Sprawozdanie z projektu z przedmiotu
Cząstki Elementarne i Ich Oddziaływania.

Jakub Ahaddad
Marcin Polok

1 Wstęp

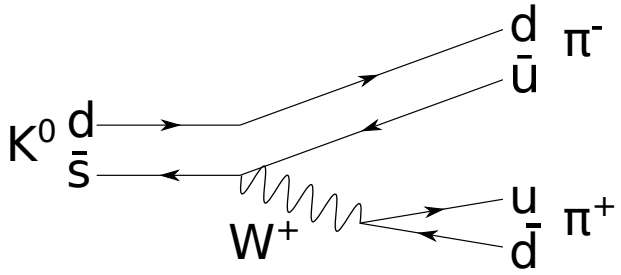
Celem projektu jest wyznaczenie masy i czasu życia jednego z 3 dostępnych mezonów przedstawionych w tabeli 1. Podstawą do wyznaczenia tych parametrów są rzeczywiste dane zebrane w spektrometrze LHCb w 2016r. Część praktyczną projektu przeprowadzono za pomocą oprogramowania ROOT, pisząc własny skrypt.

Tab. 1: Parametry możliwych do wyboru mezonów[1][2][3]

Cząstka	masa, MeV	spin	ładunek	J^{PC}	Średni czas życia, s
D^{*+}	2010.26 ± 0.05	0	+1	1^{-+}	$(1040 \pm 7) \cdot 10^{-10}$
D^0	1863.84 ± 0.05	0	0	0^{-+}	$(410.1 \pm 1.5) \cdot 10^{-10}$
K^0_S	497.611 ± 0.013	0	0	0^{-+}	$(0.8954 \pm 0.0004) \cdot 10^{-10}$

2 Teoria

2.1 Wybór mezonu



Rys. 1: Diagram Feynmana rozpadu mezonu K^0_s

Wybrany przez autorów mezon to kaon K^0_s. Proces rozpadu kaonu zachodzi przez oddziaływanie słabe, w tym przypadku kwark $\bar{s}$ rozpada się na $\bar{u}$ z emisją bozonu W^+. Diagram Feynmana rozpadu tej cząstki na piony π^+ i π^- przedstawiony został na rysunku 1.

Zauważyć można, że z kwarków wyjściowych powstać mogły też piony π^0 ($d\bar{d}$ i $u\bar{u}$), jednak prawdopodobieństwo na ten rozpad (30%[1]) jest dużo mniejsze od prawdopodobieństwa na rozpad π^+ i π^- (69%[1]) ze względu na tłumienie przez kolor.

2.2 Sposób obliczeń

W celu wykonania histogramu masy mezonu wykorzystano bezpośrednio zmienną D_Ks0_M zawartą w drzewie. Do wyliczenia czasu życia posłużono się wzorem

$$\tau = \frac{md}{pc} \ , \tag{1}$$

gdzie m - masa, d - przebyty dystans a p - wartość pędu cząstki. Pęd uzyskano bezpośrednio ze zmiennej D_Ks0_P w drzewie, zaś d wyliczono na podstawie współrzędnych wierzchołka początkowego i końcowego.

