

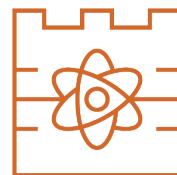


POLITECHNIKA KRAKOWSKA

im. Tadeusza Kościuszki

Wydział Inżynierii Materiałowej i Fizyki

Katedra Fizyki



Kierunek studiów: Fizyka Techniczna

Specjalność: Modelowanie komputerowe

STUDIA STACJONARNE

PRACA DYPLOMOWA

MAGISTERSKA

Mateusz Myśliwiec

135310

BADANIA ROZPADÓW WYSOKOENERGETYCZNYCH STANÓW
WZBUDZONYCH W LEKKICH JĄDRACH ATOMOWYCH
PRODUKOWANYCH W REAKCJI NIEELASTYCZNEGO
ROZPRASZANIA PROTONÓW

A STUDY OF THE DECAYS OF HIGH-ENERGY EXCITED STATES IN
LIGHT ATOMIC NUCLEI PRODUCED IN THE REACTION OF
INELASTIC SCATTERING OF PROTONS

Promotor pracy dyplomowej:
dr Natalia Cieplicka-Oryńczak

Recenzent pracy dyplomowej:
dr hab., prof. PK Agnieszka Łuszczak

Kraków, rok akademicki 2023/24

Załącznik nr 5 do Zarządzenia nr 15 Rektora PK z dnia 18 lutego 2022 r.

 Kraków, dnia 2024-06-21

MATEUSZ MYŚLIWIEC

imię i nazwisko

D135310|

nr albumu

Wydział Inżynierii Materiałowej i Fizyki

wydział PK

Fizyka Techniczna

kierunek studiów

stacjonarne II stopnia

forma studiów i poziom kształcenia

OŚWIADCZENIE O SAMODZIELNYM WYKONANIU PRACY DYPLOMOWEJ

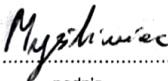
Oświadczam, że przedkładana przeze mnie praca dyplomowa magisterska/inżynierska/licencjacka* przydzielony fragment pracy dyplomowej, w przypadku pracy dyplomowej dwuautorskiej, pt.:**Badania rozpadów wysokoenergetycznych stanów wzbudzonych w lekkich jądrach atomowych produkowanych w reakcji nieelastycznego rozpraszań protonów**

została napisana przeze mnie samodzielnie. Jednocześnie oświadczam, że ww. praca:

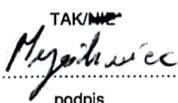
- 1) nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 r.o prawie autorskim i prawach pokrewnych (Dz.U. z 2021 r. poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym, a także nie zawiera danych i informacji, które uzyskałem/sem* w sposób niedozwolony,
- 2) nie była wcześniej podstawą żadnej innej procedury związanej z nadawaniem tytułów zawodowych, stopni lub tytułów naukowych.

Jednocześnie wyrażam zgodę na:

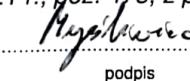
- 1) poddanie mojej pracy kontroli za pomocą systemu Antyplagiat oraz na umieszczenie tekstu pracy w bazie danych uczelni, w celu ochrony go przed nieuprawnionym wykorzystaniem. Oświadczam, że zostałem/sem* poinformowany/-a* i wyrażam zgodę, by system Antyplagiat porównywał tekst mojej pracy z tekstem innych prac znajdujących się w bazie danych uczelni, z tekstami dostępnymi w zasobach światowego Internetu oraz z bazą porównawczą systemu Antyplagiat,
- 2) to, aby moja praca pozostała w bazie danych uczelni przez okres wynikający z przepisów prawa. Oświadczam, że zostałem poinformowany i wyrażam zgodę, że tekst mojej pracy stanie się elementem porównawczej bazy danych uczelni, która będzie wykorzystywana, w tym także udostępniana innym podmiotom, na zasadach określonych przez uczelnię, w celu dokonywania kontroli antyplagiatowej prac dyplomowych/doktorskich, a także innych tekstów, które powstaną w przyszłości.


podpis

- 3) Wyrażam zgodę na udostępnianie mojej pracy dyplomowej w Akademickim Systemie Archiwizacji Prac na PK do celów naukowo-badawczych z poszanowaniem przepisów ustawy o prawie autorskim i prawach pokrewnych (Dz.U. z 2021 r. poz. 1062).

TAK/NIE

podpis

Jednocześnie przyjmuję do wiadomości, że w przypadku stwierdzenia popełnienia przeze mnie czynu polegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzej pracy, lub ustalenia naukowego, Rektor PK stwierdzi nieważność postępowania w sprawie nadania mi tytułu zawodowego (art. 77 ust. 5 ustawy z dnia 18 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce, (Dz.U. z 2021 r., poz. 478, z późn. zm.).


podpis

*) niepotrzebne skreślić

Streszczenie

Eksperyment związany z realizacją pracy polegał na zbadaniu rozpadu stanu rozciągniętego (ang. „stretched”) M4 przy energii 21.47 MeV w jądrze ^{13}C produkowanego w reakcji nieelastycznego rozproszenia protonów. Pomiary te zostały przeprowadzone w krakowskim Centrum Cyklotronowym Bronowice przy Instytucie Fizyki Jądrowej PAN. Stany „stretched” są jednymi z najprostszych znanych wzbudzeń jądrowych, a ich badania powinny dostarczyć przejrzystych informacji o szczegółach działania sił jądrowych. Jak dotąd brakuje zwłaszcza informacji na temat ścieżek rozpadów rezonansów M4 w lekkich jądrach. Do badania rozpadu stanu przy energii 21.47 MeV w izotopie ^{13}C wykorzystano szereg detektorów: rozproszone protony oraz kwanty γ były wykrywane w koincydencji odpowiednio przez detektory scyntylacyjne systemów KRATTA oraz PARIS, i cztery detektory LaBr_3 . Dodatkowo, w celu zbadania rozpadu rezonansów M4 przez pomiary koincydencyjne rozproszonego protonu oraz lekkiej cząstki naładowanej (proton, deuterон, alfa), zastosowano dwustronny paskowy detektor krzemowy (Double-sided Silicon Strip Detector). Przy takiej ilości danych potrzebne jest narzędzie, za pomocą którego jesteśmy w stanie z łatwością analizować wyniki eksperymentu. Niniejsza praca jest skoncentrowana na zoptymalizowaniu przedstawienia wyników pomiarów z wykonanych eksperymentów w środowisku ROOT przez utworzenie tzw. „drzewa” ROOT z zebranych danych pomiarowych. Struktura drzewa pozwala na szybkie sprawdzanie korelacji pomiędzy rozproszonymi protonami, kwantami gamma i lekkimi cząstками naładowanymi. Dzięki utworzeniu gałęzi zawierających skalibrowane energie i czasy detekcji możliwe jest szybkie wyświetlanie widm lub dwuwymiarowych macierzy wielkości zarejestrowanych przez detektory oraz dodawanie różnych warunków. Program został przetestowany na danych z eksperymentu poświęconego izotopowi ^{13}C , a w najbliższej przyszłości zostanie wykorzystany w trakcie badań poświęconych izotopowi ^{12}C .

Słowa kluczowe: stany rozciągnięte, analiza danych, TTree ROOT

Abstract

The thesis presents the data analysis from the experiment devoted to the study of the decay of the stretched M4 state at an energy of 21.47 MeV in the ^{13}C nucleus produced in the inelastic proton scattering reaction. The measurement was carried out at the Bronowice Cyclotron Centre at the Institute of Nuclear Physics PAN in Krakow. Stretched states are one of the simplest known nuclear excitations and their study should provide transparent information on the details of nuclear forces. So far, information on the decay paths of M4 resonances in light nuclei is particularly lacking. To study the decay of the state at an energy of 21.47 MeV in the ^{13}C isotope, a series of detectors was used: scattered protons and γ -quanta were detected in coincidence by the scintillation detectors of the KRATTA and PARIS systems, and four LaBr₃ detectors, respectively. In addition, to study the decay of M4 resonances by coincidence measurements of a scattered proton and a light charged particle (proton, deuteron, alpha), a double-sided silicon strip detector was used. With such an amount of data, we need a tool with which we can easily analyze the results of the experiment. This work is focused on optimizing the representation of measurement results from performed experiments in the ROOT environment by creating a so-called ROOT 'tree' from the collected data. The tree structure allows to quickly check the correlation between scattered protons, gamma quanta and light charged particles. By creating branches containing calibrated energies and detection times, it is possible to quickly display spectra or two-dimensional matrices of the quantities recorded by the detectors and add different conditions. The program has been tested on data from the experiment dedicated to the ^{13}C isotope and will be used in the near future during research dedicated to the ^{12}C isotope.

Keywords: stretched states, data analysis, TTree ROOT

Spis treści

1 Wstęp	1
1.1 Podstawowe właściwości jądra atomowego	2
1.2 Siły jądrowe	3
1.3 Wprowadzenie do rozpadów promieniotwórczych	4
1.3.1 Rozpad alfa	6
1.3.2 Rozpad beta	7
1.3.3 Wychwyt elektronu	9
1.3.4 Rozpad gamma	9
1.3.5 Emisja cząstek ze stanów wzbudzonych	10
1.4 Model powłokowy jądra atomowego	11
1.5 Rozcięgnięte stany w lekkich jądrach	13
2 Eksperyment i układ pomiarowy	16
3 Działanie programów tworzenia struktury danych oraz kalibracji czasowej detektora DSSSD	22
3.1 Tworzenie struktury TTree ROOT	22
3.2 Deklarowanie gałęzi i liści drzewa	24
3.3 Wczytywanie współczynników kalibracyjnych	26
3.4 Kalibracja macierzy do identyfikacji lekkich cząstek naładowanych	32
4 Analiza danych	47
4.1 Wyniki eksperymentalne pomiarów promieniowania gamma	47
4.2 Analiza korelacji pomiędzy detektorami oraz ich wpływ na dokładność pomiarów	52
4.3 Pomiar rozproszonych protonów	56
5 Podsumowanie	64
A Wydruk pliku nagłówkowego dla detektora krzemowego DSSSD, tu zdefiniowany jako Silicon, plik: Silicon.h	65
B Wydruk pliku nagłówkowego dla detektora Kratta, plik: Kratta.h	67
C Wydruk pliku nagłówkowego dla detektora Plastic, plik: Plastic.h	70
D Wydruk pliku nagłówkowego dla detektora LaBr, plik: LaBr.h	72

- E Wydruk programu do tworzenia struktury danych „drzewa ROOT”, plik:
rewrite_tree.C 74
- F Wydruk programu do kalibracji macierzy pomiarów naładowanych częstek z
detektora krzemowego DSSSD, plik: Peaks.C 89

Rozdział 1

Wstęp

Współczesne badania z zakresu fizyki jądrowej wymagają precyzyjnych narzędzi i zaawansowanych metod analizy danych, aby móc dokładnie zrozumieć złożone procesy zachodzące w jądrach atomowych. Eksperymenty często koncentrują się na badaniu specyficznych stanów jądrowych, takich jak na przykład stany rozciagnięte (ang. „stretched”) położone przy wysokich energiach wzbudzenia w lekkich jądrach. Stany te posiadają względnie prostą strukturę, a w związku z tym mogą być niezwykle istotne dla naszego zrozumienia natury sił jądrowych. Jak dotąd brakuje jednak szczegółowych informacji na temat rozpadów tych wzbudzeń, zwłaszcza w lekkich izotopach.

Przykładem eksperymentu, w którym badano rozpad stanu rozciagniętego przy energii 21.47 MeV w jądrze ^{13}C , jest pomiar przeprowadzony w Krakowskim Centrum Cyklotronowym Bronowice przy Instytucie Fizyki Jądrowej PAN. W ramach tego eksperymentu wykorzystano zaawansowane detektory, takie jak systemy KRATTA i PARIS oraz detektory LaBr₃, aby zbierać dane o rozproszonych protonach i kwantach gamma. Dodatkowo, zastosowano dwustronny paskowy detektor krzemowy (DSSSD) do pomiaru lekkich cząstek naładowanych, takich jak protony, deuterony czy trytony. Taka złożona infrastruktura pomiarowa generuje duże ilości danych, które wymagają efektywnych narzędzi do analizy.

Celem niniejszej pracy jest zoptymalizowanie analizy danych z eksperymentów poprzez za-programowanie dedykowanej struktury danych w środowisku ROOT, tzw. „drzewa”. Tworzenie „drzew” ROOT z zebranych danych pozwala na szybkie i efektywne sprawdzanie korelacji pomiędzy różnymi zarejestrowanymi wielkościami, jak również na skalibrowanie tych wielkości, co jest tutaj pokazane na przykładzie macierzy czasowych pozyskanych przy pomocy detektorów DSSSD. Skalibrowane macierze czasowe są kluczowe dla uzyskania informacji na temat identyfikacji lekkich cząstek naładowanych, ułatwiają ocenę jakości i poprawności działania całej struktury danych.

1.1 Podstawowe własności jądra atomowego

Jądro atomowe to centralna część atomu, która zawiera niemal całą jego masę. Składa się z dwóch rodzajów cząstek subatomowych: protonów, które są dodatnio naładowane, oraz neutronów, które nie mają ładunku elektrycznego. Protony i neutrony, nazywane nukleonami są związane ze sobą przez silne oddziaływanie jądrowe, które są znacznie silniejsze od oddziaływań elektromagnetycznych, lecz działają na bardzo krótkich dystansach, rzędu kilku femtometrów ($1 \text{ femtometr} = 10^{-15} \text{ metra}$). Liczba protonów w jądrze określa liczbę atomową (Z), która decyduje o właściwościach chemicznych pierwiastka, podczas gdy liczba neutronów może się różnić, prowadząc do istnienia różnych izotopów tego samego pierwiastka. Liczba masowa (A) jest równa liczbie nukleonów w jądrze i daje przybliżoną wartość masy jądra.

Podstawowe właściwości jąder atomowych można zrozumieć jako wynikające z działania siły przyciągania jądrowego pomiędzy nukleonami. Jądra są w przybliżeniu kuliste, co jest zgodnie z oczekiwaniemi przy sile przyciągania krótkiego zasięgu, i działa na podobnej zasadzie co siły międzycząsteczkowe w kropli wody. Model kropli został wcześniej wykorzystany do zrozumienia aspektów rozszczepienia jądrowego. Podobnie jak w kropli wody, cząsteczki w pobliżu powierzchni są słabiej związane, a efekt ten można opisać jako energię powierzchniową.

Jeśli wyobrażymy sobie duże, kuliste jądro o promieniu (R), złożone z (A) nukleonów, energia wiążania wynosi objętość V wynosi:

$$V = \frac{4}{3}\pi R^3 = \frac{4}{3}\pi R_o^3 A$$

gdzie $R = R_o A^{1/3}$, a R_o to parametr charakterystyczny dla rozmiarów jąder atomowych [1]. Energia wiążania B jądra jest różnicą energii masowej pomiędzy jądem i składającymi się na nie protonami Z i neutronami N :

$$B = (Zm_p + Nm_n - [m(^A X) - Zm_e])c^2$$

Grupując masy protonów i elektronów w neutralne atomy wodoru, możemy przepisać powyższe równanie jako

$$B = [Zm(^1 H) + Nm_n - m(^A X)] c^2$$

Ponieważ masy są zwykle podawane w jednostkach masy atomowej, wygodnie jest uwzględnić współczynnik konwersji jednostek w $c^2 = 931,50 \text{ MeV/u}$ [2].

Kolejnym aspektem jest odpychanie kulombowskie pomiędzy protonami. Energia odpychania elektrostatycznego jednorodnego, kulistego rozkładu ładunku $Q = Ze$ o promieniu R wynosi:

$$U_{\text{Coul}} = \frac{3}{5} \frac{k_e(Ze)^2}{R}$$

Stabilność jądra wymaga prawie równej liczby neutronów i protonów. Analogicznie do zasady wypełniania stanów w atomach, gdzie preferuje się po jednym elektronie o przeciwnych spinach w tym samym stanie, stabilność jądra jest osiągana przy zbliżonej liczbie protonów i neutronów. Energia odpychania kulombowskiego, tym większa im większe Z , jest powodem, dla którego liczba protonów Z jest mniejsza niż liczba neutronów N w cięższych jądrach [1] [2].

Możemy znaleźć gęstość masy materii jądrowej z wzoru $V = 4/3\pi R^3 = (4/3)\pi R_o^3 * A$, ponieważ masa wynosi $A * m_p = A * 1,67 * 10^{-27} \text{ kg}$. Ustawiając $R = 1 \text{ m}$, objętość wynosi

$(4/3)\pi R^3 = (4/3)\pi(1,2 * 10^{-15})^3 A m^3$. Gęstość wynosi $2,307 * 10^{17} kg/m^3$. Gęstość gwiazd neutronowych szacuje się na dwu- lub trzykrotność tej wartości. Wartości te są znacznie większe niż gęstość jądra słonecznego, która wynosi $1,62 * 10^5 kg/m^3$ [1].

Jądra atomowe, takie jak ^{238}U , mają często kilka izotopów, odpowiadających różnym liczbom neutronów N dla tego samego Z . Chemiczne własności są określane przez liczbę Z , będącą również liczbą elektronów, które gromadzą się wokół tego jądra.

1.2 Siły jądrowe

Siły jądrowe, znane również jako oddziaływanie silne, są fundamentalnym elementem współczesnej fizyki, odgrywającym kluczową rolę w wyjaśnieniu właściwości materii. Te niezwykle potężne oddziaływanie działają na poziomie subatomowym, odpowiadając za utrzymanie kwarków w hadronach, takich jak protony i neutrony, a także za spajanie nukleonów w jądrach atomowych. Siły te, mimo że działają na bardzo krótkich dystansach, są niezbędne do zrozumienia stabilności atomów oraz procesów zachodzących w gwiazdach. Badania eksperymentalne przeprowadzone dotychczas pozwoliły na sformułowanie następujących wniosków na temat charakterystyki sił jądrowych [2]:

- Za pomocą żadnego ze znanych klasycznych oddziaływań czy to grawitacyjnych czy elektromagnetycznych nie możemy wyjaśnić sił wiążących nukleony w jądrach. Wszystkie te oddziaływanie są zbyt słabe.
- Oddziaływanie nukleon-nukleon jest silnie zależne od spinu.
- Siła jądrowa musi spełniać pewne symetrie, takie jak parzystość ($r \rightarrow -r$) i odwrócenie czasu ($t \rightarrow -t$).
- Oddziaływanie nukleon-nukleon zawiera niecentralny człon, zwany potencjałem tensorowym. Dowody na istnienie siły tensorowej pochodzą głównie z obserwacji momentu kwadrupolowego stanu podstawowego deuteronu. Oddziaływanie nukleon - nukleon staje się odpychające na małych odległościach. Wynika to z jakościowych rozważań nad gęstością jądrową: gdy dodajemy więcej nukleonów, jądro rośnie w taki sposób, że jego centralna gęstość pozostaje w przybliżeniu stała, a zatem coś powstrzymuje nukleony przed zbyt bliskim stłoczeniem.
- Oddziaływanie nukleon-nukleon może również zależeć od względnej prędkości lub pędu nukleonów. Siły zależne od prędkości lub pędu nie mogą być opisane przez potencjał skalarny, ale można je uwzględnić, wprowadzając wyrażenia liniowe, kwadratowe i wyższych rzędów. Siła jądrowa wykazuje symetrię ładunkową. Oznacza to, że oddziaływanie proton-proton (pp) i neutron-neutron (nn) są identyczne po uwzględnieniu siły Coulomba w przypadku protonów. W tym kontekście „ładunek” odnosi się do typu nukleonu (protonu lub neutronu), a nie do jego ładunku elektrycznego.
- Siła nukleon - nukleon jest prawie niezależna od ładunku. Trzy siły jądrowe nn , pp i pn są identyczne, ponownie korygując o siłę Coulomba. Niezależność ładunkowa jest więc silniejszym wymogiem niż symetria ładunkowa.

1.3 Wprowadzenie do rozpadów promieniotwórczych

Rozpad promieniotwórczy jest procesem losowym. W związku z tym nie można z całą pewnością stwierdzić, kiedy niestabilny nuklid ulegnie rozpadowi. Prawdopodobieństwo rozpadu atomu w czasie dt jest określone przez λdt , gdzie λ jest stałą proporcjonalności znaną jako stała rozpadu. W granicy bardzo małych przedziałów czasowych można to wyrazić jako

$$\frac{dN}{dt} = -\lambda N$$

Całkowanie względem czasu daje liczbę atomów obecnych w dowolnym czasie t:

$$N(t) = N(0)e^{-\lambda t}$$

Liczba rozpadów na jednostkę czasu, tj. aktywność A , jest zdefiniowana przez

$$A = -\frac{dN}{dt} = \lambda N$$

W tej definicji należy zauważać, że zakłada się, że N maleje z powodu rozpadu. Jednostką aktywności jest Becquerel, 1 Bq = 1 rozpad na sekundę.

Okres połowicznego rozpadu nuklidu, oznaczany τ , jest właściwością statystyczną i jest ważną koncepcją tylko ze względu na bardzo dużą liczbę atomów. Jest używany do oznaczenia czasu, w którym liczba atomów zmniejszyła się do połowy wartości początkowej, $\frac{1}{2} = e^{-\lambda\tau}$. Stąd czas połowicznego rozpadu jest związany ze stałą rozpadu poprzez zależność

$$\tau = \frac{\ln 2}{\lambda} \approx \frac{0.93}{\lambda}$$

Do niektórych obliczeń wygodnie jest używać średniego czasu życia radionuklidu. Średni czas życia definiuje się jako sumę czasów życia poszczególnych atomów podzieloną przez całkowitą liczbę atomów obecnych pierwotnie. W przedziale czasu od t do $t + dt$ całkowita liczba transformacji wynosi $\lambda N dt$. Każdy atom, który rozpadł się w tym przedziale czasu, istniał przez całkowity czas życia t . Suma czasów życia wszystkich atomów, które zostały przekształcone w przedziale czasu dt , po przetrwaniu od $t = 0$, wynosi $t \lambda N dt$. Średni czas życia jest wtedy dany przez

$$l = \frac{1}{(N(0))} \int_0^\infty t \lambda N dt$$

Łatwo jest wykazać, że zależność między średnim czasem życia a okresem połowicznego rozpadu wynosi $l = 1,44\tau$.

Wiele nuklidów ma więcej niż jeden sposób rozpadu. Rozważmy nuklid, w którym występują dwa takie sposoby. Prawdopodobieństwo, że jądro rozpadnie się w procesie 1 w czasie dt wynosi $\lambda_1 dt$. Analogicznie, prawdopodobieństwo, że rozpadnie się w procesie 2 w czasie dt wynosi $\lambda_2 dt$. Stąd równanie rządzace rozpadem promieniotwórczym można zapisać jako

$$\frac{dN}{dt} = -(\lambda_1 + \lambda_2)N$$

Całkowita stała rozpadu dla rozpadu nuklidu macierzystego jest sumą częściowych stałych rozpadu, tj. $\lambda = \lambda_1 + \lambda_2$. Stąd współczynniki rozgałęzienia dla kanałów 1 i 2 są zdefiniowane jako

$$BR_1 = \frac{\lambda_1}{\lambda}, \quad BR_2 = \frac{\lambda_2}{\lambda}$$

Ogólnie rzecz biorąc, stosunek rozgałęzień (BR, ang. „branching ratio”) dla określonego trybu rozpadu definiuje się jako stosunek liczby jąder rozpadających się w tym trybie rozpadu do liczby jąder rozpadających się ogółem, tj.

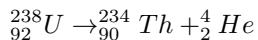
$$BR_i = \frac{\lambda_i}{(\lambda_1 + \lambda_2 + \dots + \lambda_i + \dots)} = \lambda_i / \lambda$$

Alternatywnie, biorąc pod uwagę całkowitą stałą rozpadu, „częściowa” stała rozpadu jest dana przez

$$\lambda_i = BR_i * \lambda$$

Bardzo często zdarza się, że produkt pochodny rozpadu jądrowego jest również radioaktywny. W takich przypadkach mówi się o szeregach promieniotwórczych. Jako przykład rozważmy szereg rozpadu $N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow \dots$, w którym początkowy lub „macierzysty” nuklid N_1 rozпадa się na pochodną N_2 . Ten z kolei jest radioaktywny i rozпадa się na N_3 . Mówiąc bardziej ogólnie, każdy nuklid w łańcuchu rozpadu N_i może „rozgałęziać się”, ze stosunkiem rozgałęzień λ_{N_i, N_j} , na więcej niż jedną pochodną. Ponadto może istnieć zewnętrzne źródło S_i do produkcji N_i (poza rozpadem pierwiastka macierzystego).

Energia rozpadu to całkowita energia uwalniana podczas rozpadu promieniotwórczego. Reakcja rozpadu promieniotwórczego jest szczególnym przypadkiem binarnej reakcji jądrowej $a + X \rightarrow Y + b$, w której nie ma cząstki a , a X jest w spoczynku. Energia rozpadu jest właśnie wartością Q dla tego typu reakcji. Rozważmy, że rozpad promieniotwórczy ^{238}U jest powszechnie zapisywany w postaci:



a wartość Q dla tej reakcji jako:

$$\begin{aligned} Q &= M(^{238}_{92}U) - M(^{234}_{90}Th) - M(^4_2He) = \\ &= (238,050788u) - (234,043601u) - (4,002603u) = 0,004584u = 4,27MeV \end{aligned}$$

Reakcje jądrowe są procesami, w których jądra atomowe oddziałują z innymi jądrami lub cząstками subatomowymi, prowadząc do zmiany ich składu. Przykłady takich reakcji to fuzja jądrowa, rozszczepienie jądrowe oraz różne reakcje wywołane bombardowaniem jąder przez cząstki takie jak neutrony, protony, czy cząstki alfa. Schematyczny zapis reakcji jądrowych to np. $\alpha + X \rightarrow Y + p$, gdzie α (cząstka alfa) bombarduje jądro X , co prowadzi do powstania jądra Y i emisji protonu (p). Inny przykład to reakcja neutronowa $n + X \rightarrow Y + \gamma$, gdzie neutron (n) oddziałuje z jądem X , prowadząc do powstania jądra Y i emisji fotonu gamma (γ).

Prawdopodobieństwo interakcji między bombardującą cząstką, a jądem atomowym jest opisywane przez pojęcie przekroju czynnego oznaczanego grecką literą σ . Nie ma gwarancji, że konkretny pocisk bombardujący wejdzie w interakcję z jądem docelowym, aby wywołać daną reakcję, σ stanowi jedynie miarę prawdopodobieństwa jej wystąpienia. Przekrój czynny zależy od właściwości jąder tarczy i padającego pocisku. Jeśli tarcza jest napromieniowana cząstkami lub

fotonami, to zmniejszenie strumienia cząstek lub fotonów po przejściu przez tarczę jest określone przez

$$-\frac{d\phi}{dx} = \sigma N \phi$$

gdzie ϕ jest strumieniem pocisków, N jest liczbą atomów tarczy na jednostkę objętości, a σ , przekrój czynny, jest stałą proporcjonalności. Z tej zależności wynika, że jednostkami przekroju czynnego są m^2 lub barn - jednostka powierzchni używana specjalnie do wyrażania przekrojów czynnych reakcji jądrowych, równa $10^{-28} m^2$.

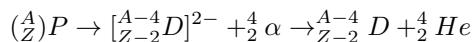
Przekroje czynne są zdefiniowane dla wszystkich typów reakcji wynikających z oddziaływania padającej cząstki i jądra. W przypadku reakcji absorpcji, wzbudzenie jądra złożonego może wiązać się z emisją cząstki alfa, elektronów, pozytonów, protonów, promieniowania γ lub, w przypadku rozszczepienia, fragmentów rozszczepienia. Wszystkie różne „ścieżki” rozpadu można określić za pomocą określonego przekroju czynnego, takiego jak σ_α dla emisji α , σ_β dla emisji β , itp [3].

Aby przedstawić konkretną reakcję, używany jest skrócony symbol padających i emitowanych cząstek w nawiasie, np. alfa-proton (α, p), alfa-neutron (α, n), neutron-proton (n, p), gamma-proton (γ, p), proton-gamma (p, γ), neutron-rozszczepienie (n, f) itp. Odpowiednie przekroje czynne są opisane w ten sam sposób, np. przekrój czynny na rozszczepienie indukowane neutronami jest zapisywany $\sigma(n, f)$.

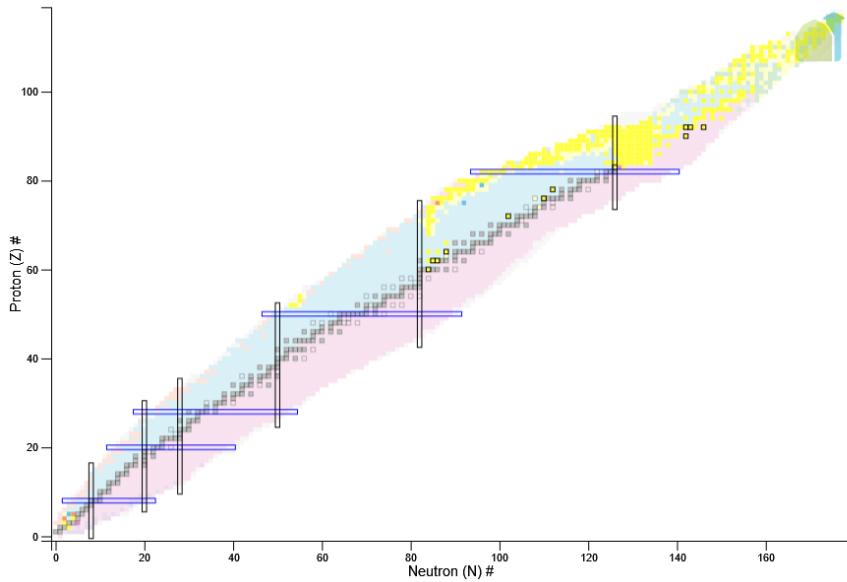
1.3.1 Rozpad alfa

Rozpad promieniotwórczy jest procesem jądrowym i jest w dużej mierze niezależny od stanu chemicznego i fizycznego nuklidu. Rzeczywisty proces rozpadu promieniotwórczego zależy od stosunku neutronów do protonów oraz od stosunku masy do energii cząstek macierzystych, pochodnych i emitowanych. Podobnie jak w przypadku każdej reakcji jądrowej, muszą obowiązywać różne prawa zachowania.

W rozpadzie alfa jądro macierzyste ${}_Z^A P$ emuluje cząstkę alfa ${}_2^4 \alpha$, w wyniku czego powstaje nuklid pochodny ${}_Z^{A-4} D$. Natychmiast po emisji cząstki alfa atom potomny nadal posiada elektrony Z atomu macierzystego - stąd atom potomny ma o dwa elektrony za dużo i powinien być oznaczony jako ${}_Z^{A-4} D^{2-}$. Te dodatkowe elektrony są tracone wkrótce po emisji cząstki alfa, pozostawiając atom potomny neutralnym elektrycznie. Ponadto cząstka alfa spowalnia i traci swoją energię kinetyczną. Przy niskich energiach cząstka alfa pozyskuje dwa elektrony i staje się neutralnym atomem helu. Proces rozpadu alfa jest opisany przez:



Proces rozpadu alfa występuje głównie w nuklidach bogatych w protony, o wysokiej liczbie atomowej, ze względu na fakt, że elektrostatyczne siły odpychające rosną szybciej w ciężkich nuklidach. Ponadto emitowana cząstka musi mieć wystarczającą energię, aby pokonać barierę potencjału w jądrze. Niemniej jednak cząstki alfa mogą pokonać tę barierę w procesie tunelowania kwantowego [3].

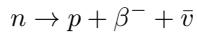


Rysunek 1.1: Emitery α (żółte), rysunek pochodzący z www.nndc.bnl.gov.

1.3.2 Rozpad beta

Radioaktywność β występuje podczas spontanicznego procesu przemiany jądra w wyniku emisji elektronu lub pozitonu lub wychwytu elektronu.

Emisja elektronu występuje, gdy nuklid ma nadmiar neutronów. Cząstka beta powstaje w wyniku przemiany jądrowej neutronu w proton w wyniku reakcji:

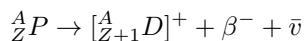


gdzie $\bar{\nu}$ jest antyneutrinem. Wyrzucony z jądra elektron o wysokiej energii oznaczany jest jako β^- , aby odróżnić go od innych elektronów oznaczanych jako e^- .

