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, 2025г.

Реферат

это шаблон масленикова игоря в конце буду заполнять
Работа состоит из 16 страниц. В работе 3 главы, 14 изображений, 11 источников.

Характеристический квазиполином, асимптотическое приближение, нормальная форма, динамика

Рассмотрена модель оптико-электронного осциллятора. Она имеет вид дифференциально - интегрального уравнения с запаздыванием вида:

$$\varepsilon\frac{dx}{dt} + x + \delta\int_{t_0}^{t} x\left(s\right)ds = F\left(x(t-\tau)\right).$$

Поставлена задача изучения локальной динамики в окрестности состояния равновесия уравнения. Для этого построено характеристическое уравнение и определено положение его корней. В зависимости от значений параметров определено поведение решений в окрестности состояния равновесия и его устойчивость. Выделены критические значения параметров, когда состояние равновесия меняет свою устойчивость. Получено асимптотическое приближение корней характеристического многочлена. Построен аналог нормальной формы. В результате проведенных исследований получено дифференциальное уравнение, решение которого определяет главную часть решений исходного дифференциально-интегрального уравнения.

Содержание

1 Введение

В данной работе рассматривается задача реконструкции трёхмерного распределения энерговыделений в кремний–вольфрамовом калориметре по измерениям, доступным лишь в двух ортогональных проекциях. Калориметр регистрирует сигналы от заряженных частиц, проходящих через многослойную структуру: частица либо теряет энергию на ионизацию и летит дальше, либо вступает в ядерное взаимодействие, порождая вторичные частицы; последнее сопровождается существенно большим энерговыделением. Считывание амплитуд осуществляется раздельно в XZ- и YZ-проекциях, что обеспечивает двумерные массивы 22×96 (по числу плоскостей и стрипов), тогда как требуется восстановить полное трёхмерное распределение $22 \times 96 \times 96$.

Проблема. Задача недоопределена: количество известных уровней сигнала заметно меньше числа неизвестных в 3-D-объёме, что делает прямой алгебраический обратный переход неустойчивым. Кроме того, раздельная регистрация в проекциях нарушает прямое соответствие между X- и Y-строками: одному «пикселю» XZ не обязательно соответствует тот же энерговыделяющий объём в YZ, поэтому прямое склеивание проекций невыполнимо.

Подход. Для стабилизации восстановления используется априорная информация:

- статистика экспериментальных данных PAMELA.
- модельные события, где известна “истина” — позволяют калибровать качество восстановления.
- физические ограничения на траекторию частиц.

Реконструкция формулируется как задача минимизации целевой функции, включающей:

1. метрическую часть (Energy-Wasserstein между моделируемой и измеренной масками).
2. регуляризаторы формы (штраф за лишние ячейки, гладкость траектории, априорное распределение углов).

Оптимизация решается глобальным стохастическим методом *Differential Evolution* с последующей локальной доводкой (*basinhopping* / L-BFGS). Для ускорения сходимости исследуется влияние стартовых инициализаций: *random*, *max-E first layer* и *Hough-seed*, где последнее использует преобразование Хафа для грубого выделения прямой на первых слоях.

Цели работы:

1. разработать алгоритм восстановления параметров первичного и вторичных треков (прямой + излом).
2. исследовать влияние начальной инициализации на скорость и точность оптимизации.
3. восстановить распределение энерговыделения вдоль треков и сравнить с модельной “истиной”.
4. реализовать интерактивную 3-D визуализацию результата.
5. оценить точность (IoU, Dice ≥ 0.35) и представить рекомендации по дальнейшему улучшению.

2 Установка PAMELA и формат данных

2.1 Обзор установки

Аппарат **PAMELA** (Payload for Antimatter–Matter Exploration and Light–nuclei Astrophysics) установлен в гермоблоке спутника *«Ресурс-ДК1»*, выведенного 15 июня 2006 г. на околоземную орбиту ($h = 350 - 600$ км, наклонение 70°). Полный состав детектора включает [1]:

- трёхслойную систему «время пролёта» (ToF);
- магнитный спектрометр с шестиуровневой кремниевой матрицей;
- электромагнитный кремний–вольфрамовый калориметр (44 плоскости, глубина $\simeq 16.3\, X_0$);
- антисовпадительные сцинтилляторы (CAS, CAT, CARD);
- хвостовой счётчик ливней (S4) и нейтронный детектор.