2.3 Kryteria wyboru przypadków sygnałowych

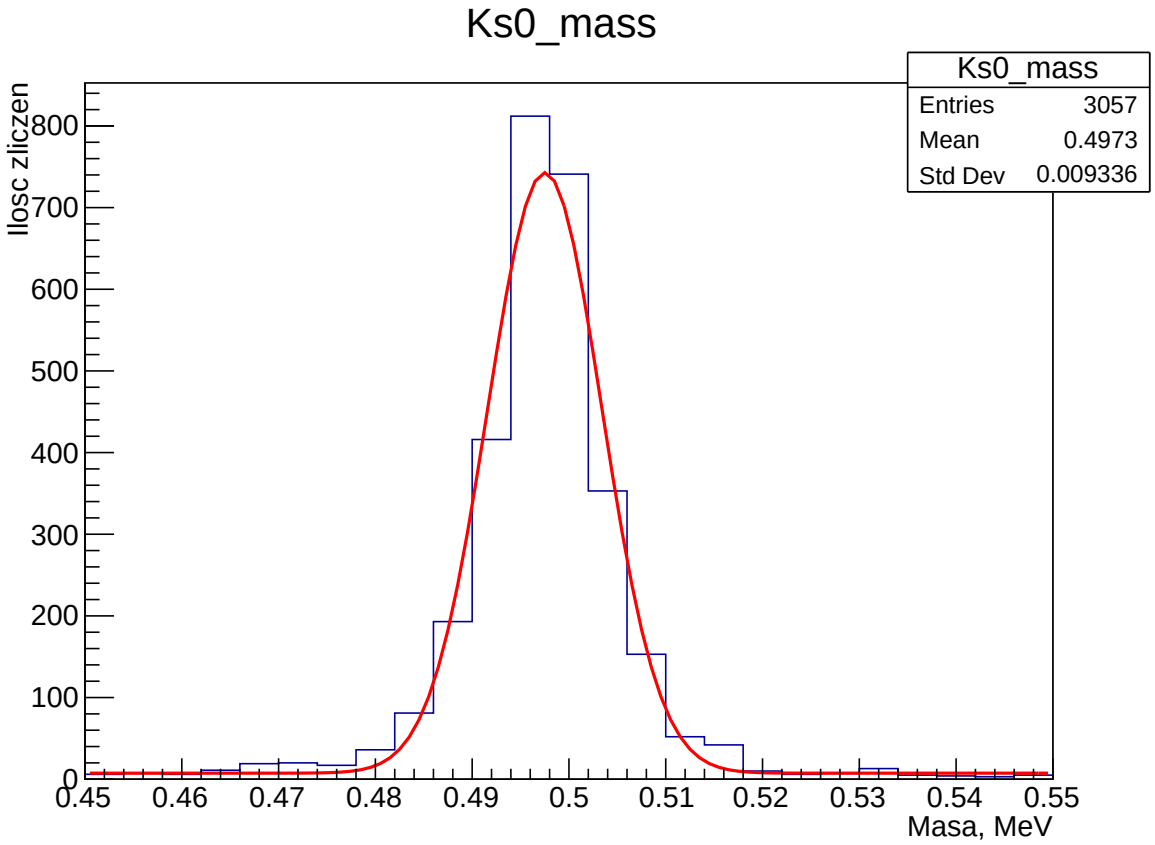
W celu wybrania przypadków zapewniających lepszą dokładność wyników zastosowano selekcję poprzez wybór zdarzeń, dla których dodatkowe zmienne mieszczą się w odpowiednich zakresach:

- parametr zderzenia w wierzchołku produkcji - powinien być jak najmniejszy dla cząstki wytworzonej w tym wierzchołku,
- wartość χ^2 parametru zderzenia w wierzchołku produkcji - jak najmniejsza,
- flight distance - wybierane parametry z jak największym.

Dodatkowo, do wyznaczenia czasu życia wykorzystano przypadki pozostałe po odrzuceniu gaussowskiego ogona rozkładu masy.

