Wstęp do PLUTO

PLUTO: Środowisko do generacji emisji cząstek.

Pole działania: Źródło – tor swobodny, zderzenie cząstek, rozpad cząstki, fireball ze zderzenia jąder

Cząstki – hadrony, leptony, foton.
stabilne i rozpadające się (w tym: rezonanse o szerokim rozkładzie masy)

Własności cząstek i kanały rozpadu pobrane z Review of Particle Physics.

Rozpady: na cząstki lub poprzez foton wirtualny.
Zgodne z kinematyką relatywistyczną + możliwe proste modele fizyczne.

Emisja: można ograniczyć pędy, kąty itp, symulując ograniczoną akceptancję detektora

Można zakodować własną cząstkę / rozpad / model.

Forma: Biblioteka dołączana do ROOT. Obiekty do użycia interaktywnie lub makrze, np:
`PParticle, PChannel, PReaction, ...`

Kompatybilność: PLUTO 5 ↔ ROOT 5, PLUTO 6 ↔ ROOT 6

Algorytm: Zdefiniowanie źródła emisji + cząstki(ek) emitowanej(ych) + aktywacja zadanych rozpadów.
Uwzględnienie energii wiązki (transformacja Lorentza)
Wywołanie pętli, która generuje pseudolosowo cząstki (w ramach zadanych warunków)
i doprowadza do rozpadów.

Wynik: Zestaw zdarzeń z cząstkami i ich pędami.

Format wyjścia: Domyślnie drzewo TTree w pliku root. Opcjonalnie: tekst, TNtuple, THnF .

krzysztof.piasecki@fuw.edu.pl

Mody pracy:

- Składanie “z klocków” , tj. obiektami są źródła, cząstki, kanały rozpadu i reakcja.
- Dedykowany “język” skryptowych poleceń

Www :

- https://www-hades.gsi.de/?q=pluto : Strona główna, problemy własności → większość niepubliczna
- https://plutouser.github.io/v6.00/ : Źródło, dokumentacja klas, przykładowe makra

Publikacje:

- I. Fröhlich et al., arXiv:0708.2382v2 “Pluto: A Monte Carlo Simulation Tool for Hadronic Physics”
- I. Fröhlich et al., arXiv:0905.2568v1 “Design of the Pluto Event Generator”

Prezentacje:

- I. Fröhlich, “The Pluto++ Event Generator” (ACAT 07 Amsterdam)
- I. Fröhlich, “Pluto: An Event Generator Framework for Hadronic Physics” (A2 CB17 Meeting)

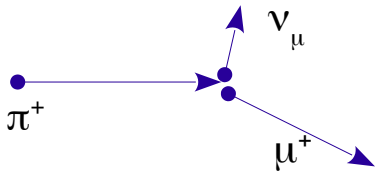
Instalacja

- Źródła:
  ```
  wget https://plutouser.github.io/v6.00/pluto_v6.00.tar.gz
  ```
  lub:
  ```
  git clone https://github.com/PlutoUser/pluto6.git
  ```
- Kroki instalacji:
  ```
  cd pluto6 ; mkdir builddir ; cd builddir
  cmake ..
  make
  ```
- Rezultat: Biblioteka `libPluto.so`
  Proponuję: `export PLUTOLIBDIR={.....}/builddir`
- Aktywacja:
  - w sesji:
    ```
    gSystem->Load ("..../libPluto.so");
    ```
  - w makrze:
    ```
    R__LOAD_LIBRARY (..../libPluto.so)
    ```
- Sieć neutronx: (kontakt z prowadzącym)

Przykład 1

Rozpad $\pi^+ \longrightarrow \mu^+ \nu_\mu$

(E_k = 1 GeV)



Makro preprocesora
do podłączania bibliotek

Tworzymy 1 x π^+ .
Pluto zidentyfikuje po nazwie.
Nadajemy E_{kin} = 1 GeV.

Tworzymy reakcję:

1. Substrat: π^+
2. Produkty: μ^+, ν_μ

   Format: “Descriptor string”

3. Plik wyjściowy

Uruchamiamy pętlę
10000 zdarzeń

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int pip_mupnu ()
{
  PParticle pip ("pi+", 1.0 );

  PReaction* my_reaction = new PReaction
      ( &pip ,
        "mu+ nu",
        "pip_mupnu", 1);

  my_reaction->Print ();
  my_reaction->Loop (10000, 1);

  return 0;
}
```

- **Wykonujemy symulację**
- **Czytamy plik wynikowy w formacie root**

Podłączamy bibliotekę

```
$ nice root -b -q pip_mupnu.C

$ nice root -l pip_mupnu.root
{...}
Warning in <TClass::Init>: no dictionary {...}
Root [1] gSystem->Load ("..../libPluto.so")
(int) 0
```

```
$ data->Scan ("pid:GetParentId()")
$ data->Draw ("M()", "pid==8")
$ data->Draw ("E()-M()", "pid==8")
              "Rapidity()"

$ data->Draw ("Rapidity()", "pid==5")
              "Pt():Rapidity()", "pid==5")

$ data->Draw ("P():Theta()", "  Name() == \"mu+\"  ")

$ data->Draw ("Rapidity():Pt()", "pid==5")
               Beta()
               Pz()  Px()  Py()
```

{PID → por. s. 6}

...

1 wpis = kolekcja cząstek z 1 reakcji (TClonesArray)

Kinematyka obliczana w Lab

PParticle – Cząstka

Reprezentuje konkretną cząstkę o danych własnościach i 4-Pędzie.

```
$ PParticle p ("pi0")
$ p.Print ()
$ cout << p.Name() << "\t" << p.ID() << endl;

$ listParticle ("pi0")   lub   ( 7 )

$ p.Set... (...)
    SetE, Px, Py, Pz,
    SetTheta (...), SetPhi(...)

$ p.E() , Px Py Pz Pt Mt Theta Phi

$ p.Boost (beta_x, beta_y, beta_z)
```

Obracają wektor pędu

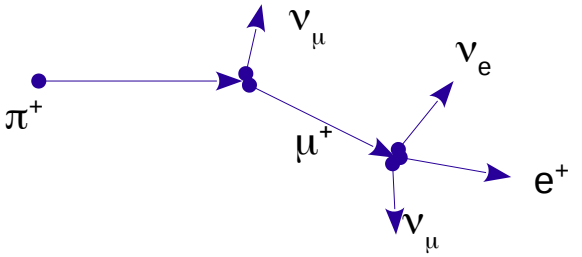
Transformacja Lorentza

Przykład 2

Rozpad $\pi^+ \longrightarrow \mu^+ \; \nu_\mu$

$\mu^+ \longrightarrow \nu_\mu \; e^+ \; \nu_e$

(E_k = 1 GeV)



μ^+ jest cząstką nietrwałą ($\tau \approx 10^{-6}$ s) i rozpada się do $\nu_\mu \; e^+ \; \nu_e$.
Dodajmy ten rozpad.

String descriptor

```
"nu mu+ [e+ nu nu]"
```

(Nawiasy umieszczamy po cząstce pierwotnej i wpisujemy produkty)

Uwaga: Pluto nie rozróżnia rodzajów ν.

Nazwa pliku wyjściowego

Flagi

f0: Zapisz (1–wszystkie cz. / 0–tylko finalne)
f1: (nieużywana)
f2: Wyznacz vertex prod. (1 – tak, 0 – nie)
f3: Generuj plik textowy (1 – tak, 0 – nie)

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int pip_mupnu_epnunu ()
{
  PParticle pip ("pi+", 1.0 );

  PReaction* my_reaction = new PReaction (
                  &pip ,
                  "nu mu+ [e+ nu nu]",
                  "pip_mupnu_epnunu" ,
                  1, 0, 0, 1
                 );
    my_reaction->Print ();

    my_reaction->Loop (10000, 1);
    return 0;
}
```

```
$ data->Scan ("Name() : pid : GetParentId() : GetSiblingIndex()")
$ data->Draw ("Theta()", "pid==8")
                          pid==5
                          pid==4 && GetParentId()==8
                          pid==4 && GetParentId()==5
```

Dostęp do danych o cząstkach i rozpadach

Typy cząstek uwzględnione w Pluto – według PDG.

```
$ listParticle ()
```
← Wypis wszystkich cząstek z bazy danych

- Cząstki trwałe oraz o szerokości Γ < 1 MeV

PID	Symbol	Typ	M[MeV]
1	g	γ	0
2	e+	e^+	0.511
3	e-	e^-	0.511
4	nu	ν	0
5	mu+	μ^+	105.7
6	mu-	μ^-	105.7
7	pi0	π^0	135.0
8	pi+	π^+	139.6
9	pi-	π^-	139.6
10	K0L	K^0_L	497.7
11	K+	K^+	493.7
12	K-	K^-	493.7

PID	Symbol	Typ	M[MeV]
13	n	n	939.6
14	p	$\underline{p}$	938.3
15	anti_p	$\overline{p}$	938.3
16	K0S	K^0_s	497.7
17	eta	η	547.5
18	Lambda	Λ	1115.7
19	Sigma+	Σ^+	1189.4
20	Sigma0	Σ^0	1192.6
21	Sigma-	Σ^-	1197.4
22	Xi0	Ξ^0	1314.9
23	Xi-	Ξ^-	1231.3
24	Omega	Ω	1672.5

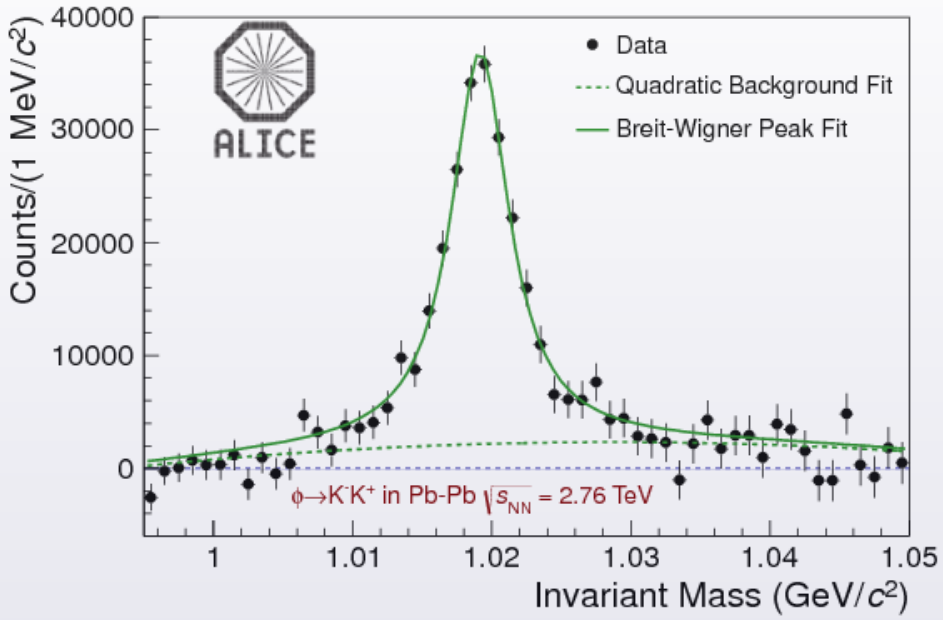
PID	Symbol	Typ	M[MeV]
25	anti_n	$\overline{n}$	939.6
26	anti_Lambda	$\overline{\Lambda}$	1115.7
27	anti_Sigma-	$\overline{\Sigma}^-$	1189.4
28	anti_Sigma0	$\overline{\Sigma}^0$	1192.6
29	anti_Sigma+	$\overline{\Sigma}^+$	1197.4
30	anti_Xi0	$\overline{\Xi}^0$	1314.9
31	anti_Xi+	$\overline{\Xi}^+$	1321.3
32	anti_Omega+	$\overline{\Omega}$	1672.5
33	K0S	η	497.7
53	eta'	η'	957.7
67	J/Psi	J/Ψ	3096.9
68	Psi'	Ψ'	3686.0

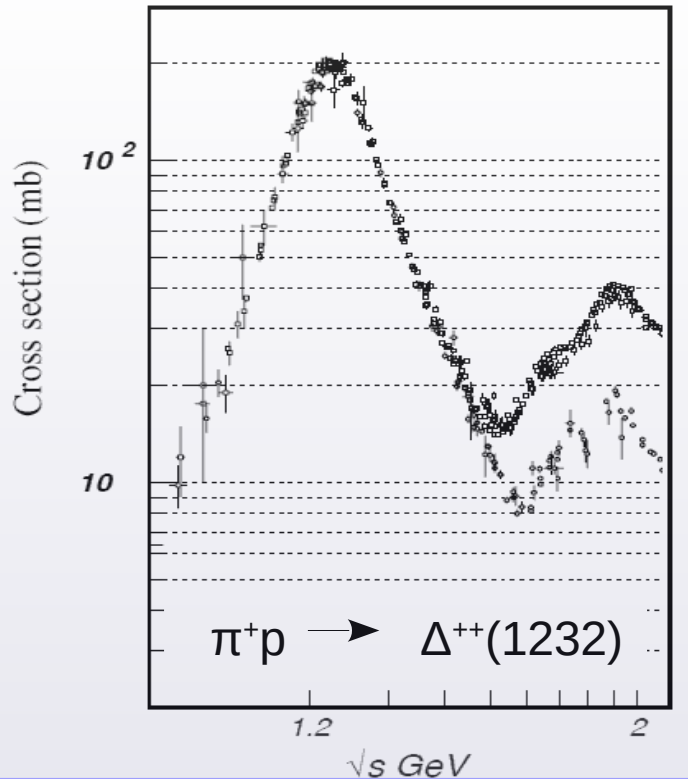
- Nukleony i najlżejsze jądra atomowe:

PID	Symbol	Typ	M[MeV]
13	n	n	939.6
14	p	$\underline{p}$	938.3
15	anti_p	$\overline{p}$	938.3
25	anti_n	$\overline{n}$	939.6

PID	Symbol	Typ	M[MeV]
45	d	d	1875.6
46	t	t	2809.3
47	alpha	^{4}He	3727.4
49	He3	^{3}He	2809.2

Rezonanse (cząstki, których masa jest rozkładem)
Stany wzbudzone układu 2 lub 3 kwarków.





