Uniwersytet Wrocawski
Wydział Fizyki i Astronomii

Artur Kobyliński
Nr albumu: 130957

Generowanie neutrin metodą Monte-Carlo. Rozszerzenie symulatora NuWro

Praca magisterska
na kierunku FIZYKA KOMPUTEROWA

Praca wykonana pod kierunkiem
dr Cezarego Juszczaka
Instytut Fizyki Teoretycznej

Styczeń 2012

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

Punktem wyjścia w pracy jest doświadczenie T2K w Japonii oraz z nim współpracujący wrocławski projekt o nazwie NuWro. Symulator NuWro do zderzeń neutrina z jądrem atomowym został rozbudowany o dwie bardziej wyrafinowane metody dostarczania neutrin do symulacji, co było celem tej pracy magisterskiej. Pierwsza metoda polega na ciągłym odczytywaniu neutrin z plików dostarczonych w T2K. Takie rozwiązanie prowadzi do konieczności przechowywania ogromenej ilości danych przez co została zaproponowana i zaimplementowana metoda dwuetapowa. W pierwszym kroku symulator generuje histogram na podstawie wspomnianych plików z T2K, który następnie może być wielokrotnie używany w symulacji. Drugie rozwiązanie cechuje szybkość i mała ilość potrzebych danych do rozpoczęcia symulacji.

The starting point of this thesis is experyment T2K from Japan and related with him NuWro project in Wroclaw. Simulator NuWro was made for simulation of neutrino-nucleus scattering and was expanded by two methods of delivery of new neutrinos, more sophisticated than the original one, that was the goal of the thesis. First approach consists of a continuous reading of neutrinos from files of T2K experiment. Such solution leads to the necessity of store huge amounts of data, so that's why was proposed and implemented second two-step approach. First step means that NuWro simulator has to generate histogram which describes the neutrinos from T2K experiment and when this is done then the histogram can be used multiple times. Second approach is faster and needs much less data to start the scattering simulation.

Słowa kluczowe

neutrina, generowanie neutrin, histogram, NuWro, Neut, T2K, Kamiokande, ND280, neutrinos, generating of neutrinos

Dziedzina pracy (kody wg programu Socrates-Erasmus)

Fizyka wysokich energii 13.5

Klasyfikacja tematyczna

242000 Fizyka cząstek elementarnych i pól

Tytuł pracy w języku angielskim

Generation of neutrinos based on Monte-Carlo approach. Extension of NuWro simulator.

Spis treści

Wprowadzenie 5

1. Neutrina 7
1.1. Rozpad beta plus 7
1.2. Promieniowanie kosmiczne 8
1.3. Oscylacja neutrin 9

2. Eksperyment T2K 13
2.1. Super-Kamiokande 13
2.2. J-PARC 15
2.3. Bliski detektor ND280 15
2.4. Ingrid 17
2.5. ROOT 19
2.6. NuWro 20
2.6.1. Zasady definiowania parametrów symulacji 20
2.6.2. Główne parametry symulacji 21
2.6.3. Definiowanie tarczy 23
2.6.4. Wiązka beam_uniform 23
2.6.5. Wiązka beam_mixed 25
2.6.6. Wiązka BeamRF 26
2.6.7. Wiązka BeamHist 26
2.6.8. Pętla symulacji 27

3. Implementacja wiązki neutrin 29
3.1. Zapis wartości fizycznych 29
3.2. Odczyt neutrin z plików 30
3.3. Generowanie neutrin na podstawie histogramu 34
3.3.1. Tworzenie histogramu 34
3.3.2. Tworzenie neutrin na podstawie histogramu jednowymiarowego 36
3.3.3. Macierz n-wymiarowa 38
3.3.4. Tworzenie neutrin na podstawie histogramu 5-cio wymiarowego 39
3.3.5. Optymalizacja czasu wykonania dla wyszukiwania w histogramie 39
3.4. Próby odnalezienia symetrii w wiązce 40
3.5. Przejścia pomiędzy układami współrzędnych w T2K 41

4. Wyniki po integracji z NuWro 43

5. Podsumowanie 47

Dodatki 51

A. Wykresy 53

B. Struktura pliku histogramowego 73
B.1. Histogram bez optymalizacji 73
B.2. Histogram z usuniętymi nadmiarowymi zerami 74

C. Program konwertujący współrzędne Kokyo⇌Neut 75

D. Fragmenty kodu C++ 79
D.1. Generowanie cząstki w klasie beamRF 79
D.2. Tworzenie histogramu 83
D.3. Tworzenie neutrina na podstawie histogramu 88

Wprowadzenie

Aktualnie w wielu miejscach na Świecie prowadzi się badania nad neutrinami. Te cząstki, które jeszcze do niedawna uważano za bezmasowe, fascynują fizyków odmiennością od innych elementów modelu standardowego. Odkrywanie ich własności to prawdziwe wyzwanie dla współczesnej nauki i technologii. Pomimo tego, że tylko przez ludzkie ciało w każdej sekundzie przelatuje ich zawrotna ilość, bo około 50 bilionów, to żadne z nich nie zatrzymuje się w człowieku. Dzieje się tak, gdyż neutrina mają bardzo niski przekrój czynny na oddziaływanie z materią. Większość z tych, które powstały, przykładowo na Słońcu, przelatują przez Ziemię nie zderzając się z ani jedną jej cząstką.

Dwa największe laboratoria zaangażowane w badania nad neutrinami to Japońskie T2K i Europejski LHC w CERN-ie. Fizyka cząstek elementarnych dalece przerosła już możliwości odkrywcze jednego człowieka - wyniki badań to efekt współpracy setek naukowców z całego świata. Wybór tego tematu na pracę magisterską traktuję więc jako szansę wniesienia własnego, choćby minimalnego wkładu do tego ogromnego eksperymentalnego dzieła.

Celem pracy jest rozszerzenie aplikacji NuWro o nowy sposób pobierania neutrin. NuWro jest symulatorem który w swoim silniku ma zaimplementowaną fizykę dla rozpadów binarnych. Praca ta koncentruje się na zjawisku rozpraszania neutrin.

Wspomniana fizyka to model standardowy opisujący oddziaływania elektromagnetyczne, słabe i silne. Warunkiem poprawnej symulacji zdarzeń neutrinowych jest znajomość wiązki neutrin. Dane takie udostępniane są w ramach eksperymentu T2K, na potrzby tej pracy zostały wykorzystane w dwojaki sposób. Po pierwsze, bezpośredni odczyt z pliku w trakcie symulacji. Po drugie tworzenie histogramu na podstawie którego są generowane nowe cząstki.

Praca składa się z dwóch części. W pierwszej, przedstawiona zotanie teoria opisująca rozproszenia neutrin oraz sposób jej implementacji w symulatorze. Drugą część stanowić będzie kod programu zapisany na nośniku danych i dołączony do tekstu. Nowy moduł jest dostępny jako integralna część symulatora NuWro i pomoże w kontynuowaniu prac badawczych.

Niniejszy tekst stanowi pierwszą część pracy i znajdziemy w nim opis historii i teorii neutrin, ich właściwości i metody badania (przykładowo T2K) - omówienie modelu standardowego i jego realizacji w NuWro. Również tutaj omówimy kod C++ dodany do symulatora i wykorzystane metody generowania wiązki neutrin. Wreszcie, znajdziemy tu wnioski odnośnie użycia nowego modułu: jego zalet i wad.

Rozdział 1

Neutrina

Pierwsze hipotezy o istnieniu neutrin pojawiły się już w latach trzydziestych zeszłego stulecia. Potwierdziły je pierwsze eksperymenty przy użyciu reaktorów jądrowych. Współcześnie fizykę neutrin opisuje model standardowy - jest to zbiór teorii i własności dotyczących cząstek elementarnych.

1.1. Rozpad beta plus

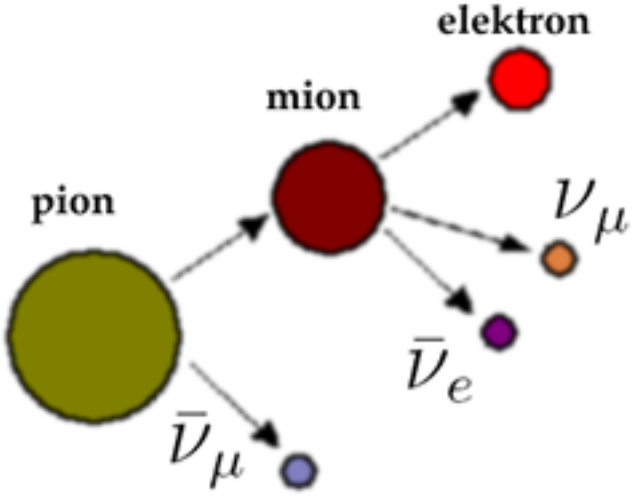
Rozróżniamy trzy neutrina, należące do grupy sześciu cząstek zwanych leptonami (zob. tabela 1.1). Każda z wymienionych leptonów posiada swojego antybrata, dla przykładu anty-neutrino mionowe będzie oznaczone nu^{-}.

elektron	mion	taon
e	μ	τ
neutrino elektronowe	neutrino mionowe	neutrino taonowe
ν_e	ν_μ	ν_τ

W latach 30-stych pracowano nad rozpadem β. początkowe założenia, iż w takim procesie powstają dwie cząstki - elektron i proton - nie zgadzały się z wynikami doświadczeń które pokazywały spektrum energetyczne elektronu. To oznaczało brak możliwości określenia jednej energii dla elektronu co łamie zasadę zachowania energii. Rozwiązanie zaproponowane przez Wolfganga Pauliego zakładało istnienie nowej, elektrycznie obojętnej cząstki, nazwanej neutrinem (od małego neutronu).

$$n^0 \rightarrow p^+ + e^- + \bar{\mu}_e$$

Równania matematyczne zaproponowane później przez Enrico Fermiego pokazywały na bardzo mały przekrój czynny w oddziaływaniach z materią. Tym samym udowodnienie istnienia antyneutrina elektronowego było możliwe dopiero 20 lat później przy użyciu reaktora jądrowego. Podstawowym założeniem dla tego doświadczenia był hipotetyczny rozpad β^+ który inicjowany jest przez neutrino uderzające w proton. Podczas tego procesu powstaje pozyton e^+. Ten dodatni elektron szybko znajduje elektron ujemny z którym się anihiluje wypromieniowując foton. Dodatkowo neutron, który powstał



Rysunek 1.1: Pełna kaskada leptonów

z protonu, pędząc przez ośrodek traci energię kinetyczną i zostaje wchłonięty przez jądro kadmu podnosząc tym samym jego poziom energetyczny. Jądro przestaje być wzbudzone po 15 sekundach wypromieniowując foton. Wszystkie wymienione błyski były obserwowane w trakcie doświadczenia za pomocą scyntylatorów. Poniżej zapis rozpadu beta plus.

$$\bar{\nu_e} + p^+ \rightarrow n^0 + e^+$$

1.2. Promieniowanie kosmiczne

W 1910 roku jezuita Teodor Wulf mierzył promieniotwórczość naturalną Ziemi - Wchodził na wieżę Eiffela i zapisywał pomiary. Ku jego zdziwieniu okazało się, że promieniowanie rośnie. Rok później odkrycie to zostało potwierdzone przez Wiktora Hessa, który dokonywał pomiarów z wznoszącego się balonu. Dziś wiemy, że to zjawisko powodowane jest wysokoenergetycznymi cząstkami wpadającymi w naszą przestrzeń z kosmosu. W atmosferze ziemskiej dochodzi do rozpadu kaskadowego: jądra oddziałując z kosmicznymi cząstkami rozpadają się i tworzą nowe cząstki, które następnie działają na inne jądra.

Proces zderzania się kosmicznego promieniowania z ziemską atmosferą był doskonałym materiałem do prac badawczych, ponieważ pozwalał podejrzewać możliwość odkrycia nowych cząstek. Te przypuszczenia okazały się słuszne.W latach 30-stych dokonano pierwszej obserwacji pozytonu, zaś w 1937 odkryto cząstkę, która mimo podobieństwa do elektronu, łatwiej przenika przez ośrodek i jest 20 razy cięższa - nazwano ją mionem. Zauważono, że nowa cząstka rozpada się na elektron. Dziesięć lat później odkryto pion i również w tym przypadku zaobserwowano samoistny rozpad, tym razem do mionu. Pełen łańcuch przedstawiony jest na rysunku 1.1. Szybko okazało się, że odkryta kaskada nie spełnia zasad zachowania energii i pędu - czyli że rozpad pionu w mion, a później mionu w elektron nie jest możliwy w tak prostej formie. Ponieważ te trzy cząstki posiadają ładunek, oznacza to, że inne hipotetycznie powstające elementy

powinny być elektrycznie obojętne. Jeżeli podczas rozpadu powstają dwie nowe cząstki to ta zawierająca ładunek będzie zawsze posiadała tą samą energię, natomiast gdy powstaną trzy to naładowana posiadać będzie dyskretny rozkład energii. Tymczasem okazało się, że energia mionu jest stała, a elektronu nie. O ile naukowcy mogli określić liczbę nowopowstałych cząstek, to niestety nie wiedzieli, co to za cząstki. Należy zwrócić uwagę, iż najprostszym rozwiązaniem dla kaskady jest wprowadzenie fotonu. Jednak ta metoda, mimo kilku lat badań, była bezowocna. Ostatecznie stwierdzono, że rozpad pionu i mionu nie może zachodzić przez wypromieniowanie kwantów światła.

W teoretycznym podejściu do omawianych rozpadów wprowadzono zasadę zachowania liczby leptonowej (mionowa, elektronowa i taonowa). Ponieważ foton, będąc bozonem, jej nie zawiera, to przykładowy rozpad pionu do mionu i fotonu nie byłby możliwy ze względu na niezgodność liczby mionowej. Liczba leptonowa L dla pionu wynosi zero.

$$\begin{cases} L_\pi = L_\mu - L_\gamma \\ 0 = 1 - 0 \end{cases}$$

Gdy w roku 55 odkryto neutrino elektronowe, a później, w 60-tym, neutrino mionowe, stało się jasne, że poszukiwanymi cząsteczkami dla omawianej kaskady są neutrina. Dowód na istnienie trzeciego neutrina został przeprowadzony w roku dwutysięcznym. W tym celu użyto emulsji jądrowej - jest to emulsja fotograficzna o dużym stężeniu bromku srebra AgBr i dziesięciokrotnie mniejszymi ziarnami (około $0.1\mu m$)w stosunku do standardowej. Lecąca naładowana cząstka powoduje zmiany w kryształach srebra i po wywołaniu filmu widoczny jest tor lotu w postaci zaczernionych kryształków. Dokładność tej metody okazała się na tyle duża, że można było szukać torów lotu krótko żyjących taonów, które przechodziły w inny długi tor. W momencie ich rozpadu, zachodziła zmiana kąta lotu cząstki, co pozwalało na identyfikację tego punktu. Odnaleziono cztery oddziaływania w których powstał taon.

1.3. Oscylacja neutrin

Teorię oscylacji neutrin, czyli przechodzenie jednego zapachu neutrina w inny, zaproponował w latach 60-tych B. Pontecorvo. Była to odpowiedź na problem niedoboru neutrin słonecznych rejestrowanych na Ziemi. Naukowcy w tamtych latach posiadali już wiedzę na temat reakcji syntezy zachodzących w Słońcu więc mogli oszacować ilość powstających tam neutrin. Poniżej przedstawione zostały reakcje zachodzące na Słońcu podczas których powstają neutrina. Pierwsza synteza jest częścią najbardziej popularnego cyklu $pp1$, produkuje on 86% energii słonecznej.

$$p + p \rightarrow {}^2H + e^+ + \nu_e$$

Kolejno $pp2$ i $pp3$

$$^7Be + e^- \rightarrow {}^7Li + \nu_e$$
$$^8B \rightarrow {}^8Be + e^+ + \nu_e$$

Bardzo rzadko zachodząca reakcja pep

$$p + e^- + p \rightarrow {}^2H + \nu_e$$

Ostatni cykl słoneczny CNO węglowo azotowo tlenowy daje zaledwie 1% wkładu do ogólnej energii Słońca, dzieje się tak dlatego, że warunki panujące w gwieździe powodują jej powolny przebieg.

$$^{13}N \rightarrow {}^{13}C + e^{+} + \nu_e$$

$$^{15}O \rightarrow {}^{15}N + e^{+} + \nu_e$$

Znając reakcje i ich procentowy udział w produkcji energii słonecznej można określić natężenie neutrin elektronowych na Ziemi. Właśnie te szacunki nie zgadzały się z wynikami doświadczalnymi prowadzonymi w eksperymencie Homestake. Doświadczenie pozwala na detekcję tylko neutrin elektronowych więc obserwacja taonowych lub mionowych nie jest możliwa. Przez co naukowcy nie byli w stanie zaobserwować zwiększonej ilości ν_μ i ν_τ a jedynie niedobór neutrin elektronowych. Podobnie jest w eksperymencie Super Kamiokande opisanym w rozdziale 2. Wyłapywanie słonecznych neutrin taonowych lub mionowych jest problematyczne ze względu na ich niską energię. Te neutrina nie są w stanie wyprodukować ciężkiego mionu lub taonu, które to przez swój ładunek mogłyby być obserwowane i pośrednio dowodzić tego, że zostały wyłapane odpowiadające im neutrina.

Pomiar natężenia docierających do ziemi ze słońca wszystkich trzech typów neutrin został przeprowadzony w Ameryce w detektorze SNO. Doświadczenie tam odbywające się są zgodne ze Standardowym Modelem Słońca. Ilość wyłapanych neutrin słonecznych w połączeniu z teorią oscylacji jest zgodna z przedstawionymi wyżej reakcjami syntezy. Po pierwsze eksperyment SNO bazuje na dwóch reakcjach w których neutrino rozbija deuter. Po drugie dochodzi tam do przekazania energii neutrina dla elektronu będącego częścią ośrodka.

$$D + \nu_e \rightarrow p + p + e^{-}$$

$$D + \nu_x \rightarrow n + p + \nu_x$$

Pierwsza reakcja jest czuła na zapach neutrina ponieważ w jej produktach znajduje się elektron, jeśli tak to po prawej stronie równania musi znajdować się ν_e. Natomiast w drugiej reakcji przeżywa neutrino co oznacza, że liczba leptonowa zawsze będzie zachowana i w konsekwencji może brać w niej udział dowolne z trzech neutrin. Rozpoznawanie pierwszej reakcji polega na obserwacji elektronu o określonej energii natomiast w drugim przypadku powstający neutron jest wychwytywany przez jądro chloru z towarzyszącym temu błyskiem światła. Trzecim i ostatnim zjawiskiem zachodzącym w SNO jest wyłapywanie Z^0 przez elektron. Wcześniej ten bozon jest produkowany przez dowolne, lecące neutrino w ośrodku.

Założeniem oscylacji neutrin jest to, że posiadają one masę. Teoria mówi, że są one obserwowane jako stany własne zapachów ale poruszają się w przestrzeni jako superpozycje stanów własnych operatora masy. Jeżeli miałyby one różne masy to ich częstość byłaby różna, czyli w trakcie propagacji przestrzennej dochodziłoby do przesunięcia fazowego. Innymi słowy, trzy stany własne oddziaływań słabych $\nu_e\nu_\mu\nu_\tau$ powstają z połączenia trzech stanów masowych $\nu_1\nu_2\nu_3$, które to przesuwają się względem siebie w zależności od odległości jaką przebyły. Z tego wynika, że stosunek ilościowy pierwotych obserwabli ($\nu_e \ldots$) również zmienia się w czasie.

Przejście pomiędzy stanami zapachów a masowymi realizowane jest unitarną transformacją, której parametrem jest kąt mieszania θ. Poniższe rozwiązanie dla uproszczenia zakłada tylko dwa typy neutrin.

$$\begin{pmatrix} |\nu_e\rangle \\ |\nu_\mu\rangle \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} |\nu_1\rangle \\ |\nu_2\rangle \end{pmatrix}$$

W chwili $t = 0$ występują tylko neutrina ν_e, ale ponieważ propagację w czasie może opisywać funkcja falowa

$$\left|\nu_{(t)}\right\rangle = e^{i\bar{p}\bar{x}} \left(\cos\theta \cdot e^{-iE_1 t} \left|\nu_1\right\rangle + \sin\theta \cdot e^{-iE_2 t} \left|\nu_2\right\rangle\right)$$

to istnieje niezerowe prawdopodobieństwo przejścia do innego stanu wyrażone wzorem

$$P(\nu_e \to \nu_\mu) = \left|\langle \nu_2 | \nu_t \rangle\right|^2$$

$$P(\nu_e \to \nu_\mu) = \sin^2 2\theta \cdot \sin^2\left(\frac{\Delta m^2}{4p} t\right)$$

Rozdział 2

Eksperyment T2K

Poniższy rozdział zawiera podstawowe informacje z dziedziny neutrin, przodujących doświadczeń i współczesnych laboratoriów. Opisy będą punktem wyjścia do podjęcia głównego celu tej pracy, jakim jest generowanie neutrin dla NuWro. Tłem jest wielki eksperyment, a właściwie grupa eksperymentów przeprowadzanych na terenie Japonii o nazwie T2K. Na kolejnych stronach przedstawiona zostanie budowa kompleksów laboratoryjnych wchodzących w skład T2K oraz symulatorów fizycznych Neut i NuWro.

2.1. Super-Kamiokande

Nazwa doświadczenia T2K oznacza „od Tokai do Kamiokande" (zob. rys. 2.1). Pierwszy człon odnosi się do miasta Tokai, gdzie wybudowano generator neutrin J-PARC, a drugi do góry Kamiokande,pod którą, w starej kopalni Mozumi, znajduje się największy na świecie detektor neutrin: Super-K. Jest to ogromny walec o wysokości ok. 50 metrów, wypełniony doskonale czystą wodą, na którego ścianach od wewnętrznej strony umieszczonych jest 11200 fotodetektorów. Takie „lampy" wykrywają wpadające weń światło - obserwowane fotony to efekt promieniowania Czerenkowa. Zjawisko to zachodzi, gdy elektrycznie naładowane cząstki o wysokiej energii, przebiegając przez dielektryk wywołują miejscową polaryzację wzdłuż swojego toru lotu. Naturalnie takie wzbudzone cząsteczki ośrodka pozbywają się nadwyżki energii wypromieniowując fotony.

Fizyka jest nauką przybliżeń i często naukowcy dokonują pewnych założeń - także i tutaj znajdziemy kilka problemów, z którymi musieli uporać się doświadczalnicy. Pierwszym z nich jest fakt, że elektrony mogą powstawać w wodzie również poprzez oddziaływanie z neutrinami innymi niż elektronowe. Dzieje się tak, gdy z reakcji materii z neutrinem powstanie bozon Z^0, który następnie przekaże energię do elektronu. Taki wzbogacony energią elektron zostałby wykryty i zaklasyfikowany jako wynik oddziaływania neutrina elektronowego. Jednak po dokonaniu szacunkowych wyliczeń okazało się, że taka sytuacja może zostać zaniedbana, gdyż jest wystarczająco mało prawdopodobna. Kolejny problem związany jest z rozpoznaniem, który z leptonów został wyprodukowany. Okazuje się, że elektrony oddziałują z ośrodkiem łatwiej niż miony. Pędzący elektron wytwarza w wodzie promieniowanie hamowania, które jest następstwem du-

Rysunek 2.1: Mapa Japonii z zaznaczonym miastem Tokai, gdzie znajduje się laboratorium J-PARC,oraz górą Kamiokande, pod którą, w starej kopalni, utworzono największy wyłapywacz neutrin na ziemi.

żej ilości oddziaływań. W praktyce oznacza to, że obraz światła, który otrzymujemy, jest rozmyty. Natomiast mion przelatuje przez wodę „cicho" - obserwowany stożek świetlny ma ostre brzegi. Ostatecznie nie można wykluczyć, iż fotodetektory wykrywają zbłądzone elektrony innego pochodzenia. Jednak sterylne środowisko stworzone głęboko pod ziemią skutecznie niweluje możliwość publikowania błędnych teorii. Inną niedogodnością tej metodologi jest brak możliwości rozróżnienia ładunków elektronu i mionu, co w konsekwencji skutkuje jednakowym traktowaniem zdarzeń neutrinowych i antyneutrinowych.

Pierwotna wersja laboratorium nosiła nazwę Kamiokande i była 10 razy mniejsza od aktualnej. Skonstruowane zostało w celu udowodnienia hipotezy rozpadu protonu, niestety nie udało się tego dokonać, badania są kontunuowane. Dużym sukcesem była obserwacja neutrin pochodzących z supernowej z Wielkiego Obłoku Magellana, wtedy podjęto decyzję o rozbudowaniu urządzenia. Nowa wersja wykrywacza nosząca nazwę Super-K jest na tyle czuła, że pozwala na obserwację neutrin pochodzących ze słońca, atmosfery i sztucznie tworzonych w laboratorium J-PARC. Obserwacje neutrin słonecznych i atmosferycznych, były pierszymi i pozwoliły udowodnić tezę, że neutrina oscylują a co za tym idzie mają masę.