3 Wyniki

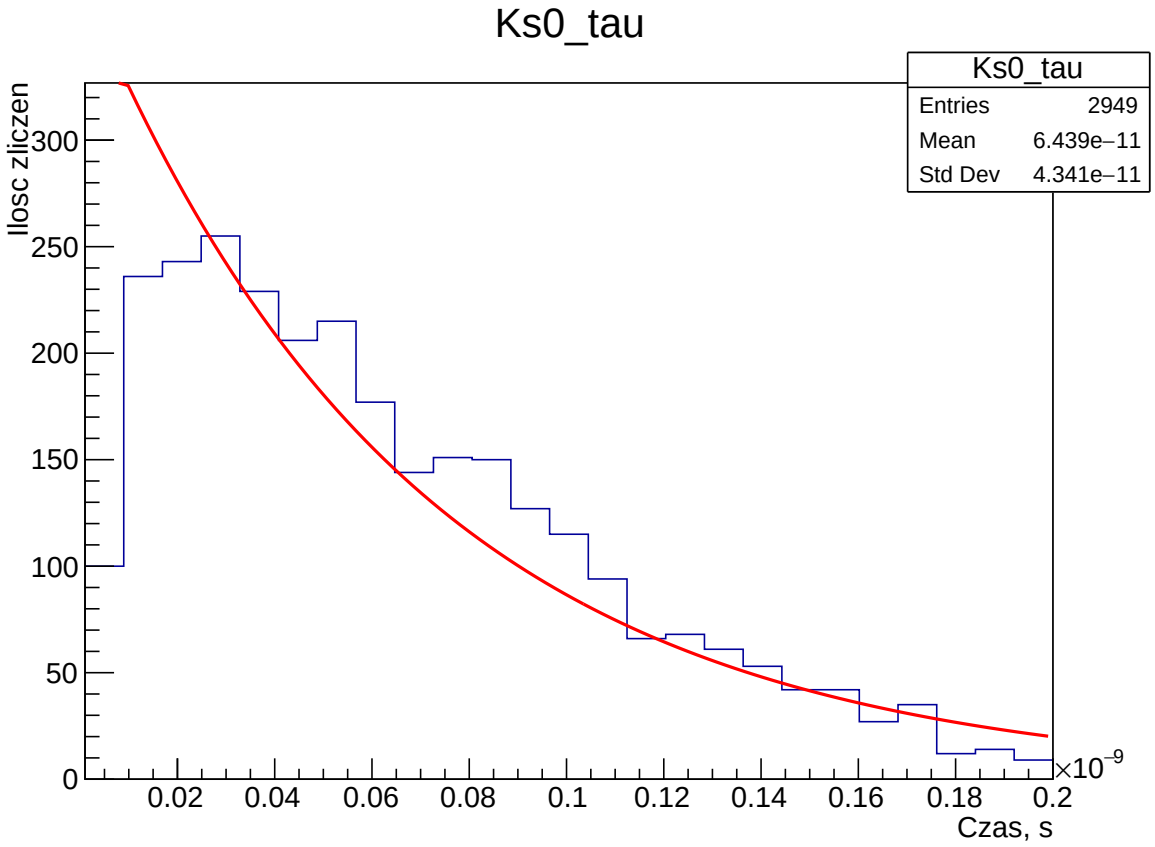
Rysunek 2 przedstawia uzyskany histogram masy mezonu K^0_s.



Rys. 2: Otrzymany rozkład masy z dopasowaną krzywą Gaussa

Do histogramu dopasowano krzywą Gaussa z dodanym stałym tłem. Uzyskano wartość średnią 497.3 MeV, oddaloną od wartości tablicowej 497.6 MeV o mniej niż wartość odchylenia standardowego.

Rysunek 3 przedstawia uzyskany histogram czasu życia mezonu K_s^0.



Rys. 3: Otrzymany rozkład czasu życia z dopasowaną eksponentą

Wartość tablicowa czasu życia zawiera się w uzyskanym wyniku, jednak otrzymana dokładność jest bardzo niska, ponieważ odchylenie standardowe $4.34 \cdot 10^{-11}$ s jest tego samego rzędu co wartość średnia $6.44 \cdot 10^{-11}$ s.

4 Podsumowanie

W niniejszej pracy zrealizowano postawione zadanie. Wybranym mezonem był K_s^0, dla którego opracowano diagram Feynmana rozpadu na piony naładowane oraz za pomocą oprogramowania w języku ROOT wyznaczono jego masę ($m = 497$ MeV) oraz czas życia ($\tau = 6,44 \cdot 10^{-11}$ s). Opisano również kryteria wyboru przypadków sygnałowych, a ich dokładne wartości można odczytać w załączonym kodzie programu.