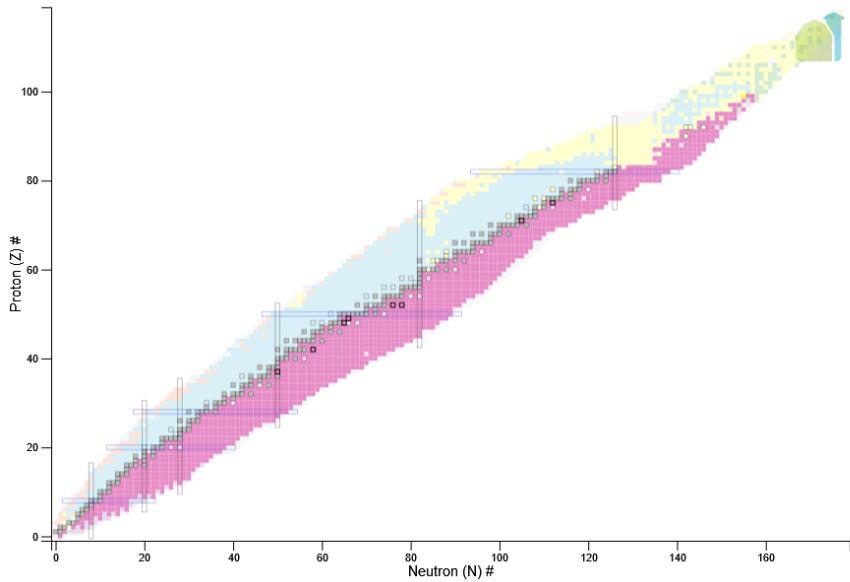
Emisja beta różni się od emisji alfa tym, że cząstki beta mają ciągłe spektrum energii od zera do pewnej maksymalnej wartości, energii końcowej, charakterystycznej dla danego nuklidu. Fakt, że cząstki beta nie są monoenergetyczne, ale mają ciągły rozkład energii aż do określonej energii maksymalnej, oznacza, że bierze w nich udział inna cząstka, tj. neutrino ν .

Ta energia punktu końcowego odpowiada różnicy mas między jądem macierzystym a jądem potomnym, zgodnie z zasadą zachowania energii. Średnia energia cząstki beta wynosi około $1/3$ energii maksymalnej. Dokładniej, „neutrino” emitowane w rozpadzie β^- jest antyneutrinem (neutrino jest emitowane w rozpadzie β^+). Neutrino ma zerowy ładunek i prawie zerową masę. Maksymalne energie cząstek beta wahają się od 10 keV do 4 MeV.

Proces rozpadu β^- można opisać następująco:



Natychmiast po rozpadzie przez emisję beta, atom potomny ma taką samą liczbę elektronów orbitalnych jak atom macierzysty, a zatem jest naładowany dodatnio. Bardzo szybko jednak atom potomny pozyskuje elektron z otaczającego go ośrodka i staje się elektrycznie neutralny.



Rysunek 1.2: Emityry β^- (różowy) , rysunek pochodzący z www.nndc.bnl.gov.

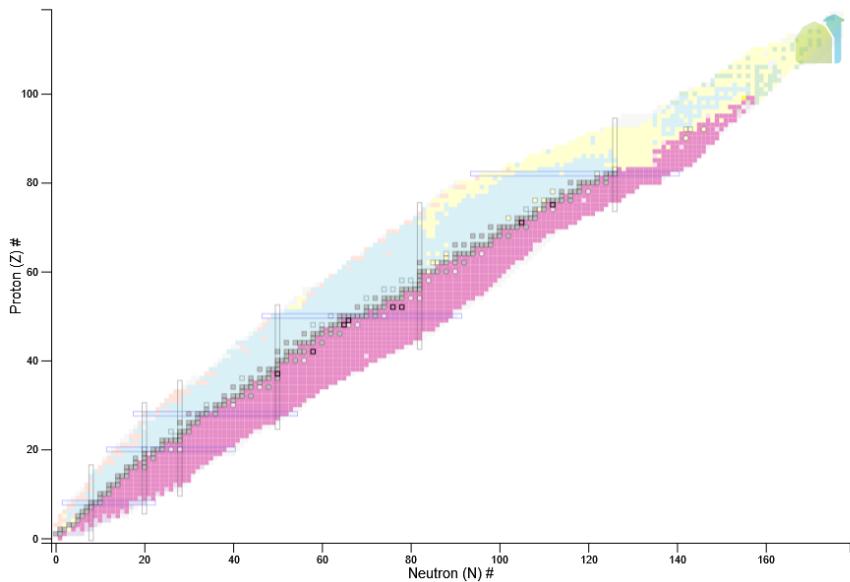
W nuklidach, w których stosunek neutronów do protonów jest niski, a emisja alfa nie jest energetycznie możliwa, jądro może emitować pozyton (antycząstka elektronu, o ładunku dodatnim). Wewnątrz jądra proton jest przekształcany w neutron, pozyton i neutrino:

$$p \rightarrow n + \beta^+ + v$$

Podobnie jak β^- , pozyton β^+ ma ciągłą dystrybucję energii aż do charakterystycznej energii maksymalnej. Pozyton po wyemitowaniu z jądra ulega silnemu przyciąganiu elektrostatycznemu z elektronami atomowymi. Pozyton i jeden z elektronów anihilują, w wyniku czego powstają dwa fotony (kwanty gamma) o energii 0,511 MeV każdy, poruszające się w przeciwnych kierunkach [2] [3].

Proces rozpadu β^+ można opisać następująco:

$$_Z^A P \rightarrow [_{Z-1}^A D]^- + \beta^+ + v$$



Rysunek 1.3: Emity β^+ (niebieski), rysunek pochodzący z www.nndc.bnl.gov.

1.3.3 Wychwyt elektronu

Nuklid z niedoborem neutronów mogą również osiągnąć stabilność poprzez wychwytywanie elektronów z wewnętrznych powłok K lub L orbit atomowych. W rezultacie proton w jądrze przekształca się w neutron:

$$p + e^- \rightarrow n + v$$

Proces ten jest podobny do rozpadu β^+ , ponieważ ładunek jądra zmniejsza się o 1. Proces rozpadu można opisać następująco:

$${}^A_zP \rightarrow {}^A_{z-1} D^* + v$$

a jądro potomne jest zwykle produkowane w stanie wzbudzonym. Powstałe jądro jest niestabilne i rozpada się poprzez wyrzucenie neutrina (v) i emisję charakterystycznego promieniowania rentgenowskiego, gdy wakat elektronowy w powłoce K lub L jest wypełniany elektronami z wyższych powłok [3].

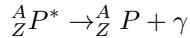
1.3.4 Rozpad gamma

Emisja promieniowania gamma nie jest podstawowym procesem rozpadu, ale zwykle towarzyszy rozpadom alfa, beta oraz wychwytem elektronu, gdy produkt pochodny powstaje w stanie wzbudzonym. Zazwyczaj ten stan wzbudzony powraca bardzo szybko ($< 10^{-9}$ s) do stanu podstawowego poprzez emisję fotonu gamma.

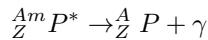
W przeciwnieństwie do szybkiej emisji gamma, która zachodzi często poprzez promieniowanie dipolowe, emisja ze stanu o wydłużonym czasie życia zachodzi poprzez emisję kwantów gamma o wyższym rzędzie multipola. Jeśli czas życia stanu wzbudzonego przekracza około jednej nanosekundy, wzbudzone jądro definiuje się jako znajdujące się w stanie metastabilnym lub

izomerycznym. Proces rozpadu z tego stanu wzbudzonego jest znany jako przejście izomeryczne (IT – ang. „isomeric transition”).

Proces rozpadu gamma można opisać przez:

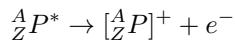


A przejścia izomerycznego:



gdzie gwiazdka * oznacza stan wzbudzony, a Am stan izomeryczny lub metastabilny.

Alternatywnie do emisji gamma, wzbudzone jądro może powrócić do stanu podstawowego poprzez wyrzucenie elektronu z powłoki wewnętrznej. Jest to zjawisko znane jako konwersja wewnętrzna i skutkuje emisją elektronu i promieniowania rentgenowskiego z powodu kaskadowego przejścia elektronów na niższe poziomy energetyczne. Stosunek elektronów konwersji wewnętrznej do fotonów emisji gamma jest znany jako współczynnik konwersji wewnętrznej [2] [3]. Elektrony konwersji są monoenergetyczne. Proces konwersji wewnętrznej można opisać następująco:



1.3.5 Emisja cząstek ze stanów wzbudzonych

Emisja lekkich cząstek, takich jak protony lub neutrony, ze stanów wzbudzonych jąder atomowych, jest procesem, który zachodzi, gdy jądro atomowe, posiadające nadmiar energii, przechodzi do niższego stanu energetycznego. Jądra atomowe mogą znajdować się w różnych stanach energetycznych. Stan podstawowy to najniższy poziom energetyczny jądra, natomiast stany wzbudzone to poziomy energetyczne wyższe od stanu podstawowego. Stany wzbudzone powstają na przykład w podczas reakcji jądrowych lub rozpadów promieniotwórczych. Gdy jądro atomowe znajduje się w stanie wzbudzonym, może powrócić do stanu podstawowego poprzez emisję cząstek lub fotonów (gamma). Proces emisji lekkich cząstek, takich jak protony lub neutrony, zachodzi, gdy energia stanu wzbudzonego przekracza odpowiednią energię separacji protonu (Sp) lub neutronu (Sn).

Energia separacji protonu odnosi się do minimalnej energii, która musi być dostarczona do jądra atomowego, aby wyrzucić proton spoza jądra. Podobnie, energia separacji neutronu jest minimalną energią niezbędną do wystrzelenia neutronu z jądra atomowego. Matematycznie można to wyrazić jako:

$$Sp = E(A, Z) - E(A - 1, Z - 1)$$

$$Sn = E(A, Z) - E(A - 1, Z)$$

gdzie $E(A, Z)$ oznacza energię jądra atomowego o liczbie masowej A i liczbie atomowej Z .

Energie separacji różnią się w zależności od izotopu. Na przykład, dla izotopu kobaltu (^{60}Co), który ulega rozpadowi beta, energia separacji neutronu wynosi około 7,49 MeV, co jest charakterystyczne dla tego konkretnego jądra.

1.4 Model powłokowy jądra atomowego

Model powłokowy atomu jest pomocny do wyjaśnienia bardziej złożonych struktur, takich jak jądra atomowe. ponieważ podstawowe zasady, które rządzą rozmieszczeniem i zachowaniem elektronów w atomie mogą zostać, poprzez analogię, wykorzystane do opisu zachowania nukleonów w jądrze. Główna różnica polega na tym, że w atomie mamy do czynienia z dobrze zdefiniowanym potencjałem Coulomba, podczas gdy w jądrze nukleony poruszają się w potencjale, który same tworzą.

W modelu powłokowym jądra atomowego wprowadzamy uśredniony potencjał oddziaływań doznawany przez nukleon wynikający z oddziaływania z pozostałymi nukleonami. Schemat poziomów energetycznych nukleonów w tym potencjale zawiera szereg stanów zdegenerowanych o tej samej parzystości, a różniące się wartościami orbitalną liczbą kwantową l . Poszczególne stany opisane są przez radialną liczbę kwantową, określającą liczbę węzłów radialnej funkcji falowej i przyjmującą wartości 1,2,3,... i l o dopuszczalnych wartościach 0,1,2,3,..., którym przyporządkujemy oznaczenia literowe s, p, d, f, g. Poziomy, szczególnie najniższe, mające te same wartości głównej liczby kwantowej n leżą blisko siebie. W wyniku tego grupowania poziomów wystąpią między niektórymi z nich, a mianowicie na poziomach 1s, 1p, 2s, 2p, większe przerwy.

Porównując niektóre zmierzone właściwości układów atomowych z przewidywaniami modelu, znajdujemy niezwykłą zgodność. W szczególności widzimy regularne i płynne zmiany właściwości atomowych w obrębie podpowłoki, ale raczej nagłe i dramatyczne zmiany właściwości, gdy wypełniamy jedną podpowłokę i wchodzimy do następnej. Kiedy próbujemy przenieść ten model do sfery jądrowej, natychmiast napotykamy kilka zastrzeżeń. W przypadku atomu potencjał jest dostarczany przez pole Coulomba jądra, więc podpowłoki elektronowe („orbity”) są ustalane przez czynnik zewnętrzny. W jądrze nie ma takiego zewnętrznego czynnika; nukleony poruszają się w potencjale, który same tworzą.

Innym aspektem teorii powłok atomowych jest istnienie orbit przestrzennych. Często bardzo przydatne jest opisywanie właściwości atomów w kategoriach orbit przestrzennych elektronów. Elektrony mogą poruszać się po tych orbitach względnie bez kolizji z innymi elektronami. Nukleony mają stosunkowo dużą średnicę w porównaniu z rozmiarem jądra. Kwestia istnienia potencjału jądrowego jest rozwiązywana przez podstawowe założenie modelu powłokowego: ruch pojedynczego nukleonu jest regulowany przez potencjał wywołany przez wszystkie inne nukleony. Jeśli potraktujemy w ten sposób każdy pojedynczy nukleon, możemy pozwolić nukleonom z kolei zajmować poziomy energetyczne szeregu podpowłok.

Istnienie określonych orbit przestrzennych zależy od zasady Pauliego. Rozważmy w ciężkim jądrze zderzenie dwóch nukleonów w stanie bliskim dna studni potencjału. Gdy nukleony zderzą się, przekażą sobie energię, ale jeśli wszystkie poziomy energetyczne są wypełnione aż do poziomu nukleonów walencyjnych, nie ma możliwości, aby jeden z nukleonów zyskał energię, z wyjątkiem przejścia na poziom walencyjny. Pozostałe poziomy w pobliżu pierwotnego poziomu są zapełnione i nie mogą przyjąć dodatkowego nukleonu. Taki transfer, z nisko położonego poziomu do pasma walencyjnego, wymaga więcej energii niż nukleony mogą przekazać w zderzeniu. Dlatego zderzenia nie mogą wystąpić, a nukleony mogą rzeczywiście orbitować tak, jakby były „przezroczyste” dla siebie nawzajem.

W fizyce atomowej oddziaływanie spin-orbita, które powoduje obserwowaną strukturę linii widmowych, powstaje w wyniku elektromagnetycznego oddziaływania momentu magnetycznego

elektronu z polem magnetycznym generowanym przez jego ruch wokół jądra. Efekty są zazwyczaj bardzo małe, żadne takie oddziaływanie elektromagnetyczne nie byłoby na tyle silne, by spowodować znaczące zmiany w odstępach między poziomami atomowymi, potrzebne do wygenerowania obserwowanych liczb magicznych. Te tak zwane „magiczne liczby” ($N = 2, 8, 20, 28, 50, 82$ i 126) reprezentują efekty wypełnionych głównych powłok. Niemniej jednak przyjmujemy koncepcję jądrowej siły spin-orbita o takiej samej formie jak atomowa siła spin-orbita, ale z pewnością nie pochodzenia elektromagnetycznego [2].

Oddziaływanie spin-orbita jest zapisywane jako $V_{so}(\mathbf{r})\mathbf{l}\circ\mathbf{s}$, gdzie $V_{so}(\mathbf{r})$ to czynnik nie wpływający na strukturę poziomów jądrowych. Istotnym czynnikiem, powodującym zmianę kolejności poziomów jest $\mathbf{l}\circ\mathbf{s}$, gdzie \mathbf{l} to orbitalny moment pędu, a \mathbf{s} to wektor spinu nukleonu. Podobnie jak w fizyce atomowej, w obecności oddziaływania spin-orbita właściwe jest oznaczanie stanów całkowitym momentem pędu $\mathbf{j} = \mathbf{l} + \mathbf{s}$. Pojedynczy nukleon ma $\mathbf{s} = 1/2$, więc możliwe wartości całkowitej liczby kwantowej momentu pędu to $\mathbf{j} = \mathbf{l} + 1/2$ lub $\mathbf{j} = \mathbf{l} - 1/2$ (z wyjątkiem $\mathbf{l} = 0$, w którym to przypadku dozwolone jest tylko $\mathbf{j} = 1/2$). Wartość oczekiwana $\mathbf{l}\circ\mathbf{s}$ można obliczyć:

$$\begin{aligned}\mathbf{j}^2 &= (\mathbf{l} + \mathbf{s})^2 \\ \mathbf{j}^2 &= \mathbf{l}^2 + 2\mathbf{l}\circ\mathbf{s} + \mathbf{s}^2 \\ \mathbf{l}\circ\mathbf{s} &= \frac{1}{2}(\mathbf{j}^2 - \mathbf{l}^2 - \mathbf{s}^2)\end{aligned}$$

Podstawiając jako wartość oczekiwana otrzymujemy [2]:

$$\langle \mathbf{l}\circ\mathbf{s} \rangle = \frac{1}{2}[j(j+1) - l(l+1) - s(s+1)]\hbar^2$$

Rozważmy poziom taki jak poziom $1f$ ($\mathbf{l} = 3$), który ma degenerację $2(2\mathbf{l}+1) = 14$. Możliwe wartości \mathbf{j} to $\mathbf{l} \pm 1/2 = 5/2, 3/2, 1/2$. Mamy więc poziomy $1f_{5/2}$ i $1f_{7/2}$. Degeneracja każdego poziomu wynosi $(2\mathbf{j}+1)$, co wynika z wartości m_j . Pojemność poziomu $1f_{5/2}$ wynosi zatem 6, a $1f_{7/2}$ wynosi 8, co daje ponownie 14 stanów - liczba możliwych stanów musi zostać zachowana, grupujemy je tylko inaczej. Dla stanów $1f_{5/2}$ i $1f_{7/2}$, które są znane jako para spin-orbita lub dublet, istnieje separacja energii, która jest proporcjonalna do wartości $\langle \mathbf{l}\circ\mathbf{s} \rangle$ dla każdego stanu. Rzeczywiście, dla dowolnej pary stanów z $\mathbf{l} > 0$, możemy obliczyć różnicę energii [2]:

$$\langle \mathbf{l}\circ\mathbf{s} \rangle_{j=l+1/2} - \langle \mathbf{l}\circ\mathbf{s} \rangle_{j=l-1/2} = \frac{1}{2}(2\mathbf{l}+1)\hbar^2$$

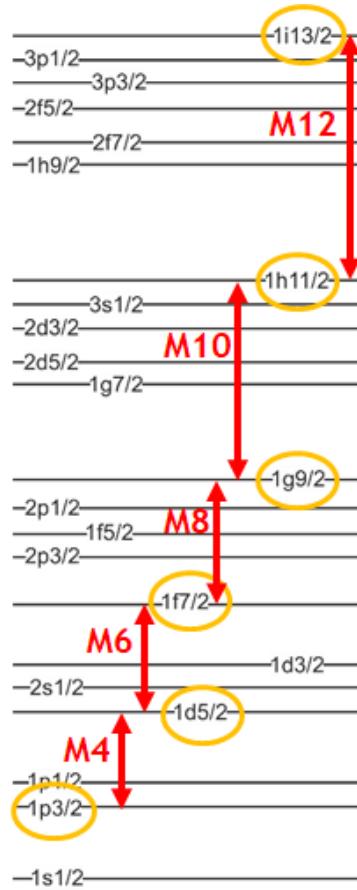
Podział energii wzrasta wraz ze wzrostem \mathbf{l} . Rozważmy efekt wybrania $V_{so}(\mathbf{r})$ jako ujemnego, tak aby członek pary z większym j został zepchnięty w dół. Poziom $1f_{7/2}$ pojawia się teraz w szczelinie między drugą i trzecią powłoką; jego pojemność 8 nukleonów daje magiczną liczbę 28. Dla porównania, rozszczepienia p i d nie skutkują żadnym większym przegrupowaniem poziomów. Kolejnym istotnym efektem oddziaływania spin-orbita jest poziom $1g$. Stan $1g_{9/2}$ zostaje zepchnięty aż do następnej niższej głównej powłoki; jego pojemność 10 nukleonów dodaje się do poprzedniej sumy 40 dla tej powłoki, dając magiczną liczbę 50. Podobny efekt występuje na szczytzie każdej głównej powłoki. W każdym przypadku członek pary spin-orbita o niższej energii z następnej powłoki jest spychany w dół do niższej powłoki.

Model powłokowy, pomimo prostoty swojej koncepcji, jest skuteczny w przewidywaniu spinów i parzystości prawie wszystkich nieparzystych stanów podstawowych i jest nieco mniej

skuteczny (ale nadal zadowalający) w przewidywaniu magnetycznych momentów dipolowych i elektrycznych momentów kwadrupolowych [2].

1.5 Rozciągnięte stany w lekkich jądrach

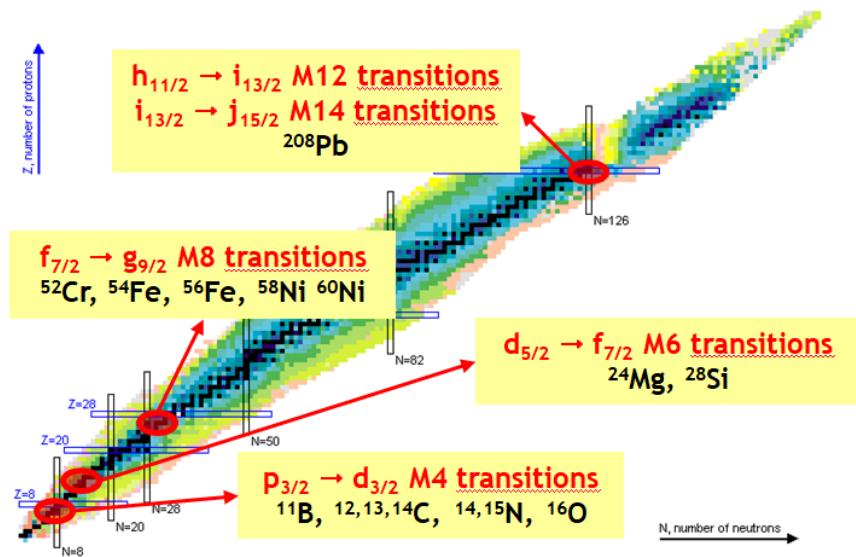
Stany „stretched” powstają w wyniku przeniesienia nukleonu (protonu lub neutronu) poprzez szczelinę energetyczną z niższej powłoki energetycznej na wyższą. W wyniku takiego wybicia na wyższym poziomie pojawia się cząstka, natomiast na niższym powstaje tzw. dziura. Cechą charakterystyczną poziomów „stretched” jest to, że zarówno cząstka jak i dziura zajmują orbitale o najwyższym momencie pędu dostępnym na swoich powłokach, a ponadto posiadają maksymalny spin wynikający z obsadzanego orbitalu. Ze względu na to, że inne jedno-cząstkowe konfiguracje o tych samych liczbach kwantowych są dostępne przy znacznie wyższych energiach, funkcja falowa stanu „stretched” jest zdominowana przez pojedynczą składową cząstka-dziura. Ta cecha sprawia, że stany tego typu mają względnie prostą strukturę, a w związku z tym mogą dostarczyć przejrzystych informacji o szczegółach działania sił jądrowych [4].



Rysunek 1.4: Stany rozciągnięte.

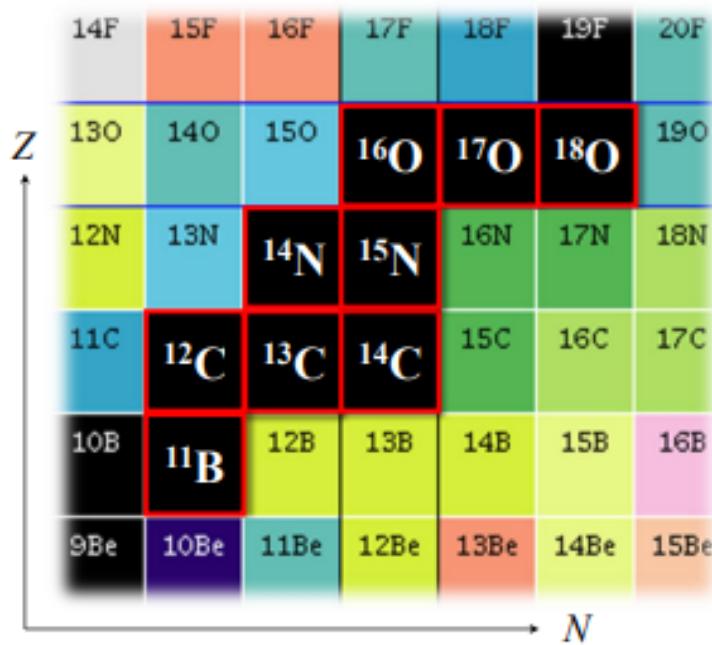
Przykładem stanów typu „stretched” w lekkich jądrach, takich jak B, C, N, O , są rezonanse M4, wynikające z przeniesienia nukleonu z orbitalu $p_{3/2}$ na orbital $d_{5/2}$ oraz maksymalnego sprzężenia spinów cząstki i dziury do spinu o wartości 4. W cięższych jądrach tego typu wzbudzenia

wynikają z transferu nukleonu pomiędzy odpowiednio wyższymi powłokami energetycznymi i przybierają postać następujących rezonansów: M6 występujących np. w jądrach ^{24}Mg , ^{28}Si , M8 zaobserwowanych w izotopach ^{52}Cr , ^{54}Fe , ^{56}Fe , ^{58}Ni , ^{60}Ni oraz M12 i M14 zidentyfikowanych w nuklidzie ^{208}Pb [4] [5].



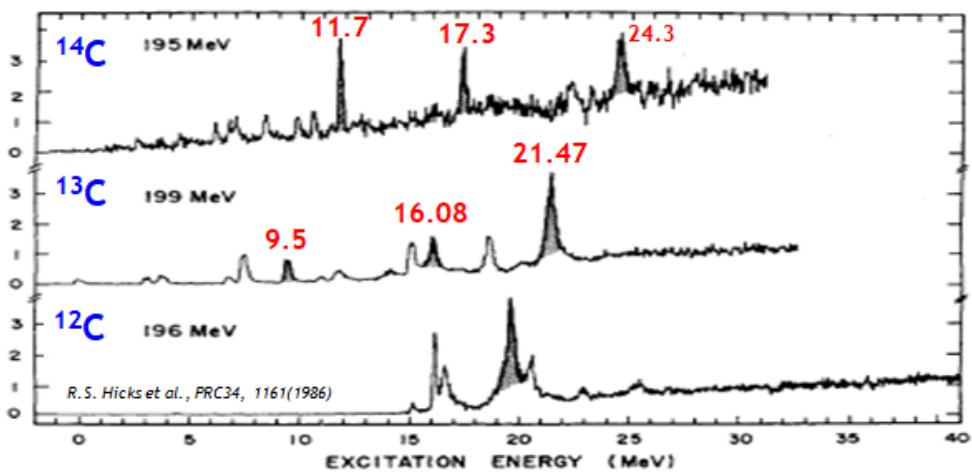
Rysunek 1.5: Jądra, w których zaobserwowane stany rozciągnięte.

Stany „stretched” w lekkich jądrach są zlokalizowane powyżej energii separacji cząstek, w obszarze tzw. continuum energetycznego, gdzie gęstość poziomów energetycznych rośnie. Pomimo wysokiej energii wzbudzenia rezonanse M4 są jednymi z najprostszych wzbudzeń jądrowych, co sprawia, że badania ich właściwości powinny dostarczyć cennych informacji do porównań z wynikami najnowszych obliczeń modeli teoretycznych. W szczególności brakuje informacji na temat ścieżek rozpadu tych wzbudzeń jądrowych.



Rysunek 1.6: Fragment tablicy nuklidów z okolic lekkich jąder atomowych. Kolorem czerwonym oznaczono izotopy, w których występują stany rozciągnięte wynikające z przejęć $p_{3/2} \rightarrow d_{3/2}$ [5].

Wśród izotopów węgla najbardziej dogodnym z punktu widzenia dostępności eksperymentalnej jest stabilny ^{13}C , w którym występują trzy silne, dobrze wyizolowane spośród innych poziomów rezonans M4. Z kolei rezonans M4 zidentyfikowany dotychczas w ^{12}C charakteryzuje się bardziej złożoną strukturą, natomiast izotop ^{14}C , choć bardzo obiecujący (posiada trzy dobrze izolowane i raczej czyste rezonanse M4), jest radioaktywny, więc zastosowanie go jako tarczy w eksperymencie wymagałoby wdrożenia specjalnych procedur ochrony radiologicznej [4].



Rysunek 1.7: Widma elektronów rozproszonych na ^{12}C , ^{13}C i ^{14}C [6].

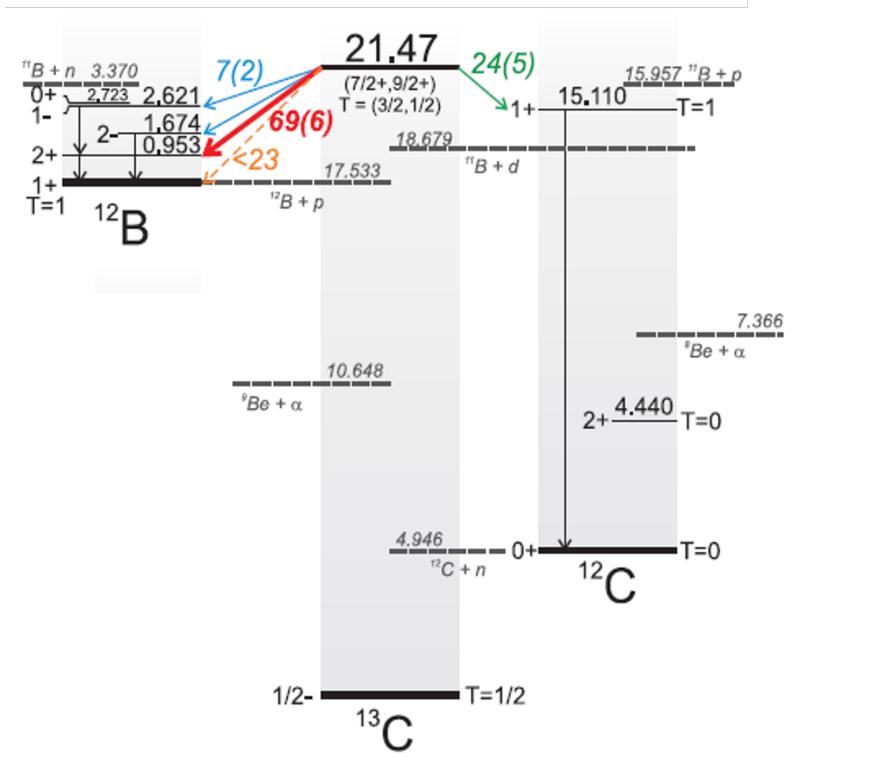
Rozdział 2

Eksperyment i układ pomiarowy

Eksperyment został przeprowadzony w Centrum Cyklotronowym Bronowice przy Instytucie Fizyki Jądrowej PAN w Krakowie. Pomiarów dokonano w dwóch etapach (marcu-czerwcu 2019 r. i czerwcu 2020 r.), a celem było zbadanie rozpadu stanu rozciagniętego M4 zlokalizowanego przy energii 21,47 MeV w jądrze ^{13}C . Stan ten jest jednym z trzech rezonansów M4 w tym izotopie przy energiach 9,50, 16,08 i 21,47 MeV zlokalizowanych w przeszłości w eksperymentach z użyciem reakcji rozpraszania protonów, elektronów i pionów [6] [7] [8]. Wybór stanu 21,47 MeV jako celu badań został podyktowany z jednej strony jego położeniem powyżej energii separacji protonu oraz neutronu, a z drugiej, jego intensywną produkcją i odizolowaniem od innych stanów w widmie energii wzbudzenia ^{13}C .

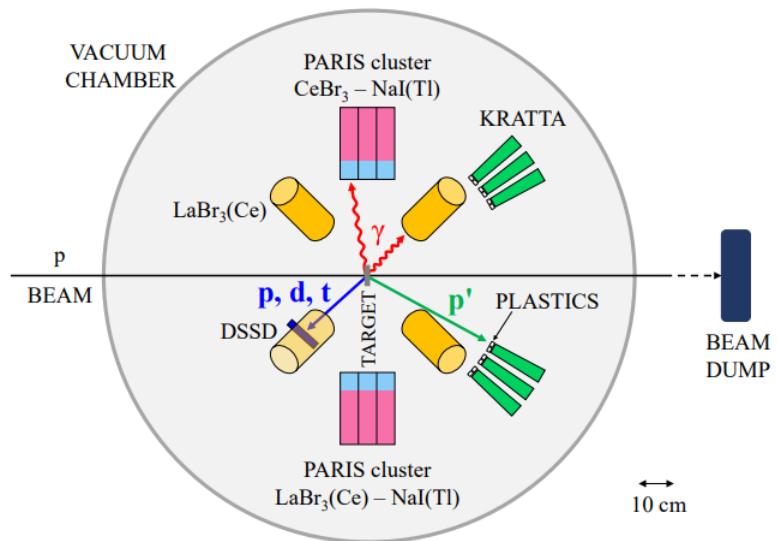
W celu produkcji stanu przy energii 21,47 MeV w ^{13}C zastosowano reakcję nieelastycznego rozproszenia wiązki protonów o energii 135 MeV. Rozkład kątowy nieelastycznie rozproszonych protonów wzbudzających ten rezonans wykaazywał maksimum pod kątem $\sim 30^\circ$ w układzie odniesienia środka masy [6]. W pierwszej części eksperymentu, rezonans został wzbudzony przy użyciu wiązki protonów uderzającej w grubą tarczę z czystego ^{13}C (197 mg/cm^2). Rozproszone protony były wykrywane przez sześć detektorów CsI KRATTA umieszczonych pod kątem $\sim 36^\circ$ względem osi wiązki, natomiast promieniowanie gamma było rejestrowane przez układ czterech kryształów LaBr₃ oraz dwóch klastrów detektorów typu „phoswich” (system PARIS [9]). Rozproszone protony oraz kwanty gamma były mierzone w tzw. koincydencji czasowej, to znaczy jednoczesnego rejestrowania tych zdarzeń w ramach wąskiego przedziału czasowego, co pozwala na wiarygodne połączenie informacji o rozproszonym protonie z odpowiednim kwantem gamma. W drugiej części eksperymentu, przetestowano dodatkową możliwość zbadania rozpadu rezonansów M4 przez rejestrację koincydencji rozproszony proton – lekka cząstka naładowana z rozpadu rezonansu (proton, deuter, tryton). Zastosowano tę samą wiązkę protonów, ale ze specjalnie zaprojektowaną cienką tarczą (1 mg/cm^2) i ulepszoną konfiguracją, z 30 modułami KRATTA i dodatkowym grubym, czułym na położenie detektorem Double-sided Silicon Strip Detector (DSSSD), umieszczonym do tyłu pod kątem 138° . Eksperyment zakończył się sukcesem, zaobserwowano wyraźny pik związany z populacją rezonansu M4 w ^{13}C przy 21,5 MeV w zrekonstruowanym widmie energii wzbudzenia oraz po raz pierwszy uzyskano wyraźną identyfikację rozpadu gamma stanów wzbudzonych w jądrach potomnych, w koincydencji z populacją rezonansu M4. W szczególności zaobserwowane rozпадy gamma w jądrach potomnych wskazują na kanały rozpadu rezonansu M4 poprzez emisję protonów i neutronów. W kanale rozpadu pro-

tonowego zaobserwowano silną gałąź prowadzącą do pierwszego wzbudzonego stanu 0,953 MeV w ^{12}B natomiast kanał rozpadu neutronowego prowadził do stanu 15,11 MeV w ^{12}C [5].

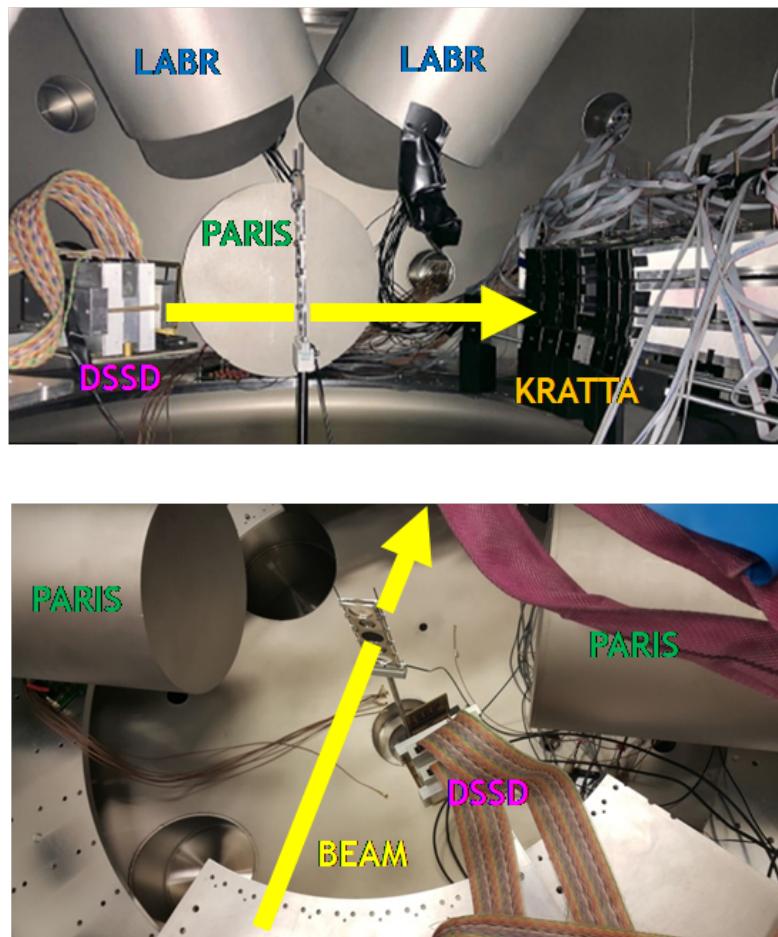


Rysunek 2.1: Eksperymentalny schemat rozpadu rezonansu rozciągłego 21,47 MeV w ^{13}C związany z detekcją promieniowania γ z jąder potomnych [4].

Co więcej, analiza drugiej części eksperymentu umożliwiła oszacowanie rozgałęzienia rozpadu prowadzącego do stanu podstawowego w ^{12}B , którego nie można było zaobserwować w koincydencjach rozproszony proton-gamma, ale co było możliwe przy użyciu koincydencji rozproszony proton-cząstka z rozpadu rezonansu.

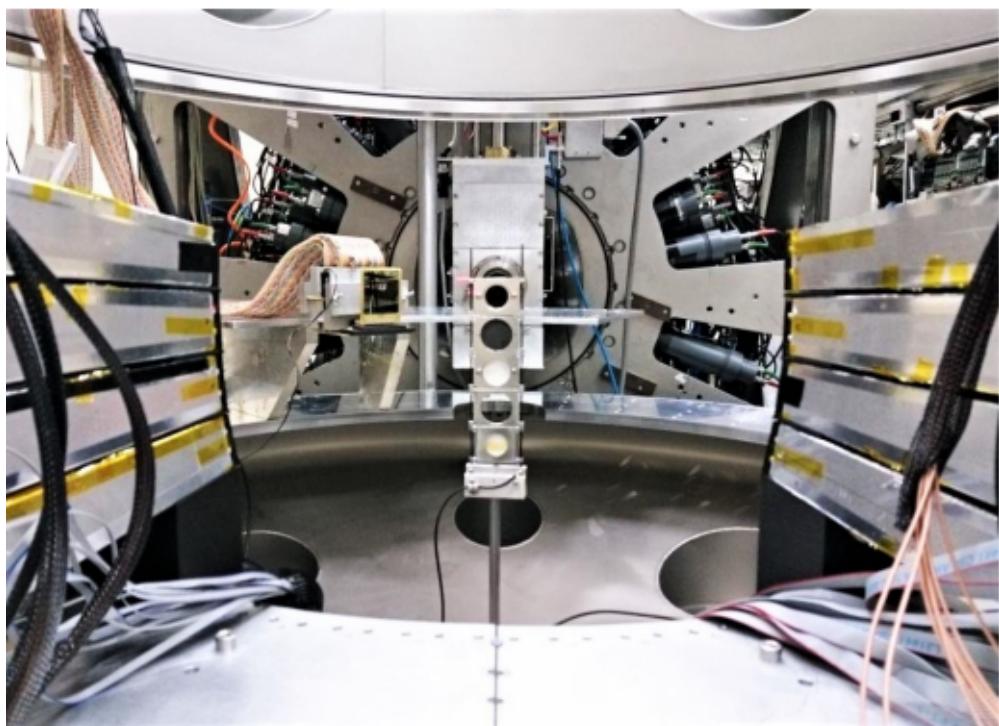


Rysunek 2.2: Widok z góry zestawu eksperymentalnego [5].



Rysunek 2.3: Zobrazowanie jak wiązka przechodzi przez próbki względem umiejscowionych detektorów.

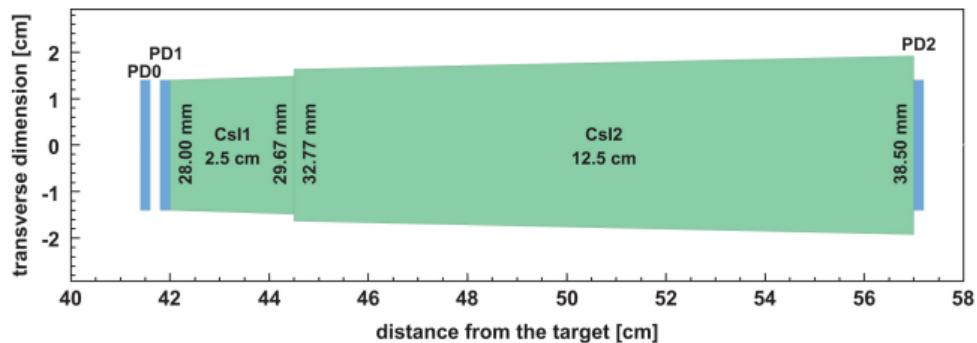
Układ KRATTA, umieszczony pod kątem $30^\circ - 42^\circ$, był sprzężony z czterema detektorami LaBr_3 ($3,5'' \times 8,0''$), dwoma klastrami PARIS umieszczonymi pod kątem 90° i detektorem DSSSD umieszczonym od tyłu pod kątem 138° . Z dwóch klastrów PARIS, jeden miał detektory photo-swich typu $\text{CeBr}_3\text{-NaI(Tl)}$, podczas gdy drugi miał detektory $\text{LaBr}_3(\text{Ce})\text{-NaI(Tl)}$. Dodatkowo, przed każdym modułem KRATTA zostały umieszczone cztery detektory plastikowe, aby uzyskać dokładniejszą informację o czasie detekcji rozproszonych protonów. W pierwszej części eksperymentu używanych było 6 modułów KRATTA (w dwóch kolumnach po trzy detektory, symetrycznie do osi wiązki), natomiast w drugiej części dodano dodatkowe kolumny, co dało łącznie 30 modułów. Moduły KRATTA i detektor DSSSD znajdowały się wewnętrz dużej komory próżniowej o średnicy $1,5\text{ m}$, podczas gdy układ PARIS i scyntylatory $\text{LaBr}_3(\text{Ce})$ zostały umieszczone w specjalnie zaprojektowanych niszach w ścianach i pokrywie komory próżniowej, co pozwoliło na zbliżenie detektorów do tarczy bez umieszczania ich w próżni. Przed PARIS i $\text{LaBr}_3(\text{Ce})$ ściana komory ze stali nierdzewnej miała grubość 2 mm [5].



Rysunek 2.4: Zdjęcie układu eksperimentalnego [5].

Do detekcji nieelastycznie rozproszonych protonów wiązki wykorzystano detektor KRATTA. Detektory scyntylacyjne, zarówno PARIS, jak i $\text{LaBr}_3(\text{Ce})$, mierzyły promieniowanie gamma emitowane ze wzbudzonych jąder w tarczy, podczas gdy DSSSD wykrywał emitowane lekkie cząstki naładowane z rozpadu jąder w materiale tarczy (protoны, deuteronы, trytony).

Kraków Triple Telescope Array (KRATTA) to system detekcji przeznaczony do pomiaru energii, kąta emisji i składu izotopowego lekkich naładowanych produktów reakcji. Układ składa się z 38 niezależnych modułów, które mogą być rozmieszczone w zależności od potrzeb eksperymentalnych.



Rysunek 2.5: Schemat pojedynczego modułu KRATTA [5].

Każdy moduł KRATTA, składa się z trzech identycznych fotodiod krzemowych Hamamatsu PIN o grubości 500 μm i dużej powierzchni ($28 \text{ mm} \times 28 \text{ mm}$) oraz dwóch kryształów CsI(Tl) o długości odpowiednio 2,5 i 12,5 cm, w konfiguracji potrójnego teleskopu. Aktywny obszar pojedynczego teleskopu obejmuje około 4,5 msr kąta brylowego w optymalnej odległości 40 cm od tarczy. Pierwsza fotodioda (PD0) służy jako detektor Si różnic energii, podczas gdy druga i trzecia fotodioda (PD1 i PD2) odczytują sygnał scyntylacyjny odpowiednio cienkiego i grubego kryształu CsI(Tl), a dodatkowo zapewniają bezpośredni sygnał jonizacji. Konfiguracja eksperymentu obejmowała 6 (30) moduły w 2019 (2020) roku, rozmieszczone w dwóch podgrupach 3×1 (3×5) umieszczonych po obu stronach linii wiązki.

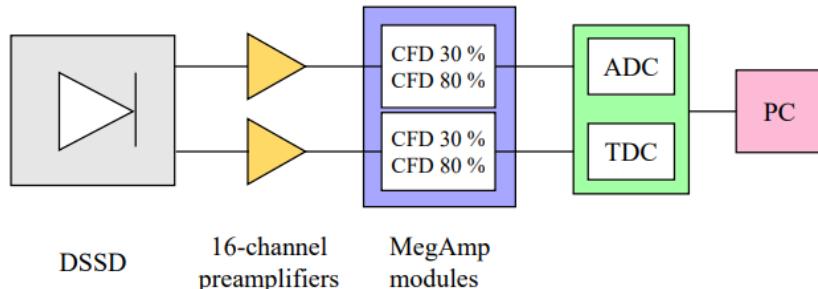
Dwustronny krzemowy detektor paskowy (DSSSD) w niniejszym eksperymencie ma obszar aktywny $49,50 \text{ mm} \times 49,50 \text{ mm}$ i grubość $1500 \mu\text{m}$.



Rysunek 2.6: Fotografia detektora DSSSD [5].

DSSSD jest podzielony na 16 pasków po stronie złącza i 16 ortogonalnych pasków po stronie

omowej, aby umożliwić zarówno poziomą, jak i pionową identyfikację położenia. Sygnały DSSSD są najpierw wstępnie wzmacniane, a następnie przechodzą przez moduły MegAmp (wzmacniacz z funkcją kształtuowania impulsu), zanim ich informacje o amplitudzie oraz czasie zostaną zdigitalizowane przez przetworniki ADC i TDC i ostatecznie przesłane na dysk komputera.



Rysunek 2.7: Schemat działania detektora DSSSD [5].

MegAmp został opracowany w INFN Milano w celu obsługi różnych sygnałów wejściowych. Może on przyjmować na wejściu standardowe sygnały z przedwzmacniaczy ładunkoczułych lub szybkie sygnały, jak w przypadku detektorów scyntylacyjnych odczytywanych przez lampy foto-powielacza. W przypadku akwizycji wielu kanałów, jak w przypadku detektatorów DSSSD, system sekwencyjnego odczytu MegAmp zapewnia zmniejszenie złożoności i kosztów systemu akwizycji. Każdy MegAmp składa się z pojedynczego modułu NIM z 16 kanałami wzmacniacza. Sekcja energetyczna składa się ze wzmacniacza spektroskopowego, podczas gdy sekcja czasowa pozwala na uzyskanie zarówno informacji o czasie, jak i kształcie impulsu. Dwa dyskryminatory stałofrakcyjne (CFD) dostarczają informacji o czasie, w którym sygnał osiąga 30% i 80% maksymalnej wartości. Różnica tych dwóch wartości jest więc proporcjonalna do czasu narastania sygnału wejściowego.

Rozdział 3

Działanie programów tworzenia struktury danych oraz kalibracji czasowej detektora DSSSD

3.1 Tworzenie struktury TTree ROOT

Większa część pracy była skoncentrowana na zoptymalizowaniu przedstawienia wyników pomiarów z poprzednich eksperymentów, pracowano nad utworzeniem tzw. „drzewa” ROOT z pozyskanych danych pomiarowych. Struktura drzewa pozwala np. na szybkie sprawdzanie korelacji pomiędzy rozproszonymi protonami, kwantami gamma i lekkimi cząstками naładowanymi. Dzięki utworzeniu gałęzi zawierających skalibrowane energie i czasy detekcji możliwe jest szybkie wyświetlanie widm lub dwuwymiarowych macierzy wielkości zarejestrowanych przez detektory oraz dodawanie różnych warunków. W ten sposób wstępnie posortowane dane pozwalały na lepsze monitorowanie informacji zbieranych podczas eksperymentu. W rozdziale 3 opisane zostanie działanie kodów na przykładach ich fragmentów, a całość kodu znajduje się w „Dodatku” na końcu pracy.



Rysunek 3.1: Struktura utworzonego w pracy drzewa ROOT.

ROOT [10] to obszerny framework opracowany przez Europejską Organizację Badań Jądrowych (CERN) dla celów analizy danych w fizyce wysokich energii. Jest to zestaw narzędzi programistycznych, które umożliwiają przechowywanie, przetwarzanie i analizowanie dużych ilości danych generowanych w eksperymentach fizycznych. ROOT jest napisany w C++ i jest w pełni zintegrowany z tym językiem, choć oferuje również wsparcie dla innych języków, takich jak Python (poprzez PyROOT). Posiada także własny interpreter języka C++, CINT (oraz nowszy Cling). Umożliwia efektywne zarządzanie dużymi wolumenami danych dzięki swojemu własnemu formatowi plików ROOT (.root), który jest zoptymalizowany pod kątem szybkiego dostępu i kompresji danych. Oferuje narzędzia do wizualizacji danych, w tym możliwość tworzenia zaawansowanych wykresów, histogramów, grafów 2D i 3D oraz interaktywnych aplikacji. Szeroko stosowany w dziedzinach, które wymagają analizy dużych zestawów danych, takich jak fizyka cząstek, astrofizyka, biologia oraz finanse. Jego elastyczność i wydajność sprawiają, że jest preferowanym narzędziem w wielu instytucjach badawczych na całym świecie.

Poniżej znajduje się kod napisany w języku C++, który za pomocą środowiska ROOT, tworzy drzewo ROOT. Program korzysta z plików wstępnie przekonwertowanych do formatu .root, a nie z tzw. „surowych” danych, które zbierane są w czasie eksperymentu. Kolejne gałęzie drzewa adresowane są tzw. liśćmi, jak pokazano na rysunku 3.1, aby utworzyć uporządkowaną strukturę.

Na początku deklarujemy potrzebne biblioteki oraz pliki nagłówkowe z framework'u ROOT, z których kod korzysta. Pliki te dostarczają funkcji do obsługi plików ROOT, drzew i różnych klas detektorów (Plastic, Kratta, Paris, LaBr, Silicon).

```
#include "TFile.h"
#include "TH2F.h"
...
#include "Kratta.h"
#include "Plastic.h"
#include "Paris.h"
#include "LaBr.h"
#include "Silicon.h"
```

Kod definiuje funkcję o nazwie `rewrite_tree`, która przyjmuje wskaźnik do drzewa TTree jako argument.

```
void rewrite_tree(TTree * tree)
```

Wskaźnik ten jest tworzony przy pomocy osobnego programu „Chain9_full.C”. Jest to krótki skrypt tworzący łańcuch danych do poukładania przez program. Zawiera ścieżkę do folderu z danymi i wytyczne, z jakich paczek danych chcemy skorzystać. Ułatwia to pracę z danymi, które są podzielone na wiele plików. Poniżej przedstawiony jest wydruk programu:

```
#include <iostream>
#include <fstream>

TChain * Chain_full(){

    static const char *filemask = "/data2/mbsdaq/e009/root/run%04d.root";

    int first = 6984;
    int last = 7838;

    std::ofstream outfile ("runs_from_chain.txt");
```

```

outfile << first << " " << last << std::endl;
outfile.close();

TChain * chain = new TChain("tree","tree");

/*for (int i=4383; i<=4594; i++)
    chain->Add(Form(filemask, i));

for (int i=4596; i<=4691; i++)
    chain->Add(Form(filemask, i));

for (int i=4718; i<=4724; i++)
    chain->Add(Form(filemask, i));*/

for (int i=6984; i<=7245; i++)
    chain->Add(Form(filemask, i));

for (int i=7368; i<=7395; i++)
    chain->Add(Form(filemask, i));

for (int i=7568; i<=7647; i++)
    chain->Add(Form(filemask, i));

for (int i=7649; i<=7706; i++)
    chain->Add(Form(filemask, i));

for (int i=7709; i<=7838; i++)
    chain->Add(Form(filemask, i));

return chain;
}

```

Ułatwia to pracę z danymi, które są podzielone na wiele plików.

3.2 Deklarowanie gałęzi i liści drzewa

Zdefiniowanie różnych klas detektorów użytych w eksperymencie oraz ustawienie adresów gałęzi dla tych klas detektorów w drzewie wejściowym pozwala na dostęp do danych przechowywanych w drzewie wejściowym w celu dalszego przetwarzania. Wykorzystywane do tego są pliki nagłówkowe zdefiniowane wcześniej, które są używane do późniejszego stworzenia liści (podfolderów). W pierwszej gałęzi „Info”, która wychwytuje dane o samym pobranym drzewie, na przykład numery pakietów danych, z jakich korzystamy. Numer ten będzie potrzebny przy kalibracji widm, ponieważ w zależności od tego, z jakiego pakietu korzystamy, używamy innych współczynników kalibracyjnych.

```

struct {
    int nrun;
    int id;
    int trig;
    int mult;
} nfo;

Plastic * plastic = new Plastic;

```

```

Kratta * kratta = new Kratta;
Paris * paris = new Paris;
LaBr * labr = new LaBr;
Silicon * silicon = new Silicon;

tree->SetBranchAddress("paris", &paris);
tree->SetBranchAddress("labr", &labr);
tree->SetBranchAddress("plastic", &plastic);
tree->SetBranchAddress("kratta", &kratta);
tree->SetBranchAddress("silicon", &silicon);
tree->SetBranchAddress("info", &nfo);

```

Kolejnym etapem jest tworzenie pliku ROOT o nazwie „13C_data.root” oraz nowego drzewa o nazwie „tree2” i przypisanej zmiennej T1, w którym przechowywane będą dane wyjściowe.

```

TFile *f1 = new TFile("13C_data.root", "recreate");
TTree *T1 = new TTree("tree2", "tree2");

```

Poniżej przedstawiona jest definicja gałęzi dla drzewa wyjściowego, określająca strukturę danych dla każdego rodzaju detektora (LaBr, Plastic, Paris, Kratta, Silicon) w programie.

```

struct
{
    int id;
    float time;
    float energyqshort;
    float energyqlong;
    float energyqshort_cal;
}
LaBr;

struct
{
    int id;
    float time;
}
Plastic;

struct
{
    int id;
    float time;
    float energyqlong;
    float energyqshort;
}
Paris;

struct
{
    int id;
    float energyPDO;
    float energyPD1;
    float energyPD2;
    float timePDO;
    float timePD1;
    float timePD2;
}

```

```

        float energyPD2_cal;
    }
Kratta;

struct
{
    int id;
    float energy;
    float energy_cal;
    float time30;
    float time80;
    float difftime;
    float difftime_cal;
}
Silicon;

```

Poniższy fragment kodu używa funkcji `TTree::Branch` do zdefiniowania gałęzi (Branch) w drzewie `T1` dla różnych detektorów i informacji ogólnych. Definiuje formaty danych dla tych gałęzi. Pierwszy argument to nazwa gałęzi. Drugi wskazuje na obiekt klasy zdefiniowany poprzednio. Trzeci argument to format danych, gdzie na początku jest nazwa liścia (podfolderu), następnie sam format: /I to liczby całkowite, /F liczby zmiennoprzecinkowe (float), poszczególne listki oddzielone są znakiem „:”,

```

T1->Branch("LaBr",&LaBr,"id/I:time/F:energyqshort
/F:energyqlong/F:energyqshort_cal/F");

T1->Branch("Plastic",&Plastic,"id/I:time/F");

T1->Branch("Paris",&Paris,"id/I:time/F:energyqlong
/F:energyqshort/F");

T1->Branch("Kratta",&Kratta,"id/I:energyPD0/F:energyPD1
/F:energyPD2/F:timePD0/F:timePD1/F:timePD2
/F:energyPD2_cal/F");

T1->Branch("Silicon",&Silicon,"id/I:energy/F:energy_cal
/F:time30/F:time80/F:difftime/F:difftime_cal/F");

T1->Branch("Info",&nfo,"run/I:id/I:trig/I:mult/I");

```

3.3 Wczytywanie współczynników kalibracyjnych

Jedną z istotnych funkcjonalności programu jest wczytywanie współczynników kalibracyjnych z zewnętrznych plików tekstowych. Pozwala to na wyświetlanie zmierzonych wielkości, np. energii cząstek, w jednostkach fizycznych oraz sumowanie danych zebranych przez różne detektory danego typu. Procedura ta została opisana poniżej, na przykładzie detektora krzemowego Silicon (DSSSD). Dla pozostałych detektorów metoda jest taka sama, zmienia się jedynie ilość współczynników potrzebnych do wykorzystania przez dany detektor.

W pierwszym kroku program otwiera plik do odczytu. Jeśli plik nie został otwarty poprawnie, program wypisuje komunikat o błędzie i kończy działanie z kodem błędu. W przypadku poprawnego otwarcia program używa pętli do wczytania danych z pliku do tablic. Każda linia

pliku powinna zawierać dwie lub trzy liczby zależne od pliku z współczynnikami kalibracyjnymi. Jeśli wczytanie danych nie powiedzie się, program wypisuje komunikat o błędzie i ustawia wartości na 0 i 1. Jeśli wartość wczytana z pliku to NaN (Not a Number), program również ustawia wartości na 0 i 1. Wartości wczytane z pliku dodawane są na koniec wektorów.

```
...
//-----
// wczytanie wspolczynników kalibacyjnych do energii Silicon
//-----

std::vector<double> sil_cal0, sil_cal1;

cout << "cal_silicon" << endl;

FILE *plik;
plik = fopen("calibrations/cal_silicon.txt", "r");
if (plik == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

double sil1[32], sil2[32];
for (int i = 0; i < 32; i++) {
    if (fscanf(plik, "%lf %lf", &sil1[i], &sil2[i]) != 2) {
        printf("Blad wczytywania danych z wiersza %d.\n",
               i + 1);
        sil1[i] = 0;
        sil2[i] = 1;
    }
    if (std::isnan(sil1[i]) || std::isnan(sil2[i])) {
        printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
        sil1[i] = 0;
        sil2[i] = 1;
    }

    sil_cal0.push_back(sil1[i]);
    sil_cal1.push_back(sil2[i]);
}

for (int i = 0; i < sil_cal0.size(); i++)
    cout << sil_cal0[i] << "\t" << sil_cal1[i] << endl;
cout << " " << endl;
fclose(plik);

//-----
// wczytanie wspolczynników kalibacyjnych do energii Kratta
//-----


std::vector<double> kratta_cal0_I, kratta_cal1_I, kratta_cal2_I;
std::vector<double> kratta_cal0_II, kratta_cal1_II, kratta_cal2_II;
double kra11[24], kra12[24], kra13[24];
double kra21[24], kra22[24], kra23[24];

...
```

Po wczytaniu odpowiednich współczynników kalibracyjnych obsługiwana jest pętla po zda-

rzeniach (**events**) w drzewie wejściowym. Dane z kolejnych detektorów są przetwarzane poprzez pobranie przypisanych im wartości oraz zapisanie ich do poszczególnych gałęzi drzewa wyjściowego T1.

Detektor	Nazwa gałęzi	Znaczenie
LaBr	Id	Indeks detektora
	Time	Czas pomiaru
	Energyqshort	Energia gamma mierzona w oknie czasowym 200ns
	Energyqlong	Energia gamma mierzona w oknie czasowym 900ns
	Energyqshort_cal	Skalibrowana energia gamma mierzona w dłuższym okresie czasowym
Plastic	Id	Indeks detektora
	Time	Czas pomiaru
Paris	Id	Indeks detektora
	Time	Czas pomiaru
	Energyqlong	Energia mierzona w oknie czasowym 200ns
	Energyqshort	Energia mierzona w oknie czasowym 900ns
Kratta	Id	Indeks detektora
	Energy PD0	Energia mierzona przez fotodiodę PD0
	Energy PD1	Energia mierzona przez fotodiodę PD1
	Energy PD2	Energia mierzona przez fotodiodę PD2
	Time PD0	Czas pomiaru fotodiody PD0
	Time PD1	Czas pomiaru fotodiody PD1
	Time PD2	Czas pomiaru fotodiody PD3
	Energy_PD2_cal	Energia skalibrowana mierzona przez fotodiodę PD2
Silicon	Id	Indeks detektora
	Energy	Energia cząstek naładowanych
	Energy_cal	Energia skalibrowana
	Time30	Czasu pomiaru z CFD 30%
	Time80	Czasu pomiaru z CFD 80%
	Difftime	Różnica czasowa pomiędzy dwoma pomiarami z CFD
	Difftime_cal	Skalibrowana różnica czasowa

Tabela 1. Struktura drzewa ROOT, znaczenie poszczególnych liści w gałęziach. Występuje także kalibracja energii detektorów w zależności od numeru użytego pakietu danych (**nfo.nrun**).

Detektor	Pakiet danych	Pliki kalibracyjne
LaBr	4383-4724	labr_cal0_I, labr_cal1_I, labr_cal2_I
	6984-7070	labr_cal0_II_1, labr_cal1_II_1, labr_cal2_II_1
	7071-7157	labr_cal0_II_2, labr_cal1_II_2, labr_cal2_II_2
	7158-7245	labr_cal0_II_3, labr_cal1_II_3, labr_cal2_II_3
	7368-7640	labr_cal0_III, labr_cal1_III, labr_cal2_III
	7641-7838	labr_cal0_IV, labr_cal1_IV, labr_cal2_IV
Kratta	4383-4724	kratta_cal0_I, kratta_cal1_I, kratta_cal2_I
	6984-7838	kratta_cal0_II, kratta_cal1_II, kratta_cal2_II

Tabela 2. Spis używanych plików kalibracyjnych do określonych przedziałów pakietów.

Dla różnych danych z różnych plików wejściowych, korzystamy z innych współczynników kalibracyjnych wcześniej wgranych. Przy detektorach LaBr oraz KRATTA korzystamy z trzech współczynników kalibracyjnych (**labr_cal0_I**, **labr_cal1_I**, **labr_cal2_I** lub **kratta_cal0_I**,

`kratta_cal1_I, kratta_cal2_I), kalibrujemy energie mierzone przez detektory przy pomocy funkcji kwadratowej.`

```

//-----
// wypelnianie galezi nowego drzewa
//-----
```

```

Long64_t Nev = tree->GetEntries(); //liczba eventow
    std::cout << "-----"
        << std::endl;
    std::cout << " NUMBER OF EVENTS: " << Nev << std::endl;
    std::cout << "-----"
        << std::endl;

    for (Long64_t i=0;i<Nev;i++)
        { //Loop over events
        tree->GetEntry(i);
        std::cout<<"Processing entry no. "<<i<<
                    out of "<<Nev<<'\r'<<std::flush;
        if (i==Nev-1) std::cout<<'n'<<std::endl;

//-----
// Galaz LaBr
//-----
```

```

for (int l = 0; l < labr->mult(); l++){
    LaBr.id = labr->det[l].id;
    LaBr.time = labr->det[l].time;
    LaBr.energyqshort = labr->det[l].qshort;
    LaBr.energyqlong = labr->det[l].qlong;

    if(nfo.nrun >= 4383 && nfo.nrun <= 4724){
        LaBr.energyqshort_cal = labr_cal2_I[LaBr.id]*
        LaBr.energyqshort*LaBr.energyqshort +
        labr_cal1_I[LaBr.id]*LaBr.energyqshort +
        labr_cal0_I[LaBr.id];

    }else if(nfo.nrun>=6984 && nfo.nrun<=7070){
        LaBr.energyqshort_cal = labr_cal2_II_1[LaBr.id]*
        LaBr.energyqshort*LaBr.energyqshort +
        labr_cal1_II_1[LaBr.id]*LaBr.energyqshort +
        labr_cal0_II_1[LaBr.id];
    }else if(nfo.nrun>=7071 && nfo.nrun<=7157){
        LaBr.energyqshort_cal = labr_cal2_II_2[LaBr.id]*
        LaBr.energyqshort*LaBr.energyqshort +
        labr_cal1_II_2[LaBr.id]*LaBr.energyqshort +
        labr_cal0_II_2[LaBr.id];
    }else if(nfo.nrun>=7158 && nfo.nrun<=7245){
        LaBr.energyqshort_cal = labr_cal2_II_3[LaBr.id]*
        LaBr.energyqshort*LaBr.energyqshort +
        labr_cal1_II_3[LaBr.id]*LaBr.energyqshort +
        labr_cal0_II_3[LaBr.id];
    }else if(nfo.nrun>=7368 && nfo.nrun<=7640){
```

```

        LaBr.energyqshort_cal = labr_cal2_III[LaBr.id]*
        LaBr.energyqshort*LaBr.energyqshort +
        labr_cal1_III[LaBr.id]*LaBr.energyqshort +
        labr_cal0_III[LaBr.id];
    }else if(nfo.nrun>=7641 && nfo.nrun<=7838){
        LaBr.energyqshort_cal = labr_cal2_IV[LaBr.id]*
        LaBr.energyqshort*LaBr.energyqshort +
        labr_cal1_IV[LaBr.id]*LaBr.energyqshort +
        labr_cal0_IV[LaBr.id];
    }else{
        LaBr.energyqshort_cal = LaBr.energyqshort;
    }

    T1->Fill();
}

//-----
// Galaz Plastic
//-----

for (int l = 0; l < plastic->mult(); l++){
    Plastic.id = plastic->det[l].id;
    Plastic.time = plastic->det[l].time;
    T1->Fill();
}
//-----
// Galaz Paris
//-----

for (int l = 0; l < paris->mult(); l++){
    Paris.id = paris->det[l].id;
    Paris.time = paris->det[l].time;
    Paris.energyqshort = paris->det[l].qshort;
    Paris.energyqlong = paris->det[l].qlong;
    T1->Fill();
}
//-----
// Galaz Kratta
//-----

for (int l = 0; l < kratta->mult(); l++){
    Kratta.id = kratta->det[l].id;
    Kratta.energyPDO = kratta->det[l].pd0.ampl;
    Kratta.energyPD1 = kratta->det[l].pd1.ampl;
    Kratta.energyPD2 = kratta->det[l].pd2.ampl;
    Kratta.timePDO = kratta->det[l].pd0.time;
    Kratta.timePD1 = kratta->det[l].pd1.time;
    Kratta.timePD2 = kratta->det[l].pd2.time;
    if(nfo.nrun >= 4383 && nfo.nrun <= 4724){
        Kratta.energyPD2_cal = kratta_cal2_I[Kratta.id]*
        Kratta.energyPD2*Kratta.energyPD2 +
        kratta_cal1_I[Kratta.id]*Kratta.energyPD2 +
        kratta_cal0_I[Kratta.id];
    }else if(nfo.nrun >= 6984 && nfo.nrun <= 7838){
        Kratta.energyPD2_cal = kratta_cal2_II[Kratta.id]*
        Kratta.energyPD2*Kratta.energyPD2 +
        kratta_cal1_II[Kratta.id]*Kratta.energyPD2 +

```

```

                kratta_cal0_II[Kratta.id];
} else {
    Kratta.energyPD2_cal = Kratta.energyPD2;
}
T1->Fill();

}

```

W przypadku detektora DSSSD, w związku z potrzebą skalibrowania macierzy czasowych, przeprowadzono dokładniejszą analizę, której szczegóły są opisane w następnym rozdziale.

```

//-----
// Galaz Silicon
//-----
for (int l = 0; l < silicon->mult(); l++){
    if(silicon->det[l].id != 26 && silicon->det[l].id != 27
       && silicon->det[l].id != 28 &&
       silicon->det[l].id != 29){

        Silicon.id = silicon->det[l].id;
        Silicon.energy = silicon->det[l].ampl;
        Silicon.energy_cal = sil_cal0[Silicon.id] +
                               sil_cal1[Silicon.id] * Silicon.energy;
        Silicon.time30 = silicon->det[l].time30;
        Silicon.time80 = silicon->det[l].time80;
        Silicon.difftime = Silicon.time30 -
                           Silicon.time80;
    }
}

```

Co można zauważyć, to wykluczenie detektorów o indeksie od 26 do 29, przez złą jakość pomiarów oraz inną metodę kalibrowania niż w przypadku pozostałych detektorów. Kalibrujemy różnicę czasowe, a nie energie, i używamy do tego funkcji liniowej. Dla detektorów o indeksach od 20 do 23 używamy innych współczynników kalibracyjnych zależnych od pakietów danych jakich używamy.

```

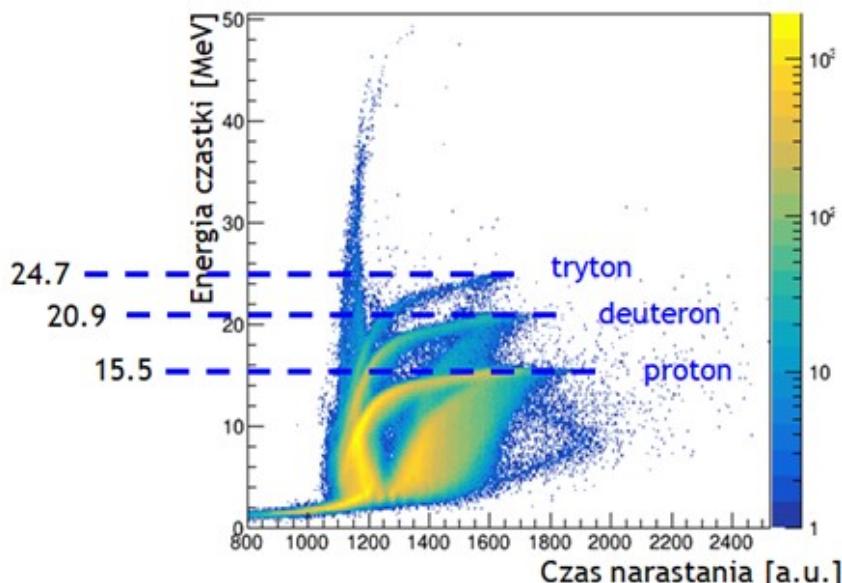
if(silicon->det[l].id >=20 && silicon->det[l].id <=23){
    if (nfo.nrun >= 7635 && nfo.nrun <= 7838) {
        Silicon.difftime_cal = diff_cal0_hist1[Silicon.id-20]+
                               diff_cal1_hist1[Silicon.id - 20] * Silicon.difftime;
    } else if (nfo.nrun >= 6984 && nfo.nrun <= 7027) {
        Silicon.difftime_cal = diff_cal0_hist1[Silicon.id-20]+
                               diff_cal1_hist1[Silicon.id - 20] * Silicon.difftime;
    } else if (nfo.nrun >= 7028 && nfo.nrun <= 7634) {
        Silicon.difftime_cal = diff_cal0_hist2[Silicon.id-20]+
                               diff_cal1_hist2[Silicon.id - 20] * Silicon.difftime;
    }
}

else {
    Silicon.difftime_cal = diff_cal0[Silicon.id] +
                           diff_cal1[Silicon.id] * Silicon.difftime;
}
}
T1->Fill();
}

```

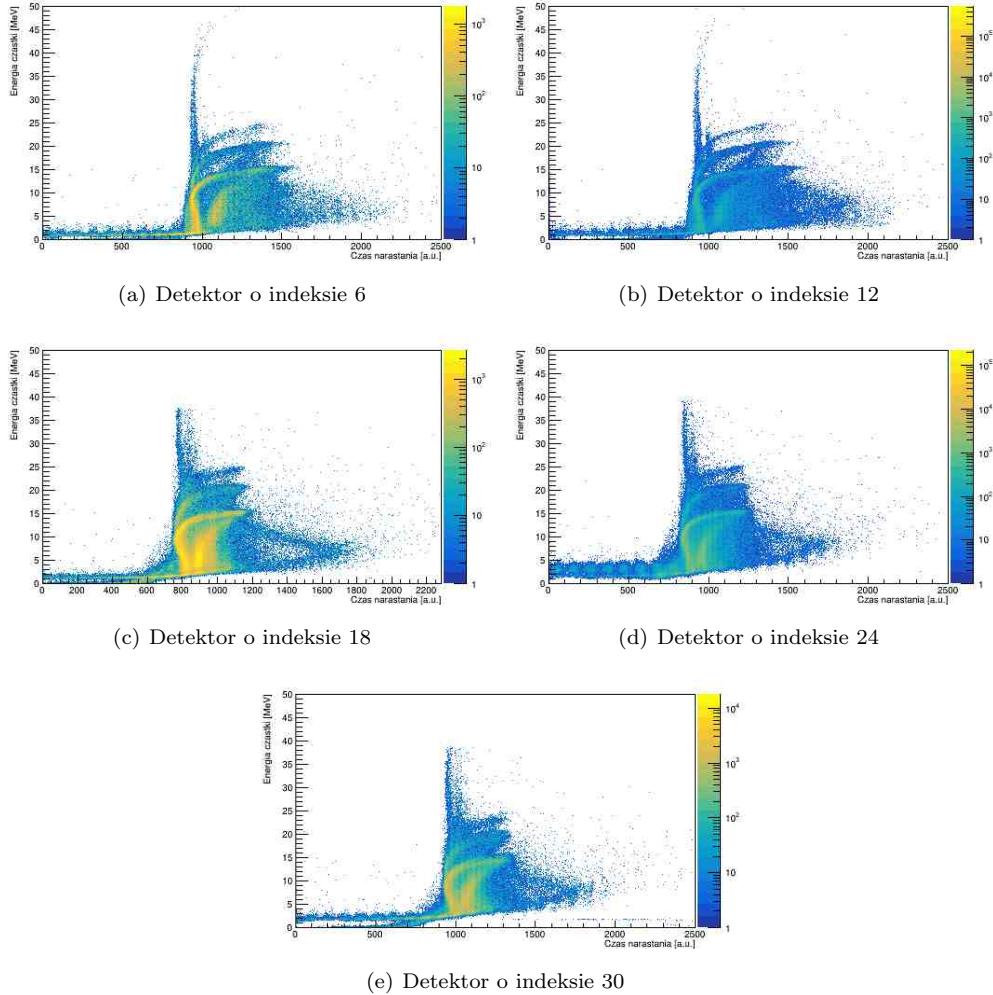
3.4 Kalibracja macierzy do identyfikacji lekkich cząstek naładowanych

Macierz identyfikacji cząstek naładowanych służy nam do klasyfikacji i identyfikacji różnych typów cząstek naładowanych, takich jak protony, deuterony, trytony. Osie macierzy reprezentują energię, na podstawie amplitudy zdigitalizowanej z impulsu oraz czas. Na osi pionowej jest przedstawiona energia zdeponowana w detektorze wyrażona w MeV. Na osi poziomej jest tzw. czas narastania. Czas ten definiowany jest jako różnica dwóch wartości z dwóch dyskryminatorów stałofrakcyjnych, które dostarczają informacji o czasie, w którym sygnał osiąga 30% i 80% maksymalnej wartości.



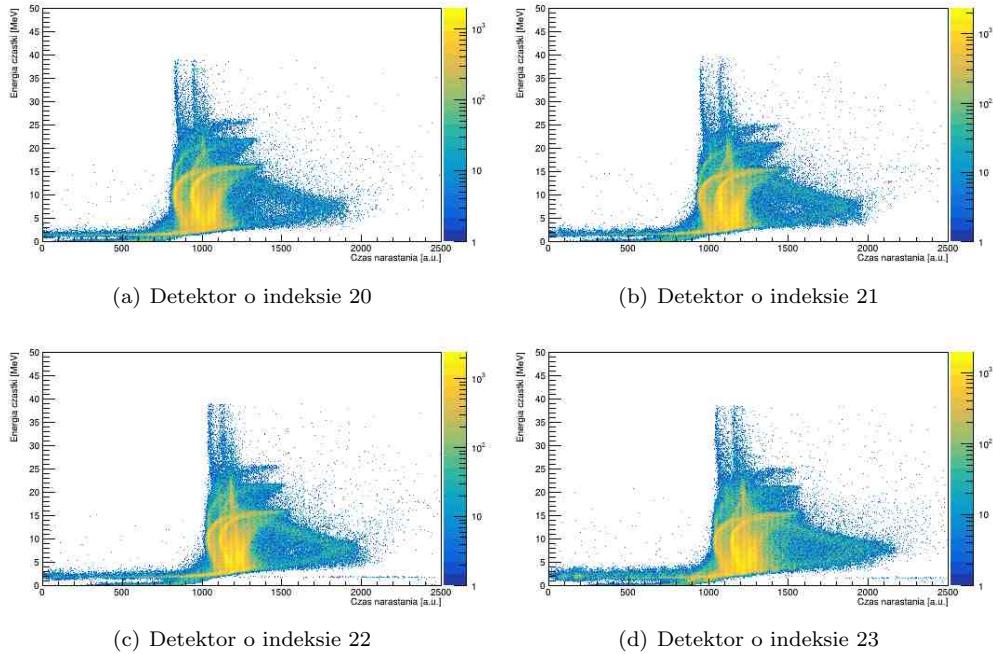
Rysunek 3.2: Przykładowa macierz identyfikacji cząstek naładowanych [4].

Dalszą częścią pracy po stworzeniu struktury danych była kalibracja czasowa detektora DSSSD, czyli wyznaczenie współczynników kalibracyjnych dla tych pomiarów. Chcemy, aby macierz powstała przez zsumowanie macierzy otrzymanych z wszystkich pasków była czytelna i użyteczna do późniejszych analiz.



Rysunek 3.3: Macierze identyfikacji cząstek naładowanych przed kalibracją.

Głównym problemem było brak regularności, tzn. w różnych detektorach, były inne problemy. Wygląd większości macierzy był podobny, tylko delikatnie przesunięty w lewą lub prawą stronę względem macierzy detektora o indeksie numer 25, do którego kalibrujemy pozostałe detektory. Wyjątkami były detektory o indeksach od 20 do 23, na wykresach pojawiały się podwójne struktury macierzowe, które należało wyodrębnić oraz osobno dla nich wyliczyć współczynniki kalibracyjne. Wykorzystano do tego informację z gałęzi Info z numerami użytych pakietów danych, ustalono przedziały dla których tworzą się poszczególne struktury, tj. lewa macierz od run'a 6984 do 7027 oraz od 7635 do 7838; prawa macierz od run'u 7028 do 7634 i użyto tych informacji do specjalnego wyliczenia współczynników co można zobaczyć przy opisie kodu.



Rysunek 3.4: Macierze do identyfikacji cząstek naładowanych z podwójnymi strukturami.

Zostały także wykluczone klika detektorów, ze względu na źle skalibrowaną energię (indeksy od 26 do 29).

Całość programu do kalibracji czasowej macierzy identyfikacji cząstek składa się z trzech części: kalibracji histogramów z jednym wykresem, kalibracji pierwszej części histogramu z dwoma wykresami oraz drugiej części wykresu. Metoda kalibracji jest taka sama dla każdej z tych części, z drobnymi różnicami wartości zmiennych przy fitowaniu funkcji, dlatego przy opisie programu posłużę się jedną trzecią częścią programu. Poniżej znajduje się kod napisany w języku C++, który za pomocą środowiska ROOT, oblicza współczynniki kalibracyjne dla pomiarów czasowych mierzonych przez detektor krzemowy DSSSD.

Na początku kodu dołączane są niezbędne biblioteki ROOT oraz C++. Zdefiniowane są różne tablice i zmienne do przechowywania danych i parametrów. Tablica `times[2]` przechowuje wartości miejsc pików wyznaczonych z histogramu czasowego z detektora o indeksie 25, do którego kalibrujemy pozostałe detektory. Tablica `cal[2]` przechowuje współczynniki kalibracyjne do późniejszego zapisu w pliku. Z kolei tablice `par_g[6]`, `par_l[2]`, `par_e[2]` zawierają parametry funkcji, którymi fitujemy histogramy, a `peaks[2]` przechowuje pozycje znalezionych pików. Ponadto tablice `timesprim[2]` oraz `div[2]` są wykorzystywane przy wizualizacji przeprowadzonych kalibracji, przy czym pierwsza z nich przechowuje wartości nowych pozycji pików, obliczonych przy zastosowaniu współczynników kalibracyjnych, natomiast w drugiej znajdują się różnice między wartościami nieskalibrowanych i skalibrowanych położen pików, potrzebne do wyrysowania wykresu residuum.

```
#include "TTree.h"
#include "TCutG.h"
#include "TH2F.h"
...

```

```

void Peaks()
{
Double_t times[2] = {1128.782994885779090, 1236.521202591038673};
//ID = 25
Double_t timesprim[2];
Double_t cal[2];
Double_t par_g[6];
Double_t par_l[2];
Double_t par_e[2];
Double_t peaks[2];
Double_t div[2]={0,0};

int p=0;

```

W pierwszym kroku po deklaracji tablic program inicjalizuje czytnik (TTreeReader), który pozwala na odczyt danych z drzewa o nazwie „tree2” znajdującego się w pliku wejściowym. Następnie definiuje trzy zmienne TTreeReaderValue, które będą używane do odczytywania danych z drzewa, czyli indeks detektora, czas oraz numer pakietu danych. Następnie tworzone są trzy mapy histogramów: histogram_1 i histogram_2 dla identyfikatorów detektorów krzemowych (Silicon.id) w zakresie od 20 do 23 oraz other_histograms dla pozostałych identyfikatorów w zakresie od 0 do 19 i od 24 do 31 z pojedynczymi strukturami macierzowymi. Histogramy będą przechowywały dane w zależności od identyfikatorów i numerów przebiegów. W pętli while czytane są kolejne zdarzenia z drzewa (myReader.Next()).

```

// ----- OPENING THE FILE AND TAKING THE HISTOGRAMS: ----- //
TFile *myFile = TFile::Open("./13C_data.root");

TTreeReader myReader("tree2", myFile);
TTreeReaderValue<Int_t> id(myReader, "Silicon.id");
TTreeReaderValue<Float_t> difftime(myReader, "Silicon.difftime");
TTreeReaderValue<Int_t> run(myReader, "Info.run");

std::map<Int_t, TH1F*> histogram_1;
std::map<Int_t, TH1F*> histogram_2;

for (int i = 20; i <= 23; ++i) {
    histogram_1[i] = new TH1F(Form("hist1_%02d", i),
        Form("Silicon.Id = %02d (Run 6984-7027, 7635-7838)", i),
        500, 0, 2500);
    histogram_2[i] = new TH1F(Form("hist2_%02d", i),
        Form("Silicon.Id = %02d (Run 7028-7634)", i),
        500, 0, 2500);
}

std::map<Int_t, TH1F*> other_histograms;

for (int i = 0; i <= 31; ++i) {
    if (other_histograms.find(i) == other_histograms.end()) {
        other_histograms[i] = new TH1F(Form("hist_other_%02d", i),
            Form("Silicon.Id = %02d", i), 500, 10, 2500);
    }
}

```

```

while (myReader.Next())
{
    Int_t currentId = *id;
    Int_t currentRun = *run;
    Float_t currentDiffTime = *diffTime;

    if (currentId >= 20 && currentId <= 23) {
        if ((currentRun >= 6984 && currentRun <= 7027)
            || (currentRun >= 7635 && currentRun <= 7838)) {
            histogram_1[currentId]->Fill(currentDiffTime);
        }
        else if (currentRun >= 7028 && currentRun <= 7634) {
            histogram_2[currentId]->Fill(currentDiffTime);
        }
    }

    other_histograms[currentId]->Fill(currentDiffTime);
}

```

Tworzony jest plik wyjściowy o nazwie „cal_diffTime.txt”, który będzie przechowywał wyniki kalibracji czasu. Następnie tworzony jest nowy plik ROOT o nazwie „graph.root”, w którym będą przechowywane wykresy związane z analizą. Zdefiniowane są również tablice FWHM[2][32] (szerokość połówkowa), ChiSqr[2][32] (wartości chi-kwadrat) dla każdego histogramu oraz tablica cal_par[2][32] do przechowywania parametrów kalibracji czasu dla każdego piku. Następnie następuje przeszukiwanie histogramów w mapie other_histograms. Na początku histogram jest rysowany, wykonywane jest wyszukiwanie pików za pomocą funkcji TSpectrum i wyświetlane są znalezione piki. Jeśli nie znaleziono żadnych (nfound == 0), zapisywane są wartości zerowe do pliku „cal_diffTime.txt”.

Następuje pobranie z histogramu pozycji pików, które zostały znalezione przez algorytm, do tablicy xpeaks. Aby brać pod uwagę piki, które nas interesują, iterujemy przez pozycje, aby wybrać tylko te, które mają wartość większą niż 750 i zapisujemy je w tablicy peaks. Tablica jest sortowana rosnąco i wyświetlane są pozycje pików, zarówno przed jak i po posortowaniu.

```

// ----- WITH ONE HISTOGRAM: -----
fstream plik;
plik.open("cal_diffTime.txt", std::fstream::out
          | std::fstream::trunc);
TFile *fout = new TFile("./graph.root","recreate");
Double_t FWHM[2][32];
Double_t ChiSqr[2][32];
Double_t cal_par[2][32];

// ----- SEARCHING FOR THE PEAKS: -----
for(auto &pair : other_histograms)
{
    TH1F *hist = pair.second;
    hist->Draw();

    printf("START for Silicon.id = %d\n", pair.first);
}

```

```

printf("\n");

Int_t nfound = 0;
TSpectrum *s = new TSpectrum();
nfound = s->Search(hist,4,"",0.06);
hist->ShowPeaks(4,"",0.05);
printf("nfound = %d \n",nfound);

if (nfound == 0){
    cal[0] = 0;
    cal[1] = 1;
    plik << cal[0] << "                " << cal[1] << endl;
    continue;
}

// ----- TAKING THE PEAK POSITIONS: -----
Double_t *xpeaks = s->GetPositionX();

int i=0;
for (p = 0; p<2; p++){
    printf("Position of the %d peak: %3.2f\n",p+1,xpeaks[p]);

    if(xpeaks[p]>750){
        peaks[i]=xpeaks[p];
        i++;
    }
}

printf("\n");

std::sort(peaks, peaks+2);

for (p = 0; p<2; p++)
    printf("Position of the %d peak after swap:
%3.2f\n",p+1,peaks[p]);

printf("\n");

```

Pętla `for` iteruje przez każdy z dwóch pików i dopasowuje funkcję w celu znalezienia położenia wierzchołka piku. Dla każdego z nich definiowane są granice dopasowania funkcji `r1`, `r2`, `r3` i `r4` na podstawie pozycji piku.

```
// ----- FITTING FUNCTIONS: ----- //
```

```

for (p=0;p<2;p++) {
    Double_t r1 = peaks[p];
    Double_t r2 = peaks[p] + 30;
    Double_t r3 = peaks[p] - 50;
    Double_t r4 = peaks[p] - 15;

```

W celu uzyskania jak najdokładniejszego położenia piku, zastosowano trzy warianty funkcji fitujących zależnych od zdefiniowanych warunków. Dzięki temu unikamy błędów z dobraniem nieodpowiedniej funkcji lub kombinacji funkcji.

Pierwszym wariantem jest kombinacja dwóch funkcji gaussowskich g oraz $g2$. Parametry dopasowanej funkcji są przechowywane w tablicy `par_g` pod warunkiem, że wartość pierwszego parametru funkcji g jest mniejsza od 10^5 . Jeśli tak nie jest, oznacza to, że dopasowanie funkcji Gaussa nie powiodło się i program przechodzi do próby fitowania za pomocą funkcji `gauss+exp`. Przy wyrysowaniu funkcji $g2$, zdefiniowano klika zmiennych (`i1`, `i2`, `p0`, `p2`, `Mean2`, `pozycja`) stanowiących parametry startowe przy fitowaniu. Dobranie tych wartości było wynikiem metody prób i błędów.

```

TF1 *g = new TF1("g","gaus",r4,r2);
hist->Fit(g,"NR+");
g->GetParameters(&par_g[0]);
peaks[p] = par_g[1];

Double_t i1 = 0.76, i2 = 0.24;
Double_t p0 = par_g[0];
Double_t p2 = par_g[2];
Double_t Mean2 = p0*i2/i1;
Double_t pozycja = peaks[p] - 5;

TF1 *g2 =new TF1("g2","gaus",r3,r4);
g2->SetParameters(Mean2,pozycja,p2);
hist->Fit(g2,"NR+");
g2->GetParameters(&par_g[3]);
Double_t pov = g->GetParameter(0);

if (abs(pov)>100000){

```

Program przechodzi do kolejnego wariantu, czyli kombinacji funkcji gaussowskiej $f1$ oraz eksponentjalnej $f2$. Warunkiem jest wartość trzeciego parametru funkcji Gaussa, σ . Jeśli przekracza 98.9 to przechodzimy do ostatniego wariantu.

```

g->Delete();
g2->Delete();
TF1 *f1 = new TF1("ga","gaus",r4,r2);
TF1 *f2 = new TF1("ex","expo",r3,r4);

hist->Fit(f1,"NR+");
f1->GetParameters(&par_g[0]);

hist->Fit(f2,"NR+");
f2->GetParameters(&par_l[0]);
Double_t pov2 = f1->GetParameter(2);

if(abs(pov2)>98.9){


```

Ostatnim wariantem jest pojedyncza funkcja Gaussa. Po fitowaniu, wyznaczone położenia pików są pobierane oraz obliczane są wartości szerokości połówkowych i wartość Chi-kwadrat dla danego histogramu.

```

f1->Delete();
f2->Delete();
TF1 *h = new TF1("h","gaus",r3,r2);
h->SetLineColor(6);

```

```

hist->Fit(h,"R+");
h->GetParameters(&par_g[0]);
ChiSqr[p][pair.first] = h->GetChisquare();
peaks[p] = h->GetParameter(1);
FWHM[p][pair.first] = 2.35*par_g[2];
}

```

Jeśli któryś z poprzednich wariantów był prawidłowy, funkcję łączymy w całość, **sum2** dla kombinacji gaus+exp, **total** dla kombinacji gaus+gaus. Na koniec pobierane są potrzebne wartości oraz wyznaczone położenia są wyświetlane.

```

else{
    TF1 *sum2 = new TF1("sum2","gaus(0)+[3]*"
                        "exp((x-[1])/[2])*TMath::Erfc((x-[1])/[2])",
                        r3,r2);
    sum2->SetParameters(par_g[0],par_g[1],par_g[2],par_l[0]);
    sum2->SetLineColor(9);
    hist->Fit(sum2,"R+");
    ChiSqr[p][pair.first] = sum2->GetChisquare();
    peaks[p] = par_g[1];
    FWHM[p][pair.first] = 2.35*par_g[2];

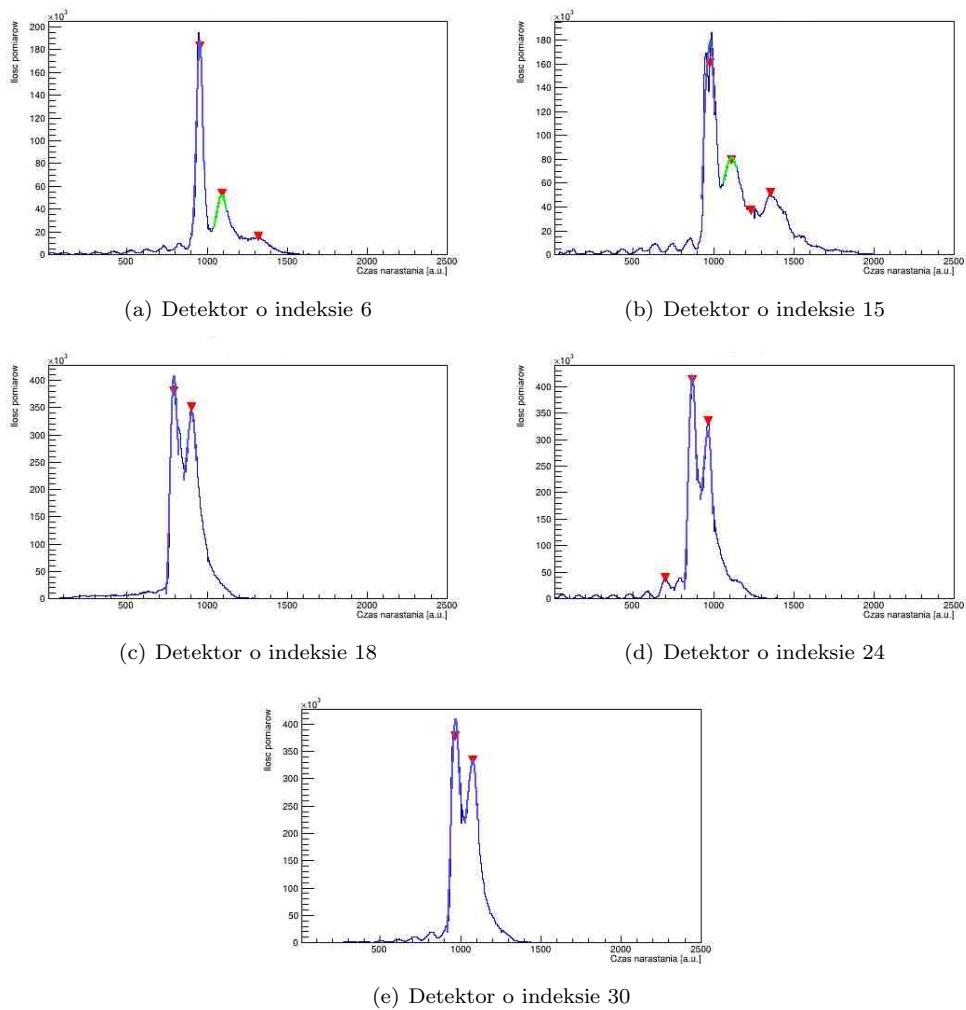
}
}
else {
    TF1 *total = new TF1("total","gaus(0)+gaus(3)",
                          r3,r2);
    total->SetLineColor(3);
    total->SetParameters(par_g);
    hist->Fit(total,"R+");
    ChiSqr[p][pair.first] = total->GetChisquare();
    peaks[p] = par_g[1];
    FWHM[p][pair.first] = 2.35*par_g[2];

}

printf("Position of the %d peak from fit: %3.2f\n",p+1,peaks[p]);
printf("FWHM: %3.2f\n",FWHM[p][pair.first]);
}

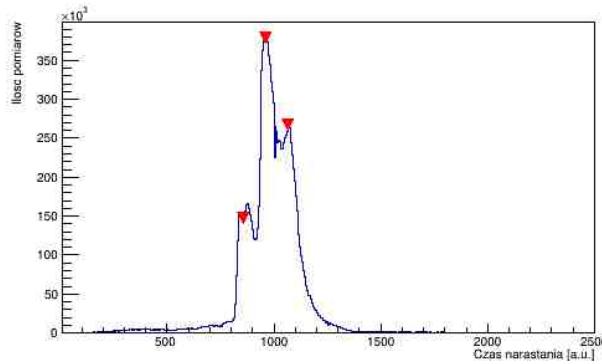
```

Program wygenerował histogramy różnic czasowych z detektora DSSSD i dopasował funkcje do znalezienia położen pików. Na rysunkach 3.5 histogramów widać dwa wyraźne piki, dlatego do wyliczenia współczynników kalibracyjnych użyto funkcji liniowej.

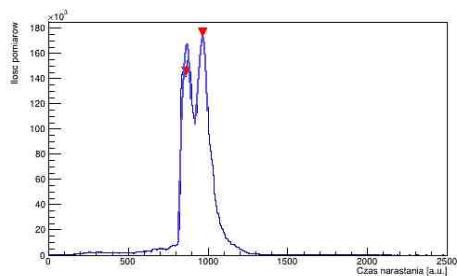


Rysunek 3.5: Przykłady histogramów czasów narastania. Czerwonymi trójkątami oznaczają położenia pików odszukane przez funkcję TSpektrum, natomiast kolorem niebieskim zostały oznaczone funkcje dofitowane do danych w celu dokładniejszego określenia położenia centroid pików.