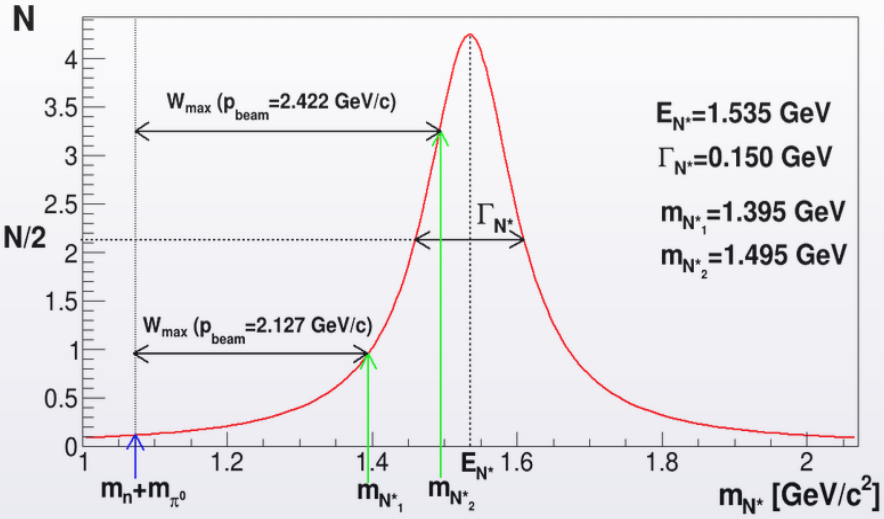
Model: relatywistyczna krzywa Breita-Wignera

$$g(m) = A \frac{m^2 \Gamma^{\mathrm{tot}}(m)}{(M_{\mathrm{R}}^2 - m^2)^2 + m^2 (\Gamma^{\mathrm{tot}}(m))^2}$$

$$\Gamma^{\mathrm{tot}}(m) = \sum_{\mathrm{k}}^{N} \Gamma^{\mathrm{k}}(m)$$

Γ^{tot} : całkowita szerokość połówkowa,
suma po wkładach od kanałów rozpadu, Γ^k

Zakres próbkowania: m ∈ [M_R - 2Γ , M_R + 12 Γ]



- Rezonanse w bazie danych Pluto:

Mezony

PID	Symbol	Typ	M[MeV]	Γ [MeV]
41	rho0	ρ^0	769.9	150.7
42	rho+	ρ^+	769.9	150.7
43	rho-	ρ^-	769.9	150.7
52	w	ω	781.9	8.43
54	sigma	σ	600.0	500.
55	phi	ϕ	1019.4	4.43

Bariony

PID	Symbol	Typ	J[ħ]	M[MeV]	Γ[MeV]
34	D0	$\Delta(1232)^0$	3/2	1232	120
35	D++	$\Delta(1232)^{++}$	3/2	1232	120
36	D+	$\Delta(1232)^+$	3/2	1232	120
37	D-	$\Delta(1232)^-$	3/2	1232	120
38	NP11+	$N(1440)^+$	1/2	1440	350
39	ND13+	$N(1520)^-$	3/2	1520	120
40	NS11+	$N(1535)^-$	1/2	1535	150
56	DP330	$\Delta(1600)^0$	3/2	1600	350
57	DP33++	$\Delta(1600)^{++}$	3/2	1600	350

PID	Symbol	Typ	J[ħ]	M[MeV]	Γ[MeV]
58	DP33+	$\Delta(1600)^+$	3/2	1600	350
59	DP33-	$\Delta(1600)^-$	3/2	1600	350
60	DS310	$\Delta(1620)^0$	1/2	1620	150
61	DS31++	$\Delta(1620)^{++}$	1/2	1620	150
62	DS31+	$\Delta(1620)^+$	1/2	1620	150
63	DS31-	$\Delta(1620)^-$	1/2	1620	150
64	NP110	$N(1440)^0$	1/2	1440	350
65	ND130	$N(1520)^0$	3/2	1520	120
66	NS110	$N(1535)^0$	1/2	1535	150

- W Przyrodzie istnieje znacznie więcej rezonansów, o wyższych masach.
- Do bazy danych można dodać nowe cząstki, por. str. 23.

Dostęp do danych o cząstkach i rozpadach c.d.

```
$ listParticle (34)
```

← Dane dot. $\Delta(1232)^0$:

- centroid i Γ masy
- Kanały rozpadu + wkład (BR ≡ stosunek rozgałęzienia)

Bezpośredni dostęp:

```
$ PStaticData* sd = makeStaticData() ;
```

← Obiekt zaw. bazę danych

```
$ sd->GetParticleMass (7)
      GetParticleID ("p")
      GetParticleName (7)
      GetParticleTotalWidth (34)
      GetParticleSpin (34)
      GetParticleParity (34)
      GetParticleIsospin (34)

$ sd->GetParticleNChannels (7)  ---> 2

$ sd->PrintParticle (7)  lub ("pi0")

$ sd->PrintDecayByKey (143)

Database key=143
Database name=pi0 --> photon + photon
Decay index=3
Branching ratio=0.988000
Decay product 1->Database name=g
Decay product 2->Database name=g

$ sd->GetDecayBR (3)   ---> 0.988
```

```
$ listModes ()
```

← Lista rozpadów

```
(...)
Database key=143
Database name=pi0 --> photon + photon
Decay index=3
Branching ratio=0.988000
Decay product 1->Database name=g
Decay product 2->Database name=g

Database key=144
Database name=pi0 --> dilepton + photon
                                    (Dalitz)
Decay index=4
Branching ratio=0.012000
Decay product 1->Database name=g
Decay product 2->Database name=dilepton
```

Dostęp do danych o cząstkach i rozpadach c.d.