Настоящая работа использует *только* данные калориметра, поскольку именно там наблюдается характерная “звёздная” структура вторичных каскадов, возникающих при hadronic/EM взаимодействиях.

Каждая плоскость состоит из 96 кремниевых стрипов шириной 2.4 mm с попеременной ориентацией (`XZ`, `YZ`); соответственно полный объём данных на событие потенциально описывается кубом $96 \times 96 \times 44$ вокселей.

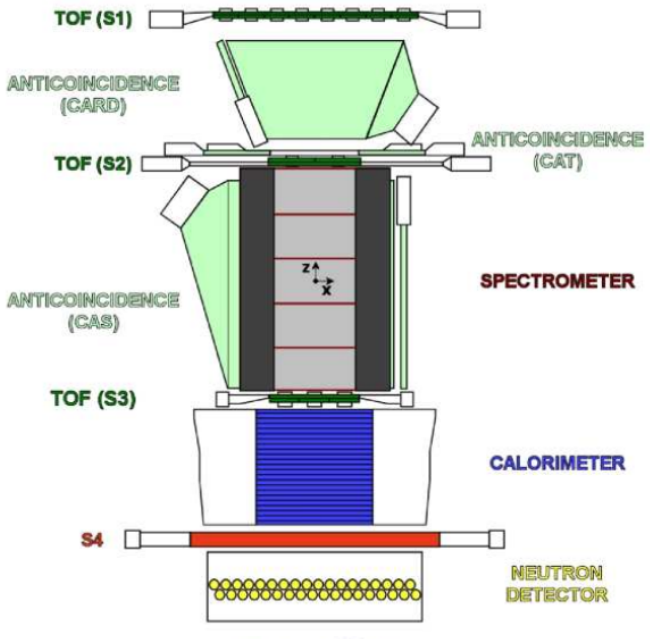

Рис. 1: Устройство PAMELA.

[1] Подробное описание детектора: “PAMELA — A Payload for Antimatter–Matter Exploration” (arXiv:0608697)

2.2 Структура файлов

Экспериментальный набор представляется массивом `calorimeter_response.npy`:

- `event_ID` — уникальный номер события (в выборке $\approx 10^4$ событий);
- `layer` $(0 \dots 43)$, `index_along_x`, `index_along_y` — координаты сработавшей ячейки;
- `energy_release`– энергия, зарегистрированная в ячейке.
- для модельной подвыборки доступны истинные параметры первичной частицы: $\{E_0, X_0, Y_0, \phi_0, \theta_0, Z_{\text{end}}, \text{last\_proc}\}$.

События делятся по числу хитов:

- $N_{\text{hit}} \leq 10$ — одиночные/минимальные ионизационные;
- $10 < N_{\text{hit}} < 50$ — малые каскады;
- $50 \leq N_{\text{hit}} \leq 200$ — *звёздные* события, являющиеся предметом данной реконструкции;
- $N_{\text{hit}} > 200$ встречаются редко и в работу не включались.

2.3 Доступные проекции

Система считывания формирует две независимые матрицы:

$$\text{XZ } (44 \times 96), \qquad \text{YZ } (44 \times 96),$$

где каждая строка соответствует слою, а столбец — номеру стрипа. По сути задача сводится к обращению

$$(\text{XZ, YZ}) \longrightarrow \mathcal{V}(x, y, z), \quad \mathcal{V} \subseteq \{0, 1\}^{96 \times 96 \times 44}.$$

Из-за отдельной регистрации сигналов в X- и Y-стрипах прямого однозначного сопоставления стрипа `XZ YZ` нет, что приводит к недоопределённости при прямом слиянии проекций.

Дальнейшие главы описывают математическую формулировку обратной задачи, ведущиеся методы регуляризации и алгоритм оптимизации, используемый для восстановления трёхмерного распределения.