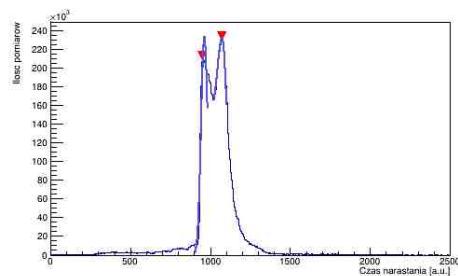
W przypadku detektorów z podwójnymi strukturami, histogramy zawierały trzy piki, jednak po oddzieleniu ze względu na numery użytych pakietów danych udało się uzyskać dwa histogramy z dwoma pikami, co można zobaczyć na rysunku 3.6.



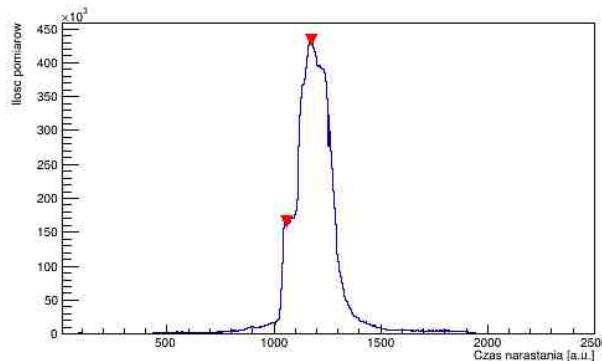
(a) Detektor o indeksie 6



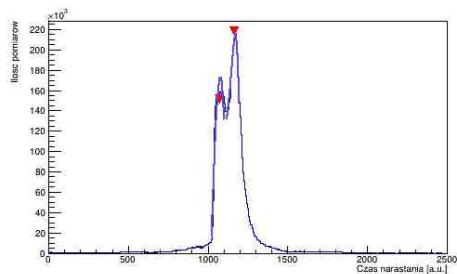
(b) Histogram czasowy od macierzy po lewej stronie wykresu, czyli pakietów danych od numeru 6984 do 7027 oraz od 7635 do 7838.



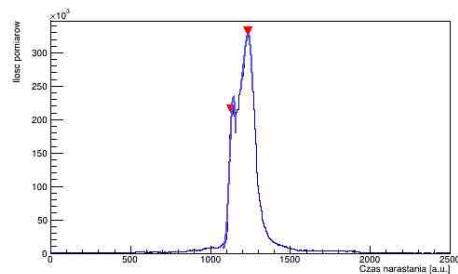
(c) Histogram czasowy od macierzy po prawej stronie wykresu, czyli pakietów danych od numeru 7028 do 7634.



(d) Detektor o indeksie 22



(e) Histogram czasowy od macierzy po lewej stronie wykresu, czyli pakietów danych od numeru 6984 do 7027 oraz od 7635 do 7838.



(f) Histogram czasowy od macierzy po prawej stronie wykresu, czyli pakietów danych od numeru 7028 do 7634.

Rysunek 3.6: Histogramy różnic czasowych z detektorów o indeksach 20(a, b, c) i 22(d, e, f), gdzie występowały dwie struktury macierzy, przed i po rozdzieleniu

W poniższej części kodu dokonuje się kalibracja poprzez dopasowanie linii prostych do punktów reprezentujących piki na histogramie. Najpierw tworzony jest obiekt `TGraph` zawierający pozycje pików (`peaks`) i odpowiadające im wartości czasu (`times`). Następnie tworzony jest obiekt `TCanvas` (`grp`) służący do wykreszenia wyniku dopasowania i residuum jako dwie części `grp`. Dla pierwszej części (`cd(1)`) wykonywane jest dopasowanie wielomianu pierwszego stopnia (`pol1`) do danych zawartych w `gr`. Parametry wielomianu są odczytywane i zapisywane w tablicy `cal` i następnie zapisywane w pliku tekstowym. Program tworzy trzy pliki tekstowe dla każdej mapy histogramów, każdego rozpatrywanego przypadku.

```
// ----- ENERGY CALIBRATION: ----- //

TGraph *gr = new TGraph((sizeof(peaks) / sizeof(Double_t)),
                        peaks, times);
TCanvas *grp = new TCanvas(Form("Fit%d", pair.first));
grp->Divide(1, 2);
gr->Fit("pol1");
gr->SetMarkerStyle(20);
gr->SetMarkerColor(4);
gr->SetLineColor(0);
gr->SetTitle(Form("Calibration Id = %d", pair.first));
gr->GetYaxis()->SetTitle("Times");
gr->GetXaxis()->SetTitle("Channel");
grp->cd(1);
gr->Draw();

TF1 *myfunc = gr->GetFunction("pol1");

for(int i = 0; i<2 ; i++){
    cal[i] = myfunc->GetParameter(i);
    printf("%d cal. coef. for %d spectrum: %3.2f\n",
           i, pair.first, cal[i]);
    cal_par[i][pair.first] = cal[i];
}
plik << cal[0] << "                " << cal[1] << endl;

for(p=0;p<2;p++){
    timesprim[p] = abs(cal[0]+cal[1]*peaks[p]);
    div[p]=times[p]-timesprim[p];
}
```

Następnie obliczane są przesunięcia czasowe (`div`) na podstawie dopasowanych kalibracji. Druga część `TCanvas` (`cd(2)`) służy do wykreszenia residuum. Wszystko jest zapisywane w jednym z wcześniejszych zdefiniowanych plików .root dla każdej mapy histogramów, `graph.root`, `graph_hist1.root`, `graph_hist2.root`.

```
TGraph *gr2 = new TGraph((sizeof(peaks) / sizeof(Double_t)),
                        peaks, div);
gr2->SetMarkerStyle(20);
gr2->SetMarkerColor(4);
gr2->SetLineColor(0);
gr2->SetTitle("Residuum");
gr2->GetYaxis()->SetTitle("Delta Times");
gr2->GetXaxis()->SetTitle("Channel");
```

```

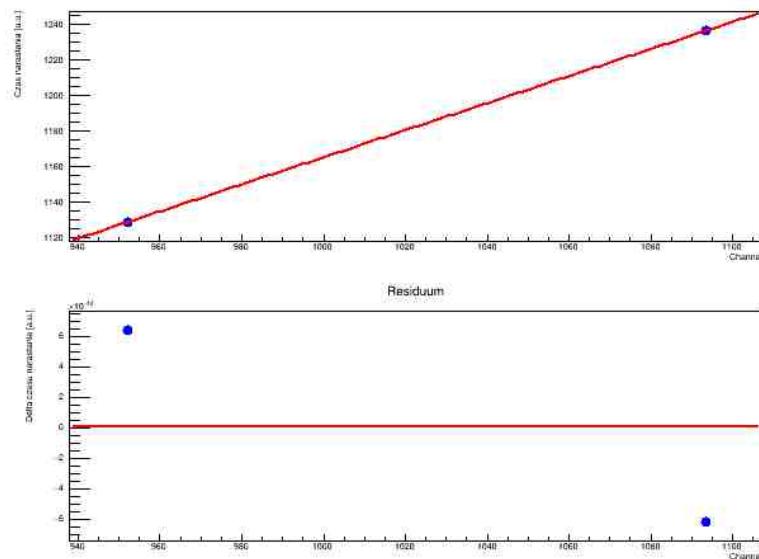
grp->cd(2);
gr2->Draw();
TF1 *l = new TF1("line","pol0");
l->SetParameters(0);
gr2->Fit(l);

fout->cd();
grp->Write();

} // end of loop on histograms

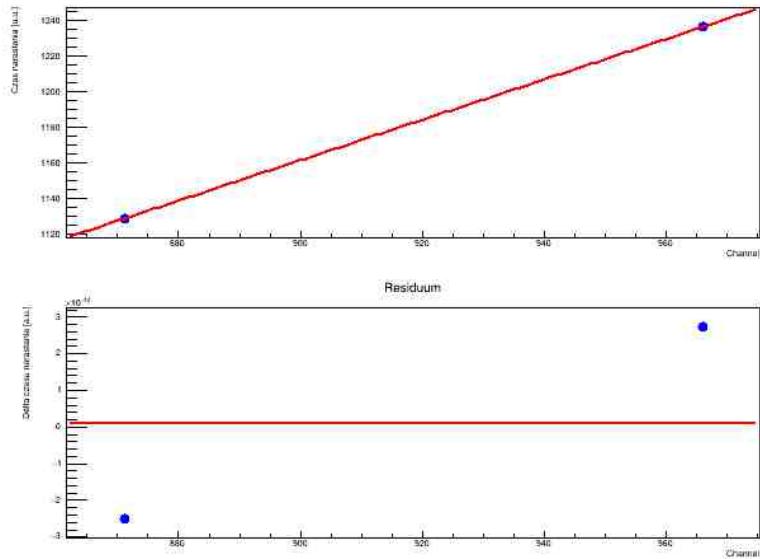
plik.close();
fout->Close();
...

```

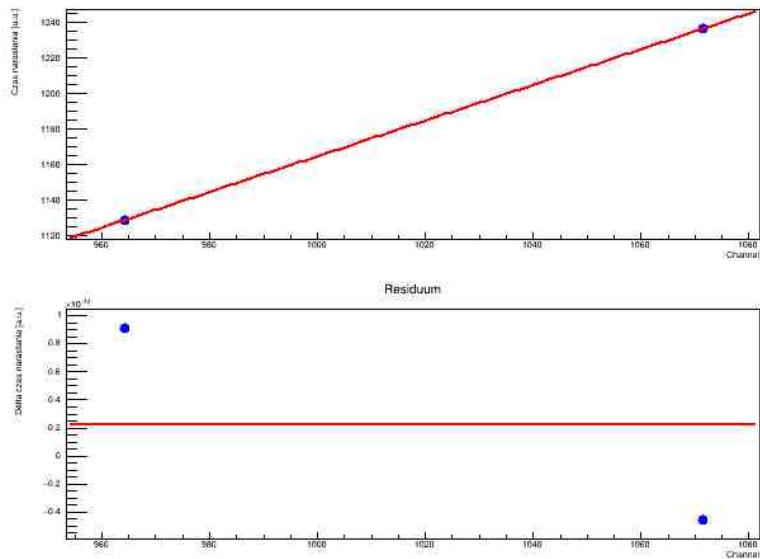


(a) Detektor o indeksie 6

Rysunek 3.7: Wyrysowane wyniki dopasowania funkcji liniowej oraz kalibracji dla detektorów o indeksie 6(a) oraz 20(b, c) (dla lewej i prawej macierzy).



(a) Detektor o indeksie 20, wykres kalibracyjny dla macierzy po lewej stronie wykresu (pakietów danych od numeru 6984 do 7027 oraz od 7635 do 7838).



(b) Detektor o indeksie 20, wykres kalibracyjny dla macierzy po prawej stronie wykresu (pakietów danych od numeru 7028 do 7634).

Rysunek 3.7: Wyrysowane wyniki dopasowania funkcji liniowej oraz kalibracji dla detektorów o indeksie 6(a) oraz 20(b, c) (dla lewej i prawej macierzy).

Ostatnia część kodu to tworzenie pliku „output.root”, w którym zapisywane są wszystkie histogramy z fitowanymi funkcjami, inaczej niż w przypadku współczynników kalibracyjnych oraz wykresów kalibracji czasu. Każdy histogram z trzech map histogramów (`histogram_1`, `histogram_2`, `other_histograms`) jest zapisywany do tego pliku za pomocą metody `Write()`.

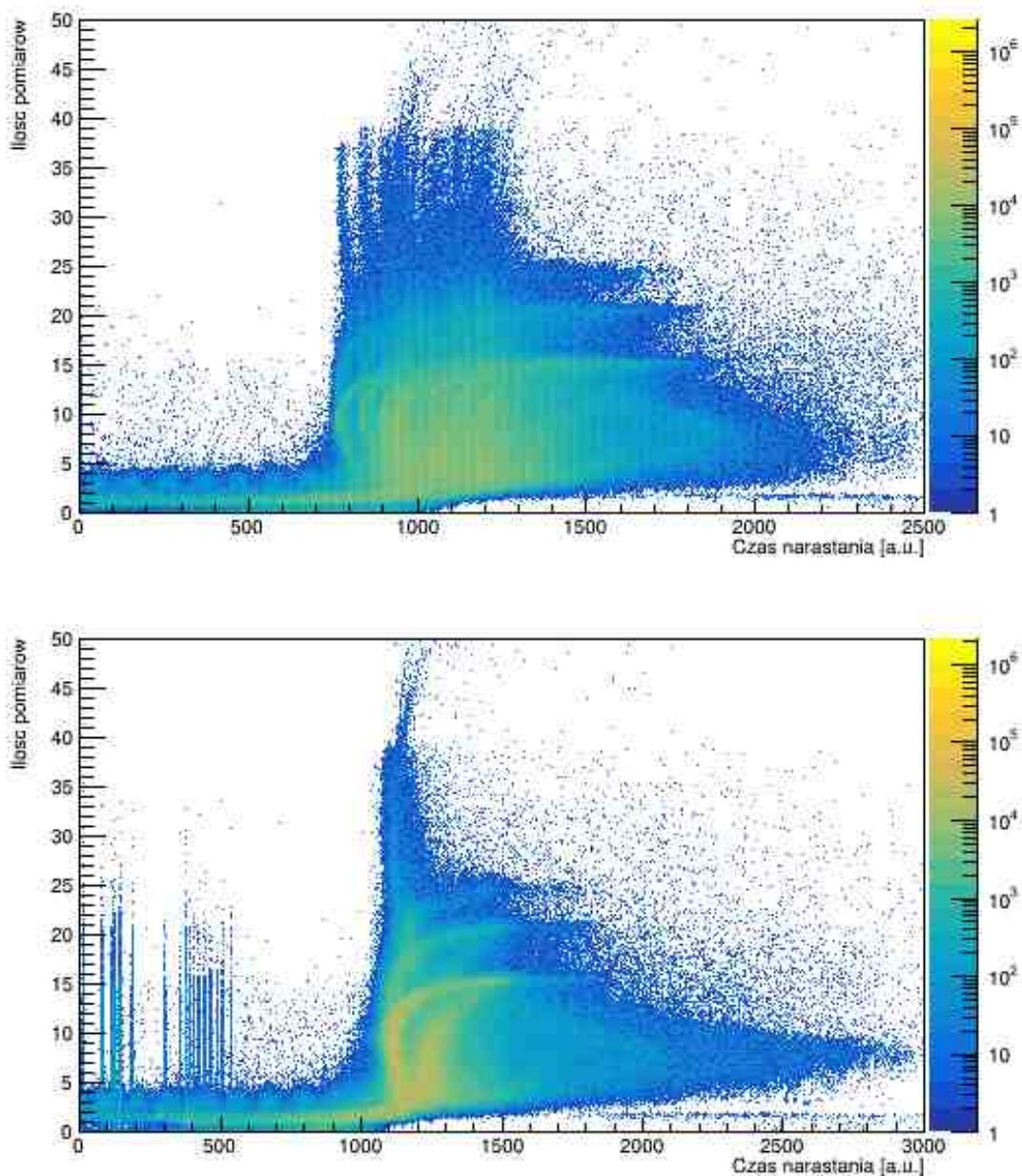
```
...
// ----- CREATING THE OUTPUT FILE WITH THE HISTOGRAMS: ----- //
```

```
TFile *outputFile = new TFile("output.root", "RECREATE");
for (auto &pair : histogram_1)
{
    pair.second->Write();
}

for (auto &pair : histogram_2)
{
    pair.second->Write();
}

for (auto &pair : other_histograms)
{
    pair.second->Write();
}

outputFile->Close();
myFile->Close();
}
```



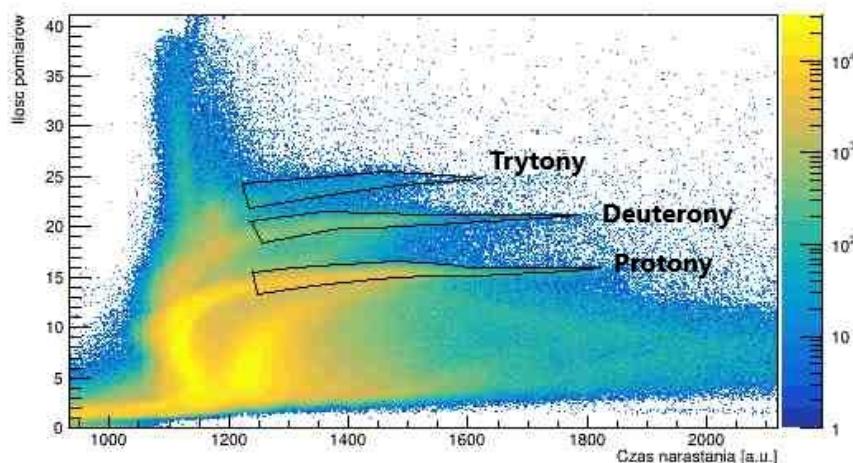
Rysunek 3.8: Macierz do identyfikowania cząstek naładowanych, przed i po kalibracji.

Rysunek 3.8 przedstawia porównanie macierzy identyfikacji lekkich cząstek naładowanych uzyskanych z detektora krzemowego. Macierz w górnym panelu została skonstruowana poprzez zsumowanie wszystkich macierzy z poszczególnych detektorów poza odrzuconymi, czyli od 26 do 29. Natomiast macierz w dolnym panelu stanowi sumę tych samych macierzy, tylko skalibrowanych z wykorzystaniem wyliczonych przez program współczynników kalibracyjnych. Widać wyraźnie, że po zastosowaniu kalibracji czasowej, macierze zgrupowały się w jedno strukturę.

Rozdział 4

Analiza danych

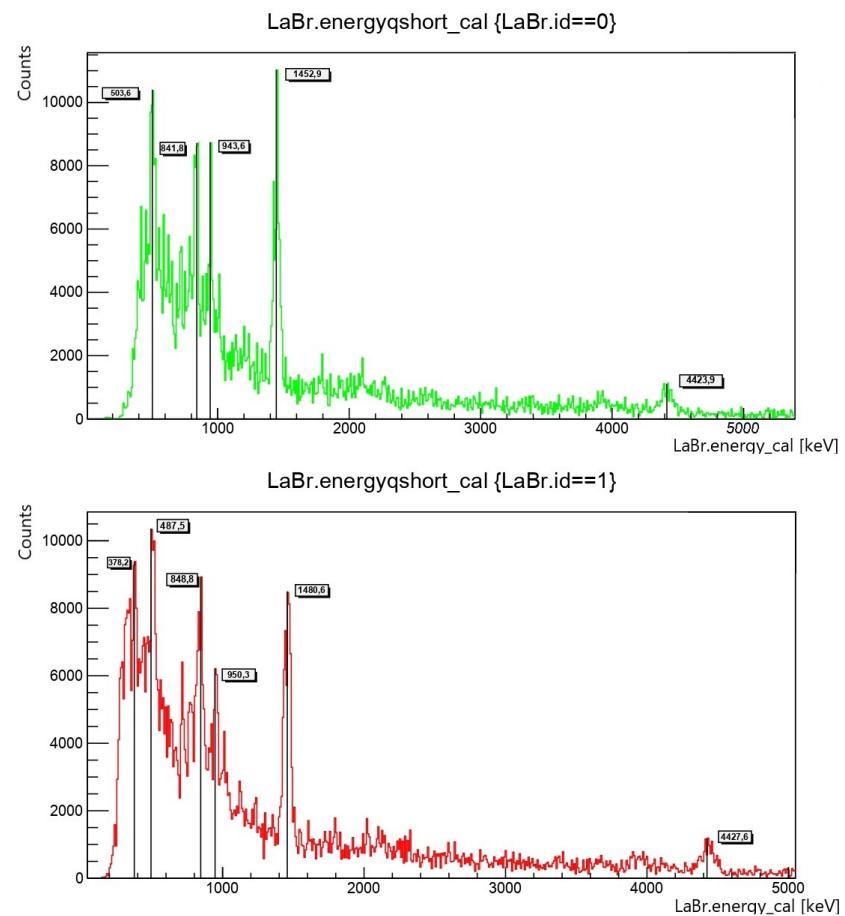
Po otrzymaniu jednolitej macierzy z prawie wszystkich detektorów, jesteśmy w stanie przy pomocy narzędzia „Graphical Cut” wyodrębnić pomiary cząstek naładowanych: protonów, deuteronów i trytonów, w ramach pokazania możliwości jakie daje zbudowana struktura danych, ponieważ wszystkie przedstawione wykresy oraz macierze, dzięki programowi zostały uzyskane przy pomocy jednej komendy.



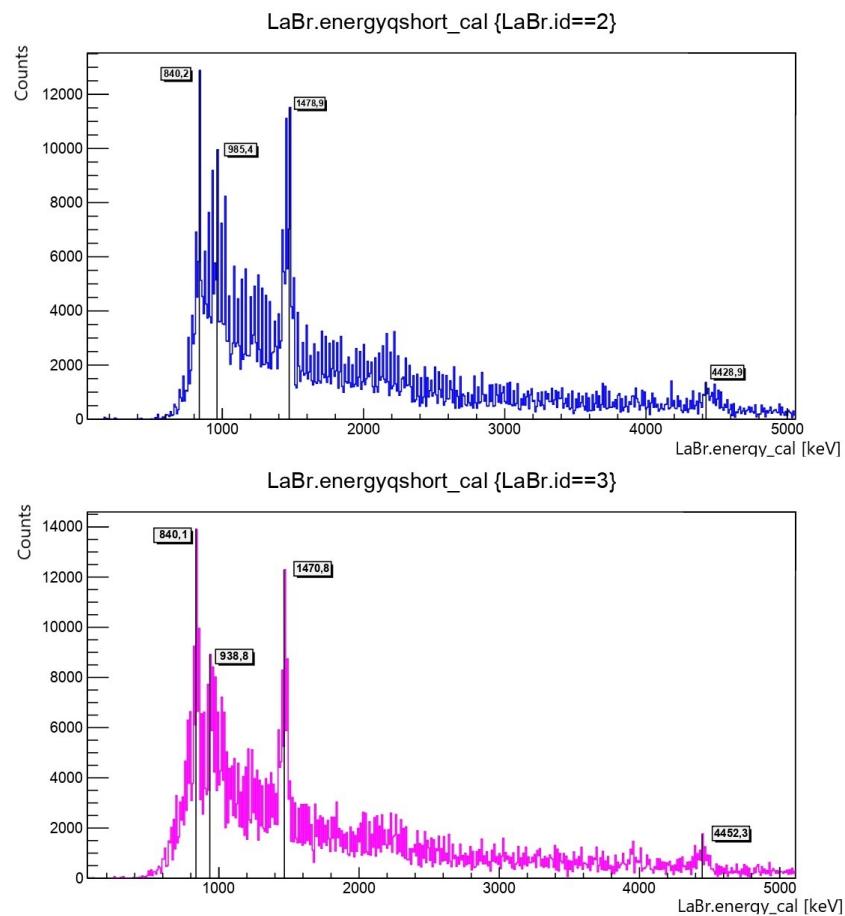
Rysunek 4.1: Wytworzzone cut'y na pomiarach cząstek naładowanych.

4.1 Wyniki eksperymentalne pomiarów promieniowania gamma

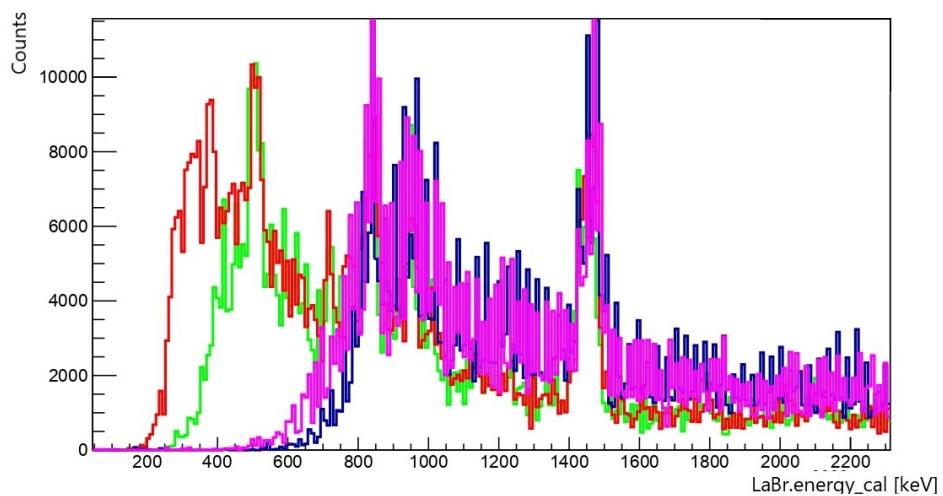
W widmach promieniowania gamma spodziewamy się zobaczyć pikи reprezentujące konkretne energie fotonów gamma emitowanych przez jądra atomowe, które zostały wzbudzone bezpośrednio w reakcji nieelestycznego rozpraszania protonów lub w wyniku emisji lekkich cząstek naładowanych z jąder wzbudzonych w tej reakcji. Zaobserwowano przejścia gamma przy ok. 511 keV, ok. 843,8 keV pochodzące z ^{27}Al , ok. 953,1 keV pochodzący z ^{12}B oraz ok. 1381 keV pochodzący z ^{40}K . Ślady przejść gamma z ^{27}Al obecne w widmie przy energii ok. 843,8 keV są spowodowane interakcjami z otaczającymi materiałami, takimi jak ramka tarczy. Widoczny jest również pik z ^{12}C przy energii 4438,9 keV.



Rysunek 4.2: Widmo promieniowania gamma zmierzonych dla poszczególnych detektorów LaBr.

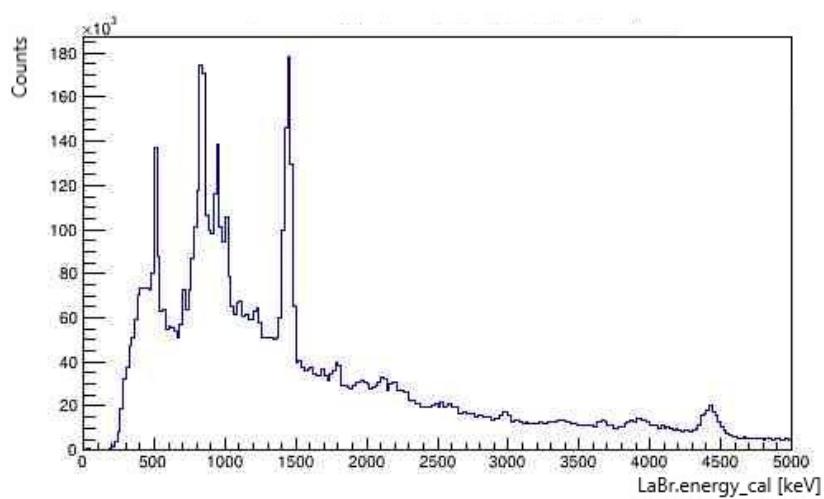


Rysunek 4.2: Widmo promieniowania gamma zmierzonych dla poszczególnych detektorów LaBr.

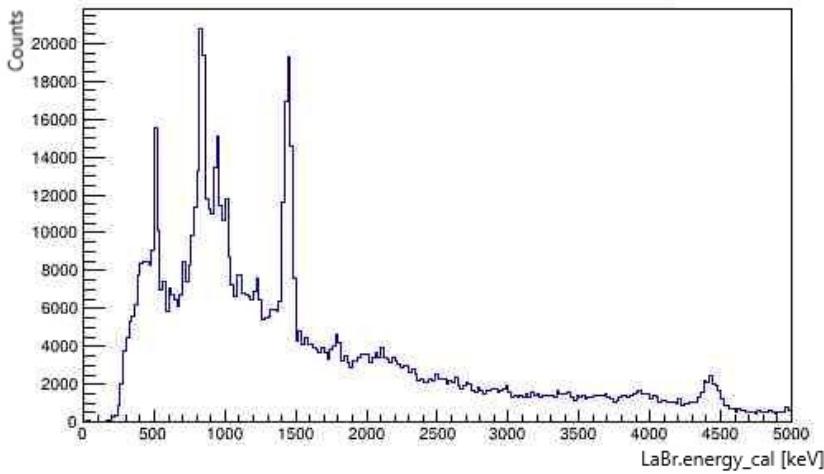


Rysunek 4.3: Porównanie widm przed wykorzystaniem pomiarów lekkich cząstek naładowanych z wszystkich detektorów LaBr.

Poprzez wykorzystanie cut'ów spodziewamy się bardziej wyrazistych pików oraz ograniczenie szumów pomiarowych.

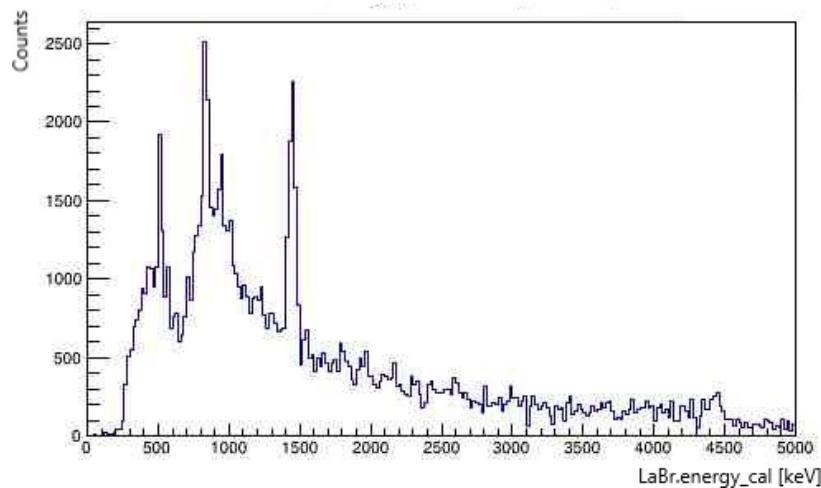


(a) Z wykorzystaniem pomiarów protonów.



(b) Z wykorzystaniem pomiarów deuteronów.

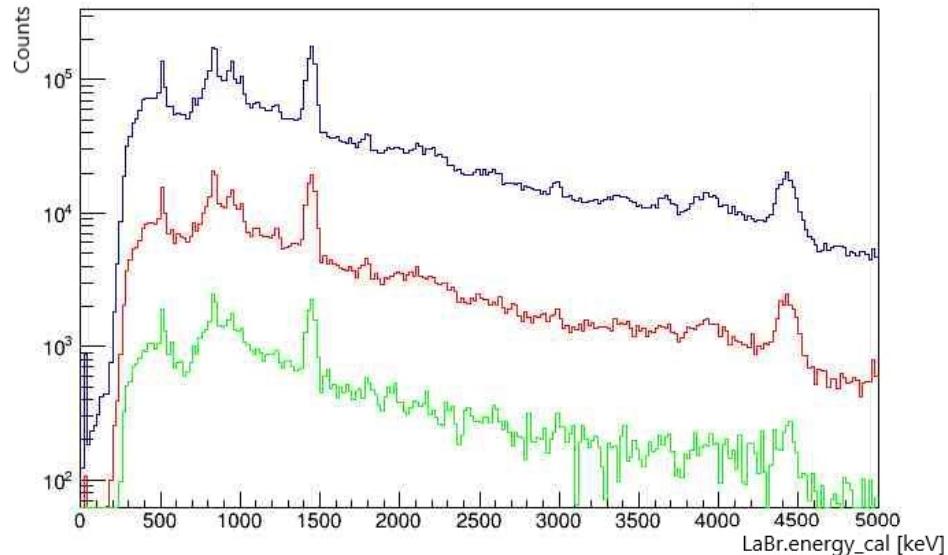
Rysunek 4.4: Widma promieniowania gamma LaBr z użyciem trzech cut'ów dla wszystkich detektorów



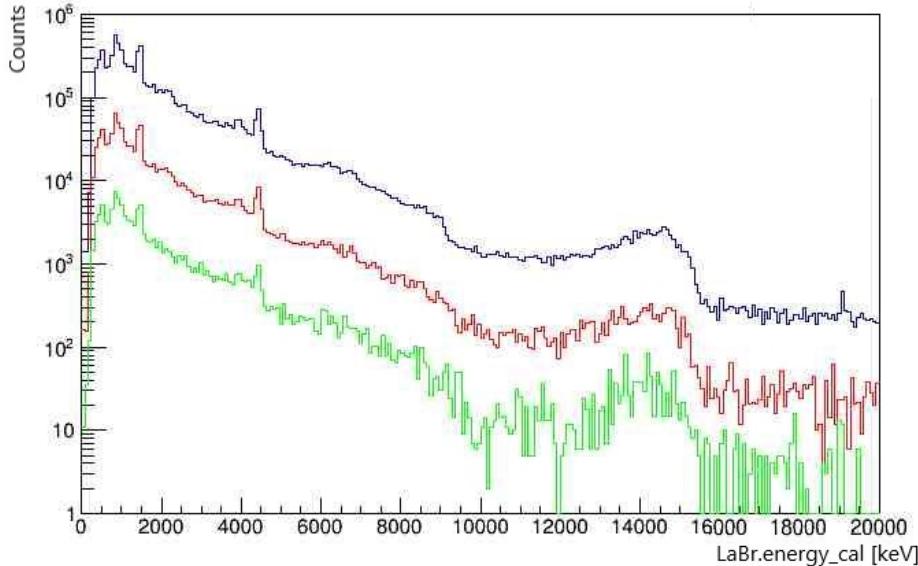
(a) Z wykorzystaniem pomiarów trytonów.

