jest kolejnym elementem wektora z.

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()`).

11 Uzyskanie informacji o pędzie poprzecznym:

```
1    TLorentzVector v;
2    v.SetPxPyPzE(px->at(a), py->at(a), pz->at(a), e->at(a));
3    cout << v.Pt() << endl;
```

12 Należy utworzyć trzeci wektor będący sumą wektorów zawierających informacje o mionach:

```
1    TLorentzVector v, v_mion_dodatni, v_mion_ujemny;
2    //wypelnic informacje o mionach
3    v = v_mion_dodatni + v_mion_ujemny;
4    cout << v.Pt() << endl;
```

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

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

18 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)
```