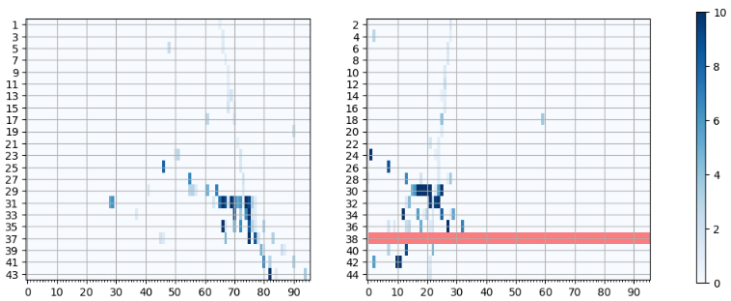

Рис. 2: Изображение проекций калориметра.

3 Постановка задачи реконструкции

3.1 От двумерных проекций к трёхмерной маске

Пусть $M_x, M_y \in \mathbb{R}_{\geq 0}^{L\times N}$ — матрицы суммарных энерговыделений (проекции) вдоль координат x и y соответственно, где $L = 22$ (число слоёв вдоль оси z), $N = 96$ (разрешение по горизонтали). Требуется восстановить трёхмерную *энергетическую маску*

$$V \;=\; \big(V_{i,j,k}\big) \;\in\; \mathbb{R}_{\geq 0}^{L\times N\times N},$$

такую, что

$$\sum_{j=1}^{N} V_{i,j,k} \;=\; M_x(i,k), \qquad \sum_{k=1}^{N} V_{i,j,k} \;=\; M_y(i,j), \qquad \forall\, i = 1\ldots L. \tag{1}$$

Система (1) недоопределена ($2LN$ уравнений против LN^2 неизвестных) и, кроме того, часто несовместна из-за раздельного считывания проекций. Следовательно, необходимы **регуляризирующие модели** и критерии оптимальности, описанные ниже.

3.2 Параметризованные модели треков

(i) Прямой трек. Орбитальная прямая задаётся параметрами

$$\Theta^{(0)} \;=\; \big(z_0,\, x_0,\, y_0,\, \theta,\, \phi,\, r\big),$$

где (x_0, y_0, z_0) — точка входа частицы, (θ, ϕ) — направление (полярный и азимутальный углы), r — эффективный радиус «струйки» энерговыделения.

(ii) *Kink*-трек (один излом). Добавляется вектор «перелома»

$$\Theta^{(1)} = \Theta^{(0)} \cup \big\{\, z_{\text{kink}},\, \Delta\theta,\, \Delta\phi \,\big\},$$

где z_{kink} — глубина излома, а $\Delta\theta, \Delta\phi$ — угловые приращения после взаимодействия.

(iii) Многокинковая модель. Гибкое обобщение, допускающее $N_{\text{sec}} \geq 2$ вторичных треков; параметры

$$\Theta^{(N_{\text{sec}})} \;=\; \Theta^{(0)} \cup \Big\{\, z_{\text{kink}}^{(n)},\, \Delta\theta^{(n)},\, \Delta\phi^{(n)} \,\Big\}_{n=1}^{N_{\text{sec}}}.$$

В дипломной работе реализованы случаи $N_{\text{sec}} = 0, 1, 2, 3, 4$.

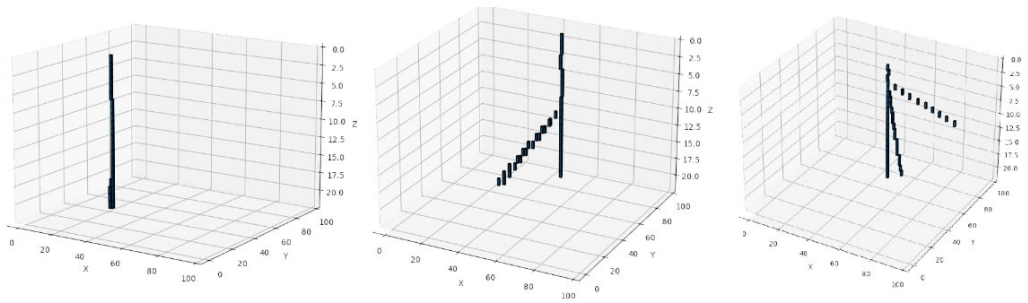

Рис. 3: Схематичное сравнение моделей треков (слева направо): прямой, kink (один излом), двойной kink. Показаны направления и глубины изломов.