Rysunek 4.4: Widma promieniowania gamma LaBr z użyciem trzech cut'ów dla wszystkich detektorów

Po użyciu „Graphical cut”, piki widma stały się trochę wyraźniejsze, za wyjątkiem linii 953,1 keV, ^{12}B . Na poniższym rysunku możemy zauważyć, że pomimo różnic w statystyce między poszczególnymi pomiarami częstek naładowanych, widma są do siebie podobne.



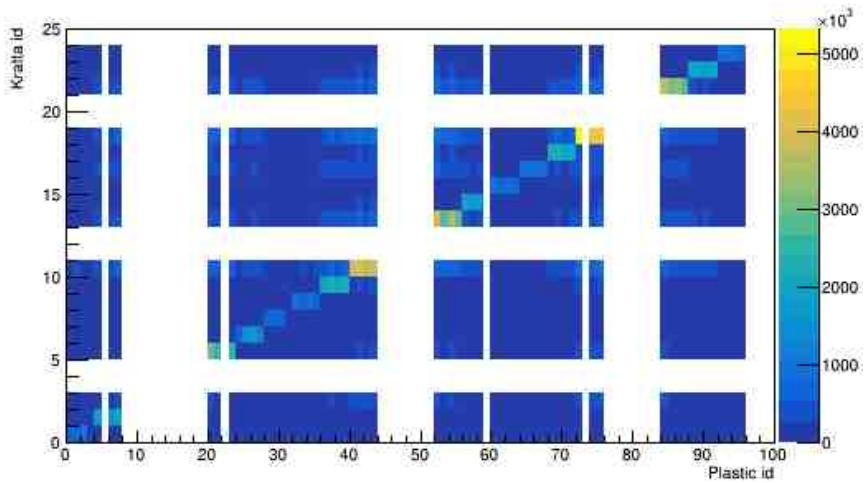
Rysunek 4.5: Porównanie widm promieniowania gamma dla wszystkich detektorów z uwzględnieniem pomiarów poszczególnych cząstek naładowanych (niebieski – proton, czerwony – deutery, zielony – tryty).



Rysunek 4.6: Porównanie widm promieniowania gamma dla wszystkich detektorów z cut'ami poszczególnych cząstek naładowanych, dla dalszych energii, aby zobaczyć rozcięgnięty pik przy ok. 15 MeV pochodzący z ^{12}C

4.2 Analiza korelacji pomiędzy detektorami oraz ich wpływ na dokładność pomiarów

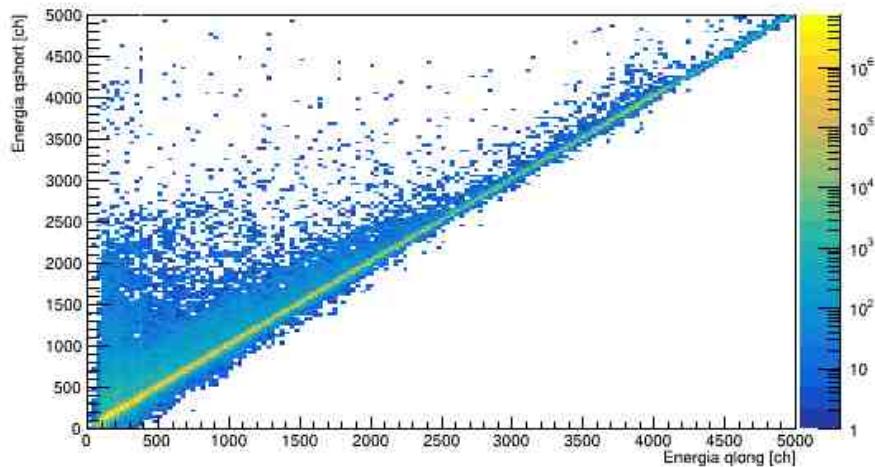
Przy pomocy skonstruowanego w ramach niniejszej pracy pliku zawierającego uporządkowaną strukturę drzewa ROOT można bardzo szybko sprawdzić poprawność połączeń detektorów w systemie pomiarowym. Na przykład: powinna istnieć odpowiednia zgodność pomiędzy detektorami KRATTA, a PLASTIC, które są ze sobą połączone w ten sposób, że na jeden moduł KRATTY przypada 4 detektory PLASTIC. Numeracja przebiega po kolej, czyli do detektora KRATTA o indeksie 0 połączone są detektory PLASTIC o indeksach od 0 do 4. Korzystając z rysunku 4.7 możemy sprawdzić, czy istnieje korelacja między detektorami, a zatem, czy detektory zostały połączone zgodnie z założeniem.



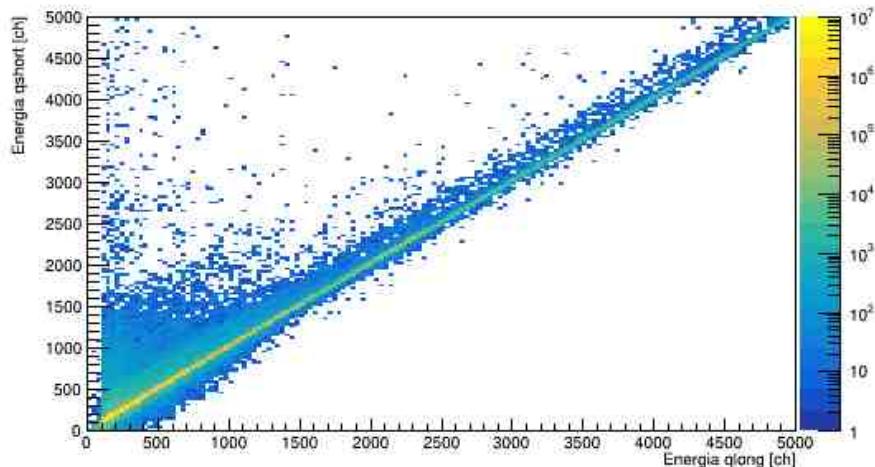
Rysunek 4.7: Wykres korelacji pracy pomiędzy detektorami KRATTA oraz PLASTIC.

Analiza rysunku pokazuje, że korelacje między detektorami KRATTA i PLASTIC są zgodne z oczekiwaniemi. Możemy zauważyc, że większość zliczeń przebiega w linii prostej zaczynającej się od zera, co stanowi potwierdzenie, że połączenia detektorów zostały wykonane prawidłowo. Wyjątki od tych wzorców mogą wskazywać na problemy z pojedynczymi detektorami, które przestały działać lub miały nieprawidłowe sygnały podczas eksperymentu.

W kolejnym przykładzie, na rysunku 4.8, pokazano energię uzyskaną z detektora LaBr przy zastosowaniu krótkiego i długiego okna czasowego integracji impulsu. Ponieważ detektor składa się z jednego kryształu scyntylacyjnego, widzimy na nim jedną linię. Sytuacja ta różni się w przypadku detektora PARIS.



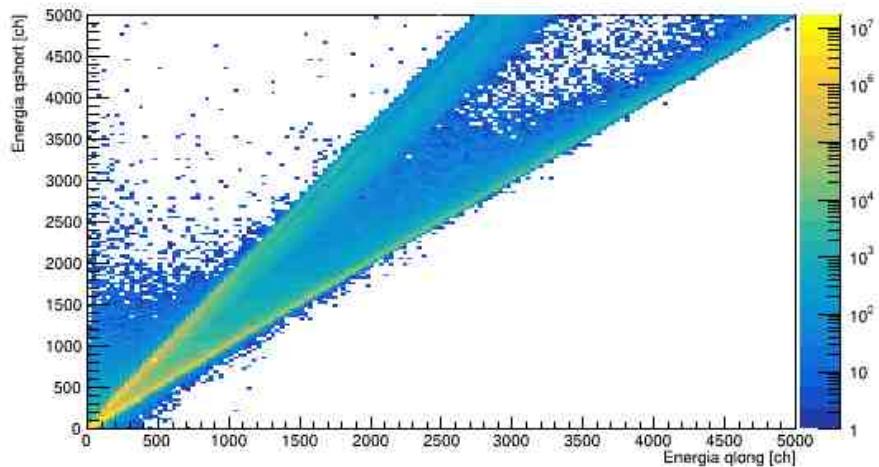
(a) Detektor o indeksie 0.



(b) Detektor o indeksie 2.

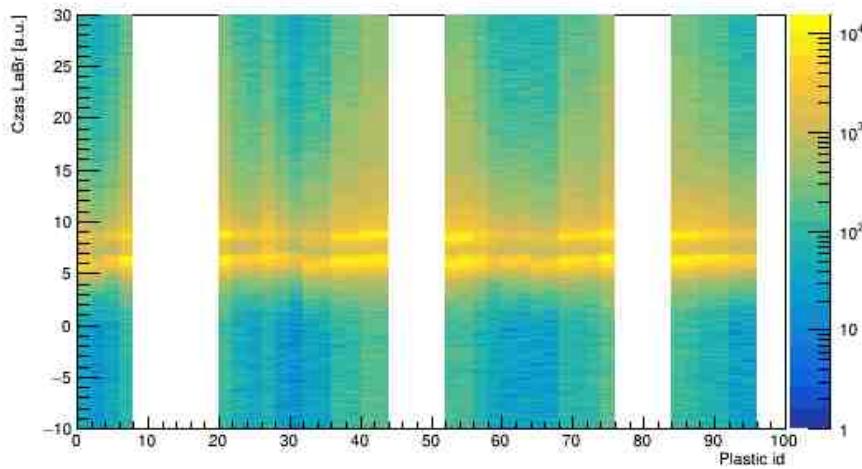
Rysunek 4.8: Wykres dla detektorów LaBr o indeksach 0 oraz 2, energii Q short z energią Q long.

Wykreślając energię z detektora PARIS uzyskaną z krótkim i długim oknem czasu integracji, można rozróżnić dwie linie, o jedną więcej niż w przypadku detektorów LaBr. Jest to ściśle związane z budową tego detektora. Składa się on z dwóch kryształów CeBr_3 oraz NaI , a ponieważ ich czas pomiaru jest rzędu odpowiednio $\sim 10 \text{ ns}$ i $\sim 250 \text{ ns}$, wartości energii odczytywanej przez oba kryształy będą przebiegać głównie wzdłuż dwóch oddzielnych linii prostych o różnym nachyleniu.

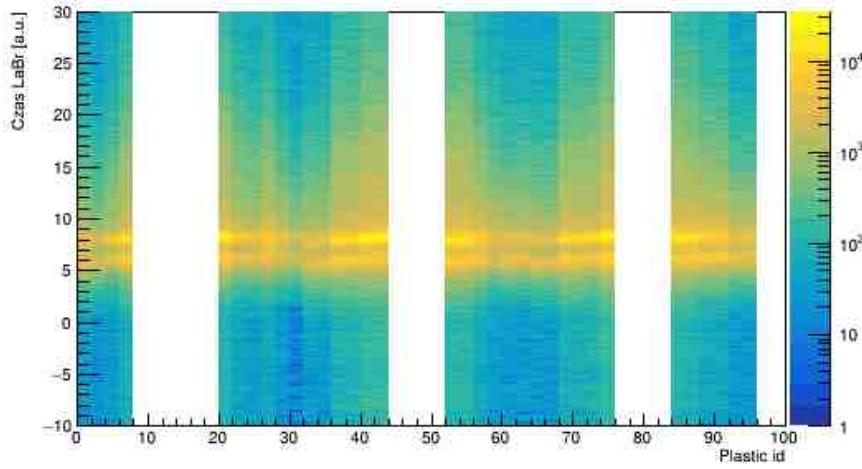


Rysunek 4.9: Wykres dla detektora PARIS energii Q short z energią Q long.

Kolejny przykład stanowi korelacja czasowa detektorów LaBr z detektorami PLASTIC, widoczna na rysunku 4.10. Dzięki temu rysunkowi można szybko ocenić, czy wyrównanie osi czasu, aby wykluczyć fałszywe zdarzenia indukowane przez wiązkę na otaczających materiałach, będzie konieczne w analizie danych eksperymentalnych. Można zauważyc, że kształtują się dwie linie ciągłe oznaczające zmierzony czas z detektorów LaBr względem detektorów PLASTIC, ale z sporymi niezgodnościami, dlatego w celu poprawy jakości danych należałoby wyrównać zmierzzone wartości czasu względem jednej pary detektorów LaBr-PLASTIC i ustawić wąską bramkę czasową obejmującą jedynie zdarzenia wynikające z reakcji wiązki na tarczy.



(a) Detektor LaBr o indeksie 0.

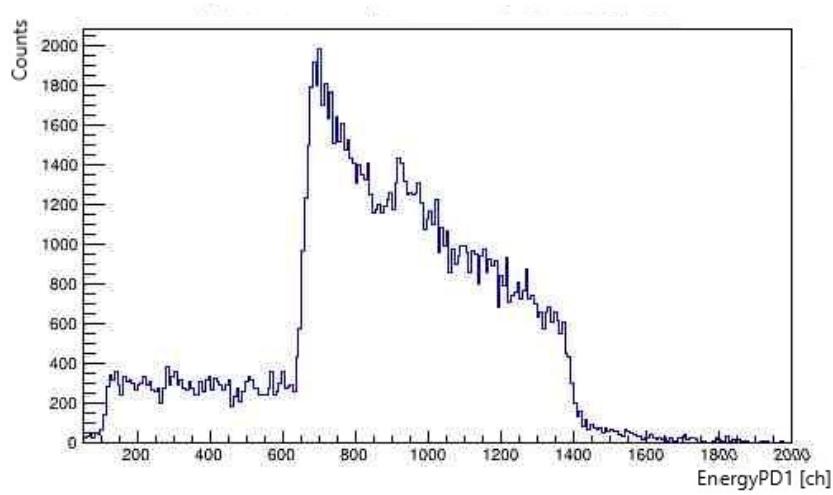


(b) Detektor LaBr o indeksie 2.

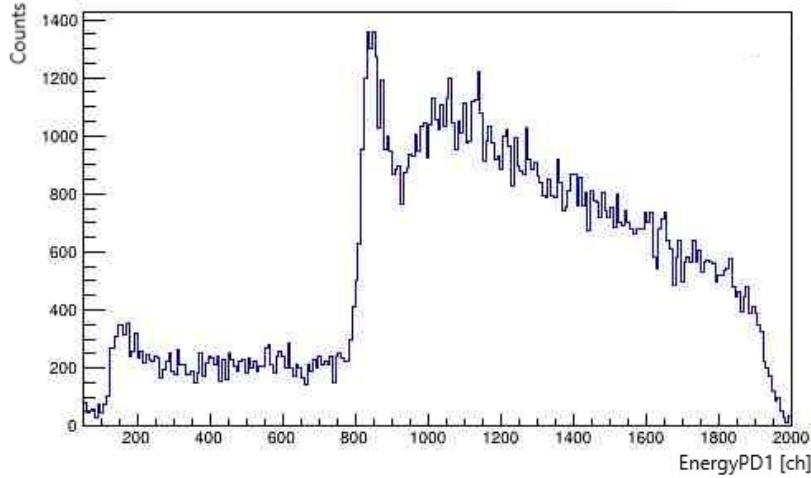
Rysunek 4.10: Histogram 2D czasu LaBr₃ i ID detektorów PLASTIC. Puste przestrzenie są powiązane z brakującymi detektorami PLASTIC, niedziałającymi z powodu awarii modułu KRATTA lub z grupami czterech detektorów powiązanych z modułem KRATTA.

4.3 Pomiar rozproszonych protonów

Program umożliwia analizę pomiarów nieelastycznie rozproszonych protonów wiązki, a badanie ich widma energii dostarcza nowych informacji na temat stanów wzbudzonych populowanych w jądrze tarczy, w tym przypadku w ^{13}C .

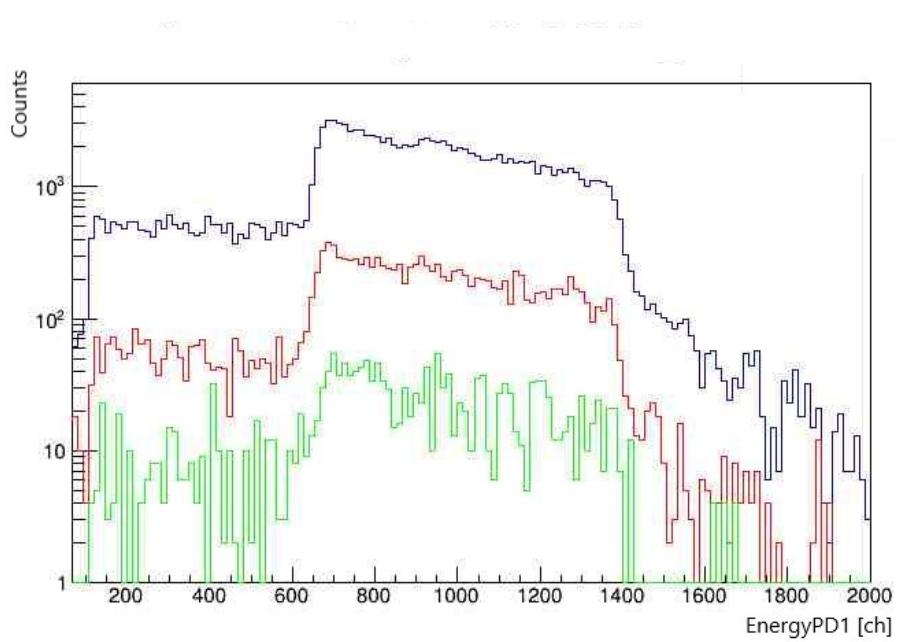


(a) Detektor o indeksie 8

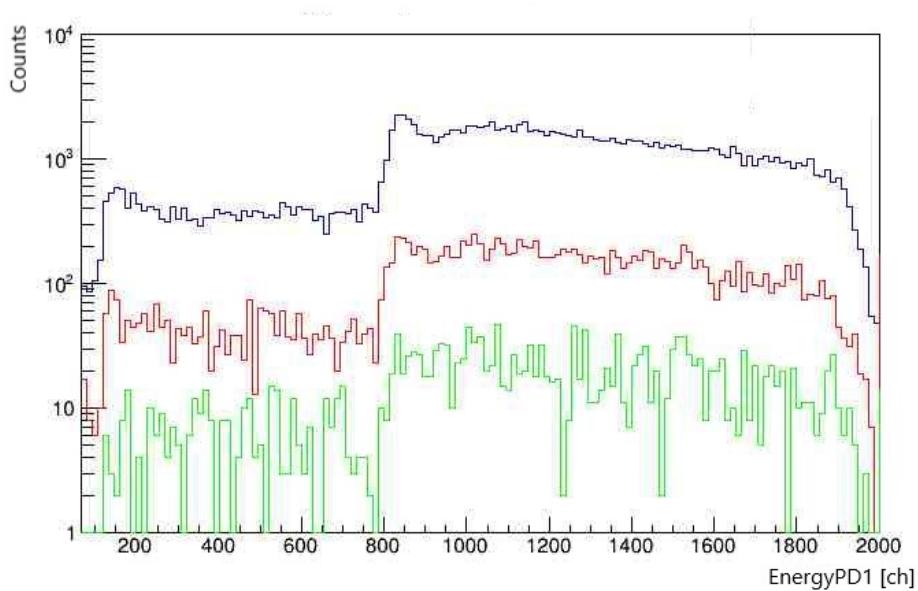


(b) Detektor o indeksie 15

Rysunek 4.11: Nieskalibrowana energia PD1 z detektorów KRATTA o indeksach 8 oraz 15 z użyciem cut'a pomiarów protonów z macierzy cząstek naładowanych pochodzących z detektora DSSSD.



(a) Detektor o indeksie 8

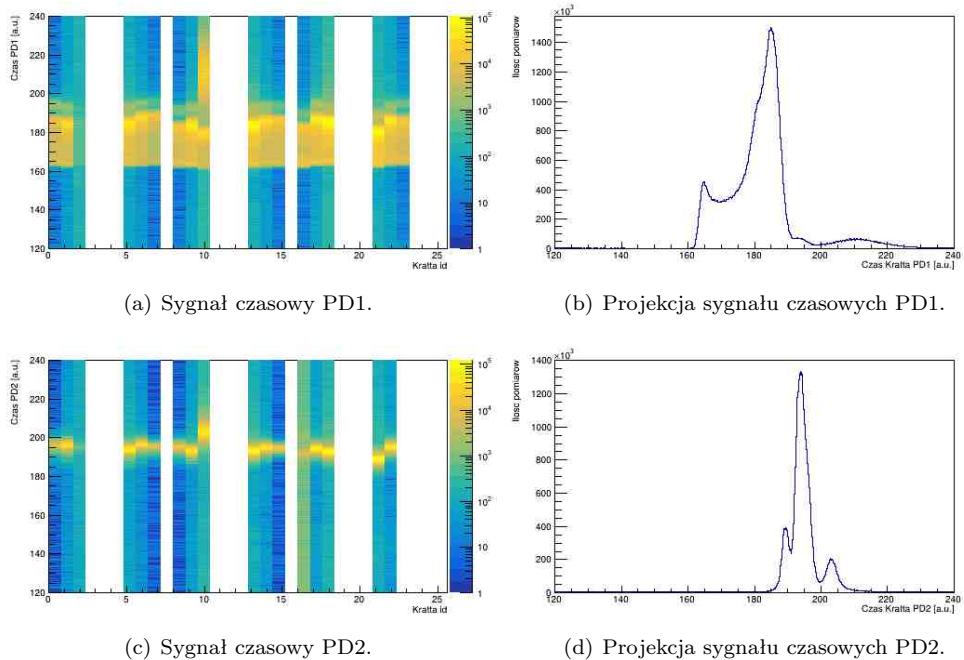


(b) Detektor o indeksie 15

Rysunek 4.12: Porównanie energii PD1 dla detektorów KRATTA o indeksach 8 i 15 z uwzględnieniem cut'ów z macierzy cząstek naładowanych pochodzących z detektora DSSSD, poszczególnych cząstek naładowanych (niebieski – proton, czerwony – deutery, zielony – tryty).

Projekcje sygnałów czasowych PD1 i PD2 powiązanych z modułem KRATTA przedstawione są na rysunku 4.13. Można zauważyć, że w niektórych modułach brakowało sygnałów czasowych, podczas gdy inne prezentowały złe wzorce sygnałów. Detektory te prawdopodobnie przestały działać w czasie eksperymentu. Dzięki sygnałom czasowym PD1 i PD2, podczas poprzednich eksperymentów, zastosowano bramki czasowe, aby wykluczyć zdarzenia spoza najbardziej inten-

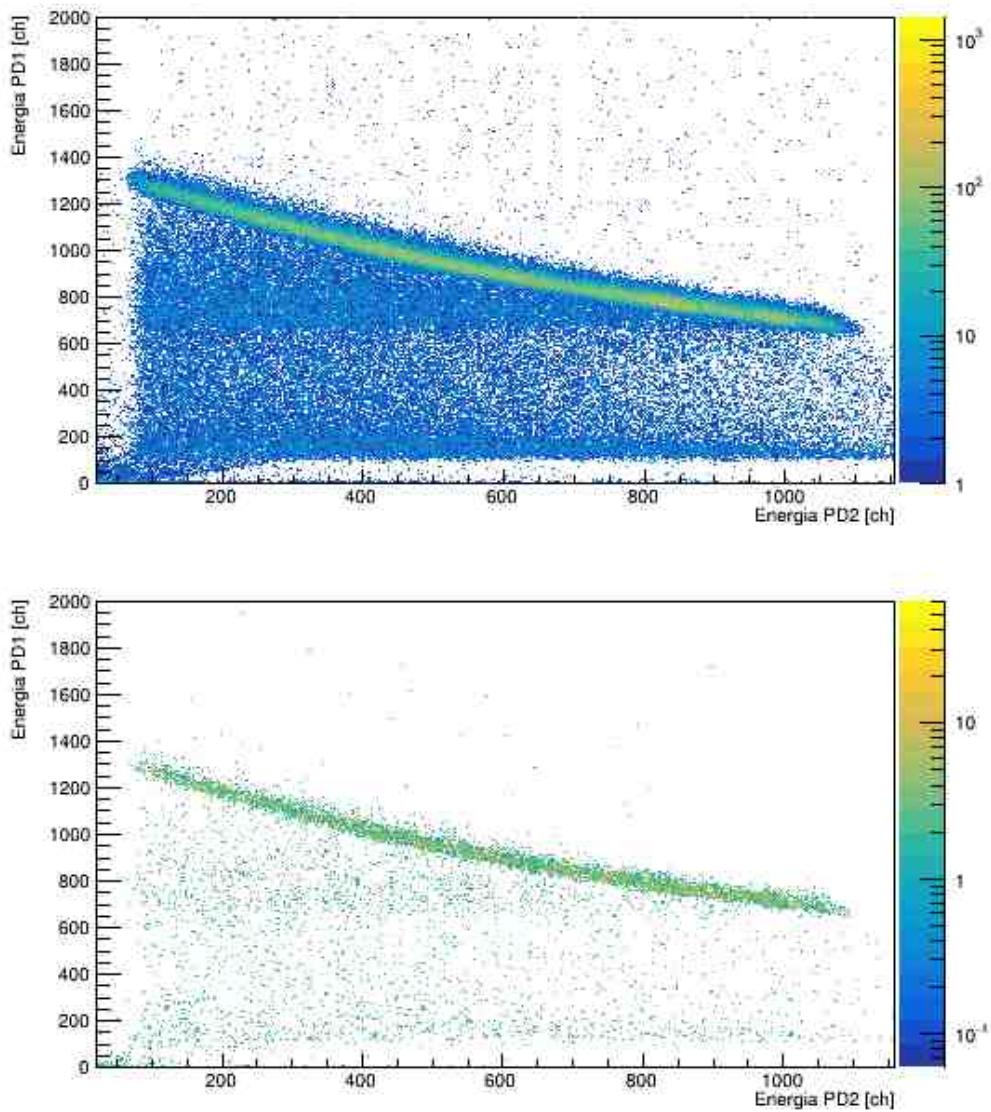
sywnego piku, związane z rozproszonymi protonami na materiale docelowym.



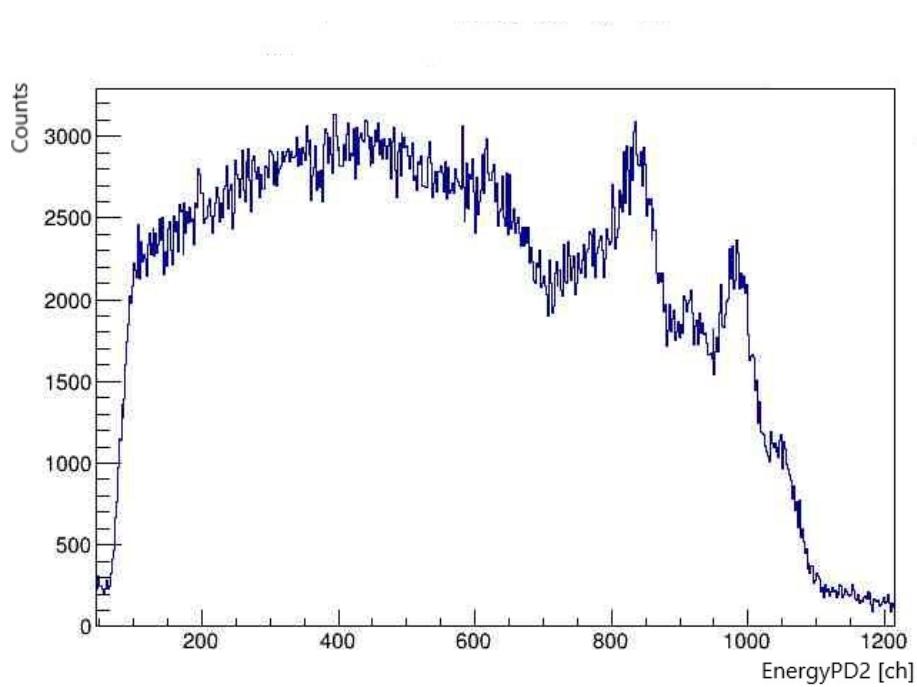
Rysunek 4.13: Sygnały czasowe PD1 i PD2 powiązane z modułem KRATTA oraz projekcje tych wykresów na oś X.

Dzięki tym bramkom możliwe było odfiltrowanie niepożądanych zdarzeń, co poprawiło jakość zbieranych danych i pozwoliło na bardziej precyzyjną analizę widm energii. Dzięki temu możliwe było uzyskanie bardziej wiarygodnych wyników dotyczących rozpraszania protonów.

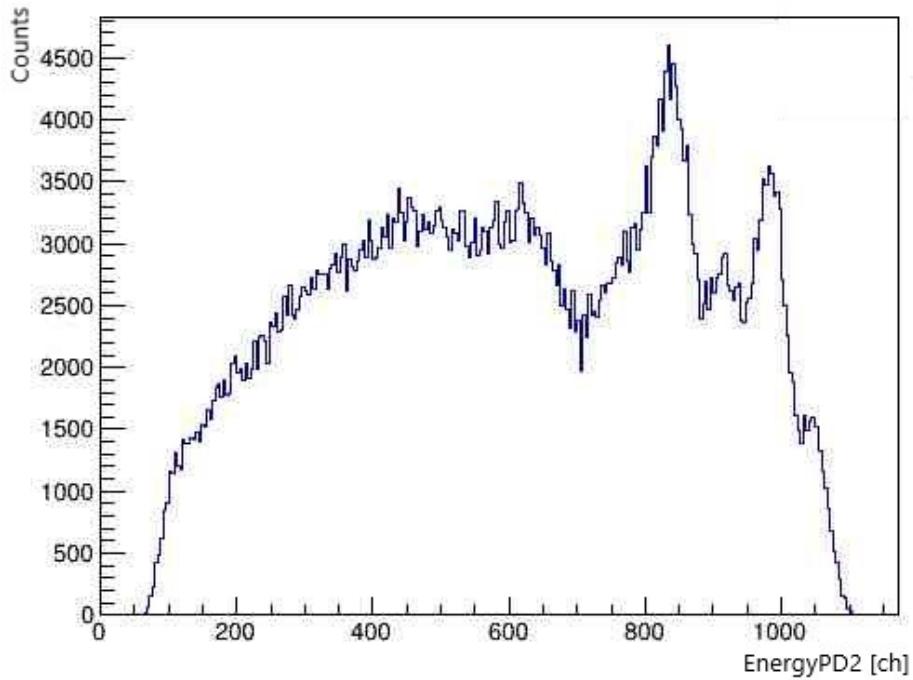
Linia protonów zarejestrowana przez detektor KRATTA, na której cut, czyli wyodrębnienie punktów leżących jedynie na tej linii, może być wykorzystane do dalszych analiz energii.



Rysunek 4.14: Nieskalibrowaną macierz energii PD1 vs. PD2 dla modułu KRATTA o indeksie 8. Najbardziej widoczna linia pomiarów, reprezentuje pomiary zebranej energii rozproszonych protonów, na które nakładamy ograniczenie (cut).

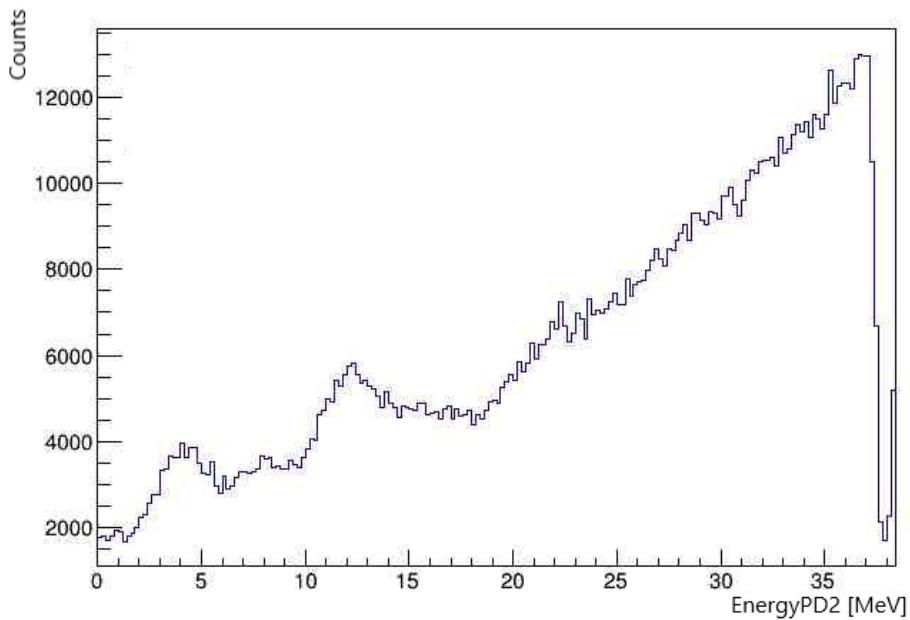


(a) Przed zastosowaniem pomiarów protonów.

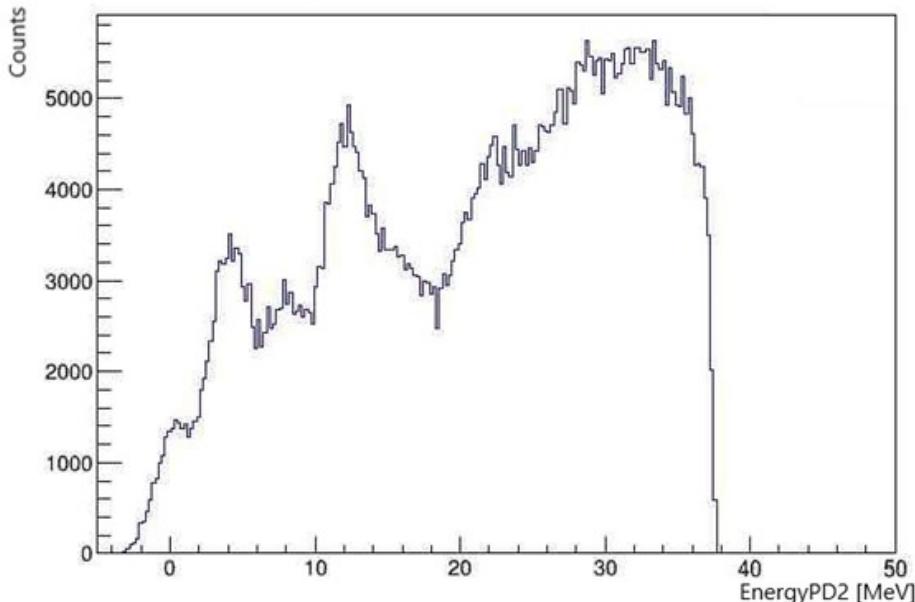


(b) Po zastosowaniu pomiarów protonów.

Rysunek 4.15: Nieskalibrowane widma energii PD2 dla modułu KRATTA o indeksie 8, przed i po zastosowaniu cut'u na protonach. Widoczne jest lepsze wyodrębnienie pików od tła po zastosowaniu cut'u.

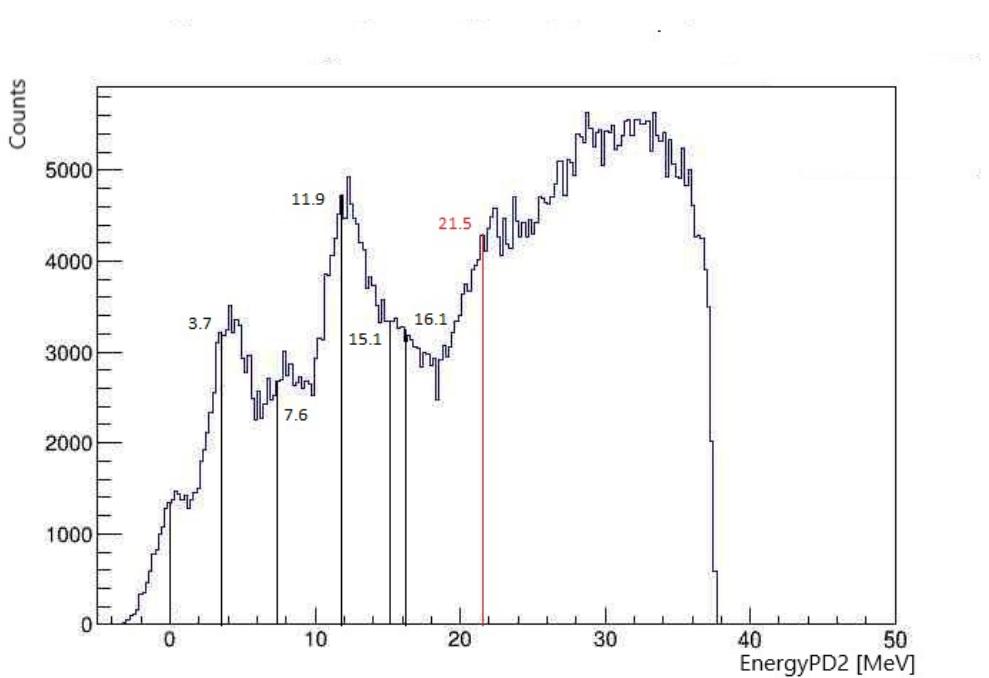


(a) Przed zastosowaniem pomiarów protonów.



(b) Po zastosowaniu pomiarów protonów.

Rysunek 4.16: Skalibrowane widma energii PD2 dla modułu KRATTA o indeksie 8, gdzie można odczytać energie wzbudzenia jądra tarczy, tj. ^{13}C (przed i po zastosowaniu cut'u). Można zaobserwować, że piki związane z populacją stanów wzbudzonych w ^{13}C są bardziej wyraźne po zastosowaniu cut'u.



Rysunek 4.17: Widmo energii wzbudzenia ^{13}C zmierzone przez detektory KRATTA pod katem $\sim 36^\circ$ do osi wiązki protonów. Zaznaczone piki odpowiadają poziomom energetycznym obserwowanym dotychczas w jądrze ^{13}C (na podstawie bazy NNDC [11]). Kolorem czerwonym zaznaczono stan rozciagnięty przy energii 21.5 MeV w ^{13}C .

Na podstawie zmierzonej energii rozproszonych protonów skonstruowano widmo energii wzbudzenia jądra ^{13}C (rysunek 4.17). Można na nim zaobserwować szereg pików. Pierwszy z nich, przy energii 0 MeV, świadczy o zachodzeniu reakcji elastycznego rozproszenia protonów, w wyniku której jądro ^{13}C pozostaje w stanie podstawowym. Pozostałe piki odpowiadają energiom stanów wzbudzonych w ^{13}C , tj. 3.7, 7.6, 11.9, 15.1 i 16.1 MeV, populowanych w reakcji nieelastycznego rozproszenia protonów. W szczególności pik oznaczony kolorem czerwonym na rysunku 4.17, przy energii 21.5 MeV, świadczy o wzbudzaniu stanu rozciagniętego M4 w przeprowadzonym eksperymencie.

Rozdział 5

Podsumowanie

Praca porusza zagadnienia związane z badaniami rozpadu rozcięgniętych stanów M4 w ^{13}C . W przypadku tego jądra najsilniejszy taki stan jest zlokalizowany przy energii 21.47 MeV. Stan będący przedmiotem zainteresowania został wzbudzony w reakcji nieelastycznego rozpraszania wiązki protonów w eksperymencie zrealizowanym w krakowskim Instytucie Fizyki Jądrowej w Centrum Cyklotronowym Bronowice. Nieelastycznie rozproszone protony były mierzone w układzie detektorów KRATTA, podczas gdy promieniowanie gamma było wykrywane w czterech scyntylatorach LaBr_3 o dużej objętości oraz w klastrach spektrometru PARIS. Dodatkowo, lekkie cząstki naładowane pochodzące z rozpadów stanów wzbudzonych populowanych w reakcji były mierzone przez detektory DSSSD.

Głównym celem pracy było napisanie programu porządkującego zebrane podczas eksperymentu dane w strukturę tzw. „drzewa”. Ponadto w ramach pracy stworzono też program do kalibracji macierzy pozyskanych przy pomiarze cząstek naładowanych. Do tego celu wykorzystano środowisko do analizy danych ROOT, a same programy napisane zostały w języku C++. Program organizujący dane w strukturę drzewa ROOT oferuje możliwość wczytania współczynników kalibracyjnych mierzonych wielkości i wyświetlenie skalibrowanych widm energii, co zostało pokazane w rozdziale 3. Kalibracja czasowa DSSSD pozwoliła wyodrębnić pomiary lekkich cząstek naładowanych: protonów, deuterów, trytów, co pozwoliło nie tylko ocenić jakość wytworznej struktury danych, czy działa prawidłowo i w jaki stopniu ułatwiało analizę pomiarową, ale także polepszyć interpretację uzyskanych wykresów, przez m.in. pozbycia się szumów pomiarowych, co także jest przedstawione w rozdziale 3.

Dodatkowo, dzięki zastosowaniu programu można szybko wyświetlić informacje o zgodności połączeń detektorów KRATTA oraz PLASTIC, korelacji pomiarów czasów przez detektory LaBr i działania detektorów PLASTIC. Ta możliwość została zademonstrowana na przykładzie macierzy 4.10, w rozdziale 4.

Podsumowując, praca pozwoliła wytworzyć narzędzie, które może zostać wykorzystane do badań rozpadów rozcięgniętych stanów M4 w ^{13}C oraz innych lekkich jądrach. Praca nie tylko osiągnęła główny cel tj. opracowanie efektywnych narzędzi analizy danych, ale także przyczyniła się do pozyskania wartościowego zestawu danych poprzez skalibrowanie detektora. Wykorzystanie napisanych programów pozwoli na bardziej efektywną analizę zebranych danych, co przełoży się na lepsze zrozumienie badanych procesów oraz precyzyjne interpretacje wyników. Ostatecznie, wyniki tej pracy, poszerzają możliwości analizy danych w celu badania rozpadów jądrowych.

Dodatek A

Wydruk pliku nagłówkowego dla detektora krzemowego DSSSD, tu zdefiniowany jako Silicon, plik: Silicon.h

```
#ifndef Silicon_h
#define Silicon_h
#include "TObject.h"
#include <vector>
#include <iostream>

#define BRANCH Silicon
#define DETECTOR SiliconDetector

class DETECTOR{
public:
    Int_t id;                      // Detector number
    Float_t time30;                 // Signal time 30 %
    Float_t time80;                 // Signal time 80 %
    Float_t ampl;                   // Signal amplitude

    bool ok(){ return
        time30 > 0;
    }

    Float_t deltaT(){return time30 - time80;}
};

class BRANCH: public TObject{

public:
    Float_ttpls;
    std::vector<DETECTOR> det;
```

```

Int_t mult() const{ // Multiplicity
    return det.size();
}

void Reset(){
    tpls_i = 0;
    det.clear();
}

void Add(DETECTOR & p){
    det.push_back(p);
}

BRANCH(){}
~BRANCH(){}

ClassDef(BRANCH, 2)
;

#define BRANCH
#define DETECTOR

#endif

```

Dodatek B

Wydruk pliku nagłówkowego dla detektora Kratta, plik: Kratta.h

```
#ifndef Kratta_h
#define Kratta_h
#include "TObject.h"
#include <vector>
#include <iostream>

#define WAVEFORMS

#define BRANCH Kratta
#define DETECTOR KrattaDetector

class PDO{
public:
    Float_t time;      // Signal time in ns
    Float_t ampl;      // Signal amplitude
    Float_t pdst;      // Signal pedestal

#ifdef WAVEFORMS

    Short_t form[512]; // Full waveform

#endif

// Build-in functions

bool ok() const{
    return time > 0;
}

bool intime() const{
    return (time > 1040 && time < 1100); // trig = 2
}

};

class PD1{
public:
```

```

    Float_t time;      // Signal time in ns
    Float_t ampl;      // Signal amplitude
    Float_t pdst;      // Signal pedestal

#define WAVEFORMS

    Short_t form[1024]; // Full waveform

#endif

// Build-in functions

bool ok() const{
    return time > 0;
}

bool intime() const{
    return (time > 1800 && time < 2000); // trig = 2
}
};

class PD2{
public:
    Float_t time;      // Signal time in ns
    Float_t ampl;      // Signal amplitude
    Float_t pdst;      // Signal pedestal

#define WAVEFORMS

    Short_t form[1024]; // Full waveform

#endif

// Build-in functions

bool ok() const{
    return time > 0;
}

bool intime() const{
    return (time > 1950 && time < 2050); // trig = 2
}
};

class DETECTOR{
public:
    Int_t id;          // Detector number
    PDO pd0;          // PDO
    PD1 pd1;          // PD1
    PD2 pd2;          // PD2

// Build-in functions

Int_t x() const{ // x position

```

```

        return id%8;
    }

Int_t y() const{ // y position
    return id/8;
}
Int_t plastic(Int_t i) const{ // id of i-th plastic (i = 0..3)
    return id*4 + i;
}

bool ok(){
    return pd0.ok() || pd1.ok() || pd2.ok();
}

bool pd0pd1() const{
    return pd0.intime() && pd1.intime();
}

bool pd1pd2() const{
    return pd1.intime() && pd2.intime();
};

class BRANCH: public TObject{
public:

std::vector<DETECTOR> det;

Int_t mult() const{ // Multiplicity
    return det.size();
}

void Reset(){
    det.clear();
}

void Add(DETECTOR & p){
    det.push_back(p);
}

BRANCH(){}
~BRANCH(){}

ClassDef(BRANCH, 1)
;

#define BRANCH
#define DETECTOR

#endif

```

Dodatek C

Wydruk pliku nagłówkowego dla detektora Plastic, plik: Plastic.h

```
#ifndef Plastic_h
#define Plastic_h
#include "TObject.h"
#include <vector>
#include <iostream>

#define BRANCH Plastic
#define DETECTOR PlasticDetector

class DETECTOR{
public:
    Int_t id;           // Detector number
    Float_t time;       // Signal time in ns
    Float_t ampl;        // Signal amplitude
    Float_t pdst;        // Signal pedestal
    Short_t form[400];   // Full waveform

    Int_t kratta() const{           // id of the kratta module
        return id/4;
    }

    bool ok() const{
        return time > 0;
    }

    bool intime() const{
        return time > 620 && time < 720;
    }
};

class BRANCH: public TObject{

public:
    std::vector<DETECTOR> det;
```

```
Int_t mult() const{ // Multiplicity
    return det.size();
}

void Reset(){
    det.clear();
}

void Add(DETECTOR & p){
    det.push_back(p);
}

BRANCH(){}
~BRANCH(){}

ClassDef(BRANCH, 1)
;

#ifndef BRANCH
#ifndef DETECTOR

#endif

```

Dodatek D

Wydruk pliku nagłówkowego dla detektora LaBr, plik: LaBr.h

```
#ifndef LaBr_h
#define LaBr_h
#include "TObject.h"
#include "Paris.h"
#include <vector>
#include <iostream>

#define WAVEFORMS

#define BRANCH LaBr
#define DETECTOR LaBrDetector

class DETECTOR{
public:
    Int_t id;          // Detector number
    Float_t time;      // Signal time in ns
    Int_t qshort; // Q integral in short gate
    Int_t qlong; // Q integral in long gate
#ifdef WAVEFORMS
    Wave wave; // Full waveform
#endif

    bool ok(){
        return time > 0;
    }

    Int_t x(){          // x position
        return id/2;
    }

    Int_t y(){          // y position
        return -id%2;
    }
};
```

```

class BRANCH: public TObject{
public:
    static const int Nmod = 2;
    std::vector<DETECTOR> det;

Int_t mult() const{ // Multiplicity
    return det.size();
}

void Reset(){
    det.clear();
}

void Add(DETECTOR & p){
    det.push_back(p);
}

BRANCH(){}
~BRANCH(){}

ClassDef(BRANCH, 1)
};

#define BRANCH
#define DETECTOR

#endif

```

Dodatek E

Wydruk programu do tworzenia struktury danych „drzewa ROOT”, plik: rewrite_tree.C

```
#include "TFile.h"
#include "TH2F.h"
#include "TRandom.h"
#include "TTree.h"
#include "TSpectrum.h"
#include "TCanvas.h"
#include "TLine.h"
#include "TCutG.h"
#include "TProfile.h"
#include <iostream>
#include <fstream>
#include <iostream>
#include <sstream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "TH3.h"
#include "TH2.h"
#include "TH1.h"
#include <TROOT.h>
#include <stdint.h>
#include "TMath.h"
#include <algorithm>
#include <string>
#include <vector>
#include <cstdlib>
#include "Kratta.h"
#include "Plastic.h"
#include "Paris.h"
#include "LaBr.h"
#include "Silicon.h"

void rewrite_tree(TTree * tree){
```

```

//-----
// otwieranie pliku istniejacego, podwiazwanie drzewa i galezi
//-----

    struct {
        int nrun;
        int id;
        int trig;
        int mult;
    } nfo;

Plastic * plastic = new Plastic;
Kratta * kratta = new Kratta;
Paris * paris = new Paris;
LaBr * labr = new LaBr;
Silicon * silicon = new Silicon;

tree->SetBranchAddress("paris", &paris);
tree->SetBranchAddress("labr", &labr);
tree->SetBranchAddress("plastic", &plastic);
tree->SetBranchAddress("kratta", &kratta);
tree->SetBranchAddress("silicon", &silicon);
tree->SetBranchAddress("info", &nfo);

//-----
// tworzenie nowego pliku, jego drzewa i galezi
//-----

TFile *f1 = new TFile("13C_data.root","recreate");
TTree *T1 = new TTree("tree2","tree2");

    struct
    {
        int id;
        float time;
        float energyqshort;
        float energyqlong;
        float energyqshort_cal;
    }
    LaBr;

    struct
    {
        int id;
        float time;
    }
    Plastic;

    struct
    {
        int id;

```

```

        float time;
        float energyqlong;
        float energyqshort;
    }
Paris;

struct
{
    int id;
    float energyPDO;
    float energyPD1;
    float energyPD2;
    float timePDO;
    float timePD1;
    float timePD2;
    float energyPD2_cal;
}
Kratta;

struct
{
    int id;
    float energy;
    float energy_cal;
    float time30;
    float time80;
    float difftime;
    float difftime_cal;
}
Silicon;

T1->Branch("LaBr",&LaBr,"id/I:time/F:energyqshort
/F:energyqlong/F:energyqshort_cal/F");

T1->Branch("Plastic",&Plastic,"id/I:time/F");

T1->Branch("Paris",&Paris,"id/I:time/F:energyqlong
/F:energyqshort/F");

T1->Branch("Kratta",&Kratta,"id/I:energyPDO/F:energyPD1
/F:energyPD2/F:timePDO/F:timePD1/F:timePD2/F:energyPD2_cal/F");

T1->Branch("Silicon",&Silicon,"id/I:energy/F:energy_cal
/F:time30/F:time80/F:difftime/F:difftime_cal/F");

T1->Branch("Info",&nfo,"run/I:id/I:trig/I:mult/I");

//T1->Branch("dowolna_nazwa",&wskaznik_do_tego_co_ma_byc_galezia,
//"lisc1/I:lisc2/D:lisc3/D");

//-----

```

```

// wczytanie wspolczynnikow kalibacyjnych do energii Silicon
//-----

std::vector<double> sil_cal0, sil_cal1;

cout << "cal_silicon" << endl;

FILE *plik;
plik = fopen("calibrations/cal_silicon.txt", "r");
if (plik == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

double sil1[32], sil2[32];
for (int i = 0; i < 32; i++) {
    if (fscanf(plik, "%lf %lf", &sil1[i], &sil2[i]) != 2) {
        printf("Blad wczytywania danych z wiersza %d.\n", i+1);
        sil1[i] = 0;
        sil2[i] = 1;
    }
    if (std::isnan(sil1[i]) || std::isnan(sil2[i])) {
        printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
        sil1[i] = 0;
        sil2[i] = 1;
    }
    sil_cal0.push_back(sil1[i]);
    sil_cal1.push_back(sil2[i]);
}

for (int i = 0; i < sil_cal0.size(); i++)
    cout << sil_cal0[i] << "\t" << sil_cal1[i] << endl;
cout << " " << endl;
fclose(plik);

//-----
// wczytanie wspolczynnikow kalibacyjnych do energii Kratta
//-----


std::vector<double> kratta_cal0_I, kratta_cal1_I, kratta_cal2_I;
std::vector<double> kratta_cal0_II, kratta_cal1_II, kratta_cal2_II;
double kra11[24], kra12[24], kra13[24];
double kra21[24], kra22[24], kra23[24];

cout << "cal_kratta_I" << endl;

FILE *plik2;
plik2 = fopen("calibrations/cal_kratta_I.txt", "r");
if (plik2 == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

for (int i = 0; i < 24; i++) {
    if (fscanf(plik2, "%lf %lf %lf", &kra11[i], &kra12[i],

```

```

    &kra13[i]) != 3) {
    printf("Blad wczytywania danych z wiersza %d.\n", i+1);
    kra11[i] = 0;
    kra12[i] = 1;
    kra13[i] = 0;
}
if (std::isnan(kra11[i]) || std::isnan(kra12[i])
    || std::isnan(kra13[i])) {
    printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
    kra11[i] = 0;
    kra12[i] = 1;
    kra13[i] = 0;
}

kratta_cal0_I.push_back(kra11[i]);
kratta_cal1_I.push_back(kra12[i]);
kratta_cal2_I.push_back(kra13[i]);
}

for (int i = 0; i < kratta_cal0_I.size(); i++)
cout << kratta_cal0_I[i] << "\t" << kratta_cal1_I[i] << "\t"
    << kratta_cal2_I[i] << endl;

cout << " " << endl;
cout << "cal_kratta_II" << endl;

FILE *plik3;
plik3 = fopen("calibrations/cal_kratta_II.txt", "r");
if (plik3 == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

for (int i = 0; i < 24; i++) {
    if (fscanf(plik3, "%lf %lf %lf", &kra21[i], &kra22[i],
        &kra23[i]) != 3) {
        printf("Blad wczytywania danych z wiersza %d.\n", i+1);
        kra21[i] = 0;
        kra22[i] = 1;
        kra23[i] = 0;
    }
    if (std::isnan(kra21[i]) || std::isnan(kra22[i])
        || std::isnan(kra23[i])) {
        printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
        kra21[i] = 0;
        kra22[i] = 1;
        kra23[i] = 0;
    }

    kratta_cal0_II.push_back(kra21[i]);
    kratta_cal1_II.push_back(kra22[i]);
    kratta_cal2_II.push_back(kra23[i]);
}

for (int i = 0; i < kratta_cal0_II.size(); i++)

```

```

        cout << kratta_cal0_II[i] << "\t" << kratta_cal1_II[i]
        << "\t" << kratta_cal2_II[i] << endl;
cout << " " << endl;

fclose(plik2);
fclose(plik3);

//-----
// wczytanie wspolczynnikow kalibacyjnych do energii LaBr
//-----

std::vector<double> labr_cal0_I, labr_cal1_I, labr_cal2_I;
std::vector<double> labr_cal0_II_1, labr_cal1_II_1, labr_cal2_II_1;
std::vector<double> labr_cal0_II_2, labr_cal1_II_2, labr_cal2_II_2;
std::vector<double> labr_cal0_II_3, labr_cal1_II_3, labr_cal2_II_3;
std::vector<double> labr_cal0_III, labr_cal1_III, labr_cal2_III;
std::vector<double> labr_cal0_IV, labr_cal1_IV, labr_cal2_IV;
double labr11[4], labr12[4], labr13[4];
double labr21[4], labr22[4], labr23[4];
double labr31[4], labr32[4], labr33[4];
double labr41[4], labr42[4], labr43[4];
double labr51[4], labr52[4], labr53[4];
double labr61[4], labr62[4], labr63[4];

cout << "cal_labr_I" << endl;

FILE *plik4;
plik4 = fopen("calibrations/cal_labr_I.txt", "r");
if (plik4 == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

for (int i = 0; i < 4; i++) {
    if (fscanf(plik3, "%lf %lf %lf", &labr11[i], &labr12[i],
    &labr13[i]) != 3) {
        printf("Blad wczytywania danych z wiersza %d.\n", i+1);
        labr11[i] = 0;
        labr12[i] = 1;
        labr13[i] = 0;
    }
    if (std::isnan(labr11[i]) || std::isnan(labr12[i])
    || std::isnan(labr13[i])) {
        printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
        labr11[i] = 0;
        labr12[i] = 1;
        labr13[i] = 0;
    }
}

labr_cal0_I.push_back(labr11[i]);
labr_cal1_I.push_back(labr12[i]);
labr_cal2_I.push_back(labr13[i]);
}

for (int i = 0; i < labr_cal0_I.size(); i++)

```

```

        cout << labr_cal0_I[i] << "\t" << labr_cal1_I[i]
        << "\t" << labr_cal2_I[i] << endl;

cout << " " << endl;
cout << "cal_labr_II_1" << endl;

FILE *plik5;
plik5 = fopen("calibrations/cal_labr_II_1.txt", "r");
if (plik5 == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

for (int i = 0; i < 4; i++) {
    if (fscanf(plik5, "%lf %lf %lf", &labr21[i], &labr22[i],
    &labr23[i]) != 3) {
        printf("Blad wczytywania danych z wiersza %d.\n", i+1);
        labr21[i] = 0;
        labr22[i] = 1;
        labr23[i] = 0;
    }
    if (std::isnan(labr21[i]) || std::isnan(labr22[i])
    || std::isnan(labr23[i])) {
        printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
        labr21[i] = 0;
        labr22[i] = 1;
        labr23[i] = 0;
    }
    labr_cal0_II_1.push_back(labr21[i]);
    labr_cal1_II_1.push_back(labr22[i]);
    labr_cal2_II_1.push_back(labr23[i]);
}

for (int i = 0; i < labr_cal0_II_1.size(); i++)
    cout << labr_cal0_II_1[i] << "\t" << labr_cal1_II_1[i]
    << "\t" << labr_cal2_II_1[i] << endl;

cout << "\n" << endl;
cout << "cal_labr_II_2" << endl;

FILE *plik6;
plik6 = fopen("calibrations/cal_labr_II_2.txt", "r");

if (plik6 == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

for (int i = 0; i < 4; i++) {
    if (fscanf(plik6, "%lf %lf %lf", &labr31[i], &labr32[i],
    &labr33[i]) != 3) {
        printf("Blad wczytywania danych z wiersza %d.\n", i+1);
        labr31[i] = 0;
        labr32[i] = 1;
        labr33[i] = 0;
    }
}

```

```

    }

    if (std::isnan(labr31[i]) || std::isnan(labr32[i])
        || std::isnan(labr33[i])) {
        printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
        labr31[i] = 0;
        labr32[i] = 1;
        labr33[i] = 0;
    }

    labr_cal0_II_2.push_back(labr31[i]);
    labr_cal1_II_2.push_back(labr32[i]);
    labr_cal2_II_2.push_back(labr33[i]);
}

for (int i = 0; i < labr_cal0_II_2.size(); i++)
    cout << labr_cal0_II_2[i] << "\t" << labr_cal1_II_2[i]
        << "\t" << labr_cal2_II_2[i] << endl;

cout << " " << endl;
cout << "cal_labr_II_3" << endl;

FILE *plik7;
plik7 = fopen("calibrations/cal_labr_II_3.txt", "r");
if (plik7 == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

for (int i = 0; i < 4; i++) {
    if (fscanf(plik7, "%lf %lf %lf", &labr41[i], &labr42[i],
        &labr43[i]) != 3) {
        printf("Blad wczytywania danych z wiersza %d.\n", i+1);
        labr41[i] = 0;
        labr42[i] = 1;
        labr43[i] = 0;
    }
    if (std::isnan(labr41[i]) || std::isnan(labr42[i])
        || std::isnan(labr43[i])) {
        printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
        labr41[i] = 0;
        labr42[i] = 1;
        labr43[i] = 0;
    }

    labr_cal0_II_3.push_back(labr41[i]);
    labr_cal1_II_3.push_back(labr42[i]);
    labr_cal2_II_3.push_back(labr43[i]);
}

for (int i = 0; i < labr_cal0_II_3.size(); i++)
    cout << labr_cal0_II_3[i] << "\t" << labr_cal1_II_3[i]
        << "\t" << labr_cal2_II_3[i] << endl;

cout << " " << endl;
cout << "cal_labr_III" << endl;

```

```

FILE *plik8;
plik8 = fopen("calibrations/cal_labr_III.txt", "r");
if (plik8 == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

for (int i = 0; i < 4; i++) {
    if (fscanf(plik8, "%lf %lf %lf", &labr51[i], &labr52[i],
    &labr53[i]) != 3) {
        printf("Blad wczytywania danych z wiersza %d.\n", i+1);
        labr51[i] = 0;
        labr52[i] = 1;
        labr53[i] = 0;
    }
    if (std::isnan(labr51[i]) || std::isnan(labr52[i])
    || std::isnan(labr53[i])) {
        printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
        labr51[i] = 0;
        labr52[i] = 1;
        labr53[i] = 0;
    }
}

labr_cal0_III.push_back(labr51[i]);
labr_cal1_III.push_back(labr52[i]);
labr_cal2_III.push_back(labr53[i]);
}

for (int i = 0; i < labr_cal0_III.size(); i++)
    cout << labr_cal0_III[i] << "\t" << labr_cal1_III[i]
    << "\t" << labr_cal2_III[i] << endl;

cout << " " << endl;
cout << "cal_labr_IV" << endl;

FILE *plik9;
plik9 = fopen("calibrations/cal_labr_IV.txt", "r");
if (plik9 == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

for (int i = 0; i < 4; i++) {
    if (fscanf(plik9, "%lf %lf %lf", &labr61[i], &labr62[i],
    &labr63[i]) != 3) {
        printf("Blad wczytywania danych z wiersza %d.\n", i+1);
        labr61[i] = 0;
        labr62[i] = 1;
        labr63[i] = 0;
    }
    if (std::isnan(labr61[i]) || std::isnan(labr62[i])
    || std::isnan(labr63[i])) {
        printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
        labr61[i] = 0;
        labr62[i] = 1;
    }
}

```

```

        labr63[i] = 0;
    }

    labr_cal0_IV.push_back(labr61[i]);
    labr_cal1_IV.push_back(labr62[i]);
    labr_cal2_IV.push_back(labr63[i]);
}

for (int i = 0; i < labr_cal0_IV.size(); i++)
    cout << labr_cal0_IV[i] << "\t" << labr_cal1_IV[i]
    << "\t" << labr_cal2_IV[i] << endl;

cout << " " << endl;

fclose(plik4);
fclose(plik5);
fclose(plik6);
fclose(plik7);
fclose(plik8);
fclose(plik9);
//-----
// wczytanie wspolczynnikow kalibacyjnych do roznicy czasu Silicon
//-----
std::vector<double> diff_cal0, diff_cal1;

cout << "cal_difftime" << endl;

FILE *plik10;
plik10 = fopen("calibrations/cal_difftime.txt", "r");
if (plik10 == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

double diff1[32], diff2[32];
for (int i = 0; i < 32; i++) {
    if (fscanf(plik10, "%lf %lf", &diff1[i], &diff2[i]) != 2) {
        printf("Blad wczytywania danych z wiersza %d.\n", i+1);
        diff1[i] = 0;
        diff2[i] = 1;
    }
    if (std::isnan(diff1[i]) || std::isnan(diff2[i])) {
        printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
        diff1[i] = 0;
        diff2[i] = 1;
    }
    diff_cal0.push_back(diff1[i]);
    diff_cal1.push_back(diff2[i]);
}

for (int i = 0; i < diff_cal0.size(); i++)
    cout << diff_cal0[i] << "\t" << diff_cal1[i] << endl;

fclose(plik10);