Dostęp do modeli rozpadów, rozkładów:

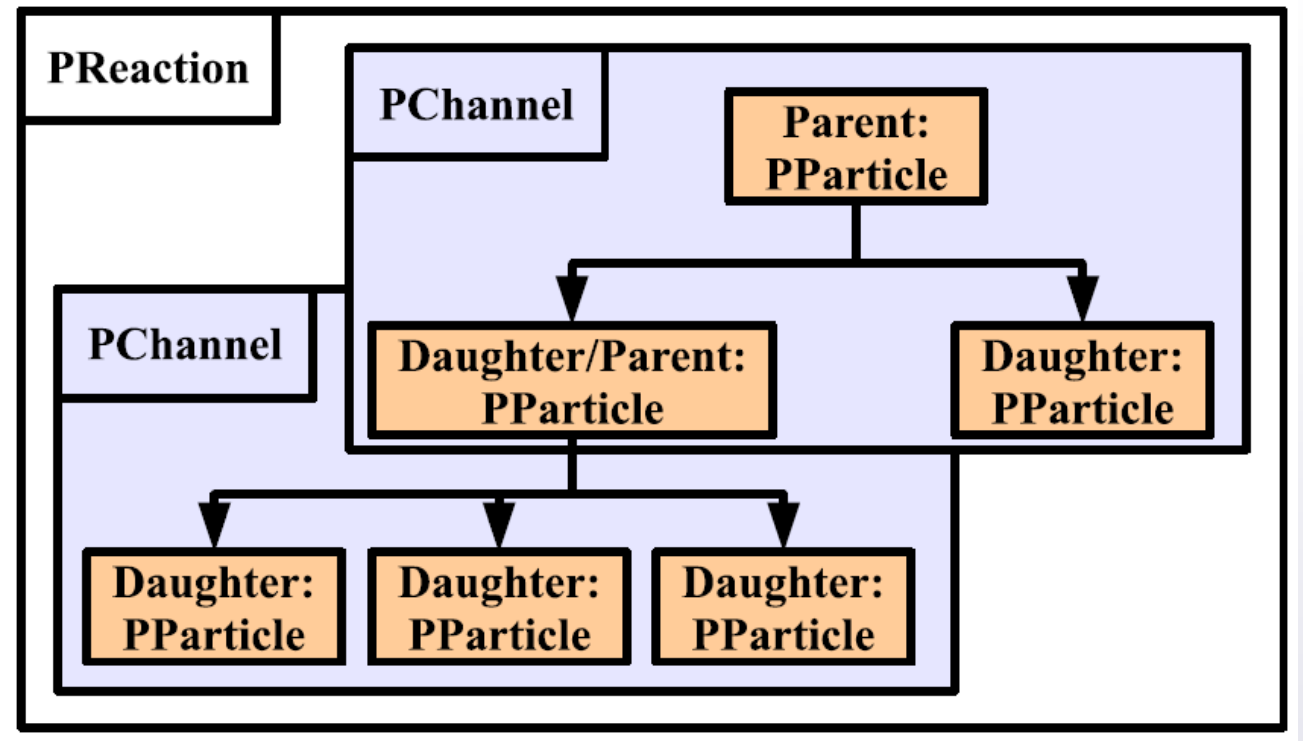
```
PDistributionManager* pdist = makeDistributionManager ()
pdist->Print ()
pdist->Print ("decay_models")
pdist->Print ("pi0_fixed_g_g")
```

nb. GenBod, cernlib.web.cern.ch/cernlib/mc/genbod.html

Przykład 3 Rozpad $\pi^+ \longrightarrow \mu^+ \nu_\mu$. Tym razem zbudujmy reakcję z klocków.

(E_k = 1 GeV)

Schemat pojęciowy:



- **Algorytm w naszym przypadku:**

1. Tworzymy 3 cząstki: π^+, μ^+ i ν, reprezentowane przez obiekty klasy `PParticle`.
2. Grupujemy cząstki: tworzymy tablicę, do której wstawiamy adresy obiektów (wpierw substrat)
3. Tworzymy 1 kanał rozpadu $\pi^+ \longrightarrow \mu^+ \nu$, reprezentowany przez obiekt klasy `PChannel`.
4. Łączymy wszystkie kanały w tablicę, do której wstawiamy adresy obiektów PChannel
   (U nas: tylko 1 kanał $\longrightarrow$ tylko 1 element tablicy)
5. Definiujemy reakcję: tworzymy 1 obiekt klasy `PReaction`, podając tablicę kanałów i plik wyjściowy.
6. Uruchamiamy pętlę.

Zaprogramujmy algorytm w makrze:

Substrat

Produkty

Tablica cząstek

Kanał rozpadu

Tablica wszystkich kanałów – uczestników reakcji

Definiujemy reakcję

Pętla 10000 zdarzeń

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int pip_mupnu_long ()
{
    PParticle pip ("pi+", 1.0 );
    PParticle mup ("mu+") , nu ("nu");

    PParticle* parttable[] = { &pip , &mup , &nu };

    PChannel* pip_decay = new PChannel ( parttable , 2 );

                                // "2" = no. of decay particles

    PChannel* tab_of_chan[] = { pip_decay };

    PReaction* my_reaction = new PReaction
                ( tab_of_chan , "pip_mupnu_long",

                  1 ,           /* No of channels in table */
                  1 , 0 , 0 , 0 );

    my_reaction->Print ();
    my_reaction->Loop (10000, 1);

    return 0;
}
```

Przykład 4 Emisja cząstki ze zderzenia 2 cząstek.

Rozpatrzmy reakcję: p (E_{kin} = 0.2797 GeV) + p ⟶ p + p + π^0

- Jakościowo nowa (dwa substraty). Użyjemy dedykowanego konstruktora klasy `PReaction`.

- Skąd ta "dziwna" wartość 0.2797 GeV? To najmniejsza E_{kin} , jaką powinien mieć nukleon wiązki, aby wyprodukować π^0 (z przeżyciem protonów). *"Energia progowa / Threshold energy"*
  Dla substratów o masach m_1 i m_2 i produkcie o masie m_X ,

$$E_K^{Min} = \frac{m_X^2 + 2m_X(m_1 + m_2)}{2m_2}$$

  Sprawdź, że gdy $E_k(p^+)$ zmniejszyć o 0.1 MeV, to Pluto odmówi utworzenia π^0.

- Jednak proces się na tym nie zakończy. π^0 nie jest trwały i po czasie $\tau \approx 10^{-16}$ s rozpadnie się.
  Dominujący kanał : $\pi^0 \longrightarrow \gamma\gamma$ (BR ≈ 99%)
  Uwzględnijmy go w symulacji.

W "" : E_{kin} cząstki wiazki
Bez "" : pęd cząstki wiązki

Substraty (wiązka i tarcza)

Przebieg reakcji (string descriptor)

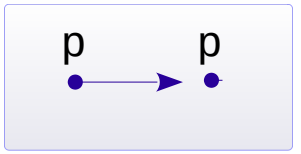
Plik wyjściowy

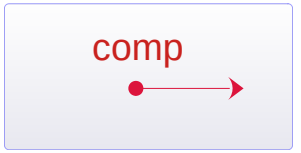
Uruchamiamy pętlę

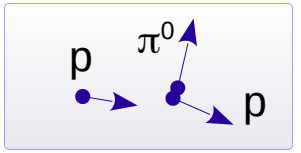
```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int pp_pppi0_gg ()
{
    PReaction* my_reaction = new PReaction
          ("0.2797",
           "p", "p",
           "p p pi0 [g g]",
           "pp_pppi0_gg",
            1, 0, 0, 0 );

    my_reaction->Print ();
    my_reaction->Loop (10000, 1);
    return 0;
}
```

```
$ data->Scan ("Name():pid:GetSiblingIndex()")

*         0 *         0 *           * *      14014 *        -1 *
*         0 *         1 *         p *         14 *         2 *
*         0 *         2 *         p *         14 *         3 *
*         0 *         3 *       pi0 *          7 *         1 *
*         0 *         4 *         g *          1 *         5 *
*         0 *         5 *         g *          1 *         4 *
```

Nieznane PID...?







- **Composite particle (cząstka złożona)**

Produkty i substraty łączy *Energia dostępna* ($\sqrt{s} = \sqrt{(\sum E_i)^2 - (\sum \vec{p}_i)^2}$) .
Jest niezmiennikiem transformacji Lorentza.
W szczególności w układzie CM z definicji $\sum \vec{p}_i = \vec{0} \quad \Rightarrow \quad \sqrt{s} = \sum E_i$
Jest to więc *zasób energii* dostępny w układzie CM.
Z tego zasobu generowane są cząstki–produkty, które dodatkowo mogą się poruszać.

Composite Particle = "cząstka zastępcza", dla której: $\vec{p}^{CM} = \sum \vec{p}_i^{CM} \quad E^{CM} = \sum E_i^{CM} \quad M = \sqrt{(\sum E_i)^2 - (\sum \vec{p}_i)^2}$

Z niej Pluto tworzy produkty, o ile na rozpad zezwolą zasady zachowania.

$PID_{CP} = 1000 \times PID_{2\ (Target)} + PID_{1\ (Beam)}$.

```
$ data->Scan ("E() : P() : M()", " pid==14014 ")
*         0 *           0 * 2.1562446 * 0.7765961 * 2.0115390 *
*         1 *           0 * 2.1562446 * 0.7765961 * 2.0115390 *
```

Jak widać,
$E^2 - P^2 = M^2$

```
$ data->Draw ("Beta()", " pid==14014 {lub 7} ")
$ data->Draw ("Rapidity()", " pid==7  {lub 1} ")
$ data->Draw ("Pt() : Rapidity()", " pid==7  {lub 1} ")
```

Rozkład β, *y dla* pionów koncentruje się wokół β, *y* cząstki złożonej.

Przykład 5 Emisja ze zderzenia 2 cząstek – konstrukcja “z klocków”

Substraty (wiązka + tarcza)

Composite particle

Produkty

Tablica cząstek uczestników pp → ppπ⁰

Kanał dla pp → ppπ⁰

Liczba produktów (3)

Tablica cząstek uczestników π⁰ → γγ

Tablica wszystkich kanałów

Definicja całej reakcji

2 = liczba kanałów

Pętla po zdarzeniach

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int pp_pppi0_gg_long ()
{
  PParticle p1 ("p", 0.2797) , p2 ("p");
  PParticle comp = p1 + p2;
  PParticle p3 ("p"), p4 ("p"), pi0 ("pi0");

  PParticle* parttab_pi0[] =
             { &comp , &pi0 , &p3 , &p4 };

  PChannel* pi0_prod = new PChannel (
             parttab_pi0 ,
             3 );

  PParticle g1 ("g") , g2 ("g");
  PParticle* parttab_gg[] = {&pi0, &g1 , &g2};
  PChannel* pi0_ggdecay = new PChannel (
            parttab_gg, 2 );

  PChannel* tab_of_chan[] =
            {pi0_prod , pi0_ggdecay};

  PReaction* my_reaction = new PReaction
         ( tab_of_chan , "pp_pppi0_gg_long",
           2 ,
           1 , 0 , 0 , 0 );

  my_reaction->Print ();
  my_reaction->Loop (10000, 1);
  return 0;
}
```

Kanał pp → ppπ⁰

Kanał π⁰ → γγ

Cała reakcja

Analiza wyjściowego TTree w makrze (Przykład 6)

- Przeanalizujemy drzewo wyjściowe z reakcji $\pi^+ \longrightarrow \mu^+ \ \nu_\mu$.

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int pip_mupnu_readtree ()
{
  TClonesArray* partArray = new TClonesArray ("PParticle", 10);
  PParticle* Part[10];
  TVector3 b;

  TFile* fileIn = new TFile ("pip_mupnu.root");
  TTree* tree = (TTree*) fileIn->Get ("data");
  tree->SetBranchAddress ("Particles", &partArray );

  Int_t nR = tree->GetEntries() ;
  cout << "\n * Analysing " << nR << " reactions.\n\n";

  for (Int_t iR = 0 ; iR < nR ; iR++ )
  {
    tree->GetEntry (iR);
    for (Int_t iP = 0; iP < partArray->GetEntries() ; iP++ )
      Part[iP] = (PParticle*) partArray->At (iP);

    Part[1]-> Boost ( -Part[0]->BoostVector() );
    Part[2]-> Boost ( -Part[0]->BoostVector() );

    for (Int_t iP = 1; iP <= 2; iP++)
      cout << Part[iP]->Name() <<'\t'<< Part[iP]->Px() <<'\t'
           << Part[iP]->Py()   <<'\t'<< Part[iP]->Pz() << endl;