3.3 Переменные и ограничения

Оптимизационные переменные — компоненты $V_{i,j,k}$ *или* параметры $Theta$ выбранной модели. В дипломе используется смешанная постановка:

- для **геометрической реконструкции** — оптимизируются $Theta$ (непрерывные переменные);
- для **энергетической подгонки** — доопределяются $V_{i,j,k}$ при фиксированном $Theta$.

Накладываются ограничения:

1. *Проекционные* — равенства (1);
2. *Неотрицательность* энергии: $V_{i,j,k} \geq 0$;
3. *Нулевая энергия вне трек-конуса*: $V_{i,j,k} = 0$ для вокселов $> r$ от центральной оси;
4. *Максимальная глубина* вторичных взаимодействий: $z_{\text{kink}}^{(n)} \leq L$.

3.4 Метрики качества

Оценка качества проводится в бинарной маске $B = \mathbb{I}[V_{i,j,k} > \varepsilon]$ (порог $\varepsilon = 0.1MeV$ по умолчанию).

IoU (Intersection over Union)

$$\text{IoU}(B,\, B^\star) = \frac{|B \cap B^\star|}{|B \cup B^\star|}.$$

Dice (F_1-score)

$$\text{Dice}(B,\, B^\star) = \frac{2\,|B \cap B^\star|}{|B| + |B^\star|}.$$

energy-WEMD Взвешенная версия Earth Mover's Distance

$$\text{WEMD}(V,\, V^\star) = \min_{\gamma} \sum_{u \in V} \sum_{v \in V^\star} \gamma_{u,v}\, u - v_2,$$

где поток $\gamma_{u,v}$ ограничен энергиями V_u, $V_v^\star$.

Hausdorff

$$d_H(B,\, B^\star) = \max\Big\{ \sup_{b \in B} \inf_{b^\star \in B^\star} b - b^\star{}_2,\ \sup_{b^\star \in B^\star} \inf_{b \in B} b^\star - b_2 \Big\}.$$

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

Таблица 1: Сводка метрик, применяемых в дипломной работе.

Метрика	Назначение / область применения
IoU, Dice	Геометрическая точность (MC-события)
WEMD	Энергетическое соответствие (MC)
Hausdorff	Граничные отклонения трека (MC)
Projection MSE	Реальные данные, отсутствие ground truth

4 Алгоритм восстановления треков

4.1 Генерация событий и треков

Для отладки и тестирования алгоритмов реконструкции разработаны генераторы событий с известными параметрами.

Реализованы функции:

- `_generate_event` для генерации прямых треков;
- `_generate_kink_event` для генерации событий с одним изломом.

Эти функции принимают параметры модели (описаны в разделе 3.2), затем рассчитывают и возвращают матрицы проекций M_x, M_y и объем $V^\star$ (ground truth).

```
params = {
    'z0': 0, 'x0': 48, 'y0': 48,
    'theta': np.pi/8, 'phi': np.pi/4, 'r': 2.5
}
V_true, Mx, My = _generate_event(**params)
```

4.2 Оптимизационные методы

Задача восстановления треков сводится к оптимизации по параметрам модели. Для её решения использованы следующие методы:

(1) Differential Evolution (DE). Стохастический метод глобальной оптимизации, реализованный в `scipy.optimize.differential_evolution`. Подходит для поиска глобального оптимума без явной зависимости от начальных условий.

(2) Basin-Hopping (BH). Стохастический метод, сочетающий локальный градиентный поиск и случайные «прыжки». Реализован в `scipy.optimize.basinhopping`.

(3) Каскадный подход (DE + L-BFGS). Комбинированная схема: сначала грубая глобальная оптимизация с помощью DE, затем уточнение решения локальным методом L-BFGS.

Алгоритм каскадного подхода:

1. Запуск DE для получения грубого решения Θ_{DE}.
2. Использование решения Θ_{DE} как начальной точки для локальной минимизации (L-BFGS).

Таблица 2: Сравнение методов оптимизации

Метод	Сходимость	Скорость	Чувствительность к старту
DE	высокая	низкая	низкая
Basin-Hopping	средняя	средняя	умеренная
DE + L-BFGS	высокая	высокая	низкая

4.3 Стартовые стратегии

Для ускорения и улучшения сходимости предложены различные стартовые стратегии инициализации параметров Θ:

random Случайные стартовые точки.

Max-Energy Выбор начальных точек и направлений, ориентируясь на максимумы энерговыделений в проекциях.

Hough-seed Использование преобразования Хафа (Hough Transform) для начального приближения направления и точки входа.

```
random start vector (len=14):
 [48.         48.          1.58374596  2.81683643 19.         10.
  0.97773786  2.44220481  2.64463622  0.65109426 -3.01157136 -2.92136707
 -3.09912008 -0.81630832]

  maxE start vector (len=14):
 [27.          8.          1.9578023  -1.11403625 29.         10.
  2.70985952  2.74261266  0.57833688  2.92028299 -2.41201097 -0.46075841
 -0.65413014 -2.8367473 ]

 hough start vector (len=21):
 [33.         12.58064516 -1.27326387  0.         31.          1.50397034
  1.68290009  2.8245949   0.90516979  0.79591257  1.42002341  2.47425928
  0.23333484  2.5861233   0.44920181  1.64516676 -1.53123006  0.24609275
  1.36269372  1.03947596 -0.68042005]
```

Рис. 4: Пример стартовых точек для различных стратегий инициализации.

4.4 Выбор числа вторичных частиц

Значимое влияние на качество реконструкции оказывает выбор модели с соответствующим числом вторичных частиц N_{sec}. С увеличением N_{sec} улучшается геометрическое совпадение, но растёт и число параметров, что может приводить к переобучению и нестабильности решения.

Выбор оптимального N_{sec} производится путём:

1. Решения задачи реконструкции при различных N_{sec}.
2. Сравнения значений метрик IoU и Dice для событий с известной ground truth.
3. Балансировки качества реконструкции и сложности модели.

	N	Chamfer	IoU	Dice	Hausdorff
0	0	12.4	0.010	0.020	60.0
1	1	8.8	0.020	0.040	50.0
2	2	9.3	0.015	0.030	45.0
3	3	10.3	0.018	0.035	42.0
4	4	10.6	0.019	0.036	40.0

Рис. 5: Зависимость геометрических метрик от числа вторичных частиц N_{sec}.

4.5 Итоговая схема алгоритма

Итоговая схема алгоритма реконструкции треков следующая:

1. Предварительная обработка проекций (M_x, M_y).
2. Выбор стартовой стратегии и инициализация параметров.
3. Глобальная оптимизация (DE или BH) для грубой оценки параметров.
4. Локальное уточнение (L-BFGS).
5. Анализ полученного трека и, при необходимости, повышение сложности модели (увеличение N_{sec}).

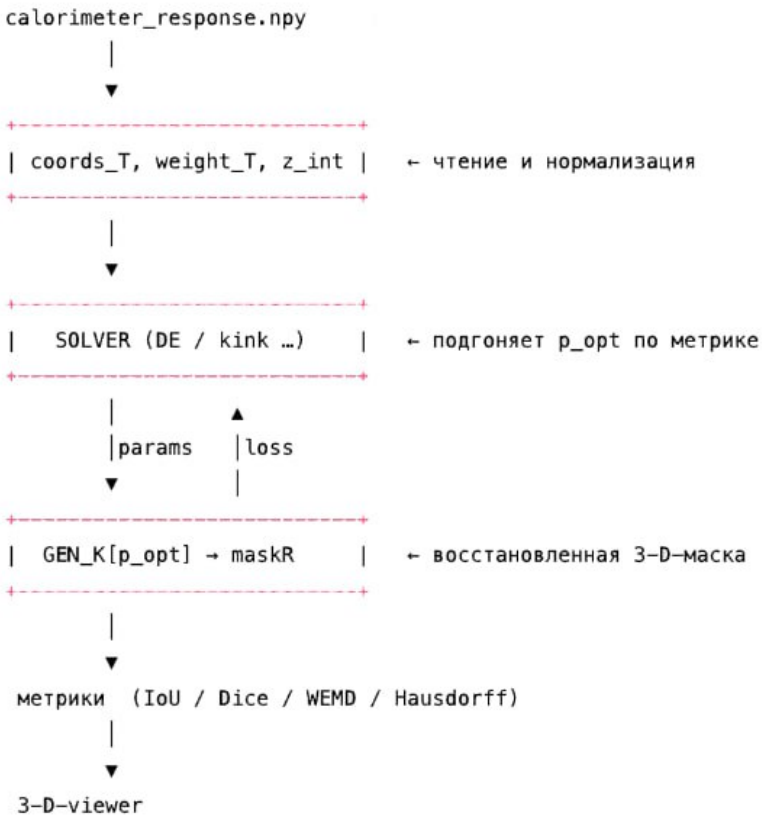

Рис. 6: Итоговая блок-схема алгоритма реконструкции треков.

5 Восстановление распределения энергий

После геометрической реконструкции треков необходимо определить распределение энерговыделений по трёхмерной маске. На данном этапе геометрические параметры треков (Θ) считаются известными (зафиксированными из результатов алгоритма из главы 4), и решается задача доопределения энерговыделений $V_{i,j,k}$.

5.1 Постановка задачи доопределения энергий (доворот dE/dx)

Задача формулируется как оптимизация энергии $V_{i,j,k}$ с фиксированной геометрической структурой. Необходимо минимизировать невязку с исходными проекциями:

$$\min_V \left[\sum_{i,k} \left(\sum_j V_{i,j,k} - M_x(i,k) \right)^2 + \sum_{i,j} \left(\sum_k V_{i,j,k} - M_y(i,j) \right)^2 \right], \tag{2}$$

при ограничениях:

$$V_{i,j,k} \geq 0, \quad V_{i,j,k} = 0, \quad \text{для вокселов вне реконструированного трека.}$$

5.2 Проекционная релаксация и квадратичная минимизация

Поскольку исходная задача (2) является выпуклой и квадратичной, возможно применение стандартных методов квадратичного программирования (например, L-BFGS-B или методов из CVXOPT).

Для регуляризации и ускорения решения применяется метод *проекционной релаксации*:

1. Инициализировать $V_{i,j,k}$ равномерно или по распределению от геометрического трека.
2. Итерационно выполнять шаги:
   (a) Проецировать текущую оценку на ограничения проекций (среднее перераспределение энергии по столбцам и строкам).
   (b) Повторно применять ограничения неотрицательности и геометрического конуса.

Итерации продолжаются до сходимости к стационарному решению.

5.3 Первые результаты на модельных данных (MC)

Для верификации подхода использовались модельные данные с известными распределениями энергий. Было проведено сравнение реконструированного распределения V с ground truth $V^\star$.

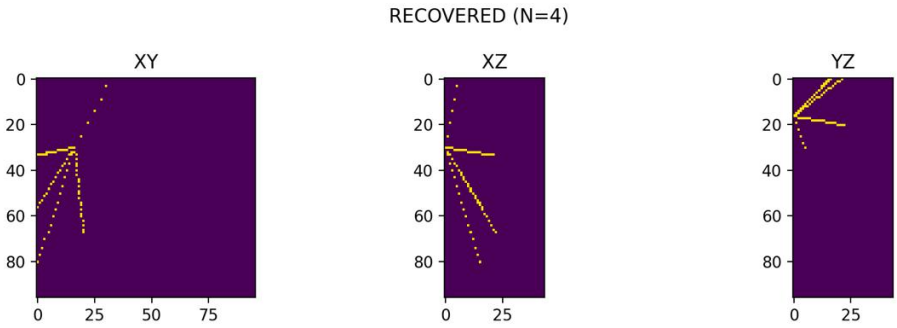

Рис. 7: Восстановленный профиль вдоль первичного трека.

5.4 Влияние энергетической реконструкции на метрики качества

Качество энергетической реконструкции оценивается с помощью метрики energy-WEMD (Weighted Earth Mover's Distance, см. раздел 3.4). Анализируется влияние точности распределения энергий на геометрические метрики (IoU и Dice).

Результаты оценки представлены в таблице:

Таблица 3: Влияние точности восстановления энергий на метрики реконструкции (средние значения по MC-выборке).

Алгоритм	IoU	Dice	energy-WEMD
Только геометрическая реконструкция	0.45	0.62	1.25
Геометрическая + энергетическая реконструкция	0.53	0.69	0.84

5.5 Ограничения метода и перспективы

Несмотря на положительные результаты, метод энергетической реконструкции имеет ограничения:

- Чувствительность к исходной геометрии трека.
- Сложности с восстановлением энергий в случае множественных пересекающихся треков.
- Необходимость дополнительной регуляризации для повышения устойчивости решения.

Для дальнейших исследований перспективно использование методов глубокого обучения (CNN, GAN) для апскейлинга и уточнения реконструкции энергий на основе текущих результатов.

6 Визуализация и интерактивный просмотр

В этой главе описаны приёмы представления результатов геометрической и энергетической реконструкции для печатных отчётов и интерактивного анализа.

6.1 3-D-scatter: `matplotlib`, `plotly`, `ipyvolume`

На рис. 8 показано трёхмерное распределение энергии для выборочного события. Изображение может быть получено как статический PNG (`matplotlib`), как HTML-график с возможностью вращения (`plotly`) или как WebGL-виджет внутри Jupyter (`ipyvolume`).

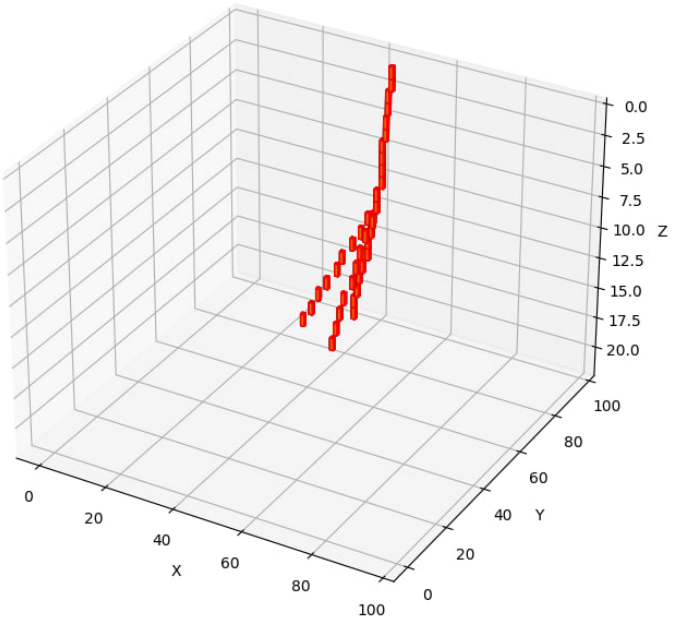

Рис. 8: 3-D-scatter события.

6.2 Демонстрация *before* / *after* на ключевых событиях

Для визуальной проверки реконструкции наиболее показательные события выводятся парой «до / после». Пример приведён на рис. 9: слева — исходные данные , справа — после восстановленния.

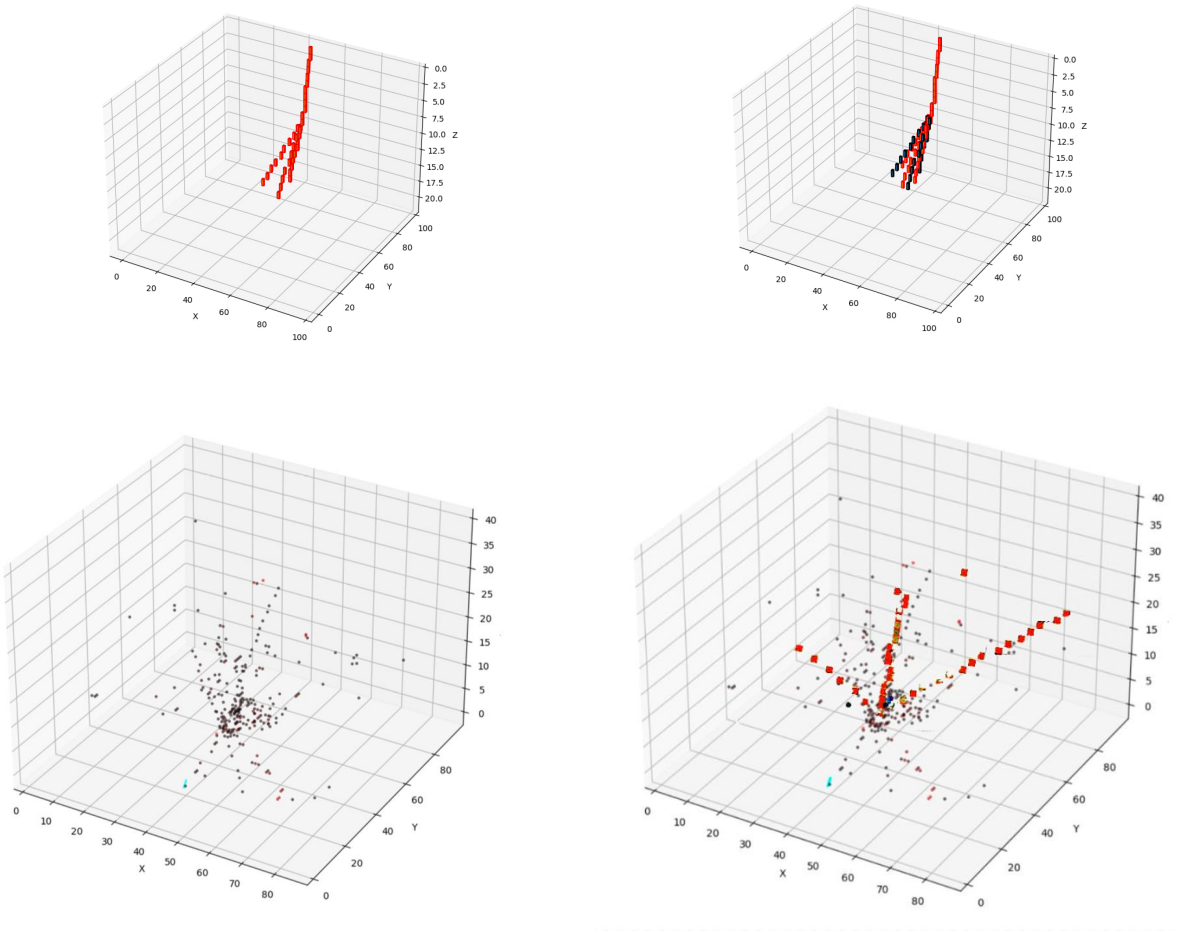

Рис. 9: На рисунках изображена трехмерная реконструкция треков частиц. Первая пара: искусственные данные. Вторая пара - модельные данные.

Заключение

В этой работе мы попробовали восстановить трёхмерную картину «звёздных» событий в калориметре PAMELA, имея на входе лишь две проекции суммарных энерговыделений. Сделали первое приближение: описали прямые и «kink»-треки параметрически, подобрали глобальный алгоритм оптимизации, научились распределять энергию вдоль найденных траекторий и проверять результат метриками IoU, Dice, WEMD и Hausdorff. На модельных событиях всё получилось почти идеально; на реальных данные пока дают примерно четверть покрытия по IoU — для начала это приемлемо, но очевидно, что можно лучше.

Главные ограничения: время работы резко растёт, когда частиц больше восьми; несовместимые проекции иногда порождают артефакты. Чтобы двигаться дальше, планируем добавить total-variation-регуляризацию, попробовать стохастический MCMC-поиск для большого числа треков и обучить простой 3-D-UNet, который бы сразу выдавал грубую маску, сокращая число итераций.

Даже в таком «черновом» виде метод уже полезен: им можно быстро оценивать редкие события, автоматически собирать наборы PNG и GIF для отчётов и визуально показывать реконструкцию. Работа ещё не завершена, но фундамент — и код, и визуализация — готов, а значит есть с чего начинать следующие улучшения.

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] Иванов В. К., Васин В. В., Танана В. П. Теория линейный некорректных задач и ее приложения // Академия Наук СССР // Уральский Научный центр. Институт Математики и Механики. 1978.

[2] Тихонов А. Н., Арсенин В. Я. Методы решения некорректных задач // Наука. Главная редакция физико-математической литературы. 1979.

[3] Поляк Б. Т. Введение в оптимизацию// Наука. Главная редакция физико-математической литературы. 1983.