```

```

cout << " " << endl;
std::vector<double>diff_cal0_hist1, diff_cal1_hist1;

cout << "cal_difftime_hist1" << endl;

FILE *plik11;
plik11 = fopen("calibrations/cal_difftime_hist1.txt", "r");
if (plik11 == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

double diff1_hist1[4], diff2_hist1[4];
for (int i = 0; i < 4; i++) {
    if (fscanf(plik11, "%lf %lf", &diff1_hist1[i],
    &diff2_hist1[i]) != 2) {
        printf("Blad wczystywania danych z wiersza %d.\n", i+1);
        diff1_hist1[i] = 0;
        diff2_hist1[i] = 1;
    }
    if (std::isnan(diff1_hist1[i]) || std::isnan(diff2_hist1[i])) {
        printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
        diff1_hist1[i] = 0;
        diff2_hist1[i] = 1;
    }
    diff_cal0_hist1.push_back(diff1_hist1[i]);
    diff_cal1_hist1.push_back(diff2_hist1[i]);
}

for (int i = 0; i < diff_cal0_hist1.size(); i++)
    cout << diff_cal0_hist1[i] << "\t" << diff_cal1_hist1[i] << endl;

fclose(plik11);
cout << " " << endl;
std::vector<double>diff_cal0_hist2, diff_cal1_hist2;

cout << "cal_difftime_hist2" << endl;

FILE *plik12;
plik12 = fopen("calibrations/cal_difftime_hist2.txt", "r");
if (plik12 == NULL) {
    printf("Nie mozna otworzyc pliku.\n");
}

double diff1_hist2[4], diff2_hist2[4];
for (int i = 0; i < 4; i++) {
    if (fscanf(plik12, "%lf %lf", &diff1_hist2[i],
    &diff2_hist2[i]) != 2) {
        printf("Blad wczystywania danych z wiersza %d.\n", i+1);
        diff1_hist2[i] = 0;
        diff2_hist2[i] = 1;
    }
    if (std::isnan(diff1_hist2[i]) || std::isnan(diff2_hist2[i])) {
        printf("Blad: Wartosc nan w wierszu %d.\n", i+1);
    }
}

```

```

        diff1_hist2[i] = 0;
        diff2_hist2[i] = 1;
    }

    diff_cal0_hist2.push_back(diff1_hist2[i]);
    diff_cal1_hist2.push_back(diff2_hist2[i]);
}

for (int i = 0; i < diff_cal0_hist2.size(); i++)
    cout << diff_cal0_hist2[i] << "\t" << diff_cal1_hist2[i] << endl;

fclose(plik12);

//-----
// wypelnianie galezi nowego drzewa
//-----


Long64_t Nev = tree->GetEntries(); //liczba eventow
std::cout << "-----" << std::endl;
std::cout << " NUMBER OF EVENTS: " << Nev << std::endl;
std::cout << "-----" << std::endl;

for (Long64_t i=0;i<Nev;i++)
{ //Loop over events
tree->GetEntry(i);
std::cout<<"Processing entry no.
"<<i<<" out of "<<Nev<<'\r'<<std::flush;
if (i==Nev-1) std::cout<<'\n'<<std::endl;

//-----
// Galaz LaBr
//-----


for (int l = 0; l < labr->mult(); l++){
    LaBr.id = labr->det[l].id;
    LaBr.time = labr->det[l].time;
    LaBr.energyqshort = labr->det[l].qshort;
    LaBr.energyqlong = labr->det[l].qlong;

    if(nfo.nrun >= 4383 && nfo.nrun <= 4724){
        LaBr.energyqshort_cal = labr_cal2_I[LaBr.id]*
        LaBr.energyqshort*LaBr.energyqshort +
        labr_cal1_I[LaBr.id]*LaBr.energyqshort +
        labr_cal0_I[LaBr.id];

    }else if(nfo.nrun>=6984 && nfo.nrun<=7070){
        LaBr.energyqshort_cal = labr_cal2_II_1[LaBr.id]*
        LaBr.energyqshort*LaBr.energyqshort +
        labr_cal1_II_1[LaBr.id]*LaBr.energyqshort +
        labr_cal0_II_1[LaBr.id];

    }else if(nfo.nrun>=7071 && nfo.nrun<=7157){
        LaBr.energyqshort_cal = labr_cal2_II_2[LaBr.id]*

```

```

        LaBr.energyqshort*LaBr.energyqshort +
        labr_cal1_II_2[LaBr.id]*LaBr.energyqshort +
        labr_cal0_II_2[LaBr.id];

    }else if(nfo.nrun>=7158 && nfo.nrun<=7245){
        LaBr.energyqshort_cal = labr_cal2_II_3[LaBr.id]*
        LaBr.energyqshort*LaBr.energyqshort +
        labr_cal1_II_3[LaBr.id]*LaBr.energyqshort +
        labr_cal0_II_3[LaBr.id];

    }else if(nfo.nrun>=7368 && nfo.nrun<=7640){
        LaBr.energyqshort_cal = labr_cal2_III[LaBr.id]*
        LaBr.energyqshort*LaBr.energyqshort +
        labr_cal1_III[LaBr.id]*LaBr.energyqshort +
        labr_cal0_III[LaBr.id];

    }else if(nfo.nrun>=7641 && nfo.nrun<=7838){
        LaBr.energyqshort_cal = labr_cal2_IV[LaBr.id]*
        LaBr.energyqshort*LaBr.energyqshort +
        labr_cal1_IV[LaBr.id]*LaBr.energyqshort +
        labr_cal0_IV[LaBr.id];

    }else{
        LaBr.energyqshort_cal = LaBr.energyqshort;
    }

    T1->Fill();
}

//-----
// Galaz Plastic
//-----

for (int l = 0; l < plastic->mult(); l++){
    Plastic.id = plastic->det[l].id;
    Plastic.time = plastic->det[l].time;
    T1->Fill();
}

//-----
// Galaz Paris
//-----

for (int l = 0; l < paris->mult(); l++){
    Paris.id = paris->det[l].id;
    Paris.time = paris->det[l].time;
    Paris.energyqshort = paris->det[l].qshort;
    Paris.energyqlong = paris->det[l].qlong;
    T1->Fill();
}

//-----
// Galaz Kratta
//-----

for (int l = 0; l < kratta->mult(); l++){
    Kratta.id = kratta->det[l].id;
    Kratta.energyPDO = kratta->det[l].pd0.ampl;
    Kratta.energyPD1 = kratta->det[l].pd1.ampl;
}

```

```

Kratta.energyPD2 = kratta->det[1].pd2.ampl;
Kratta.timePD0 = kratta->det[1].pd0.time;
Kratta.timePD1 = kratta->det[1].pd1.time;
Kratta.timePD2 = kratta->det[1].pd2.time;
if(nfo.nrun >= 4383 && nfo.nrun <= 4724){
    Kratta.energyPD2_cal =
        kratta_cal2_I[Kratta.id]*
        Kratta.energyPD2*Kratta.energyPD2 +
        kratta_cal1_I[Kratta.id]*Kratta.energyPD2 +
        kratta_cal0_I[Kratta.id];

}else if(nfo.nrun >= 6984 && nfo.nrun <= 7838){
    Kratta.energyPD2_cal =
        kratta_cal2_II[Kratta.id]*
        Kratta.energyPD2*Kratta.energyPD2 +
        kratta_cal1_II[Kratta.id]*Kratta.energyPD2 +
        kratta_cal0_II[Kratta.id];

}else {
    Kratta.energyPD2_cal = Kratta.energyPD2;
}
T1->Fill();

}

//-----
// Galaz Silicon
//-----
for (int l = 0; l < silicon->mult(); l++){
    if(silicon->det[l].id != 26 &&
       silicon->det[l].id != 27 && silicon->det[l].id != 28
       && silicon->det[l].id != 29){

        Silicon.id = silicon->det[l].id;
        Silicon.energy = silicon->det[l].ampl;
        Silicon.energy_cal = sil_cal0[Silicon.id] +
            sil_cal1[Silicon.id] * Silicon.energy;
        Silicon.time30 = silicon->det[l].time30;
        Silicon.time80 = silicon->det[l].time80;
        Silicon.difftime = Silicon.time30 - Silicon.time80;

        if(silicon->det[l].id >=20 &&
           silicon->det[l].id <=23){
            if (nfo.nrun >= 7635 && nfo.nrun <= 7838) {
                Silicon.difftime_cal =
                    diff_cal0_hist1[Silicon.id - 20] +
                    diff_cal1_hist1[Silicon.id - 20] *
                    Silicon.difftime;

            } else if (nfo.nrun >= 6984 &&
                      nfo.nrun <= 7027) {
                Silicon.difftime_cal =
                    diff_cal0_hist1[Silicon.id - 20] +
                    diff_cal1_hist1[Silicon.id - 20] *

```

```

        Silicon.difftime;

    } else if (nfo.nrun >= 7028 &&
    nfo.nrun <= 7634) {
    Silicon.difftime_cal =
        diff_cal0_hist2[Silicon.id - 20] +
        diff_cal1_hist2[Silicon.id - 20] *
        Silicon.difftime;
    }
} else {
    Silicon.difftime_cal =
    diff_cal0[Silicon.id] +
    diff_cal1[Silicon.id] *
    Silicon.difftime;
}
}

T1->Fill();
}

T1->Write();
}

```

Dodatek F

**Wydruk programu do kalibracji
macierzy pomiarów naładowanych
cząstek z detektora krzemowego
DSSSD, plik: Peaks.C**

```
#include "TTree.h"
#include "TCutG.h"
#include "TH2F.h"
#include "TProfile.h"
#include "TFile.h"
#include "TSpectrum.h"
#include "TCanvas.h"
#include "TLine.h"
#include <TH2.h>
#include <TF1.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cstring>
#include <algorithm>
#include "TTreeReader.h"
#include "TTreeReaderValue.h"

void Peaks()
{
    Double_t times[2] = {1128.782994885779090, 1236.521202591038673};
    //ID = 25
    Double_t timesprim[2];
    Double_t cal[2];
    Double_t par_g[6];
    Double_t par_l[2];
    Double_t par_e[2];
    Double_t peaks[2];

    Double_t div[2]={0,0};
```

```

int p=0;

// ----- OPENING THE FILE AND TAKING THE HISTOGRAMS: ----- //
TFile *myFile = TFile::Open("./13C_data.root");

TTreeReader myReader("tree2", myFile);
TTreeReaderValue<Int_t> id(myReader, "Silicon.id");
TTreeReaderValue<Float_t> difftime(myReader, "Silicon.diffTime");
TTreeReaderValue<Int_t> run(myReader, "Info.run");

std::map<Int_t, TH1F*> histogram_1;
std::map<Int_t, TH1F*> histogram_2;

for (int i = 20; i <= 23; ++i) {
    histogram_1[i] = new TH1F(Form("hist1_%02d", i),
        Form("Silicon.Id = %02d (Run 6984-7027, 7635-7838)", i), 500, 0, 2500);

    histogram_2[i] = new TH1F(Form("hist2_%02d", i),
        Form("Silicon.Id = %02d (Run 7028-7634)", i), 500, 0, 2500);
}

std::map<Int_t, TH1F*> other_histograms;

for (int i = 0; i <= 31; ++i) {
    if (other_histograms.find(i) == other_histograms.end()) {
        other_histograms[i] = new TH1F(Form("hist_other_%02d",
            i), Form("Silicon.Id = %02d", i), 500, 10, 2500);
    }
}

while (myReader.Next())
{
    Int_t currentId = *id;
    Int_t currentRun = *run;
    Float_t currentDiffTime = *diffTime;

    if (currentId >= 20 && currentId <= 23) {
        if ((currentRun >= 6984 && currentRun <= 7027)
            || (currentRun >= 7635 && currentRun <= 7838)) {
            histogram_1[currentId]->Fill(currentDiffTime);
        }
        else if (currentRun >= 7028 && currentRun <= 7634) {
            histogram_2[currentId]->Fill(currentDiffTime);
        }
    }

    other_histograms[currentId]->Fill(currentDiffTime);
}

printf("\n");
printf("Single histograms\n");

```

```

printf("\n");
// ----- WITH ONE HISTOGRAM: ----- //
    fstream plik;
    plik.open("cal_difftime.txt", std::fstream::out
              | std::fstream::trunc);
    TFile *fout = new TFile("./graph.root","recreate");
    Double_t FWHM[2][32];
    Double_t ChiSqr[2][32];
    Double_t cal_par[2][32];
// ----- SEARCHING FOR THE PEAKS: ----- //
    for(auto &pair : other_histograms)
    {

        TH1F *hist = pair.second;
        hist->Draw();

        printf("START for Silicon.id = %d\n", pair.first);
        printf("\n");

        Int_t nfound = 0;
        TSpectrum *s = new TSpectrum();
        nfound = s->Search(hist,4,"",0.06);
        hist->ShowPeaks(4,"",0.05);
        printf("nfound = %d \n",nfound);

        if (nfound == 0){
            cal[0] = 0;
            cal[1] = 1;
            plik << cal[0] << "
                                         " << cal[1] << endl;
            continue;
        }

// ----- TAKING THE PEAK POSITIONS: ----- //

        Double_t *xpeaks = s->GetPositionX();

        int i=0;
        for (p = 0; p<2; p++){
            printf("Position of the %d peak: %3.2f\n",p+1,xpeaks[p]);

            if(xpeaks[p]>750){
                peaks[i]=xpeaks[p];
                i++;
            }
        }

        printf("\n");

        std::sort(peaks, peaks+2);

        for (p = 0; p<2; p++)
            printf("Position of the %d peak after swap:
                  %3.2f\n",p+1,peaks[p]);
    }
}

```

```

printf("\n");

// ----- FITTING FUNCTIONS: ----- //

for (p=0;p<2;p++) {
    Double_t r1 = peaks[p];
    Double_t r2 = peaks[p] + 30;
    Double_t r3 = peaks[p] - 50;
    Double_t r4 = peaks[p] - 15;

    TF1 *g = new TF1("g","gaus",r4,r2);
        hist->Fit(g,"NR+");
        g->GetParameters(&par_g[0]);
        peaks[p] = par_g[1];

    Double_t i1 = 0.76, i2 = 0.24;
    Double_t p0 = par_g[0];
    Double_t p2 = par_g[2];
    Double_t Mean2 = p0*i2/i1;
    Double_t pozycja = peaks[p] - 5;

    TF1 *g2 =new TF1("g2","gaus",r3,r4);
    g2->SetParameters(Mean2,pozycja,p2);
    hist->Fit(g2,"NR+");
    g2->GetParameters(&par_g[3]);
    Double_t pov = g->GetParameter(0);

    if (abs(pov)>100000){

        g->Delete();
        g2->Delete();
        TF1 *f1 = new TF1("ga","gaus",r4,r2);
        TF1 *f2 = new TF1("ex","expo",r3,r4);

        hist->Fit(f1,"NR+");
        f1->GetParameters(&par_g[0]);

        hist->Fit(f2,"NR+");
        f2->GetParameters(&par_l[0]);
        Double_t pov2 = f1->GetParameter(2);

        if(abs(pov2)>98.9){

            f1->Delete();
            f2->Delete();
            TF1 *h = new TF1("h","gaus",r3,r2);
            h->SetLineColor(6);
            hist->Fit(h,"R+");
            h->GetParameters(&par_g[0]);
            ChiSqr[p][pair.first] =
                h->GetChiSquare();
            peaks[p] = h->GetParameter(1);
            FWHM[p][pair.first] = 2.35*par_g[2];
        }
    }
}

```

```

        }

        else{
            TF1 *sum2 = new TF1("sum2","gaus(0)+[3]*
                exp((x-[1])/[2])*"
                TMath::Erfc((x-[1])/[2])",
                r3,r2);
            sum2->SetParameters(par_g[0],par_g[1],
                par_g[2],par_l[0]);
            sum2->SetLineColor(9);
            hist->Fit(sum2,"R+");
            ChiSqr[p][pair.first] = sum2->GetChisquare();
            peaks[p] = par_g[1];
            FWHM[p][pair.first] = 2.35*par_g[2];

        }
    }

else {

    TF1 *total = new TF1("total",
        "gaus(0)+gaus(3)",r3,r2);
    total->SetLineColor(3);
    total->SetParameters(par_g);
    hist->Fit(total,"R+");
    ChiSqr[p][pair.first] =
        total->GetChisquare();
    peaks[p] = par_g[1];
    FWHM[p][pair.first] = 2.35*par_g[2];

}

printf("Position of the %d peak from fit: %3.2f\n",
    p+1,peaks[p]);
printf("FWHM: %3.2f\n",FWHM[p][pair.first]);
}

// ----- ENERGY CALIBRATION: ----- //

TGraph *gr = new TGraph(sizeof(peaks) / sizeof(Double_t)),
    peaks, times);
TCanvas *grp = new TCanvas(Form("Fit%d",pair.first));
grp->Divide(1,2);
gr->Fit("pol1");
gr->SetMarkerStyle(20);
gr->SetMarkerColor(4);
gr->SetLineColor(0);
gr->SetTitle(Form("Calibration Id = %d", pair.first));
gr->GetYaxis()->SetTitle("Times");
gr->GetXaxis()->SetTitle("Channel");
grp->cd(1);
gr->Draw();

TF1 *myfunc = gr->GetFunction("pol1");

```

```

        for(int i = 0; i<2 ; i++){
            cal[i] = myfunc->GetParameter(i);
            printf("%d cal. coef. for %d spectrum:
                    %3.2f\n",i,pair.first,cal[i]);
            cal_par[i][pair.first] = cal[i];
        }
        plik << cal[0] << "           "
                           " << cal[1] << endl;

        for(p=0;p<2;p++){
            timesprim[p] = abs(cal[0]+cal[1]*peaks[p]);
            div[p]=times[p]-timesprim[p];
        }

        TGraph *gr2 = new TGraph((sizeof(peaks)
                                  / sizeof(Double_t)), peaks, div);
        gr2->SetMarkerStyle(20);
        gr2->SetMarkerColor(4);
        gr2->SetLineColor(0);
        gr2->SetTitle("Residuum");
        gr2->GetYaxis()->SetTitle("Delta Times");
        gr2->GetXaxis()->SetTitle("Channel");

        grp->cd(2);
        gr2->Draw();
        TF1 *l = new TF1("line","pol0");
        l->SetParameters(0);
        gr2->Fit(l);

        fout->cd();
        grp->Write();

    } // end of loop on histograms

    plik.close();
    fout->Close();
    printf("\n");
    printf("Double histograms part 1\n");
    printf("\n");

// ----- LEFT PART WITH TWO HISTOGRAMS: ----- //
    fstream plik2;
    plik2.open("cal_difftime_hist1.txt", std::fstream::out
               | std::fstream::trunc);
    TFile *fout2 = new TFile("./graph_hist1.root","recreate");
    Double_t FWHM2[2][32];
    Double_t ChiSqr2[2][32];
    Double_t cal_par2[2][32];
// ----- SEARCHING FOR THE PEAKS: ----- //
    for(auto &pair : histogram_1)
    {

        TH1F *hist2 = pair.second;
        hist2->Draw();

```

```

printf("START for Silicon.id = %d\n", pair.first);
printf("\n");

Int_t nfound = 0;
TSpectrum *s2 = new TSpectrum();
nfound = s2->Search(hist2,4,"",0.06);
hist2->ShowPeaks(4,"",0.05);
printf("nfound = %d \n",nfound);

if (nfound == 0){
    cal[0] = 0;
    cal[1] = 1;
    plik2 << cal[0] << "           " << cal[1] << endl;
    continue;
}

// ----- TAKING THE PEAK POSITIONS: -----
Double_t *xpeaks = s2->GetPositionX();

int i=0;
for (p = 0; p<2; p++){
    printf("Position of the %d peak: %3.2f\n",p+1,xpeaks[p]);

    if(xpeaks[p]>750){
        peaks[i]=xpeaks[p];
        i++;
    }
}

printf("\n");

std::sort(peaks, peaks+2);

for (p = 0; p<2; p++)
    printf("Position of the %d peak after swap: %3.2f\n",
           p+1,peaks[p]);

printf("\n");

// ----- FITTING FUNCTIONS: -----
for (p=0;p<2;p++) {
    Double_t r1 = peaks[p];
    Double_t r2 = peaks[p] + 30;
    Double_t r3 = peaks[p] - 50;
    Double_t r4 = peaks[p] - 15;

    TF1 *g = new TF1("g","gaus",r4,r2);
    hist2->Fit(g,"NR+");
    g->GetParameters(&par_g[0]);
    peaks[p] = par_g[1];

    Double_t i1 = 0.76, i2 = 0.24;
}

```

```

Double_t p0 = par_g[0];
Double_t p2 = par_g[2];
Double_t Mean2 = p0*i2/i1;
Double_t pozycja = peaks[p] - 5;

TF1 *g2 = new TF1("g2","gaus",r3,r4);
g2->SetParameters(Mean2,pozycja,p2);
hist2->Fit(g2,"NR+");
g2->GetParameters(&par_g[3]);
Double_t pov = g->GetParameter(0);

if (abs(pov)>100000){

    g->Delete();
    g2->Delete();
    TF1 *f1 = new TF1("ga","gaus",r4,r2);
    TF1 *f2 = new TF1("ex","expo",r3,r4);

    hist2->Fit(f1,"NR+");
    f1->GetParameters(&par_g[0]);

    hist2->Fit(f2,"NR+");
    f2->GetParameters(&par_l[0]);
    Double_t pov2 = f1->GetParameter(2);

    if(abs(pov2)>98.9){

        f1->Delete();
        f2->Delete();
        TF1 *h = new TF1("h","gaus",r3,r2);
        h->SetLineColor(6);
        hist2->Fit(h,"R+");
        h->GetParameters(&par_g[0]);
        ChiSqr2[p][pair.first] =
            h->GetChiSquare();
        peaks[p] = h->GetParameter(1);
        FWHM2[p][pair.first] = 2.35*par_g[2];

    }

    else{
        TF1 *sum2 = new TF1("sum2","gaus(0)+[3]*
            exp((x-[1])/[2])*"
            "TMath::Erfc((x-[1])/[2])",
            r3,r2);
        sum2->SetParameters(par_g[0],par_g[1],
            par_g[2],par_l[0]);
        sum2->SetLineColor(9);
        hist2->Fit(sum2,"R+");
        ChiSqr2[p][pair.first] = sum2->GetChiSquare();
        peaks[p] = par_g[1];
        FWHM2[p][pair.first] = 2.35*par_g[2];

    }

}

```

```

        }
    else {

        TF1 *total = new TF1("total",
                            "gaus(0)+gaus(3)", r3, r2);
        total->SetLineColor(3);
        total->SetParameters(par_g);
        hist2->Fit(total, "R+");
        ChiSqr2[p][pair.first] =
            total->GetChisquare();
        peaks[p] = par_g[1];
        FWHM2[p][pair.first] = 2.35*par_g[2];

    }

    printf("Position of the %d peak from fit: %3.2f\n",
           p+1, peaks[p]);
    printf("FWHM: %3.2f\n", FWHM2[p][pair.first]);
}

// ----- ENERGY CALIBRATION: ----- //

TGraph *gr = new TGraph((sizeof(peaks) / sizeof(Double_t)),
                        peaks, times);
TCanvas *grp = new TCanvas(Form("Fit%d", pair.first));
grp->Divide(1, 2);
gr->Fit("pol1");
gr->SetMarkerStyle(20);
gr->SetMarkerColor(4);
gr->SetLineColor(0);
gr->SetTitle(Form("Calibration Id = %d", pair.first));
gr->GetYaxis()->SetTitle("Times");
gr->GetXaxis()->SetTitle("Channel");
grp->cd(1);
gr->Draw();

TF1 *myfunc = gr->GetFunction("pol1");

for(int i = 0; i<2 ; i++){
    cal[i] = myfunc->GetParameter(i);
    printf("%d cal. coef. for %d spectrum:
           %3.2f\n", i, pair.first, cal[i]);
    cal_par2[i][pair.first] = cal[i];
}
plik2 << cal[0] << "
                           " << cal[1] << endl;

for(p=0;p<2;p++){
    timesprim[p] = abs(cal[0]+cal[1]*peaks[p]);
    div[p]=times[p]-timesprim[p];
}

TGraph *gr2 = new TGraph((sizeof(peaks)
                           / sizeof(Double_t)), peaks, div);
gr2->SetMarkerStyle(20);

```

```

        gr2->SetMarkerColor(4);
        gr2->SetLineColor(0);
        gr2->SetTitle("Residuum");
        gr2->GetYaxis()->SetTitle("Delta Times");
        gr2->GetXaxis()->SetTitle("Channel");

        grp->cd(2);
        gr2->Draw();
        TF1 *l = new TF1("line","pol0");
        l->SetParameters(0);
        gr2->Fit(l);

        fout2->cd();
        grp->Write();

    } // end of loop on histograms

    plik2.close();
    fout2->Close();
printf("\n");
printf("Double histograms part 2\n");
printf("\n");
// ----- RIGHT PART WITH TWO HISTOGRAMS: -----
fstream plik3;
plik3.open("cal_difftime_hist2.txt", std::fstream::out
           | std::fstream::trunc);
TFile *fout3 = new TFile("./graph_hist2.root","recreate");
Double_t FWHM3[2][32];
Double_t ChiSqr3[2][32];
Double_t cal_par3[2][32];
// ----- SEARCHING FOR THE PEAKS: -----
for(auto &pair : histogram_2)
{
    TH1F *hist3 = pair.second;
    hist3->Draw();

    printf("START for Silicon.id = %d\n", pair.first);
    printf("\n");

    Int_t nfound = 0;
    TSpectrum *s3 = new TSpectrum();
    nfound = s3->Search(hist3,1.4,"",0.06);
    hist3->ShowPeaks(1.4,"",0.06);
    printf("nfound = %d \n",nfound);

    if (nfound == 0){
        cal[0] = 0;
        cal[1] = 1;
        plik3 << cal[0] << "
                                         " << cal[1] << endl;
        continue;
    }

// ----- TAKING THE PEAK POSITIONS: -----

```

```

Double_t *xpeaks = s3->GetPositionX();

int i=0;
for (p = 0; p<2; p++){
    printf("Position of the %d peak: %3.2f\n",p+1,xpeaks[p]);

    if(xpeaks[p]>750){
        peaks[i]=xpeaks[p];
        i++;
    }
}

printf("\n");

std::sort(peaks, peaks+2);

for (p = 0; p<2; p++)
    printf("Position of the %d peak after swap:
           %3.2f\n",p+1,peaks[p]);

printf("\n");

// ----- FITTING FUNCTIONS: -----
// 

for (p=0;p<2;p++) {
    Double_t r1 = peaks[p];
    Double_t r2 = peaks[p] + 30;
    Double_t r3 = peaks[p] - 50;
    Double_t r4 = peaks[p] - 15;

    TF1 *g = new TF1("g","gaus",r4,r2);
    hist3->Fit(g,"NR+");
    g->GetParameters(&par_g[0]);
    peaks[p] = par_g[1];

    Double_t i1 = 0.76, i2 = 0.24;
    Double_t p0 = par_g[0];
    Double_t p2 = par_g[2];
    Double_t Mean2 = p0*i2/i1;
    Double_t pozycja = peaks[p] - 5;

    TF1 *g2 =new TF1("g2","gaus",r3,r4);
    g2->SetParameters(Mean2,pozycja,p2);
    hist3->Fit(g2,"NR+");
    g2->GetParameters(&par_g[3]);
    Double_t pov = g->GetParameter(0);

    if (abs(pov)>100000){

        g->Delete();
        g2->Delete();
        TF1 *f1 = new TF1("ga","gaus",r4,r2);
        TF1 *f2 = new TF1("ex","expo",r3,r4);
}

```

```

hist3->Fit(f1,"NR+");
f1->GetParameters(&par_g[0]);

hist3->Fit(f2,"NR+");
f2->GetParameters(&par_l[0]);
Double_t pov2 = f1->GetParameter(2);

if(abs(pov2)>98.9){

    f1->Delete();
    f2->Delete();
    TF1 *h = new TF1("h","gaus",
                      r3,r2);
    h->SetLineColor(6);
    hist3->Fit(h,"R+");
    h->GetParameters(&par_g[0]);
    ChiSqr3[p][pair.first] =
        h->GetChisquare();
    peaks[p] = h->GetParameter(1);
    FWHM3[p][pair.first] = 2.35*par_g[2];

}

else{

TF1 *sum2 = new TF1("sum2","gaus(0)+[3]*
                     exp((x-[1])/[2])*"
                     TMath::Erfc((x-[1])/[2])",
                     r3,r2);
sum2->SetParameters(par_g[0],par_g[1],par_g[2],
                     par_l[0]);
sum2->SetLineColor(9);
hist3->Fit(sum2,"R+");
ChiSqr3[p][pair.first] = sum2->GetChisquare();
peaks[p] = par_g[1];
FWHM3[p][pair.first] = 2.35*par_g[2];

}
}
else {

TF1 *total = new TF1("total",
                      "gaus(0)+gaus(3)",
                      r3,r2);
total->SetLineColor(3);
total->SetParameters(par_g);
hist3->Fit(total,"R+");
ChiSqr3[p][pair.first] =
    total->GetChisquare();
peaks[p] = par_g[1];
FWHM3[p][pair.first] = 2.35*par_g[2];

}
printf("\n");

```

```

        printf("Position of the %d peak from fit: %3.2f\n",
               p+1, peaks[p]);
        printf("FWHM: %3.2f\n", FWHM3[p][pair.first]);
        printf("\n");
    }

// ----- ENERGY CALIBRATION: ----- //

TGraph *gr = new TGraph((sizeof(peaks) / sizeof(Double_t)),
                        peaks, times);
TCanvas *grp = new TCanvas(Form("Fit%d", pair.first));
grp->Divide(1, 2);
gr->Fit("pol1");
gr->SetMarkerStyle(20);
gr->SetMarkerColor(4);
gr->SetLineColor(0);
gr->SetTitle(Form("Calibration Id = %d", pair.first));
gr->GetYaxis()->SetTitle("Times");
gr->GetXaxis()->SetTitle("Channel");
grp->cd(1);
gr->Draw();

TF1 *myfunc = gr->GetFunction("pol1");
printf("\n");
for(int i = 0; i<2 ; i++){
    cal[i] = myfunc->GetParameter(i);
    printf("%d cal. coef. for %d spectrum:
           %3.2f\n", i, pair.first, cal[i]);
    cal_par3[i][pair.first] = cal[i];
}
plik3 << cal[0] << "
                           " << cal[1] << endl;

for(p=0;p<2;p++){
    timesprim[p] = abs(cal[0]+cal[1]*peaks[p]);
    div[p]=times[p]-timesprim[p];
}
printf("\n");
TGraph *gr2 = new TGraph((sizeof(peaks)
                           / sizeof(Double_t)), peaks, div);
gr2->SetMarkerStyle(20);
gr2->SetMarkerColor(4);
gr2->SetLineColor(0);
gr2->SetTitle("Residuum");
gr2->GetYaxis()->SetTitle("Delta Times");
gr2->GetXaxis()->SetTitle("Channel");

grp->cd(2);
gr2->Draw();
TF1 *l = new TF1("line","pol0");
l->SetParameters(0);
gr2->Fit(l);

fout3->cd();
grp->Write();

```

```

        printf("\n");
    } // end of loop on histograms

    plik3.close();
    fout3->Close();

// ----- CREATING THE OUTPUT FILE WITH THE HISTOGRAMS: ----- //

TFile *outputFile = new TFile("output.root", "RECREATE");
for (auto &pair : histogram_1)
{
    pair.second->Write();
}

for (auto &pair : histogram_2)
{
    pair.second->Write();
}

for (auto &pair : other_histograms)
{
    pair.second->Write();
}

outputFile->Close();
myFile->Close();
}

```

Bibliografia

- [1] Wolf, Edward L. Physics and technology of sustainable energy. Oxford University Press, 2018.
- [2] Krane, Kenneth S. Introductory nuclear physics. John Wiley & Sons, 1991.
- [3] Magill, Joseph, and Jean Galy. Radioactivity, radionuclides, radiation. Vol. 259. Berlin: Springer, 2005.
- [4] Cieplicka-Oryńczak, N., et al. "The decay of the 21.47-mev stretched resonance in ^{13}C : a precise probe of the open nuclear quantum system description." Physics Letters B 834 (2022): 137398.
- [5] Ziliani, Sara. "Comprehensive investigation of light neutron-rich nuclei to test modern nuclear theoretical models." (2021).
- [6] R.S. Hicks et al., "M4 excitations in ^{13}C ". Phys. Rev. C, 34 (1986) 1161.
- [7] S.F. Collins et al., "The spectroscopy of ^{13}C by inelastic proton scattering". Nucl. Phys. A, 481 (1988) 494.
- [8] S.J. Seestrom-Morris et al., "Inelastic Pion Scattering from ^{13}C at 162 MeV". Phys. Rev. C, 26 (1982) 594
- [9] A. Maj et al., Acta Phys. Pol. B, 40 (2009) 565.
- [10] <https://root.cern/>
- [11] <https://www.nndc.bnl.gov/nudat3/>