**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук  
Департамент программной инженерии

|  |  |
| --- | --- |
| СОГЛАСОВАНО  доцент департамента математики  факультета экономических наук, кандидат физико-математических наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. Р. Горяинова «\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г. | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. А. Павлочев «\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г. |



# **Разработка программного комплекса для исследования влияния аномальных наблюдений на точность прогнозирования в регрессионных моделях**

**Пояснительная записка**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.04.04-01 ТЗ 01-1-ЛУ**

**Исполнители:**

студент группы БПИ221  
/ Знатнов Е. П. /  
«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г

**Москва 2025**

УТВЕРЖДЕН

RU.17701729.04.04-01 ТЗ 01-1-ЛУ

# **Разработка программного комплекса для исследования влияния аномальных наблюдений на точность прогнозирования в регрессионных моделях**

**Пояснительная записка**

**RU.17701729.04.04-01 ТЗ 01-1**

**Листов 22**

**Москва 2025**

# СОДЕРЖАНИЕ

**СОДЕРЖАНИЕ 2**

**1. ВВЕДЕНИЕ 3**

**1.1. Наименование программы 3**

**1.2. Краткая характеристика и область назначения 3**

**2. НАЗНАЧЕНИЕ РАЗРАБОТКИ 4**

**2.1. Функциональное назначение 4**

**2.2. Эксплуатационное назначение 4**

**3. ТРЕБОВАНИЯ К ПРОГРАММЕ 5**

**3.1. Постановка задачи на разработку программы 5**

**3.2. Описание алгоритма и функционирования клиентской части 5**

**3.2.1. Теоретическая часть 5**

**3.2.2. Архитектура проекта 5**

**3.2.3. Описание структуры приложения 7**

**3.2.4. Описание содержания исходного кода 7**

**3.5 Описание выбора технических и программных средств 14**

**4 ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ 16**

**4.3 Ориентировочная экономическая эффективность 16**

**4.4 Предполагаемая потребность 16**

**4.5 Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами 16**

**ПРИЛОЖЕНИЕ 1 17**

**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 17**

**ТЕРМИНЫ И СОКРАЩЕНИЯ 21**

# ВВЕДЕНИЕ

## Наименование программы

Наименование программы – "Разработка программного комплекса для исследования влияния аномальных наблюдений на точность прогнозирования в регрессионных моделях ".

Наименование программы на английском языке – "Development of a Software Package to Study the Influence of Anomalous Observations on the Prediction Accuracy in Regression Models".

Краткое наименование – "MSnOutliers".

## Краткая характеристика и область назначения

"MSnOutliers" – приложение для исследования качества различных статистических методов на выборках данных с большим числом зашумленных (т. е. содержащих в себе помимо полезной нагрузки некоторый шум известного распределения) данных.

# НАЗНАЧЕНИЕ РАЗРАБОТКИ

## Функциональное назначение

Приложение «MsnOutliers» предоставляет возможность изучить влияние выбросов (данных, отличающихся по абсолютному значению на порядок от среднего по всей остальной выборке) на качество работы различных регрессионных методов математической статистики. Предполагается, что основное использование приложения будет происходить в образовательных целях.

## Эксплуатационное назначение

Приложение «MsnOutliers» позволяет отслеживать влияние выбросов в данных на качество работы регрессионных методов. Для анализа могут использоваться данные, подготовленные пользователем либо же сгенерированные на стороне приложения.

Целевая аудитория – школьники и студенты, проходящие подготовку по дисциплине “Математическая статистика” или каким-либо смежным с ней, а также преподаватели, читающие вышеупомянутые курсы.

# ТРЕБОВАНИЯ К ПРОГРАММЕ

## Постановка задачи на разработку программы

Приложение должно предоставлять возможность построения линейной многомерной регрессии на основе некоторого статистического метода. Результатом работы приложения на некотором наборе данных является не только построенная регрессия (т. е. набор коэффициентов), но и график, отражающий качество работы метода при той или иной степени зашумления. Данные, на которых предполагается строить регрессию, могут быть как предоставлены пользователем, так сгенерированы на стороне приложения. Если данные, которые подаются на вход приложению, не являются зашумленными изначально, предполагается, что они будут зашумлены перед построением регрессии на стороне приложения. Характер шума (вид распределения, параметры, количество наблюдений, которое будет зашумлено) также настраивается пользователем.

## Описание алгоритма и функционирования клиентской части

### Теоретическая часть

3.2.1.1 Описание статистических методов

Существует большое количество различных методов построения многомерной линейной регрессии. В рамках нашего приложения было принято решение сосредоточиться на так называемых робастных (англ. «robust» - крепкий, устойчивый) методах регрессионного анализа. Такие методы устойчивы к различным выбросам, качество построенной регресии на них падает значительно меньше, в сравнении с классическими регрессионными методами.

Существует два принципиальных подхода к построению робастных оценок — M-оценки и R-оценки. М-оценки представляют собой комбинацию двух других регрессионных методов. Один из них — это обычно классическая оценка методом наименьших квадратов (далее — МНК), а другой — какой-либо устойчивый к выбросам, но менее качественный метод. Разделение происходит с помошью так называемого тюнингового параметра *c.* Если абсолютное значение входного параметра меньше *c,* то ошибка на этом параметре считается по функции потерь (или лосс) МНК, в противном случае — по функции потерь второго метода. Таким образом, функция потерь М-оценки, которая и используется для построения регресии, является кусочной.

R- оценки, в отличие от М-оценок, опираются на такое понятие, как ранг. Рангом некоторго наблюдения в выборке называется его индекс (положение) в отсортированной выборке. R-оценки являются робасными по своей природе — от абсолютных значений мы переходим к порядкам наблюдений, а это означает, что наблюдения, отличающиеся на несколько порядков, и наблюдения с разницей в рамках одного порядка для такого метода оценки неразличимы.

3.2.1.2 Характеристика использующихся в приложении статистических регрессионных методов.

3.2.1.2.1 Метод наименьших квадратов

Классический регрессионный метод, который основан на минимизации квадратичной функции потерь (см. Приложение 2). Включен в проект в для сравнения качества работы на выборках с выбросами в сравнении с робасными оценками.

3.2.1.2.2 М-оценка Хьюбера

М-оценка, функция потерь которой сочетает в себе лосс МНК и метода наименьших модулей (см. Приложение 3).

3.2.1.2.3 М-оценка Тьюки

М-оценка, функция потерь которой сочетает в себе лосс МНК и полином 6-ой степени (см. Приложение 4)

3.2.1.2.4 Метод наименьших модулей

Классический R-метод, где функцией потерь является модуль отклонения предсказания регрессионной модели от истинного значения, в отличие, например, от МНК, где минимизируется квадрат отклонения

3.2.1.2.5 R-оценка Тейл-Сена

Метод, опирающийся на такое понятие, как наклон между точками выборки. Наклоном называется отношение значений разности целевых переменных к разности значений признаков. Для многомерной оценки этим методом строятся всевозможные наклоны в рамках каждого из признаков независимо друг от друга. Далее, для каждой из полученных последовательностей наклонов находится медиана. Полученные медианы являются оценками коэффициентов результирующей регресии.

3.2.1.3 Характеристика использующихся в приложении регрессионных методов машинного обучения.

ЭТО СТЕПЕ

### Архитектура проекта

Приложение состоит из трех основных модулей:

1. UI — интерфейс пользователя. Этот блок — едниственное, с чем взаимодействует пользователь. Через интерфейс, который представлен в UI приложения, пользователь может выбирать какие модели и с какими параметрами будут использованы в анализе тех или иных данных. Через интерфейс пользователь также выбирает, хочет ли он «замусоривать» данные(т. е. добавлять к значениям целевой переменной шум) и, если да, то каким именно должен быть этот шум (вид распределения, параметры распределеия, к скольки объектам нужно добавить шум и т. д.). Результатом работы интерфейса является файл формата JSON, который описывает вид моделей, шумов и прочих настраиваемых параметров (см. Приложение 5). Далее этот файл поступает на вход основной части программы. Написан UI на языке С++ с использованием фреймворка Qt и библиотеки nlohmann::json для работы с JSON-файлами.

2. Main — основной вычислительный блок. Этот блок получает на вход JSON-файл с описанием моделей, видов шумов и данных, на которых должны применяться модели. Этот блок содержит в себе классы, описывающие статистические методы и методы машинного обучения, перечисленные в пункте 3.2.1.2 и 3.2.1.3 настоящей пояснительной записки. Помимо этого, в блоке Main содержатся вспомогательные классы. Эти классы используются для генерации данных заданного распределения, для парсинга (обработки и извлечения необходимых данных). Также в проекте содержится реализация интерфейса для работы с матрицами. По результатам работы этого блока формируется еще один JSON-файл, который отправляется на вход блоку для визуализации (см. Приложение 6). Написан блок Main на языке С++ с использованием библиотек nlohmann::json (для работы с JSON-файлам) и Eigen (для работы с матрицами).

3. Блок визуализации данных. На вход блоку поступает JSON-файл с описанием результатов работы блока Main. Исходя из этих данных строится визуализация зависимости значения ошибки от количества «замусоренных данных». Написан блок на Python с использованием библиотеки matplotlib для построения графиков.

### Описание структуры приложения

Приложение разделено на три логические компоненты:

1. ui — компонента, отвечающая за интерфейс приложения. Интерфейс собирается в бинарный файл, который запускается для взаимодействия с приложением
2. src — компонента, содержащая в себе движок приложения (функцию main) и компоненты, ответственные за отрисовку результатов работы блока Main. Блок Main также собирается в бинарный файл, который вызывается в ui. Собранному бинарному файлу передается на вход путь до JSON-файла c описанием информации о моделях, данных, шумах и проч.
3. lib – компонент, хранящий в себе весь библиотечный код (реализация методов регресии, как статистических, так и основанных на подходах из области машинного обучения, код для парсинга файлов с данными, генерации значений из заданных распределений и проч.)

### Описание содержания исходного кода

* + - 1. Проект приложения состоит из следующих файлов и папок:
         1. Папка ui содержит классы, отвечающие за реализацию пользовательского интерфейса:

modeltemplate.h/modeltemplate.cpp – файлы, содержащие реализацию класса ModelTemplate. Класс наследуется от класса Qdialog (класс фреймворка Qt, описывающий диалоговое окно). Этот класс содержит реализацию шаблона для представления окна с некоторым числом текстовых полей и кнопками подтверждения («OK») и отмены («Cancel») (реализация — 3.2.5.1.)

modeldialog.h/modeldialog.cpp – файлы, содержащие реализацию модуля ModelDialog. Этот класс является дочерним от ModelTemplate, отвечает за экран ввода данных о модели в общем пайплайне обработки моделей. (реализация – см. 3.2.5.2).

generatedialog.h/generatedialog.cpp – файлы, GenerateDialog (реализация – см. 3.2.5.3).

mainwindow.h/mainwindow.cpp – файлы, содержащие в себе реализацию классов MainWindow (основного окна приложения) (реализация — см. 3.2.5.4) и ModelTable (класса, отвечающего за отображение списка текущих моделей со всеми параметрами)(реализация — см. 3.2.5.5)

main.cpp — файл с функцией main, движок ui.

* + - * 1. Папка lib/MS содержит код, отвечающий за реализацию методов математической статистики. В этой папке содержатся файлы:

StatsMethod.hpp/StatsMethod.cpp – файлы, содержащие реализацию базового класса, описывающего некоторый метод математической статистики для регрессионного анализа (реализация – см. 3.2.5.7.)

GradMethod.hpp/GradMethod.cpp – файлы, содержащие реализацию класса GradMethod, дочернего от StatsMethod. Класс описывает статистический метод, который использует в своей основе метод градиентного спуска (реализация 3.2.5.8.)

LeastSquares.hpp/LeastSquares.cpp — файлы, содержащие реализацию класса LeastSquares. Этот класс наследуется от StatsMethod и реализует метод наименьших квадратов (реализация — см . 3.2.5.9).

Huber.hpp/Huber.cpp — файлы, содержащие реализацию класса Huber. Класс Huber наследуется от класса GradMethod и реализует М-оценку Хьюбера (реализация — см. 3.2.5.10).

Tukey.hpp/Tukey.cpp — файлы, содержащие реализацию класса Tukey. Класс Tukey наследуется от класса GradMethod и реализует М-оценку Тьюки (реализация — см. 3.2.5.11).

MinAbsDeviation.hpp/MinAbsDeviation.cpp — файлы, содержащие реализацию класса MinAbsDeviation. Класс MinAbsDeviation является дочерним от GradMethod и реализует оценку, основанную на методе наименьших модулей.

* + - * 1. В папке Kernel содержится ядро приложения, то есть все модели с необходимыми контроллерами, а также вспомогательные файлы, описывающие структуры запросов того или иного типа. В этой папке находятся файлы:

Pallette.h/Pallette.cpp – файл, содержащий реализацию структуры Pallette (структуры, содержащей в себе цвета, использующиеся для визуализации стека) и перечисление NodeStatus (перечисление статусов вершин стека, необходимых для визуализации).

MouseState.h – файл, содержащий реализацию структуры MouseState. Структура описывает состояние мыши путем записи места взаимодействия на окне и статус взаимодействия (нажатие, перемещение или отпускание).

PersistentStackModel.h/PersistentStackModel.cpp – файлы, содеражщие реализацию класса PersistentStackModel. Этот класс является реализацией модуля Model (реализация – см. 3.2.5.5).

PersistentStackGeomModel.h/PersistentStackGeomModel.cpp – файлы, содержащие реализацию класса PersistentStackGeomModel. PersistentStackGeomModel – промежуточная модель стека. Она содержит в себе структуру DrawData (cм. 3.2.2.1.3.6.), которая преобразует персистентный стек в данные, необходимые для отрисовки. Эта модель выступает промежуточным звеном между PersistentStackModel и PersistentStackView (реализация – см. 3.2.5.3).

PersistentStackGeomController.h/PersistentStackGeomController.cpp – файлы, содержащие реализацию класса PersistentStackGeomController. Этот контроллер связывает PersistentStackModel и PersistentStackGeomModel и похож по реализаци на PersistentStackViewController(см. 3.2.2.1.1.2). Отличие заключается в том, что этот контроллер подключается к PersistentStackModel, а не к PersistentStackGeomModel.

DrawData.h – файл, содержащий реализацию структуры DrawData, описывающей информацию, необходимую для визуализации той или иной версии персистентного стека.

Animation.h – файл с реализацией класса Animation.h, использующегося для анимации визуализации.

* + - * 1. Файлы, не входящие ни в один из вышеописанных каталогов:

Application.h/Application.cpp – файл с реализацией класса Application, являющегося основным для приложения. В этом классе собираются все необходимые подписки для модулей приложения. Для удобства сборки подписки каждого из модулей разнесены по файлам:

ApplicationGUI.h/ApplicationGUI.cpp – для сборки интерфейса.

ApplicationImpl.h/ApplicationImpl.cpp – для сборки ядра.

Except.h/Except.cpp – файл, содержащий реализацию функции react(). Эта функция реагирует на возможные исключения, которые могут появляться в процессе работы приложения.

QObserver.h- файл, содержащий реализации разных видов Observer’ов и Observable. В работе проекта используется Observer типа CColdInput – этот Observer выполняет передаваемую ему функцию только на вызов notify(). Важно отметить, что связь между модулями с помощью CObservable и CColdInput является блокирующим.

* + 1. **Описание реализации компонент приложения**
       1. Реализация класса ModelTemplate.
          1. \_lineEdits — структура данных, хранящая отображение имени поля на указатель на это поле. Используется для сохранения введных пользователем данных.
          2. setupLineEdit() – метод, принимающий имя текстового поля, текст, который должен быть выставлен в этом поле в начальном состоянии и указатель на валидатор (объект, который фильтрует данные, вводящиеся в это поле). Метод создает новое текстовое поле по заданным параметрам и добавляет его в \_lineEdits
          3. ModelTemplate() - конструктор, принимающий на вход указатель на родительский объект (объект, до момента уничтожения которого будет существовать ModelTemplate)
          4. getData() – метод, возвращающий текущие значения в текстовых полях объекта ModelTemplate. Текст из каждого поля получается в формате объекта Qstring, далее из всех результатов собирается объект типа QstringList. У класса ModelDialog это метод объявлен виртуальным и должен быть переопределен в дочерних классах.
       2. Реализация класса ModelDialog.
          1. В состав \_lineEdits для класса ModelDialog входят следуюшие поля:

1. Delta — поле для ввода тюнингового параметра (актуально только для М-оценок, в остальных оценках игнорируется). Значение по умолчанию — 1.0, в поле можно ввести только корректно написанное действительное число.

2. Epochs — поле для ввода количества итераций градиентного спуска. Значение по умолчанию — 1000, в поле можно ввести только корректно написанное целое число.

3. Learning Rate — поле для ввода шага градиентного спуска. Значение по умолчанию — 0.001, в поле можно ввести только корректно написанное действительное число

4. Parameter 1 и Parameter 2 — поля для вводов двух параметров распределения шума. Каждый параметр имеет свое значение в зависимости от выбранного распределения шума, получить эту информацию можно по нажатию кнопки «Info» (см. 3.2.5.2.6.). Значения по умолчанию — 1.0 и 0.0 для параметров 1 и 2 соответственно, в оба поля можно ввести только корректно написанное действительное число.

5. Path to data — поле для ввода пути до файла с данными для анализа. Значение по умолчанию — source.csv, в поле может быть введена любая строка.

6. Num. features — поле для ввода количества признаков в выбранном датасете. Значение по умолчанию — 3, в поле может быть введено только корректное целое число.

* + - * 1. \_comboBoxes — структура данных, которая хранит в себе отображение имен выпадающих списков на указатели на эти выпадающие списки. В состав \_comboBoxes входят следующие поля:

1. Model — поле выбора статистической модели. Значение по умолчанию — LSM, доступен выбор из элементов modelsList (см. 3.2.5.2.5.)

2. Noise type — поле выбора распределения шумов. Значение по умолчанию — Normal, доступен выбор из элементов distributionsList (см. 3.2.5.2.3)

3. ML model — поле выбора модели машинного обучения. Значение по умолчанию — None, доступен выбор из mlModelsList(3.2.5.2.4)

* + - * 1. distributionsList — список строк с названиями распределений. Распределения в distributionsList используются для генерации шумов при обработке и анализе данных. В приложении доступен выбор из следующего набора распределений:

1. Normal — Нормальное распределение

2. Student — распределение Стьюдента

3. Cauchy — распределение Коши

4. Laplace — распределение Лапласа

* + - * 1. mlModelsList — список строк с названиями моделей машинного обучения. Эти модели могут использоваться для анализа данных няряду со статистическими методами. В приложении доступен выбор из следующего набора моделей:

1. IForest - Isolation Forest

2. DBSCAN

3. OCSVM

Также доступна опция None — она означает, что пользователь не выбрал какую-либо модель

* + - * 1. modelsList – список строк с названиями статистических методов, использующихся в анализе данных. В приложении доступен выбор из следующего набора методов:

1. LSM — метод наименьших квадратов

2. HUB — М-оценка Хьюбера

3. TUK — М-оценка Тьюки

4. LAD — метод наименьших модулей

5. THS — R-оценка Тейл-Сена

* + - * 1. \_info — указатель на кнопку Info на ModelDialog. При нажатии на кнопку Info пользователь получае информацию о том, какой смысл имеют параметры «Param. 1» и «Param. 2» в контексте каждого из распределений.
        2. SetupComboBox() - метод, получающий на вход имя выпадающего списка и создающий список с заданным именем, заполняющий его значениями в соответствии с именем (см. 3.2.5.2.2.) и добавляющий его в \_comboBoxes.
        3. getLabel() — метод, создающий объект типа Qlabel (неинтерактивное текстовое поле) с заданным текстом. Метод используется для заполнения таблицы с описанием параметров для каждого из распределений шумов.
        4. getData() - метод, переопределяющий метод базового класса. При получении значений во всех полях объекта ModelDialog происходит проверка, существует ли файл, путь до которого указан в поле «Path to data». Если нет — значение в этом поле заменятся на пустое.
        5. ModelDialog () - конструктор, инициализирующий все поля и создающий объект типа ModelDialog (см. Приложение 7)
      1. Реализация класса GenerateDialog.
         1. В состав \_lineEdits входят следующие поля:

1. Num. features — поле для ввода желаемого числа признаков в датасете. Значение по умолчанию — 3, в поле можно ввести только корректно написанное целое число.

2. Num. samples — поле для ввода желаемого числа объектов в датасете. Значение по умолчанию — 100, в поле можно ввести только корректно написанное целое число.

* + - * 1. GenerateDialog() - конструктор, инициализирующий поля и создающий объект типа GenerateDialog() (см. Приложение 8).
      1. Реализация класса ModelTable.
         1. Класс ModelTable представляет собой объект, который отображается на главном экране приложения. ModelTable хранит в себе информацию о том, какие модели были выбраны пользователем для анализа в рамках текущей сессии (сессией в рамках приложения является процесс от генерации модели до запуска на выбранных моделях). Данные, которые хранятся в памяти этого класса представлены в виде таблицы (см. Приложение 9). Конструктор ModelTable инициализирует объект.
         2. \_data — таблица с данными о моделях в текущей сессии.
         3. rowCount() и columnCount() - методы, возвращающие текущие размеры таблицы с информацией о моделях.
         4. data() - метод, возвращающий строку (т. е. полную конфигурацию некоторой модели) по индексу в таблице.
         5. headersData() - метод, возвращающий строку с заголовками (т. е. верхнюю строку с именами всех столбцов) (см. Приложение 9).
         6. setData() - метод, обновляющий таблицу с данными о моделях в текущей сессии.
      2. Реализация класса MainWindow.
         1. \_models — список моделей, выбранных в рамках текущей сессии. Модели хранятся в виде массивов строк, элементы в этих массивах описывают параметры моделей (см. реализацию класса ModelDialog).
         2. model – указатель на объект типа ModelTable, используется для визуального отображения моделей в текущей пользовательской сессии.
         3. generate() - метод, принимающий на вход желаемое количество признаков, желаемое количество объектов и некоторую искомую зависимость и генерирующий файл с данными выбранных параметров. Генерируемый файл имеет расширение csv, данные, разделителем данных выступает точка с запятой (этот выбор обусловлен тем, что при парсинге файла в действительных числа целая и дробная часть отделены между собой не запятой, а точкой). Сгенерированные данные помещаются в файл sample.csv.
         4. openModelDialog() - метод, который связывается с кнопкой «Create model» и создает окно типа ModelDialog (см. реализацию класса ModelDialog). После нажатия кнопки «OK» на окне ModelDialog, данные, которые были введены пользователем, сохраняются в \_models.
         5. dumpModels() - метод, который связывается с кнопкной «Dump models». При вызове, метод сохраняет все данные, которые содержатся в \_models в JSON-файл. Структура JSON-файла представлена в Приложении 4.
         6. runMethods() - метод, связываюшийся с кнопкой «Run on models». По нажатии на эту кнопку, вызывается бинарный файл, в который был собран модуль приложения Main, на вход этому модулю передается путь до файла с описанием выбранных пользователем моделей.
         7. generateData() - метод, связывающийся с кнопкой «Generate data». По нажатии на эту кнопку создается объект типа GenerateDialog для генерации синтетических данных (см. реализацию класса GenerateDialog).
         8. showImages() - метод, вызывающий отрисовку результатов работы блока Main.
      3. Функция createDialog используется для создания окон с сообщениями. Эти окна сообщают о статусах операций в процессе взаимодействия с приложением. На вход функция принимает название окна и текст, который будет использован в качестве сообщения.
      4. Реализация класса StatsMethod.
         1. \_name – имя статистического метода.
         2. \_delta – поле для хранения параметра Delta, т.е. тюнингового параметра (используется только для методах, основанных на М-оценках).
         3. \_epochs — поле для хранения параметра Epochs, количества итераций алгоритма градиентного спуска.
         4. \_learningRate — поле для хранения параметра Learning Rate, шага градиентного спуска.
         5. \_featuresMatrix и \_targetMatrix — объекты типа Eigen::Matrix, использующиеся для хранения выборки, поступившей на вход выбранной модели. \_featuresMatrix хранит в себе наборы признаков, а \_targetMatrix — целевые переменные. Между i-ой строкой \_featuresMatrix и i-ым элементом \_targetMatrix есть соответствие — набор признаков на искомой зависимости дает элемент из \_targetMatrix.
         6. compute() - виртуальный метод, который используется для запуска выбранного статистического метода на загруженных данных. Метод compute() переопределяется во всех дочерних классах в зависимости от статистического метода.
         7. makeNoise() – метод, замусоривающий данные, загруженные в выбранный статистический метод. makeNoise() принимает на вход целое число — количество элементов выборки, которое предполагается замусорить — и вид распределения, которое будут иметь шумы. Замусоривание происходит следующим образом — классом, отвечающим за выбранное распределение, генерируется значение, которое прибавляется к целевой переменной. Так происходит с выбранным числом случайных объектов в загруженной выборке.
         8. Конструктор, инициализирующий класс. В конструктор передаются данные для инициализации полей класса.
      5. Реализация класса GradMethod
         1. \_initialWeights — изначальные веса вектора, который будет описывать результирующую регрессию. Изначальные веса могут отличаться в зависимости от метода.
         2. gradient() и loss() - виртуальный функции, описывающие то, как нужно считать градиент и лосс (т. е. функции потерь) для конкретного метода. Дочерние классы переопределяют эти методы в зависимости от тех статистических методов, которые они описывают.
         3. Конструктор, инициализирующий веса, хранящиеся в классе. По умолчанию все веса инициализируются нулями.
         4. сompute() - переопределение метода из родительского класса. В переопределнной версии метода описывается алгоритм градиентного спуска.
         5. Также в файле GradMethod.cpp определена функция median. Функция принимает на вход вектор и возвращает медиану полученного вектора.
      6. Реализация класса LeastSquares
         1. compute() - переопределение метода из родительского класса. Метод наименьших квадратов, который реализуется классом LeastSquares, имеет известное решение, поэтому, этот метод просто применяет известное решение.
         2. Конструктор класса, который обращается к конструктору родительского и ставит именем класса «LSM».
      7. Реализация класса Huber.
         1. Методы loss() и gradient() переопределены в соответствии с формулами функции потерь и градиента М-оценки Хьюбера
         2. Конструктор класса, который обращается к конструктору родительского и ставит именем класса «HUB».
      8. Реализация класса Tukey.
         1. Методы loss() и gradient() переопределены в соответствии с формулами функции потерь и градиента М-оценки Тьюки.
         2. Конструктор класса, который обращается к конструктору родительского и ставит именем класса «TUK».
      9. Реализация класса MinAbsDeviation.
         1. compute() - переопределение метода из родительского класса. Метод оптимизирует функцию потерь, основанную на методе наименьших модулей (модуль отклонения от целевого значения).
         2. Конструктор класса, который обращается к конструктору родительского и ставит именем класса «LAD».
  1. **Организация входных данных**

Взаимодействие с приложением происходит через UI. При запуске приложения пользователь попадает на главный экран (см. Приложение 10). На главном экране доступно 4 кнопки. По нажатии на кнопку «Create model», пользователь переходит на экран генерации модели (см. Приложение 6), где вводит желаемые параметры модели. Одним из параметров является путь до файла с данными, на котором предполагается запускать ту или иную модель — если файла по выбранному пути нет, это поле будет оставлено пустым. При этом будет создано окно с сообщением об ошибке поиска файла и уведомлением о том, что путь до данных будет оставлено пустым (см. Приложение 11). После создания модели на главном экране в таблице моделей в текущей сессии появляется новая модель (см. Приложение 9). По нажатии на кнопку «Dump models» происходит генерация JSON-файла с описанием моделей в текущей сессии (см. Приложение 5). При этом будет выведено сообщение о том, что файл с моделями был сгенерирован и будет указан путь до этого файла (см. Приложение 12). При нажатии на кнопку «Run on models» происходит запуск модуля Main, если файла с описанием моделей нет — появляется окно с сообщением об ошибке (см. Приложение 13). При нажатии на кнопку «Generate data» пользователь переходит на окно генерации данных (см. Приложение 8).

* 1. **Организация выходных данных**

Выходными данными для приложения являются графики, визуализирующие результат работы блока Main на выбранных пользователем моделей (см. Приложение 14). Помимо этих графиков, приложение производит два типа файлов — JSON-файл с описанием моделей (см. Приложение 5) и JSON-файл с описанием результатов работы блока Main (для визуализации) (см. Приложение 6)

## Описание выбора технических и программных средств

Приложение написано на языке С++. Этот язык выбран из-за того, что он предоставляет возможность низкоуровневого взаимодействия с объектами, создаваемыми в программе. Визуализация в проекте опирается на паттерн Model-View-Controller, описанный выше. Взаимодействие между модулями устроено с использованием паттерна Observer. Для корректной и быстро работающей визуализации важно обеспечить блокирующее поведение при обмене запросами через Observer-Observable, также важно сделать обмен сообщениями быстрым. Это важно для создания качественной и быстро работающей визуализации. Так как многие детали пересылки сообщений и налаживания контакта между модулями находятся на достаточно низком уровне, для работы с этими деталями был выбран язык С++.

Другой важной причиной выбора именно С++ было то, что выбранный для создания приложения фреймворк Qt написан именно на этом языке. Qt – мощный и удобный фреймворк, предоставляющий большое количество средств для создания отзывчивого, понятного и быстро работающего интерфейса для пользователя. Также, фреймворк Qt предоставляет множество средств для визуализации, например, Qwt. Qwt используется в проекте, так как он имеет обширный функционал и очень хорошо оптимизирован, то есть отвечает требованиям по созданию быстро работающей визуализации и интерфейса).

Для управления версиями приложения используется система контроля версий Git (версия не ниже 2.43.2) (см. “Список используемой литературы”, пункт 17). Репозиторий проект расположен на сайте GitHub. Приложение использует утилиту Clang-Format для контроля код-стайла (код-стайл проекта опирается на код-стайл Google, версия Clang-Format не ниже 17.0) (см. “Список используемой литературы”, пункт 18), а также статический анализатор кода Clang-Tidy (версия не ниже 17.0) (см. “Список используемой литературы”, пункт 19). RTTI в приложении не используется. Приложение написано с помощью фреймворка Qt (версия не ниже 5.15), следовательно, зависит от библиотек, необходимых для корректной работы этого фреймворка.

# ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

## Ориентировочная экономическая эффективность

Приложение “Персистентный стек” является очень полезным ресурсом при изучении дисциплины “Алгоритмы и структуры данных” или какой-либо смежной с ней. Распространение приложение по платной модели не предусмотрено, исходный код приложения расположен в свободном доступе в репозитории на сайте GitHub.

## Предполагаемая потребность

Из вышесказанного следует, что предполагаемая потребность приложения “Персистентный стек” базируется на полезности при изучении множества дисциплин различных ИТ направлений. Визуализация структуры данных позволяет лучше понять ее устройство. Также, само приложение может использоваться для изучения паттерна MVC на примере исходного кода приложения.

## Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами

Основными конкурентами приложения можно назвать ресурсы от университета Сан-Франциско (см. “Список используемой литературы”, пункт 23) и VisuAlgo (см. “Список используемой литературы”, пункт 24). Эти ресурсы сущетсвуют только в онлайн формате, скорость и доступность работы с ними зависит от подключения пользователя к интернету. Приложение “Персистентый стек” не требует подключения к интернету и может использоваться в образовательных учреждениях без создания риска выхода обучающихся в интернет для пользования запрещенными материалами.

Преимуществом приложения “Персистентный стек” также можно назвать факт модульности приложения. Используемый паттерн MVC пи его реализация в проекте позволили создать архитектуру из независимых модулей в приложении. Таким образом, приложение можно легко развивать, добавляя новые модули, отвечающие за реализацию какой-либо структуры данных или новые модули типа View

# ПРИЛОЖЕНИЕ 1

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

* + 1. ГОСТ 19.101-77: Виды программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    2. ГОСТ 19.102-77: Стадии разработки. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    3. ГОСТ 19.103-77: Обозначения программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    4. ГОСТ 19.104-78: Основные надписи. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    5. ГОСТ 19.105-78: Общие требования к программным документам. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    6. ГОСТ 19.106-78: Требования к программным документам, выполненным печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    7. ГОСТ 19.201-78: Техническое задание. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    8. ГОСТ 19.602-78: Правила дублирования, учета и хранения программных документов, выполненных печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    9. ГОСТ 19.603-78: Общие правила внесения изменений. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    10. ГОСТ 19.604-78: Правила внесения изменений в программные документы, выполненные печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    11. ГОСТ 19.404-79: Пояснительная записка. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    12. ГОСТ 19.301-79: Программа и методика испытаний. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    13. ГОСТ 19.401-78: Текст программы. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    14. ГОСТ 19.505-79: Руководство оператора. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
    15. Сайт CMake [электронный ресурс] /Режим доступа: <https://cmake.org/> свободный (дата обращения 14.02.2024)
    16. Сайт GCC, the GNU Compiler Collection [электронный ресурс] /Режим доступа: <https://gcc.gnu.org/> свободный (дата обращения 14.02.2024)
    17. Сайт Git [электронный ресурс] /Режим доступа: https://git-scm.com/ свободный (дата обращения 14.02.2024)
    18. Сайт Clang-Format [электронный ресурс] /Режим доступа: <https://releases.llvm.org/17.0.1/tools/clang/docs/ClangFormat.html> свободный (дата обращения 14.02.2024)
    19. Сайт Clang-Tidy [электронный ресурс] /Режим доступа: <https://releases.llvm.org/17.0.1/tools/clang/tools/extra/docs/clang-tidy/index.html> свободный (дата обращения 14.02.2024)
    20. Конспект ИТМО «Персистентный стек» [Электронный ресурс] /Режим доступа:

<https://neerc.ifmo.ru/wiki/index.php?title=Персистентный_стек>

свободный (дата обращения: 02.01.2024).

* + 1. Лекторий ФПМИ МФТИ «Алгоритмы и структуры данных (продвинутый поток). 2. Амортизированный анализ. Персистентная очередь» [Электронный ресурс] /Режим доступа: <https://www.youtube.com/watch?v=r6NnmynZMRs>, свободный (дата обращения: 04.01.2024).
    2. Репозиторий Library [Электронный ресурс] /Режим доступа: <https://github.com/DimaTrushin/Library> свободный (дата обращения: 02.03.2024).
    3. Сайт Data Structures Visualisation [Электронный ресурс] /Режим доступа: <https://www.cs.usfca.edu/~galles/visualization/Algorithms.html> свободный (дата обращения 13.03.2024).
    4. Сайт VisuAlgo [Электронный ресурс] /Режим доступа: <https://visualgo.net/en> свободный (дата обращения 13.03.2024).

**ПРИЛОЖЕНИЕ 2**

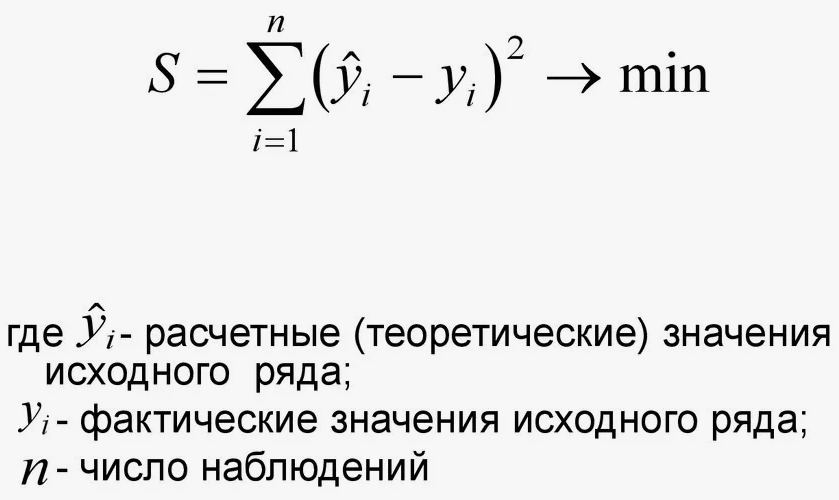
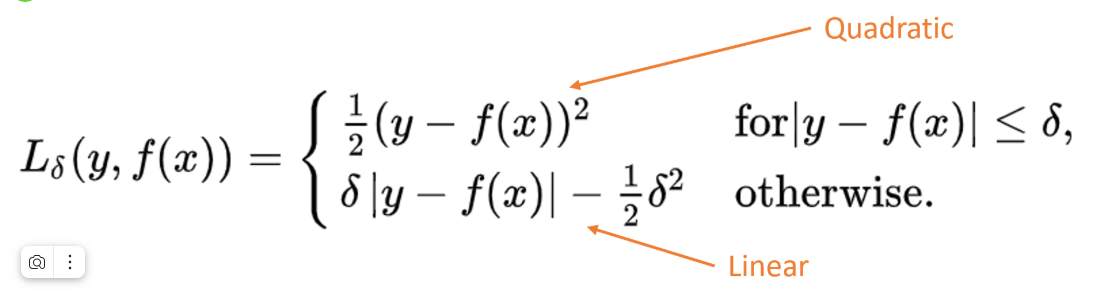


Рис.1 Функция потерь МНК

**ПРИЛОЖЕНИЕ 3**

Рис.2 Функция потерь Хьюбера

**ПРИЛОЖЕНИЕ 4**

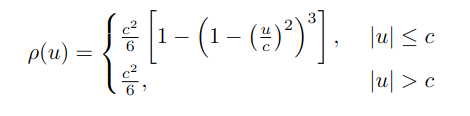


Рис.3 Функция потерь Тьюки

**ПРИЛОЖЕНИЕ 5**

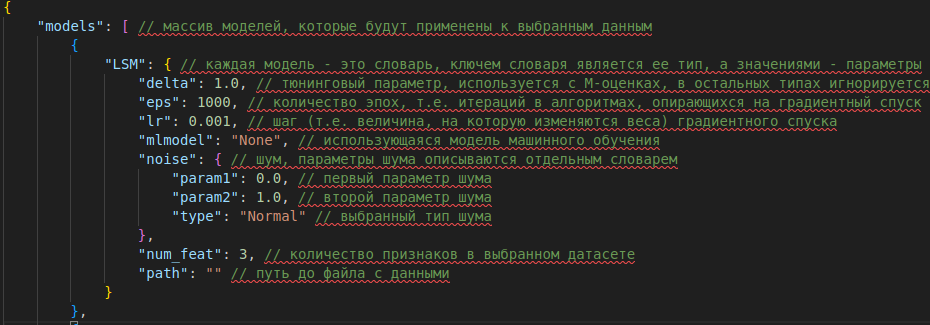
****

Рис.4 Описание структуры JSON-файла «ui->main»

**ПРИЛОЖЕНИЕ 6**

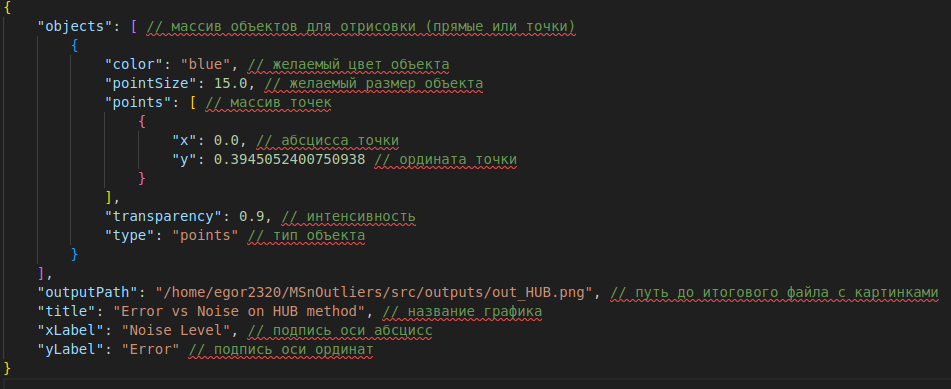
****

Рис.5 Описание структуры JSON-файла «main->paintings»

**ПРИЛОЖЕНИЕ 7**

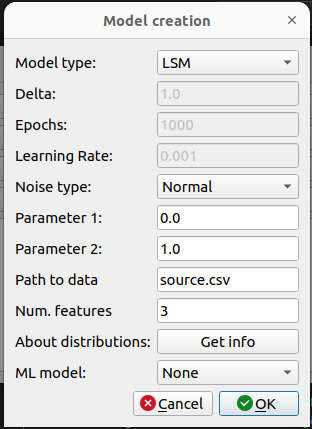
****

Рис.6 Экран создания модели (ModelDialog)

**ПРИЛОЖЕНИЕ 8**

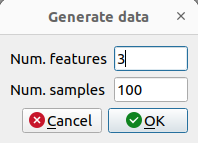
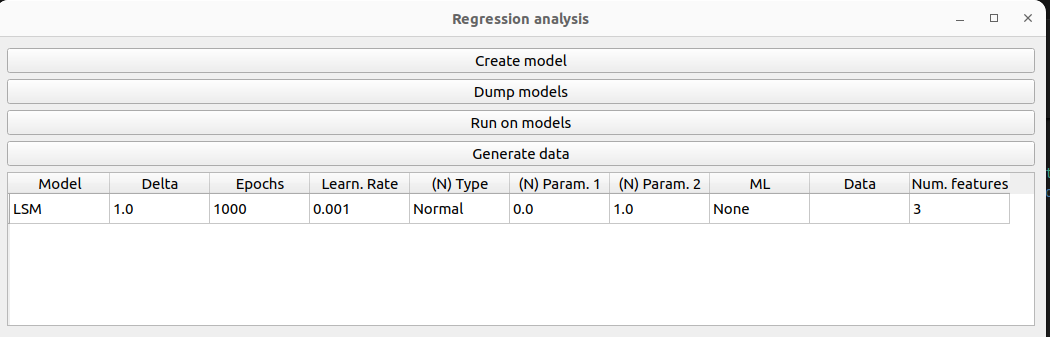
****

Рис.7 Экран генерации данных (GenerateDialog)

**ПРИЛОЖЕНИЕ 9**

Рис.8 Данные о моделях в текущей сессии

**ПРИЛОЖЕНИЕ 10**

Рис.9 Главный экран приложения (MainWindow)

**ПРИЛОЖЕНИЕ 11**

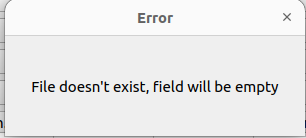


Рис.10 Сообщение об ошибкев случае пустого файла

**ПРИЛОЖЕНИЕ 12**

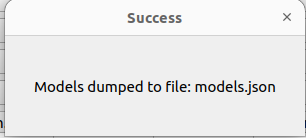
****

Рис.11 Сообщение об успешной генерации файла с моделями

**ПРИЛОЖЕНИЕ 13**

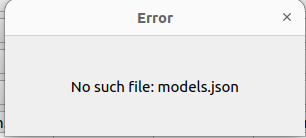
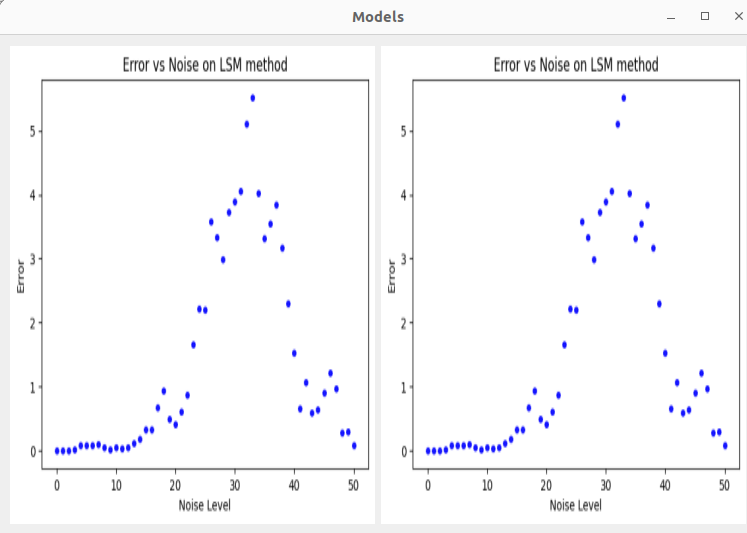
****

Рис.12 Сообщение об отсутствии файла с описанием моделей

**ПРИЛОЖЕНИЕ 14**

Рис.13 Визуализация данных

## ТЕРМИНЫ И СОКРАЩЕНИЯ

1. **API (Application programming interface) –** интерфейс приложения, к которому имеет доступ пользователь.
2. **Одномерный массив** – структура данных, представляющая собой набор переменных одного типа, расположенных в памяти компьютера последовательно.
3. **ОС** – Операционная система.
4. **Персистентность** – свойство структуры данных, заключающееся в возможности доступа к каждой прошлой версии структуры данных. В тексте технического задания речь идет о полной персистентности – доступы могут быть как на чтение, так и на запись.
5. **ПК** – Персональный компьютер.
6. **Сериализация** – процесс перевода объекта в языке программирования в его битовое представление.
7. **Стек** – структура данных, работающая по принципу last-in-first-out: записи из стека извлекаются в порядке, обратном тому, в котором они были в него добавлены.
8. **Фреймворк** – приложение, предназначенное для разработки собственных приложений на определенном языке программирования.
9. **Шаблон –** механизм языка программирования С++, позволяющий параметризовать некоторое поле класса, в случае классов проекта параметризованное поле – тип хранимых данных.

**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

| Лист регистрации изменений | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |