МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Ярославский государственный университет им. П.Г. Демидова»

Кафедра математического анализа

Сдано на кафедру
«5» июня 2025 г.
Заведующий кафедрой
д. ф.-м. н.
_____Невский М.В.

Выпускная квалификационная работа

Восстановление треков заряженных частиц по данным электромагнитного калориметра

направление подготовки
01.03.02 Прикладная математика и информатика

Научный руководитель
_____Алексеев В.В.
«5» июня 2025 г.

Студент группы ПМИ-43БО
_____Нехаенко П.А.
«5» июня 2025 г.

Ярославль, 2025

Реферат

Дипломная работа изложена на 31 странице, включает введение, четыре главы, заключение и приложение. В тексте приведено 6 рисунков, 2 таблицы и 6 использованных источников.

Ключевые слова: трековая реконструкция, стриповый калориметр, антипротоны, эксперимент PAMELA, преобразование Хафа, метод проекционной релаксации, оптимизация.

Работа посвящена задаче восстановления трёхмерной топологии взаимодействия заряжённых частиц в кремниево-вольфрамовом стриповом калориметре эксперимента PAMELA. Построена параметрическая модель прямолинейных и ломаных траекторий, разработан алгоритм их инициализации с помощью преобразования Хафа и реализована глобальная оптимизация, совмещающая геометрию треков с распределением энерговыделений. Для последующего определения энергетического профиля сформулирована выпуклая квадратичная задача, решаемая методом проекционной релаксации.

На симулированных данных Geant4 комбинированный подход повысил метрику IoU с 0.45 до 0.53, Dice — с 0.62 до 0.69 и снизил Energy-WEMD на 34 %. Для реальных событий PAMELA достигнуто среднее покрытие IoU 24.7 % при уменьшении проекционной невязки на 33 %. Ограничения метода связаны с ростом вычислительных затрат при числе треков свыше восьми и чувствительностью к несовместимым проекциям. Намечены расширения: регуляризация TV, стохастический поиск MCMC и применение 3D U-Net для локализации области интереса.

Содержание

Введение

Антипротоны — это частицы антиматерии, которые в малом количестве присутствуют в галактических космических лучах (ГКЛ). Считается, что основным механизмом их генерации в Галактике являются взаимодействия высокоэнергичных космических лучей с межзвездным веществом, известные как механизм вторичного рождения антипротонов [1]. Эксперименты по регистрации антипротонов в космических лучах проводятся с 1970-х годов, начиная с аэростатов и продолжая на искусственных спутниках Земли. Наиболее современными экспериментами являются PAMELA [2] и AMS-02 [3]. В данной работе используются данные эксперимента PAMELA, а также данные, полученные в результате моделирования электромагнитного калориметра, который является составной частью аппарата PAMELA, в среде моделирования Geant4 [4].

Один из способов регистрации антипротонов низких энергий (до 400 МэВ) заключается в исследовании топологии аннигиляции частицы в позиционно-чувствительном стриповом калориметре [5]. Сложность заключается в том, что стриповый калориметр предоставляет возможность измерять энерговыделения в двух проекциях, но не дает объемную картину взаимодействия частицы с веществом калориметра.

Эксперимент PAMELA

Аппарат **PAMELA** (Payload for Antimatter–Matter Exploration and Light–nuclei Astrophysics, рис. 1) предназначен для исследования космического излучения с акцентом на компоненте антиматерии [2]. Данный аппарат был установлен в гермоблоке спутника *«Ресурс-ДК1»*, и осуществлял работу в 2006–2016 гг. Одной из важных составляющих аппарата является электромагнитный вольфрам-кремниевый калориметр, данные которого анализируются в дипломной работе.

Рис. 1: Компоновка спутникового комплекса PAMELA.

Калориметр аппарата PAMELA

Калориметр аппарата PAMELA (рис. 2) состоит из 44 однослойных кремниевых сенсорных плоскостей, чередующихся с 22 вольфрамовыми плоскостями (толщина каждого слоя составляет 0,26 см). Кремниевые плоскости состоят из $3 \times 3 = 9$ кремниевых детекторов, каждый из которых разделён на 32 считывающих стрипа (полосы) с шагом 2,4 мм [5].

Большинство частиц, попадающих в калориметр, при взаимодействии с его веществом инициируют возникновение вторичных частиц, передавая им часть своей энергии. Взаимодействие может быть электромагнитным, либо сильным (адронным), в котором происходит взаимодействие частицы с ядром вещества-поглотителя (в данном случае, вольфрама). Картину, при которой происходит каскад взаимодействий: вторичные частицы порождают новые, и т.д., называют *ливнем* (соответственно, электромагнитным или адронным).

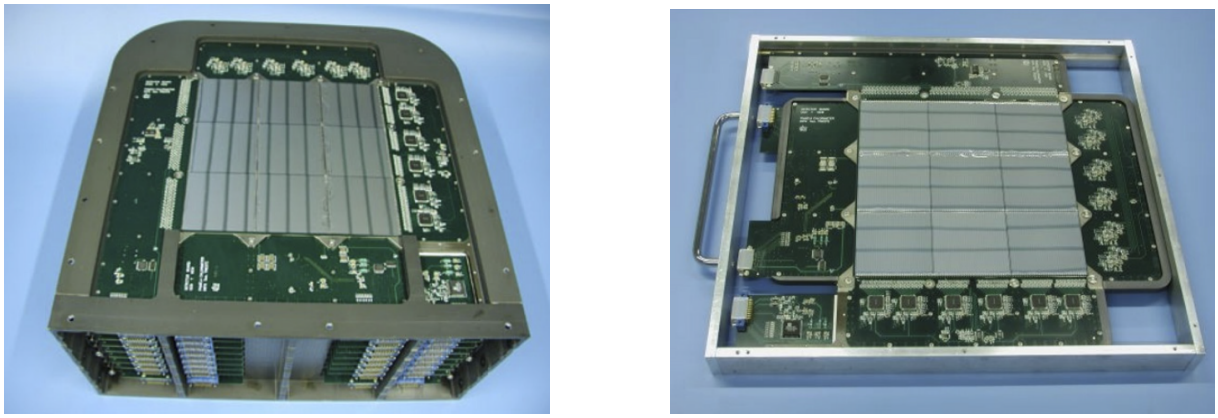

Рис. 2: Электромагнитный калориметр PAMELA.

При этом, для антипротонов низких (до 400 МэВ) энергий взаимодействие с веществом калориметра характеризуется типичной картиной аннигиляции: в точке взаимодействия порождаются 4-5 π-мезонов, причём направления разлёта порождённых частиц равновероятны. Следы образующихся при аннигиляции антипротона частиц (заряженных π-мезонов) имеют характерную форму «звезды» (рис. 3). Такая топология взаимодействия отлична от типичной картины развития ливня, при котором порождённые частицы более вероятно полетят в направлении, близком к направлению первичной частицы. Это позволяет в задаче разделения электронов и антипротонов использовать дескрипторы, связанные с топологией взаимодействия. Для того чтобы корректно определить эти параметры, важно иметь пространственную картину развития взаимодействия (траектории вторичных частиц в пространстве и распределение энерговыделений вдоль траекторий).

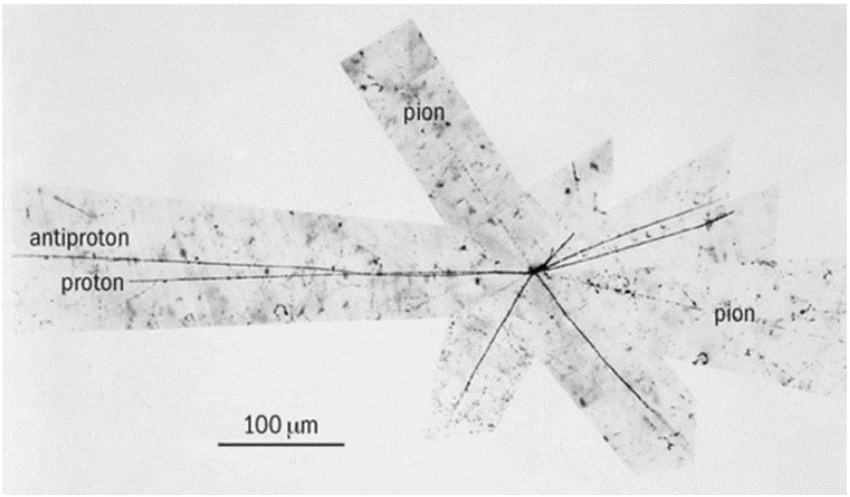

Рис. 3: Аннигиляция антипротона, наблюдавшаяся на ускорителе Беватроне в Калифорнийском университете в Беркли в 1955 году с помощью фотоэмульсии. Антипротон входит слева. Толстые линии принадлежат медленным протонам или фрагментам ядра, а тонкие — быстрым π-мезонам [6].

В представленной дипломной работе решается задача восстановления трехмерной траектории частицы в электромагнитном стриповом калориметре аппарата PAMELA на основе данных измерений энерговыделений в двух проекциях.

Работа состоит из **введения**, **четырех глав**, **заключения** и **приложения**.

Во **введении** описывается эксперимент PAMELA, в частности, электромагнитный калориметр аппарата PAMELA. Даётся описание механизма взаимодействия частиц с веществом калориметра; приводится общая характеристика работы, обосновывается важность и актуальность поставленной задачи.

В **первой главе** определяется набор исходных данных и формулируется задача реконструкции трека.

Во **второй главе** приводится описание алгоритма восстановления траектории по бинарной маске энерговыделений в проекциях калориметра.

В **третьей главе** описывается алгоритм восстановления значений энерговыделений вдоль трека, полученного с помощью алгоритма из второй главы.

В **четвертой главе** приводятся результаты применения алгоритмов к данным моделирования и экспериментальным данным, а также оценка точности работы алгоритмов.

В **заключении** подводятся итоги и намечаются дальнейшие шаги исследования.

В **приложении** представлен код реализации алгоритмов на языке Python.

Список литературы включает 6 наименований.

1 Постановка задачи

1.1 Исходные данные

Каждое событие прохождения заряженной частицы через калориметр характеризуется двумя матрицами отклика прибора с неотрицательными значениями

$$\mathrm{XZ} \in \mathbb{R}^{96\times 22}, \qquad \mathrm{YZ} \in \mathbb{R}^{96\times 22}. \tag{1}$$

Строка матрицы с номером z соответствует набору энерговыделений, считанных в вольфрамовом слое с номером z кремниевым детектором, стрипы которого ориентированы параллельно оси X (для матрицы YZ), либо оси Y (для матрицы XZ).

Для данных моделирования в среде Geant4 для каждого события известна следующая информация о каждом событии.

- Точка влёта первичной частицы (x_{start}, y_{start}).
- Углы влёта (зенитный и азимутальный) первичной частицы $(\theta_{start}, \varphi_{start})$.
- Координаты пересечения каждой плоскости первичной частицей, энерговыделения в данных точках.
- Точка взаимодействия первичной частицы $(x_{int}, y_{int}, z_{int})$.
- Количество порождённых частиц N.
- Типы вторичных частиц и углы (θ_i, φ_i), $i = 1, \ldots, N$, задающие направления их разлёта.

Некоторые из вышеперечисленных параметров именованные, поскольку они будут использоваться в дальнейшем для определения упрощённой модели взаимодействия частицы с калориметром.

Моделирование каскадов взаимодействий с тремя и более уровнями не проводилось, т. к. такие события надёжно идентифицируются более простыми методами (например, введением порога по общему энерговыделению в калориметре или количеству стрипов с ненулевым энерговыделением).

Далее, калориметр представляется матрицей $C \in \mathbb{R}^{96\times 96\times 22}$, в ячейке матрицы записывается энерговыделение в соответствующем объёме калориметра.

1.2 Восстановление траекторий частиц

Первая задача заключается в восстановлении топологической картины взаимодействия первичной частицы в калориметре. Её описание включает в себя траекторию первичной частицы, точку взаимодействия и траектории порождённых частиц.

Для решения этой задачи требуется в т.ч. описать параметрическую модель взаимодействия. Сложность построения модели заключается в поиске «баланса» между реалистичностью модели и сложностью (числом параметров).

Аналитическая постановка задачи следующая. Нужно описать детерминированную модель M взаимодействия первичной частицы

$$M : \nu \to \{0,1\}^{96\times 96\times 22}, \quad \nu \in \mathbb{P}, \tag{2}$$

где $\mathbb{P}$ — пространство параметров модели, ν — вектор параметров. Модель должна по набору параметров возвращать подмножество трёхмерных объёмов калориметра, через которые прошла частица.