Literatura

[1] Dane PDG na temat mezonu K_s^0:
`http://pdg.lbl.gov/2016/listings/rpp2016-list-K-zero-S.pdf`
dostęp 01.2016r.

[2] Dane PDG na temat mezonu D^0:
`http://pdg.lbl.gov/2016/listings/rpp2016-list-D-zero.pdf`
dostęp 01.2016r.

[3] Dane PDG na temat mezonu D^{*+}:
`http://pdg.lbl.gov/2016/listings/rpp2016-list-D-star-2010-plus-minus.pdf`
dostęp 01.2016r.

A Załącznik

```
// ************************************************ //
// ** Malopolska Chmura Edukacyjna 2016 @ AGH UST **
// ** A. Oblakowska-Mucha **
// ** na podstawie make_meson_2 wykonali: Marcin Polok, Jakub Ahaddad
// ** it can be comptiled in Root: .L lab03meson.C
//  root: lab03meson()
// ************************************************ //

#include <iostream>
#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>
#include <TCanvas.h>
#include <TStyle.h>
#include <TH1F.h>
#include <TMath.h>

Double_t getTau(Double_t p, Double_t d, Double_t m)
{
    Double_t c = 299792458000; // mm/s
    Double_t tau = (m*d)/(p*c);
    return tau;
}

Double_t fitGauss(Double_t *x, Double_t *par){
        Double_t arg = 0;
        if(par[2] != 0) arg = (x[0]-par[1])/par[2];
        Double_t y = par[0]*TMath::Exp(-.5*arg*arg)+par[3];
```

```
        return y;
}

Double_t fitExp(Double_t *x, Double_t *par){
        Double_t arg=0;
        if(par[1] != 0) arg = x[0]/par[1];
        Double_t y = par[0]*TMath::Exp(-arg);
        return y;
}

void lab03meson() {
        //otwarcie pliku
        TChain* BDK_chain = new TChain("/MyDstarTuple/DecayTree");
        BDK_chain->Add("MyDstar2D0Pi_1.root");
        TTree* BDK_tree = BDK_chain;

        //masa, pęd
        Double_t Ks0_M, Ks0_P ;
        //składowe 4-pędu
        Double_t Ks0_PX, Ks0_PY, Ks0_PZ, Ks0_PE;
        //położenie primary vertexa
        Double_t Ks0_PVX, Ks0_PVY, Ks0_PVZ;
        //położenie end vertexa
        Double_t Ks0_EVX, Ks0_EVY, Ks0_EVZ;
        //dystans, impact param., chi2 IP, flight distance
        Double_t Ks0_dist, Ks0_OIP, Ks0_OIP_X2, Ks0_FD;

        //przypisanie
        BDK_tree->SetBranchAddress("D_Ks0_M", &Ks0_M);
        BDK_tree->SetBranchAddress("D_Ks0_P", &Ks0_P);
        BDK_tree->SetBranchAddress("D_Ks0_PX", &Ks0_PX);
        BDK_tree->SetBranchAddress("D_Ks0_PY", &Ks0_PY);
        BDK_tree->SetBranchAddress("D_Ks0_PZ", &Ks0_PZ);
        BDK_tree->SetBranchAddress("D_Ks0_PE", &Ks0_PE);
        BDK_tree->SetBranchAddress("D_Ks0_OWNPV_X", &Ks0_PVX);
        BDK_tree->SetBranchAddress("D_Ks0_OWNPV_Y", &Ks0_PVY);
        BDK_tree->SetBranchAddress("D_Ks0_OWNPV_Z", &Ks0_PVZ);
        BDK_tree->SetBranchAddress("D_Ks0_ENDVERTEX_X", &Ks0_EVX);
        BDK_tree->SetBranchAddress("D_Ks0_ENDVERTEX_Y", &Ks0_EVY);
        BDK_tree->SetBranchAddress("D_Ks0_ENDVERTEX_Z", &Ks0_EVZ);
        BDK_tree->SetBranchAddress("D_Ks0_IP_OWNPV", &Ks0_OIP);
        BDK_tree->SetBranchAddress("D_Ks0_IPCHI2_OWNPV", &Ks0_OIP_X2);
        BDK_tree->SetBranchAddress("D_Ks0_FD_ORIVX", &Ks0_FD);

        //ilość binów
        Int_t nBin=25;
        //przelicznik jednostek
        Double_t GeV = 0.001;
        //minimalna i maksymalna masa
```

```
Double_t mMin = .45;
Double_t mMax = .55;
//tablicowa masa Ks0
Double_t mTab = 497.611*GeV;
//minimalny i maksymalny czas życia
Double_t tauMin = 1e-12;
Double_t tauMax = 2e-10;

//deklaracja histogramow
TH1D* hist_M = new TH1D( "Ks0_mass", "Ks0_mass ",nBin, mMin, mMax);
TH1D* hist_tau = new TH1D("Ks0_tau", "Ks0_tau", nBin, tauMin,
    tauMax);

Long64_t nEvTot = BDK_tree->GetEntries();

//kryteria wyboru
for(Int_t event = 0; event < nEvTot; ++event){
        BDK_chain->GetEvent(event);
        if(     Ks0_OIP > 3.0 ||                //mały impact
            parameter
                Ks0_OIP_X2 > 5.0 ||     //małe chi^2
                Ks0_FD < 100) continue; //duży flight distance
        hist_M->Fill(Ks0_M*GeV);
        //odrzucony ogon masy
        if(fabs(Ks0_M*GeV-hist_M->GetMean()) > 0.03) continue;
        //wyliczenie przebytego dystansu
        Ks0_dist = sqrt((Ks0_EVX-Ks0_PVX)*(Ks0_EVX-Ks0_PVX) + (
            Ks0_EVY-Ks0_PVY)*(Ks0_EVY-Ks0_PVY) + (Ks0_EVZ-Ks0_PVZ)*(
            Ks0_EVZ-Ks0_PVZ));
        //wypełnienie histogramu czasem życia
        hist_tau->Fill(getTau(Ks0_P*GeV, Ks0_dist, mTab));
}

//tytuly na osie
hist_M->GetXaxis()->SetTitle("Masa, MeV");
hist_M->GetYaxis()->SetTitle("Ilosc zliczen");
hist_tau->GetXaxis()->SetTitle("Czas, s");
hist_tau->GetYaxis()->SetTitle("Ilosc zliczen");

//ustawiamy parametry startowe dopasowania masy
TF1 *gauss = new TF1("gauss",fitGauss,mMin,mMax,4);
gauss->SetParameters(300,hist_M->GetMean(),hist_M->GetRMS());
gauss->SetParNames("Mass_Const","Mass_Mean","Mass_Sigma","
    Background");

//ustawiamy parametry startowe dopasowania czasu zycia
TF1 *exponent = new TF1("exponent",fitExp,1e-13,1e-9,2);
exponent->SetParameters(100,hist_tau->GetMean(),hist_tau->GetRMS())
    ;
```

```
        exponent->SetParNames("Tau_Const","Tau_Mean","Background");

        //rysujemy mase & fitujemy
        TCanvas* canv_m = new TCanvas("canv_m","canv_m",0,0,800,600);
        //mass_Ks0_can->cd(1);
        hist_M->Draw();
        hist_M->Fit("gauss");

        //rysujemy czas zycia & fitujemy
        TCanvas* canv_tau = new TCanvas("canv_tau","canv_tau",0,0,800,600);
        //tau_Ks0_can->cd(1);
        hist_tau->Draw();
        hist_tau->Fit("exponent","","",5e-12,1e-9);
}
```