    cout << endl;
  }
  return 0;
}
```

Pętla po reakcjach

Wczytanie i-tej reakcji

Pętla po cząstkach

4Pędy produktów transformujemy do układu własnego rodzica

Wypis danych

Produkcja cząstek ze strefy zderzenia ciężkich jonów

Zagadnienie: produkcja nowych cząstek
w zderzeniach ciężkich jonów
przy E_{kin} od 100 MeV/nukleon do LHC.

- Krotnością produkcji wielu cząstek w zderzeniu wydaje się rządzić jedna wielkość: √s .

- Jest to zgodne z podejściem statystycznym do strefy zderzenia ("Fireball" / "kula ognista"), złożonej z cząstek (fermionów i bozonów) o danych energiach–pędach.
Zakłada się równowagę termodynamiczną całości.

- Wyprowadzony rozkład emisji cząstek (F-D lub B-E) zwykle upraszcza się do *rozkładu Boltzmanna*.

$$\frac{dN}{dE} \propto p\, E\, e^{-E/T}$$

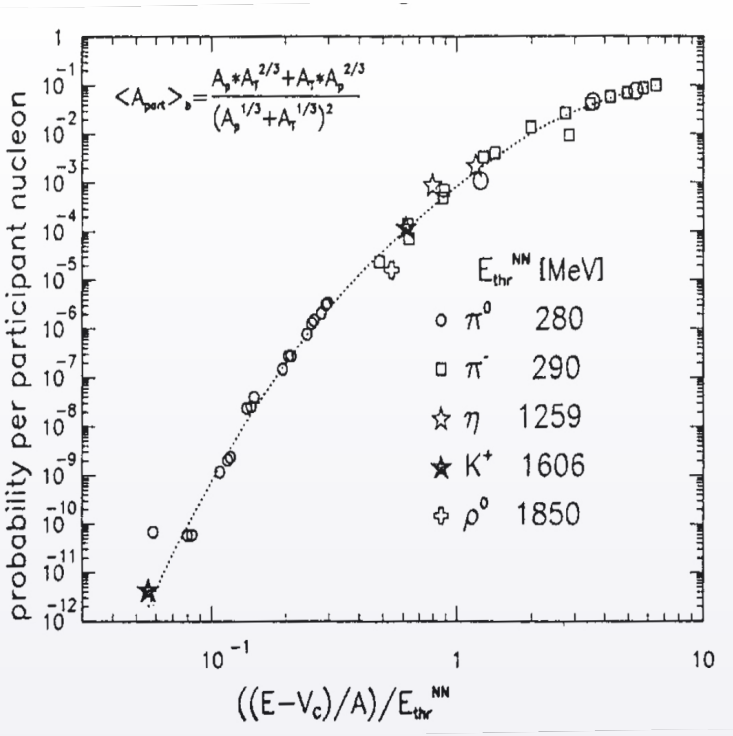
Rozkład Boltzmanna:

(1) jest izotropowy (ekwipartycja energii)
(2) zachodzi <E> = $^3/_2$ kT (nie zależy od *m* cząstki).

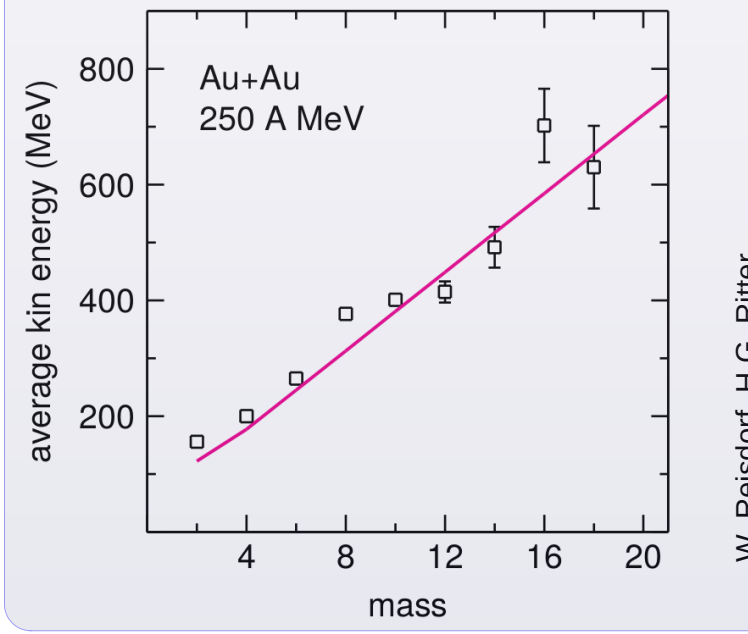
- Bardzo często te cechy nie są spełnione.

Rozkłady pędowe niektórych cząstek wyraźnie dają się opisać dopiero po przyjęciu sumy 2 rozkładów (2 źródeł term.)

Zależność <E> = f(m) → daje asumpt do hipotezy:
oprócz ruchu termicznego jest też
wspólna ekspansja radialna (ruch kolektywny).

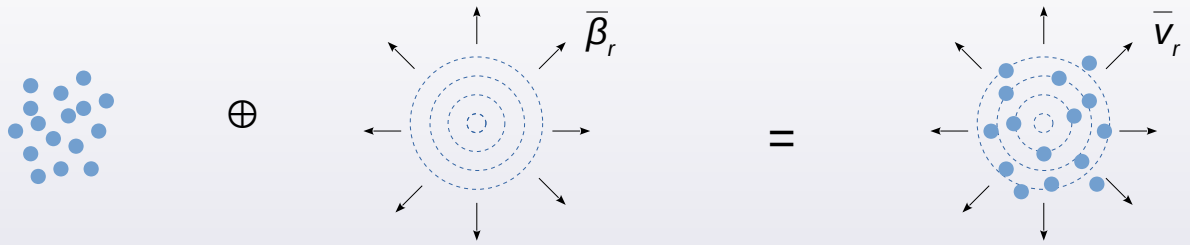


V.Metag, Prog. Part. Nucl. Phys. 30, 75 (1993)



W. Reisdorf, H.G. Ritter,
Annu. Rev. Nucl. Part. Sci. 47, 663 (1997)

Model Siemensa-Rasmussena: opisuje złożenie ruchu termicznego (T) z ekspansją radialną o prędkości β.



ruch termiczny (Boltzmann) ⊕ ruch kolektywny (ekspansja. radialna) = Złożenie

$$\left.\frac{d^3n}{dp^3}\right|_{CM} = \frac{N_i}{Z(T)} \cdot e^{-\frac{\gamma_r E}{T}} \cdot \left[\left(\gamma_r + \frac{T}{E}\right)\frac{sh\,\alpha}{\alpha} - \frac{T}{E}ch\,\alpha\right]$$

gdzie: $\alpha = \gamma_r \beta_r p/T$

N_i, Z : stałe normalizacyjne

P.J. Siemens, J.O. Rasmussen, Phys. Rev. Lett. 42, 880 (1979)

Rozkłady kątowe (w ϑ i φ) często nie są izotropowe.

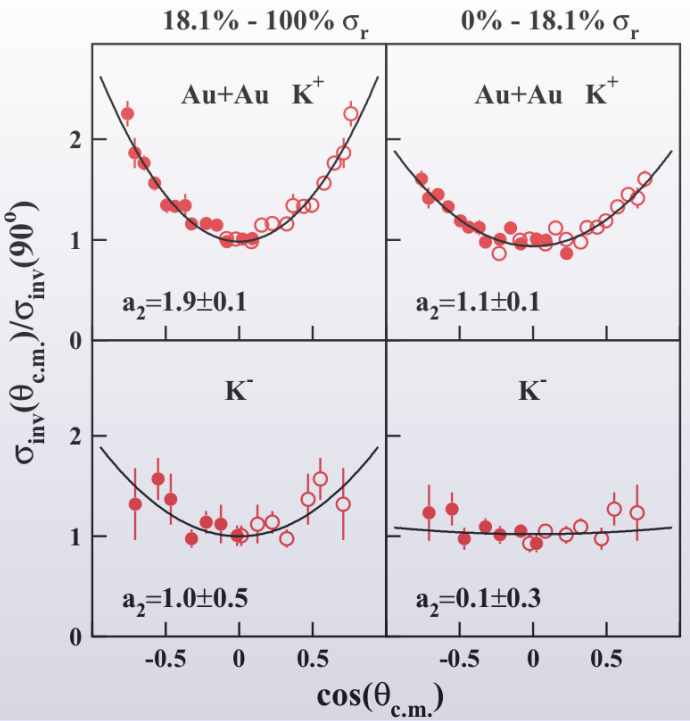
Rozkład w kącie ϑ jest fenomenologicznie parametryzowany rozkładem Legendre'a z kolejnymi (parzystymi) współczynnikami.

$$\frac{dN}{d\cos\theta_{NN}} \sim 1 + \sum_n a_n P_n(\cos\theta_{NN})$$

Rozkład w kącie φ jest fenomenologicznie parametryzowany rozkładem Fouriera z kolejnymi współczynnikami:

$$\frac{dN}{d\varphi} \sim \frac{1}{2\pi}\left(1 + 2\sum_{n\geq 1}^{\infty} v_n \cos n\varphi\right)$$

W Pluto można zadać dominujące w/w przyczynki.



A. Förster et al., Phys. Rev. C 75, 024906 (2007)

Produkcja cząstek ze strefy zderzenia ciężkich jonów

Wzorzec części energetycznej rozkładu: suma 2 funkcji Siemens-Rasmussena.

$$\frac{dN}{dE} \propto p\,E \left\{ f e^{-\gamma_r \frac{E}{T_1}} \left[\left(\gamma_r + \frac{T_1}{E}\right) \frac{\sinh\alpha_1}{\alpha_1} - \frac{T_1}{E}\cosh\alpha_1 \right] + (1-f) e^{-\gamma_r \frac{E}{T_2}} \left[\left(\gamma_r + \frac{T_2}{E}\right) \frac{\sinh\alpha_2}{\alpha_2} - \frac{T_2}{E}\cosh\alpha_2 \right] \right\}$$

Uwaga: (1) przy β → 0 funkcja Siemensa-Rasmussena → funkcja Boltzmanna.
(2) Dla f = 1 występuje tylko pierwsze źródło.

Wzorce rozkładu w θ i φ: $\frac{dN}{d\cos\theta_{CM}} \sim 1 + a_2\cos^2\theta_{CM} + a_4\cos^4\theta_{CM}$ $\frac{dN}{d\varphi_{RP}} \sim 1 + v_1\cos\varphi_{RP} + v_2\cos^2\varphi_{RP}$

PFireball – źródło emisji "naszych" cząstek.

Programistycznie, obiekt PFireball "udaje cząstkę", z której wypada "nasza" cząstka
=> trzeba zaimplementować kanał PChannel, który złączy obiekt PFireball i cząstkę wyemitowaną.

```
PFireball* source_pi0 = new PFireball (
  "pi0",  /* typ cząstki emitowanej ze źródła                              */
  0.040,  /* Ekin wiązki / nukleon  [GeV/A]                                */
  0.015,  /* Temperatura źródła termicznego cząstki [GeV]                  */
  0.,     /* Temperatura 2. źródła termicznego, jeśli istnieje [GeV]       */
  1.,     /* Waga 1. źródła                                                */
  0.,     /* Param. Beta (v/c) ekspansji radialnej źródła, jeśli istnieje  */
  0.,     /* Współczynnik a2 rozkładu kąta theta                           */
  0.,     /* Współczynnik a4 rozkładu kąta theta                           */
  0.,     /* Współczynnik v1 rozkładu kąta phi                             */
  0.      /* Współczynnik v2 rozkładu kąta phi                             */
);
```

Przykład 7

Emisja mezonów π^0 ze strefy zderzenia o temperaturze 15 MeV
Jon wiązki poruszał się z energią 40 MeV na nukleon.

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int thermal_pi0_gg ()
{
  PFireball* source_pi0 = new PFireball (
      "pi0", 0.040,  0.015,
      0., 1., 0.,
      0., 0.,
      0., 0.
  );
  source_pi0 ->Print ();

  PParticle* pi0 = new PParticle ("pi0");
  PParticle* parttable_fireball_pi0[] = {source_pi0, pi0};
  PChannel*  chan_fireball_pi0 = new PChannel (parttable_fireball_pi0, 1, 1);

  PParticle* g1  = new PParticle ("g");
  PParticle* g2  = new PParticle ("g");
  PParticle* parttable_pi0_gg [] = {pi0, g1, g2};
  PChannel*  chan_pi0_gg = new PChannel (parttable_pi0_gg, 2, 1);

  PChannel*  tab_of_chan[] = { chan_fireball_pi0 , chan_pi0_gg };

  PReaction* reaction_thermal_pi0 = new PReaction (
    tab_of_chan, "thermal_pi0_gg_Eb40MeV_T15MeV",
     2,
     1, 0, 0, 0
    );
   reaction_thermal_pi0 ->Print ();
   reaction_thermal_pi0 ->loop (10000, 1);
   return 0;
}
```

Definicja Fireballu

Typ cząstki, E_{kin} wiązki [GeV], T źródła [GeV]
T 2. źródła [GeV], wkład 1. źródła, β radialna
Współczynniki a_2 i a_4 rozkładu kąta θ
Współczynniki v_1 i v_2 rozkładu kąta φ

Złożenie Fireballu i π^0 w kanał

Złożenie π^0 oraz γγ w kanał

Tablica kanałów

Definicja reakcji

Liczba kanałów w tablicy

Podgląd danych w bazie TNtuple i histogramach THnF

Pluto umożliwia zdefiniowanie zmiennych fizycznych dot. uczestniczących cząstek, a następnie ich **podgląd w bazie TNtuple**. Uwaga: 1 reakcja = 1 wpis do bazy.

① Tworzymy bazę TNtuple i dedykowany (osobny) plik wyjściowy:

```
TFile*   file_TNtuple = new TFile   ("ntufile.root", "RECREATE");
TNtuple* my_ntuple    = new TNtuple ("ntu", "Name", "var1:var2:var3:..." );
```

② Definiujemy zmienne fizyczne:

```
my_reaction->Do ("thgam = [g]->Theta()* TMath::RadToDeg()" );
my_reaction->Do ("phgam = [g]->Phi()  * TMath::RadToDeg()" );
```

③ Polecamy reakcji generowanie wyjściowej bazy TNtuple. Można tu zadać **filtr** :

```
my_reaction->Output (my_ntuple , "if (thgam>20 && thgam < 50) " );
```

④ Można też nakładać **filtry** na główne drzewo (`TTree data`). Każda zmienna o nazwie poprzedzonej przez `#` jest filtrem. Wpis do `TTree data` nastąpi wtedy, gdy żaden z filtrów nie wyniesie 0.

```
my_reaction->Do ("#myaccept= 1; if (thgam<20 || thgam > 50) ; #myaccept= 0");
```

Podgląd przez kontrolne histogramy 1/2/3-wymiarowe (THnF)

① Tworzymy histogram:

```
TH1F* my_histo = new TH1F ("myhisto", "Photon azim. angle", 90, -180., 180.);
```

② Definiujemy w reakcji specjalne zmienne **_x (_y, _z)**. Możemy tu zadać filtr (uwaga: osobny od TNtuple):

```
my_reaction->Do (my_histo, "if thgam>20 && thgam < 50; _x=phi");
```

Uwaga: Zapis TNtuple i THnF do pliku, trzeba zwykle jawnie polecić :

```
gDirectory->cd ("ntufile.root:/");
my_ntuple ->Write ();                    my_histo ->Write();
```

Przykład 8 Rozpraszanie Comptona – podgląd zmiennych w TNtuple i THnF

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int compton ()
{
  PReaction* R = new PReaction ("0.010",
            "g", "e-", "g e-", "compton_ttree");
  R->Print ();

  TFile* file_TNtuple = new TFile ("compton_tntuple.root", "RECREATE");

  TNtuple* my_ntuple = new TNtuple ("ntu", "Monitor ntuple",
                       "thgam:phgam:thel:phel:opang" );

  R->Do ("RTD = TMath::RadToDeg()");
  R->Do ("thgam = [g]->Theta()    * RTD");
  R->Do ("phgam = [g]->Phi()      * RTD");
  R->Do ("thel  = [e-]->Theta()   * RTD");
  R->Do ("phel  = [e-]->Phi()     * RTD");
  R->Do ("opang = [g]->Angle([e-])* RTD");
  R->Output (my_ntuple, "if (thgam>20 && thgam < 50)" );

  R->Do ("#myAccept=1; if (thgam<20 || thgam > 50); #myAccept=0");

  TH1F* my_histo = new TH1F ("myhisto", "Opening angle", 90, 0., 90.);
  R->Do (my_histo , "if thgam > 20 && thgam < 50 ; _x = opang");

  R->Loop (10000, 1);
  gDirectory->cd ("ntufile.root:/");
  my_ntuple->Write ();
  my_histo ->Write ();

  return 0;
}
```

Definicje zmiennych

Filtr w TNtuple

TNtuple

Filtr TTree

TH1F

Zapis

Język skryptowy

Dedykowany przewodnik:

https://www.fuw.edu.pl/~kpias/nkfj/pluto/ifroehlich_pluto_script_manual.pdf

Dodawanie nowych cząstek do bazy danych Pluto

Aby dodać nową cząstkę i ew. zapewnić jej oczekiwane rozpady,

① Zaczynamy od utworzenia obiektu z bazą danych.

```
$ PStaticData* sd = makeStaticData() ;
```

② Podajemy PID, nazwę i masę cząstki. PID = -1 przydzieli pierwszy wolny numer. Ew. dodajemy szer. Γ .

```
$ sd->AddParticle ( -1, "New", mass [GeV]);
$ sd->SetParticleTotalWidth ( "New", width [GeV]);
```

③ Przydzielamy istotne cechy cząstki.
Uwaga: brak ważnych danych może skutkować brakiem modelu rozpadu i błędnymi rozkładami.

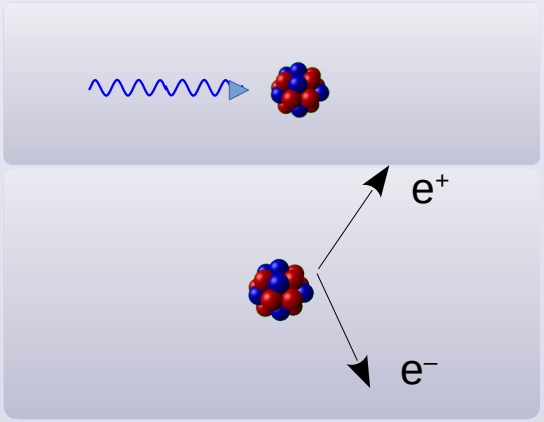
```
$ sd->SetParticleBaryon ("New", 1);
                Meson ( )     Charge ( )    Spin ( )     Isopin ( )   ....
```

④ Dla cząstek nietrwałych: wskazujemy kanał(y) rozpadu. PID = -1 przydzieli pierwszy wolny nr. z listy.

```
$ sd->AddDecay (-1, "New -> b + c", "New", "b,c", 1.);
```

Przykład 9 Kreacja par e^+e^- w pobliżu jądra atomowego

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int gamma_pairproduction ()
{
  double mN = 0.938, A = 40, mass_nucl = mN * A;
  PStaticData* sd = makeStaticData() ;
  sd->AddParticle (-1, "A", mass_nucl );

  PReaction* R = new PReaction ( "1.", "g", "A", "A e+ e-", "gamma_ee_ttree");
  R->Print ();

  TFile* file_TNtuple = new TFile ("gamma_pairprod_tntuple.root", "RECREATE");
  TNtuple* my_ntuple = new TNtuple
         ("ntu",  "My Ntuple", "pxep:pyep:pzep:pxem:pyem:pzem:pxA:pyA:pzA" );

  R->Do ("RTD = TMath::RadToDeg()");
  R->Do ("pxep = [e+]->Px() * RTD");
  R->Do ("pyep = [e+]->Py() * RTD");
  R->Do ("pzep = [e+]->Pz() * RTD");
  R->Do ("pxem = [e-]->Px() * RTD");
  R->Do ("pyem = [e-]->Py() * RTD");
  R->Do ("pzem = [e-]->Pz() * RTD");
  R->Do ("pxA  = [A]->Px()  * RTD");
  R->Do ("pyA  = [A]->Py()  * RTD");
  R->Do ("pzA  = [A]->Pz()  * RTD");

  R->Output (my_ntuple);
  R->Loop (10000, 1);

  file_TNtuple->Write ();
  file_TNtuple->Close ();
  return 0;
}
```



Nowa cząstka