2.2. J-PARC

W środkowo-wschodniej części Japonii znajduje się centrum J-PARC (Japan Proton Accelerator Research Complex) w którym, jak nazwa wskazuje, mieści się akcelerator rozpędzający protony. Synchrotron z maksymalną mocą do 50GeV rozpędza protony i ukierunkowuje je na blok grafitowy. W trakcie kolizji dochodzi do powstawania pionów które samoistnie rozpadają się na miony i neutrina mionowe. Istnieje drugi kanał rozpadu do elektronu ale prawdopodobieństwo wynosi zaledwie 0.0001. Główna część wiązki, zwana on-axis, jest ukierunkowana na detektory ND280 i Ingrid (rozdz.2.3 i 2.4) które tworzą statystyki wyjścia. Natomiast linia off-axis prowadzi do Super-K i jest ona odchylona o 2.5^o. Poprzez wspomniane odsunięcie detektora na bok dociera doń wiązka o lepszym spektrum energetycznym, czyli takim gdzie większość neutrin posiada energię < 1GeV dla której przewiduje się największą oscylację.

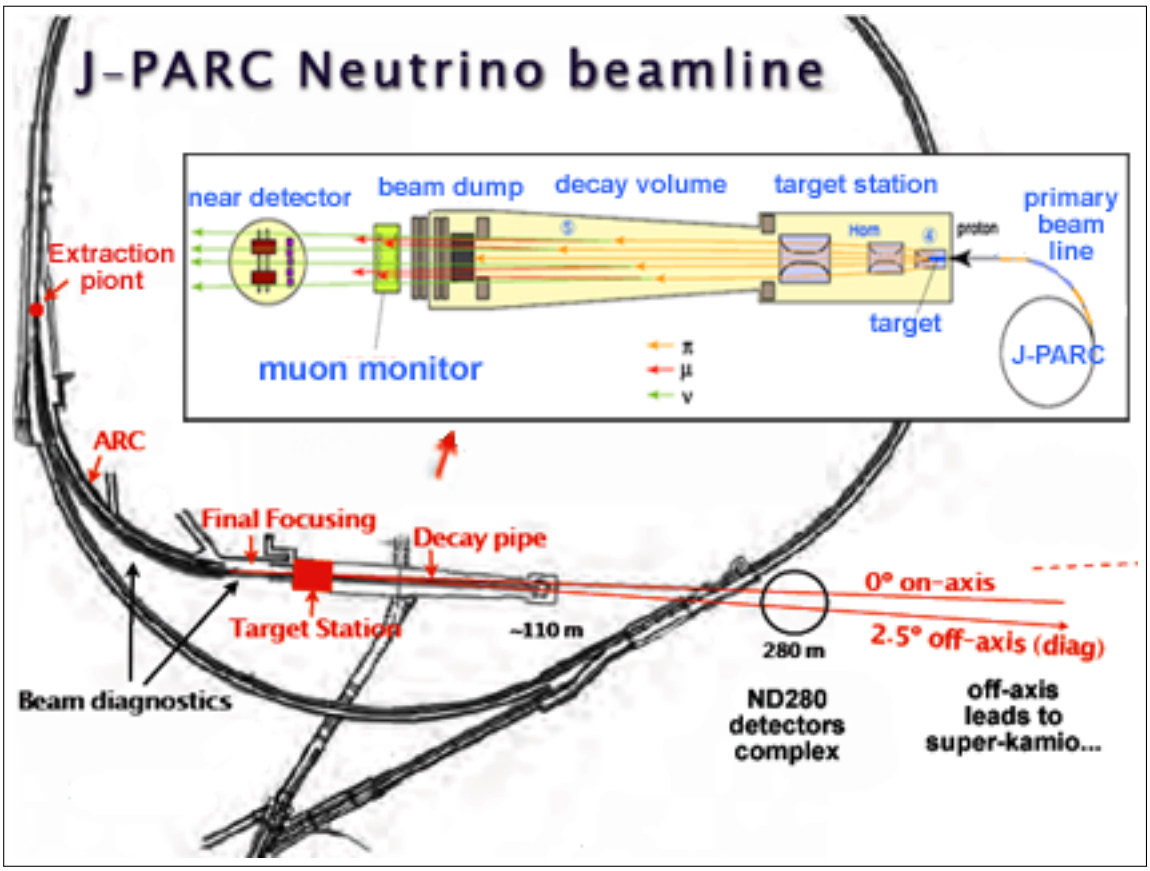
Pierścień na całej długości składa się z elektromagnesów które przekazują energię i sterują wiązką protonów. Na rysunku 2.2 jest zakrzywienie oznaczone ARC, to newralgiczne miejsce synchrotronu w którym protony mają maksymalną prędkość i dodatkowo ich tor lotu musi zostać mocno zakrzywiony, te dwa czynniki powodują, że inżynierowie zastosowali elektromagnesy nadprzewodnikowe. Cewki zrobione są z kabla niobowo-tytanowego, umocowane za pomocą niemagnetycznych klamer przytwierdzonych do żelaznego jarzma. Magnes schłodzony jest do temperatury ciekłego helu czyli około -269^oC. Rura izolowana jest próżnią i folią odbijającą promienie cieplne. Ta ogromna inwestycja pozwoliła na osiągnięcie 50GeV dla wiązki.

2.3. Bliski detektor ND280

Część zwana ND280 to kompleks precyzyjnie ułożonych elementów pomiarowych wychwytujących cząstki oddziałujące z neutrinami. Jak nazwa wskazuje jest on bliskim detektorem, oddalonym o 280 metrów od końca komory rozpadu, miejsca gdzie powstają neutrina i wylatują poza synchrotron.

Przejdźmy do omówienia konstrukcji detektora bazując na rysunku 2.3. Strumień wpada od lewej strony i wylatuje po przeciwnej, tam gdzie jest napisane „Downstream ECAL". DS-Ecal, Barrel Ecal i P0D Ecal to wszystko to są kalorymetery z płyt z tworzywa sztucznego służące do badania promieniowania jonizującego (scyntylatory). Każda z płyt oddzielona jest ołowianą folią. Otaczają one główne elementy które mają za zadanie oddziaływać z cząsteczkami, jednym z produktów tych oddziaływań jest promieniowanie γ. To promieniowanie mierzy ECAL, jest ono bardzo istotne do rekonstrukcji rozpadu π^0. Kolejne moduły to:

- Magnet: ND280 używa magnetu UA1, używanego oryginalnie w CERNie a później przewiezionego do Japonii, który wytwarza pole magnetyczne o wartości 0.2 T. Ta część detektora, a dokładniej wytwarzane pole, jest potrzebna do badania pędu cząstek naładowanych w polu magnetycznym. Rozmiary okowy od wewnątrz to 3.5 m $\times$ 3.6 m $\times$ 7 m.



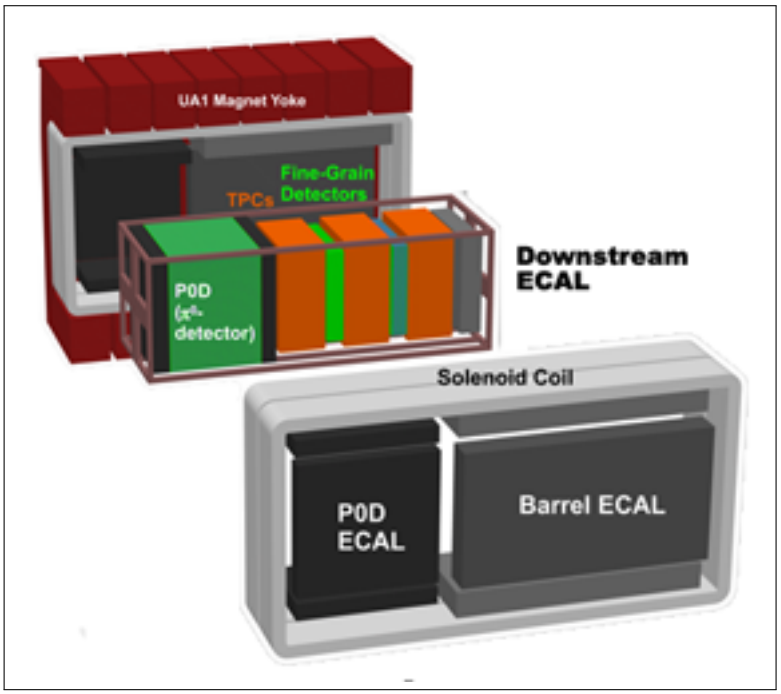
Rysunek 2.2: Kompleks J-PARC, generator i rozpędzacz neutrin

- Tracker: Detektor zoptymalizowany do badania pędu naładowanych cząstek, takich jak miony i piony.
  - TPCs, Time Projection Chambers: trzy komory wypełnione gazem i odgrodzone od siebie płytami wytwarzającymi pole elektrostatyczne. Dodatkowo linie pola magnetycznego są równoległe z elektrycznymi w celu zminimalizowania dyfuzji (rozbiegania się) elektronów powstałych podczas jonizacji. Cząstka wpadająca w gaz powoduje jego jonizację wzdłuż toru lotu. Badania w TPC pozwalają na opisanie spektrum energetycznego neutrina, ponadto śledzenie cząstki plus jej rozkład dE/dx pozwala na określenie ładunku i identyfikację czy jest to miuon, pion czy elektron.
  - FGDs, Fine Grained Detectors: Dwa moduły FGD znajdują się pomiędzy trzema TPC i są to opisywane wcześniej scyntylatory. Urządzenia służą do badania krótkich ścieżek protonów które powstają w oddziaływaniach z ładunkiem (CC). Dwa moduły FGD różnią się konstrukcją, pierwszy jest tylko plastikowy a drugi plastikowo-wodny do badania przekroju czynnego na wodzie.
- P0D, π^0 Detector: Detektor pionów zero umieszczony jest na początku urządzenia, jako pierwszy oddziałuje z wiązką. Jest największym z wewnętrznych elementów.P0D również jest scyntylatorem i część jego sekcji wypełniona jest wodą, podobnie jak w FGD.
- SMRD, Side Muon Range Detector: Są to elementy ulokowane pomiędzy pierścieniami magnetu, służy do mierzenia energii „large angle muons (co to jest??)" które powstają wewnątrz urządzenia. SMRD w połączeniu z danymi o promieniowaniu kosmicznym służy również do kalibracji ND280.

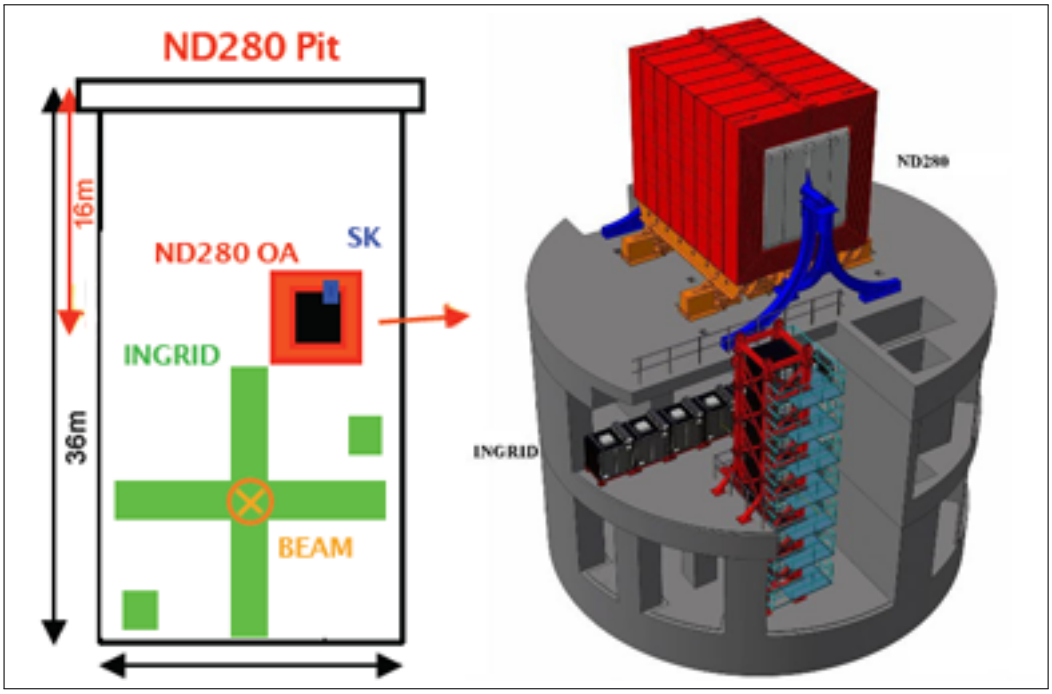
2.4. Ingrid

W podziemnej budowli o kształcie walca umieszone zostały dwa detektory, patrz. rys. 2.4 pierwszy ND280 i drugi INGRID. Ponieważ ND jest umieszczony na zewnątrz głównego promienia neutrin to nie może on służyć do weryfikowania kierunku wiązki. Dodatkowo kąt off-axis nie jest stały w trakcie eksperymentu. Zmiany wynikają z wielu czynników, gdzie dwa główne to dokładność toru lotu protonów w synchrotronie lub horn allignment. Ten drugi to proces nastawiania tarczy umieszczonej pomiędzy komorą rozpadu a synchrotronem. Wiązka ma badany kierunek i położenie przed powstaniem neutrin, czyli lokalizowane są hadrony, oraz po, przez detektor INGRID. Leżący na linii strumienia (on-axis) detektor daje możliwość bezpośredniego mierzenia kierunku wiązki. Niedokładność pomiaru wynosi 1 mili radian, co odpowiada 2% przesunięcia dla czubka (max) energii neutrin.

Po lewej stronie, na dwuwymiarowej części rysunku 2.4, widać przesunięcie dwóch detektorów. INGRID leży na osi czoło akceleratora - INGRID - Super Kamiokande. Detaktor ma kształa krzyża i składa się z 16 identycznych modułów, pokrywa obszara



Rysunek 2.3: Schemat detektora ND280



Rysunek 2.4: Detektory INGREAD i ND280

do 5m od środka strumienia. Każdy z elementów to przekładaniec z płyt stalowych (najgrubsza na przedzie) i scyntylatorów. Naładowane cząstki inne niż miony, takie jak protony i piony powstające przy oddziaływaniach nie mogą zostać wykryte przez urządzenie gdyż są zatrzymywane przez stalowe płyty, czyli jest to kolejny detektor neutrin.

2.5. ROOT

Ilość neutrin wychwytywanych w Super-Kamiokande zależy od eksperymentu, generalnie nie przekracza 16000 na dobę. [1] Cały eksperyment posiada wiele detektorów, część z nich została opisana w poprzednich akapitach, można założyć, że w każdym z nich zapisywana jest podobna ilość danych. Ponieważ w pracę badawczą zaangażowanych jest tysiące ludzi i odbywa się wiele eksperymentów to pojawia się problem ze sprawnym gromadzeniem i obrabianiem dużych ilość informacji. Jest to specyfika pracy naukowej związanej z wysokimi energiami gdzie dominuje statystyka. Z pomocą przychodzi, lubiany przez doświadczalników, program o nazwie ROOT (nie mylić z nazwą unixowego konta administratora).

Aplikacja służy do gromadzenia i szybkiej obróbki dużych ilości danch. Wszystkie informacje zapisane w pliku mają strukturę drzewiastą a każdy element w niej zapisany jest dowolnie definiowaną strukturą. Dostęp do elementów przypomina operowanie na bazie danych. Użytkownik za pomocą dostępnego interfejsu jest w stanie nałożyć ograniczenia na informacje a następnie przedstawić je graficznie lub w postaci tekstowej. Przedstawienie graficzne może oznaczać narysowanie wykresu lub histogramu. Przykładowe polecenie rysujące wykres sinusa

```
root [1] TF1 * fun = new TF1("fun", "sin(x)", 0, 10);
root [2] f1->Draw();
root [3] _
```

gdzie fun wskaźnik na obiekt klasy $TF1$ (w C++) reprezentującej jednowymiarową funkcję z określonym początkiem i końcem. Po wywołaniu metody $Draw()$ zostanie narysowany $\sin x, x \in < 0, 10 >$.

Innym sposobem pracowania z danymi w roocie jest zobaczenie ich w postaci tekstu na konsoli. Przykładem jest zrzut konsoli widoczny w rozdziale 3 na rysunku 3.2. Na górze screenu widoczna jest komenda

```
root [4] h3002->Scan(''nnu[0]:nnu[1]:nnu[2]:xnu:ynu:Enu'');
```

która każe rootowy wyświetlić konkretne dane w kolumnach. Każdy wiersz to zbiór parametrów opisujących jedno neutrino złapane w eksperymencie T2K na detektorze ND280.

ROOT jest systemem otwartym który może być dowolnie używany i modyfikowany (licencja LGPL). Został zaprojektowany tak, aby być rozszerzalnym o kolejne moduły

[1] Neutrino bardzo słabo oddziałuje z materią przez co powstające w procesie rozpadu beta może przeniknąć przez zbiornik wodny o długości około 1000 lat świetlnych zanim zostanie zaabsorbowane w odwrotnym rozpadzie beta, ten problem jest rekompensowany przez dużą ilość generowanych neutrin.

przez linkowanie go z zewnętrznymi bibliotekami. Ma wbudowany interpreter CINT C++ przez co może być sterowany za pomocą skryptów pisanych w tym języku. Ponadto jego shell bazuje na wspomnianym interpreterze więc wydawanie komend jest tak naprawdę programowaniem w języku C++.

Podsumowując ROOT to potężne narzędzie które ułatwia prowadzenie doświadczeń w T2K. Warto wspomnieć, że celem powyższej pracy jest napisanie modułu który wstrzykuje dane z plików root'owych do symulatora NuWro.

2.6. NuWro

Symulator ma za zadanie doprowadzić do oddziaływania neutrina z jądrem a następnie wyliczyć różniczkowe przekroje czynne i przedstawić produkty reakcji. Cały proces składa się z kilku etapów które przedstawione są na schemacie 2.6 i zgodnie z tym rysunkiem będą opisywane poszczególne procesy.

2.6.1. Zasady definiowania parametrów symulacji

W pierwszym kroku program wczytuje z linii komend wszystkie parametry opisujące doświadczenie.

(-i) nazwa pliku ze zbiorem parametrów, domyślnie params.txt

(-o) nazwa pliku wynikowego, domyślnie eventsout.root

(-p) nazwa_par=wartość, każdy z takich parametrów nadpisuje swój odpowiednik w params.txt, -p może być używany wielokrotnie

(-progress) parametr do doświadczeń online

Jak zostało to wspomniane, pobieranie danych które dokładnie definiują przebieg symulacji polega na wczytywaniu ich z odpowiednio przygotowanego pliku. Domyślnie nazywa się on params.txt ale możliwe jest przedefiniowanie jego nazwy przy pomocy flagi -i użytej z linii komend w trakcie uruchamiania NuWro.

Lista parametrów, które mogą być umieszczone i modyfikowane w pliku params.txt jest zdefiniowana w params.h w postaci makra #define PARAMS_ALL(). W pliku .txt modyfikujemy zmienne symulacji za pomocą dwóch operatorów, pierwszy to przypisanie wartości, znak równości a drugi to operator plus-równa się, który dodaje nową wartość do zmiennej. Oto schematy zapisu dla obu przypadków

nazwa_par=wartość1

nazwa_par+=wartość2

gdzie ostateczna wartość zmiennej *nazwa_par* jest sumą pierwszej i drugiej wartości. Kolejna dostępna operacja to wstawianie komentarzy, które są krytyczne dla plików z dużą ilością zdefiniowanych parametrów. Do tego używamy symbolu #

#komentarz który, z punktu widzenia piszącego, wyjaśnia wszystko

Możliwe jest również włączenie (include) zbioru parametrów z innego pliku, w tym celu używamy operatora @. Przykładowo w dziale *Beam specification*:

@beam/ND280.txt

ND280.txt jest jednym z grupy plików definiujących wiązkę, które umieszczone są one w projekcie w katalogu *beam*. Różnią się one od siebie zdefiniowanymi w nich parametrami co wynika z faktu, że generowanie każdego z typów wiązek bazuje na innej metodzie. Przykładowy plik ND280.txt zawiera dwie linie:

beam_type=4

beam_folder=../flux,

gdzie drugi parametr określa folder, z którego NuWro powinien pobrać dane, natomiast pierwszy może przyjąć jedną z wartości:

0 *beam_uniform*

1 *beam_mixed*

2 *BeamRF*

3 *BeamHist*

4 *Tworzenie histogramu*

Cyfry z przedziału $\langle 0, 3\rangle$ to kolejne rodzaje wiązek (patrz. rozdziały 2.6.4, ...). Liczba cztery jest tutaj wyjątkowa i definiuje proces tworzenia histogramu dla wiązki typu BeamHist. Oznacza to, że NuWro wchodzi do folderu flux poszukując rootowych plików T2K. Otwiera kolajno każdy z nich czytając wszystkie dostępne tam neutrina. Na ich podstawie symulator wylicza i zapisuje do pliku histout.txt wielowymiarowy histogram. Dla 50-ciu plików zajmuje to około jednej minuty. NuWro kończy czwartą procedurę i może zostać ponownie uruchomiony po ustawieniu *beam_type*=3.

2.6.2. Główne parametry symulacji

Pierwszą istotną zmienną w pliku params.txt jest

number_of_test_events=100000

określająca ilość oddziaływań wstępnych. Są one wykonywane w celu określenia dynamiki dostępnych dla określonych warunków. Innymi słowy NuWro przeprowadza wstępną symulację dla wszystkich zdefiniowanych dynamik sprawdzając która jest poprawna. Ostatecznie otrzymujemy histogram określający częstość występowania każdej z nich. Ta statystyka jest później używana w głównej pętli symulacji. Drugi główny parametr

number_of_events=500000

definiuje ile zdarzeń, pozytywnie przeprowadzonych w głównej symulacji, powinno być zapisanych w pliku wynikowym eventsout.root.

Użytkownik może określić które dynamiki nie będą dostępne w symulacji. W poniższym przykładzie zostały wyłącznone oba koherentne, z ładunkiem oraz neutralne:

- dyn_qel_cc =1 Quasi elastic charged current
- dyn_qel_nc =1 Quasi elastic neutral current
- dyn_res_cc =1 Resonant charged current
- dyn_res_nc =1 Resonant neutral current
- dyn_dis_cc =1 Deep inelastic charged current
- dyn_dis_nc =1 Deep inelastic neutral current
- dyn_coh_cc =0 Coherent charged current
- dyn_coh_nc =0 Coherent neutral current

Kolejna grupa parametrów to ustawienia form faktorów dla dynamik kwazi- elastycznych

- qel_vector_ff_set = 2 #BBBA05, hep-ex/0602017 BBBA05 for Q2¡18[GeV]
- qel_axial_ff_set = 1 #dipol

Użycie parametru

cc_smoothing=1

zapewni optymalizację ze względu na czas wykonywania doświadczenia, ponieważ zablokowane zostaną próby przeprowadzenia niemożliwej reakcji neutrina z protonem (i antyneutrina z neutronem). Za pomocą zmiennej

pauli_blocking= 1

możemy włączyć lub wyłączyć zakaz Pauliego. Parametr

beam_test_only

nie pozwala nuwro przeprowadzić symulacji do końca - po zaakceptowaniu cząstki jako gotowej do oddziaływania we wnętrzu detektora jest ona zapisywana i nie dochodzi do symulowania dynamiki.

kaskada_redo

pozala na użycie pliku wynikowego z poprzedniej symulacji, a następnie przeprowadzenie na nim kaskad.

2.6.3. Definiowanie tarczy

Kolejny etap parametryzowania symulacji to określanie tarczy. W pliku params.txt znajduje się dział *Target specification* w którym umieszczone są nazwy plików ze zbiorem definicji dla konkretnego rodzaju tarczy. Pliki z tej grupy znajdują się w folderze target.