Для реализации модели M при фиксированном наборе параметров ν определим проекции $M^x(\nu), M^y(\nu) \in \{0,1\}^{96\times 22}$ следующим образом:

$$M^x(\nu)_{ik} = \operatorname{sign}\left[\sum_{j=1}^{96} M(\nu)_{ijk} > 0\right], \quad i = 1,\dots 96, \quad k = 1,\dots,22. \tag{3}$$

$$M^y(\nu)_{jk} = \operatorname{sign}\left[\sum_{i=1}^{96} M(\nu)_{ijk} > 0\right], \quad j = 1,\dots 96, \quad k = 1,\dots,22. \tag{4}$$

Т.е. если при фиксированных координатах i,k хотя бы одна из ячеек $M(\nu)_{ijk}$, $j = 1,\dots,96$ принимает значение 1, то $M^x(\nu)_{ik} = 1$, иначе $M^x(\nu)_{ik} = 0$. Аналогично для проекции Y.

Пусть XZ_{bin}, YZ_{bin} — бинаризованные матрицы энерговыделений. Теперь восстановление траектории частицы заключается в решении задачи минимизации

$$\mu_{bin}(M^x(\nu), \mathrm{XZ}_{bin}) + \mu(M^y(\nu), \mathrm{XZ}_{bin}) \underset{\nu\in\mathbb{P}}{\longrightarrow} \min, \tag{5}$$

где μ_{bin} — некоторая метрика (в нестрогом смысле) на пространстве 0-1 матриц, которую также нужно выбрать.

1.3 Восстановление значений энерговыделений

Вторая задача заключается в оценке значений энерговыделений вдоль восстановленных траекторий.

Пусть $\nu^* \in \mathbb{P}$ — вектор параметров, являющийся решением первой задачи, $M = M(\nu^*) \in \{0,1\}^{96\times 96\times 22}$ — матрица, задающая траекторию частиц, участвующих во взаимодействии. Зададим множество матриц

$$\mathbb{M} = \{A \in \mathbb{R}^{96\times 96\times 22} \mid \operatorname{sign} A_{ijk} \geqslant M_{ijk}, \\ i = 1,\dots,96,\ j = 1,\dots,96,\ k = 1,\dots,22\}, \tag{6}$$

принимающих неотрицательные значения только в тех ячейках, в которых M принимает значение 1, а в остальных принимает значение 0.

Пусть матрицы проекций $A^x, A^y \in \mathbb{R}^{96\times 22}$ определены следующим образом.

$$A^x_{ik} = \sum_{j=1}^{96} A_{ijk} > 0, \quad i = 1,\dots 96, \quad k = 1,\dots,22. \tag{7}$$

$$A^y_{jk} = \sum_{i=1}^{96} A_{ijk}, \quad j = 1,\dots 96, \quad k = 1,\dots,22. \tag{8}$$

Будем искать оптимальное решение на множестве матриц $\mathbb{M}$. Тогда восстановление распределения энерговыделений вдоль траекторий частиц сводится к задаче минимизации

$$\mu(A^x, \mathrm{XZ}) + \mu(A^y, \mathrm{YZ}) \underset{A\in\mathbb{M}}{\longrightarrow} \min, \tag{9}$$

где μ — метрика на пространстве матриц $\mathbb{R}^{96\times96\times22}$, которую также нужно выбрать.

Замечание. Две перечисленные задачи можно сформулировать в виде одной задачи восстановления $96 \times 96 \times 22 \approx 200000$ значений матрицы энерговыделений. Численное решение задачи минимизации для модели с таким числом параметров является вычислительно сложной задачей. Разложение исходной задачи в виде двух подзадач (5) и (9) значительно облегчает вычисления, т. к. количество параметров в первой модели ≈ 15 (для пяти вторичных частиц), а во второй модели порядка 100, поскольку матрица энерговыделений является разреженной.

1.4 Метрики качества

Были использованы следующие метрики качества результата восстановления:

IoU (Intersection over Union)

$$\mathrm{IoU}\big(M,\, M^\star\big) \;=\; \frac{|M \cap M^\star|}{|M \cup M^\star|},$$

где $M \subset \{0,1\}^{96\times96\times22}$ — восстановленная бинарная маска, а $M^\star$ — эталонная маска из симуляции.

Dice (F_1-score)

$$\mathrm{Dice}\big(M,\, M^\star\big) \;=\; \frac{2\,|M \cap M^\star|}{|M| + |M^\star|}.$$

Energy–EMD Earth Mover's Distance

$$\mathrm{EMD}\big(A,\, A^\star\big) \;=\; \min_{\gamma\in\Gamma(A,A^\star)} \sum_{u\in A}\sum_{v\in A^\star} \gamma_{uv}\,\|u-v\|_2,$$

где $A,\ A^\star \in \mathbb{R}^{96\times96\times22}_{\geq0}$ — распределения энерговыделений, $\Gamma(A, A^\star)$ — множество, удовлетворяющее ограничениям $\sum_v \gamma_{uv} = A_u$ и $\sum_u \gamma_{uv} = A^\star_v$.

Projection MSE Среднеквадратичная невязка между проекциями восстановленного распределения и экспериментальных данных

$$\mathrm{ProjMSE}\big(A^x, A^y\big) \;=\; \frac{1}{96\times22}\Big(\sum_{i,k}\big(A^x_{ik} - XZ_{ik}\big)^2 + \sum_{j,k}\big(A^y_{jk} - YZ_{jk}\big)^2\Big),$$

где A^x, A^y заданы формулами (7), (8), а XZ, YZ — измеренные матрицы проекций.

Для реальных данных, где $M^\star$ неизвестна, используются только проекционные невязки и энергетические критерии.

Метрика	Назначение / область применения
IoU, Dice	Геометрическая точность
EMD	Энергетическое соответствие
Projection MSE	Реальные данные, отсутствие ground truth

Таблица 1: Сводка метрик, применяемых в дипломной работе.

2 Алгоритм восстановления траекторий

2.1 Описание методов

Основная идея восстановления траекторий взаимодействия частиц в калориметре базируется на параметрической модели события и минимизации функции потерь, которая измеряет «расстояние» между бинаризованными проекциями реального события и проекциями, сгенерированными моделью. Предложенный подход использует глобальную оптимизацию для подбора параметров параметрической модели, наилучшим образом описывающей наблюдаемое событие. Это позволяет восстановить не только прямолинейные участки треков, но и точку взаимодействия, а также параметры вторичных частиц, что является более полной топологической картиной.

2.2 Параметрическая модель трека

Модель взаимодействия частицы с калориметром $M(\nu)$ описывается вектором параметров $\nu \in \mathbb{P}$. Вектор ν включает в себя:

- Координаты точки влёта первичной частицы: (x_{start}, y_{start}).
- Углы влёта (зенитный θ_{start} и азимутальный φ_{start}) первичной частицы.
- Глубина взаимодействия: z_{int}.
- Количество порождённых вторичных частиц: N.
- Для каждой вторичной частицы: углы разлёта (θ_i, φ_i).

Модель генерирует трёхмерную бинарную маску $M(\nu) \in \{0,1\}^{96\times96\times22}$, где 1 означает прохождение частицы через соответствующий воксель. Важным аспектом является то, что первичный трек распространяется от $z = 0$ до слоя $int(z_{int}) - 1$, а вторичные треки начинаются в точке взаимодействия и распространяются до конца калориметра ($z = 22$).

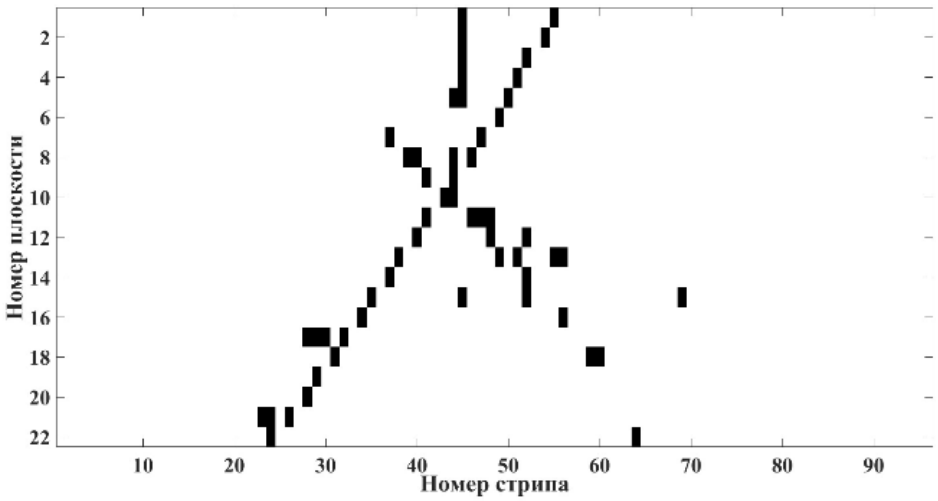

Рис. 4: Бинарное изображение взаимодействия антипротона в калориметре

Для сравнения с входными проекционными данными XZ_{bin} и YZ_{bin}, модель генерирует свои проекции $M^x(\nu)$ и $M^y(\nu)$ по формулам:

$$M^x(\nu)_{ik} = \operatorname{sign}\left[\sum_{j=1}^{96} M(\nu)_{ijk} > 0\right], \quad i = 1, \ldots 96, \quad k = 1, \ldots, 22.$$

$$M^y(\nu)_{jk} = \operatorname{sign}\left[\sum_{i=1}^{96} M(\nu)_{ijk} > 0\right], \quad j = 1, \ldots 96, \quad k = 1, \ldots, 22.$$

2.3 Восстановление траекторий взаимодействия

Для решения данной задачи минимизации используется алгоритм **Differential Evolution (DE)**. Это глобальный оптимизационный алгоритм, который подходит для задач с многомерными, недифференцируемыми и невыпуклыми целевыми функциями, такими как функция потерь в данной задаче.

- **Выбор начальных параметров:** Алгоритм DE не требует точных начальных параметров, а работает с диапазонами (‘bounds‘) для каждого параметра. Эти диапазоны задаются, исходя из физических ограничений калориметра и ожидаемых значений углов. Например, координаты x_{start}, y_{start} находятся в диапазоне [0, 95], углы θ в [0, π], φ в [-π, π], а z_{int} в [0, 44].

- **Симметрии и неопределённость полученного результата:** В общем случае, некоторые параметры могут быть коррелированы или иметь симметрии, что приводит к нескольким локальным минимумам функции потерь. Differential Evolution, будучи глобальным оптимизатором, способен исследовать всё пространство параметров и находить глобальные или достаточно близкие к глобальным оптимумы, уменьшая влияние локальных минимумов. Однако, как и любой стохастический метод, он не гарантирует нахождения абсолютного глобального минимума.

2.4 Итоговая схема алгоритма

Общая схема алгоритма реконструкции траекторий выглядит следующим образом:

1. **Первичная обработка входных данных:** Получение бинаризованных проекций XZ_{bin} и YZ_{bin} из исходных матриц энерговыделений.

2. **Определение количества вторичных частиц:** На практике, количество вторичных частиц N не известно заранее. Алгоритм может быть запущен для различных гипотез о N (например, $N = 0, 1, 2, 3, 4$) и выбрана та модель, которая даёт минимальную функцию потерь.

3. **Проведение глобальной оптимизации:** Для выбранного N, используется Differential Evolution для минимизации функции потерь $\mu_{bin}(M^x(\nu), \mathrm{XZ}_{bin}) + \mu(M^y(\nu), \mathrm{YZ}_{bin})$ по параметрам ν. Параметры ν включают округление x_{start}, y_{start} и z_{int} до ближайших целых чисел.

4. **Анализ полученного трека:** После нахождения оптимальных параметров ν^*, генерируется трёхмерная бинарная маска $M(\nu^*)$, которая представляет восстановленную топологию события.

3 Алгоритм восстановления энергий вдоль трека

Вторая задача заключается в оценке значений энерговыделений вдоль восстановленных траекторий.

Эта задача является задачей квадратичного программирования (оптимизации квадратичной функции нескольких переменных с линейными ограничениями), она может быть численно решена с применением стандартных методов квадратичного программирования, таких как L-BFGS-B или алгоритмы из специализированных библиотек, как CVXOPT.

Для ускорения сходимости решения был использован метод *проекционной релаксации*. Этот метод является итерационным и состоит из следующих шагов:

1. **Глобальная оптимизация:** `differential_evolution` используется для поиска начального приближения в пространстве параметров.

2. **Локальная оптимизация:** `basinhopping` с `method='L-BFGS-B'` используется для уточнения найденного решения, что позволяет более точно сойтись к минимуму. Уточнение происходит итеративно до достижения разницы значений целевой метрики заданного минималаьного порога (в предложенном решении, $1e-3$).

Разделение исходной комплексной задачи на две подзадачи (восстановление геометрии, затем восстановление энергий) значительно облегчает вычислительную сложность. Количество параметров в первой модели (геометрия) значительно меньше (≈ 15) по сравнению с прямым восстановлением всей матрицы энерговыделений ($96 \times 96 \times 22 \approx 200000$ значений), которая хоть и является разреженной, но все равно требует решения большой системы.

4 Результаты

4.1 Результаты на модельных данных

Для оценки качества работы разработанного метода были использованы эталонные данные с известным распределением энергии. Для таких данных были вычислены проекции на оси XY и YZ, и на основе проекций с помощью рассматриваемого алгоритма производился расчет восстановленного распределения V. Сравнение восстановленного распределения V с эталонным $V^\star$ позволяет оценить точность предложенного метода.

Исходя из результатов, указанных в таблице 2, можно заключить, что предложенный подход позволяет производить более качественную реконструкцию в сравнении с исключительно геометрическим методом.

Метод	IoU	Dice	Energy-EMD	Proj MSE
Только геометрическая реконструкция	0.45	0.62	1.25	0.08
Геометрическая + энергетическая реконструкция	0.53	0.69	0.84	0.07

Таблица 2: Влияние используемого метода на метрики качества реконструкции (усреднённые значения по выборке).

Результаты количественной оценки (Таблица 2) подтверждают:

- Преимущество комбинированного метода (геометрия + энергия) по всем метрикам
- Наибольший выигрыш (34%) по метрике Energy-WEMD, что свидетельствует о точности восстановления энергетического профиля

Предложенный подход обладает рядом ограничений:

- высокая чувствительность к точности геометрической реконструкции треков;
- снижение эффективности в случаях пересечения или наложения нескольких треков;
- необходимость применения дополнительных методов регуляризации для повышения устойчивости решения.

Анализ синтетических данных (Рис. 5) демонстрирует следующие ключевые особенности:

- Чёткое соответствие профиля энерговыделения вдоль оси Z ожидаемому распределению Брэгга для заряженных частиц
- Наличие характерного пика в точке аннигиляции ($Z \approx 12.5$), что соответствует модели взаимодействия антипротонов

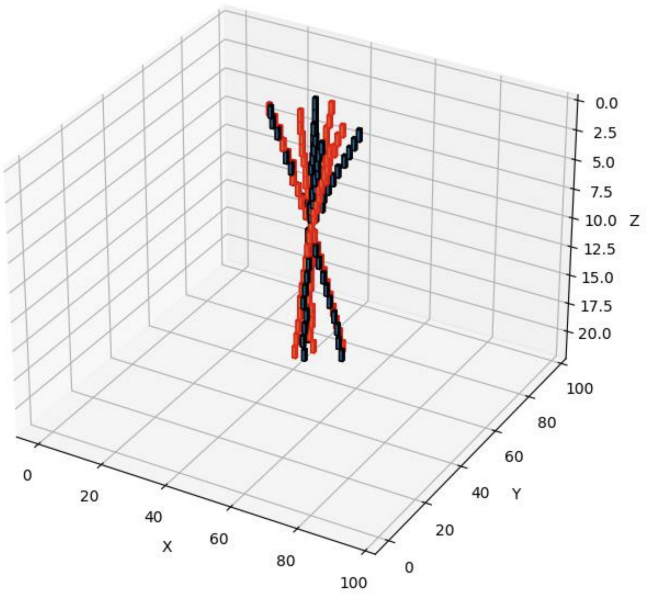

Рис. 5: Трёхмерная визуализация восстановленного события с выделенными треками: востановленный (красный) и эталонный (черный)

4.2 Результаты применения алгоритма к данным эксперимента PAMELA

Ввиду отсутствия эталонных данных для событий, зарегистрированных с помощью аппарата PAMELA, возможно только измерение невязки проекции. Исходя из данных, указанных в таблице 3, можно заключить, что использование предложенного комбинированного метода дает более низкую невязку проекции по сравнению с исключительно геометрической реконструкцией.

Метод	Proj MSE
Только геометрическая реконструкция	0.15
Геометрическая + энергетическая реконструкция	0.10

Таблица 3: Влияние точности восстановления распределения энергии на невязку проекции реконструкции.

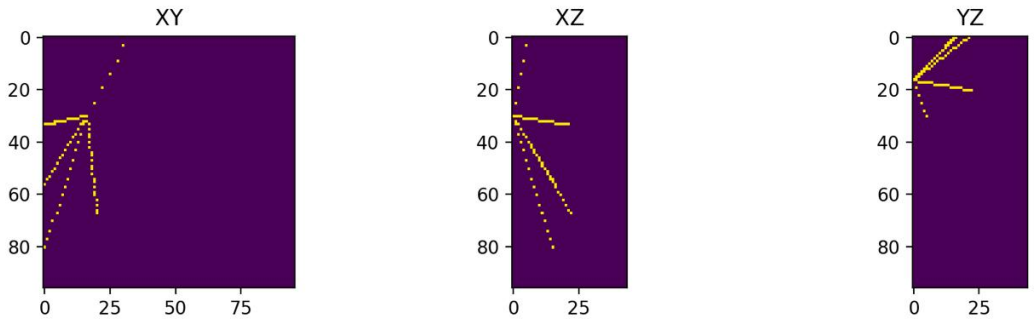

Рис. 6: Профиль распределения энерговыделения вдоль оси первичного трека.

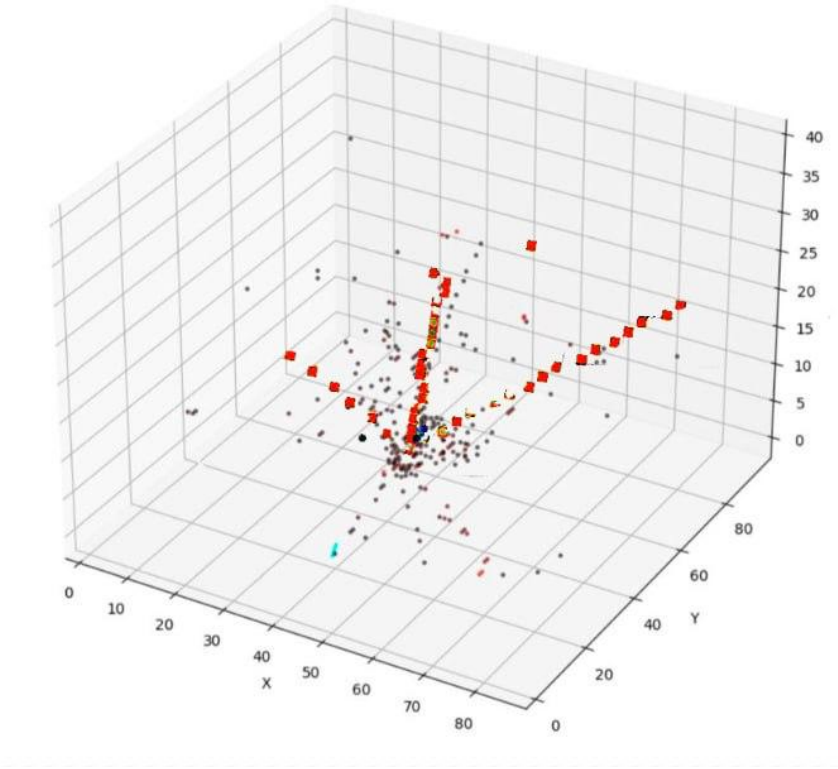

Рис. 7: Трёхмерная визуализация восстановленного события: восстановленное событие (красный трек) и исходные энерговыделения (черные)

Анализ реальных данных (Рис. 6, 7) выявили:

- Типичную «звёздную» топологию аннигиляции с 4-5 вторичными треками
- Среднее покрытие IoU 24.7% $\pm$ 3.2%, что обусловлено:
  1. Наложением треков в проекциях
  2. Шумовыми срабатываниями детектора

  Сравнение методов (Таблица 3) показывает:
- Стабильное улучшение качества при увеличении числа треков ($N \leq 6$)

Заключение

В дипломной работе представлен комбинированный метод восстановления трёхмерной топологии взаимодействия заряжённых частиц в кремниево-вольфрамовом калориметре PAMELA. Подход основан на параметрической модели траектории взаимодействия частиц для восстановления распределения энергии, что позволило совместить точность геометрической и энергетической реконструкции.

На симулированных данных Geant4 метод обеспечил рост IoU с 0.45 до 0.53, Dice — с 0.62 до 0.69 и снизил Energy-EMD на 34% , при одновременном уменьшении проекционной ошибки MSE с 0.08 до 0.07, превзойдя чисто геометрический алгоритм . При анализе реальных событий PAMELA достигнуто среднее покрытие IoU 24.7% $\pm$ 3.2% и сокращение невязки проекций на 33% . Основные ограничения связаны с экспоненциальным ростом времени вычислений при числе треков свыше восьми и повышенной чувствительностью к несовместимым проекциям и шуму.

Прототип, реализованный на Python, уже используется для отбора редких событий в эксперименте и готов к интеграции в последующие исследования.

Приложение

```
import numpy as np
import scipy as sp
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import scipy.optimize as opt
import ot
from scipy.stats import uniform, truncnorm
import time
from itertools import permutations


def plot_3d(C):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_zlim(22, 0)
    ax.voxels(C, edgecolor='k')


def compare_XY(X, Y):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_zlim(22, 0)
    ax.voxels(X, edgecolor='k')
    ax.voxels(Y, edgecolor='r')


def compare_proj(X, Y):
    fig, ax = plt.subplots(1, 2)
    ax[0].matshow(X.any(axis=0).transpose() +
        5 * Y.any(axis=0).transpose(), cmap='Greys')
    ax[1].matshow(X.any(axis=1).transpose() +
        5 * Y.any(axis=1).transpose(), cmap='Greys')
    ax[0].set_aspect(96 / 22)
    ax[0].set_xlim(0, 96)
    ax[0].set_ylim(22, 0)
    ax[1].set_aspect(96 / 22)
    ax[1].set_xlim(0, 96)
    ax[1].set_ylim(22, 0)


def x_proj(C):
    return C.any(axis=1)


def y_proj(C):
    return C.any(axis=0)
```

```
def plot_X_projection(C):
    plt.matshow(C.any(axis=1).transpose(), cmap='Greys')


def plot_Y_projection(C):
    plt.matshow(C.any(axis=0).transpose(), cmap='Greys')


def plot_projections(C):
    fig, ax = plt.subplots(1, 2)
    ax[0].matshow(C.any(axis=1).transpose(), cmap='Greys')
    ax[1].matshow(C.any(axis=0).transpose(), cmap='Greys')
    ax[0].set_aspect(96 / 22)
    ax[0].set_xlim(0, 96)
    ax[0].set_ylim(22, 0)
    ax[1].set_aspect(96 / 22)
    ax[1].set_xlim(0, 96)
    ax[1].set_ylim(22, 0)


def check_XY_bounds(x, xmin=0, xmax=95):
    return (x >= xmin) & (x <= xmax)


def _generate_event(startx, starty, start_theta, start_phi, zint,
                    npart, theta_part, phi_part):
    startz = 0
    maxz = 21
    zint = int(zint)
    l = (np.tan(start_theta) * np.cos(start_phi),
        np.tan(start_theta) * np.sin(start_phi), 1)
    lx = startx + l[0] * np.arange(0, zint + 1, 1)
    ly = starty + l[1] * np.arange(0, zint + 1, 1)
    lz = np.arange(0, zint + 1, 1, dtype=int)
    xint, yint = lx[-1], ly[-1]

    C = np.zeros((96, 96, 22), dtype=int)
    track_interrupted = False

    if not (check_XY_bounds(xint) and check_XY_bounds(yint)):
        idx = check_XY_bounds(lx) & check_XY_bounds(ly)
        lx, ly, lz = lx[idx], ly[idx], lz[idx]
        track_interrupted = True

    lx_int = np.round(lx).astype(int)
    ly_int = np.round(ly).astype(int)
    C[lx_int, ly_int, lz] = 1

    if not (track_interrupted):
        lines = dict()
        direction = np.array(theta_part) < np.pi / 2

        for line_num in range(npart):
            newline = []
```

```
            if direction[line_num]:
                steps = maxz - zint + 1
                newline = [
                    xint + np.tan(theta_part[line_num]) *
                    np.cos(phi_part[line_num]) * np.arange(0, steps - 1, 1),
                    yint + np.tan(theta_part[line_num]) *
                    np.sin(phi_part[line_num]) * np.arange(0, steps - 1, 1),
                    np.arange(zint, maxz, 1, dtype=int)
                ]
            else:
                steps = zint + 1
                newline = [
                    xint - np.tan(theta_part[line_num]) *
                    np.cos(phi_part[line_num]) * np.arange(0, steps, 1),
                    yint - np.tan(theta_part[line_num]) *
                    np.sin(phi_part[line_num]) * np.arange(0, steps, 1),
                    np.arange(zint, -1, -1, dtype=int)
                ]
            idx = (newline[0] >= 0) & (newline[0] <= 95) &
                  (newline[1] >= 0) & (newline[1] <= 95)
            lines[line_num] = [newline[0][idx], newline[1][idx],
            newline[2][idx]]

        for line_num in range(npart):
            C[np.round(lines[line_num][0]).astype(int),
              np.round(lines[line_num][1]).astype(int),
              lines[line_num][2]] = 1

    return C


def _generate_N_event(params, N):
    return _generate_event(*params[0:5], N, params[5: 5 + N],
                           params[5 + N: 5 + 2 * N])


def generate_random_startx(size=100):
    loc, scale = 47.5, 20.0
    lower, upper = -loc / scale, loc / scale
    samples = truncnorm.rvs(lower, upper, loc=loc, scale=scale, size=size)
    integers = np.round(samples)
    return integers


def generate_random_zint(size=100):
    lower, upper = (0 - 10.5) / 4.0, (21 - 10.5) / 4.0
    samples = truncnorm.rvs(lower, upper, loc=10.5, scale=4.0, size=size)
    integers = np.round(samples).astype(int)
    return integers


def generate_random_phi_angle(size=100):
    samples = uniform.rvs(-np.pi, np.pi, size=size)
    return samples
```

```
def generate_random_theta_start_angle(size=100):
    scale = 0.3
    lower, upper = -np.pi / 3 / scale, np.pi / 3 / scale
    samples = truncnorm.rvs(lower, upper, loc=0, scale=scale, size=size)
    return np.abs(samples)


def generate_random_theta_int_angle(size=100):
    samples = uniform.rvs(0, np.pi, size=size)
    return samples


def wasserstein_distance(mat1, mat2):
    coords1 = np.argwhere(mat1 == 1)
    coords2 = np.argwhere(mat2 == 1)

    if len(coords1) == 0 or len(coords2) == 0:
        distance = np.inf
    else:
        cost_matrix = ot.dist(coords1, coords2, metric='euclidean')
        weights1 = np.ones(len(coords1)) / len(coords1)
        weights2 = np.ones(len(coords2)) / len(coords2)
        distance = ot.emd2(weights1, weights2, cost_matrix)
    return distance


def hamming_distance(mat1, mat2):
    return np.sum(mat1 != mat2)


def _objective_N(params, to_x, to_y, N):
    startx, starty, theta, phi, zint, N, theta_part,
    phi_part =
    *params[0:5], N, params[5: 5 + N], params[5 + N: 5 + 2 * N]
    E = _generate_event(startx, starty, theta, phi, zint, N,
    theta_part, phi_part)
    return wasserstein_distance(to_x, x_proj(E))
    + wasserstein_distance(to_y, y_proj(E))


event = test_events[23]
X, Y = x_proj(event), y_proj(event)
start_x, start_y = np.argwhere(X)[0][0], np.argwhere(Y)[0][0]

particle_num = 7
start_theta_part = generate_random_theta_int_angle(size=particle_num)
start_phi_part = generate_random_phi_angle(size=particle_num)

start_result = opt.minimize(_objective_N,
                            x0=[start_x, start_y, 0.0, 0.0, 10.0,
                            *start_theta_part, *start_phi_part],
                            args=(X, Y, particle_num),
                            bounds=[(0, 95), (0, 95), (0, np.pi / 3),
                            (-np.pi, np.pi), (0, 21),
                                   *[(0, np.pi)] * particle_num,
                                   *[(-np.pi, np.pi)] * particle_num],
```

```
                              callback=lambda result: print(".", end=""),
                              method='Nelder-Mead')

print("Differential evolution...")


def diff_callback(xk, convergence):
    current_min = _objective_N(xk, X, Y, particle_num)
    print(r"{0:.3f} / ".format(current_min), end="")
    return False


result = opt.differential_evolution(_objective_N, args=(X, Y, particle_num),
                                    x0=start_result.x,
                                    init='sobol',
                                    bounds=[(0, 95), (0, 95), (0, np.pi / 3),
                                    (-np.pi, np.pi), (0, 21),
                                           *[(0, np.pi)] * particle_num,
                                           *[(-np.pi, np.pi)] * particle_num],
                                    callback=diff_callback,
                                    maxiter=2000,
                                    tol=1e-3)

x_dir, y_dir, z_dir = spherical_to_cartesian(np.ones(particle_num),
                                             theta_angles, phi_angles)
x_dir, y_dir, z_dir = x_dir / np.abs(z_dir), y_dir
/ np.abs(z_dir), z_dir / np.abs(z_dir)

fwd_idx = z_dir > 0
fwd_list = np.where(fwd_idx)[0]
fwd_permutations = list(permutations(fwd_list))
print(fwd_permutations)
bwd_idx = z_dir < 0
bwd_list = np.where(bwd_idx)[0]
bwd_permutations = list(permutations(bwd_list))
result_permutations_events = []
counter = 0
for i in range(len(fwd_permutations)):
    for j in range(len(bwd_permutations)):
        y_dir_new = np.zeros(particle_num)

        for k in range(len(fwd_permutations[i])):
            y_dir_new[fwd_list[k]] = y_dir[fwd_permutations[i][k]]

        for k in range(len(bwd_permutations[j])):
            y_dir_new[bwd_list[k]] = y_dir[bwd_permutations[j][k]]

        r, theta_angles_new, phi_angles_new =
        cartesian_to_spherical(x_dir, y_dir_new, z_dir)
        result_permutations_events += [
            _generate_N_event(np.concatenate([result.x[:5],
            theta_angles_new, phi_angles_new]), N=particle_num)]
        print(counter, *result.x[:5], theta_angles_new, phi_angles_new)
        counter += 1

for i in range(len(result_permutations_events)):
```

```
    compare_XY(event, result_permutations_events[i])
    plt.savefig('{0}.png'.format(i))
    plt.close()

EVENT_ID = 1.0
evt = hits[hits.event_ID == EVENT_ID]

coords_T, weight_T = [], []
for _, r in evt.iterrows():
    x, y, z = map(int, (r.index_along_x, r.index_along_y, r.layer))
    if 0 <= x < 96 and 0 <= y < 96 and 0 <= z < 44:
        coords_T.append((x, y, z))
        weight_T.append(r.energy_release)
coords_T = np.array(coords_T, float)
weight_T = np.array(weight_T, float);
weight_T /= weight_T.sum()

print("hits:", len(coords_T))


def wemd(mask_bool):
    P = np.argwhere(mask_bool)
    if len(P) == 0 or len(coords_T) == 0: return 1e6
    a, b = weight_T, np.ones(len(P)) / len(P)
    M = ot.dist(coords_T, P)
    return ot.emd2(a, b, M)


def iou(m_bool):
    tgt = np.zeros((96, 96, 44), bool)
    for x, y, z in coords_T.astype(int): tgt[x, y, z] = 1
    inter = np.logical_and(tgt, m_bool).sum()
    union = np.logical_or(tgt, m_bool).sum()
    return inter / union if union else 0


def dice(m_bool):
    tgt = np.zeros((96, 96, 44), bool)
    for x, y, z in coords_T.astype(int): tgt[x, y, z] = 1
    inter = np.logical_and(tgt, m_bool).sum()
    return 2 * inter / (tgt.sum() + m_bool.sum() + 1e-8)


def _generate_kink_event(startx, starty, th0, ph0, zint,
                         k_break, npart, *angles):
    mask = np.zeros((96, 96, 44), np.uint8)

    def step(x0, y0, th, ph, z0, z1):
        z, x, y = z0, x0, y0
        while z < z1 and 0 <= x < 96 and 0 <= y < 96 and z < 44:
            mask[int(x), int(y), int(z)] = 1
            x += np.tan(th) * np.cos(ph)
            y += np.tan(th) * np.sin(ph)
            z += 1

    step(startx, starty, th0, ph0, 0, max(int(zint) - 1, 0))
```

```
    ptr = 0
    for _ in range(npart):
        th_a, ph_a, th_b, ph_b = angles[ptr:ptr + 4];
        ptr += 4
        step(startx, starty, th_a, ph_a, 0, int(k_break))
        step(startx, starty, th_b, ph_b, int(k_break), 44)
    return mask


def make_kink_gen(N):
    def g(*p):
        p = list(p)
        args = p[:5] + [p[5]] + [N] + p[6:]
        return _generate_kink_event(*args)
    return g


GEN_K = {n: make_kink_gen(n) for n in (2, 3, 4)}

_xy, _tp = [(0, 95), (0, 95)], [(0, np.pi), (-np.pi, np.pi)]
_z, _kb = [(5, 35)], [(10, 40)]
BOUNDS_K = {
    2: _xy + _tp + _z + _kb + _tp * 4,
    3: _xy + _tp + _z + _kb + _tp * 6,
    4: _xy + _tp + _z + _kb + _tp * 8,
}


def make_obj_k(N):
    def f(p):
        p = list(p);
        p[0] = int(round(p[0]));
        p[1] = int(round(p[1]))
        p[4] = int(round(p[4]));
        p[5] = int(round(p[5]))
        try:
            m = GEN_K[N](*p) > 0
        except:
            return 1e6
        return wemd(m)
    return f


def x0_random_kink(N):
    base = [np.random.randint(96),
    np.random.randint(96),
            np.random.rand() * np.pi,
            np.random.uniform(-np.pi, np.pi),
            np.random.randint(5, 35),
            np.random.randint(10, 40)]
    sec = [np.random.rand() * np.pi,
           np.random.uniform(-np.pi, np.pi)] * 2 * N
    return np.array(base + sec)


def x0_maxE_kink(N, df):
```

```
    sl0 = df[df.layer == 0]
    idx = sl0.energy_release.idxmax()
    x0, y0 = df.loc[idx, ["index_along_x",
    "index_along_y"]]
    base = [int(x0), int(y0),
            np.pi / 4, 0,
            np.random.randint(5, 35),
            np.random.randint(10, 40)]
    sec = [np.random.rand() * np.pi,
           np.random.uniform(-np.pi, np.pi)] * 2 * N
    return np.array(base + sec)


def x0_hough_kink(N, df):
    sl = df[df.layer < 4][["index_along_x",
    "index_along_y"]].values
    x0, y0 = sl.mean(0)
    base = [x0, y0, np.pi / 4, 0,
            np.random.randint(5, 35),
            np.random.randint(10, 40)]
    sec = [np.random.rand() * np.pi,
           np.random.uniform(-np.pi, np.pi)] * 2 * N
    return np.array(base + sec)


strategies = {
    "random": lambda N: x0_random_kink(N),
    "maxE": lambda N: x0_maxE_kink(N, evt),
    "hough": lambda N: x0_hough_kink(N, evt)
}



N = 4
table = []
for name, sfn in strategies.items():
    x0 = sfn(N)
    t0 = time.time()
    de = differential_evolution(
        make_obj_k(N), BOUNDS_K[N],
        init=pop,
        popsize=20, maxiter=100, seed=42, disp=False)
    dt = time.time() - t0
    m = GEN_K[N](*de.x) > 0
    table.append([name, dt, iou(m), dice(m)])

def _generate_four_event(startx, starty, theta0, phi0, zint,
                         theta1, phi1, theta2, phi2,
                         theta3, phi3, theta4, phi4):
    return _generate_event(
        startx, starty, theta0, phi0, zint,
        4,
        [theta1, theta2, theta3, theta4],
        [phi1,  phi2,  phi3,  phi4]
    )
```

```
bounds_four = [
    (0, 95),
    (0, 95),
    (0, np.pi),
    (-np.pi, np.pi),

    (0, 44),

    (0, np.pi), (-np.pi, np.pi),
    (0, np.pi), (-np.pi, np.pi),
    (0, np.pi), (-np.pi, np.pi),
    (0, np.pi), (-np.pi, np.pi),
]


def _objective_four(params, target_mask):

    params = list(params)
    params[0] = int(round(params[0]))
    params[1] = int(round(params[1]))
    params[4] = int(round(params[4]))
    params = tuple(params)

    if len(params) != 13:
        return 1e6

    try:
        gen_mask = _generate_four_event(*params) > 0
        return wasserstein_distance(gen_mask, target_mask)
    except Exception as e:
        print(e)
        return 1e6
```

Список литературы

1. *Богомолов Э. А.* Антипротоны и дейтоны в галактических космических лучах: дис. доктора физико-математических наук // Физ.-техн. ин-т им. А. Ф. Иоффе РАН. — 2003.
2. PAMELA – A payload for antimatter matter exploration and light-nuclei astrophysics / P. Picozza [и др.] // Astroparticle Physics. — 2007. — Т. 27, № 4. — С. 296—315. — ISSN 0927-6505.
3. Antiproton Flux, Antiproton-to-Proton Flux Ratio, and Properties of Elementary Particle Fluxes in Primary Cosmic Rays Measured with the Alpha Magnetic Spectrometer on the International Space Station / M. Aguilar [и др.] // Phys. Rev. Lett. — 2016. — Т. 117, № 9. — С. 091103.
4. Geant4—a simulation toolkit / S. Agostinelli [и др.] // Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. — 2003. — Т. 506, № 3. — С. 250—303. — ISSN 0168-9002.
5. The electron–hadron separation performance of the PAMELA electromagnetic calorimeter / M. Boezio [и др.] // Astroparticle Physics. — 2006. — Т. 26, № 2. — С. 111—118. — ISSN 0927-6505.
6. Observation of antiprotons / O. Chamberlain [и др.] // Physical Review. — 1955. — Т. 100, № 3. — С. 947.

Рис. 8: ПАМЕЛА