**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, будь то методы математической статистики или же машинного обучения. 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).

* + - * 1. Папка Library содержит весь код, реализующий внутреннюю логику самой структуры данных "Персистентный стек" и сопутствующих ей элементов. Код в этой папке является библиотечным, т.к. он реализован с помощью такого механизма языка С++, как шаблоны. В это папке содержатся файлы:

Node.h – файл, содержащий в себе реализацию структуры Node. Node описывает то, как устроена каждая вершина стека. Node хранит в себе номер следующей вершины стека в той или иной его версии (next), номер, соответствующий номеру текущей вершины (current), а так же значение, хранящееся в текущей вершине (value).

ActionRequest.h – файл, содержащий в себе реализацию структуры ActionRequest. Структура ActionRequest описывает запрос на изменение к стеку. Структура содержит в себе код операции, которую необходимо выполнить (actionCode), значение, которое должно использоваться при выполнении операции (value) и версия, над которой нужно провести ту или иную операцию (version).

PersistentStack.h – файл, содержащий в себе реализацию класса PersistentStack. Этот класс является той структурой данных, которая визуализируется приложением (реализация – см. 3.2.5.4).

* + - * 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. Lognormal — Логнормальное распределение

5. 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).
        2. out\_port\_ - Observable, по которому осуществляется контакт с геометрической моделью. Через этот порт View отдает геометрической модели сообщения типа MouseState (см. //раздел про MouseState), приходящие от пользователя.
        3. void subscribe (ObserverAction\* obs) – метод, осуществляющий подписку на out\_port\_ (ObserverAction – Observer по MouseState).
        4. ObserverState\* port() – метод, возвращающий указатель на in\_port\_ (ObserverState – Observer по данным, нужным для отрисовки).
        5. get\_action() и send\_action() – методы, осуществляющие контроль за получением запросов на изменение структуры данных.
        6. mouse…(const QPointF& pos) – методы, принимающие точку, в которой произошло взаимодействие пользователя с главным окном приложения и осуществляющие реакцию модуля View на тот или иной вид взаимодействия.
        7. Методы отрисовки различных компонент визуализации.
      1. Реализация класса PersistentStack.
         1. versions\_ - массив версий персистентного стека.
         2. sizesOfVersions\_ - массив, хранящий размер каждой версии стека.
         3. void push(T val, size\_t numOfVersion) – метод, добавляющий в версию стека по номером nuOfVersion элемент со значением val. При добавлении элемента создается новая версия стека и добавляется в конец списка версий. Внутри себя метод с помощью программного прерывания проверяет, существует ли версия, в которую мы хотим добавить элемент.
         4. void pop(size\_t numOfVersion) – метод, удаляющий из версии стека с номером numOfVersion “голову”, т.е. первую вершину. При удалении элемента создается новая версия стека и добавляется в конец списка версий. Внутри себя метод с помощью программного прерывания проверяет, существует ли версия, из которой мы хотим удалить элемент и, если такая версия есть, не пуста ли она.
         5. bool empty(size\_t numOfVersion) – метод, проверяющий, не пуста ли версия стека с номером numOfVersion. Внутри себя метод с помощью программного прерывания проверяет, существует ли версия, которую мы проверяем на пустоту. Метод может вызываться на константном объекте.
         6. RetTypeConst top(size\_t numOfVersion) – метод, возвращающий значение верхушки стека с номером версии numOfVersion. Внутри себя метод с помощью программного прерывания проверяет, существует ли выбранная версия и, если да, то не пуста ли она. Метод может вызываться на константном объекте. Возвращаемое значение может быть представлено либо самим значением, либо константной ссылкой на него. Такой подход нужен для оптимизации при попытке вернуть из стека “тяжелый”, т.е. занимающий большое количество места объект.
         7. size\_t size(size\_t numOfVersion) – метод, возвращающий размер версии стека с номером numOfVersion. Внутри себя метод с помощью программного прерывания проверяет, существует ли выбранная. Метод может вызываться на константном объекте.
         8. size\_t currentNumOfVersion() – метод, возвращающий текущее количество версий стека. Может вызываться на константном объекте.
         9. std::vector<T> get\_version (size\_t version) – метод, возвращающий содержимое стека с номером версии version в виде одномерного массива. Индексы массива показывают порядок того, как элементы лежат в стеке. Внутри себя метод с помощью программного прерывания проверяет, существует ли выбранная версия.
         10. void perform\_action (const ActionRequest<T> request) – метод, применяющий к стеку запрос, описанный в request. Заметим, что запрос может быть только на изменение структуры стека. Если запрашивается удаление элемента, то в качестве параметра value может быть передано любое число, это значение просто проигнорируется.
         11. Оператор вывода стека в заданный поток вывода.
      2. Реализация класса PersistentStackModel.
         1. Объект типа PersistentStack<int> (тип данных, хранимый в стеке, не играет значения в контексте визуализации).
         2. port\_ - Observabvle по PersistentStack<int>. С помощью этого порта осуществляется обмен сообщениями между Model и GeomController (см. 3.2.2.1.3.5).
         3. void subscribe (ObserverStack\* obs) – метод, осуществляющий подписку на port\_ (ObserverStack – Observer по PersistentStack<int>).
         4. void make\_action(const ActionRequest<int>& request) – метод, который позволяет совершать на объектом типа PersistentStack<int>, хранящемся в классе PersistentStackModel действия по соответствующим запросам. Запросы поступают через port\_.
      3. Реализация класса PersistenStackGeomModel.
         1. in\_port\_ - Observer по PersistentStack<int>
         2. port\_ - Observable по DrawData (см. 3.2.4.1.3.6).
         3. Конструктор, инициализирующий in\_port\_;
         4. void subscribeToAction(ObserverAction\* obs) и void subscribeToData(Observer\* obs) – методы, осуществляющие подписки по ActionRequest (см. 3.2.2.1.2.2.) или PersistentStack<int> соответственно.
         5. ObserverStack\* port () – метод, возвращающий указатель на in\_port\_.
         6. void handleAction(ActionRequest<int>& request) – метод, реагирующий на поступивший запрос на изменение персистентного стека и отправляющий этот запрос дальше в PersistentStackModel.
         7. void onStackData (const PersistentStack<int>& data) – метод, использующий для инициализации in\_port\_ в конструкторе.
  1. **Организация входных данных**

Основное взаимодействие пользователя с приложением происходит через GUI, то есть визуальный интерфейс. Входными данными для приложения являются нажатия пользователя на кнопки, соответствующие операциям, совершаемым над стеком (см. Приложение 2, элементы 3-5). При нажатии на кнопку, отвечающую за добавление элемента (см. Приложение 2, элемент 3) в стек появляется диалоговое окно, в которое нужно ввести значение, которое нужно добавить в ту или иную версию стека. Заметим, что для визуализации используется структура данных PersistentStack<int> входящее значение, имеющее строковый тип, должно быть корректным для приведения к целочисленному знаковому типу. Корректной является входная строка, содержащая исключительно цифры от 0 до 9 или знак “-” (минус), причем если присутствует минус, он должен быть в начале строки, а первая цифра в записи не должна быть нулем. Также вводом может быть выбор текущей версии с помощью выпадающего списка, расположенного на главном окне приложения.

* 1. **Организация выходных данных**

Выходными данными для приложения является визуализация структуры данных PersistentStack<int>. Визуализация персистентного стека определяется реализацией модуля View, так как именно этот модуль получает и может воспользоваться данными в структуре DrawData. Визуализация стека, помимо особенностей, связанных с реализацией модуля View, включает в себя отображение значения вершины стека, введенной в диалоговое окно (см. 3.2.3).

## 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**

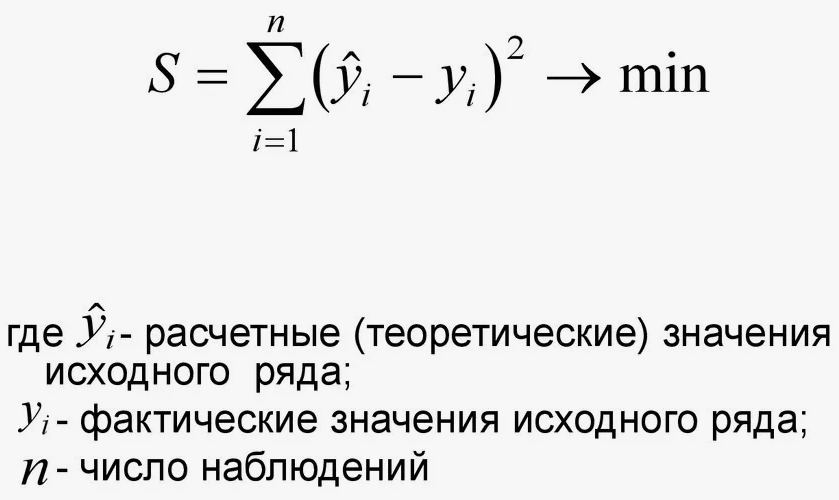
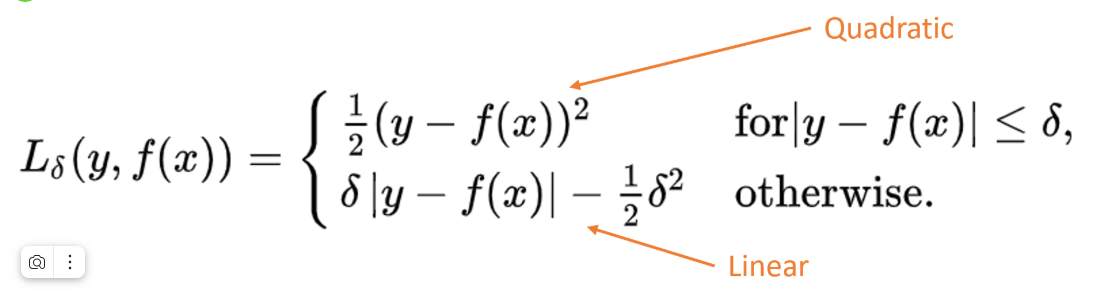


Рис.1 Функция потерь МНК

**ПРИЛОЖЕНИЕ 3**

Рис.2 Функция потерь Хьюбера

**ПРИЛОЖЕНИЕ 4**

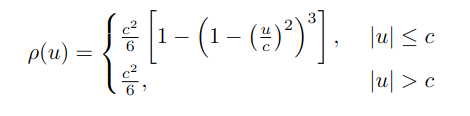


Рис.3 Функция потерь Тьюки

**ПРИЛОЖЕНИЕ 5**

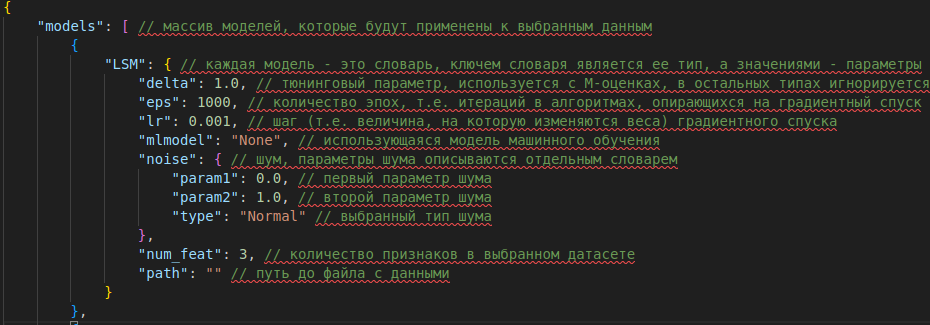
****

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

**ПРИЛОЖЕНИЕ 6**

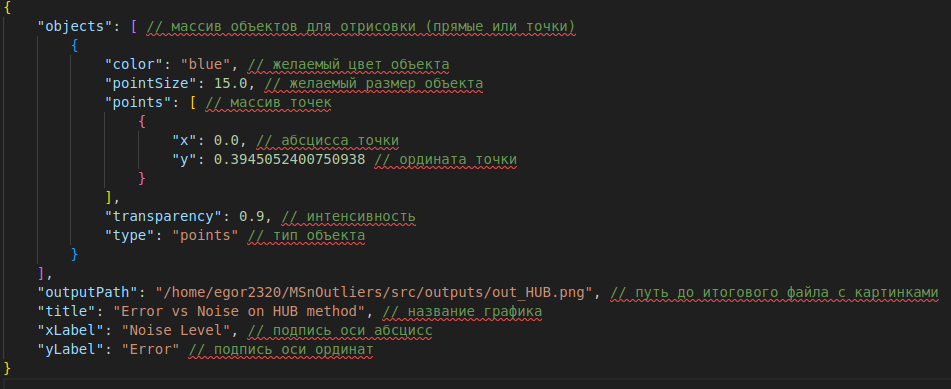
****

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

**ПРИЛОЖЕНИЕ 7**

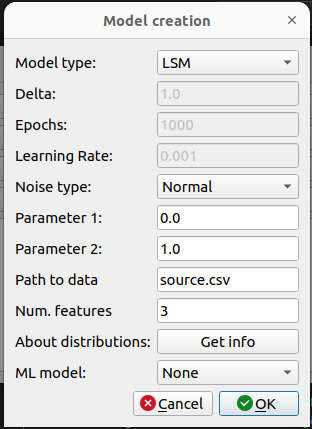
****

Рис.6 Экран создания модели (ModelDialog)

**ПРИЛОЖЕНИЕ 8**

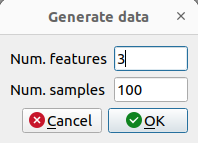
****

Рис.7 Экран генерации данных (GenerateDialog)

## 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 |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |