МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**Национальный исследовательский ядерный университет «МИФИ»**

Изображение выглядит как коллекция картинок

Автоматически созданное описание**Институт**

**интеллектуальных кибернетических систем**

**Кафедра кибернетики**

Направление подготовки 01.03.02 Прикладная математика и информатика

Расширенное содержание пояснительной записки

к учебно-исследовательской работе студента на тему:

Создание программной системы для прогноза физических характеристик изотопов в области температур термоядерной плазмы.

|  |  |  |
| --- | --- | --- |
| Группа | Б18-501 | Ломтев П.А. |
| Студент |  |  |
|  | (подпись) | (ФИО) |
| Руководитель |  | Глебов В.Б. |
|  | (0-5 баллов) (подпись) | (ФИО) |

Москва 2021

**Реферат**

Пояснительная записка содержит: 57 страниц. Количество использованных источников – 17, таблиц - 6, рисунков - 31.

Ключевые слова: сечение поглощения нейтронов, дейтерий, тритий, машинное обучение, экстраполяция, базы данных, Python, Django, PostgreSQL, HTML

В работе рассмотрена программная система для предсказания нейтронно-физических характеристик изотопов (He-3, H-2, H-3) в области температур термоядерной плазмы. Исходными данными для функционирования разрабатываемого ПС «Плазма» являются результаты расчетного моделирования при температурах от К до К. Расчетное моделирование взаимодействия нейтронов осуществляется методом Монте Карло и реализуется с помощью программного комплекса SERPENT 2.

Основой целью ПС «Плазма» является расширение возможностей программного комплекса SERPENT 2 по подготовке нейтронно-физических данных для плазменного состояние вещества (с температурами большими программного предела - ).

Внедрение ПС позволит:

1. Оценить значения физических характеристик отдельных изотопов плазмы при температурах протекания термоядерной реакции;
2. Ускорить получение значений физических характеристик изотопов плазмы для интересующих температур, внутри программного предела;
3. Реализовать расчетное исследование взаимодействия нейтронного потока бланкета с плазмой в широком диапазоне ее температур.

**Оглавление**

[**Введение** 5](#_Toc91278385)

[**Раздел 1. Аналитическая часть** 6](#_Toc91278386)

[**1.1** **Изучение и анализ формирования групповых микросечений нуклидов и их температурной зависимости.** 6](#_Toc91278387)

[**1.2** **Формирование констант взаимодействия нейтронов с топливом в SERPENT 2.** 8](#_Toc91278388)

[**1.2.1 Метод Монте-Карло. Общее представление о методе.** 8](#_Toc91278389)

[**1.2.2 Общая схема метода Монте-Карло** 9](#_Toc91278390)

[**1.2.3 Моделирование случайной величины**  9](#_Toc91278391)

[**1.2.4 Характерные черты и специфика метода прямого моделирования в расчетах реакторных систем** 10](#_Toc91278392)

[**1.2.5 Общая схема моделирования методом Монте-Карло** 13](#_Toc91278393)

[**1.2.6 Формирование констант взаимодействия нейтронов с топливом** 13](#_Toc91278394)

[**1.3 Взаимодействие нейтронов бланкета с термоядерной плазмой.** 14](#_Toc91278395)

[**1.4 Анализ имеющихся систем для реализации программной системы** 16](#_Toc91278396)

[**1.4.1 Кластер Senpai в НИЯУ МИФИ** 16](#_Toc91278397)

[**1.4.2 Характеристики кластера** 17](#_Toc91278398)

[**Раздел 2. Теоретическая часть** 17](#_Toc91278399)

[**2.1 Функциональные возможности программной системы «Плазма».** 17](#_Toc91278400)

[**2.2 Линейная регрессия** 17](#_Toc91278401)

[**2.2.1 Основы линейной регрессии** 17](#_Toc91278402)

[**2.2.2 Линейная регрессия для предсказания трендов** 18](#_Toc91278403)

[**2.2.3 Регрессионный анализ** 18](#_Toc91278404)

[**2.3 Задание оптимального по времени входного файла для формирования групповых микросечений нуклидов** 19](#_Toc91278405)

[**2.4 Компоненты программной системы** 19](#_Toc91278406)

[**2.4.1 База данных** 20](#_Toc91278407)

[**2.4.1.1 Реляционная база данных** 20](#_Toc91278408)

[**2.4.1.2 Структура реляционных баз данных** 20](#_Toc91278409)

[**2.4.1.3 Реляционная модель** 20](#_Toc91278410)

[**2.4.1.4 Преимущества реляционных баз данных** 21](#_Toc91278411)

[**2.4.1.5 Целостность данных** 21](#_Toc91278412)

[**2.4.2 Модель машинного обучения** 22](#_Toc91278413)

[**2.4.2.1 Формальная постановка** 22](#_Toc91278414)

[**Раздел 3. Инженерная часть** 23](#_Toc91278415)

[**3.1 Проектирование системы для прогноза физических характеристик изотопов в области температур термоядерной плазмы** 23](#_Toc91278416)

[**3.2 Проектирование базы данных для хранения считанных и предсказанных данных** 24](#_Toc91278417)

[**3.3 Выбор алгоритмов машинного обучения для решения задачи интерполяции** 29](#_Toc91278418)

[**3.4 Проектирование интерфейса программного комплекса** 30](#_Toc91278419)

[**Раздел 4. Технологическая часть** 31](#_Toc91278420)

[**4.1 Создание входного файла для расчета физических характеристик изотопов** 33](#_Toc91278421)

[**4.2 Реализация базы данных для хранения считанных и предсказанных данных** 37](#_Toc91278422)

[**4.3 Реализация базовых моделей машинного обучения** 43](#_Toc91278423)

[**4.4 Реализация Web-интерфейса** 49](#_Toc91278424)

[**Заключение** 55](#_Toc91278425)

[**Список литературы** 56](#_Toc91278426)

[**Приложение** 58](#_Toc91278427)

# **Введение**

При работе ядерных реакторов протекает цепная реакция деления тяжелых ядер. Деление урана под действием нейтронов не только приводит к выделению энергии, но и сопровождается генерацией нейтронов следующего поколения, обеспечивающих последующие деления урана и генерацию новых нейтронов.

В термоядерных установках с магнитным удержанием (DT) - плазмы в результате реакции синтеза наряду с выделением энергии тоже рождаются нейтроны (термоядерные с энергией En = 14.1 МэВ). Эти нейтроны размножаются в бланкете, замедляются и диффундируют по бланкету, претерпевая утечку и поглощение. Естественно, что в результате диффузии эти нейтроны в плазменной камере взаимодействуют с плазмой. Однако в силу чрезвычайно низкой концентрации ионов в плазме по сравнению с концентрацией атомных ядер в бланкете баланс расхода нейтронов в установке практически полностью определяется физическими свойствами бланкета.

Если в плазме будут содержаться компоненты с большим сечением поглощения нейтронов, можно ожидать, что некоторая значимая доля нейтронов будет поглощаться также и в плазме. В качестве таких компонентов можно назвать He-3, Li-6 и B-10, которые рассматриваются в перспективных топливных циклах управляемого термоядерного синтеза. В результате, например, реакции He-3 (n, H) T в плазме появляется тритий, взаимодействующий с дейтерием (что приводит к новой реакции синтеза с генерацией нейтрона), а энергия реакции передается плазме, нагревая ее (что приводит к интенсификации реакций синтеза с дополнительной генерацией нейтронов). Таким образом, в термоядерной установке тоже может возникнуть управляемая цепная реакция синтеза с участием нейтронов (ЦРСн).

Возможность такой реакции уже рассматривалась в 40-е и 50-е гг. прошлого столетия при разработке термоядерного оружия. Тогда предполагалось использовать эту цепную реакцию в первом испытании термоядерного устройства «Mike» (испытание США 1 ноября 1952 г. на атолле Эниветок в Тихом океане). В Советском Союзе предполагалось использовать эту цепную реакцию синтеза с участием нейтронов при испытании «слойки» А. Д. Сахарова (12 августа 1953 г. на Семипалатинском полигоне). И в СССР, и в США реакция с использованием 6LiD (это соединение в то время ласково называлось «лидочкой») была успешно применена при испытаниях последующих термоядерных зарядов. Строго говоря, упомянутую реализацию ЦРСн можно отнести к категории «неуправляемого» термоядерного синтеза.

# **Раздел 1. Аналитическая часть**

# **Изучение и анализ формирования групповых микросечений нуклидов и их температурной зависимости.**

Двухгрупповое приближение, как и одногрупповое, может быть применено к реактору с произвольным спектром в том случае, если имеются удовлетворительные групповые константы, так как точность результатов определяется в первую очередь ими. Теория двух групп чаще используется в реакторе на тепловых нейтронах. Шагом вперед по сравнению с теорией одной группы является существенное увеличение точности распределения плотности потока тепловых нейтронов и энерговыделения вблизи отражателя или областей, содержащих замедлитель, ввиду более правильного описания источников тепловых и быстрых нейтронов.

Будем считать  — уровень энергии, разделяющий нейтроны на тепловые и быстрые. Обозначим:

Тогда уравнение баланса для каждой группы можно записать в виде:

Данную систему уравнений обычно упрощают, исходя из следующих физических фактов:

При этом источник быстрых нейтронов  можно получить в виде:

Тогда систему уравнений можно записать в виде:

Плотности потоков нейтронов, входящие в уравнения, являются функциями пространственных координат и энергии. Энергетическая зависимость проявляется в самом факте разделения нейтронов на две группы. В многозонном реакторе (например, с отражателем) отношение плотностей быстрых и тепловых нейтронов меняется по объему.

Типичное решение системы для реактора с отражателем приводится на рисунке 1

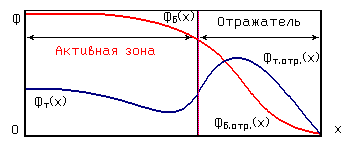


Рисунок 1. Радиальное распределение плотности потока быстрых и тепловых нейтронов

Вдали от границы зон отношение практически не меняется (реактор как бы гомогенный). Вблизи границы и в отражателе картина принципиально меняется. Здесь наблюдается всплеск плотности потока тепловых нейтронов. Особенно велик перепад для активных зон легководных реакторов на тепловых нейтронах, так как в активной зоне много урана, снижающего замедляющую способность среды и увеличивающего поглощение. В отражателе же имеется существенный источник тепловых нейтронов: образование тепловых вследствие замедления быстрых. У последних, наоборот, в отражателе источники отсутствуют (так как там нет делящегося материала) и кривая плотности потока быстрых нейтронов резко спадает к нулю.

# **Формирование констант взаимодействия нейтронов с топливом в SERPENT 2.**

## **1.2.1 Метод Монте-Карло. Общее представление о методе.**

Метод Монте-Карло появился в 1949 году, после публикации статьи “The Monte Carlo method[[1]](#footnote-1)”. Создателями метода считают Д. Неймана и С. Улама. Интересно, что теоретическая основа была известна еще до публикации статьи. Однако, до появления ЭВМ, метод не пользовался популярностью, так как подразумевал большой объём работы.

Метод имеет две особенности[[2]](#footnote-2):

1. Простая структура вычислительного алгоритма – с помощью программного кода описывают одно испытание, затем это испытание повторяют N раз. Испытания являются независимыми. Необходим лишь хороший датчик случайных чисел.
2. Ошибка вычислений, как правило, пропорциональна , где D – постоянная, N – число испытаний. Для того, чтобы уменьшить ошибку в k раз, нужно увеличить N в раз, поэтому точность увеличивают точность с помощью подбора постоянной D.

Этот метод применяется в тех случаях, когда построение аналитической модели по той или иной причине является затруднительным.

Идея метода Монте-Карло достаточно проста и состоит в следующем[[3]](#footnote-3). Вместо того, чтобы описывать случайное явление с помощью аналитических зависимостей, производится «розыгрыш», то есть моделирование случайного явления с помощью специальной процедуры, дающей случайный результат. В результате «розыгрыша» получается одна реализация случайного явления. Произведя такой розыгрыш большое число раз, можно получить статистический материал, который обрабатывается обычными методами мат. статистики.

Большое число реализаций, требующихся при применении метода Монте-Карло, делает его достаточно трудоёмким, поэтому всегда вначале имеет смысл попытаться решить задачу аналитически, и только если это не удается, прибегать к статистическому моделированию.

Вследствие этих особенностей, метод Монте-Карло иногда называют *методом статистических испытаний,* целью которого является моделирование случайных величин, а затем вычисления характеристик их распределений.

Таким образом, Метод Монте-Карло позволяет моделировать процесс, на протекание которого влияют случайные факторы. Также, в задачах, в которых отсутствуют случайные события, можно искусственно придумать вероятностную модель, чтобы решать эти задачи. Следовательно, этот метод является универсальным при решении математических задач.

## **1.2.2 Общая схема метода Монте-Карло**

Пусть требуется вычислить неизвестную величину m. Подберём случайную величину такую, что и . Рассмотрим N случайных величин , распределения которых совпадают с распределением . По центральной предельной теореме теории вероятностей распределение суммы будет иметь параметры и .

По правилу “трех сигм” получим:

После преобразований имеем:

Полученное соотношение дает метод для расчета и оценку погрешности.

## **1.2.3 Моделирование случайной величины**

Для решения задач методом Монте-Карло нужны наборы значений случайных величин. Получение таких наборов называется розыгрышем случайной величины.

Широко известны три способа розыгрыша случайных величин[[4]](#footnote-4):

1. Таблицы случайных чисел
2. Генераторы случайных чисел
3. Метод псевдослучайных чисел

Первая таблица случайных чисел была опубликована в 1955 году[[5]](#footnote-5). Достоинством таблицы является то, что проверка свойств однократная, но недостатком является ограниченный запас чисел.

Генераторы случайных величин могут быть физическими или программными.

Физический генератор преобразует в числовую последовательность результат природного случайного процесса. Он обладает преимуществом, что дает действительно случайную последовательность. Однако, недостатки генератора в том, что его периодически нужно проверять на работоспособность и невозможно воспроизводить последовательность.

Программный генератор чаще всего реализован в виде программы, которая при каждом обращении к ней возвращает псевдослучайное число. То есть на самом деле, числа не являются по-настоящему случайными. Если программа качественная, то полученная последовательность случайных чисел удовлетворяет статистическим тестам на независимость и равномерность.

Метод псевдослучайных чисел – это получение чисел с помощью формул. Данные значения могут считаться случайными для решения некоторого класса задач. Достоинством является однократная проверка, малый занимаемый объем памяти и большая скорость получения значений. Недостаток – ограниченный запас чисел, потому что все известные алгоритмы получения псевдослучайных чисел имеют конечный период.

## **1.2.4 Характерные черты и специфика метода прямого моделирования в расчетах реакторных систем**

Метод Монте-Карло – наиболее универсальный метод, применяемый для расчета переноса излучений. Программы, использующие данный метод, позволяют моделировать системы с произвольной геометрией, используя комбинаторный подход, основанный на описании сложных пространственных форм комбинациями простых тел или поверхностей с помощью теоретико-множественных операций пересечения, дополнения и объединения. При расчётах такие программы используют значения микроконстант, относящиеся непосредственно к рассматриваемому нуклиду. Таким образом, память вычислительной техники используется только для хранения информации о нуклидах в материале. Константные значения, применяемые в данном методе, являются поточечными. Такой подход обеспечивает возможность моделирования с непрерывным слежением за энергией частицы.

Преимущество метода Монте-Карло заключается в возможности точного описания любой геометрии и использовании досконального (поточечного) представления микроконстант. Это сокращает до минимума количество применяемых при расчёте приближений. Таким образом, метод Монте-Карло способен заменить реальный эксперимент. Также данный метод позволяет моделировать полёт частиц без какой-либо дискретизации. Но есть и недостатки. Метод Монте-Карло затрачивает большое количество времени на получение результата. В последнее время этот недостаток компенсируют с помощью многопроцессорных кластеров для увеличения мощности вычислительной техники, чтобы ускорить процесс расчёта данным методом.

При расчёте методом Монте-Карло в реакторных системах моделируется поведение индивидуальных частиц от момента рождения до поглощения или вылета частицы из системы. Так как нейтральные частицы практически не взаимодействуют друг с другом, то их траектории движения можно считать независимыми. Этот факт позволяет применять метод Монте-Карло для моделирования траекторий нейтральных частиц. Для расчёта переноса заряженных частиц данный метод применяется реже и сталкивается со значительными трудностями. Совокупность всех событий, произошедших с частицей от момента рождения до “смерти”, называется *траекторией* или *историей частицы[[6]](#footnote-6)*.

В экспериментах измеряют параметры расчёта вероятности взаимодействия нейтральных частиц с ядрами. Полученные значения хранят в специальных файлах оцененных ядерных данных.

Пробег частицы до столкновения в бесконечной однородной среде с полным макроскопическим сечением можно получить с помощью случайных чисел γ, равномерно распределенных в интервале (0, 1):

- плотность вероятности случайной величины .

При столкновении необходимо определить нуклид, с которым столкнулся нейтрон. Для этого генерируют случайное число равномерно распределенное на интервале (0, 1). Считается, что взаимодействие происходит с нуклидом c номером k, если:

где K – число нуклидов в материале, – парциальное полное сечение нуклида c номером j.

После того, как будет определен номер нуклида, на котором было зафиксировано столкновение, необходимо определить реакцию, которая произойдет в результате столкновения. Результатами столкновения могут быть рассеяние или поглощение, но это не конечная реакция, которая может произойти. В результате поглощения нейтрона ядром могут произойти деление ядра или захват нейтрона ядром. Определение типа реакции происходит с помощью неравенства. Считают, что происходит реакция типа n, если:

где N – полное число реакций для заданного нуклида.

В процессе моделирования траектории частицы выделяются события. Для регистрации используются следующие события: свободный пробег частицы внутри выделенной области пространства, столкновение частицы с ядром и поглощение частицы.

Так как метод Монте-Карло состоит в генерации последовательности случайных величин , то обычно генерируют большое количество последовательностей случайных величин, каждая из которых даёт свой целевой функционал. Затем для каждого целевого функционала рассчитывается величина, называемая вкладом одного события в данный функционал.

Основными функционалами являются интегралы потока по объёмам и энергетическим группам[[7]](#footnote-7):

1. При регистрации по столкновениям вклад есть , при условии, что столкновение произошло в объеме , а энергия лежала в интервале , иначе ноль;
2. При регистрации по поглощениям вклад есть ;
3. При регистрации по пробегам вклад есть , где – длина пробега в объеме а энергия лежит в интервале .

События на разных траекториях независимы, а события на одной траектории являются зависимыми. Вклады по событиям одного типа по одной траектории объединяют и получают вклады траекторий по столкновениям, поглощениям и пробегам. При расчёте окончательной оценки суммируют все вклады данного типа всех траекторий и делят на суммарное число траекторий.

## **1.2.5 Общая схема моделирования методом Монте-Карло**

При моделировании переноса нейтронов и фотонов методом Монте-Карло на вход программы подаётся начальное расположение частицы, то есть её координаты, энергия частицы, направление полёта и тип частицы. В процессе моделирования могут возникать новые частицы, что соответствует реальному физическому процессу. Частицы, появившиеся в процессе моделирования называют потомками нейтрона. Таким образом, начальный нейтрон порождает дерево траекторий, которое называют *расширенной траекторией.*

Существует две задачи переноса частиц[[8]](#footnote-8):

1. задача с источником (неоднородная);
2. задача на (однородная)

## **1.2.6 Формирование констант взаимодействия нейтронов с топливом**

Первоначальной целью Serpent было создание гомогенизированных групповых констант (иногда неправильно называемых просто гомогенизированными сечениями) для решателей пониженного порядка, таких как коды узловой диффузии. Идея генерации групповых констант состоит в том, чтобы рассчитать репрезентативные нейтронные свойства системы (например, топливной сборки) таким образом, чтобы сохранить скорость физических реакций в системе. По сути, пространственные вариации внутри системы усредняются, что приводит к единому набору групповых констант для всей системы (пространственная гомогенизация нейтронных свойств), кроме того, энергетические вариации нейтронных свойств усредняются до структуры энергетической группы (энергия конденсация нейтронных свойств).

Поскольку групповые константы (нейтронно-физические свойства) системы зависят от нуклидного состава различных материалов в системе, а также от их температуры и плотности, групповые константы обычно генерируются в широком диапазоне условий и параметризуются, чтобы покрыть это диапазон условий.

SERPENT предлагает автоматизированные инструменты для таких целей, а вариации системных параметров делятся на[[9]](#footnote-9):

1. Исторические вариации, которые учитывают условия, сохраняющиеся в течение длительного периода времени, такие как температура и плотность замедлителя, концентрация бора и расположение регулирующих стержней. Эти условия отражаются в обедненных топливных композициях и, следовательно, в генерируемых групповых константах.
2. Варианты ответвлений, которые учитывают мгновенные изменения рабочих условий, таких как температура топлива, плотность и температура замедлителя, концентрация бора и установка регулирующих стержней.

Исторические вариации учитываются путем истощения топливной сборки отдельно с каждым набором исторических условий для получения нуклидных композиций, которые затем могут использоваться для получения групповых констант. Затем изменения ветвей покрываются путем взятия составов нуклидов, соответствующих некоторым историческим условиям, и получения групповых констант, при этом мгновенно изменяя рабочие условия, чтобы отразить изменение ветвей.

# **1.3 Взаимодействие нейтронов бланкета с термоядерной плазмой.**

Рассматривать цепную реакцию синтеза с участием нейтронов (ЦСРн) целесообразно применительно к термоядерным установкам с магнитным удержанием плазмы, снабженным бланкетами для преобразования энергии. Принципиальная особенность таких установок состоит в сочетании плазменной зоны с низкой концентрацией компонентов и зоны бланкета с твердыми (жидкими) материалами (т.е. зоны с намного порядков более высокими концентрациями нуклидов). Утилизация нейтронов, генерируемых термоядерными реакциями, в установке с такими разнородными зонами, как правило, ограничивается бланкетом, а плазма для нейтронов практически прозрачна.

Однако можно представить себе условия, когда поглощение нейтронов в плазме и в бланкете окажется сопоставимым. Такое положение может возникнуть, во-первых, если в плазме в качестве компонента содержится нуклид с большим сечением поглощения нейтронов, во-вторых, если одновременно в бланкете применяются материалы с рекордно низким сечением захвата нейтронов.

Для плазмы таким нуклидом (сильно поглощающим нейтроны) с малым атомным весом может быть, например, He-3, который характеризуется большим сечением поглощения тепловых нейтронов (в тепловой точке ~5330 барн) в реакции He-3(n,1H)T. Энергия этой экзотермической реакции составляет +0.764 МэВ, которая может использоваться для нагрева плазмы. Если так, то можно говорить об использовании не только, например, (D–3He), но и даже (DT–3He) термоядерного топливного цикла.

Для бланкета в качестве нуклидов с малым сечением поглощения могут рассматриваться, например, Pb-208, D (тяжелая вода), графит, для которых сечение поглощения тепловых нейтронов составляет величину порядка или менее миллибарна[[10]](#footnote-10).

Далее будем опираться на использование изотопа He-3 (отметим, что гелий уже рассматривается в качестве примеси для дополнительного нагрева плазмы методом ионноциклотронного резонанса). Традиционно считается, что термоядерные нейтроны будут размножаться в (n,2n) и (n,3n) реакциях с материалами бланкета (см. табл. 1), замедляться и диффундировать в его объеме.[[11]](#footnote-11)

Таблица 1

Полное число нейтронов в бланкете (бесконечных размеров), замедляющихся ниже порога (n, 2n) - реакции, в расчете на один термоядерный нейтрон

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Элемент | Be | C | Fe | Mo | W | Pb |
| Концентрация атомов, | 0.120 | 0.08 | 0.085 | 0.064 | 0.063 | 0.063 |
| Полное число нейтронов, замедляющихся под порог (n, 2n) - реакции | 1.78 | 1.0 | 1.29 | 1.84 | 1.82 | 1.84 |

Из таблицы 1 видно, что термоядерные нейтроны существенным образом могут размножиться в слое бланкета (Pb, Mo, W, Be), прилегающем к первой стенке.

Из всех нейтронов при дальнейшем замедлении и диффузии некоторая часть будет поглощаться ядрами He-3 в плазме и приводить вновь к рождению ядра трития, участию его в термоядерной реакции и вновь к рождению нейтронов. Таким образом, замкнется звено цепной нейтронной реакции, включающей в себя и термоядерные реакции. Если основным компонентом будет (DT) плазма, то рождающиеся в результате (DT) синтеза нейтроны будут также размножаться и диффундировать в бланкете, в плазменной камере и захватываться ядрами He-3, то есть участвовать в цепной реакции.[[12]](#footnote-12)

# **1.4 Анализ имеющихся систем для реализации программной системы**

## **1.4.1 Кластер Senpai в НИЯУ МИФИ**

Для проведения расчётов было выполнено подключение к кластеру НИЯУ МИФИ Senpai[[13]](#footnote-13). Для подключения к кластеру требуется сохраненный на компьютере непубличный SSH-ключ и логин. Для связи с кластером указывается используемый протокол - SFTP и координаты кластера. Для запуска Serpent на кластере используется планировщик задач Slurm, позволяющий размещать, удалять и отслеживать большое количество процессов, исполняемых на распределенных системах.

Для того, чтобы поставить задачу на расчет, используется команда sbatch task.sh, где task.sh – текстовый файл, описывающий задачу и ресурсы, которые необходимо выделить для ее решения.

Кластер состоит из четырех нод по 40 ядер каждый. Пользователи ограниченны одним нодом и, соответственно, 40 ядрами.

Таким образом, для запуска расчета нужно:

1. Подготовить входной файл
2. Создать в папке с входным фалом файл task.sh (может называться иначе)
3. Запустить расчет командой sbatch task.sh из папки, где расположен task.sh
4. Файлы с результатами расчета будут расположены в папке с входным файлом

Другие полезные команды:

1. squeue – показывает текущие задачи, исполняемые на кластере, или стоящие в очереди на исполнение
2. scancel id – отменяет задачу с id равным номеру задачи (id задачи можно посмотреть с помощью squeue)

## **1.4.2 Характеристики кластера**

4 nodes, 160 cores, 1.5 TB RAM, 8 TB HDD, 10 TFLOPS

Intel(R) Xeon(R) Gold 6230 CPU @2.10GHz

# **Раздел 2. Теоретическая часть**

# **2.1 Функциональные возможности программной системы «Плазма».**

Разработка системы основана на оптимальном выборе оборудования с использованием современных технических средств.

Система состоит из четырех режимов:

1. Режим сбора данных из результирующего файла на вычислительном кластере Senpai, полученного с помощью SERPENT
2. Режим сохранения полученных данных в базу данных.
3. Режим запуска конвейера для создания предсказания с помощью модели машинного обучения.
4. Сохранение предсказанных значений в базу данных.

Также, система должна иметь интерфейс для ввода параметров входного файла SERPENT.

# **2.2 Линейная регрессия**

## **2.2.1 Основы линейной регрессии**

Пусть функция регрессии Y на X имеет вид:

Тогда регрессионная модель отклика Y выглядит следующим образом:

Данная модель также называется простейшей линейной регрессионной моделью.

## **2.2.2 Линейная регрессия для предсказания трендов**

Наиболее обоснованный подход — использование динамических (трендовых) моделей.[[14]](#footnote-14) Сама построенная модель дает оценку тенденции изменения показателя. Предполагая, что эта тенденция сохранится в будущем, полученная трендовая модель используется для расчета прогнозируемого значения. Следует обратить внимание на то, что при прогнозировании предпочтение отдается простым моделям, содержащим меньшее количество параметров. Чаще всего используется линейный тренд. Заметим, что получаемое точечное значение вряд ли будет достигнуто в точности, поскольку фактические уровни, как правило, не совпадают с трендом, а колеблются около него. Именно поэтому прогноз может даваться с учетом этих колебаний: в интервале от прогнозируемого значения минус точность модели до прогнозируемого значения плюс точность модели. Такой прогноз учитывает направление тренда, и колеблемость уровней вокруг него.

## **2.2.3 Регрессионный анализ**

Регрессионный анализ — метод моделирования измеряемых данных и исследования их свойств[[15]](#footnote-15). Данные состоят из пар значений зависимой переменной (переменной отклика) и независимой переменной (объясняющей переменной). Регрессионная модель есть функция независимой переменной и параметров с добавленной случайной переменной. Параметры модели настраиваются таким образом, что модель наилучшим образом приближает данные. Критерием качества приближения (целевой функцией) обычно является среднеквадратичная ошибка: сумма квадратов разности значений модели и зависимой переменной для всех значений независимой переменной в качестве аргумента. Регрессионный анализ — раздел математической статистики и машинного обучения. Предполагается, что зависимая переменная есть сумма значений некоторой модели и случайной величины. Относительно характера распределения этой величины делаются предположения, называемые гипотезой порождения данных. Для подтверждения или опровержения этой гипотезы выполняются статистические тесты, называемые анализом остатков. При этом предполагается, что независимая переменная не содержит ошибок. Регрессионный анализ используется для прогноза, анализа временных рядов, тестирования гипотез и выявления скрытых взаимосвязей в данных.

Регрессия — зависимость математического ожидания (например, среднего значения) случайной величины от одной или нескольких других случайных величин (свободных переменных), то есть  . Регрессионным анализом называется поиск такой функции , которая описывает эту зависимость. Регрессия может быть представлена в виде суммы неслучайной и случайной составляющих:

где  — функция регрессионной зависимости, а  — аддитивная случайная величина с нулевым матожиданием. Предположение о характере распределения этой величины называется гипотезой порождения данных. Обычно предполагается, что величина    имеет гауссово распределение с нулевым средним и дисперсией .

Задача нахождения регрессионной модели нескольких свободных переменных ставится следующим образом. Задана выборка — множество   значений свободных переменных и множество   соответствующих им значений зависимой переменной. Эти множества обозначаются как , множество исходных данных . Задана регрессионная модель — параметрическое семейство функций  зависящая от параметров  и свободных переменных . Требуется найти наиболее вероятные параметры :

Функция вероятности  зависит от гипотезы порождения данных и задается Байесовским выводом или методом наибольшего правдоподобия.

# **2.3 Задание оптимального по времени входного файла для формирования групповых микросечений нуклидов**

# **2.4 Компоненты программной системы**

Система состоит из 3 основных компонент:

* Базы данных, в которой хранятся выходные данные из SERPENT 2, полученные предсказания, введенные пользователем параметры
* Конвейера модели машинного обучения
* Веб-интерфейса для введения пользователем параметров

## **2.4.1 База данных**

### **2.4.1.1 Реляционная база данных**

Реляционные базы данных представляют собой базы данных, которые используются для хранения и предоставления доступа к взаимосвязанным элементам информации. Реляционные базы данных основаны на реляционной модели — интуитивно понятном, наглядном табличном способе представления данных. Каждая строка, содержащая в таблице такой базы данных, представляет собой запись с уникальным идентификатором, который называют ключом. Столбцы таблицы имеют атрибуты данных, а каждая запись обычно содержит значение для каждого атрибута, что дает возможность легко устанавливать взаимосвязь между элементами данных.

### **2.4.1.2 Структура реляционных баз данных**

Реляционная модель подразумевает логическую структуру данных: таблицы, представления и индексы. Логическая структура отличается от физической структуры хранения. Такое разделение дает возможность администраторам управлять физической системой хранения, не меняя данных, содержащихся в логической структуре. Например, изменение имени файла базы данных не повлияет на хранящиеся в нем таблицы.

Разделение между физическим и логическим уровнем распространяется в том числе на операции, которые представляют собой четко определенные действия с данными и структурами базы данных. Логические операции дают возможность приложениям определять требования к необходимому содержанию, в то время как физические операции определяют способ доступа к данным и выполнения задачи.

Чтобы обеспечить точность и доступность данных, в реляционных базах должны соблюдаться определенные правила целостности. Например, в правилах целостности можно запретить использование дубликатов строк в таблицах, чтобы устранить вероятность попадания неправильной информации в базу данных.

### **2.4.1.3 Реляционная модель**

В первых базах данных данные каждого приложения хранились в отдельной уникальной структуре. Если разработчик хотел создать приложение для использования таких данных, он должен был хорошо знать конкретную структуру, чтобы найти необходимые данные. Такой метод организации был неэффективен, сложен в обслуживании и затруднял оптимизацию эффективности приложений. Реляционная модель была разработана, чтобы устранить потребность в использовании разнообразных структур данных.

Она обеспечила стандартный способ представления данных и отправки запросов, которые могли быть использованы в любых приложениях. Разработчики уяснили, что таблицы являются ключевым преимуществом реляционных баз данных, так как обеспечивают интуитивно понятный, эффективный и гибкий способ хранения структурированной информации и получения к ней доступа.

Со временем, когда разработчики стали использовать язык структурированных запросов (SQL) для записи данных в базу и отправки запросов, стало очевидным и другое преимущество реляционной модели. Вот уже на протяжении многих лет SQL широко используется в качестве языка запросов в базах данных. Он основан на алгоритмах реляционной алгебры и четкой математической структуре, что обеспечивает простоту и эффективность при оптимизации любых запросов к базе данных. Для сравнения: при использовании других подходов приходится создавать отдельные, уникальные запросы.

### **2.4.1.4 Преимущества реляционных баз данных**

Компании всех типов и размеров используют простую, но функциональную реляционную модель для обслуживания разнообразных информационных потребностей[[16]](#footnote-16). Реляционные базы данных применяются для отслеживания товарных запасов, обработки торговых транзакций через Интернет, управления большими объемами критически важных данных заказчиков и т. д. Реляционные базы данных можно рекомендовать для обслуживания любых информационных потребностей, где элементы данных связаны между собой и необходимо обеспечивать безопасное и надежное управление ими на основе правил целостности.

Реляционные базы данных появились в 1970-х годах. На сегодняшний день преимущества реляционного подхода сделали его самой распространенной моделью для баз данных в мире.

### **2.4.1.5 Целостность данных**

Реляционная модель наиболее эффективно поддерживает целостность данных во всех приложениях и копиях (экземплярах) базы данных. Например, когда заказчик кладет деньги на счет с помощью банкомата, а затем проверяет баланс на мобильном телефоне, он ожидает, что поступившие средства сразу же отобразятся на счете. Реляционные базы данных отлично подходят для обеспечения целостности данных в различных экземплярах базы в одно и то же время.

Другие типы баз данных не могут одновременно поддерживать целостность больших объемов данных. Некоторые современные типы баз данных, такие как NoSQL, обеспечивают только так называемую “окончательную целостность.” Это значит, что, когда выполняется масштабирование данных или несколько пользователей одновременно используют одни и те же данные, необходимо некоторое время на “внесение изменений”. В некоторых случаях окончательная целостность вполне приемлема (например, для обновления позиций в товарном каталоге), однако для критически важной операционной деятельности бизнеса (например, транзакций с использованием корзины) реляционные базы представляют собой фундаментальный стандарт.

## **2.4.2 Модель машинного обучения**

Обучение с учителем (Supervised learning) — один из разделов машинного обучения, посвященный решению следующей задачи. Имеется множество объектов (ситуаций) и множество возможных ответов (откликов, реакций). Существует некоторая зависимость между ответами и объектами, но она неизвестна. Известна только конечная совокупность прецедентов — пар «объект, ответ», называемая обучающей выборкой. На основе этих данных требуется восстановить зависимость, то есть построить алгоритм, способный для любого объекта выдать достаточно точный ответ. Для измерения точности ответов определённым образом вводится функционал качества.

Под учителем понимается либо сама обучающая выборка, либо тот, кто указал на заданных объектах правильные ответы. Существует также обучение без учителя, когда на объектах выборки ответы не задаются.

## **2.4.2.1 Формальная постановка**

Пусть X — множество описаний объектов, Y — множество допустимых ответов. Существует неизвестная целевая зависимость — отображение , значнения которой известны только на объектах конечной обучающей выборки .Требуется построить алгоритм , который приближал бы неизвестную целевую зависимость как на элементах выборки, так и на всём множестве X.

Говорят также, что алгоритм должен обладать способностью к обобщению эмпирических фактов, или выводить общее знание (закономерность, зависимость) из частных фактов (наблюдений, прецедентов).

Данная постановка является обобщением классических задач аппроксимации функций. В классической аппроксимации объектами являются действительные числа или векторы. В реальных прикладных задачах входные данные об объектах могут быть неполными, неточными, неоднородными, нечисловыми. Эти особенности приводят к большому разнообразию методов обучения с учителем.

# **Раздел 3. Инженерная часть**

# **3.1 Проектирование системы для прогноза физических характеристик изотопов в области температур термоядерной плазмы**

Реализация системы подразумевает наличие:

* Базы данных;
* Конвейера машинного обучения
* Веб-интерфейса
* Вычислительного кластера с установленным SERPENT 2

Входные файлы можно сформировать с помощью веб-интерфейса. По входным файлам для SERPENT 2 должны получены выходные файлы с суффиксом mdx0.m. Эти входные файлы рассчитываются с помощью библиотек ядерных данных ENDF[[17]](#footnote-17). Далее происходит парсинг выходных файлов[[18]](#footnote-18) и микросечения из этих файлов заносятся в базу данных. Полученные данные обрабатываются, чтобы можно было подать их в модель машинного обучения. Далее происходит расчет предсказания и полученные значения заносятся в базу данных. Также, по полученным значениям строятся графики, состоящие из данных, рассчитанных с помощью SERPENT 2 и интерполированного значения температуры, выходящего за пределы возможностей SERPENT 2.



Рисунок 2. Схема система для прогноза физических характеристик изотопов в области температур термоядерной плазмы

# **3.2 Проектирование базы данных для хранения считанных и предсказанных данных**

База данных должна содержать 2 схемы:

* Схема для таблиц с полученными данными из SERPENT 2 и предсказанными значениями;
* Схема для таблиц сгенерированных фреймворком Django.

Изображение выглядит как текст, стол

Автоматически созданное описание

Рисунок 3. Схема для полученных данных из SERPENT 2 и предсказанных значений

Таблица 2. Описание схемы *uir*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Схема | Таблица | Поле | Тип | Описание |
| uir | data | nuclide | int4 | код нуклида |
| temperature | int4 | температура, при которой выполнялся расчет |
| reaction\_102\_fast | float8 | значение микросечения для 102 канала быстрая группа |
| reaction\_103\_fast | float8 | значение микросечения для 103 канала быстрая группа |
| reaction\_102\_warm | float8 | значение микросечения для 102 канала теплая группа |
| reaction\_103\_warm | float8 | значение микросечения для 103 канала теплая группа |
| run\_time | timestamp | время внесения в базу данных |
| predicts | nuclide | int4 | код нуклида |
| temperature | int4 | температура, при которой выполнялось пресказание |
| reaction\_102\_fast | float8 | значение микросечения для 102 канала быстрая группа |
| reaction\_103\_fast | float8 | значение микросечения для 103 канала быстрая группа |
| reaction\_102\_warm | float8 | значение микросечения для 102 канала теплая группа |
| reaction\_103\_warm | float8 | значение микросечения для 103 канала теплая группа |
| metric\_value\_102\_fast | float8 | метрика при кросс-валидации для микросечения для 102 канала быстрая группа |
| metric\_value\_103\_fast | float8 | метрика при кросс-валидации для микросечения для 103 канала быстрая группа |
| metric\_value\_102\_warm | float8 | метрика при кросс-валидации для микросечения для 102 канала теплая группа |
| metric\_value\_103\_warm | float8 | метрика при кросс-валидации для микросечения для 103 канала теплая группа |
| run\_time | timestamp | время предсказания |

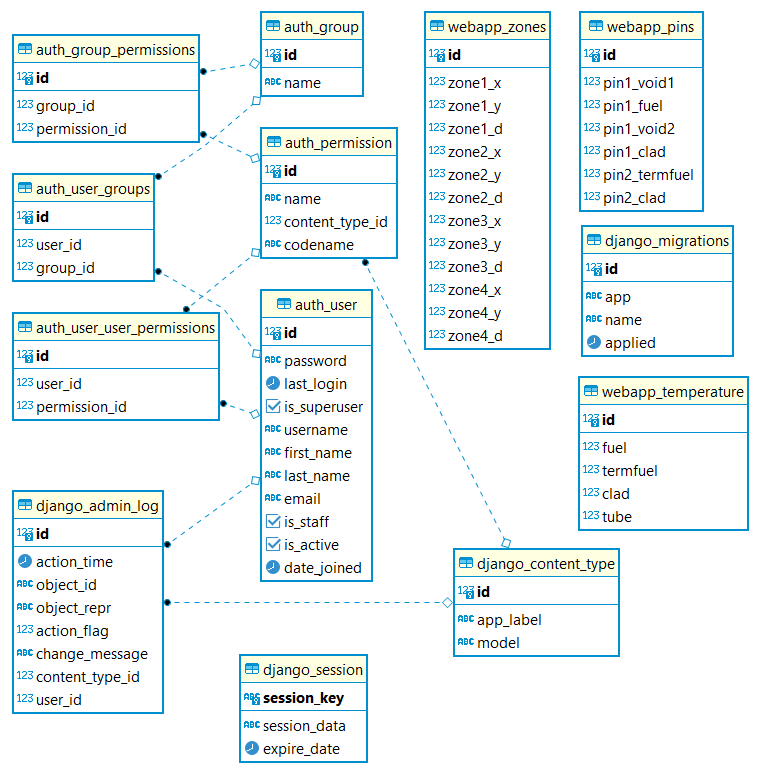


Рисунок 4. Схема для веб-интерфейса

Таблица 3. Описание схемы public

|  |  |  |  |
| --- | --- | --- | --- |
| Схема | Таблица | Поле | Тип |
| public | auth\_group\_permissions | id | bigserial |
| group\_id | int4 |
| permission\_id | int4 |
| auth\_user\_groups | id | bigserial |
| user\_id | int4 |
| group\_id | int4 |
| auth\_user\_user\_permissions | id | bigserial |
| user\_id | int4 |
| permission\_id | int4 |
| django\_admin\_log | action\_time | serial |
| object\_id | timestamptz |
| object\_repr | text |
|  |  | action\_flag | varchar |
| change\_message | text |
| content\_type\_id | int4 |
| user\_id | int4 |
| auth\_group | id | serial |
| name | varchar |
| auth\_permission | id | serial |
| name | varchar |
| content\_type\_id | int4 |
| codename | varchar |
| auth\_user | id | serial |
| password | varchar |
| last\_login | timestamptz |
| is\_superuser | bool |
| username | varchar |
| first\_name | varchar |
| last\_name | varchar |
| email | varchar |
| is\_staff | bool |
| is\_active | bool |
| data\_joined | timestamptz |
| django\_session | session\_key | varchar |
| session\_data | text |
| expire\_date | timestamptz |
| django\_content\_type | id | serial |
| app\_label | varchar |
| model | varchar |
| webapp\_zones | id | bigserial |
| zone1\_x | float8 |
| zone1\_y | float8 |
| zone1\_d | float8 |
| zone2\_x | float8 |
| zone2\_y | float8 |
| zone2\_d | float8 |
| zone3\_x | float8 |
| zone3\_y | float8 |
| zone3\_d | float8 |
| zone4\_x | float8 |
| zone4\_y | float8 |
| zone4\_d | float8 |
| webapp\_pins | id | bigserial |
| pin1\_void1 | float8 |
| pin1\_fuel | float8 |
| pin1\_void2 | float8 |
| pin1\_clad | float8 |
| pin2\_termfuel | float8 |
| pin2\_clad | float8 |
| django\_migrations | id | bigserial |
| app | varchar |
| name | varchar |
|  |  | applied | timestamptz |
| webapp\_temperature | id | bigserial |
| fuel | int4 |
| termfuel | int4 |
| clad | int4 |
| tube | int4 |

# **3.3 Выбор алгоритмов машинного обучения для решения задачи интерполяции**

Для решения задачи интерполяции был выбран алгоритм линейной регрессии.

Чтобы алгоритм работал корректно, нужно произвести преобразование данных:

* Преобразование категориальных признаков
* Стандартизация данных

В силу того, что зависимость микросечения нуклида от температуры немонотонна, cгенерирована новая матрица признаков, состоящая из всех полиномиальных комбинаций признаков. Степень признаков подбиралась с помощью поиска по сетке.

Таким образом, для получения предсказания используется следующий конвейер:

* Стандартизация данных
* Преобразование категориальных признаков
* Добавление полиномиальных комбинаций

Изображение выглядит как текст, внутренний

Автоматически созданное описание

Рисунок 5. Конвейер машинного обучения

Для подбора гиперпараметров используются поиск по сетке. Для оценки тех или иных параметров используется оценка по кросс-валидации на 10 фолдах.

Также, для оценки модели используется разделение на тренировочную и тестовую выборки в соотношении 70 к 30.

# **3.4 Проектирование интерфейса программного комплекса**

В данной версии программного комплекса имеется возможность задать параметры для шаблона входного файла. Можно задавать такие параметры как:

* Температура материала
* Параметры ячейки
* Параметры поверхности

Для каждого из параметров должна быть создана отдельная страница, с полями для ввода параметров, кратким описанием того, какие данные нужно ввести и картинкой с примером или поясняющей значение параметров.

Также, должна быть создана домашняя страница, на которой описываются параметры, которые можно ввести, приводится полный шаблон файла, который будет сформирован по итогу введения параметров, ссылками на документацию SERPENT 2 по всем параметрам и навигационное меню.

Все введенные данные должны быть занесены в базу данных, из которой введенные параметры будут перенаправлены в конечный входной файл для SERPENT 2.

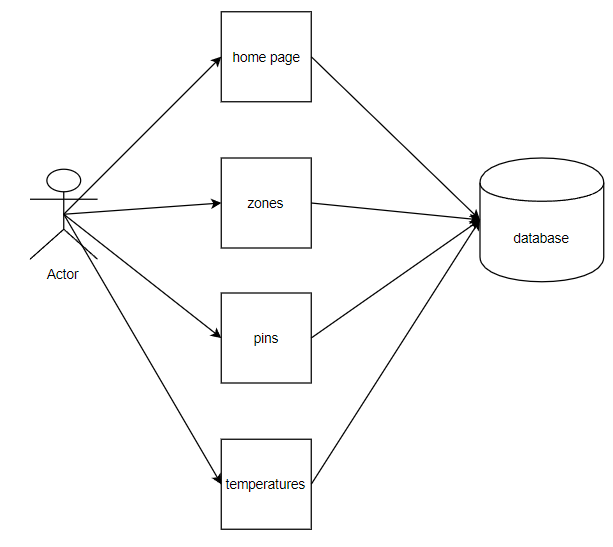


Рисунок 6. Схема веб-приложения

# **Раздел 4. Технологическая часть**

В данном разделе описывается реализация модулей программного комплекса “ПЛАЗМА” и демонстрируется работоспособность.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7. Структура кода, обеспечивающего работу конвейера машинного обучения

Таблица 4. Описание модулей кода конвейера модели машинного обучения

|  |  |
| --- | --- |
| Название модуля | Описание модуля |
| data | Модуль для хранения данных для обучения модели, то есть тренировочные данные |
| model\_code/html\_graphics | Модуль для хранения графиков с тренировочными данными и полученной интерполяцией |
| model\_code/logs\_files | Модуль для хранения логирования работы системы |
| model\_code/python | Модуль, внутри которого содержатся основные функции для работы конвейера машинного обучения |
| model\_code/settings | Модуль, в котором хранятся основные настройки: данные для подключения к базе данных, пути к SQL скриптам и тд. |
| model\_code/sql | Модуль, в котором хранятся SQL скрипты |
| model\_code/templates | Модуль, в котором хранится шаблон входного файла и новый входной файл, полученный путем вставки данных, введенных пользователем |

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8. Структура кода, обеспечивающего работу веб-приложения

Таблица 5. Описание модулей веб-приложения

|  |  |
| --- | --- |
| Название модуля | Описание модуля |
| static/img | Модуль, в котором хранятся используемые изображения |
| templates | Модуль, в котором хранится код верстки страниц, то есть html код |
| UIRWeb | Модуль, в котором хранятся основные настройки проекта, подключения к базе данных и тд. |
| webapp | Модуль, в котором описываются формы, модели, представления |
| manage.py | Модуль, в котором находится main функция |

# **4.1 Создание входного файла для расчета физических характеристик изотопов**

Входной файл был существенно ускорен в сравнении с предыдущими версиями. Ранее, расчет проводился за 16 -18 часов. Теперь расчет микросечений занимает не более 40 секунд.

*% --- VVER-440 Assembly --------------------------------------*

*set title "vver-440"*

*% --- Fuel pin with central hole:*

*pin 1*

*void Enter radius for PIN1 void*

*fuel Enter radius for PIN1 fuel*

*void Enter radius for PIN1 void*

*clad Enter radius for PIN1 clad*

*water*

*% --- Central tube with termfuel:*

*pin 2*

*termfuel Enter radius for PIN2 termfuel*

*clad Enter radius for PIN2 clad*

*water*

*% --- Empty lattice position:*

*pin 3*

*water*

*% --- Lattice (type = 2, pin pitch = 1.23 cm):*

*lat 10 2 0.0 0.0 15 15 1.23*

*3 3 3 3 3 3 3 3 3 3 3 3 3 3 3*

*3 3 3 3 3 3 3 1 1 1 1 1 1 1 3*

*3 3 3 3 3 3 1 1 1 1 1 1 1 1 3*

*3 3 3 3 3 1 1 1 1 1 1 1 1 1 3*

*3 3 3 3 1 1 1 1 1 1 1 1 1 1 3*

*3 3 3 1 1 1 1 1 1 1 1 1 1 1 3*

*3 3 1 1 1 1 1 1 1 1 1 1 1 1 3*

*3 1 1 1 1 1 1 2 1 1 1 1 1 1 3*

*3 1 1 1 1 1 1 1 1 1 1 1 1 3 3*

*3 1 1 1 1 1 1 1 1 1 1 1 3 3 3*

*3 1 1 1 1 1 1 1 1 1 1 3 3 3 3*

*3 1 1 1 1 1 1 1 1 1 3 3 3 3 3*

*3 1 1 1 1 1 1 1 1 3 3 3 3 3 3*

*3 1 1 1 1 1 1 1 3 3 3 3 3 3 3*

*3 3 3 3 3 3 3 3 3 3 3 3 3 3 3*

*% --- Surfaces (assembly pitch = 14.7 cm):*

*surf zone1 hexyc Enter zone1 x0, y0, d*

*surf zone2 hexyc Enter zone2 x0, y0, d*

*surf zone3 hexyc Enter zone3 x0, y0, d*

*surf zone4 hexyc Enter zone4 x0, y0, d*

*% --- Cells:*

*cell cell1 100 fill 10 zone1 -zone2*

*cell cell2 101 fill 2 -zone1*

*cell cell1d 0 fill 100 zone1 -zone2*

*cell cell2d 0 fill 101 -zone1*

*cell cell3 0 tube zone2 -zone3*

*cell cell4 0 water zone3 -zone4*

*cell cell5 0 outside zone4*

*% --- UO2 fuel*

*mat fuel -10.45700 vol 1 tmp Enter fuel temperature*

*92235.03c -0.03173*

*92238.03c -0.84977*

*8016.03c -0.11850*

*% --- termfuel*

*mat termfuel -0.0001 vol 1 tmp Enter termfuel temperature*

*1002.03c -0.33*

*1003.03c -0.33*

*2003.03c -0.34*

*% --- Zr-Nb cladding and shroud tube:*

*mat clad -6.55000 vol 1 tmp Enter clad temperature*

*40000.03c -0.99000*

*41093.03c -0.01000*

*mat tube -6.58000 vol 1 tmp Enter tube temperature*

*40000.03c -0.97500*

*41093.03c -0.02500*

*% --- Water:*

*mat water -0.7207 moder lwtr 1001 vol 1 tmp 800*

*1001.06c 2.0*

*8016.06c 1.0*

*% --- Thermal scattering data for light water:*

*therm lwtr lwj3.11t*

*% --- Cross section library file path:*

*set acelib "/home/SHARED/Serpent/xsdata/ENDFB7/sss\_endfb7.xsdata"*

*set declib "/home/SHARED/Serpent/xsdata/ENDFB7/sss\_endfb7.dec"*

*set nfylib "/home/SHARED/Serpent/xsdata/ENDFB7/sss\_endfb7.nfy"*

*% --- Periodic boundary condition:*

*set bc 3*

*% --- Neutron population and criticality cycles:*

*set pop 2000 500 20*

*% ----GROUP CONSTANT GENERATION--------------------------------------*

*set gcu 100 101*

*set nfg 2*

*set mcvol 10000*

*set mdep 101 1 1 termfuel 10020 102 10020 103 10030 102 10030 103 20030 102 20030 103*

*set mdep 100 1 1 fuel 922350 102 922350 18 922380 102 922380 18*

*set mdep 100 1 1 clad 922350 102 922350 18 922380 102 922380 18*

*set mdep 100 1 1 water 92235 102 922350 18 922380 102 922380 18*

*% ----SPECTR---------------------------------------------------------*

*ene eg1 1 0. 0.625E-6 20*

*det spectr\_cell1*

*de eg1*

*dc cell1d*

*det spectr\_cell2*

*de eg1*

*dc cell2d*

В данном входном файле задаются:

* Параметры ячеек: 1, 2, 3
* Решетки
* Поверхностей: 1, 2, 3, 4
* Параметры ячеек: 1, 2, 3, 4, 5, 1d, 2d, 3d
* Описание материалов: fuel, termfuel, clad, tube, water
* Подключаемые библиотеки
* Внешние условия
* Популяция нейтронов и количество циклов
* Параметры, задающие генерацию групповых констант
* Параметры, задающие построение спектра

Для того, чтобы поставить задачу на расчет, используется команда sbatch task.sh, где task.sh – текстовый файл, описывающий задачу и ресурсы, которые необходимо выделить для ее решения. Содержание файла приведено ниже:

*#!/bin/bash*

*#SBATCH --nodes=1*

*#SBATCH --tasks-per-node=1*

*#SBATCH --cpus-per-task=20*

*#SBATCH --time=1000:00:00*

*/home/SHARED/Serpent/sss2\_32 -omp 20 input*

Последняя строка – команда, которая будет исполняться на кластере, здесь:

* /home/SHARED/Serpent/sss2\_32 – путь к исполняемому файлу Serpent
* -omp 20 – параметр параллелизации
* input – название входного файла

Данная задача будет считаться на 20 ядрах. Кластер состоит из четырех нод по 40 ядер каждый. Пользователи ограниченны одним нодом и, соответственно, 40 ядрами.  
Для изменения числа ядер, выделяемых на расчет необходимо изменить число выделяемых ядер в строке #SBATCH --cpus-per-task=20 и число потоков в параметре -omp Serpent.

Таким образом, для запуска расчета нужно:

* Подготовить входной файл
* Создать в папке с входным фалом файл task.sh (может называться иначе)
* Запустить расчет командой sbatch task.sh из папки, где расположен task.sh
* Файлы с результатами расчета будут расположены в папке с входным файлом

# **4.2 Реализация базы данных для хранения считанных и предсказанных данных**

База данных была поднята на личном компьютере. Используемая база данных – Postgres.

Для создания таблиц были написаны SQL команды с описанием названием таблицы, типами полей, значениями по умолчанию.

Созданы следующие таблицы:

* uir.data
* uir.predicts
* public.webapp\_pins
* public.webapp\_temperature
* public.webapp\_zones

Написаны следующие DDL описания:

**CREATE** **TABLE** uir."data" (

nuclide **int4** **NOT** **NULL**,

temperature **int4** **NOT** **NULL**,

reaction\_102\_fast **float8** **NULL**,

reaction\_103\_fast **float8** **NULL**,

reaction\_102\_warm **float8** **NULL**,

reaction\_103\_warm **float8** **NULL**,

run\_time **timestamp** **NOT** **NULL** **DEFAULT** **clock\_timestamp**(),

**CONSTRAINT** data\_pkey **PRIMARY** **KEY** (nuclide, temperature)

);

**CREATE** **TABLE** uir.predicts (

nuclide **int4** **NULL**,

temperature **int4** **NULL**,

reaction\_102\_fast **float8** **NULL**,

reaction\_103\_fast **float8** **NULL**,

reaction\_102\_warm **float8** **NULL**,

reaction\_103\_warm **float8** **NULL**,

metric\_value\_102\_fast **float8** **NULL**,

metric\_value\_103\_fast **float8** **NULL**,

metric\_value\_102\_warm **float8** **NULL**,

metric\_value\_103\_warm **float8** **NULL**,

run\_time **timestamp** **NOT** **NULL** **DEFAULT** **clock\_timestamp**()

);

**CREATE** **TABLE** public.webapp\_pins (

id **bigserial** **NOT** **NULL**,

pin1\_void1 **float8** **NOT** **NULL**,

pin1\_fuel **float8** **NOT** **NULL**,

pin1\_void2 **float8** **NOT** **NULL**,

pin1\_clad **float8** **NOT** **NULL**,

pin2\_termfuel **float8** **NOT** **NULL**,

pin2\_clad **float8** **NOT** **NULL**,

**CONSTRAINT** webapp\_pins\_pkey **PRIMARY** **KEY** (id)

);

**CREATE** **TABLE** public.webapp\_temperature (

id **bigserial** **NOT** **NULL**,

fuel **int4** **NOT** **NULL**,

termfuel **int4** **NOT** **NULL**,

clad **int4** **NOT** **NULL**,

tube **int4** **NOT** **NULL**,

**CONSTRAINT** webapp\_temperature\_pkey **PRIMARY** **KEY** (id)

);

**CREATE** **TABLE** public.webapp\_zones (

id **bigserial** **NOT** **NULL**,

zone1\_x **float8** **NOT** **NULL**,

zone1\_y **float8** **NOT** **NULL**,

zone1\_d **float8** **NOT** **NULL**,

zone2\_x **float8** **NOT** **NULL**,

zone2\_y **float8** **NOT** **NULL**,

zone2\_d **float8** **NOT** **NULL**,

zone3\_x **float8** **NOT** **NULL**,

zone3\_y **float8** **NOT** **NULL**,

zone3\_d **float8** **NOT** **NULL**,

zone4\_x **float8** **NOT** **NULL**,

zone4\_y **float8** **NOT** **NULL**,

zone4\_d **float8** NOT **NULL**,

**CONSTRAINT** webapp\_zones\_pkey **PRIMARY** **KEY** (id)

);

Также, были созданы функции на языке python3, с помощью которых происходит создание таблиц, удаление таблиц, вставка данных в базу данных, запрос к базе, чтобы получить данные.Изображение выглядит как текст, экран

Автоматически созданное описание

Рисунок 9. model\_code/python/create\_tables\_module.py

Изображение выглядит как текст, экран, снимок экрана

Автоматически созданное описание

Рисунок 10. model\_code/python/drop\_tables\_module.py

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11. model\_code/python/get\_data\_module.py

Изображение выглядит как текст

Автоматически созданное описание Изображение выглядит как текст

Автоматически созданное описание Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12. Модуль по получению данных из базы данных

SQL скрипты по запросы данных из базы данных:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 13. Модуль по получению данных из таблицы webapp\_pins из базы данных

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 14. Модуль по получению данных из таблицы webapp\_temperature из базы данных

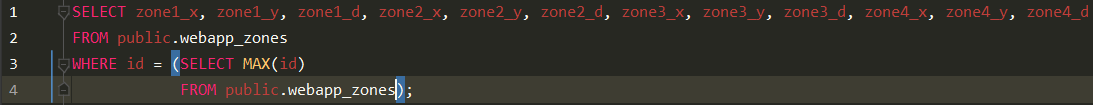


Рисунок 15. Модуль по получению данных из таблицы webapp\_zones из базы данных

# **4.3 Реализация базовых моделей машинного обучения**

В модуле ***data*** содержатся следующие собранные тренировочные данные:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 16. Тренировочные данные для модели

В модуле **model\_code/settings.yml** содержатся настройки, пути к тем или иным данным.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 17. model\_code/settings.yml

В модуле **model\_code/pstgrs.yml** содержатся настройки для подключения к базе данных.

Изображение выглядит как текст, стена, снимок экрана

Автоматически созданное описание

Рисунок 18. model\_code/pstgrs.yml

Данные вынесены в файлы формата YAML, чтобы избежать дублирования хардкода и данные, необходимых для подключения к базам данных.

В данном программном комплексе реализовано логирование состояния системы с уровнями:

* DEBUG
* INFO
* WARN
* ERROR

Реализовано с помощью библиотеки *argparse*.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 19. model\_code/python/logging\_module.py

Логирование ведется в отдельный файл: log\_file.log

Изображение выглядит как текст, электроника, клавиатура, компьютер

Автоматически созданное описание

Рисунок 20. model\_code/logs\_files/log\_file.log

Также, в данном программном комплексе реализован парсинг аргументов командной строки, через которую можно задать данные:

* Нуклид, для которого нужно сделать предсказание
* Температуру, при которой интересует будет найдено микросечение
* Метрика, по которой будет оцениваться качество модели

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 21. Пример аргументов командной строки

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 22. Код по созданию парсера командной строки

Также, в программном комплексе присутствует модуль, по созданию входного файла из данных, введенных пользователем.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 23. Код по созданию входного файла из введенных данных

Также, в программном комплексе присутствуют модули:

* main.py
* utils.py
* graphics.py
* machine\_learning\_module.py

Но их код будет приложен в приложении в силу объемности. Описание функций в каждом из модулей приведено ниже:

Таблица 6. Описание модулей конвейера машинного обучения

|  |  |  |
| --- | --- | --- |
| **Название модуля** | **Название функции** | **Описание** |
| graphics.py | def choose\_data\_nuclide | Отбирает из базы данных записи с конкретным полем nuclide |
| def choose\_predicted\_data | Отбирает из базы данных записи с конкретным полем nuclide |
| def plot | Функция для построения графика |
| def create\_plot | Более общая функция для построения графика, использует предыдущую |
| utils.py | def get\_yaml\_conf | Устанавливает yaml конфигурации |
| def timing | Создание функции обертки – декоратор |
| class Configuration | Класс Синглтон для создания единого экземпляра файла настроек |
| def postgres\_connection | Функция для установления соединения с базой данных |
| def load\_from\_postgres\_filename | Запрос данных по SQL скрипту в файле |
| def load\_from\_postgres\_query | Запрос данных по SQL скрипту |
| def load\_to\_postgres | Загрузить данные в базу данных |
| def is\_table\_exists | Проверка существования таблицы |
| machine\_learning\_module.py | def choose\_data\_nuclide | Отбирает из базы данных записи с конкретным полем nuclide |
| def find\_best\_params | Поиск лучших параметров с помощью GridSearch |
| def train\_best\_model | Обучение модели с лучшими параметрами |
| def predict | Создание предсказание |
| def process\_find\_train\_predict | Обертка функций find\_best\_params, train\_best\_model, predict |
| def machine\_learning\_model\_train | Обертка всех функций в модуле |
| main.py | class MainModel | Класс модели – создает экземпляр конвейера машинного обучения |
| def create\_tables | Функция создания таблиц в базе данных |
| def drop\_tables | Функция удаления таблиц из базы данных |
| def check\_exists\_tables | Проверка существования таблиц |
| def add\_train\_data | Функция сбора тренировочных данных |
| def run\_prediction | Функция запуска выдачи предсказания |

# **4.4 Реализация Web-интерфейса**

Для реализации веб-приложения понадобилось написать на языке python3 и с помощью фреймворка Django формы, модели, представления. Также, для отображения страниц, нужно написать код на html и css. Написанный код является объемным, поэтому вынесен в приложение. В данном разделе будет продемонстрировано само веб-приложение.

Изображение выглядит как текст

Автоматически созданное описание

…

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 24. Демонстрация Home page

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 25. Демонстрация Get Temperatures

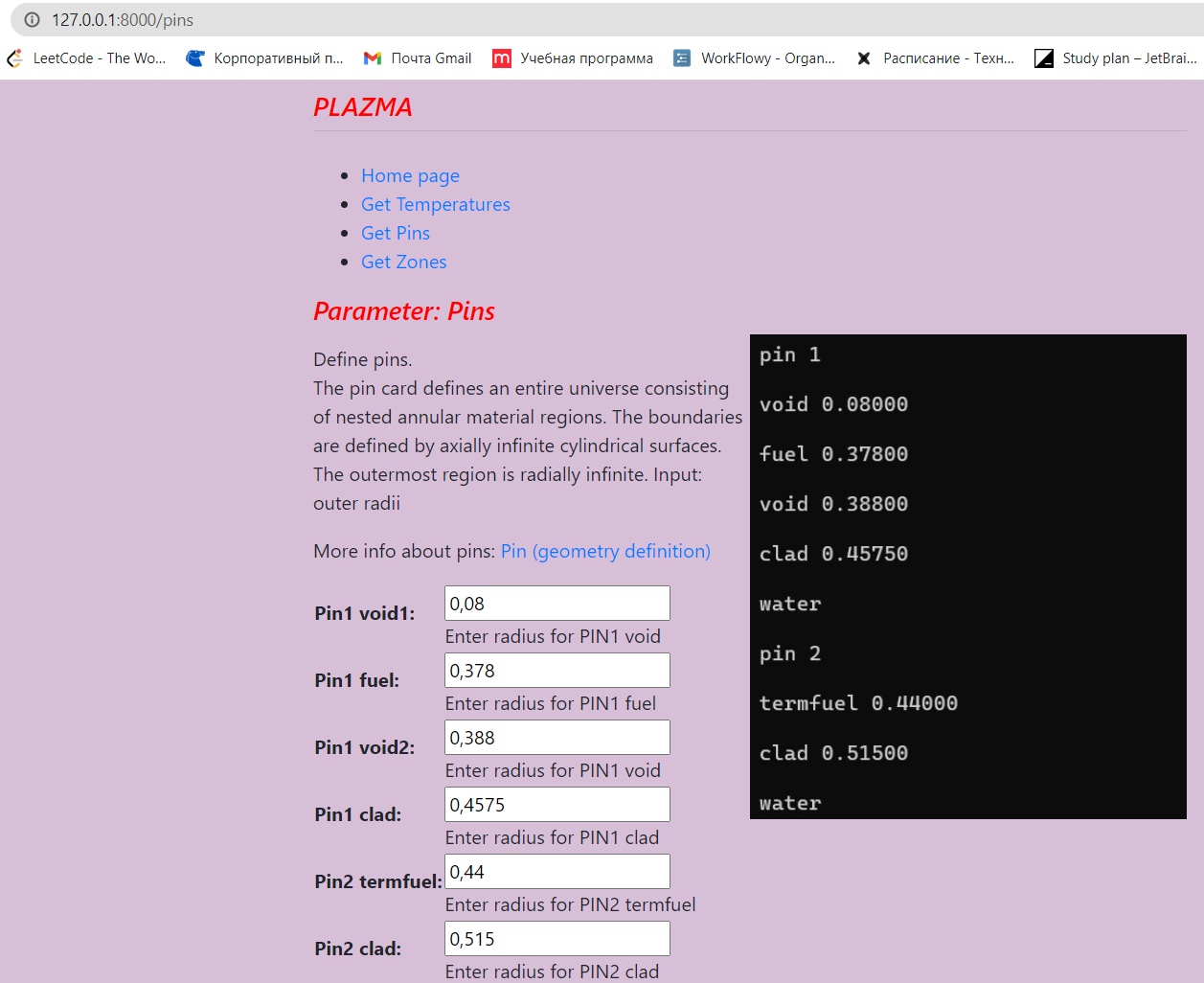


Рисунок 26. Демонстрация Get Pins

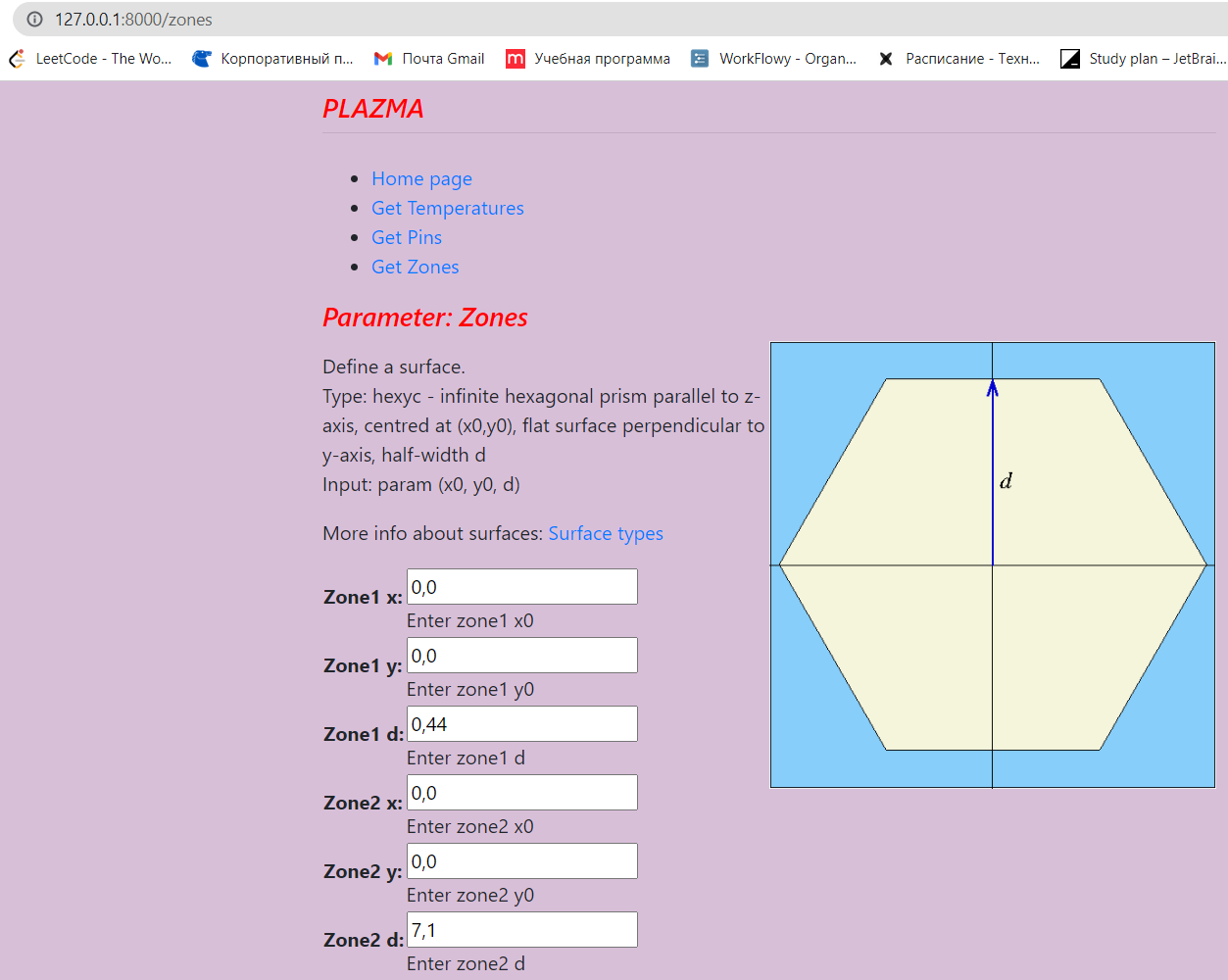


Рисунок 27. Демонстрация Get Zones

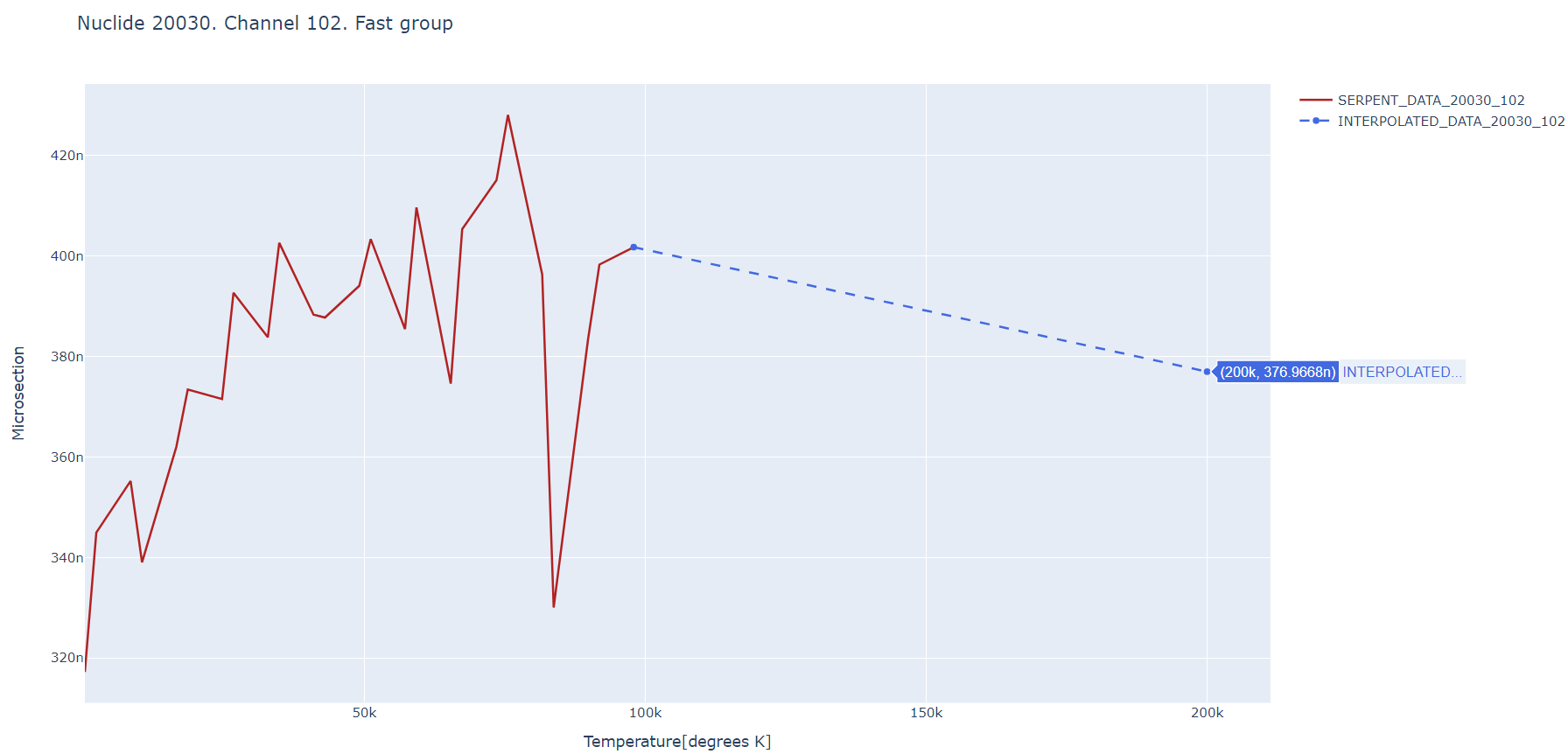


Рисунок 28. Демонстрация работы комплекса “ПЛАЗМА” для He-3 микросечение захвата (z, ) – реакция. Быстрая группа

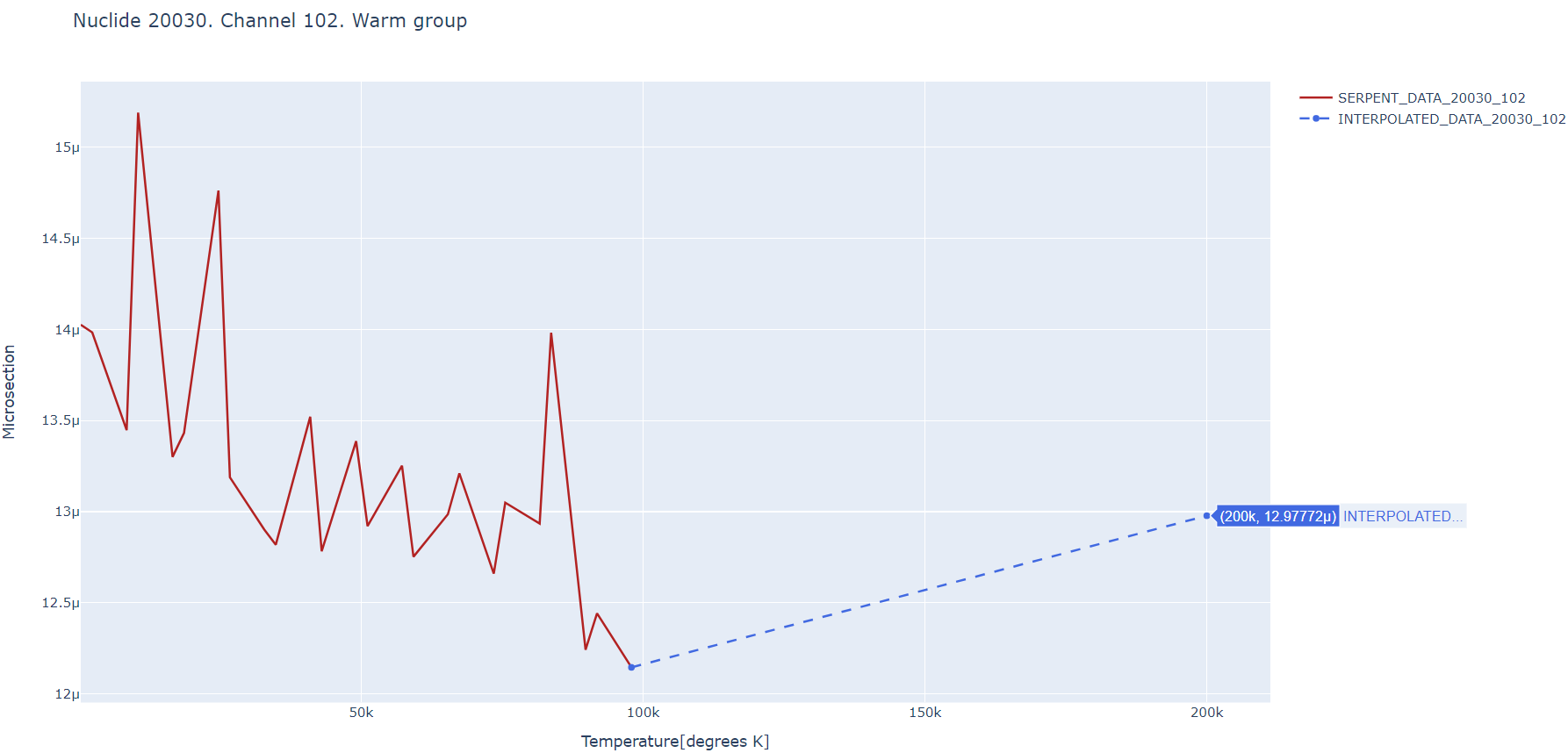


Рисунок 29. Демонстрация работы комплекса “ПЛАЗМА” для He-3 микросечение захвата (z, ) – реакция. Теплая группа

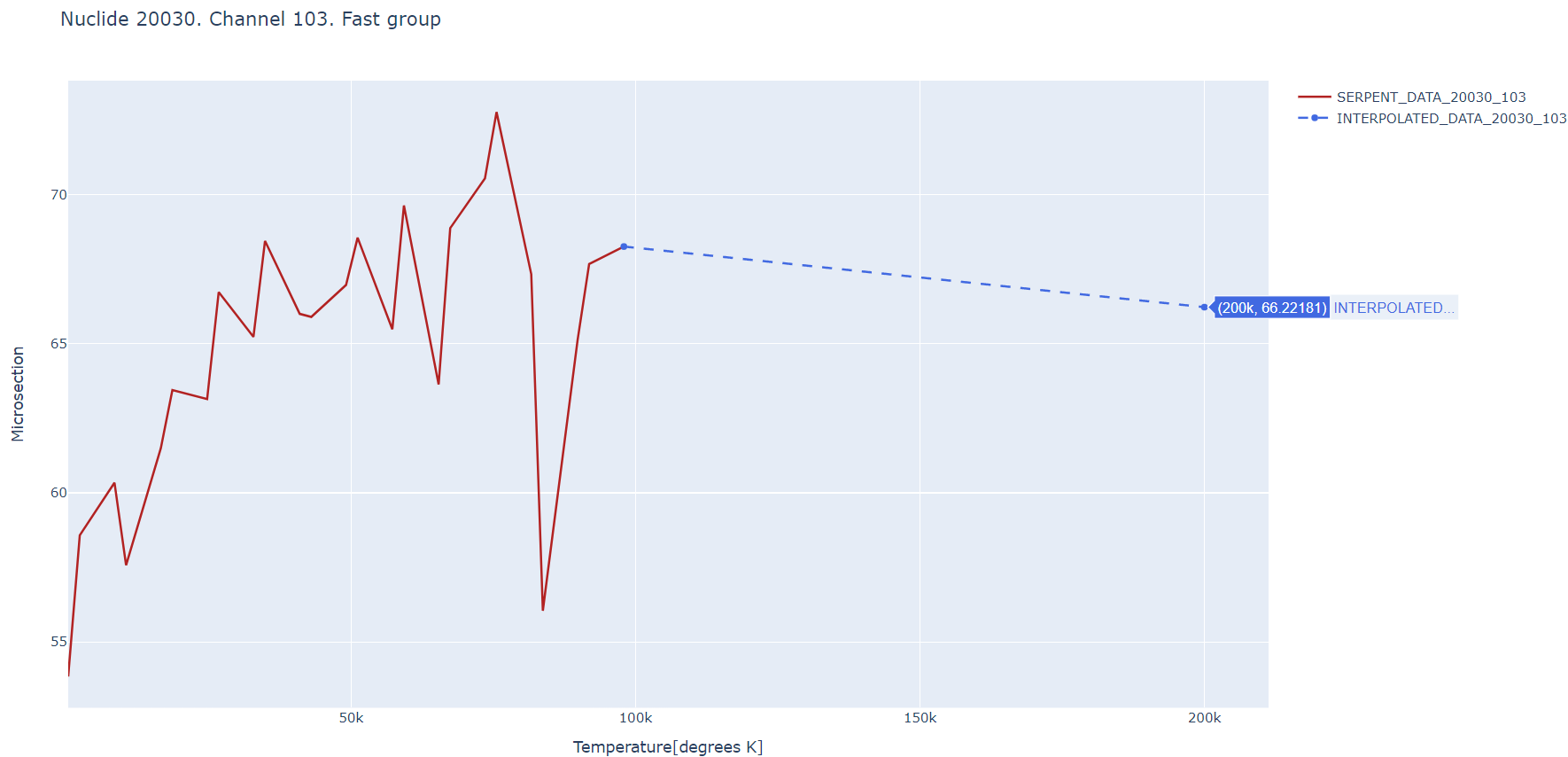


Рисунок 30. Демонстрация работы комплекса “ПЛАЗМА” для He-3 создание протона (z, p) – реакция. Быстрая группа

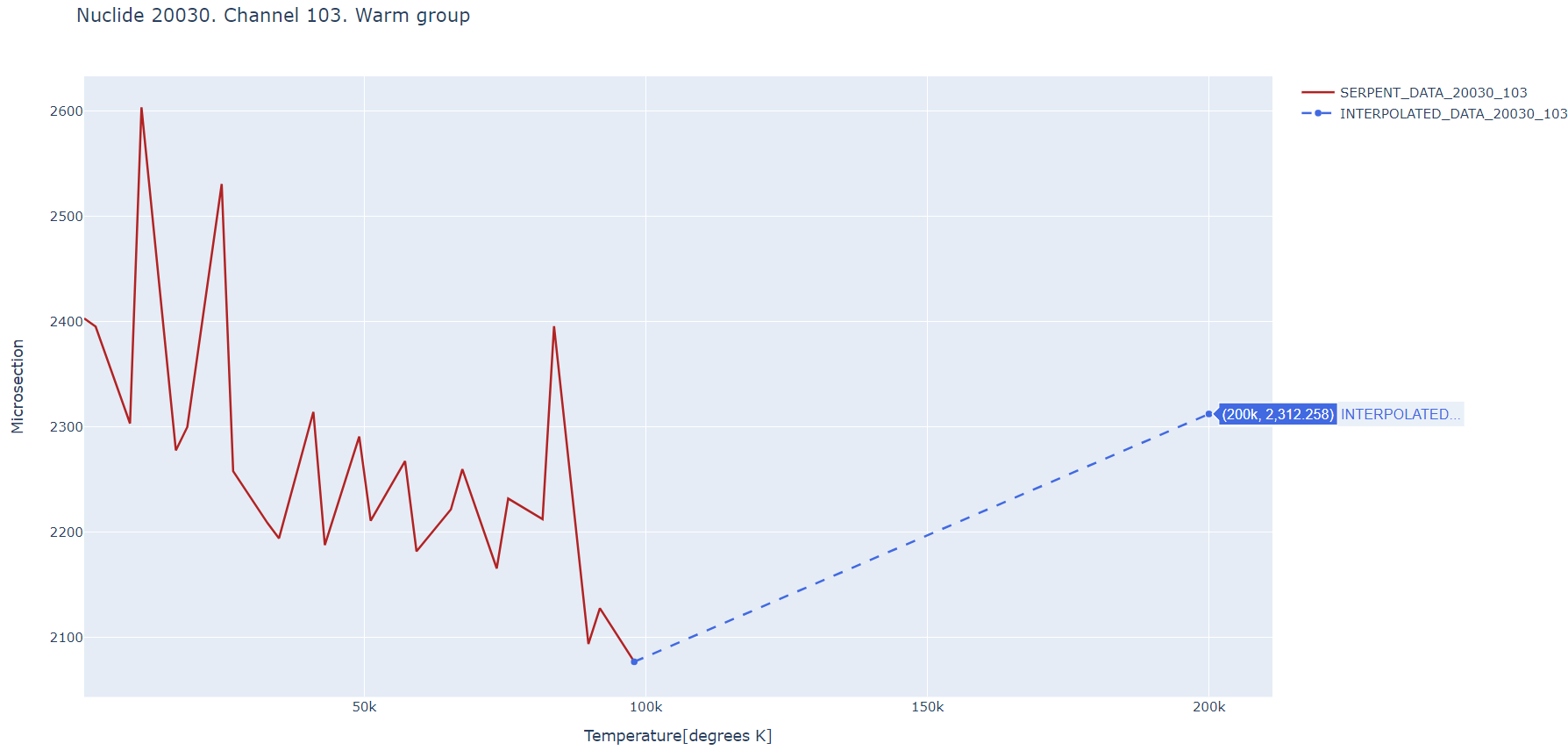


Рисунок 31. Демонстрация работы комплекса “ПЛАЗМА” для He-3 создание протона (z, p) – реакция. Теплая группа

# **Заключение**

В данной учебно-исследовательской работе был разработан программный комплекс “ПЛАЗМА”. С физической стороны, программный комплекс предназначен для оценки констант взаимодействия нуклидов плазмы с нейтронами бланкета. С программной стороны, комплекс расширяет возможности программного комплекса SERPENT 2 по подготовке нейтронно-физических данных для плазменного состояние вещества (с температурами большими программного предела - ).

Программный комплекс “ПЛАЗМА” позволяет сохранять тренировочные данные и предсказанные значения в базу данных, представляет пользователю интерфейс для задания параметров входного файла и интерактивные графики для просмотра полученных графиков.

С помощью разработанного программного комплекса “ПЛАЗМА” выполнена серия пробных расчетов, показавших его удовлетворительную работоспособность.

В дальнейшем планируется улучшение результатов предсказания констант взаимодействия нуклидов плазмы за счет использования программного комплекса NJOY для генерации библиотек данных. Также, планируется улучшение интерфейса для пользователей за счет более глубокого изучения фреймворка Django, html и css.

# **Список литературы**

1. *Б.А. Надыкто, Л.Ф. Тимофеева* Плутоний фундаментальные проблемы // РФЯЦ-ВНИИЭФ
2. *Соболь И.М.*, Метод Монте-Карло // Наука
3. Rand Corporation. A million random digits with 1 000 000 normal deviats – Glencoe: The Free Press, 1955.
4. *Савелова Т.И.* Метод Монте-Карло: Учебное пособие. М.: НИЯУ МИФИ, 2011. – 152 с.
5. *Гуревич М.И., Шкаровский Д.А.* Расчет переноса нейтронов методом Монте-Карло по программе MCU: Учебное пособие. М.: НИЯУ МИФИ, 2012. – 156 с.
6. Serpent a Continuous-energy Monte Carlo Reactor Physics Burnup Calculation Code. URL: <http://montecarlo.vtt.fi/index.htm>
7. Evaluated Nuclear Data File (ENDF). URL: <https://www-nds.iaea.org/exfor/endf.htm>
8. Nuclear Science. A VVER-1000 LEU and MOX Assembly Computational Benchmark. Specification and Results NEA/NSC/DOC(2002)10.
9. serpentTools URL: <https://serpent-tools.readthedocs.io/en/master/install.html>
10. **Высокопроизводительный вычислительный центр НИЯУ МИФИ.** URL: <https://it.mephi.ru/hpc>
11. Matplotlib. Release: 3.4.2. Date: May 08, 2021. URL: <https://matplotlib.org/>
12. Oracle. What is a relational database? URL: <https://www.oracle.com/ru/database/what-is-a-relational-database/>
13. *Шмелев А.Н., Куликов Г.Г.* О физических условиях для возникновения управляемой цепной реакции // Известия вузов: ядерная энергетика: №4 2014.
14. *Готт Ю.*В., Курнаев В.А. На пути к энергетике будущего: Учебное пособие. М.:НИЯУ МИФИ, 2017. -292 с.
15. *Куликов Г.Г., Шмелев А.Н., Апсэ В.А., Куликов Е.Г.* Потенциальная роль термоядерного нейтронного источника в ядерных энергетических системах. UDC 621.039.54; 621.039.6
16. machinelearning wiki. Регрессионный анализ: URL: <http://www.machinelearning.ru/wiki/index.php?title=%D0%A0%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D1%8F>
17. *Максимова Т.Г., Попова И.Н.* Эконометрика: учебно-методическое пособие / Т.Г. Максимова, И.Н. Попова. – СПб.: Университет ИТМО, 2018. – 70 с.

# **Приложение**

**model\_code/python/argparse\_module.py**

*import* os  
*from* argparse *import* ArgumentParser, ArgumentDefaultsHelpFormatter  
  
*from* logging\_module *import* get\_logger  
  
logger = get\_logger("")  
  
  
*def* setup\_parser(*parser*):  
 subparsers = *parser*.add\_subparsers(help="choose command")  
  
 full\_parser = subparsers.add\_parser(  
 "full\_parser",  
 help="full python code parser",  
 formatter\_class=ArgumentDefaultsHelpFormatter,  
 )  
  
 full\_parser.add\_argument(  
 "-n", "--nuclide",  
 help="nuclide number need to start machine learning model",  
 required=*True*,  
 dest="nuclide",  
 type=int,  
 )  
  
 full\_parser.add\_argument(  
 "-t", "--temperature",  
 help="temperature need to start machine learning model",  
 required=*True*,  
 dest="temperature",  
 type=int,  
 )  
  
 full\_parser.add\_argument(  
 "-m", "--metric",  
 help="metric of model",  
 dest="metric",  
 type=str,  
 default="neg\_mean\_squared\_error",  
 )  
  
  
*def* parser():  
 parser = ArgumentParser(  
 description="parse arguments of model PLASMA",  
 formatter\_class=ArgumentDefaultsHelpFormatter,  
 )  
 setup\_parser(parser)  
 arguments = parser.parse\_args()  
 *return* arguments

**model\_code/python/create\_tables\_module.py**

*from* psycopg2 *import* OperationalError  
  
*from* utils *import* get\_postgres\_connection  
*from* logging\_module *import* get\_logger  
  
logger = get\_logger("create tables")  
  
  
*def* create\_table(*script\_filename*: str):  
 con = get\_postgres\_connection()  
 con.autocommit = *True* cursor = con.cursor()  
 *try*:  
 cursor.execute(open(*script\_filename*, "r").read())  
 logger.info("{} table create".format(*script\_filename*))  
 *except* OperationalError *as* e:  
 logger.exception("Something gone wrong", e)

**model\_code/python/drop\_tables\_module.py**

*from* psycopg2 *import* OperationalError  
  
*from* utils *import* get\_postgres\_connection  
*from* logging\_module *import* get\_logger  
  
logger = get\_logger("drop tables")  
  
  
*def* drop\_table(*script\_filename*: str):  
 con = get\_postgres\_connection()  
 con.autocommit = *True* cursor = con.cursor()  
 *try*:  
 cursor.execute(open(*script\_filename*, "r").read())  
 logger.info("{} table drop".format(*script\_filename*))  
 *except* OperationalError *as* e:  
 logger.exception("Something gone wrong", e)

**model\_code/python/get\_data\_module.py**

*import* pandas *as* pd  
  
*from* logging\_module *import* get\_logger  
*from* utils *import* (  
 load\_from\_postgres\_query,  
 is\_table\_exists,  
)  
  
logger = get\_logger("get data")  
  
*def* get\_data\_function(*nuclide*: int, *tablename*: str):  
 *if* is\_table\_exists(*tablename*):  
 query = f"""  
 SELECT \*  
 FROM {*tablename*}  
 WHERE 'nuclide' = {*nuclide*}  
 """  
 ans = load\_from\_postgres\_query(query)  
 *else*:  
 *return None  
 return* ans  
  
  
*def* get\_test\_function(*nuclide*: int, *temperature*: int):  
 test\_dict = {  
 "nuclide": [*nuclide*],  
 "temperature": [*temperature*]  
 }  
 *return* pd.DataFrame(test\_dict)

**model\_code/python/graphics.py**

*import* os  
  
*import* plotly.graph\_objects *as* go  
*import* pandas *as* pd  
  
*from* logging\_module *import* get\_logger  
*from* utils *import* (  
 Configuration,  
 load\_from\_postgres\_query,  
)  
  
os.environ['CONFIG\_PATH'] = r"C:\Users\pavel\PycharmProjects\UIR\model\_code\settings\settings.yml"  
conf = Configuration()  
HTML\_GRAPHICS\_PATH = conf["PATHS"]["HTML\_GRAPHICS\_PATH"]  
POSTGRES\_TABLENAMES\_predicts = conf["TABLENAMES\_POSTGRES"]['predicts']  
  
  
logger = get\_logger("create Graphics")  
  
  
*def* choose\_data\_nuclide(*nuclide*: int, *reaction*: str, *data*: pd.DataFrame):  
 data\_nuclide = *data*[*data*['nuclide'] == *nuclide*]  
 X\_nuclide, y\_nuclide = data\_nuclide[['nuclide', 'temperature']], data\_nuclide[*reaction*]  
  
 *return* X\_nuclide['temperature'], y\_nuclide  
  
  
*def* choose\_predicted\_data(*nuclide*: int):  
 query = """  
 SELECT \*  
 FROM {}  
 WHERE nuclide = {} AND run\_time IN (SELECT MAX(run\_time)  
 FROM {}  
 WHERE nuclide = {})  
 """.format(POSTGRES\_TABLENAMES\_predicts ,*nuclide*, POSTGRES\_TABLENAMES\_predicts ,*nuclide*)  
  
 *return* load\_from\_postgres\_query(query)  
  
  
*def* plot(*temperatures*: list, *values*: list, *nuclide*: int, *reaction*: int, *temperature\_interpolate*: int, *value\_interpolate*: int, *group*: str):  
 fig = go.Figure()  
 # Create and style traces  
 fig.add\_trace(go.Scatter(x=*temperatures*,  
 y=*values*,  
 name='SERPENT\_DATA\_{}\_{}'.format(*nuclide*, *reaction*),  
 line=dict(color='firebrick',  
 width=2)  
 ))  
  
 fig.add\_trace(go.Scatter(x=[*temperatures*[-1], *temperature\_interpolate*],  
 y=[*values*[-1], *value\_interpolate*],  
 name='INTERPOLATED\_DATA\_{}\_{}'.format(*nuclide*, *reaction*),  
 line = dict(color='royalblue',  
 width=2,  
 dash='dash')  
 ))  
  
  
 # Edit the layout  
 fig.update\_layout(title='Nuclide {}. Channel {}. {} group'.format(*nuclide*, *reaction*, *group*),  
 xaxis\_title='Temperature[degrees K]',  
 yaxis\_title='Microsection')  
  
 fig.write\_html(HTML\_GRAPHICS\_PATH + "\\" +"{}\_{}\_{}.html".format(*nuclide*, *reaction*, *group*))  
  
  
*def* create\_plot(*data*, *nuclide*):  
 # reaction\_102\_fast  
 X\_102f\_t, y\_102f = choose\_data\_nuclide(*nuclide*, 'reaction\_102\_fast', *data*)  
 # reaction\_103\_fast  
 X\_103f\_t, y\_103f = choose\_data\_nuclide(*nuclide*, 'reaction\_103\_fast', *data*)  
 # reaction\_102\_warm  
 X\_102w\_t, y\_102w = choose\_data\_nuclide(*nuclide*, 'reaction\_102\_warm', *data*)  
 # reaction\_103\_warm  
 X\_103w\_t, y\_103w = choose\_data\_nuclide(*nuclide*, 'reaction\_103\_warm', *data*)  
  
 predicted\_df = choose\_predicted\_data(*nuclide*)  
  
 print(predicted\_df['temperature'].values[0])  
  
 plot(X\_102f\_t.values, y\_102f.values, *nuclide*, 102, predicted\_df['temperature'].values[0], predicted\_df['reaction\_102\_fast'].values[0],  
 'Fast')  
 plot(X\_103f\_t.values, y\_103f.values, *nuclide*, 103, predicted\_df['temperature'].values[0], predicted\_df['reaction\_103\_fast'].values[0],  
 'Fast')  
 plot(X\_102w\_t.values, y\_102w.values, *nuclide*, 102, predicted\_df['temperature'].values[0], predicted\_df['reaction\_102\_warm'].values[0],  
 'Warm')  
 plot(X\_103w\_t.values, y\_103w.values, *nuclide*, 103, predicted\_df['temperature'].values[0], predicted\_df['reaction\_103\_warm'].values[0],  
 'Warm')

**model\_code/python/logging\_module.py**

*import* logging  
  
FILENAME\_TO\_LOG = r"C:\Users\pavel\PycharmProjects\UIR\model\_code\logs\_files\log\_file.log"  
  
  
*def* get\_file\_handler():  
 simple\_formatter = logging.Formatter(  
 fmt="%(asctime)s - %(name)s - %(levelname)s - %(message)s",  
 datefmt="%Y-%m-%d %H:%M:%S"  
 )  
 file\_handler = logging.FileHandler(  
 filename=FILENAME\_TO\_LOG  
 )  
 file\_handler.setLevel(logging.INFO)  
 file\_handler.setFormatter(simple\_formatter)  
 *return* file\_handler  
  
  
*def* get\_logger(*name*):  
 logger = logging.getLogger(*name*)  
 logger.setLevel(logging.INFO)  
 logger.addHandler(get\_file\_handler())  
 *return* logger

**model\_code/python/machine\_learning\_module.py**

*import* os  
  
*import* pandas *as* pd  
*from* sklearn.preprocessing *import* StandardScaler  
*from* sklearn.pipeline *import* Pipeline  
*from* sklearn.linear\_model *import* LinearRegression  
*from* sklearn.preprocessing *import* PolynomialFeatures  
*from* sklearn.model\_selection *import* GridSearchCV  
*from* sklearn.model\_selection *import* train\_test\_split  
  
*from* logging\_module *import* get\_logger  
*from* utils *import* (  
 Configuration,  
 get\_postgres\_connection,  
 load\_to\_postgres,  
)  
  
FILENAME\_SCRIPT = ""  
  
logger = get\_logger("machine learning model")  
os.environ['CONFIG\_PATH'] = r"C:\Users\pavel\PycharmProjects\UIR\model\_code\settings\settings.yml"  
conf = Configuration()  
POSTGRES\_TABLENAMES = conf["TABLENAMES\_POSTGRES"]  
  
  
*def* choose\_data\_nuclide(*nuclide*: int, *reaction*: str, *data*: pd.DataFrame):  
 data\_nuclide = *data*[*data*['nuclide'] == *nuclide*]  
 X\_nuclide, y\_nuclide = data\_nuclide[['nuclide', 'temperature']], data\_nuclide[*reaction*]  
  
 X\_nuclide\_new = pd.DataFrame()  
 X\_nuclide\_new['nuclide'] = X\_nuclide['nuclide'].astype(str)  
 X\_nuclide\_new['temperature'] = X\_nuclide['temperature']  
 X\_nuclide\_new = pd.get\_dummies(X\_nuclide\_new)  
 *return* X\_nuclide\_new, y\_nuclide  
  
  
*def* find\_best\_params(*pipe*, *metric*: str, *reaction*: str, *nuclide*: int, *X\_train*, *y\_train*):  
 poly\_degrees = [1, 2, 3, 4, 5, 6, 7]  
 param\_grid = [{'poly\_\_degree': poly\_degrees}]  
 gs = GridSearchCV(estimator=*pipe*,  
 param\_grid=param\_grid,  
 scoring=*metric*,  
 cv=10,  
 n\_jobs=-1)  
 gs = gs.fit(*X\_train*, *y\_train*)  
 logger.info("Best score of METRIC {} for model: NUCLIDE {}, REACTION {}: {}".format(*metric*, *nuclide*, *reaction*, -gs.best\_score\_))  
  
 *return* gs.best\_estimator\_, -gs.best\_score\_  
  
  
*def* train\_best\_model(*best\_estimator*, *X\_train*, *y\_train*):  
 clf = *best\_estimator* clf.fit(*X\_train*, *y\_train*)  
 *return* clf  
  
  
*def* predict(*model*, *X*):  
 *return model*.predict(*X*)  
  
  
*def* process\_find\_train\_predict(*pipe*, *metric*: str, *reaction*: str, *nuclide*: int, *X\_train*, *y\_train*, *X*):  
 best\_estimator, best\_score = find\_best\_params(*pipe*, *metric*, *reaction*, *nuclide*, *X\_train*, *y\_train*)  
 model = train\_best\_model(best\_estimator, *X\_train*, *y\_train*)  
 y\_pred = predict(model, *X*)  
 *return* y\_pred, best\_score  
  
  
*def* machine\_learning\_model\_main(*metric*, *X\_test*, *data*):  
 """  
 Make prediction and save it in Database  
  
 Parameters:  
 metric: str -> accuracy, f1, recall, precision...  
 X\_test: dict -> {'nuclide': , 'temperature': }  
 """  
 nuclide, temperature = *X\_test*['nuclide'][0], *X\_test*['temperature'][0]  
  
 pipe = Pipeline([('scaler', StandardScaler()),  
 ('poly', PolynomialFeatures()),  
 ('reg', LinearRegression()),  
 ])  
  
 # reaction\_102\_fast  
 X\_102f, y\_102f = choose\_data\_nuclide(nuclide, 'reaction\_102\_fast', *data*)  
 # reaction\_103\_fast  
 X\_103f, y\_103f = choose\_data\_nuclide(nuclide, 'reaction\_103\_fast', *data*)  
 # reaction\_102\_warm  
 X\_102w, y\_102w = choose\_data\_nuclide(nuclide, 'reaction\_102\_warm', *data*)  
 # reaction\_103\_warm  
 X\_103w, y\_103w = choose\_data\_nuclide(nuclide, 'reaction\_103\_warm', *data*)  
  
 X\_train\_102f, X\_test\_102f, y\_train\_102f, y\_test\_102f = train\_test\_split(X\_102f, y\_102f, test\_size=0.2)  
 X\_train\_103f, X\_test\_103f, y\_train\_103f, y\_test\_103f = train\_test\_split(X\_103f, y\_103f, test\_size=0.2)  
 X\_train\_102w, X\_test\_102w, y\_train\_102w, y\_test\_102w = train\_test\_split(X\_102w, y\_102w, test\_size=0.2)  
 X\_train\_103w, X\_test\_103w, y\_train\_103w, y\_test\_103w = train\_test\_split(X\_103w, y\_103w, test\_size=0.2)  
  
 y\_pred\_102f, score\_102f = process\_find\_train\_predict(pipe=pipe, metric=*metric*, reaction='102f', nuclide=nuclide,  
 X\_train=X\_train\_102f, y\_train=y\_train\_102f, X=X\_test\_102f)  
 y\_pred\_103f, score\_103f = process\_find\_train\_predict(pipe=pipe, metric=*metric*, reaction='103f', nuclide=nuclide,  
 X\_train=X\_train\_103f, y\_train=y\_train\_103f, X=X\_test\_103f)  
 y\_pred\_102w, score\_102w = process\_find\_train\_predict(pipe=pipe, metric=*metric*, reaction='102w', nuclide=nuclide,  
 X\_train=X\_train\_102w, y\_train=y\_train\_102w, X=X\_test\_102w)  
 y\_pred\_103w, score\_103w = process\_find\_train\_predict(pipe=pipe, metric=*metric*, reaction='103w', nuclide=nuclide,  
 X\_train=X\_train\_103w, y\_train=y\_train\_103w, X=X\_test\_103w)  
  
 predictions\_dict = {  
 "nuclide": nuclide,  
 "temperature": temperature,  
 "reaction\_102\_fast": y\_pred\_102f,  
 "reaction\_103\_fast": y\_pred\_103f,  
 "reaction\_102\_warm": y\_pred\_102w,  
 "reaction\_103\_warm": y\_pred\_103w,  
 "metric\_value\_102\_fast": score\_102f,  
 "metric\_value\_103\_fast": score\_103f,  
 "metric\_value\_102\_warm": score\_102w,  
 "metric\_value\_103\_warm": score\_103w  
 }  
  
 predictions\_df = pd.DataFrame(predictions\_dict)  
  
 print(y\_pred\_102f)  
  
 conn = get\_postgres\_connection()  
 load\_to\_postgres(conn, predictions\_df, POSTGRES\_TABLENAMES["predicts"])  
 conn.close()  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 *pass*

**model\_code/python/main.py**

*import* os  
  
*from* argparse\_module *import* parser  
*from* logging\_module *import* get\_logger  
*from* utils *import* (  
 Configuration,  
 load\_from\_postgres\_query,  
 is\_table\_exists,  
)  
*from* create\_tables\_module *import* create\_table  
*from* drop\_tables\_module *import* drop\_table  
*from* machine\_learning\_model *import* machine\_learning\_model\_main  
*from* get\_data\_module *import* get\_test\_function  
*from* set\_data\_module *import* set\_data\_function  
*from* graphics *import* create\_plot  
  
  
os.environ['CONFIG\_PATH'] = r"C:\Users\pavel\PycharmProjects\UIR\model\_code\settings\settings.yml"  
  
conf = Configuration()  
CREATE\_TABLE\_DATA\_SCRIPT = conf["PATHS"]["SCRIPTS\_PATH"]  
DROP\_TABLE\_DATA\_SCRIPT = conf["PATHS"]["SCRIPTS\_PATH"]  
CREATE\_TABLES\_SCRIPTS = ["create\_data.sql", "create\_predictions.sql"]  
DROP\_TABLES\_SCRIPTS = ["drop\_data.sql", "drop\_predictions.sql"]  
TABLES\_NAMES = [conf["TABLENAMES\_POSTGRES"]["data"], conf["TABLENAMES\_POSTGRES"]["predicts"]]  
  
arguments = parser()  
nuclide = arguments.nuclide  
metric = arguments.metric  
temperature = arguments.temperature  
  
logger = get\_logger("main")  
  
  
*class* MainModel():  
  
 @staticmethod  
 *def* create\_tables():  
 *for* table *in* CREATE\_TABLES\_SCRIPTS:  
 create\_table(CREATE\_TABLE\_DATA\_SCRIPT + "\\" + table)  
  
 @staticmethod  
 *def* drop\_tables():  
 *for* table *in* DROP\_TABLES\_SCRIPTS:  
 drop\_table(DROP\_TABLE\_DATA\_SCRIPT + "\\" + table)  
  
 *def* check\_exists\_tables(self, *tablenames*: list) -> bool:  
 *for* tablename *in tablenames*:  
 *if not* is\_table\_exists(tablename):  
 *return False  
 return True  
  
 def* add\_train\_data(self):  
 logger.info("add: start create tables")  
 self.create\_tables()  
 logger.info("add: finish create tables")  
  
 logger.info("add: start check exists tables")  
 *if not* self.check\_exists\_tables(TABLES\_NAMES):  
 logger.error(f"One of the table does not exist")  
 exit(1)  
 logger.info("add: finish check exists tables")  
  
 logger.info("add: start to add data to db")  
 set\_data\_function()  
 logger.info("add: finish to add data to db")  
  
 *def* run\_prediction(self):  
 logger.info("run: start create tables")  
 self.create\_tables()  
 logger.info("run: finish create tables")  
  
 logger.info("run: start check exists tables")  
 *if not* self.check\_exists\_tables(TABLES\_NAMES):  
 logger.error(f"One of the table does not exist")  
 exit(1)  
 logger.info("run: finish check exists tables")  
  
 logger.info(f"run: start load data from Database for nuclide: {nuclide}")  
 query = f"""SELECT \*   
 FROM {TABLES\_NAMES[0]}  
 """  
 data = load\_from\_postgres\_query(query)  
 logger.info(f"run: finish load data from Database for nuclide {nuclide}")  
  
 logger.info(f"run: start create test data for nuclide {nuclide} and temperature {temperature}")  
 X\_test = get\_test\_function(nuclide, temperature)  
 logger.info(f"run: finish create test data for nuclide {nuclide} and temperature {temperature}")  
  
 logger.info(f"run: start machine learning model with metric {metric}")  
 machine\_learning\_model\_main(metric=metric, X\_test=X\_test, data=data)  
 logger.info(f"run: finish machine learning model with metric {metric}")  
  
 logger.info(f"run: start create graphics for nuclide {nuclide}")  
 create\_plot(data, nuclide)  
 logger.info(f"run: finish create graphics for nuclide {nuclide}")  
  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 mainModel = MainModel()  
  
 mainModel.add\_train\_data()  
 mainModel.run\_prediction()

**model\_code/python/set\_data\_module.py**

"""  
MODULE LOAD RESULTS (TRAIN FOR MODEL) DATA FROM SERPENT FILES  
"""  
*import* os  
*from* collections *import* namedtuple  
  
*import* serpentTools  
*import* pandas *as* pd  
*from* typing *import* Tuple  
*from* tqdm *import* tqdm  
  
*from* logging\_module *import* get\_logger  
*from* utils *import* (  
 Configuration,  
 load\_to\_postgres,  
 get\_postgres\_connection,  
)  
  
os.environ['CONFIG\_PATH'] = r"C:\Users\pavel\PycharmProjects\UIR\model\_code\settings\settings.yml"  
conf = Configuration()  
  
DATA\_PATH = conf["PATHS"]["DATA\_PATH"]  
TABLE\_NAME = conf["TABLENAMES\_POSTGRES"]["data"]  
TEMPERATURES = [300, 2334, 8438, 10473, 16577, 18612, 24716, 26751, 32855, 34889, 40993, 43028,  
 49132, 51167, 57271, 59306, 65410, 67444, 73548, 75583, 81687, 83722, 89826, 91861, 97964]  
NUCLIDES = [10020, 10030, 20030]  
REACTIONS = [102, 103]  
UNIVERSE = '101'  
logger = get\_logger("set data")  
  
  
*def* fast\_warm(*val*) -> Tuple[list, list]:  
 val\_f = [*val*[i][1] *for* i *in* range(len(*val*))]  
 val\_w = [*val*[i][2] *for* i *in* range(len(*val*))]  
 *return* val\_f, val\_w  
  
  
*def* set\_data(*nuclide*: int, *reaction*):  
 """values for different temperatures"""  
 nuclide\_val = []  
  
 files = os.listdir(DATA\_PATH)  
 *for* file *in* tqdm(files):  
 mdx = serpentTools.read(DATA\_PATH + "\\" + file)  
  
 vals, unc = mdx.getXS(universe=UNIVERSE, isotope=*nuclide*, reaction=*reaction*)  
 nuclide\_val.append(vals)  
  
 nuclide\_reaction\_f, nuclide\_reaction\_w = fast\_warm(nuclide\_val)  
 *return* nuclide\_reaction\_f, nuclide\_reaction\_w  
  
  
*def* data\_to\_db(*nuclide*: int):  
 DBtable = namedtuple('DBtable', ['nuclide',  
 'temperature',  
 'reaction',  
 'values\_102\_f',  
 'values\_102\_w',  
 'values\_103\_f',  
 'values\_103\_w'  
 ])  
 size = len(TEMPERATURES)  
 *return* DBtable(nuclide = [*nuclide*] \* size,  
 temperature = TEMPERATURES,  
 reaction = [102] \* size,  
 values\_102\_f = set\_data(*nuclide*, 102)[0],  
 values\_102\_w = set\_data(*nuclide*, 102)[1],  
 values\_103\_f = set\_data(*nuclide*, 103)[0],  
 values\_103\_w = set\_data(*nuclide*, 103)[1],  
 )  
  
  
*def* set\_data\_function():  
 LIST\_DATA = []  
 *for* nuclide *in* NUCLIDES:  
 LIST\_DATA.append(  
 data\_to\_db(nuclide)  
 )  
  
 d = {  
 "nuclide": [],  
 "temperature": [],  
 "reaction\_102\_fast": [],  
 "reaction\_103\_fast": [],  
 "reaction\_102\_warm": [],  
 "reaction\_103\_warm": []  
 }  
  
 *for* table *in* LIST\_DATA:  
 *for* i *in* range(len(TEMPERATURES)):  
 d['nuclide'].append(table.nuclide[i])  
 d['temperature'].append(table.temperature[i])  
 d['reaction\_102\_fast'].append(table.values\_102\_f[i])  
 d['reaction\_103\_fast'].append(table.values\_103\_f[i])  
 d['reaction\_102\_warm'].append(table.values\_102\_w[i])  
 d['reaction\_103\_warm'].append(table.values\_103\_w[i])  
  
 df = pd.DataFrame(d)  
 conn = get\_postgres\_connection()  
 load\_to\_postgres(conn, df, TABLE\_NAME)  
 conn.close()

**model\_code/python/template\_serpent\_file.py**

*import* os  
  
*from* utils *import* (  
 Configuration,  
 load\_from\_postgres\_filename,  
 get\_postgres\_connection,  
)  
*from* logging\_module *import* get\_logger  
  
os.environ['CONFIG\_PATH'] = r"C:\Users\pavel\PycharmProjects\UIR\model\_code\settings\settings.yml"  
conf = Configuration()  
TEMPLATE\_PATH = conf["PATHS"]["TEMPLATE\_PATH"]  
TEMPLATE\_FILE = 'template\_vver440'  
SELECT\_TABLE\_DATA\_SCRIPT = conf["PATHS"]["SCRIPTS\_PATH"]  
  
logger = get\_logger("create SERPENT file")  
  
*def* input\_data\_into\_template(*template\_file*: str):  
 conn = get\_postgres\_connection()  
 temperatures\_df = load\_from\_postgres\_filename(SELECT\_TABLE\_DATA\_SCRIPT + "\\" + "select\_temperatures.sql", conn)  
 pins\_df = load\_from\_postgres\_filename(SELECT\_TABLE\_DATA\_SCRIPT + "\\" + "select\_pins.sql", conn)  
 zones\_df = load\_from\_postgres\_filename(SELECT\_TABLE\_DATA\_SCRIPT + "\\" + "select\_zones.sql", conn)  
  
 *with* open(TEMPLATE\_PATH + "\\" + *template\_file*, "r", encoding='utf-8') *as* file:  
 filedata = file.read()  
  
 *for* df, name *in* ((temperatures\_df, 'temperatures'), (pins\_df, 'pins'), (zones\_df, 'zones')):  
 *for* col *in* df.columns:  
 col\_rep = col  
 *if* name == 'temperatures':  
 col\_rep = 'tmp\_' + col  
 filedata = filedata.replace(col\_rep, str(\*df[col].values))  
  
 *with* open(TEMPLATE\_PATH + "\\" + "replaced\_file", 'w') *as* file:  
 file.write(filedata)  
  
input\_data\_into\_template(TEMPLATE\_FILE)

**model\_code/python/utils.py**

*import* os  
*from* io *import* StringIO  
*from* time *import* time  
*from* functools *import* wraps  
  
*import* pandas *as* pd  
*import* yaml  
*import* psycopg2  
*from* psycopg2 *import* extras  
  
*from* logging\_module *import* get\_logger  
  
logger = get\_logger("utils logger")  
CURRENT\_DIR = ""  
  
  
*def* get\_yaml\_conf(*settings\_filename*):  
 *with* open(*settings\_filename*, "r") *as* file:  
 settings = yaml.load(file, Loader=yaml.FullLoader)  
 *return* settings  
  
  
*def* timing(*f*):  
 @wraps(f)  
 *def* wrap(*\*args*, *\*\*kw*):  
 ts = time()  
 result = f(\**args*, \*\**kw*)  
 te = time()  
 print("func: {} took: {} sec".format(f.\_\_name\_\_, round(te - ts)))  
 *return* result  
  
 *return* wrap  
  
  
*class* Configuration(object):  
 """  
 Класс - Синглтон  
 Он нужен для хранения всех настроек и доступа к ним из любой части пакета  
  
 Доступ к элементам осуществляется через []  
 Источники параметров - конфигурационный файл и аргументы командной строки. Они передаются словарями  
 Этому контейнеру можно задавать результаты и напрямую, через []  
 """  
  
 \_instance = *None* \_d = {}  
  
 *def \_\_new\_\_*(class\_, *\*args*, *\*\*kwargs*):  
 """Реализация синглтона"""  
 *if not* isinstance(class\_.\_instance, class\_):  
 Configuration.\_instance = object.\_\_new\_\_(Configuration, \**args*, \*\**kwargs*)  
 settings = get\_yaml\_conf(os.environ['CONFIG\_PATH'])  
 Configuration.\_instance.store\_dict(settings)  
 *if* "POSTGRES" *in* settings.keys():  
 *if* "CREDENTIALS\_PATH" *in* settings["POSTGRES"]:  
 account\_settings = get\_yaml\_conf(CURRENT\_DIR + settings["POSTGRES"]["CREDENTIALS\_PATH"])  
 Configuration.\_instance.\_d["POSTGRES"].update(account\_settings)  
 *return* class\_.\_instance  
  
 *def* store\_dict(self, *d*):  
 """  
 Сохранение параметров из словаря  
 Args:  
 d: dict  
 """  
 *for* key, value *in d*.items():  
 # Если значение уже есть, а новое None, то None записан не будет  
 *if not* (key *in* self.\_d *and* value *is None*):  
 self.\_d[key] = value  
  
 *def \_\_setitem\_\_*(self, *key*, *value*):  
 self.\_d[*key*] = *value  
  
 def \_\_getitem\_\_*(self, *key*):  
 *return* self.\_d[*key*]  
  
 *def \_\_contains\_\_*(self, *key*):  
 *return key in* self.\_d  
  
  
*def* get\_postgres\_connection():  
 """Return POSTGRES connection from settings.yml credentials"""  
 postgres\_conf = Configuration()["POSTGRES"]  
  
 hostname = postgres\_conf['host']  
 login = postgres\_conf['login']  
 database = postgres\_conf['database']  
 password = postgres\_conf['password']  
  
 logger.info(  
 "Postgres connection: HOSTNAME {}, LOGIN {}, DATABASE {}".format(hostname, login, database))  
  
 *try*:  
 conn = psycopg2.connect(  
 dbname=database,  
 user=login,  
 host=hostname,  
 password=password,  
 )  
 *except* Exception *as* e:  
 logger.exception("Something gone wrong", e)  
 *raise  
 return* conn  
  
  
@timing  
*def* load\_from\_postgres\_filename(*filename*, *conn*):  
 *with* open(*filename*, "r") *as* file:  
 sql = file.read()  
 ans = pd.read\_sql(sql, *conn*)  
 *return* ans  
  
  
@timing  
*def* load\_from\_postgres\_query(*query*):  
 conn = get\_postgres\_connection()  
 *try*:  
 ans = pd.read\_sql(*query*, conn)  
 *except* pd.io.sql.DatabaseError *as* e:  
 logger.exception(e)  
 *return* ans  
  
  
@timing  
*def* load\_to\_postgres(*conn*, *df*, *table*):  
 logger.info("start copy dataframe to Postgres {}".format(*table*))  
 buffer = StringIO()  
 *df*.to\_csv(buffer, header=*False*)  
 buffer.seek(0)  
  
 cur = *conn*.cursor()  
 *try*:  
 df\_columns = list(*df*)  
 # create (col1,col2,...)  
 columns = ",".join(df\_columns)  
  
 # create VALUES('%s', '%s",...) one '%s' per column  
 values = "VALUES({})".format(",".join(["%s" *for* \_ *in* df\_columns]))  
  
 # create INSERT INTO table (columns) VALUES('%s',...)  
 insert\_stmt = "INSERT INTO {} ({}) {}".format(*table*, columns, values)  
  
 cur = *conn*.cursor()  
 psycopg2.extras.execute\_batch(cur, insert\_stmt, *df*.values)  
 *conn*.commit()  
 cur.close()  
 *except* (Exception, psycopg2.DatabaseError) *as* e:  
 logger.exception("Something gone wrong", e)  
 *conn*.rollback()  
 cur.close()  
 logger.info("finish copy dataframe to Postgres {}".format(*table*))  
 cur.close()  
  
  
*def* is\_table\_exists(*tablename*: str):  
 """expected tablename with structure: schema.table"""  
 conn = get\_postgres\_connection()  
 query = f"""  
 SELECT \* FROM {*tablename*} LIMIT 1;  
 """  
 *try*:  
 pd.read\_sql(query, conn)  
 *except* pd.io.sql.DatabaseError *as* e:  
 logger.exception(e)  
 *return False  
 finally*:  
 conn.close()  
 *return True*

1. *N. Metropolis, S. Ulam,* The Monte Carlo method // J. Amer. Statistical assoc., 1949, 44, № 247, 335-341 [↑](#footnote-ref-1)
2. *И.М. Соболь*, Метод Монте-Карло // Наука [↑](#footnote-ref-2)
3. *Б.А. Надыкто, Л.Ф. Тимофеева* Плутоний фундаментальные проблемы // РФЯЦ-ВНИИЭФ [↑](#footnote-ref-3)
4. *И.М. Соболь*, Метод Монте-Карло // Наука [↑](#footnote-ref-4)
5. Rand Corporation. A million random digits with 1 000 000 normal deviats – Glencoe: The Free Press, 1955. [↑](#footnote-ref-5)
6. *Гуревич М.И., Шкаровский Д.А.* Расчет переноса нейтронов методом Монте-Карло по программе MCU: Учебное пособие. М.: НИЯУ МИФИ, 2012. – 156 с. [↑](#footnote-ref-6)
7. *Гуревич М.И., Шкаровский Д.А.* Расчет переноса нейтронов методом Монте-Карло по программе MCU: Учебное пособие. М.: НИЯУ МИФИ, 2012. – 156 с. [↑](#footnote-ref-7)
8. *Гуревич М.И., Шкаровский Д.А.* Расчет переноса нейтронов методом Монте-Карло по программе MCU: Учебное пособие. М.: НИЯУ МИФИ, 2012. – 156 с. [↑](#footnote-ref-8)
9. Serpent a Continuous-energy Monte Carlo Reactor Physics Burnup Calculation Code URL: <http://montecarlo.vtt.fi/index.htm> [↑](#footnote-ref-9)
10. *Шмелев А.Н., Куликов Г.Г.* О физических условиях для возникновения управляемой цепной реакции // Известия вузов: ядерная энергетика: №4 2014 [↑](#footnote-ref-10)
11. *Готт Ю.*В., Курнаев В.А. На пути к энергетике будущего: Учебное пособие. М.:НИЯУ МИФИ, 2017. -292 с. [↑](#footnote-ref-11)
12. *Куликов Г.Г., Шмелев А.Н., Апсэ В.А., Куликов Е.Г.* Потенциальная роль термоядерного нейтронного источника в ядерных энергетических системах. UDC 621.039.54; 621.039.6 [↑](#footnote-ref-12)
13. **Высокопроизводительный вычислительный центр НИЯУ МИФИ.** URL: <https://it.mephi.ru/hpc> [↑](#footnote-ref-13)
14. *Максимова Т.Г., Попова И.Н.* Эконометрика: учебно-методическое пособие / Т.Г. Максимова, И.Н. Попова. – СПб.: Университет ИТМО, 2018. – 70 с. [↑](#footnote-ref-14)
15. machinelearning wiki. Регрессионный анализ: URL: <http://www.machinelearning.ru/wiki/index.php?title=%D0%A0%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D1%8F> [↑](#footnote-ref-15)
16. Oracle. What is a relational database? URL: <https://www.oracle.com/ru/database/what-is-a-relational-database/> [↑](#footnote-ref-16)
17. Evaluated Nuclear Data File (ENDF). URL: <https://www-nds.iaea.org/exfor/endf.htm> [↑](#footnote-ref-17)
18. serpentTools URL: <https://serpent-tools.readthedocs.io/en/master/install.html> [↑](#footnote-ref-18)