Санкт-Петербургский государственный университет

Кафедра, системного программирования

Группа 24.М41-мм

Разработка прототипа ядра для проекта PySATL. Модуль families

Михайлов Михаил Дмитриевич

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
доцент кафедры системного программирования, к. ф.-м. н., Гориховский В. И.

Санкт-Петербург
2025

Оглавление

Введение

В рамках проекта PySATL ведётся разработка вычислительного ядра, предназначенного для представления и обработки вероятностных распределений и их семейств. Планируется, что ядро будет предоставлять средства для задания распределений, выполнения операций над ними и построения сложных структур путём функциональных и алгебраических преобразований. Система должна поддерживать как конкретные распределения с заданными параметрами, так и абстрактные семейства, а также обеспечивать возможность расширения за счёт пользовательских распределений и преобразований.

Необходимость создания собственного ядра обусловлена ограничениями существующих инструментов при решении задач статистики и стохастического моделирования в контексте экосистемы PySATL. В предыдущем семестре был проведён обзор доступных решений [4], выполнены сравнительный анализ функциональности и оценка архитектурных подходов, а также сформулированы некоторые функциональные требования к ядру. По результатам анализа было принято решение о разработке прототипа, поскольку ни одна из существующих библиотек не удовлетворяет всем требованиям проекта.

Данная работа посвящена проектированию и реализации модуля `families` в составе прототипа ядра. В отчёте описывается архитектура модуля, механизм поддержки нескольких параметризаций одного семейства, а также анализируется его взаимодействие с модулем `distributions`, разработанным Ёлкиным Л. Целью работы является создание прототипа, покрывающего функциональность SciPy в части работы с параметрическими семействами, и проверка соответствия предложенной архитектуры поставленным требованиям.

1. Постановка задачи

Общей целью является разработка библиотеки `pysatl-core`, которая сможет заменить SciPy в качестве вычислительного ядра.

Целью настоящей работы является разработка прототипа ядра, в частности модуля параметрических семейств и анализ прототипа на соответствие требованиям.

Для её выполнения были поставлены следующие задачи:

- Формализовать требования к ядру. Выделить из них те, которые покрывают функциональность SciPy
- Реализовать модуль families покрывающий функциональность SciPy
    1. Доработать архитектуру, предложенную Ёлкиным Л.
    2. Реализовать механизм поддержки нескольких параметризаций у семейств
    3. Сформулировать вывод о применимости модуля `distributions` в качестве модуля для работы с распределениями
- Проанализировать прототип на соответствие требованиям и предложить дальнейшее направление разработки

2. Обзор

2.1. Обзор существующих решений

В работе за предыдущий семестр [4], был сформулирован список требований к ядру по математической составляющей, которая нужна для проекта PySATL. Анализ существующих инструментов, проведённый в той же работе, показал, что универсального инструмента, полностью покрывающего все задачи работы с вероятностными распределениями, нужными в рамках PySATL, на данный момент не существует.

В работе рассматривалась возможность расширить уже существующую библиотеку, чтобы использовать её в качестве ядра. Рассматривались библиотеки, удовлетворяющие следующим требованиям: наличие базовой функциональности, активная поддержка, совместимая лицензия и реализация на языке, подходящем для научных вычислений (Python, R, Julia, C++, Rust). В результате были выделены и подробно сравнены семь инструментов:

- SciPy (Python)
- Экосистема языка R
- Distributions.jl (Julia)
- Boost.Math (C++)
- TensorFlow Probability (Python/C++)
- PyTorch Distributions (Python/C++)
- statrs (Rust).

С точки зрения математической функциональности библиотеки SciPy, R и Distributions.jl оказались лидерами по поддержке функциональных и числовых характеристик распределений. Однако при анализе архитектурных подходов и расширяемости картина меняется. Библиотеки на Python (SciPy, PyTorch, TensorFlow) и C++ (Boost)

обладают сложной иерархией классов, в результате чего добавление новых числовых или функциональных характеристик требует модификации исходного кода библиотек. Экосистема R, обладая огромным арсеналом пакетов, страдает от недостатков процедурного стиля и отсутствия единообразия, что затрудняет её интеграцию в качестве единого ядра.

Наиболее перспективной с архитектурной точки зрения была признана библиотека Distributions.jl для языка Julia, построенная на основе множественной диспетчеризации. Данный подход обеспечивает высокую гибкость и простоту расширения: добавление нового семейства распределений, характеристики распределения или алгоритма вычислений не требует модификации существующего кода библиотеки, а сводится к определению новых методов или типов.

По результатам анализа было принято решение о разработке прототипа ядра на основе библиотеки Distributions.jl. Однако после обсуждения с коллективом разработчиков PySATL было решено, что первая итерация прототипа должна быть на Python — это необходимо, чтобы зафиксировать предоставляемые ядром интерфейсы. Тем не менее несколько идей из Distributions.jl послужили основой для архитектурных решений, принятых при разработке центрального модуля прототипа — модуля `distributions` (см. следующий раздел).

2.2. Состояние прототипа ядра проекта PySATL

В настоящем разделе описаны детали, касающиеся прототипа ядра проекта PySATL, которые не относятся к основной части работы.

Совместно с Ёлкиным Л. Н. была предложена композиционная структура, изображенная на рис 1. Модуль `distribution` является основой для вычислений над распределениями, в то время как модуль `families` должен предоставлять функциональность работы с параметрическими семействами распределений. Под параметрическим семейством в данном случае понимается любое отображение из некоторого множества Θ, понимаемого как множество возможных значений пара-

метров, в пространство всех распределений. Модуль `transformations` должен предоставлять возможность производить функциональные преобразования распределений (например, вычислять свертки или производить замену переменных).

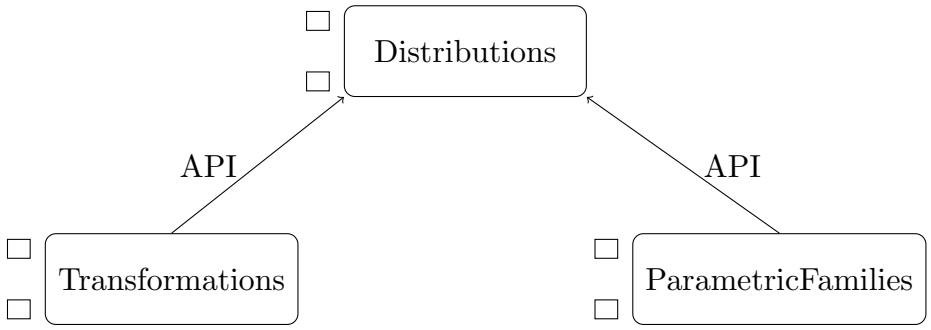

Рис. 1: Диаграмма модулей для прототипа ядра

Далее Ёлкиным Л. Н. была предложена и реализована архитектура для модуля `distribution`, изображенная на 2.

Основой модуля является протокол (интерфейс) `Distribution`. Протокол требует, чтобы реализации обладали следующими свойствами: тип распределения, стратегии генерации выборки и стратегии вычислений. В рамках такой архитектуры предполагается, что объекты-стратегии будут применимы к любой реализации протокола `Distribution`.

Абстракцией над конкретными вычислениями (например, вычисление CDF через интегрирование PDF) является интерфейс `Computation`. Его реализация `AnalyticalComputation` абстрагирует способ вычисления некоторой характеристики, для которой доступно аналитическое выражение.

Также на основе архитектуры `distributions` Ёлкин Л. Н. предложил первичное приближение дизайна для модуля `families`, изображённое на рис. 3. Каждое семейство в предложенном дизайне является объектом класса `ParametricFamilyDistribution`. Все семейства привязаны к глобальному реестру семейств — это необходимо, в частности, для модуля `transformations`. У каждого семейства может быть одна или несколько параметризаций; механизм работы с несколькими параметризациями был оставлен на уточнение автору данного отчёта.

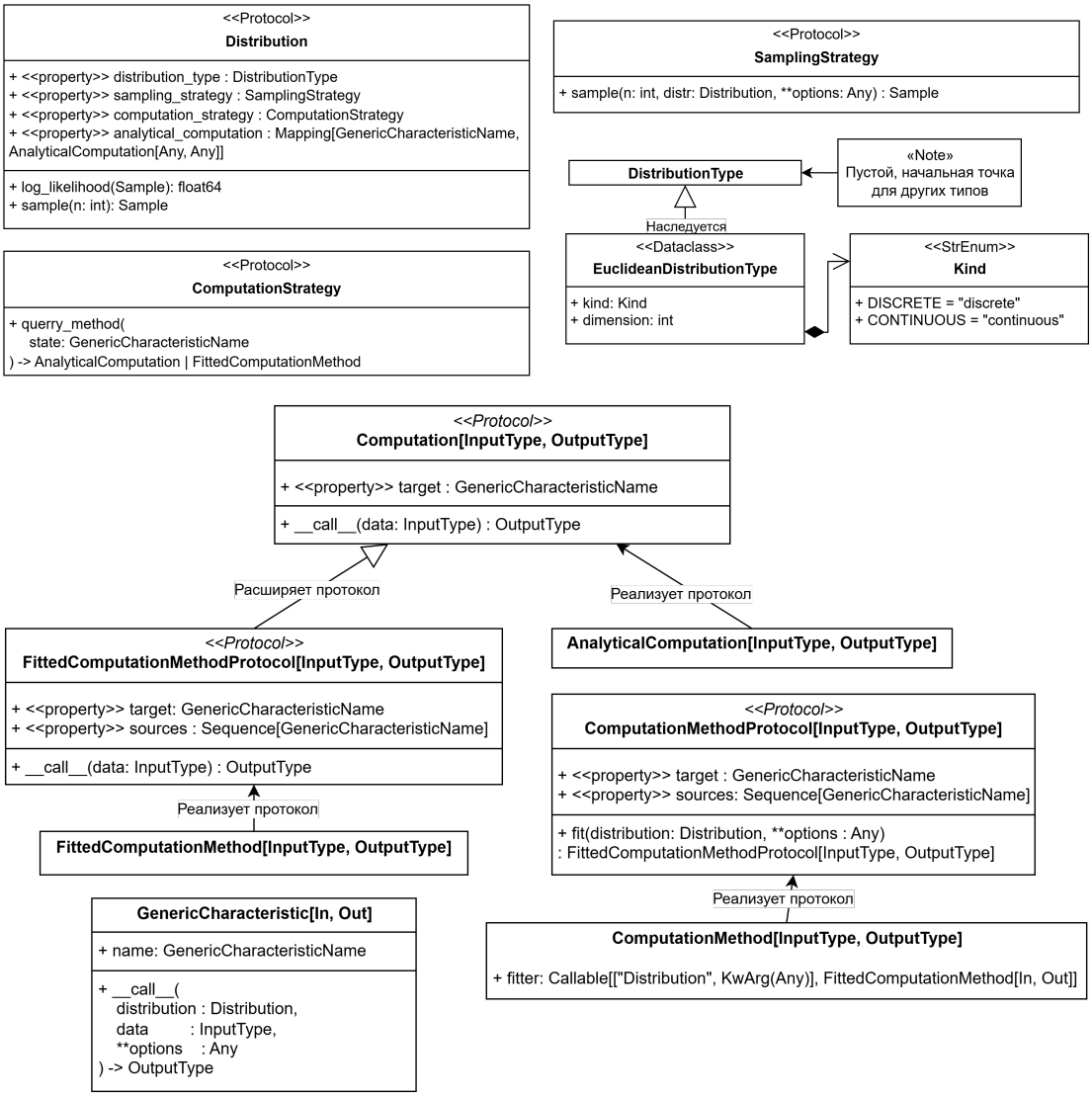

Рис. 2: Диаграмма классов модуля `distribution`

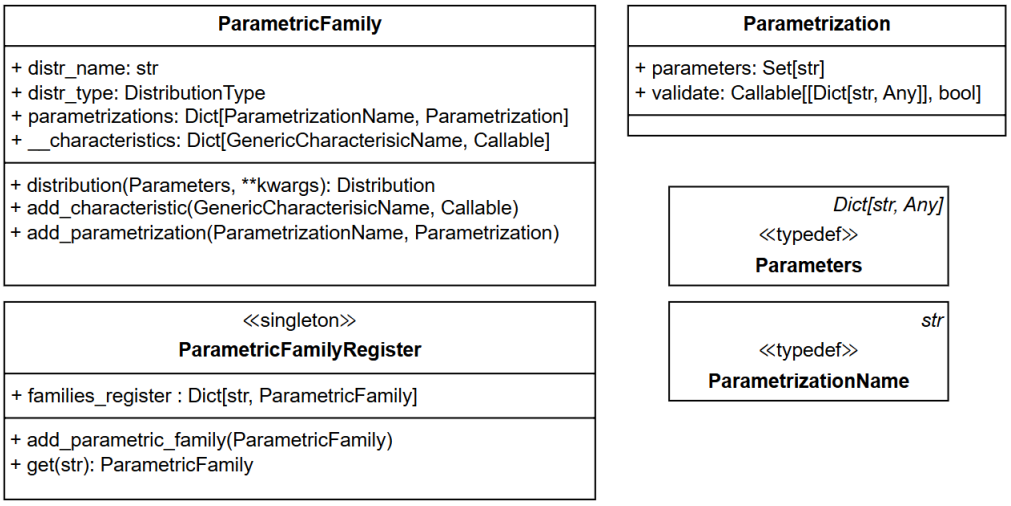

Рис. 3: Первичный дизайн модуля `families`

2.3. Выводы

В результате работы в рамках прошлого семестра [4] имеется некоторый набор требований к математической составляющей ядра, которым не удовлетворяет ни одна из популярных библиотек для работы с вероятностными распределениями. В этой же работе была показана нецелесообразность расширения существующих популярных библиотек. Было принято решение о разработке собственного ядра на Python.

При этом, помимо требований к математической функциональности, есть также набор требований, не рассмотренных в [4] (детально или вообще), касающихся инженерных аспектов: например, возможность выбирать численный метод для вычисления характеристики, не заданной аналитически. Требований достаточно много, и нельзя сказать, являются ли они непротиворечивыми. Одна из задач прототипа — ответить на этот вопрос.

Совместно с Ёлкиным Л. Н. была предложена композиционная структура прототипа ядра, включающая в себя три модуля: `distribution` (вычисления характеристик распределений), `families` (работа с семействами) и `transformations` (операции над распределениями). Также Ёлкиным Л. Н. был предложен начальный дизайн модуля `families` для работы с параметрическими семействами распределений.

Задачи, поставленные в настоящей работе, являются продолжением работы предыдущего семестра и направлены на получение первого варианта архитектуры ядра и проверку её непротиворечивости.

3. Архитектурная документация

Для формальной работы с требованиями, была создана архитектурная документация. Текущая итерация доступна по ссылке [3]

Были выделены следующие заинтересованные лица:

- Разработчики ядра
- Разработчики других библиотек PySATL
- Независимые от PySATL инженеры-исследователи
- Руководители проекта PySATL

В то время как разработчики ядра и руководители проекта PySATL не нуждаются в выделенных в [3] функциональных требованиях напрямую, основная группа — разработчики других библиотек PySATL, имеют достаточно обширный список требований (полный список требований доступен в разделе 3 архитектурной документации).

В результате были выделены следующие группы требований следующие группы требований

(i) Требования по управлению потоком вычислений и предоставляемым интерфейсам

(ii) Требования по наличию базовых распределений и характеристик распределений

(iii) Требования к функциональности параметрических семейств

(iv) Требования к функциональности семплирования

(v) Требования к функциональности операций над распределениями

Необходимо отметить, что требования к функциональности включают себя и нефункциональные требования (корректность, полнота, конфигурируемость). С точки зрения проекта PySATL в первую очередь в ядре должны быть реализованы требования по управлению потоком

вычислений и предоставляемым интерфейсам и требования к функциональности параметрических семейств, так как это множество требований, покрывающих возможности SciPy (исключая полноту и оптимизации), используемых внутри проекта. Отдельно отметим что было принято решение не реализовывать оценку максимального правдоподобия в рамках модуля семейств и вынести ее за рамки ядра в целом, так как такого рода оценки относятся уже непосредственно к статистике, и должны находится в другом контексте, отличном от ядра.

Можно отметить требования, которые относятся к п.(i), (iii), но которые выходят за рамки SciPy:

- Возможность динамически расширять семейства распределений, добавляя новые функциональные/числовые характеристики, заданные аналитически
- Возможность выбирать метод расчета функциональной или числовой характеристики, не заданной аналитически
- Возможность работать с несколькими параметризациями внутри одного параметрического семейства
- Возможность использовать аппроксимации к некоторым аналитически заданным характеристикам, вместо «честного» вычисления

4. Реализация прототипа модуля `families`

4.1. Предложенная архитектура и реализация

Модуль `families` реализован как слабо связанный слой над модулем `distributions`. На рис. 4 представленна диаграмма классов модуля `families`.

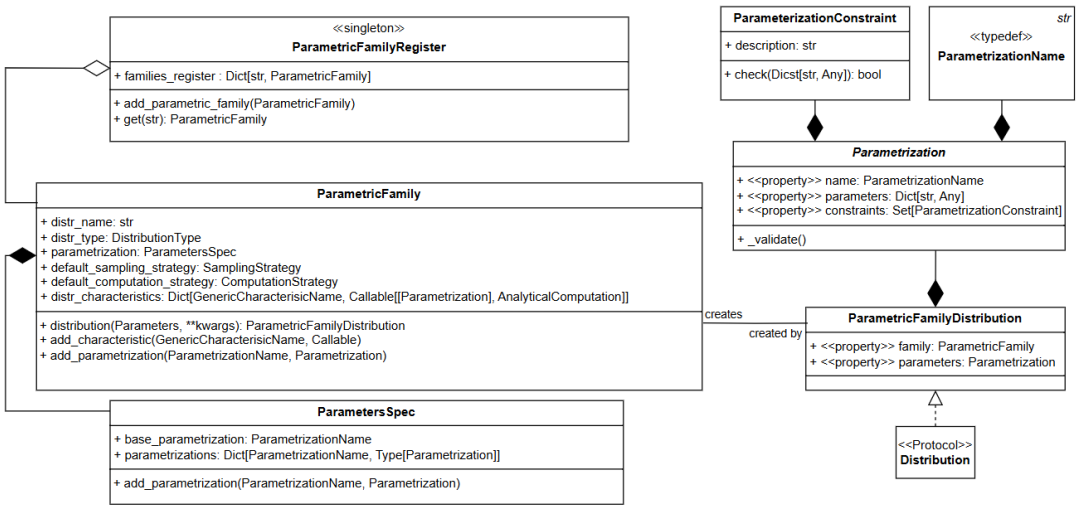

Рис. 4: UML-диаграмма классов families

По сравнению с начальным дизайном были внесены следующие изменения:

- Каждая конкретная параметризация теперь представляется как наследник абстрактного класса `Parametrization`. Для хранения ограничений на параметры предоставлен специальный класс `ParametrizationConstrain`;
- Вместо того чтобы хранить параметризации одного семейства внутри объекта-семейства, за параметризации отвечает теперь специальный объект класса `ParametrizationSpec`, который хранит в себе список классов-параметризаций, относящихся к данному семейству.

При реализации был выявлен следующий недостаток протокола `Distribution`. Чтобы определить функцию правдоподобия, требуется

заранее знать тип распределения (см. соответствующую issue [2]), который вообще говоря для семейства становится известен только во время исполнения программы (так как семейство это объект). На данный момент `ParametricFamilyDistribution`, не вычисляет логарифм правдоподобия (стоит заглушка). Обсуждение см. в заключительном подразделе настоящего раздела.

4.2. Поддержка нескольких параметризаций

Одним из требований к параметрическим семействам распределений является поддержка множественных способов параметризации одного и того же семейства распределений. Например, логнормальное распределение может быть задано как через параметры μ и σ, так и через математическое ожидание и дисперсию самого логнормального распределения.

Были выявлены следующие потенциальные проблемы:

- Параметризации могут быть связаны аналитически невычислимыми преобразованиями
- Для различных параметризаций могут быть доступны разные аналитические характеристики распределения
- Необходимость обеспечения возможно перехода между параметризациями

Рассматривались два подхода к решению данной проблемы:

1. **Разрешение вычислений `AnalyticalComputation` в любых параметризациях**. Данный подход предоставляет максимальную гибкость, но требует механизма разрешения конфликтов, при наличии нескольких параметризаций, вычисляющих одну и ту же характеристику.
2. **Централизованная базовая параметризация**. Требование вычисления всех `AnalyticalComputation` в одной центральной пара-

метризации с предоставлением методов преобразования из альтернативной параметризации в базовую.

Был выбран второй подход, основанный на концепции *базовой параметризации*. Данное решение проще в реализации, с точки зрения логики вычислений. При этом теряется возможность перехода между параметризациями средствами ядра — однако, пользователь может сам добавить такую возможность, если это необходимо, так как параметризации наследуются от абстрактного класса.

В текущей реализации базовая параметризация определяется как первая в списке параметризаций, передаваемом при создании семейства. Для небазовых параметризаций требуется реализация метода `transform_to_base_parametrization()`, обеспечивающего преобразование к базовому представлению. Реализация поддержки множественных параметризаций основана на системе декораторов, обеспечивающих декларативный синтаксис описания параметризаций, см. листинг 1.

4.3. Анализ применимости модуля `distributions`

Экземпляры семейств делегируют вычисления и семплирование стратегиям `distributions`. На уровне `families` определяются только декларации аналитически доступных характеристик, стратегии вычисления и семплирования по умолчанию и преобразования параметров между параметризациями перед вызовом низкоуровневых рутин.

Как уже упоминалось ранее, предлагается убрать функцию вычисления лог-правдоподобия выборки `loglikelihood` из протокола `Distribution`: решение с классом `GenericCharacteristic` позволяет вычислять правдоподобие в том же ключе, что и характеристики распределения; необходимости отдельно реализовывать `loglikehood` внутри `Distribution` нет. [1]

Также текущее решение никак не связывает между собой свойство `computation_strategy` и метод `sample`, поскольку `Distribution` явля-

[1] https://github.com/PySATL/pysatl-core/issues/17

Листинг 1: Демонстрация реализованных декораторов для создания параметризаций

```
# Базовая параметризация (canonical)
@parametrization(family=Lognormal, name='canonical')
class NormalParametrization:
    mu: float
    sigma: float

    @constraint(description='sigma > 0')
    def check_sigma_positive(self) -> bool:
        return self.sigma > 0

# Альтернативная параметризация (meanvar)
@parametrization(family=Lognormal, name='meanvar')
class MeanVarParametrization:
    mean: float
    var: float

    @constraint(description='mean > 0')
    def check_mean_positive(self) -> bool:
        return self.mean > 0

    @constraint(description='var > 0')
    def check_var_positive(self) -> bool:
        return self.var > 0

    def transform_to_base_parametrization(self):
        # Преобразование из (mean, var) в (mu, sigma)
        mu = ...
        sigma = ...
        return NormalParametrization(mu=mu, sigma=sigma)
```

ется интерфейсом. Это означает, что потенциально изменение стратегии семплирования не приведет к смене поведения метода `sample`

Можно заключить `distribution` полностью абстрагирует вычисления, что и было его изначальной целью, и архитектуру модуля `distributions`, возможно с небольшими изменениями, можно считать подходящей для ядра.

5. Анализ прототипа на соответствие требованиям

Как уже упоминалось, в архитектурной документации выделены 6 групп функциональных требований. В настоящем разделе нас интересует требования из групп (i), (iii): управление потоком вычислений и предоставляемым интерфейсам, требования к функциональности параметрических семейств.

Внутри этих двух групп, требования, покрывающие функциональность SciPy, выполнены (за исключением требований по наполнению и оптимизации). С другой стороны, ряд требований расширяющих SciPy на данный момент не могут быть выполнены.

В частности, на данный момент нет механизма частичной фиксации параметров семейства. Также, недоступно использование аппроксимаций и соотношений между распределениями при вычислении характеристик. Поясним что это означает.

Рассмотрим следующий сценарий использования ядра: при вычислении плотности Student(ν) при $\nu >> 1$ имеет место аппроксимация нормальным распределением. Пользователь хочет использовать её для вычислений.

На данный момент такое поведение невозможно реализовать с помощью средств ядра, не заводя отдельное семейство, в котором вместо точного вычисления плотности используется аппроксимация. Потенциальное решение может выглядеть так: изменить поведение свойства `analytical_computations()` у класса `ParametricFamilyDistribution`, так, чтобы при например $\nu > 100$ (или любом другом значении заданным пользователем) плотность вычислялась с использованием аппроксимации.

Для такого решения нужно завести некоторое хранилище подобных аппроксимаций (как набор разрешенных правил вычислений) и переопределить поведение свойства, так, чтобы применялись допустимые правила аппроксимации. При этом, так как такие аппроксимации находятся вне зоны стратегии вычислений, возникают сразу потенциаль-

ные проблемы, связанные с тем, как поступать в случае если доступно несколько аппроксимаций/способов вычисления. Несколько предложений выдвинуты в ;дискуссии [1].

Таким образом, архитектура покрывает возможности SciPy, однако было выделено два требования, которые требуют расширения архитектуры.

Заключение

В ходе работы над учебной практикой были выполнены поставленные задачи и получены следующие результаты:

1. Составлена архитектурная документация к ядру `pysatl-core` [3]. Выделена группа требований, покрывающих функциональность библиотеки SciPy, а также требования, расширяющие её возможности.

2. Разработан прототип модуля `families`, обеспечивающий базовую функциональность работы с параметрическими семействами распределений. Архитектура модуля доработана по сравнению с первоначальным предложением: введены классы `Parametrization` и `ParametrizationSpec`, реализована поддержка нескольких параметризаций на основе концепции *базовой параметризации*.

3. Проведён анализ применимости модуля `distributions`, реализованного Ёлкиным Л. Установлено, что модуль пригоден для использования в ядре. Сформулировано предложение по переносу метода `loglikelihood` из протокола `Distribution` в систему общих характеристик.

4. Выполнен анализ прототипа на соответствие требованиям. Показано, что базовая функциональность SciPy покрыта, однако выявлены два ключевых требования, требующих дальнейшего развития архитектуры: поддержка частичной фиксации параметров и механизма аппроксимаций характеристик.

Таким образом, прототип модуля `families` успешно реализует заявленные цели и служит основой для дальнейшей разработки ядра PySATL.

Реализация предложенной архитектуры доступна по ссылке: https://github.com/PySATL/pysatl-core/pull/7 (дата обращения 18 сентября 2025 г.)

Список литературы

[1] Discussion GitHub. More flexible handling of multiple parametrizations. — URL: https://github.com/PySATL/pysatl-core/discussions/15 (дата обращения: 18 сентября 2025 г.).

[2] Issue GitHub. API: Remove loglikehood method from Distribution protocol. — URL: https://github.com/PySATL/pysatl-core/issues/17 (дата обращения: 18 сентября 2025 г.).

[3] М. Д. Михайлов Л.Н. Ёлкин. Архитектурная документация PySATL core. — URL: https://github.com/PySATL/pysatl-core-design-document (дата обращения: 18 сентября 2025 г.).

[4] Михайлов М. Д. Обзор и сравнение инструментов для работы с вероятностными распределениями. — URL: https://github.com/spbu-se/mag_practices_2024-2026/blob/main/МО/МихайловМихаилДмитриевич/semester_1/Mikhailov-report.pdf (дата обращения: 18 сентября 2025 г.).