@target/O.txt

zawiera najprostszy sposób definiowania tarczy, tutaj oczekuje się, że neutrina oddziaływują z jądrem tlenu. Wewnątrz pliku zdefiniowana jest ilość nukleonów

nucleus_p= 8 #number of protons

nucleus_n= 8 #number of neutrons

Naturalnym rozwinięciem opisanej metody jest możliwość definiowania proporcji poszczególnych nukleonów. Przykładem może być cząsteczka wody opisana w pliku h2o.txt

target_content= 1 0 2x

target_content+= 8 8 1x

[ilość protonów] [ilość neutronów] [wielokrotność]x

Wymienione tutaj opisy tarcz nie zawierają przestrzennego rozkładu materiału. Wczytanie trójwymiarowej geometrii detektora jest możliwe tylko przy użyciu plików roota. W praktyce używane są oryginalne definicje detektora pochodzące z doświadczenia T2K. Rysunek 2.5 przedstawia ogólny schemat detektora wykreślony na podstawie wspomnianych definicji detektora. Wyniki tej pracy sporządzone zostały na bazie *@target/ND280.txt*:

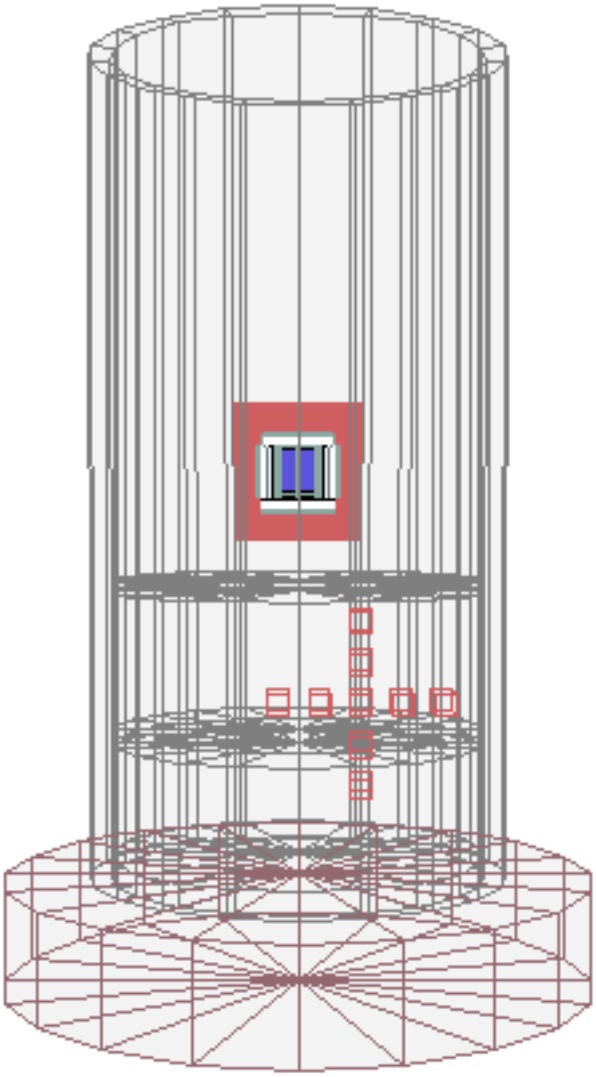
```
#ND280 geometry
target_type=2
geo_file=target/ND280.root
geo_name=ND280Geometry
geo_o = 0 0 0 //coordinates of the center of the box:ox/oy/oz
geo_d = 1500 1500 3200 //half dimension of the box:dx/dy/dx
nucleus_target=2
```

Plik ND280.root zawiera pełną geometrię detektora.

W kolejnych rozdziałach dowiemy się, że wzbogacenie Nuwro o możliwość rozpraszanie na konkretnej geometrii detektora, pozwoliło na pełne wykorzystanie nowego modułu generującego wiązkę neutrin. Jego zaimplementowanie było celem tej pracy.

2.6.4. Wiązka beam_uniform

Wiązka o nazwie beam_uniform jest pierwotną i zarazem najbardziej prymitywną ze wszystkich dostępnych w NuWro. Używa się jej w symulatorze definiując cztery parametry. Przykładem może być plik *@beam/ANL.txt*. Pierwszym zdefiniowanym tam parametrem jest



Rysunek 2.5: Całościowy schemat detektora ND280, wykreślony na podstawie plików root dostępnych na stronie labu. Widoczna jest konstrukcja ścian, detektory INGRID oraz off-axis. Kierunek i zwrot wiązki jest skierowany do obserwatora.

beam_type = 0

co pozwala symulatorowi zidentyfikować typ wiązki która zostanie mu przekazana. Drugi, najbardziej kluczowy parametr opisuje rozkład energii neutrin dla danej wiązki. Rozkład ten zapisany jest w postaci histogramu.

$$\text{beam\_energy} = 203 \quad 6100 \quad \underbrace{0.567 \quad 1.13 \quad 1.11 \quad 0.854 \quad \dots \quad 0.00151}_{\text{N}}$$

Innymi słowy NuWro otrzymuje ciąg liczb w którym dwie pierwsze to minimum i maksimum energii jaką może posiadać neutrino. Wszystkie kolejne liczby to wagi, inaczej wysokości słupków histogramu. Szerokość przedziałów dana jest wzorem

$$dE = \frac{E_{max} - E_{min}}{N}$$

Trzecim parametrem jest

beam_particle = 14

które przyjmuje kod PDG cząstki, w tym wypadku jest to neutrino mionowe. [2] Ostatni parametr to kierunek pędu neutrina, który w większości przypadków nie ma znaczenia i jest ustawiany tak aby wiązka leciała wzdłuż osi Z.

beam_direction = 0 0 1

2.6.5. Wiązka beam_mixed

Kolejną, bardziej rozbudowaną wersją wiązki jest beam_mixed, oparta na histogramie histogramie zapisywanym podobnie jak w przypadku beam_uniform. Tym razem jednak zaimplementowana została możliwość operowania na kilku rodzajach cząstek (kody PDG). Aby użyć tej wiązki należy ustawić beam_type równe 1 a zmienna beam_direction zadaje współrzędne kierunku wiązki. W tym przypadku parametr odpowiadający elementom histogramu nazywa się beam_content a nie jak poprzednio beam_energy. Poniższy przykład bazuje na pliku *@beam/newMB.txt*.

$$\text{beam\_content}= 14 \quad 93.8\% \quad 0 \quad 7200 \quad 2.272e-12 \quad 8.566e-12 \quad \dots$$

$$\text{beam\_content+=} \underbrace{-14}_{\text{A}} \quad \underbrace{6.2\%}_{\text{B}} \quad \underbrace{0 \quad 7200}_{\text{C}} \quad \underbrace{2.56e-12 \quad 5.671e-12 \dots}_{\text{N}}$$

Została tutaj zdefinowana wiązka złożona z neutrina i antyneutrina mionowgo (grupa A) przy czym ich udział procentowy to jedna liczba w grupie B. Kolejno zdefinowane są minima, maksima oraz wagi, analogicznie jak w przypadku beam_uniform. Histogramy dla kolejnych cząstek dopisujemy przy pomocy operatora + =. W wielu aktualnie dostępnych plikach można spotkać się z notacją, w której element procentowy dla pierwszej wiązki równy jest 100%. Jest to zapis poprawny lecz utrudniający weryfikację danch przez osoby trzecie.[3] Przykład:

[2] http://pdg.lbl.gov/mc_particle_id_contents.html

[3] Subiektywna opinia autora pracy

beam_content = 14 100% ...

beam_content += -14 6.27702% ...

beam_content += 12 0.0552426% ...

beam_content += -12 0.0577346% ...

2.6.6. Wiązka BeamRF

Nazwa wiązki BeamRF pochodzi od trzech słów beam, root i file. Ten sposób generowania neutrin jest aktualnie używany w NuWro i jest najlepszy ze wszystkich tam dostępnych.[4] Jego pierwotna implementacja zakładała pobieranie kolejnych cząstek z plików root, pochodzących z eksperymentu T2K, które NuWro następnie wykorzystuje w trakcie symulacji. Z czasem do metody został dodany algorytm akumulowania i losowania neutrin, [5] mający na celu zniwelowania problemu związanego z uzwględnianiem wag każdej z cząstek. Istotą problemu było to, że pliki z T2K mają zdefiniowaną wagę dla każdego neutrina opisującą prawdopodobieństwo jego wystąpienia, więc użycie elementu zależało od tego, czy test losowy zaakceptuje daną wagę. Takie rozwiązanie prowadziło do wielu niepotrzebnych odczytów z pliku.

Wybór wiązki BeamRF odbywa się w trzech krokach, pierwszy to ustawienie parametru typ

beam_type = 2

następnie podanie ścieżki do folderu w którym zawarte są pliki rootowe

beam_folder = ../flux

Trzeci krok jest opcjonalny i daje możliwość zawężenia przedziału plików w folderze które mają być brane pod uwagę przez NuWro w trakcie symulacji

beam_file_first = 1

beam_file_limit = 0

wartości 1 i 0 podane w przykładzie są ustawione domyślnie i oznaczają komendę „zacznij czytać od pierwszego pliku i idź do ostatniego dostępnego w folderze".

2.6.7. Wiązka BeamHist

BeamHist (z ang. beam based on histogram) to dwu-etapowy sposób generowania neutrin, wymagający ponownego uruchomienia aplikacji w obu fazach, z róznymi wartosciami parametru beam_type. W pierszym etapie, gdy beam_type jest równe 3, NuWro opisuje neutrina zawarte w plikach root w postaci siedmiowymiarowego histogramu. Plik wynikowy nazywa się histout.txt, a opis jego struktury znajduje się w dodatkach

[4] Dane z wynikami eksperymentów i porównania z innymi symulatorami dostępne są w rozdziale 4

[5] Algorytm zaimplementowany przez dr C. Juszczaka

(patrz B.1). Natomiast neutrina, na bazie których powstaje histogram, pochodzą z eksperymentu T2K. Podobnie jak w przypadku BeamRF należy podać ścieżkę do plików z danymi, przykładowo:

beam_folder = ../flux

Podczas testów okazało się, że otrzymywany histogram 5D zawiera dużą ilość nadmiarowych zer (macierz rzadka). Zaimplementowane algorytmy usuwania powtórzeń bez szkody dla wiązki skracają czas wykonywania symulacji i zmniejszają ilość rezerwowanej pamięci komputera.

Drugi etap to uruchamianie NuWro w normalnym trybie symulacji. Aplikacja, mając parametr beam_type = 4, oczekuje pliku wejściowego histout.txt w tym samym folderze.

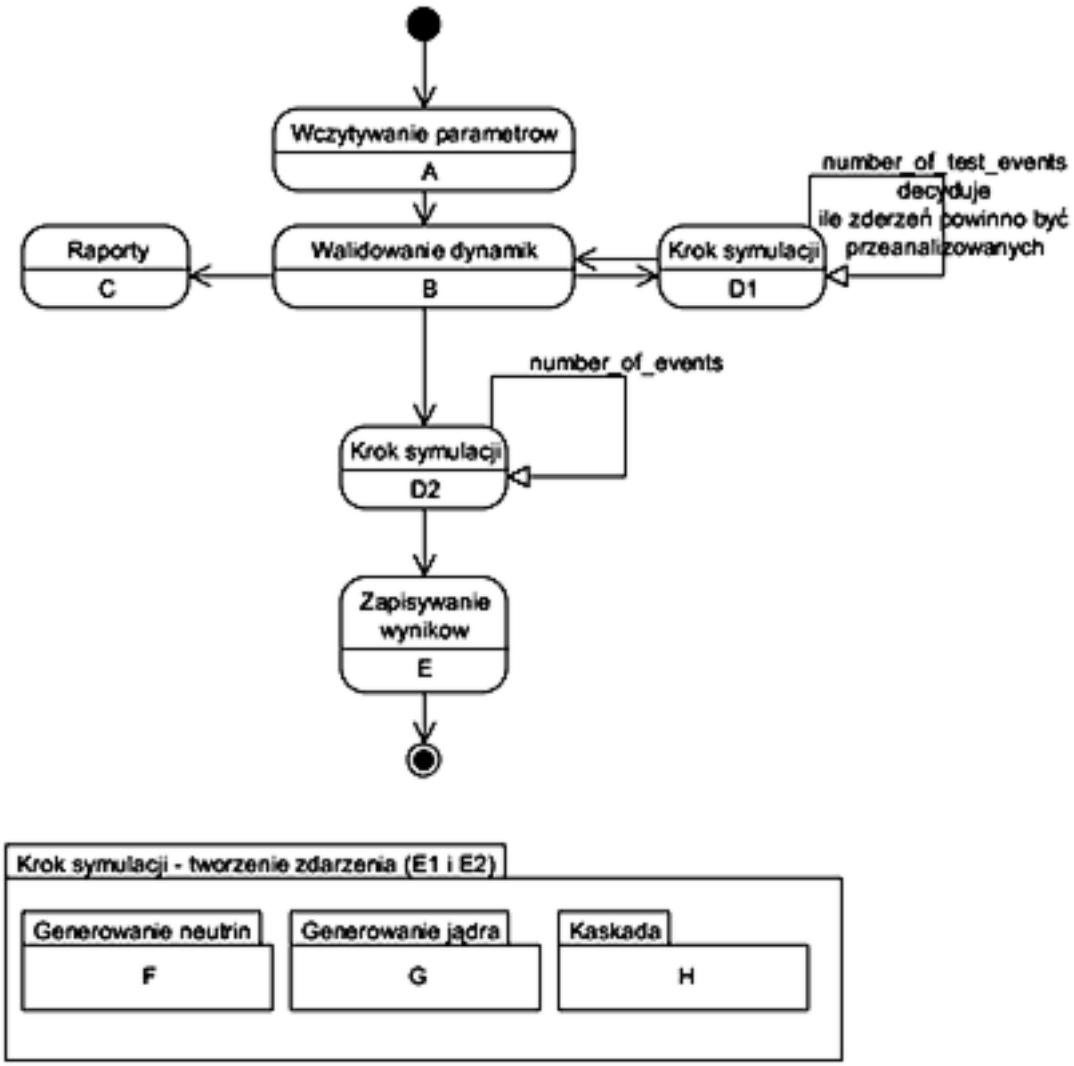
2.6.8. Pętla symulacji

W momencie, kiedy NuWro wczytał wszystkie parametry, przechodzi do wyznaczenia przekrojów czynnych dla każdej dynamiki i ilości zdarzeń jakie należy wygenerować. Polega to na przeprowadzeniu pewnej ilości oddziaływań (bez kaskady) i generowaniu statystyki o wagach z jakimi zachodzą wszystkie z dostępnych dynamik. W głównej pętli symulacji zdarzenia są akceptowane w prawdopodobieństwem proporcjonalnym do wagi zdarzenia:

generowanie neutrina → akceptacja lub ponowne generowanie neutrina → oddziaływanie neutrino-jądro → akceptacja z prawdopodobieństwem proporcjonalnym do wagi zdarzenia (czyli do przekroju czynnego) → kaskada → zapisanie neutrina do pliku wynikowego

Jeśli nie zostało to zanegowane za pomocą parametrów wejściowych, wówczas NuWro sprawdza czy powinien przeprowadzać kaskadę wewnątrz-jądrową. Polega to na tym, że produkty poprzedniej reakcji oddziałują z kolejnymi elementami w jądrze. Tak powstałe drzewo zamyka się w momencie przekroczenia minimalnego poziomu energetycznego.

Podejście do rozwiązania oddziaływań elastycznych i kwazielastycznych jest podobne do rozwiązania zastosowanego w Neucie, czyli użyty został formalizm Llewellyna-Smitha. Natomiast NuWro posiada ulepszoną wersję rozwiązania dla oddziaływań rezonansowych.



Rysunek 2.6: Diagram prezentujący główne grupy funkcyjne w NuWro

Rozdział 3

Implementacja wiązki neutrin

NuWro[1] jest symulatorem typu Monte Carlo, który wykonuje rozproszenia neutrin na jądrze atomowym. Do przeprowadzenia symulacji potrzebuje wiązki (ang. beam), którą stanowi pojedyńcze neutrino lub ich grupa oraz tarczy (ang. target), czyli pojedyńczego nukleonu lub większej struktury atomowej, w które taka wiązka celuje.

W tym rozdziale zostaną opisane dwa podejścia do generowania wiązki, które są używane w NuWro:

sekwencyjny odczyt neutrin z plików

generowanie neutrin na podstawie histogramu

Kolejność opisu powyższych metod wynika z kolejności faz projektowania i implementacji. Rozdział ten poświęcony jest przedstawieniu idei leżących u podstaw każdej z tych metod oraz ich technicznej stronie.

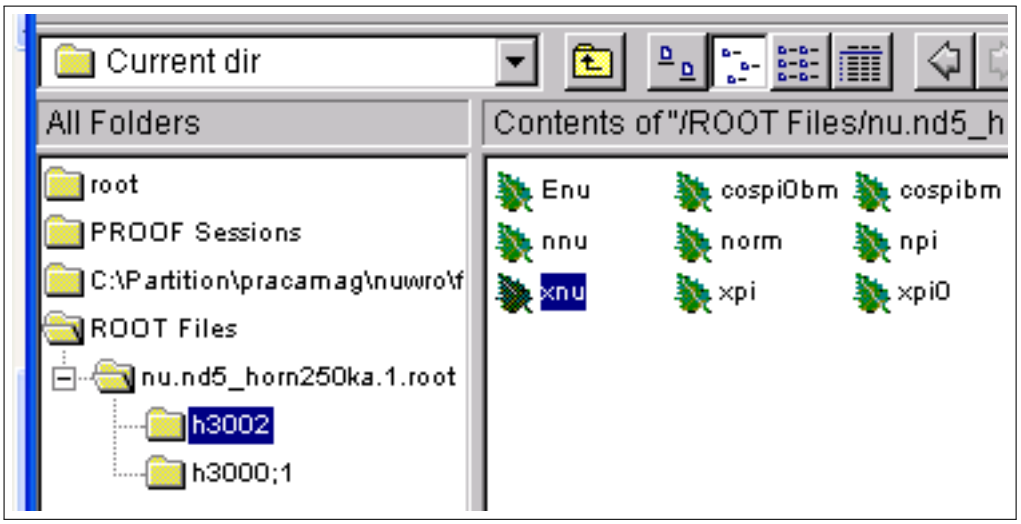
3.1. Zapis wartości fizycznych

Do utworzenia neutrina w pakiecie NuWro konieczna jest znajomość jego zapachu (kod PDG), trzech składowych pędu oraz (jeśli rozpraszanie ma miejsce na detektorze) trzech współrzędnych położenia punktu z którego neutrino startuje. W praktyce wystarczające jest otrzymanie dwóch współrzędnych położenia czyli dowolnego punktu znajdującego się na torze neutrino. W plikach T2K nie występuje z-etowa składowa położenia.

Podsumowując: mamy trójwymiarowy pęd neutrin i ich dwuwymiarowe położenie. Struktura w kodzie NuWro, która opisuje cząstkę (jest to część symulatora, nie wtyczki), wygląda w następujący sposób:

```
// elementary particle on a mass shell is represented
// as a lorentz fourvector and a pdg number.
class particle : public vect
{
  double _mass;      // On shell mass.
public:
```

[1] NuWro można pobrać ze strony http://borg.ift.uni.wroc.pl/nuwro/



Rysunek 3.1: Struktura danych T2K przedstawiona w graficznym interfejsie roota

```
  vect r;              // Position and time
                       // relative to the centre of
                       // the Nucleus event start time.
  int pdg;             // PDG code of the particle.
  ...
};
```

Klasa *particle* ze względu na samo dziedziczenie z czterowymiarowego wektora, składa się z energii i trzech współrzędnych pędu.

$$\vec{p}(t,x,y,z), \quad \text{gdzie } t \text{ jest energią.}$$

Dodatkowo obiekt klasy *particle* przechowuje położenie w zmiennej r, która jest wektorem trójwymiarowym.

3.2. Odczyt neutrin z plików

Rejestracja na stronie www.t2k.org umożliwia dostęp do plików w formacie .root z gotowymi do użycia neutrinami. Jeden rodzaj wartości, przykładowo x-ową składową położenia, można wyobrazić sobie jako plik z kolumną danych, które zachowują kolejność tak aby odpowiadać jednej cząstce (rys.3.1). Położenie na osi X znajduje się w wewnątrz rootowym folderze

$$nu.nd5\_horn250ka.1/h3002/xnu/$$

gdzie h3002 jest rootowym obiektem typu TTree, natomiast xnu to obiekt klasy TBranch. W programie root (rys. 3.2) widocznych jest 6 kolumn. Nazwa każdej z nich reprezentuje jedną ze zmiennych. Przykładowo X-ową składową pędu jest nnu[0]. Pozostałe zmienne przedstawione są w tabeli 3.1. Pęd całkowity wyraża się wzorem

```
root [4] h3002->Scan("nnu[0]:nnu[1]:nnu[2]:xnu:ynu:Enu");
*****************************************************************************************
*    Row   *    nnu[0] *    nnu[1] *    nnu[2] *       xnu *       ynu *       Enu *
*****************************************************************************************
*        0 * -0.005287 * -0.004780 * 0.9999746 * -20.81812 * 47.774734 * 0.3143146 *
*        1 * -0.005178 * -0.005605 * 0.9999709 * -18.89467 * 33.133956 * 0.7506660 *
*        2 * -0.000706 * -0.031512 * 0.9995031 * -88.09220 *  53.93536 * 0.0629788 *
*        3 * 0.0046490 * -0.029786 * 0.9995454 * 63.202888 * 102.73848 * 0.1464388 *
*        4 * -0.006941 * -0.025711 * 0.9996452 * -109.8935 * -45.73457 * 0.4224793 *
*        5 * -0.005845 * -0.027382 * 0.9996079 * -82.37426 * -87.71368 * 1.0052356 *
*        6 * -0.008545 * -0.031027 * 0.9994819 * -118.8887 * 28.116102 * 0.0552103 *
*        7 * -0.002991 * -0.033755 * 0.9994256 * 37.048713 * -48.52777 * 0.1314591 *
*        8 * -0.002750 * -0.031512 * 0.9994995 * -73.88336 * -91.84289 * 0.3545358 *
*        9 * 0.0014465 * -0.032728 * 0.9994632 * 40.890102 * -125.1140 * 0.8323459 *
*       10 * -0.005784 * -0.033699 * 0.9994153 * -46.65221 * -121.8065 * 0.0728443 *
*       11 * -0.002600 * -0.031302 * 0.9995065 * 21.140239 * -70.72806 * 0.1698602 *
*       12 * 0.0027872 * -0.028463 * 0.9995909 * 130.66537 * 0.6048320 * 0.0759842 *
```

Rysunek 3.2: Część danych w plikach roota dla bliskiego detektora zwanego ND280

Nazwa zmiennej	Opis
PDG	Międzynarodowe oznaczenie cząstki. Używane tylko do weryfikacji
nnu[0]	X-owa składowa pędu z wektora unormowanego
nnu[1]	Y-owa składowa pędu z wektora unormowanego
nnu[2]	Z-owa składowa pędu z wektora unormowanego
xnu	X-owa składowa położenia
ynu	Y-owa składowa położenia
Enu	Wrtość energii kinetycznej neutrina
norm	norma określająca częstość występowania cząstki

Tabela 3.1: Dane pobierane z plików root'owych

$$\vec{p} = (nnu[0], nnu[1], nnu[2]) \cdot Enu$$

a położenie

$$\vec{r} = (xnu, ynu, 0).$$

Na żądanie NuWro wtyczka pobiera kolejne neutrino z pliku i formatuje do struktury używanej w symulatorze. W międzyczasie kontroluje, czy dotarliśmy do ostatniego pliku w podanym folderze. Jeśli tak - wysyłana jest notyfikacja o ponownej pętli. Pobierane cząstki nie są w żaden sposób modyfikowane po wyciągnięciu ich z pliku, a zatem każda kolejna pętla wprowadza do doświadczenia te same elementy. W dodatku D.1 dostępne są najważniejsze fragmenty kodu w języku C++, które odpowiadają za odczyt i wprowadzanie neutrin do NuWro.

Biblioteka root udostępnia szereg funkcjonalności ułatwiających korzystanie z plików jej formatu. Po pierwsze daje możliwość otwarcia pliku jako zasobu w postaci wskaźnika do obiektu TFile. Konstruktor oczekuje nazwy pliku ze ścieżką względną do pliku wykonywalnego NuWro.

```
fH = new TFile( fname.c_str() );
tree = static_cast<TTree*>( fH->Get(treename.c_str()) );
```

Przykładem nazwy pliku może być

../flux/nu.nd5_horn250ka.1.root.

Następnie ustalamy dostęp do drzewa (folderu root) w którym przechowywane są interesujące nas dane. Metodą Get, która oczekuje nazwy drzewa, pobieramy wskaźnik na obiekt TTree. W kolejnym kroku podajemy bibliotece root adres zminnych do których będą zapisywane interesujące nas wartości. Proces ten widać poniżej w funkcji o nazwie OpenBranches.

```
struct ND5Event
{
 Float_t Enu;
 UChar_t gipart;
 ...

 void OpenBranches( TTree * tree )
 {
  tree->SetBranchAddress( "Enu", &Enu );
  tree->SetBranchAddress( "gipart", &gipart );
  ...
 }
};
```

Dla każdego parametru, jak energia, kod PDG itd., podawana jest nazwa identyfikująca parametr w pliku root oraz adres pamięci pod który biblioteka zapisze wartość. Odczyt ilości zdarzeń, równiej ilości neutrin w pliku, dokonuje się przy użyciu metody GetEntries.

```
count = tree->GetEntries();
```

W momencie użycia tej funkcji zostaną zapisane: energia, trzy współrzędne kierunku pędu, dwie wsp. położenia, kod PDG oraz norma dla danego neutrina. Funkcja zwraca numer (count) identyfikujący aktualnie pobrane neutrino.

Należy zwrócić uwagę, że każda zmienna pobierana z pliku root ma ustaloną wielkość pamięci którą rezerwuje. Przykładowo energia jest typu Float_t i tak powinna zostać zadeklarowana. Zastąpienie typów zdefiniowanych w roocie innymi, przykładowo float'em używanym w C/C++ może doprowadzić do zafałszowania danych.

Odczyt wszystkich plików root z danego folderu dokonuje się przy użyciu POSIX'owej biblioteki dirent.h, która pozwala na wyciągnięcie nazw plików.

```
DIR * _dp = opendir( directory.c_str() );
dirent * dirp = readdir( _dp );
string name( dirp->d_name );
if(    name.find( string(".root") )  !=  string::npos    )
  ...
```

Pierwsza linia otwiera folder, druga tworzy obiekt opisujący pierwszy plik tam zawarty. Ponowne użycie tej funkcji przeskoczy do pliku następnego. Ostatnie linie odczytują i przechowują nazwę pliku, o ile jest to plik root.

Poniższe kilka linii kodu przedstawiają miejsce, gdzie konkretne neutrino wyciągnięte z pliku root i zapisane w obiekcie e jest przygotowywane do wprowadzenia do eksperymentu przez wewnętrzny obiekt NuWro p

```
Nd280Element e
particle p(14, 0.0);
...
p.r.t = 0;
double E = e.Enu * 1000;
p.r.x = e.xnu * 10;
p.r.y = e.ynu * 10;
p.r.z = 0;
p.t = E;
p.x = e.nnu[0] * E;
p.y = e.nnu[1] * E;
p.z = e.nnu[2] * E;
```

Liczba 14 w drugiej linii to kod PDG i zerowa masa. Następnie ustawiane są położenia i wartości czteropędu. Mnożenie przez 1000 i 10 wynikają z przeliczenia jednostek. Widoczne w kodzie przyrównanie poszczególnych składowych pędu do energii wynika z faktu, że NuWro oczekuje masy równej zero i prędkości światła równej jeden, czyli:

$$E^2 = p^2 \cdot c^2 + m^2 \cdot c^4, \quad c = 1, \quad m = 0,$$

$$\vec{p} = E \cdot \vec{n},$$

gdzie $\vec{n}$ to jednostkowy wektor kierunku.

Powyższa metoda generowania neutrin ma kilka słabych stron. Pierwsza jest związana z przymusem przechowywania dużej ilości danych. Druga to długi czas dostępu do

pliku, który spowalnia symulację. Trzecie uniedogodnienie to brak możliwości rozmywania danych wejściowych, co przy długich symulacjach powoduje powtarzalność zjawisk. To z kolei w najgorszym wypadku może doprowadzić do pojawienia się niechcianych artefaktów i zjawisk w wynikach symulacji.

3.3. Generowanie neutrin na podstawie histogramu

Ostateczna zaproponowana i zaimplementowana została metoda generowania cząstek na podstawie histogramu. Proces ten składa sie z dwóch etapów. Pierwszy to tworzenie histogramu na podstawie dostępnych plików rootowych. Drugi to generowanie cząstek powstałych przy użyciu takiego rozkładu. Aby ten ostatni był zrozumiały, jego opis zostanie przeprowadzony w dwóch krokach. Najpierw zajmiemy się algorytmem jenowymiarowym, który pozwala w przejrzysty sposób zilustrować istotę pracy z histogramem. Następnie rozszerzymy algorytm do przypadku 5-cio wymiarowego.

3.3.1. Tworzenie histogramu

Tworzenie histogramu odbywa się w dwóch etapach, przy czym oba przechodzą przez wszystkie dostępne pliki w wybranym folderze. Dane potrzebne do obliczeń pobierane są z plików root, które wygenerowano za pomocą symulatora NEUT w ramach projektu T2K.

Pierwszy etap polega na zebraniu danych statystycznych. Do stworzenia histogramu potrzebne są minimalne i maksymalne wartości dla każdego z parametrów opisujących neutrino: trzech składowych pędu oraz dwóch składowych położenia. Innymi słowy, histogram musi posiadać informację o tym jaki przedział wartości ma reprezentować dla danego parametru. Dodatkowo zliczane są użyte neutrina - parametr *count*.

W kodzie można znaleźć dwie dodatkowe stałe, które służą do weryfikacji danych, są to

gipart - reprezentujący kod PDG neutrina czyli 14

idfd - identyfikator detektora w T2K, w tym wypadku jest to ND280 który posiada numer 5

Drugi etap polega na przyporządkowywaniu pobranego neutrina z plików root/Neut do odpowiedniej kostki w 5-cio wymiarowej tablicy, przez co zwiększamy zmienną całkowitą w tym miejscu. Dla przykładu zakładamy, że położenie x-owe minimalne to 2 a maksymalne to 10. Zaimplementowany wymiar położenia indeksujemy od zera do czterech, czyli rozdzielczość tego wymiaru wynosi 4 (programista może powiedzieć, że tablica ma cztery elementy). Mając neutrino z położeniem x=7 wiemy, że trzeba powiększyć (inkrementować) element tablicy o indeksie 2. Tak samo wygląda to dla pięciu wymiarów z tą różnicą, że jednocześnie rozpatrujemy pięć indeksów.

$$I_x = \frac{(\alpha_x - a) \cdot N}{b - a}, \quad \alpha_x \in \langle a, b \rangle, \quad x \in \langle 1, 5 \rangle$$

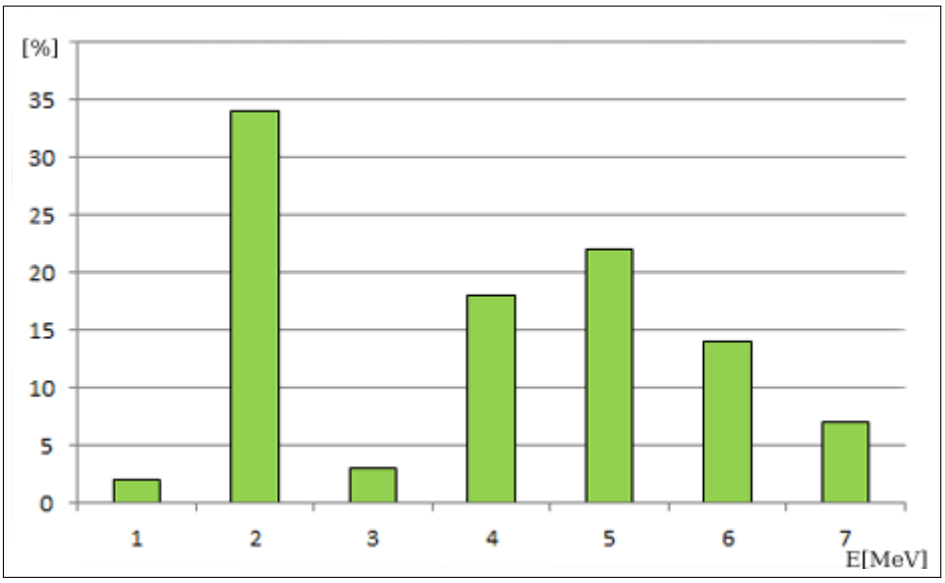
gdzie x jest jednym z wymiarów histogramu i jednocześnie identyfikuje konkretny parametr, przykładowo drugą składową kierunku pędu p_y. I_x to indeks w histogramie, którego wartość ma zostać podniesiona. α_x jest zmienną pobraną z pliku. N ilość neutrin. a, b to minimum i maksimum dla danego parametru.

```
unsigned int Nd280Statistics::Idx( cdouble & val,
cdouble & min, cdouble & max, cuint & count )
{
  double fraction = (val-min) * count;
  double scope = max - min;
  unsigned int result = static_cast<int>(fraction/scope);
  if( result >= count )
    result = count - 1;

  return result;
}

void Increment( Nd280Element x,  NArray< double > & hist )
{
  AdjustUnits(x);

  hist(
    Idx(x.xnu, _min.xnu, _max.xnu, hist.Count(0)),
    Idx(x.ynu, _min.ynu, _max.ynu, hist.Count(1)),
    Idx(x.nnu[0], _min.nnu[0], _max.nnu[0], hist.Count(2)),
    Idx(x.nnu[1], _min.nnu[1], _max.nnu[1], hist.Count(3)),
    Idx(x.Enu, _min.Enu, _max.Enu, hist.Count(4)))
        += x.norm;
}
```

Funkcja *(*Increment) zwiększa wartość elementu histogramu (5D macierzy) o normę rozpatrywanego neutrina. Natomiast metoda o nazwie *Idx* wykonuje obliczenia za pomocą wzoru przedstawionego powyżej listingu. Warunek logiczny, który występuje w kodzie *if(result >= count)* zabezpiecza granicę tablicy w pamięci komputera. Związane jest to z wartością neutrina lub grupy neutrin, które wyznaczają jedną z wartości maksymalnych. Obliczenie indeksu dla nich oznacza przekroczenie ilości elementów o jeden. *AdjustUnits* dokonuje przeliczenia jednostek dla każdego z parametrów. *hist* jest obiektem klasy *NArray*, która udostępnia funkcjonalności dla macierzy n-wymiarowej. Cały kod znajduje się w plikach

```
narray.h
makeHist.h
nd280stats.cc
nd280stats.h
beam.cc - funkcja CreateNewHistogram
```

zaś jego fragmenty w załączniku D.2.



Rysunek 3.3: Przykładowy histogram energii. Pionowa oś to procentowy udział pewnych energii a pozioma to konkretne przedziały energetyczne liczone w [MeV].

3.3.2. Tworzenie neutrin na podstawie histogramu jednowymiarowego

Za przykład jednowymiarowy posłuży tu rozkład energii przedstawiony na rysunku 3.3 (wartosci przedstawione na wykresie sa przykładowe i zostały dobrane tak, aby ułatwic zrozumienie używania histogramu w symulacji). Na dolnej osi mamy wartości energii w mega elektronowoltach a na pionowej procentowy udział każdego z przedziałów energetycznych. Przykładowo, w wiązce mamy 14% neutrin o energii 6 [MeV] i 2% takich które posiadają niską energię o wartości około 1 [MeV].

Naszym zadaniem jest losowe generowanie kolejnych cząstek zgodnie z podanym rozkładem. Przykładowo: po wystrzeleniu N neutrin chcemy otrzymać $14 \cdot N/100$ o energii równej $5[MeV]$.

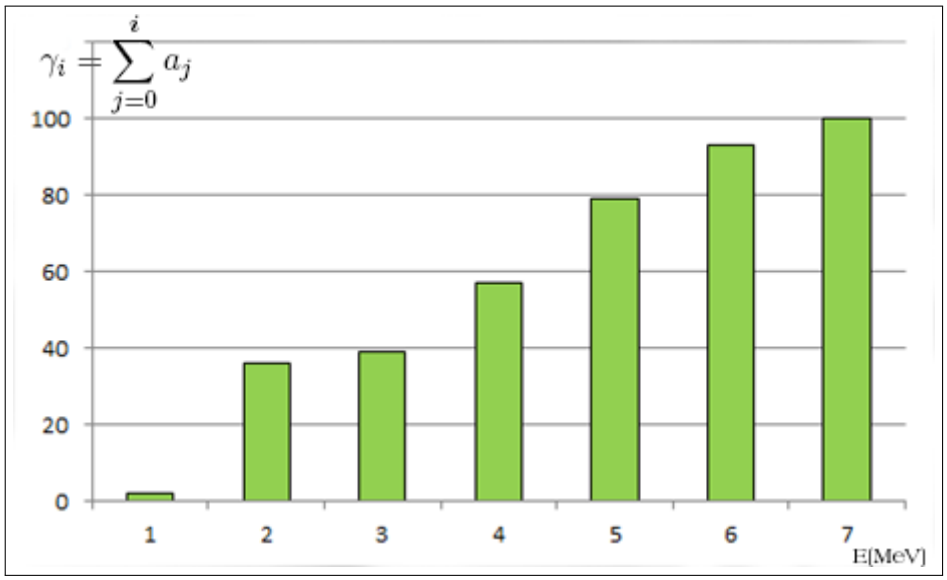
Rozpatrujemy histogram jednowymiarowy. W pierwszym kroku sumujemy wartości to znaczy wysokości sąsiadujących ze sobą słupków tak, że do następnego dodajemy wartość poprzedniego (rys. 3.4). Zakładamy, że a_j będzie oznaczało j-ty słupek procentowy z rys. 3.3. Natomiast γ_i będzie docelowym i-tym słupkiem z rys. 3.3, to:

$$\gamma_i = \sum_{j=0}^{i},$$

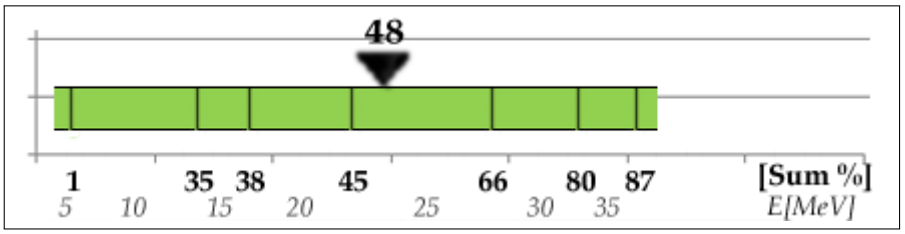
czyli

$$\gamma_0 = (0+2) = 2, \quad \gamma_1 = (2+34) = 36, \quad \gamma_2 = (36+3) = 39, \ldots$$

aż na końcu otrzymujemy liczbę 100 która jest sumą wszystkich wysokości słupków, przy czym liczba sto została tu podana wyłącznie w charakterze przykładu, w rzeczywistości może być to dowolna wartość. Każdy z nowo powstałych słupków niesie za sobą przedział, dokładniej mówiąc wagę swojego odpowiednika o tym samym indeksie z oryginalnego histogramu - szczegółowo rozwinięte zostanie to w dalszej części pracy.



Rysunek 3.4: Modyfikacja histogramu z rys. 3.3, gdzie kolejny słupek jest sumą siebie i poprzednika.



Rysunek 3.5: Słupki histogramu ułożone poziomo. Liczba 48 jest losową dla której będzie szukana wartość parametru (np. E) odpowiadająca danemu słupkowi

W kolejnym kroku algorytmu losujemy liczbę z przedziału $\langle 0, 100)$. Załóżmy, że tą liczbą jest 66 (jak 66th interstate). Teraz wykonujemy trzeci krok, skaczemy po wszystkich słupkach od końca i sprawdzamy (rys. 3.4), który jest ostro mniejszy od wylosowanej wartości 66. Jak widać na wykresie jest to słupek o indeksie 4. Ostatnim krokiem algorytmu jest sprawdzenie, jaka energia kryje się pod tym indeksem na oryginalnym histogramie, patrz 3.3, w naszym przykładzie jest to $18[MeV]$.

Tworzenie nowego neutrina można również zobrazować rysunkiem 3.5. Przedstawione tam słupki ułożone poziomo sumują się tworząc prostą z przedziałami. Koniec każdego przedziału jest sumą jego oraz wszystkich poprzednich elementów. Każdy przedział to waga z jaką występuje jej przypisana wartość. W celu wygenerowania pewnej wartości należy wylosować liczbę x z przedziału

$$x \in < 0, \gamma)$$

gdzie γ to suma wszystkich wag. Następnie idąc za pomocą bisekcji odnajdujemy przedział i przypisaną do niego wartość.

3.3.3. Macierz n-wymiarowa

Przedstawiony wyżej algorytm dla jednowymiarowego histogramu może zostać zaadoptowany do histogramu 5-cio wymiarowego. Aby to stwierdzenie było prawdziwe, muszą być spełnione dwa założenia. Pierwsze, że istnieje jednoznaczne jednowymiarowe odwzorowanie macierzy 5-cio wymiarowej. Drugie, że histogram jest nieczuły na taką transformację, czyli zachowuje wszystkie swoje właściwości pomimo traktowania go jako tablicę jednowymiarową zamiast wielowymiarowej.

Poniżej rozpisana jest jedna z metod przedstawienia macierzy 5-cio wymiarowej, zwanej tutaj α, jako jednowymiarowej tablicy β. Tak zostało to zaimplementowane we wtyczce.

$$\begin{aligned} \alpha_i \quad &\Rightarrow \quad \beta_I \\ i = a_1 \times a_2 \times \ldots \times a_5 \quad &\Rightarrow \quad I = 1 \times (a_1 \cdot \ldots \cdot a_5) \end{aligned} \tag{3.1}$$

gdzie a_i jest ilością elementów dla jednego i-tego wymiaru. Przeliczenie indeksów dla kolejnych wymiarów wyraża się wzorami przedstawionymi poniżej, gdzie d jest macierzą

$$\begin{aligned} &i \Rightarrow I = i, \\ &(i,j) \Rightarrow I = i + j \cdot d_1 \\ &(i,j,k) \Rightarrow I = i + j \cdot d_1 + k \cdot d_2 \\ &(i,j,k,l) \Rightarrow I = i + j \cdot d_1 + k \cdot d_2 + l \cdot d_3 \\ &(i,j,k,l,m) \Rightarrow I = i + j \cdot d_1 + k \cdot d_2 + l \cdot d_3 + m \cdot d_4 \end{aligned}$$

jenowymiarową, $(N-1)$ elementową, której rozmiar wyraża się wzorem:

$$d_f = \prod_{f=1}^{f \leqslant N} a_f$$

Dla przykładu przeliczymy położenie jednego elementu o wartości γ z wielowymiarowej macierzy do jednowymiarowej tablicy. Tak jak było wspomniane wcześniej, macierz α jest 5-cio wymiarowa i zakładamy, że posiada 3 elementy w pierwszym wymiarze, 4 w drugim itd., schematycznie α[3][4][2][8][6]. Reperezentacja jednowymiarowa jest tablicą z 1152 elementami według iloczynu:

$$I_{max} = \prod_{f=1}^{f \leqslant N} a_f$$

Przeliczenie indeksów z macierzy α do β wygląda następująco:

$$\begin{aligned} (2,2,1,2,2) \quad &\Rightarrow \quad I = 2 + 2 \cdot 3 + 1 \cdot 12 + 2 \cdot 24 + 2 \cdot 192 \\ (2,2,1,2,2) \quad &\Rightarrow \quad I = 452 \end{aligned} \tag{3.2}$$

Czyli element macierzy pod indeksami (2,2,1,2,2) w tablicy znajdzie się pod indeksem 452. Aby dokonać konwersji odwrotnej trzeba posłużyć się poniższymi wzorami, gdzie operator % oznacza resztę z dzielenia. Natomiast przy zwykłym dzieleniu, które tutaj występuje, bierzemy tylko wartość całkowitą i w ten sposób otrzymujemy szukane indeksy tablicy.

$m = I/192$	2
$l = (I\%192)/24$	2
$k = ((I\%192)\%24)/12$	1
$j = (((I\%192)\%24)\%12)/3$	2
$i = (((I\%192)\%24)\%12)\%3$	2

3.3.4. Tworzenie neutrin na podstawie histogramu 5-cio wymiarowego

W poprzednich podrozdziałach pokazaliśmy, jak wygenerować histogram i jak operować na n-wymiarowej macierzy. Zostało również wytłumaczone jak generować kolejne dane na podstawie histogramu. W tym podrozdziale omówiony zostanie sposób w jaki zostały użyte wspomniane etapy do tworzenia kolejnych neutrin. Poniższy wycinek kodu przedstawia polimorficzną metodę *shoot()* która generuje cząstki.

Najistotniejsze trzy miejsca to: po pierwsze, losowanie liczby która mieści się pomiędzy zerem a sumą wszystkich wag (słupków) histogramu.

```
int x = frandom() * _arr[_count - 1];
```

Następnie za pomocą bisekcji odnajdujemy indeks elementu w pięciowymiarowej macierzy. Ostanit krok to wyliczenie wartość energii, dwóch wymiarów położenia oraz trzech kierunku pędu. Poniżej kod obliczający położenie na OX.

```
part.r.x = _stats.GetMin().xnu + (idx[0]+frandom())
                * _inf[0];
```

Na początek pobierana jest minimalna wartość położenia na której określony jest histogram. Zmienna *_inf[0]* mówi jaka jest różnica wartości pomiędzy kolejnymi słupkami, czyli o ile zmienia się położenie w histogramie przechodząc do kolejnych elementów. Ta wartość jest mnożona przez indeks przedziału plus pewne odchylenie. Kod znajduje się w plikach *beamHist.cc* i *narray.h* lub jego fragmenty w dodatku D.3.

3.3.5. Optymalizacja czasu wykonania dla wyszukiwania w histogramie

Kilka słów o pracy nad przyśpieszeniem wyszukiwania elementu w histogramie. Odnalezienie go oznacza posiadanie jego indeksów w macierzy co dalej prowadzi do określania jego parametrów.

Omawiana wtyczka potrafi zapisywać i odczytywać histogram z pliku tekstowego. W celu zapoznania się z kodem, który implementuje wczytywnanie i zapisywanie histogramu odsyłam do nagłówka narray.h. Ponieważ oglądanie danych z histogramu zawartych w plikach jest stosunkowo łatwe, można zauważyć, że posiada on ogromną ilość nadmiarowych danych. Mówiąc dokładniej histogram składa się głównie z samych zer. Ponieważ bisekcja odpowiedzialna za wyszukiwanie elementu jest newralgicznym miejscem ze względu na czas wykonywania, zostało wprowadzone usprawnienie polegające na usunięciu zer. Taki krok wymagał przedefiniowania indeksowania elementów

w tablicy. Elementem jest NArrayItem który posiada indeks *idx* z oryginalnego histogramu sprzed etapu usunięcia zer, i *value* odpowiadającą wartości spod oryginalnego indeksu.

```
template< typename T >  struct NArrayItem
{
  int idx;
  T value;
  // ..
};
```

Wprowadzenie tej zmiany powoduje oszczędność pamięci w trakcie wykonywania symulacji. Niestety w momencie tworzenia histogramu powstaje macierz rzadka i nie została przeanalizowana kwestia bezpośredniego wyliczania macierzy zredukowanej, aktualna wersja NuWro dokonuje konwersji po wygenerowaniu histogramu z dużą ilością zer.

Zaletą konwersji jest wielokrotne zmniejszenie ilości kroków w trakcie wyszukiwania elementu. Przyśpieszenie jest widoczne pomimo użycia wydajnego algorytmu bisekcji. Porównanie ilości elementów znajduje się w dodatkach, w rozdziale B.2.

3.4. Próby odnalezienia symetrii w wiązce

Pierwotne podejście do rozwiązania problemu bazowało na przejściu z narzuconych w dokumentacji T2K układów kartezjańskich, do układów sferycznych. Zostały podjęte próby zaimplementowania algorytmów, które miały tworzyć pięciowymiarowy histogram dla trzech składowych pędu i dwóch położenia.

Ideą która przyświecała tym staraniom była próba zmniejszenia ilości wymiarów histogramu, co miało wynikać z domniemanej symetrii w układzie (wiązce). Gdyby okazało się prawdą, że lecące neutrina tworzą wiązkę o kształcie stożka byłoby możliwe dobranie układu współrzędnych tak aby jego początek pokrywał się z wierzchołkiem stożka, co w rezultacie znacząco uprościłoby model. Jedynym utrudnieniem, wynikającym z przesunięcia detektora w bok od osi wiązki (off axis), miało być nałożenie warunków brzegowych na strumień. Natomiast silną stroną rozwiązania byłaby możliwość opisania wiązki przez dwie zmienne niezależne energię i kąt.

Późniejsze prace dowiodły błędności tych założeń. Obliczenia prowadzone przez zaimplementowany program transformujący dwustronnie układy Kokyo-Zahyo i Neut stosowane w T2K (zob. załącznik C w połączeniu z analizą korelacji między energią i kierunkiem pędu pokazały, że są to zmienne niezależne. Dodatkowo nawet gdyby wiązka była cylindrycznie symetryczna, to trudno byłoby wypełnić histogram mając do dyspozycji dane tylko tych neutrin, które przechodzą przez detektor położony poza osią wiązki.

Podsumowując, przejście do układu sferycznego wprowadza niepotrzebne obliczenia i nie daje możliwoście opisu układu w mniej niż pięciu wymiarach. Brak symetrii w układzie jest prawdopodobnie związany z metodą tworzenia neutrin, które mogą powstawać na całej długości komory rozpadu a dodatkowo statystyczny punkt rozpadu jest związany z energią. To rozwiązanie nie będzie zatem dokładniej opisywane w tej

Gałąź	Opis
Nfd	Identyfikator detektora. Jest ich kilka ponieważ w całym detektorze można wydzielić kilka elementów i przez to różne ich konfiguracje mogą być określane jako inne urządzenie. Przykładowo może być to cały magnet ND280, jego sąsiad INGRID lub Tracker czyli część wewnętrzna ND280
Xfd, Yfd, Zfd	Środek detektora w globalnym układzie współrzędnych. Innymi słowy przesunięcie detektora względem układu symulatora Geany
Hfd, Vfd	Wysokość i szerokość detektora

Tabela 3.2: Opis parametrów magnetu na root'owej gałęzi h3000.

pracy.

3.5. Przejścia pomiędzy układami współrzędnych w T2K

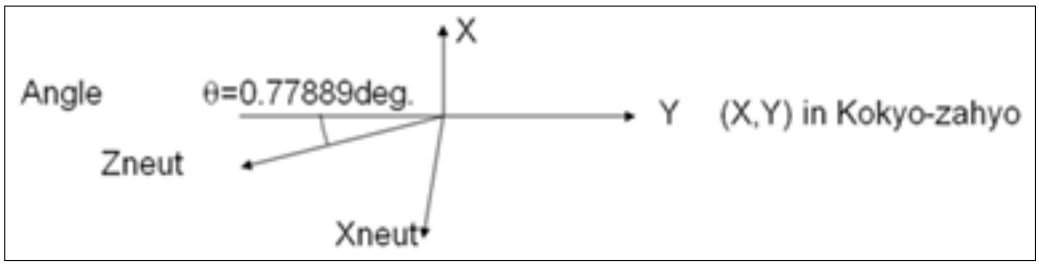
Opisane dwie metody generowania neutrin nie wyczerpują wszystkich możliwości podejścia do tego tematu. Następnym etapem może być ulepszenie użycia histogramu. Powinien on zostać heurystycznie dopasowany do danych, które opisuje. Aby tego dokonać, należy zbadać przestrzenny rozkład cząstek: gdzie są generowane, jakim torem lecą i gdzie są przechwytywane. Poczynione zostału już pewne badania, poniżej opisane są trzy interesujące nas układy współrzędnych:

1. Układ współrzędnych Kokyo-Zahyo który ma swój środek na wylocie akceleratora. Na rysunku 2.2 to miejsce jest oznaczone strzałką i podpisane „target".

2. Układ współrzędnych Global. W jego środku umieszczony jest wirtualny generator neutrin Neut, czyli symulator który nie jest urządzeniem lecz programem służącym do tworzenia hipotetycznej wiązki neutrin.

3. Trzeci układ umiejscowiony jest w środku magnetu ND280.

Przejście pomiędzy układem globalnym a magnesem jest proste ponieważ sprowadza się ono jedynie do translacji. Odległość pomiędzy nimi jest dobrana tak, aby cały magnet był objęty wiązką neutrin. To że urządzenie wycina przestrzeń w wiązce można sprawdzić w następujący sposób: gdy weźmiemy dane z plików root'a i na ich podstawie wykreślimy położenia neutrin, otrzymamy prostokąt, czyli kształt obiektu. W pliku zawarte są neutrina przechodzące przez magnes. Położenia i inne parametry określające neutrina dostępne są na gałęzi root h3002, natomiast h3000 zawiera rozmiary detektora i przesunięcie jego układu odniesienia względem globalnego. Gałąź h3000 jest opisana w tabeli 3.2, tutaj zostały wymienione tylko najważniejsze elementy. Jeśli chcemy od-



Rysunek 3.6: Skręcenie układów Kokyo-Zahyo i Neut (globalny).

czytać położenia neutrin w układzie środka magnetu należy przesunąć punkty o wektor Global → ND280 w metrach.

$$\vec{R}_{G\mapsto ND} = [-3.22, 8.15, -280.0][m]$$

Przejście do bazowego układu w T2k zwanego Kokyo-Zahyo jest bardziej skomplikowane ponieważ jest on skręcony względem dwóch pozostałych. W przedstawionych przejściach używany jest kąt θ przedstawiony na rys. 3.6.

$$Z_{Neut} = -(x - 49754.892)\sin\theta - (y - 69413.452)\cos\theta - 8.929$$

$$X_{Neut} = -(x - 49754.892)\cos\theta + (y - 69413.452)\sin\theta$$

$$Y_{Neut} = -12.14843$$

W dodatkach można znaleźć kod źródłowy krótkiego programu, który dokonuje tych przeliczeń (zob. rozdział C).

Rozdział 4

Wyniki po integracji z NuWro

Równolegle do implementacji pluginu, będącej celem tej pracy, we wrocławskim IFT przebudowywano symulator, aby rozszerzyć jego możliwości. Pełne wykorzystanie możliwości wtyczki umożliwiły dwie istotne zmiany. Pierwsza to wprowadzenie przestrzenności w silniku symulatora - wcześniejsza wersja nie wspierała trójwymiarowego rozkładu zdarzeń, tym samym pozycja neutrina nie była istotna - aktualnie zlokalizowanie obiektu jest możliwe. To z kolei prowadzi do drugiego ważnego ulepszenia: wczytywania schematu budowy detektora i przeprowadzania kolizji w jego objętości. Pozwala to pełniej wykorzystać możliwości wtyczki generującej neutrino również w tej samej objętości.

Natomiast zmiany dokonane w kodzie symulatora podczas tej prac dotyczą rozbudowanego sposobu definiowania źródła nowych cząstek. Poprzednia wersja NuWro pobierała charakterystykę wiązki w postaci jednowymiarowego histogramu energii. Histogram wraz z ogólną konfiguracją był dostępny w pliku *params.txt*. Pierwotnie zakładano możliwość swobodnego definiowania rozkładu energii, co przestało być wystarczające w momencie posiadania innych źródeł definicji wiązki. Obecnie, użytkownik symulatora w pliku *params* określa metodę generowania neutrin, którą symulator powinien użyć. W przypadku naszej wtyczki parametr *beam_type* może przyjmować wartości 2, 3 lub 4 - dokładny opis znajduje w tabeli 4.

W kodzie został użyty polimorfizm, co pozwoliło na uplastycznienie tworzenia i integrowania nowych metod podowania neutrin. Klasa beam wymusza implementację metody *shoot()* dostarczającej cząstkę.

```
class beam
{
public:
        virtual particle shoot(bool dis=0) = 0;
};
```

Pomiar szybkości działania NuWro został dokonany przy działającej wtyczce będącej podajnikiem neutrin. Wyniki testu przeprowadzonego 7 stycznia 2011 na aktualnej wersji przedstawia tabela 4. Jak widać metoda generowania nutrin na podstawie histogramu jest dużo szybsza mimo konieczności przeszukiwania elementów w dużej tablicy danych. Przyczyną wolniejszego działania pierwszej wersji jest długi czas dostępu do

beam_type	Opis
2	Źródłem nowych neutrin są kolejno pobierane elementy z plików root.
3	Źródłem neutrin jest histogram pobierany z pliku *histout.txt*.
4	Zamiast przeprowadzania symulacji symulator przechodzi w tryb generowania histogramu na podstawie plików root.

Tabela 4.1: Nowe dostępne tryby pracy symulatora związane z wtyczką.

Sposób generowania	Powt.	Czas
wczytywanie z plików	40 000	27 min 4 sek
wczytywanie z plików	10 000	4 min 51sek
wczytywanie z plików	2 500	43 sek
histogramowa	100 000	4 min 48 sek
z plików (modyf)	100 000	6 min 5 sek

Tabela 4.2: Porównanie szybkości działania NuWro w trzech trybach wtyczki. Dane pobierane bezpośrednio z plików root, generowane na podstawie histogramu lub implementacja dr C. Juszczaka.

pliku z danymi zapisanego na dysku komputera. W metodzie histogramowej nie był brany pod uwagę czas tworzenia histogramu i jego wczytywania z pliku. Natomiast czas ostatniego rozwiązania nie zawiera wczytywania plików roota do pamięci.

Tabela nie zawiera wyników pomiaru otrzymanych na podstawie pierwotej metody generowania neutrin, opartej na jednowymiarowym histogramie energii zapisanym w pliku *params.txt*. Pomiar ten nie byłby w tym zestawieniu miarodajny, gdyż w tle testu dochodzi do zmiany warunków: stare podejście nie używa symulatora w trybie przestrzennym, brak jest tam obliczeń związanych z geometrią. Zmierzony czas jest siedmiokrotnie krótszy w porównaniu z czasem przy metodzie histogramowej (przy najdłuższym wariancie 40tys powtórzeń).

Należy zwrócić uwagę, że uwzględniony został istotny czynnik charakteryzujący wiązkę. Wejściowe pliki root posidają dla każdego neutrina przypisaną normę (wagę) - wartość opisującą, jak często dana cząstka może się pojawiać. Neutrina o mniejszych energiach powstają z większym prawdopodobieństwem niż te wysokoenergetyczne. Ma to krytyczne znaczenie dla prędkości przebiegu symulacji w momencie gdy warunkiem jej zajścia jest dostarczanie cząstek o dużych energiach. Przykładem takiej symulacji mogą być wszystkie rozpraszania głęboko nieelastyczne.

W trakcie powstawania tej pracy dr C. Juszczak implementował rozszerzenie pierwszej metody tworzenia neutrin. Po pierwsze wczytywane są neutrina z plików root do pamięci komputera. Po drugie, idąc od początku tak powstałej tablicy, sumowane są kolejne wartości wag lub iloczynów energii i wagi (dodatkowe przeważanie profilu energetycznego wiązki energią neutrina stosowane jest w zdarzeniach głęboko nieelastycznych by efektywniej uwzględnić rosnący z energią przekrój czynny). Sumowany jest

aktualny element z jego poprzednikiem. Ostatecznie, losowa wartość wagi odnajdywana jest za pomocą bisekcji w celu wygenerowania pozostałych parametrów neutrina. Takie podejście jest hybrydą wszystkich opisanych w tej pracy algorytmów. Cechuje ją doskonała statystyka parametrów neutrin, zgodna z oryginalnymi cząstkami z T2K.

Wyniki symulacji NuWro przy użyciu zaimplementowanej wtyczki są przedstawiane na poniższych wykresach. Wszytkie przedstawiane dalej wyniki zostały wygenerowane na potrzeby tej pracy przy pomocy NuWro, z użyciem metody histogramowej oraz pierwszej metody generowania neutrin poprawionej przez dr C. Juszczaka. Wyjątkiem są A.4, A.6, A.8, których dolne, referencyjne wykresy pochodzą z Neuta.[1]

Pierwszy, dwuwymiarowy wykres A.3 pokazuje położenia neutrin w momencie rozpraszania. Widoczna jest płaszczyzna $X \times Z$, gdzie oś OZ jest równoległa do wiązki, OX prostopadła pionowa a OY prostopadła pozioma ze zwrotem od środka wiązki do zewnątrz. Ilość detali rozpoznawalna na rysunku związana jest z dużą szczegółowością schematu detektora. Jest to istotne przy interpretacji kolejnych wykresów.

Kolejne dwa wykresy to porównanie ilości neutrin wzdłuż osi OZ dla NuWro i Neuta. Wyniki są identyczne - należy tu podkreślić, że pozorne różnice między nimi to efekt uszczegółowienia schematu detektora. Po pierwsze, w NuWro objętość urządzenia jest mniej więcej w przedziale $\langle -3900, 3900 \rangle$ natomiast Neut zakłada, że detektor jest mniejszy i jego przedział na osi OZ jest mniejszczy $z \in \langle -3300, 2750 \rangle$. Dlatego widzimy dwa dodatkowe piki na brzegach wykresu NuWro. Po drugie, wartości wykresu dla Neut są mniejsze co ponownie wynika z różnic wielkości detektora w NuWro. Ta sytuacja związana jest z pozostałymi dwoma wymiarami x i y. Obraz powstały po wykreśleniu ilości neutrin wzdłuż detektora doskonale pokrywa się ze schematem widocznym w początkowej części pracy w rozdziale 2.3.

Kolejne bliźniacze wykresy (patrz. A.6) przedstawiają ilość neutrin wzdłuż OY. Asymetria wynika z rozrzedzającej się wiązki neutrin w miarę oddalania się od środka wiązki. Detektor ND280 jest off-axis co oznacza, że środek wiązki nie przechodzi przez jego środek, lecz jest umieszczony na jej brzegu.

Wykresy A.8 przedstawiają porównanie ilości neutrin wzdłuż OX. Warto tutaj zatrzymać się przy dwóch istotnych kwestiach. Pierwsza to różnice w środkach wykresów. W przypadku Neuta widoczne jest lokalne maksimum natomiast w NuWro ten pik jest znacznie obniżony. Sąsiedztwo tego elementu jest propocjonalnie wyższe w NuWro. Rozwiązanie wspomnianego problemu ponownie sprowadza się do różnic w szczegółowości wymiarów detektora. Na rys. A.11 widoczne jest pionowe zagęszczenie neutrin z którego wynika podwyższony słupek na wykresie. Natomiast ciemne poziome paski na górze i na dole rysunku powodują podniesienie wykresów wokół tego lokalnego maksimum. Można bezpiecznie założyć, że schemat detektora używany w Neut nie posiada tych elementów. Drugą kwestią godną uwagi jest brak gradientu jaki mieliśmy w przypadku OY. Wydawało się, że przesunięcie detektota względem środka osi w dwóch wymiarach, a nie tylko na osi Y pozwala zakładać, że zmiana gęstości cząstek powinna być symetryczna w obu przypadkach. Tak jednak nie jest. Nie potrafię wyjaśnić przyczyny, co więcej Neut i NuWro mają zgodne wyniki.

W przypadku energii neutrin przedstawionej na rysunkach A.14, A.15 porównują-

[1] Trzy wykresy zrbione zostały przy pomocy Neuta przez Pawła Przewłockiego

cymi wyniki dla metody histogramowej przy rosnącej rozdzielczości z wykresem dla metody dr C. Juszczaka widać, że metoda histogramowa zbiega do tej drugiej. Widać, że zwiększanie rozdzielczości w przypadku energii neutrina przybliża rozwiązanie beamHist do beamRF, czyli poprawia jakość. Na rysunku A.16 przedstawiony jest wykres wrozcowy na podstawie plików T2K.

Rozdział 5

Podsumowanie

Cel tej pracy, jakim było stworzenie nowej metody generowania neutrin jako rozszerzenie symulatora NuWro, udało się zrealizować. Zostały stworzone dwie metody generowania neutrin. Pierwsza polegająca na wczytywaniu neutrin z plików root i bezpośrednim wprowadzaniu ich do symulacji. Na jej podstawie powstała implementacja dr C. Juszczaka, która wczytywane neutrina akumuluje w pamięci jako jednowymiarowy, długi histogram norm.

Druga metoda jest dwustopniowa ponieważ przed rozpoczęciem symulacji, NuWro generuje histogram na podstawie neutrin z eksperymentu T2K. Tak powstała pięciowymiarowa macierz może być używana wielokrotnie i przekazywana w postaci pliku tekstowego. Prócz macierzy zapisane tam są informacje o ekstremach oraz o gęstości podziału każdego z wymiarów. Należy zwrócić uwagę, że użytkownik może zdefiniować różną grubość przedziałów histogramu dla każdego z parametrów, pozwala to na zwiększenie dokładności w miejscach newralgicznych, przykładowo dla energii neutrina. Zaletą metody są wystarczająca dokładność odwzorowania oryginalnych danych, szybkość i mała ilość danych potrzebna do rozpoczęcia symulacji. Wielkość histogramu, o dobrej charakterystyce generowanych neutrin, oscyluje pomiędzy 1 a 10mb, dodatkowo po spakowaniu wielkość spada do kilkuset kilo co pozwala załączyć go na stałe do projektu NuWro.

W ramach pracy nie udało się doprowadzić do optymalizacji ilości wymiarów histogramu. Było to spowodowane specyfiką powstawania neutrin: neutrina powstające podczas rozpadu nie posiadają wspólnego punktu, miejsce powstania jest skorelowane z energią neutrina. To doprowadziło do konieczności używania wielowymiarowego histogramu a nie, jak zakładano pierwotnie, dwuwymiarowego składającego się z kąta kierunku pędu i energii neutrina. Próby optymalizacji wymagają osobnego zajęcia się tematem.

Uzupełnieniem dla beamHist w przyszłości będzie dodanie przeważania energią dla DIS i RES tak jak zostało to zrobione dla beamRF przez dr C. Juszczaka. Obecnie korzystanie z wiązki histogramowej w kanałach DIS i RES wymaga wyłączenia w nuwro opcji przeważania wiązki energią.

Tworzenie precyzyjnego histogramu tak aby dawał wartościowe wyniki to proces czasochłonny i wieloetapowy. Przykładowy kierunek rozwoju dla metody histogramowej można iść przez poprawianie rozmycia dla każdego przedziału z liniowego do gaus-

sowskiego.

Obiecujące są wyniki otrzymywane ze skompresowanych histogramów o lepszej rozdzielczości. W chwili obecnej jednak generowanie histogramu jest możliwe tylko w postaci nieskompresowanej - nie korzysta z faktu, ze jest to macierz rzadka - jest więc ograniczone ilością pamięci operacyjnej dostępnej dla programu. Implementując histogram za pomocą słownika std::map można w przyszłości obejść to ograniczenie.

Bibliografia

[Ju11] Cezary Juszczak, *Running NuWro*, IFT UWr, 29 June 2011, running_nuwro.pdf

[Sob08] Jan T. Sobczyk, *NuWro - Monte Carlo generator of neutrino interactions*, IFT UWr, NuWro NuFact08, 01 July 2008, nuwro_nufact08.pdf

[Ver07] Antonin Vacheret - for the T2K collaboration, *T2K beam line and near detectors*, NuInt07, May 30, June 3 2007 Fermilab, AntoninT2KNuint07s.pdf

[Gif11] Spencer Giffin On behalf of the T2K FGD group UBC, Kyoto University, University of Regina, TRIUMF, University of Victoria, *The Fine Grained Detectors For The T2K Experiment*, CAP Congress, St. John's, June 13-17, 2011, fine_grained_detectors.pdf

[TechRep] *Technical Design Report for T2K-ND280*, ver. 1.0, ND280.org, 18 November 2006, ND280-review-Tech.pdf

[Prz11] Paweł Przewłocki, *NuWro 2011 (2)*, 01 Lipiec 2011, pp-nuwro2011-2-010711.ppt

[Dav08] Gavin Davies, Athanasios Hatzikoutelis, Laura Kormos, *Construction of a Downstream Ecal for the T2K ND280 Detector*, Lancaster University, IoP HEPP Lancaster, 31 March, 2 April 2008, ecal_docs.pdf

[Gol09] Tomasz Golan, *Oddziaływanie stanów końcowych w reakcjach neutrin z jądrami atomowymi*, IFT UWr, 21 Grudzień 2009, golan_nuwro_results.pdf

[Hay09] Yoshinari Hayato, *A Neutrino Interaction Simulation Program Library NEUT*, Kamioka Observatory, ICRR, University of Tokyo, received 02 July 2009, Neut.pdf

[Sze10] Tomasz Szeglowski, *SMRD radial profile official plot canditates*, Institute of Physics, University of Silesia, 13 December 2010, TSzeglowski_RadialProfile_officialPlotCandidates_update2.pdf

[Min10] Akihiro Minamino, *Center positions of Off-diagonal/Proton modules*, Kyoto, INGRID meeting, 09 November 2010, ingrid_101111_minamino.pdf

[Sak11] Ken Sakashita for T2K collaboration, *New results from T2K experiment*, KEK Physics Seminar, 15 June 2011, kek_seminar_20110615.pdf

[Yam05] Yoshikazu Yamada for T2K collaboration and J-PARC Neutrino facility construction group, *The T2K program*, Talk at plenary session of NUFACT05, 21 June 2005, Yamada_Plenary.pdf

[Kis05] Jan Kisiel, *Perspektywy Akcelaratorowej Fizyki Neutrin*, Uniwersytet Śląski, 11 Kwietnia 2005, janek_kisiel_oscylacje.ppt

[Ogi09] Toru Ogiittsu, Yasuhiiro Makiida, Ken-iichii Sasakii, Tattsushii Nakamotto, Hiirokattsu Ohatta, Nobuhiiro Kiimura, Takahiiro Okamura, Akiio Terashiima, Yasuo Ajjiima, Noriio Hiigashii, Takayukii Tomaru, Masahiis IIiida, Keniichii Tanaka, Osamu Araoka, Shojjii Suzukii, Akiira Yamamotto, Takashii Kobayashii, Attsuko IIchiikawa, Takeshii Nakadaiira, Ken Sakashiitta, Takuya Hasegawa, Yoshiiakii Fujjiiii, Hiidekazu Kakuno, *Superconducting Combiined Functiion Magnet System for J-PARC Neutriino Beam Liine*, 17 July 2009, CERN-Saclay-RAL-r.pdf

[Wgn] Warszawska Grupa Neutrinowa, http://neutrino.fuw.edu.pl/

[Fuw] Strona wydziału fizyki UW, http://neutrino.fuw.edu.pl/˜neutrina/taon.html

[Ift] Strona wydziału IFT UWr, NuWro - Wrocław Neutrino Event Generator, http://borg.ift.uni.wroc.pl/nuwro

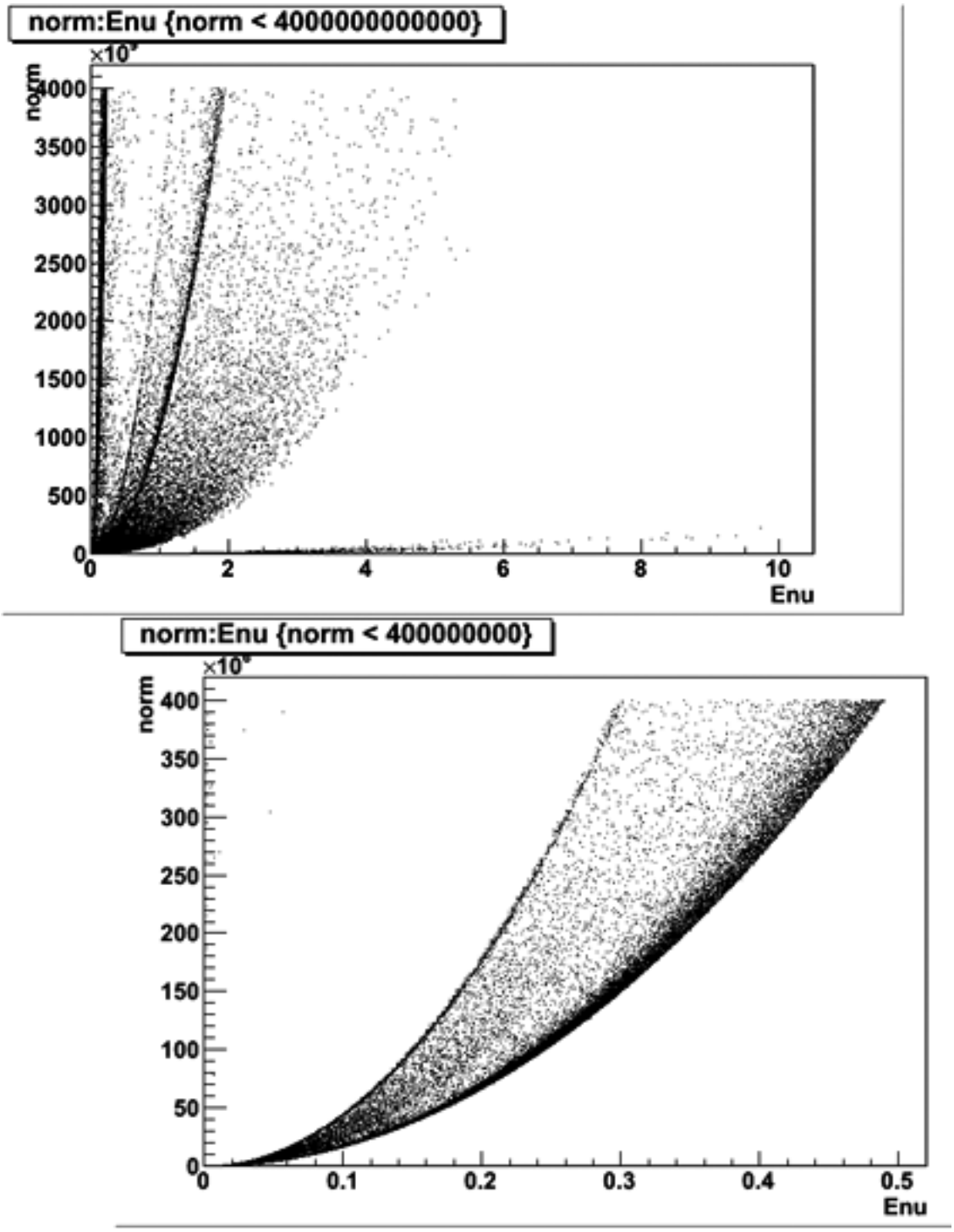
[T2k] Strona eksperymentu T2K, http://www.t2k.org

[Per] Perkins Donald H. *Wstęp do fizyki wysokich energii*, Wydawnictwo Naukowe PWN

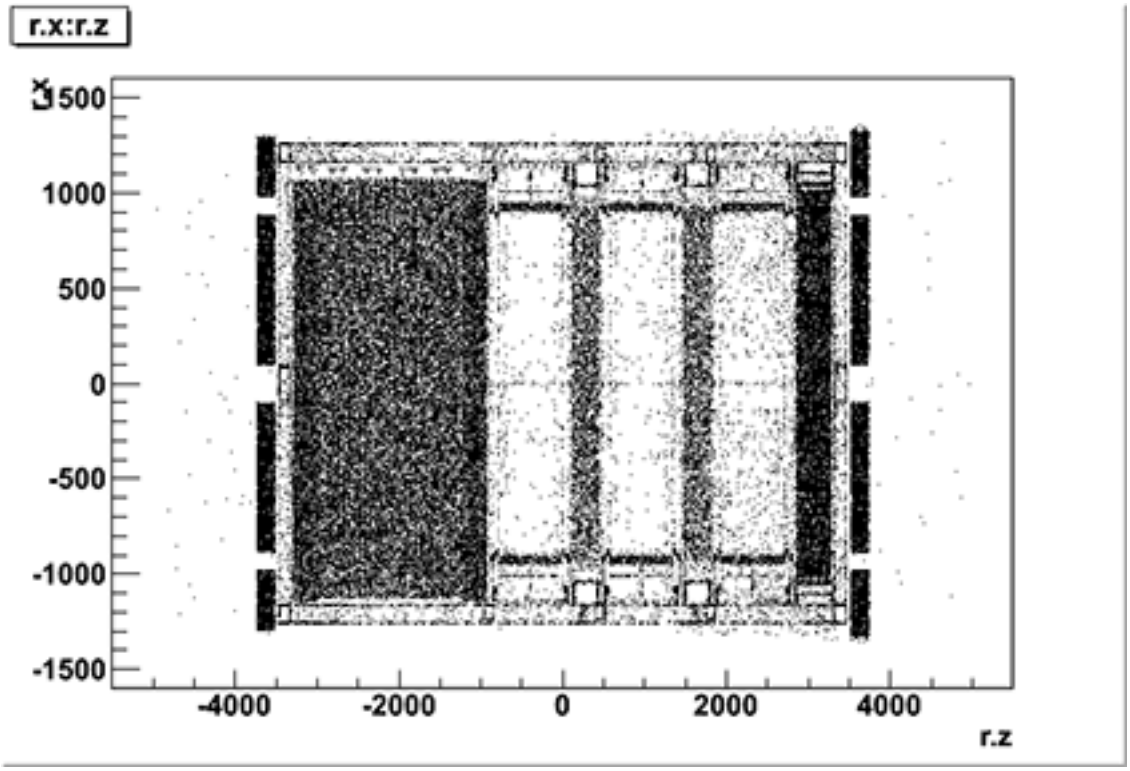
Dodatki

Dodatek A

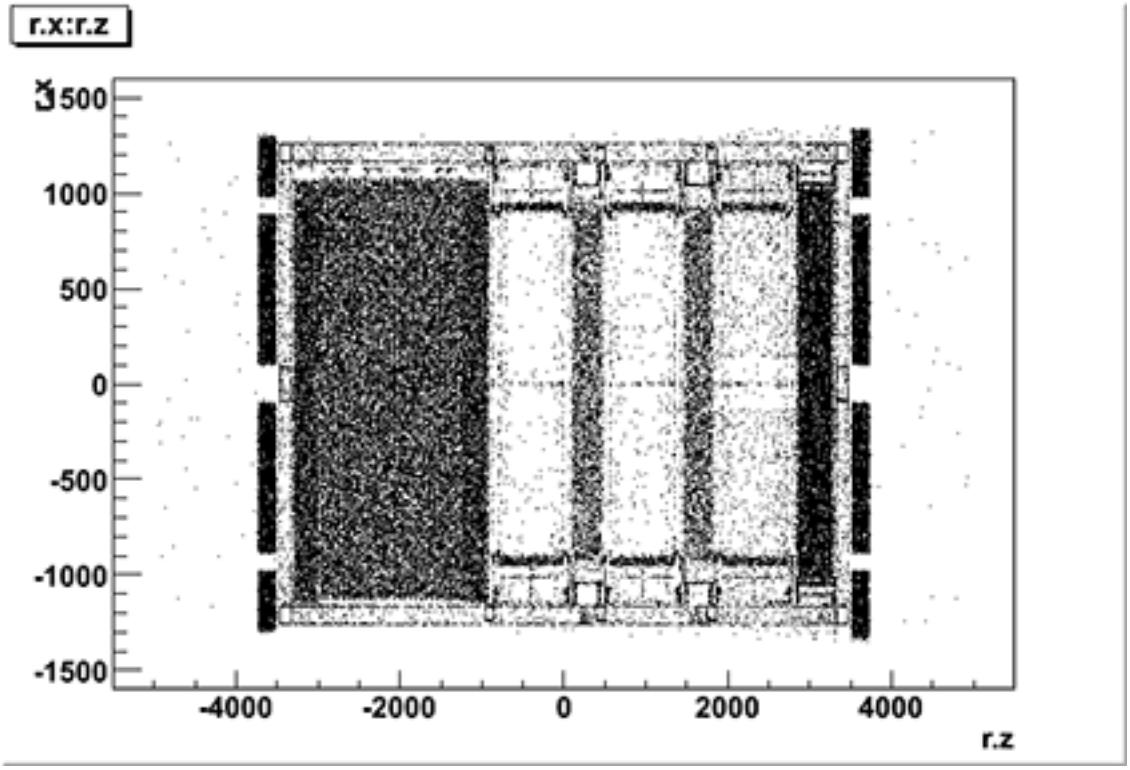
Wykresy



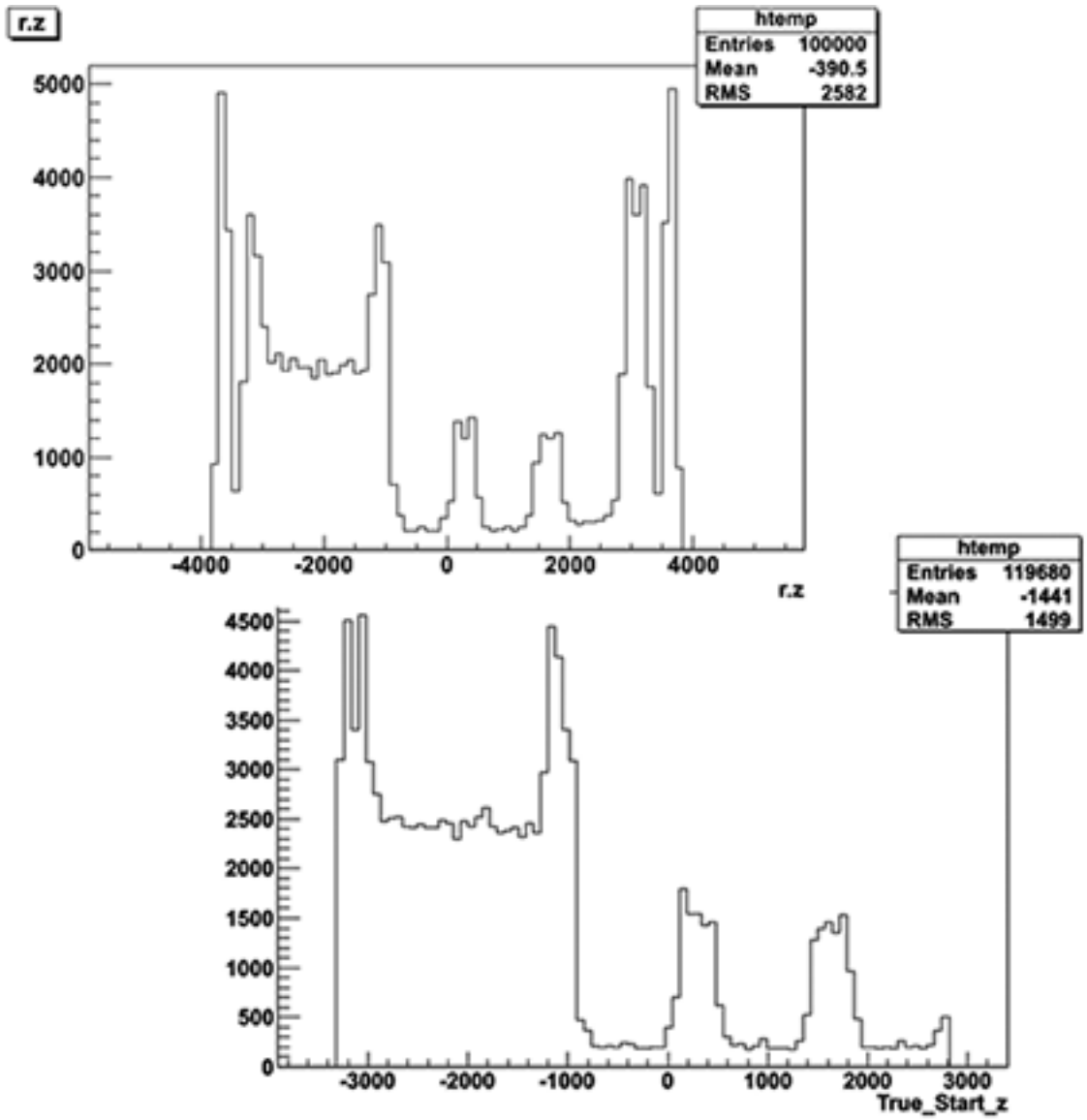
Rysunek A.1: Neutrina w plikach root posiadają swoje normy.



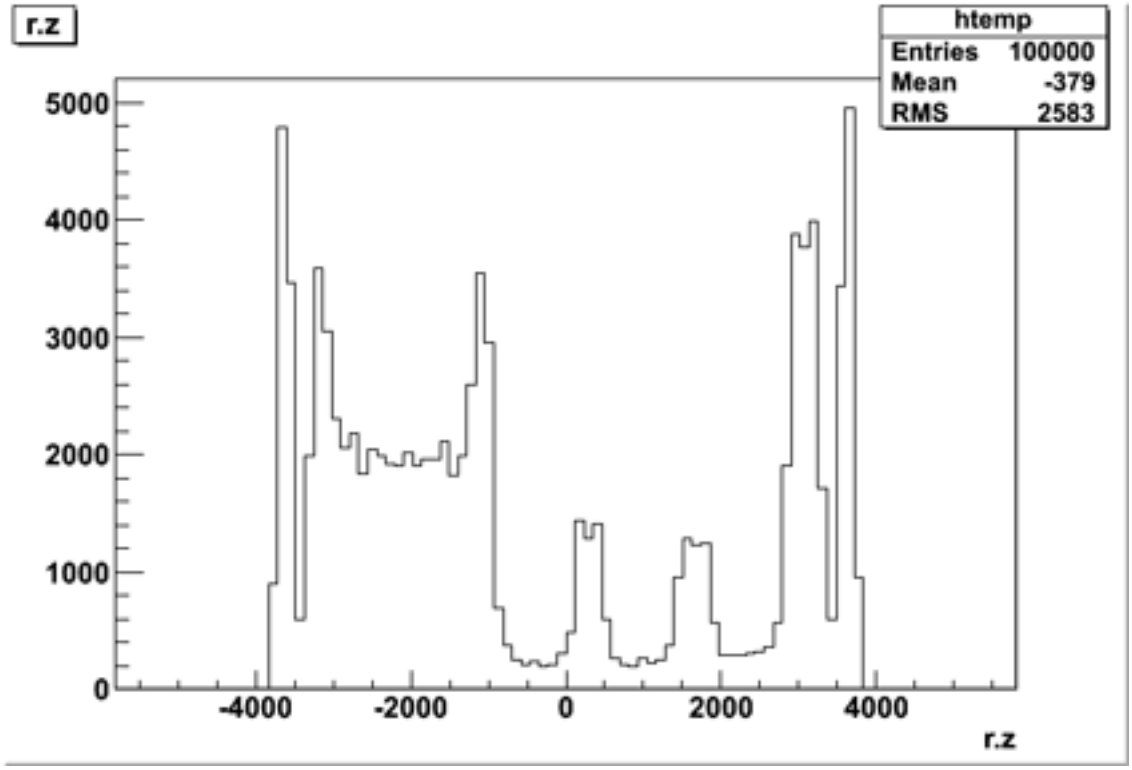
Rysunek A.2: Neutrina wyłapane w przestrzeni detektora. Jest to wykres na podstawie danych zebranych podczas symulacji w NuWro (metoda histogramowa). Wzdłuż OZ leci wiązka - czyli widzimy detektor z boku. Dokładny opis poszczególnych elementów znajduje się w rozdziale 2.3.



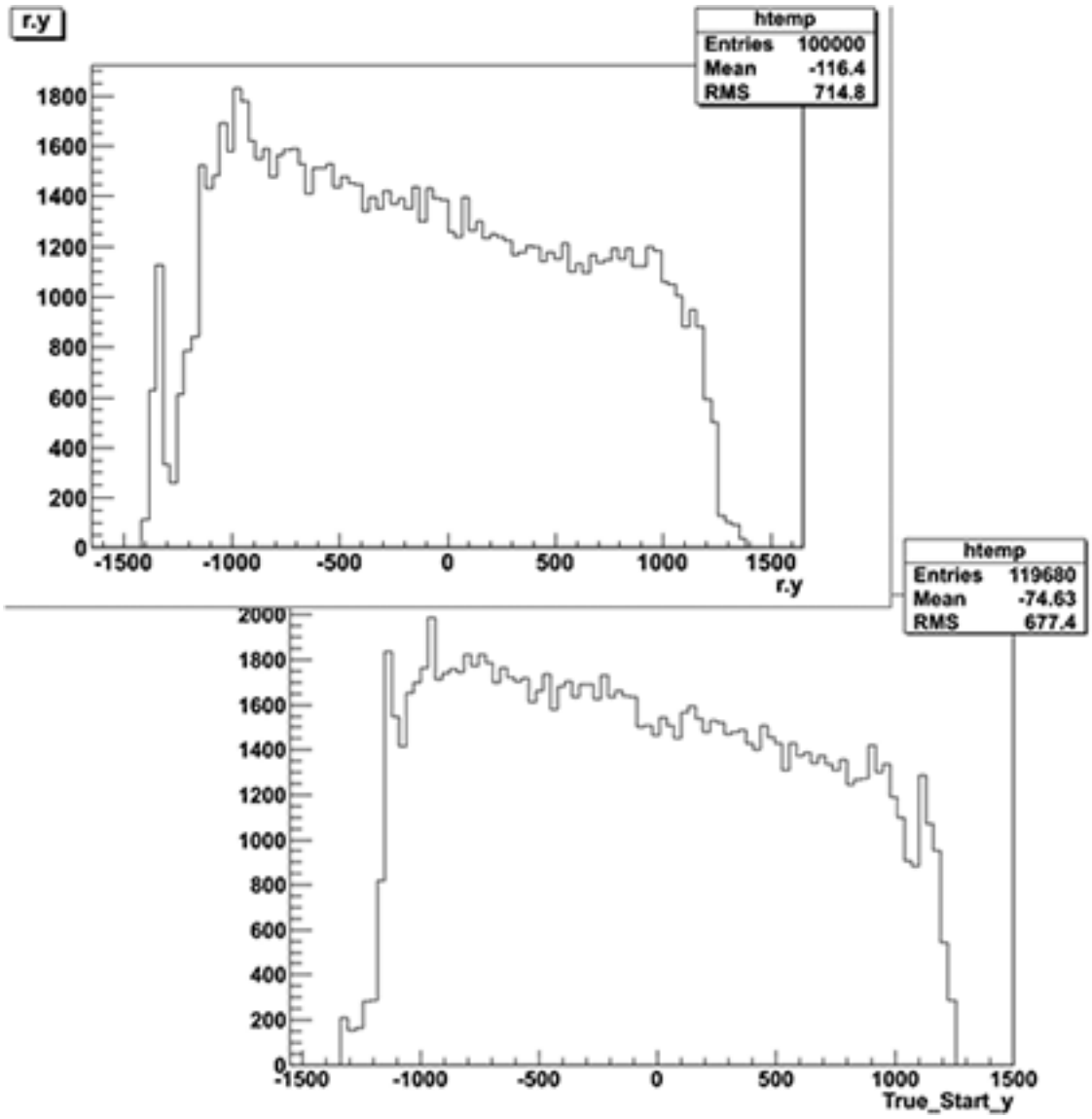
Rysunek A.3: Metoda dr C. Juszczaka



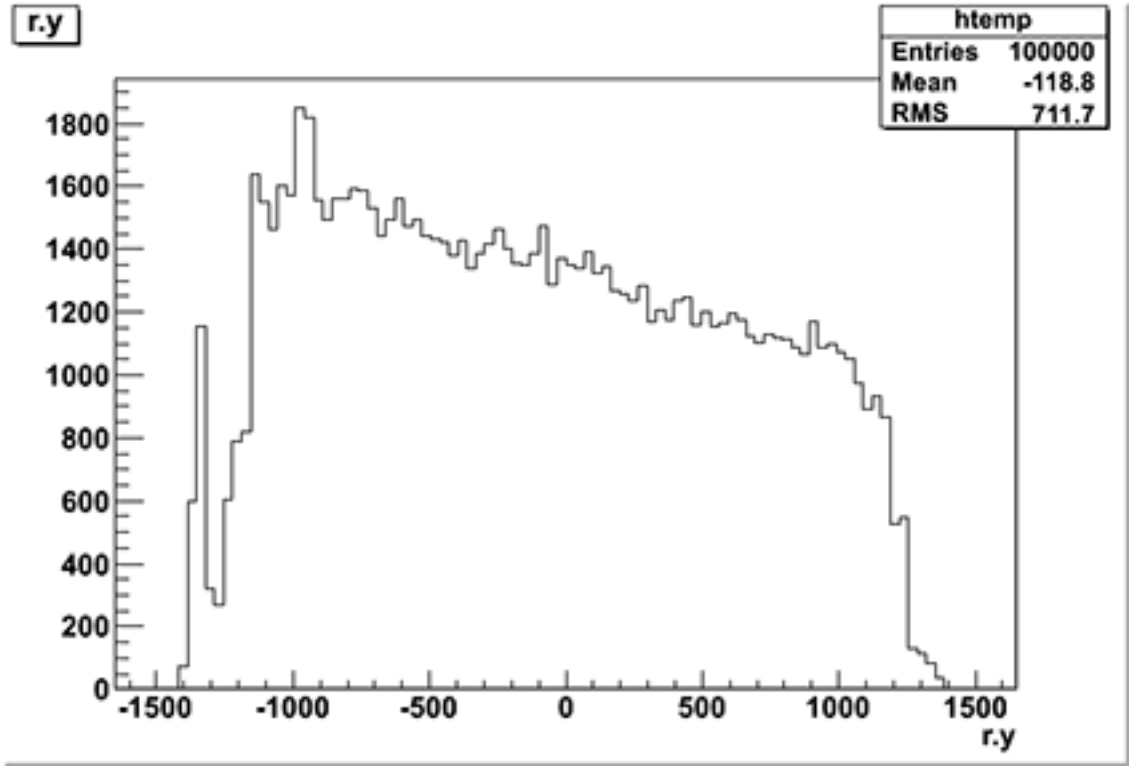
Rysunek A.4: Porównanie ilości neutrin wyłapanych wzdłuż OZ. Górny wykres przedstawia dane z NuWro (metoda dr C. Juszczaka), dolny z Neuta. Przedział w przypadku NuWro jest większy bo dokładniejszy jest schemat detektora.



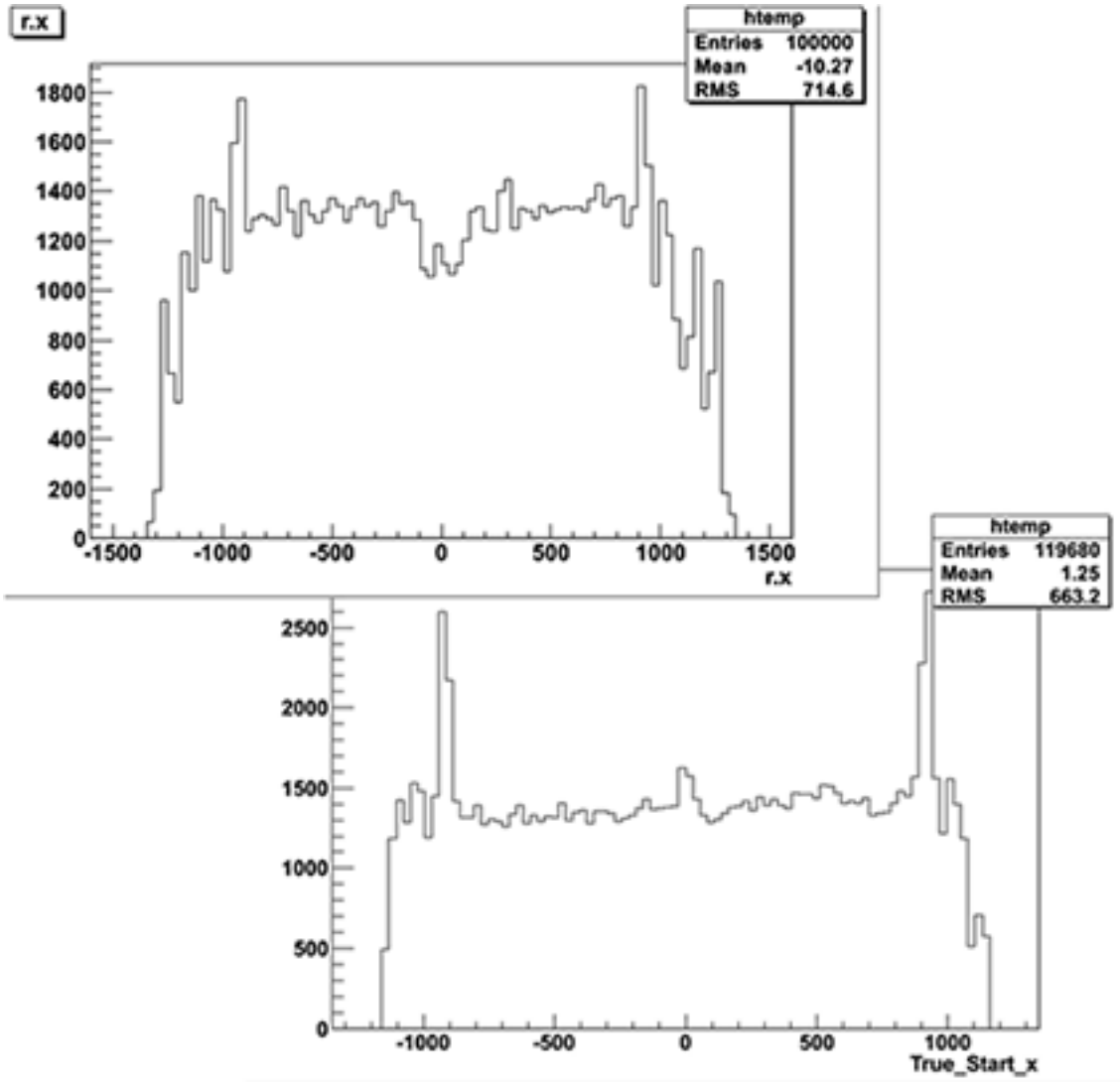
Rysunek A.5: Metoda histogramowa



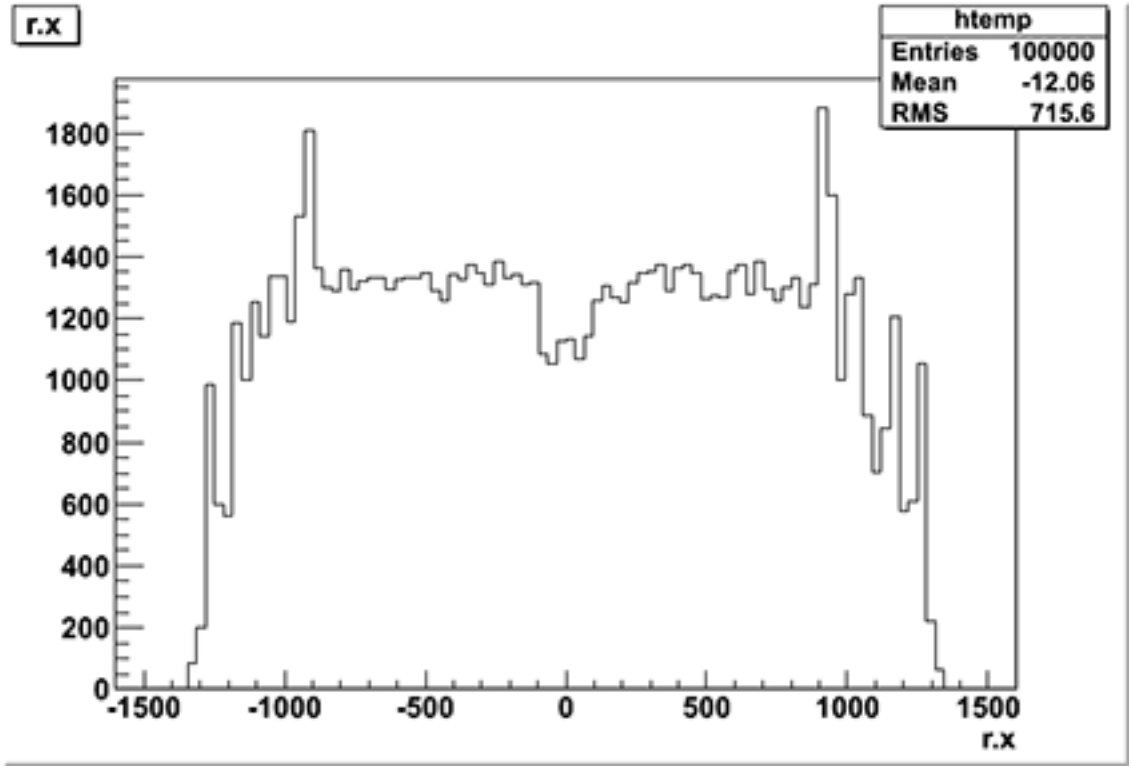
Rysunek A.6: Porównanie ilości neutrin wyłapanych wzdłuż OY. Górny wykres przedstawia dane z NuWro (metoda dr C. Juszczaka), dolny z Neuta.



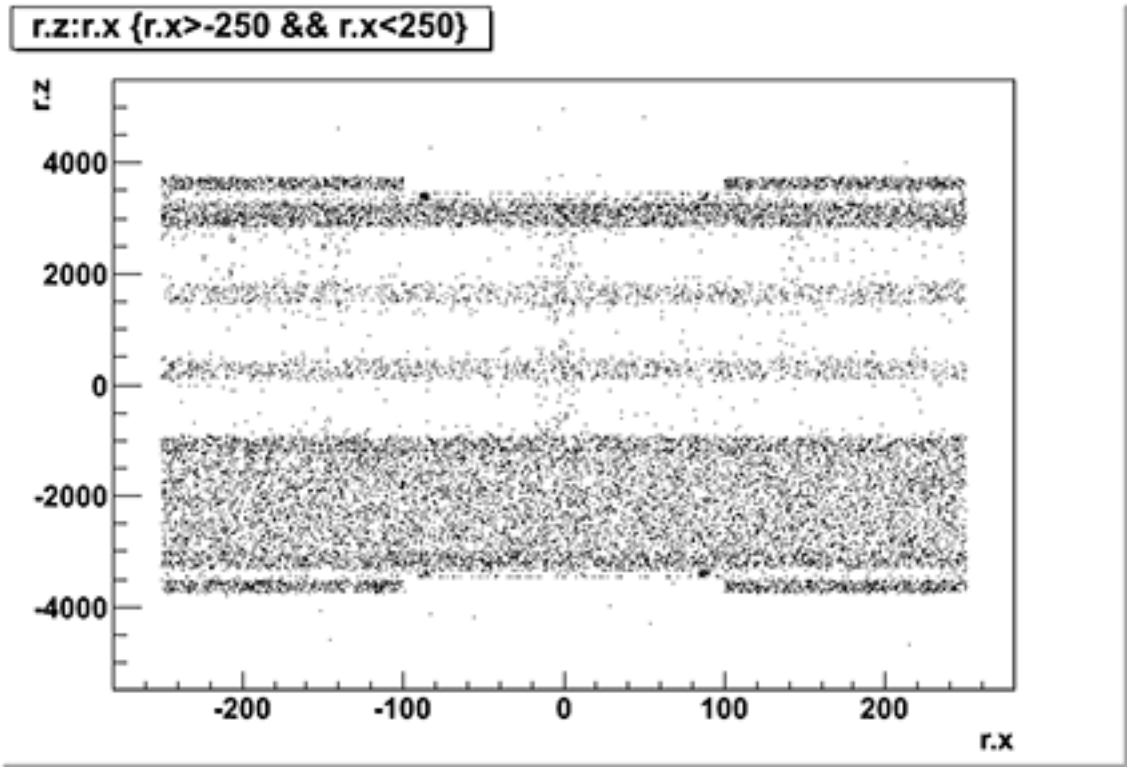
Rysunek A.7: Metoda histogramowa



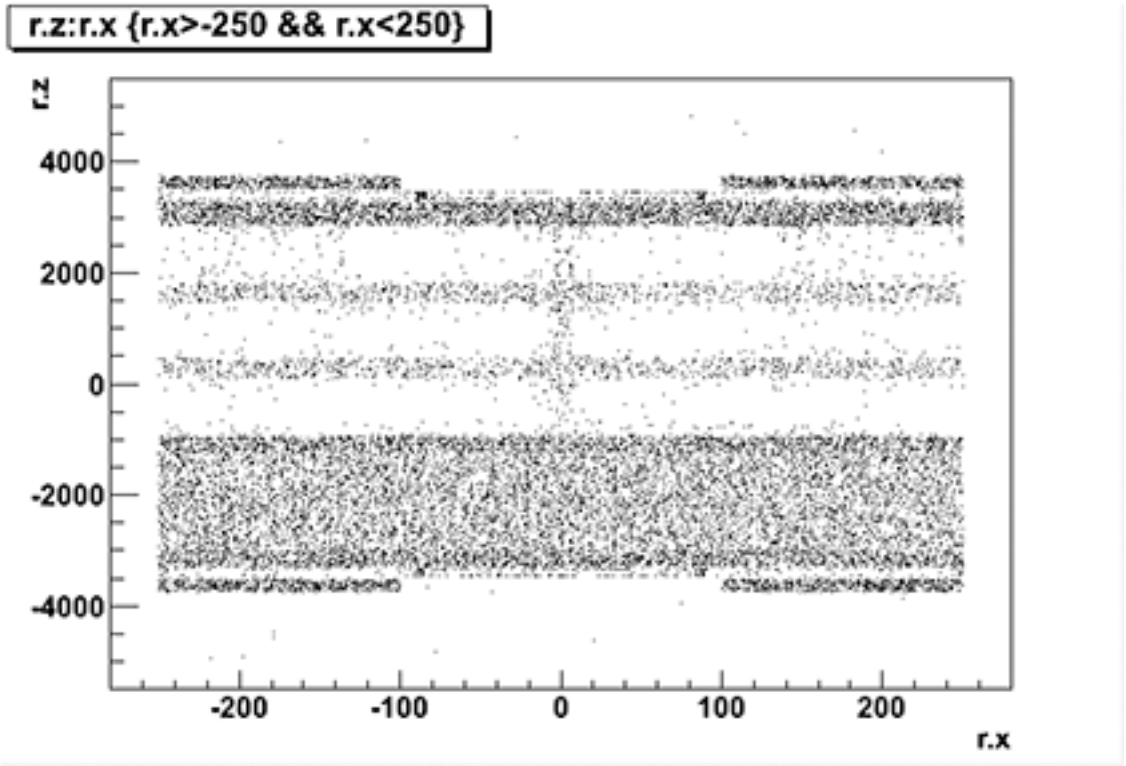
Rysunek A.8: Porównanie ilości neutrin wyłapanych wzdłuż OX. Górny wykres przedstawia dane z NuWro (metoda dr C. Juszczaka), dolny z Neuta.



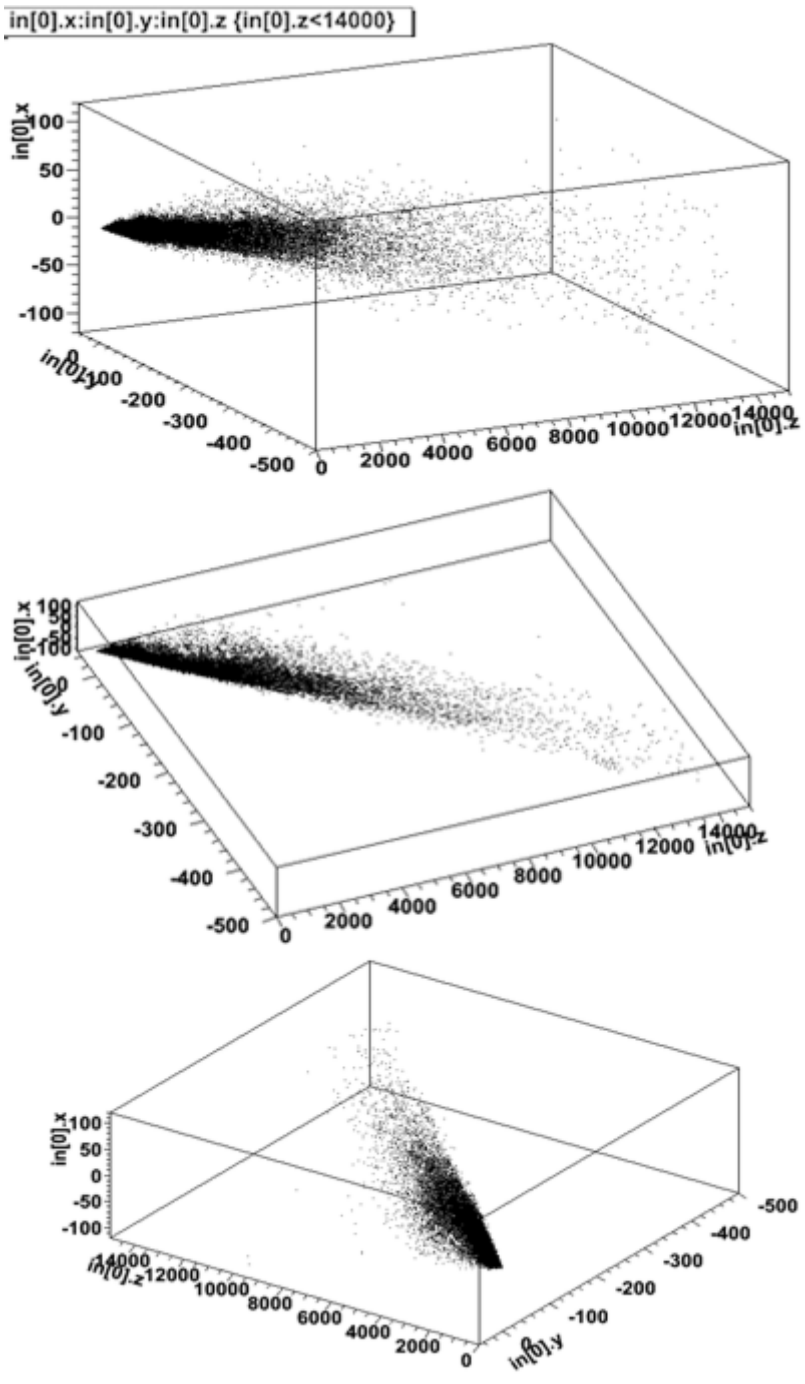
Rysunek A.9: Metoda histogramowa



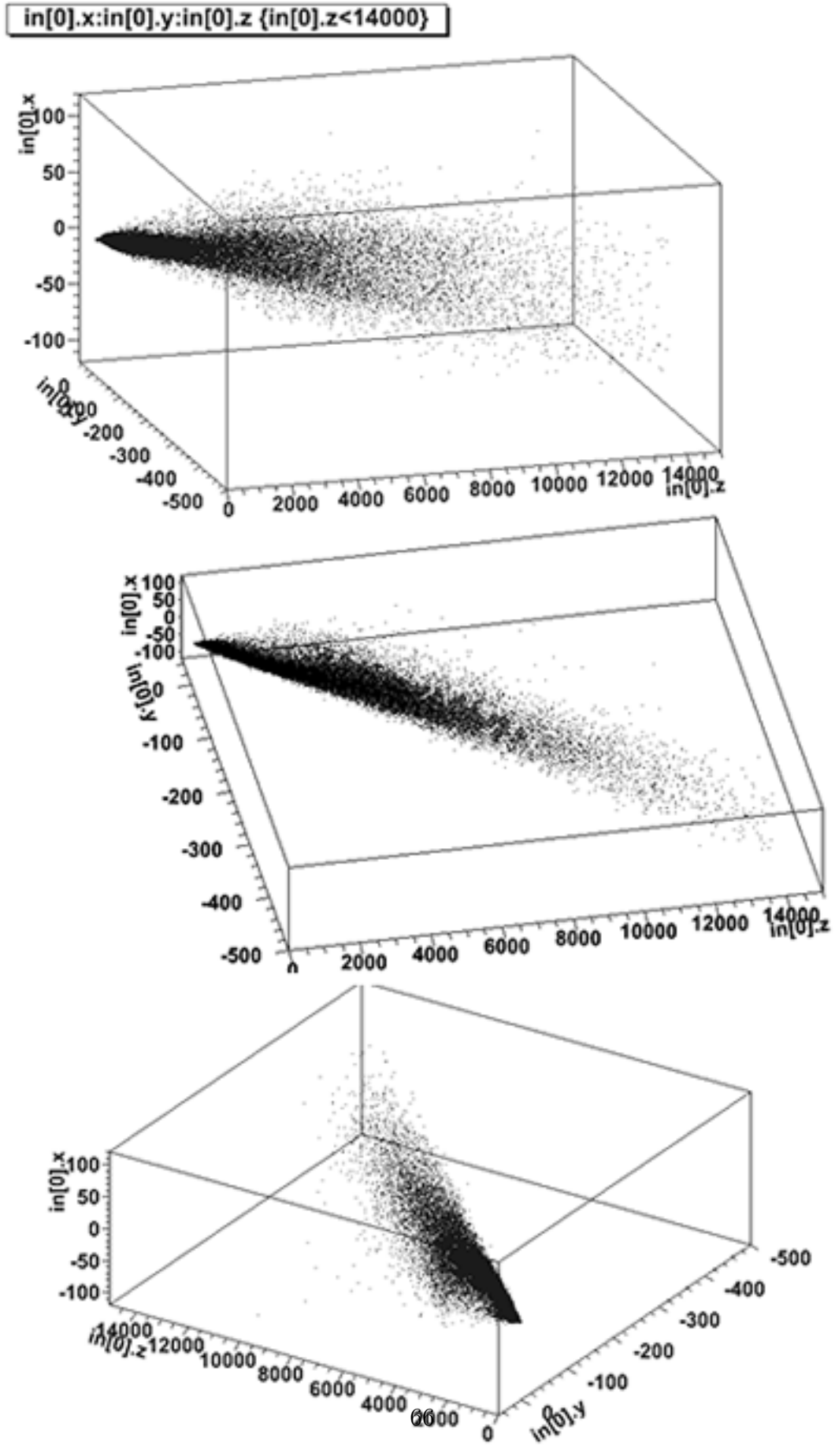
Rysunek A.10: Neutrina wyłapane w charakterystycznym miejscu detektora. Wykres przedstawia dane z NuWro metodą histogramową.



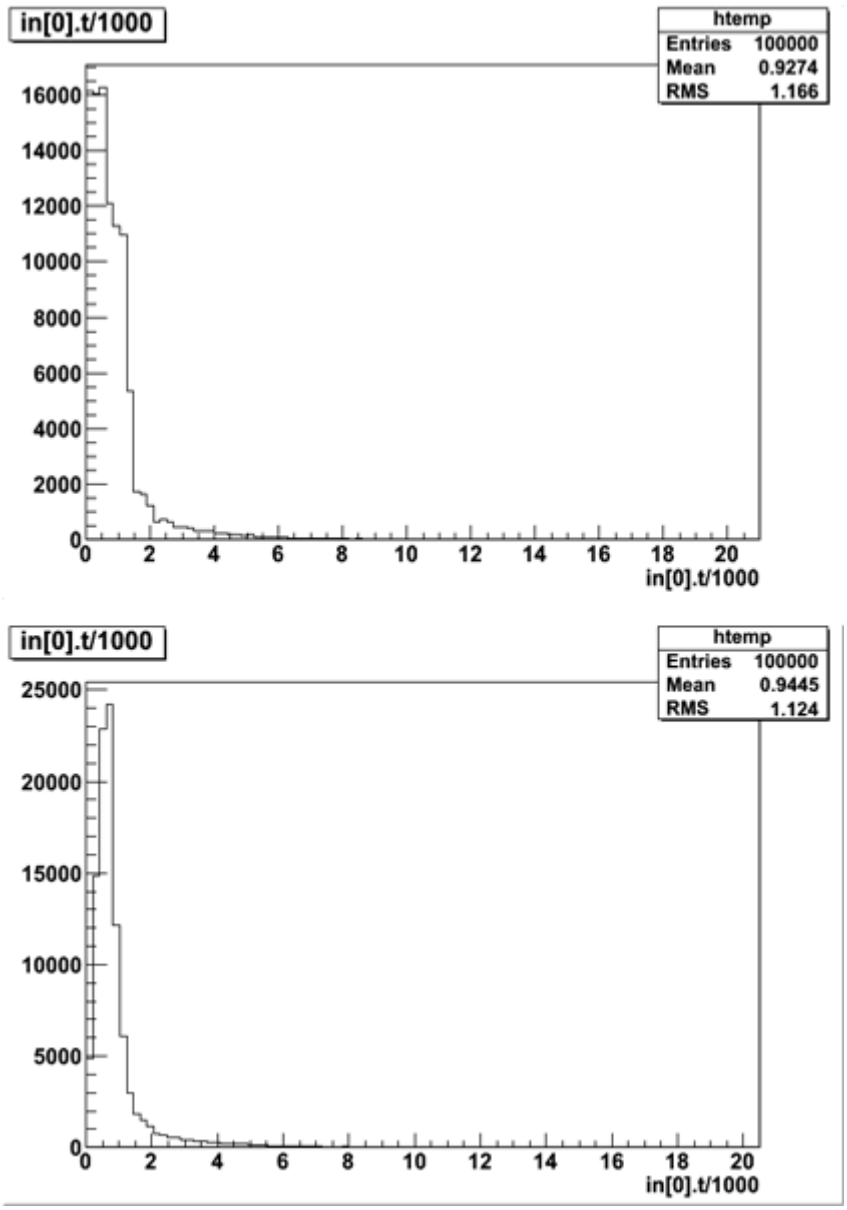
Rysunek A.11: Metoda dr C. Juszczaka



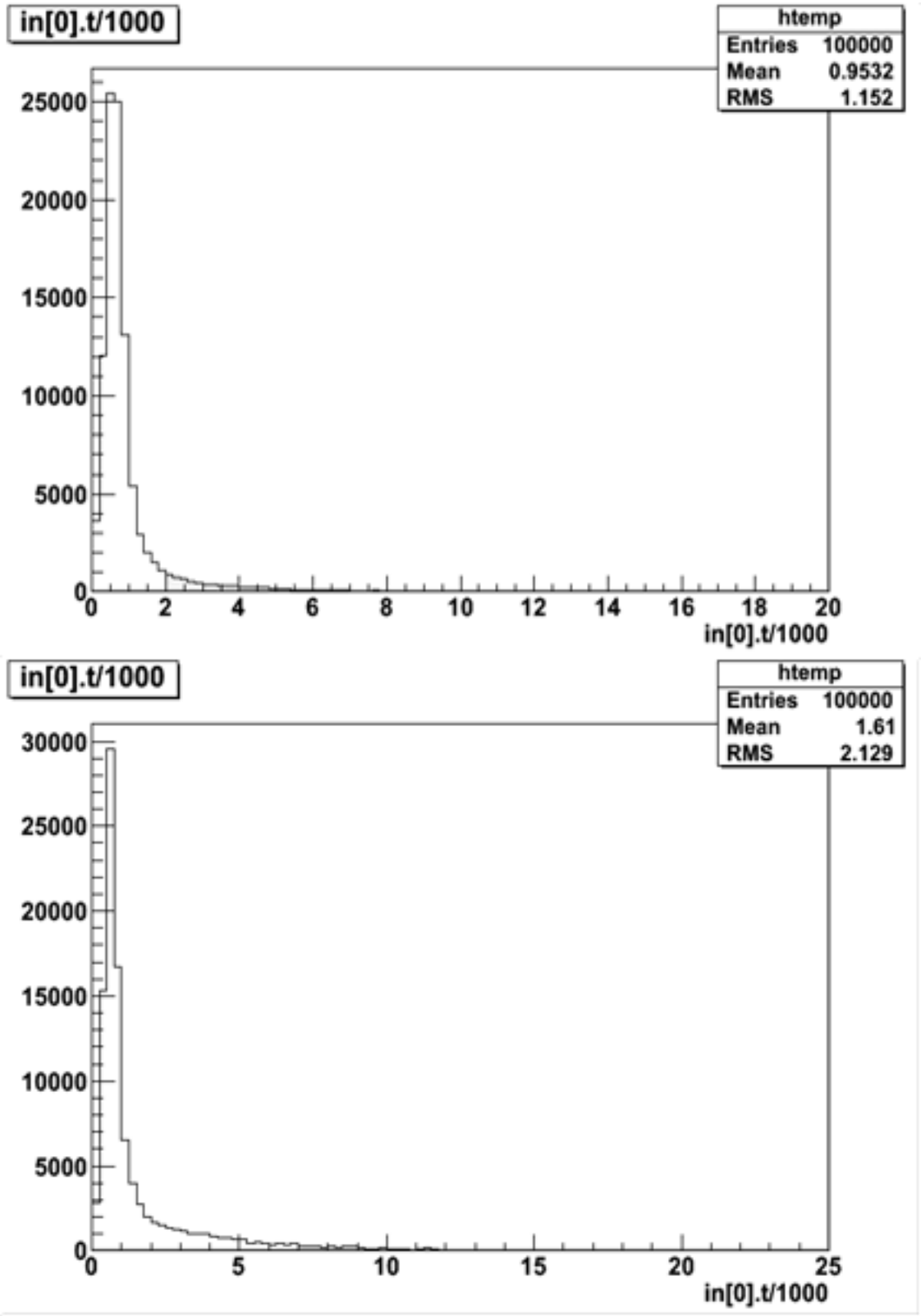
Rysunek A.12: Pęd neutrin w momencie zderzenia z jądrem (metoda histogramowa).



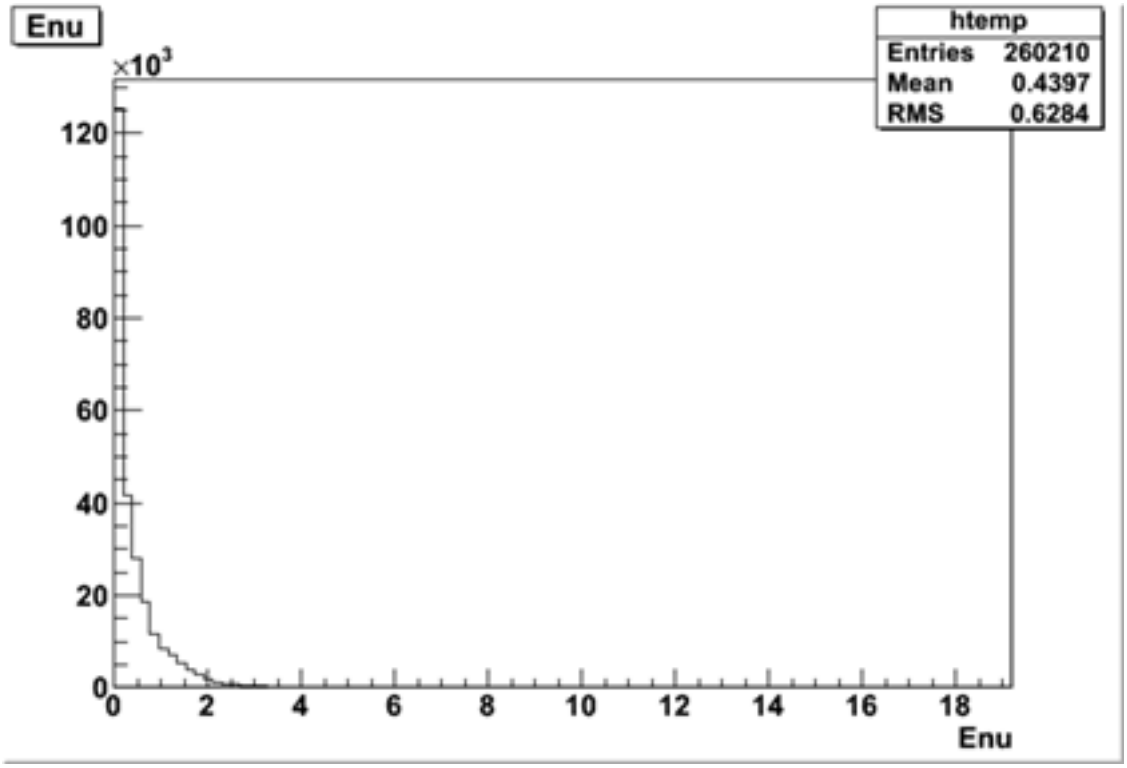
Rysunek A.13: Pęd neutrin w momencie zderzenia z jądrem (metoda dr C. Juszczaka).



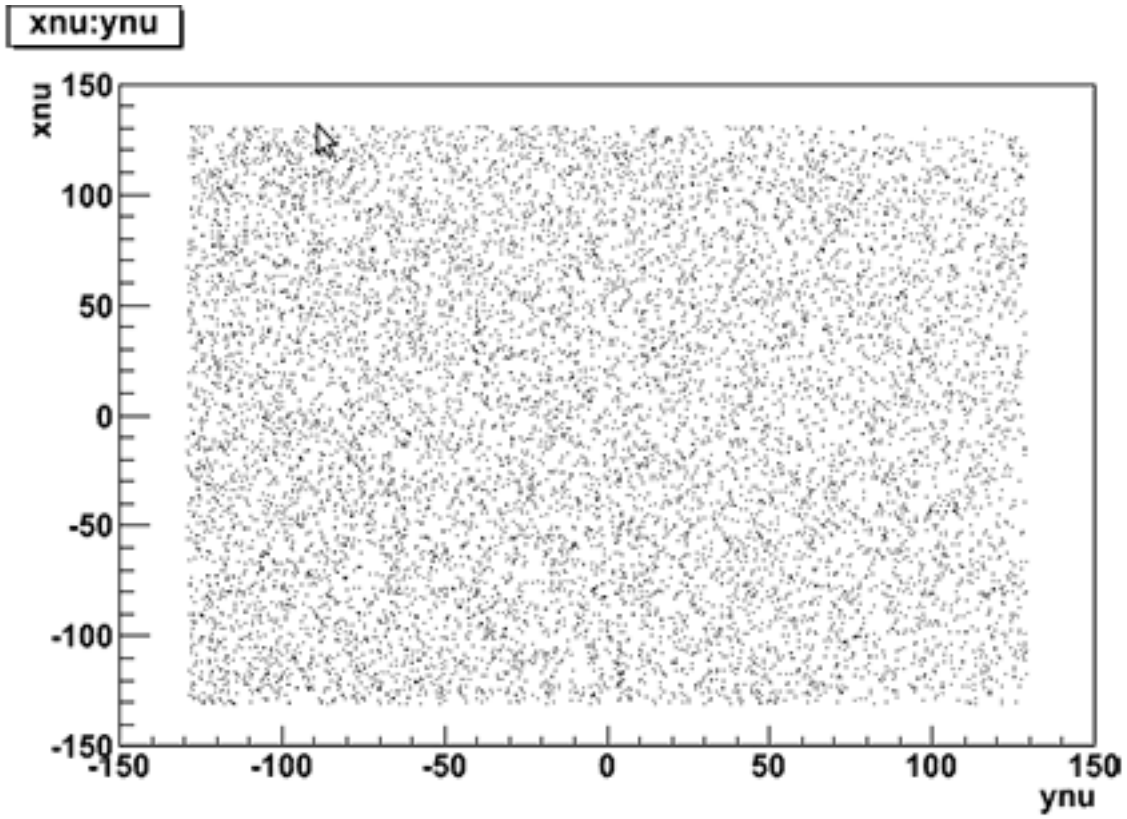
Rysunek A.14: Porównanie energii neutrin, metoda histogramowa. Górny zrobiony jest dla rozdzielczości 35 dla energii a dolny 80.



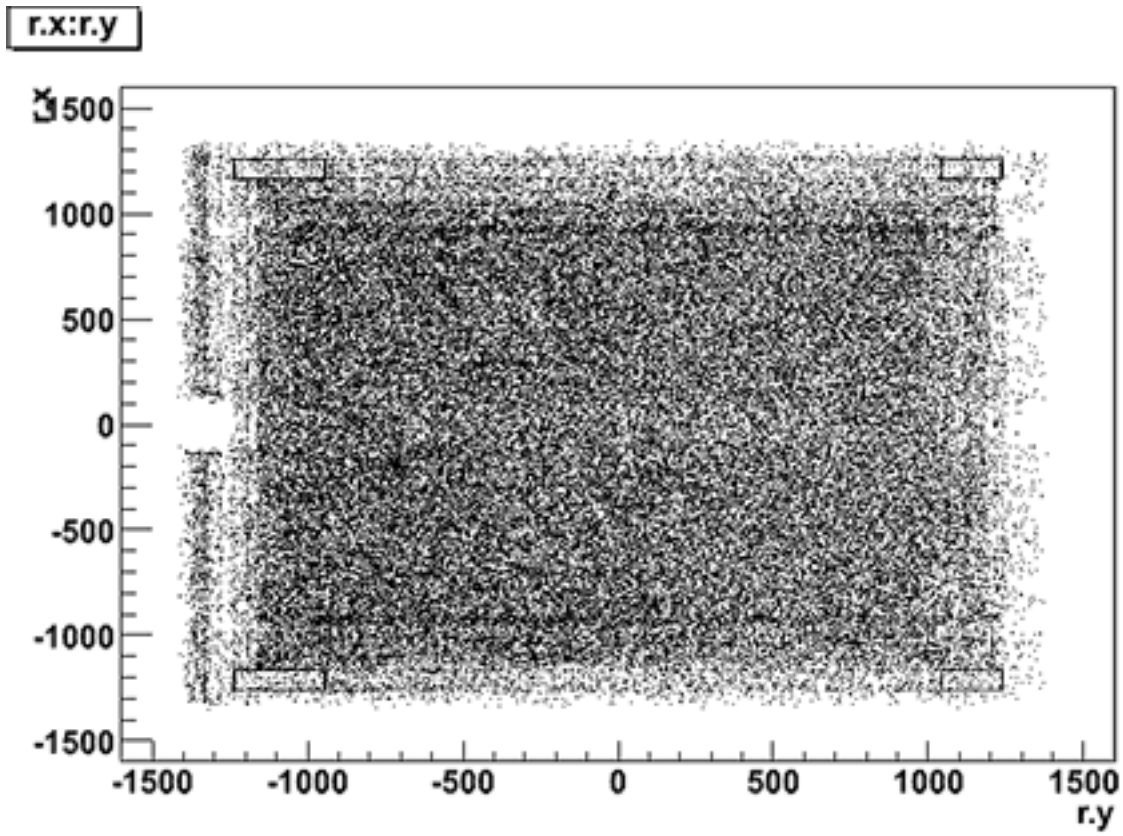
Rysunek A.15: Porównanie energii neutrin, metoda histogramowa i dr C. Juszczaka.



Rysunek A.16: Wzorcowy wykres energii, na podstawie plików root.



Rysunek A.17: Położenia neutrin generowanych przez wtyczkę, przed zaaplikowaniem ich do symulacji. Widoczna tutaj płaszczyzna $X \times Y$ jest czołem detektora - tutaj wiązka uderza w urządzenie. Położenia po symulacji przedstawia Rys.A.18.



Rysunek A.18: Położenia neutrin po przeprowadzonej symulacji. W odróżnieniu do rys. A.17 jest tu już uwzględniona konstrukcja detektora, jego dokładny kształt i rozmiary. Wartości są odpowiednio przeskalowane.

Dodatek B

Struktura pliku histogramowego

B.1. Histogram bez optymalizacji

Plik ma nazwę *histout.txt* i jest generowany po stworzeniu histogramu. W pierszej linii słowo extremes sygnalizuje, że w kolejnych znajdują się minima i maksima parametrów zapisanych w histogramie.

Pierwsza liczba w drugiej części pliku, tutaj jest to 5, oznacza ilość wymiarów histogramu. Kolejne liczby w ilość zgodnej z wymiarem to długości histogramu w kolejnych jego wymiarach. Inaczej mówiąc jest to rozdzielczość dla każdego parametru, przykładowo położenie na osi x rozciąga się na przedziale $< -131, 131 >$, zostanie on podzielony na 17 odcinków czyli dokładność z jaką histogram może odtworzyć położenie na tej osi jest równa 15.4.

Kolejny parametr mówi o ilości wszystkich elementów w histogramie. W tym przykładzie jest to 17^5 czyli 1419857. Po tym zaczyna się zapis elementów. Macierz n wymiarowa jest implementowana jako tablica jednowymiarowa z odpowiednim rozmieszczeniem elementów dlatego dane w pliku widzimy jako jednostajny ciąg cyfr.

```
extremes
min.xnu -1310
min.ynu -1290
min.nnu0 -0.066817
min.nnu1 -0.0480834
min.nnu2 0.997713
min.Enu 0.358551
min.norm 11408.4
max.xnu 1310
max.ynu 1290
max.nnu0 0.0300913
max.nnu1 0.0218572
max.nnu2 1
max.Enu 23406.1
max.norm 1.12756e+16
```

```
5 17 17 17 17 17 1419857 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

B.2. Histogram z usuniętymi nadmiarowymi zerami

Wyrażenie *nozero_array* oznacza, że plik przechowuje histogram zoptymalizowany, taki który nie posiada słupków o zerowej wysokości.Kolejne elementy to ilość wymiarów, rozdzielczości dla konkretnych wymiarów, ilość elementów. Należy zrówcić uwagę, w poprzedni pliku było 1419857 wartości a w tym 78055 czyli uzyskaliśmy wielokrotne kompresję, która ma największe znaczenie dla prędkości generowania cząstek.

Kolejno zapisywane wartośći to pary, index z oryginalnego histogramu i dpowiadająca mu wartość.

index	0	916	1020	1045	...	2600
wartość	0	1.86024e+10	8.05245e+10	8.05321e+10	...	1.7365e+14

```
extremes
min.xnu -1310
min.ynu -1290
min.nnu0 -0.066817
min.nnu1 -0.0480834
min.nnu2 0.997713
min.Enu 0.358551
min.norm 11408.4
max.xnu 1310
max.ynu 1290
max.nnu0 0.0300913
max.nnu1 0.0218572
max.nnu2 1
max.Enu 23406.1
max.norm 1.12756e+16

nozero_array 5 10 10 16 16 160 78055 0 0 916 1.86024e+10
1020 8.05245e+10 1045 8.05321e+10 2500 4.51981e+13
2501 7.45619e+13 2502 8.70012e+13 2503 9.05346e+13
2510 9.48323e+13 2511 9.66887e+13 2512 9.68481e+13
2513 9.74643e+13 2514 9.75994e+13 2520 9.75994e+13
2521 9.76816e+13 2523 9.8189e+13 2600 1.7365e+14
```

Dodatek C

Program konwertujący współrzędne Kokyo⇌Neut

```
#include <iostream>
#include <cmath>
#include "../../vec.h"

using namespace std;



/* Row * nnu[0] * nnu[1] * nnu[2] *  Enu *  xnu *  ynu *
*******************************************************
 * 0 * -0.005287 * -0.004780 * 0.9999746 * 0.3143146 *
 -20.81812 * 47.774734 *
 nnu          neutrino direction in the neut coordinate
 xnu, ynu     position neutrino at the detector in
              thedetector coordinates.
             NM pit - nd280 near module pit

 Our goal is to find kokyo zahyo vales for momentum
 and position (see above)
 *
 */




vec kokyo_to_neut(vec & v)
{
```

```
    const double TP = -10.147;
    const double theta = 0.0135942;
    // kz means kokyo zahyo
    double X_kz=v.x, Y_kz=v.y;

    // conversion of kokyo-zahyo coordinates to neut
    // (all in unit of meter)
    double X_neut = ( 49754.892 - X_kz ) * cos(theta)
                    + ( Y_kz - 69413.452) * sin(theta);

    double Y_neut = 2.00143 + TP;

    double Z_neut = ( 49754.892 - X_kz ) * sin(theta)
                    - ( Y_kz - 69413.452 ) * cos(theta)
                    - 8.929;

    return vec(X_neut,Y_neut,Z_neut);
}


vec to_kokyo(double x_neut, double z_neut)
{
    const double a = 49754.892;
    const double b = 69413.452;
    const double d = 8.929;
    const double t = 0.0135942; //theta
    const double ct = cos(t);
    const double st = sin(t);

    const double frac = ct*ct+st*st;
    const double A = st / frac;
    const double B = ct / frac;
    const double C1 = (a*st*st + a*ct*ct - d*st) / frac;
    const double C2 = b - d*ct/frac;

    vec v;
    v.x = C1 - A * z_neut - B * x_neut;
    v.y = C2 - B * z_neut + A * x_neut;

    return v;
}


int main()
{
```

```
    // center of whole magnet, off-axis
    double X_kz = 49754.185;
    double Y_kz = 69124.406;
    cout << "center of magnet in Kokyo coord ("
         << X_kz << "," << Y_kz << ")" << endl;

    vec kokyo_coord(X_kz,Y_kz,0.0);

    vec neut_coord = kokyo_to_neut(kokyo_coord);
    cout << "Neut " << neut_coord << endl;

    vec y_kokyo = to_kokyo(neut_coord.x, neut_coord.z);
    cout << "Kokyo (" << y_kokyo.x << ","
         << y_kokyo.y << ")" << endl;

    return 0;
}
```

Dodatek D

Fragmenty kodu C++

Kolejne podrozdziały zawierają fragmenty kodu NuWro które zostały zaimplementowane w trakcie powstawania tej pracy.

D.1. Generowanie cząstki w klasie beamRF

```
@kod dostępny w plikach nd280stats.h i nd280stats.cc
class Nd280Element
{
public:
        Float_t Enu;        /// energy
        Float_t nnu[3];     /// x, y, z direction of momentum
        Float_t xnu, ynu;   /// position in with Z=0
        Float_t norm;       /// weight (probability) of the element

        ...
};


@kod dostępny w pliku beamRF.h
virtual particle BeamRF::shoot()
{
        Nd280Element e;
        particle p( 14, 0.0 );

        // ask for next element
        if( NextElement( e ) == false )
        {
                // if was not possible to get next element
```

```
                // then back to the first file
                NextLoop();
                // get element
                NextElement( e );
        }

        // if this step ends up with last element
        // then notify the client
        if( _cb )
                if( LastElem() && LastFile() )
                        _cb->Done();

        // translate root file event to nuwro particle
        p.r.t = 0;
        double E = e.Enu * 1000;
        p.r.x = e.xnu * 10;
        p.r.y = e.ynu * 10;
        p.r.z = 0;
        p.t = E;
        p.x = e.nnu[0] * E;
        p.y = e.nnu[1] * E;
        p.z = e.nnu[2] * E;

        return p;
}

@kod dostępny w pliku rootf.h
***
* root file reader.
*/
template< typename T> class RootFReader
{
        TFile   *       fH;
        TTree   *       tree;
        T                       event;
        int                     count;

public:

        RootFReader( string fname, string treename )
        : fH( 0 ), tree( 0 ), count( 0 )
        {
                fH = new TFile( fname.c_str() );
                tree = static_cast<TTree*>(  fH->Get(treename.c_str())
);
```

```
                event.OpenBranches( tree );
                count = tree->GetEntries();
        }

        ~RootFReader()
        {
                Close();
        }

        void Close()
        {
                if( fH != 0 )
                {
                        fH->Close();
                        delete fH;
                        fH = 0;
                }
        }

        const int Count() { return count; }

        const T * GetEntry( const int & idx )
        {
                tree->GetEntry( idx );
                return &event;
        }

};
```

@kod dostępny w pliku rootf.h

```
/*************************************************************
* Object of this class looks for root files in the selected path.
* The second input variable, called treename,
* describes name of the tree inside a root file.
*/
template<typename T> class RootFolder
{
        T               *       _file;
        vector<string>  _fnames;
        DIR     *       _dp;
        string  _treename;

public:
        RootFolder( string path, string treename );
```

```
        ~RootFolder()
        {
                delete _file;
                if( _dp )
                        closedir( _dp );
        }

        const int Count() const { return _fnames.size(); }

        T * File( int i )
        {
                delete _file;
                _file = 0;
                if( i >= 0 && i < _fnames.size() )
                        _file = new T( _fnames[i], _treename );
                return _file;
        }
};


template<typename T>
RootFolder<T>::RootFolder( string path, string treename )
        : _file( 0 ), _dp( 0 ), _treename( treename )
{
        if( path[path.size() - 1] != '/' )
                path += '/';

        _dp = opendir( path.c_str() );

        while( true )
        {
                dirent * dirp = readdir( _dp );
                if( dirp == 0 )
                        break;
                string name( dirp->d_name );
                if( name.find( string(".root") )
                                        != string::npos )
                {
                        _fnames.push_back( path + name );
                }
        }
        if( _fnames.size() > 0 )
                cout << "List of root files ready to open:\n";
        for( int i = 0; i < _fnames.size(); ++i )
```

```
            cout<< i+1 << '\t' << _fnames[i] << endl;
}
```

D.2. Tworzenie histogramu

```
class Nd280Statistics
{
  typedef const double cdouble;
  typedef const unsigned int cuint;

  Nd280Element       _min;
  Nd280Element       _max;

  /// Number of elements under porocess.
  /// It's -1 if object is initialised from a file
  int                _count;

  /// values from root files must be changed to proper units,
  /// this is realted with difference between neutrinos' and
  /// (target) geometry units
  inline void AdjustUnits( Nd280Element & x );
public:
  Nd280Statistics() : _count(0) {}

  Nd280Statistics( string fname );

  Nd280Statistics(Nd280Element minimum, Nd280Element maximum)
  :_min(minimum),
  _max(maximum),
  _count(0)
  {}

  /// get min and maximum value in histogram
  Nd280Element GetMin() const { return _min; }
  Nd280Element GetMax() const { return _max; }

  /// get number of elements in histogram.
  /// this is equal -1 if object was init from a file
  int Count() const { return _count; }

  /// an element might be a min or maximum,
  /// use the function to check it
  void FindExtremes( const Nd280Element & entry )
```

```
  {
          if( _count == 0 )
          {
                  Init(entry);
          }
          else
          {
                  CheckMin( entry );
                  CheckMax( entry );
          }
          ++_count;
  }

  /// increment one pick in the histogram, another words
  /// icrease number of element in one histogram cell
  void Increment( Nd280Element x,  NArray< int > & hist )
  {
    AdjustUnits(x);
    /// Increase one cell in n-dimensional array (histogram).
    hist(
      Idx(x.xnu, _min.xnu, _max.xnu, hist.Count(0)),
      Idx(x.ynu, _min.ynu, _max.ynu, hist.Count(1)),
      Idx(x.nnu[0], _min.nnu[0], _max.nnu[0], hist.Count(2)),
      Idx(x.nnu[1], _min.nnu[1], _max.nnu[1], hist.Count(3)),
      Idx(x.Enu, _min.Enu, _max.Enu, hist.Count(4)))
        += x.norm;
  }

  /// return the string ready for saving in a file
  string String() const;

  /// fill the object using the file (input file stream)
  bool Load( ifstream & file );


private:
  /// check if x element is smaller then current minimum,
  /// if yes then overwrite old value
  void CheckMin( Nd280Element x );

  /// check if x element is larger then current maximum,
  /// if yes then overwrite old value
  void CheckMax( Nd280Element x );

  /// use it at the begining, run with first element
```

```
  void Init( Nd280Element entry );

  /// count one of indexes in N-dim array.
  /// bunch of these indexes discribe one cell
  /// which value will be incremented
  inline unsigned int Idx( cdouble & val,
        cdouble & min, cdouble & max,
        cuint & count );
};



unsigned int Nd280Statistics::Idx( cdouble & val, cdouble & min, cdouble
{
        double fraction = (val-min) * count;
        double scope = max - min;
        unsigned int result = static_cast<int>(fraction/scope);
        if( result >= count )
                result = count - 1;

        return result;
}


/// values from root files must be changed to proper units,
/// this is realted with difference between neutrinos' and
/// (target) geometry units
void Nd280Statistics::AdjustUnits( Nd280Element & x )
{
        x.xnu *= 10;
        x.ynu *= 10;
        x.Enu *= 1000;
}


@kod dostępny w pliku makeHist.h
template< typename TEvent, typename TStats >
class HistMaker
{
  /// folder with root files which
  /// handle specific type of event, e.g. Nd280Statistics
  RootFolder< RootFReader< TEvent > >          _folder;
  /// object knows what kind of root data
  /// we are looking for and how to
  /// fetch min, max values and also how
```

```
  /// to count index of histogram,
  /// e.g. Nd280Statistics
  TStats                                    _stats;

  /// looks for min and max value
  /// of each parameters inside event
  void GetStats();

public:
  HistMaker( string foldername, string treename )
    :_folder( foldername, treename )
    {}

  /// Create histogram
  void Create( NArray< int > & hist );

  /// Get access to statistics
  TStats * Stats();
};


/// looks for min and max value
/// of each parameters inside event
template< typename TEvent, typename TStats >
void HistMaker<TEvent, TStats>::GetStats()
{
  cerr << "HistMaker preparing statistics.";
  if( _folder.Count() <= 0 )
  {
    cerr << endl;
    return;
  }
  else
        cerr << endl << "Current file: ";

  int i = 0, j = 0;
  /// initialize parameters in first step
  RootFReader< TEvent > * file = _folder.File( i );

  ++j;

  /// go through all root files in folder
  while(  i < _folder.Count()  )
  {
        cerr << i << " ";
```

```
    /// and now... for all events
    /// in one currently opened root file
    for( ; j < file->Count(); ++j )
    {
      const TEvent * entry = file->GetEntry( j );
      _stats.FindExtremes( *entry );
    }
    j = 0;

    /// take next file
    file = _folder.File( ++i );
  }
  cerr << endl;
}



/// Create histogram
template< typename TEvent, typename TStats >
void HistMaker<TEvent, TStats>::Create( NArray< int > & hist )
{
  if( _stats.Count() <= 0 )
        GetStats();

  if( _stats.Count() <= 0 )
  {
    cerr << "error, number of elements
          in statistic's object is 0 or less" << endl;
    throw 0;
  }

  cerr << "HistMaker creates histogram,
        goes again through all files."
       << endl << "Current file: ";
  for( int i = 0; i < _folder.Count(); ++i )
  {
    RootFReader< TEvent > * file = _folder.File( i );
    cerr << i << " ";

    for( int j = 0; j < file->Count(); ++j )
    {
      const TEvent * entry = file->GetEntry( j );

      /// let TEvent decide which histogram
```

```
      /// cell should be increased
      _stats.Increment( *entry, hist );
    }
  }
  cerr << endl;
}


/// Get access to statistics
template< typename TEvent, typename TStats >
TStats * HistMaker<TEvent, TStats>::Stats()
{
  if( _stats.Count() > 0 )
  {
    return &_stats;
  }
  else
  {
    cerr << "error, statistics are not
        prepared yet, run GetStats() method first." << endl;
  }
  return 0;
}
```

D.3. Tworzenie neutrina na podstawie histogramu

```
@kod dostępny w pliku beamHist.h
BeamHist::BeamHist( int dim_res[EDims], string hist_fname )
    :_hist( new NonZeroArray<T>() ),
    _inf( EDims ),
    _f( EDims )
{
  ifstream histfile( hist_fname.c_str() );
  /// load min and max values for histogram
  _stats.Load( histfile );
  /// load height of the histogram picks
  _hist->Load( histfile );
  /// convert histogram

  _inf[0] = (_stats.GetMax().xnu - _stats.GetMin().xnu)
            / _hist->Count(0);
  _inf[1] = (_stats.GetMax().ynu - _stats.GetMin().ynu)
            / _hist->Count(1);
  _inf[2] = (_stats.GetMax().nnu[0] - _stats.GetMin().nnu[0])
            / _hist->Count(2);
```

```
    _inf[3] = (_stats.GetMax().nnu[1] - _stats.GetMin().nnu[1])
                / _hist->Count(3);
    _inf[4] = (_stats.GetMax().Enu - _stats.GetMin().Enu)
                / _hist->Count(4);

    _f[0] = _hist->Count(0);
    for(int i=1; i<EDims; ++i)
    {
      _f[i] = _f[i-1] * _hist->Count(i);
    }

}



particle BeamHist::shoot( bool dis )
{
    // ustawienie kodu pdg i masy neutrina
    particle part( 14, 0.0 );

    // tablica zostanie zapisana indeksami
    // dla wylosowanego neutrina
    int idx[7];

    // losowanie odbywa się na podstawie histogramu.
    // tutaj _hist jest obiektem klasy NArray<int>
    _hist->RandomIdx(idx);

    // Obliczenie parametrów neutrina
    // Przypisanie danych do nowej cząstki
    // Wygładzenie - etoda frandom() pozwala
    //     na rozmycie neutrin w przestrzeni
    // Tablica _inf (infinitezymalny) posida szerokość
    // przedziałów histogramu (słupka)
    // dla jednego konkretnego wymiaru
    part.r.t = 0;
    part.r.x = _stats.GetMin().xnu + (idx[0]+frandom())
                * _inf[0];
    part.r.y = _stats.GetMin().ynu + (idx[1]+frandom())
                * _inf[1];
    part.r.z = 0;

    // pęd i energia
    part.x = (_stats.GetMin().nnu[0] + (idx[2]+frandom())
                        * _inf[2]);
```

```
  part.y = (_stats.GetMin().nnu[1] + (idx[3]+frandom())
                    * _inf[3]);
  double pow = part.x*part.x + part.y*part.y;
        part.z = sqrt( 1 - pow );

  double energy = _stats.GetMin().Enu + (idx[4]+frandom()) * _inf[4];
        part.set_energy( energy );

  part.travelled = 1;

  // zwracane jest nowe neutrino
  return part;
}
```

@kod dostępny w pliku narray.h

```
template<typename T> void NArray< T >::RandomIdx( int idx[] )
{
  // to jest sprawdzenie i ewentualne wykonanie
  // sumowania na wszystkich elementach tablicy
  DoSum();

  // _arr jest jednowymiarową reprezentacją 5cio
  // wymiarowej macierzy. Kazdy kolejny element
  // jest sumą siebie i poprzedniego elementu.
  // Ostatni element tablicy, który widzimy po prawej
  // stronie mnożenia, to suma wszystkich wartości
  // w tej tablicy.
  // Po wymnożeniu losowej wartość z przedziału <0,1)
  // ze wspomnianą sumą otrzymujemy liczbę
  // losową w całym zbiorze punktów histogramu.
  // W następnym kroku zostanie odnaleziony jej index
  // za pomocą bisekcji.
  int x = frandom() * _arr[_count - 1];

  // sprawdzenie jakiemu indeksowi tablicy
  // jednowymiarowej odpowiada wylosowana wartość.
  // Metoda bisekcji zapewnia krótki czas
  // wyszukiwania elementu w posortowanej tablicy
  int i = Bisection( x );

  // pobranie prawdziwych danych opisujących neutrino
  // z macierzy 5cio wymiarowej.
  // _D to tablica zawierające ilości elementów
  // dla danego wymiaru, wynika to stąd, że
```

```
  // N wymiarowa macierz nie musi być jednakowo
  // wielka w każdym wymiarze.
  for( int a = 0; a < _dim; ++a )
  {
    idx[a] = i%_D[a];
    i/= _D[a];
  }
}


template<typename T> int NArray< T >::Bisection( cir x )
{
  // delikatna sugestia dla kompilatora żeby trzymał
  // te zmienne w miejscu o krótkim czasie dostępu,
  // generalnie w rejestrach
  register int a = 0;
  register int b = _count-1;
  register int i;

  // bisekcja ma złożoność czasową log(n)
  while( b > a )
  {
    i = (a+b)/2;

    if( x<_arr[i] )
      b = i;
    else
      a = i + 1;
  }

  // zwracamy index w tablicy dla szukanego
  // elementu
  return a;
}
```