МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки / специальности)

Мобильные и сетевые технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

на тему: «Разработка мобильного приложения для решения задачи условной оптимизации, при бизнес-планировании»

Обучающийся Н.Р. Доронин _____
(Инициалы Фамилия) (личная подпись)

Руководитель С.В. Митин
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант кандидат педагогических наук, С. А. Гудкова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия

Тольятти 2024

Аннотация

В данной выпускной квалификационной работе проведена разработка мобильного приложения решающего задачу условной оптимизации.

Выполненная квалификационная работа состоит из введения, теоретической части, представленной четырьмя главами, практической части, представленной шестью главами, заключения, списка используемых источников

Во введение показана актуальность разрабатываемого приложения, поставлены цели и задачи для получения готового приложения.

В теоретической части разобраны применяемые в работе приложения алгоритмы и методы, выполнено проектирование интерфейса.

В практической части разобрана реализация алгоритмов и методов на языке python, проведена оценка их работоспособности, разработан пользовательский интерфейс приложения и рассмотрено применение приложения в решении целевой задачи.

В заключении подведены итоги проделанной работы.

Abstract

In this final qualifying work, the development of a mobile application solving the problem of conditional optimization was carried out.

The completed qualification work consists of an introduction, a theoretical part presented in four chapters, a practical part presented in six chapters, a conclusion, a list of sources used and an appendix.

The introduction shows the relevance of the application being developed, sets goals and objectives for obtaining a ready-made application.

In the theoretical part, the algorithms and methods used in the application are analyzed, and the interface is designed.

In the practical part, the implementation of algorithms and methods in python is analyzed, their performance is evaluated, the user interface of the application is developed and the application is considered in solving the target problem.

In conclusion, the results of the work done are summarized.

Оглавление

Введение

Современные мобильные устройства получают большие вычислительные мощности и больший объем памяти, в этих условиях разработка приложений для мобильных устройств более востребована чем раньше. Следуя этой тенденции в этой работе будет разрабатываться нативное мобильное приложение помогающее в бизнес планировании.

Большинство современных мобильных приложений, для бизнес планирования, не нативные, например как инструменты для бизнеса сбербанка. Преимущества таких приложений заключается в том что данные, используемые в этих приложениях, синхронизируются на разных устройствах и хранятся на серверах организаций, обеспечивающих их безопасность. Но такие приложения не будут работать без стабильного соединения с сетью интернет.

Преимущества нативных приложений заключается в том что эти приложения будут работать в любом месте и будут задействовать полную мощность вычислительных устройств будет задействована полностью, в отличии от вычислительных мощностей веб приложений, задаваемых организацией владеющей серверами приложения.

Целью работы является создание мобильного приложения, решающего задачу условной оптимизации и выдающее результаты решения задачи в сравнение с прогнозируемыми результатами спроса.

Рассматриваемым в работе объектами являются задача условной оптимизации и задача прогнозирования спроса на продукцию, а рассматриваемым предметом работы является решение этих задач таким образом, при котором оно может быть интегрировано в разрабатываемый программный продукт.

Для достижения цели необходимо решить ряд задач: сформулировать методы решения задач, спроектировать интерфейс приложения, разработать и оценить работоспособность приложения.

Разрабатываемое приложение будет выполнять следующую функции: оно будет предсказывать спрос на продукцию в определенном месяце и определять сколько, из имеющихся на складе запасов, можно произвести продукции, в соотношении дающем максимальную прибыль. Сравнение этих значений позволит скорректировать запасы на складе.

Помимо решения основной задачи приложение также должно позволять работать с, хранящимися на устройстве, данными: удалять и добавлять записи таблиц.

Тут следует зафиксировать то какие данные будут использоваться в приложении. Очевидно, что для работы приложения потребуется хранение и обработка списков продуктов и ресурсов. В таблице ресурсов потребуется параметр отвечающий за запасы ресурса на складе. Для предсказания спроса на продукцию потребуется таблица с историей продаж, а для решения задачи условной оптимизации потребуется таблица затрат ресурсов на производство продукции.

Помимо работы с данными и решения задачи приложение потребует дополнительную функцию. Пользователи привыкли работать с графиками и диаграммами больше чем с таблицами, поэтому для большей наглядности в приложении в качестве отчета должна формироваться гистограмма на основе полученных решений. Это поможет пользователю быстрее сориентироваться и облегчит для него сравнения полученных решений на каждом продукте.

Таким образом появляется представление о назначении приложения, что позволит начать проектировать экранные формы в процессе детального разбора его функционирования.

Глава 1. Анализ задачи

1.1 Анализ потребителя программного продукта

Прежде чем приступать к проектированию или разработке программного продукта необходимо определиться с тем кто будет его использовать.

Разрабатываемый продукт предназначен для небольших предприятий, занимающихся производством и реализацией продукции, например продукт подойдет цветочным магазинам, в цветочном магазине продукция (букеты цветов) собираются из доставленных ресурсов (коробок с цветами, оберточной пленки, ленты, …) и продаются в этом же магазине потребителю продукции.

В случае с цветочным магазином важно знать какой спрос будет на тавры в следующем месяце и какую максимальную прибыль можно получить на основе запасов к следующему месяцу, ожидаемый спрос на продукцию интересно сравнить с количеством продукции, разных видов, которое дает максимальную прибыль.

Разработка программы дающей понимание ожидаемого спроса и лучшего количества производимой, из имеющихся ресурсов, продукции и будет цель всей далее проделанной работы.

Такая программа интересна не только для цветочных магазинов но и для многих других предприятий, поэтому необходимо четко поставить какие функции будут у программного продукта, так чтобы продукт получился более универсальным.

1.2 Функционал разрабатываемого программного продукта

Для того чтобы с программным продуктом можно было работать надо четко определить его функции.

Во первых в задаче используются разные данные: запасы ресурсов на складе, затраты ресурсов на производство продукции, продукция и история её продаж, поэтому в программном продукте должна быть возможность внесения, хранения и удаления этих данных.

Во вторых должна быть форма для проведения расчетов. Оба продукта потребуют список рассматриваемых продуктов. Для решения задачи прогнозирования необходимо получить данные истории продаж. Для решения задачи условной оптимизации понадобятся: запас ресурсов на складе, актуальная цена продукции, таблица расхода ресурсов на продукцию. Решение задач должно выводиться таблично по каждому продукту.

В третьих табличный вывод решения будет в более наглядной форме если преобразовать его в гистограмму, поэтому необходима дополнительная форма с гистограммой по полученному решению.

Таким образом функции программного продукта определены, можно приступать к его проектированию.

Глава 2. Логическое проектирование программного продукта

2.1 Проектирование интерфейса пользователя

Для того чтобы понять с какими экранными формами необходимо будет работать необходимо разобраться в том что в этих экранных формах будет отображено. Экранная форма - это настроенное визуальное представление, содержащее поля для ввода и элементы управления [7]. Для этого необходимо рассмотреть модель сущность-связь (ERD), показанную на следующем рисунке.

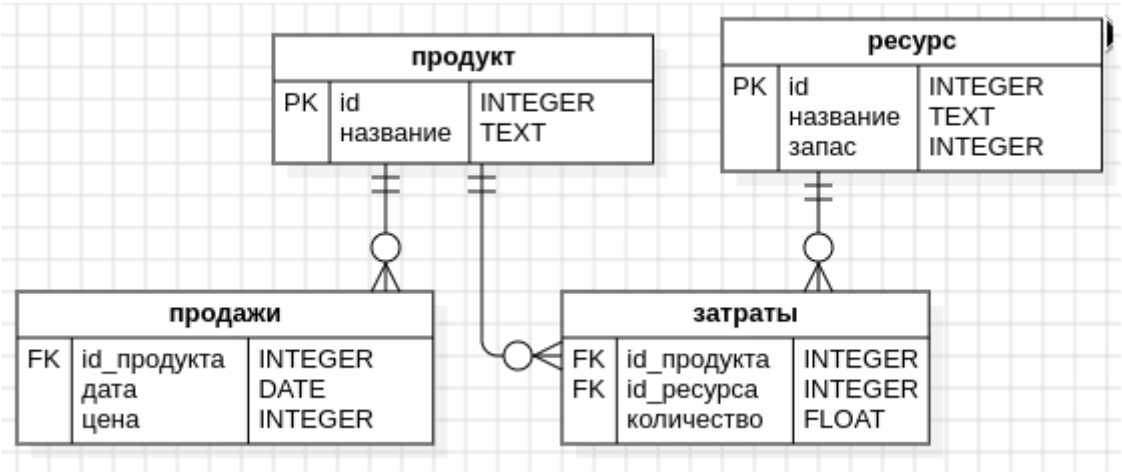

Рисунок 1 - ERD модель

На рисунке 1 показана модель сущность-связь в нотации Мартина, соответствующая тем данным которые будут задействованы в разрабатываемом приложении.

В этой модели показаны четыре сущности: продукт, ресурс, продажи и затраты, соответствующие обрабатываемым таблицам данных. У каждой сущности есть набор колонок, соответствующим колонкам таблиц. Для каждой колонки есть тип её значений, записываемый справа от названия колонки, и метка внешнего или внутреннего ключа записываемая слева.

Линиями показаны связи между сущностями, причем концы линий символизируют количество сущностей задействованных в связи: две черты —

одна сущность, круг — много сущностей. Так например один продукт может быть задействован во многих сделках о продаже, но не наоборот.

Рассмотрев модель сущность-связь стало понятно что в интерфейсе приложения потребуется отдельная экранная форма для отображения элементов и добавления новых элементов в каждой из таблиц. Кроме этого потребуются экранные формы для отображения решения и отчета а так-же отдельная форма для выбора продуктов по которым будет произведен расчет. Это сделано для того чтобы упростить вычисления в ситуациях когда необходимо исключить продукты из решения.

Подводя итог описания экранных форм можно получить следующий эскизный проект экранных форм приложения.

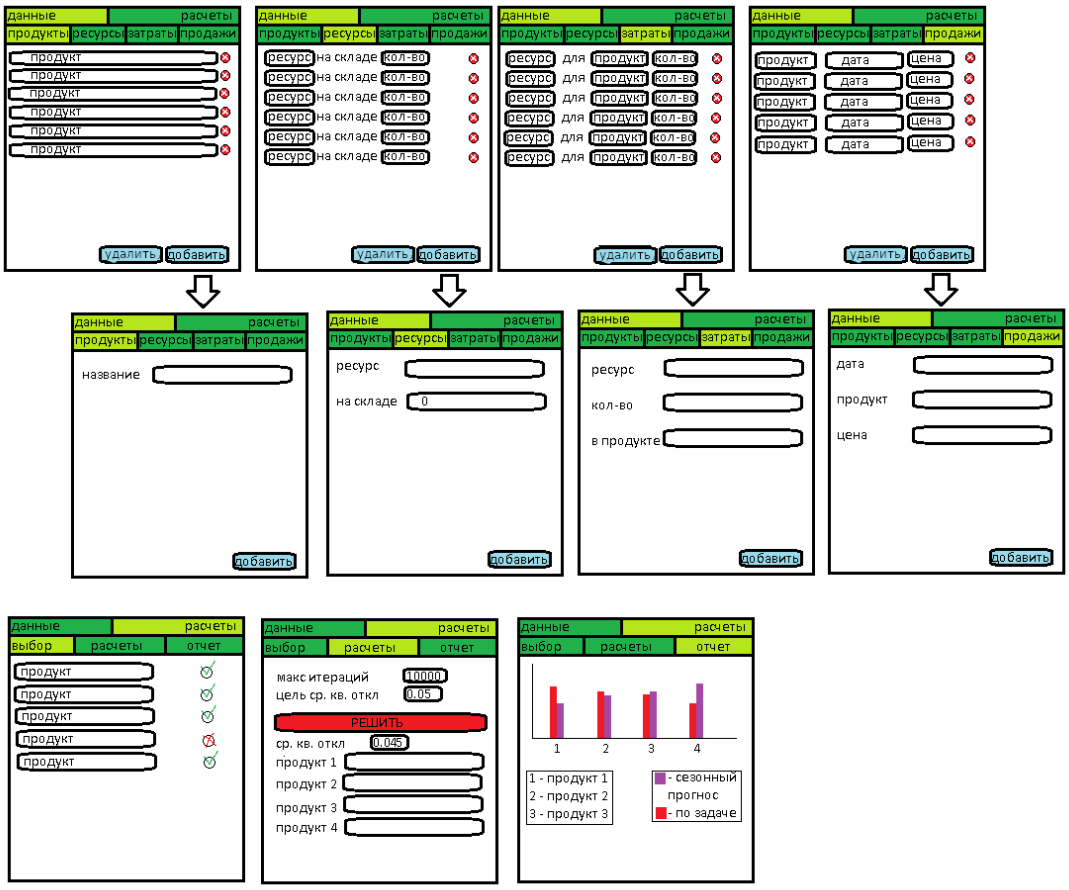

Рисунок 2 - Эскизный проект для экранных форм

На рисунке 2 показаны все ранее описанные одиннадцать экранных форм. Рассматривая эти формы можно выделить ряд интерфейсных решений

которые будут задействованы в проекте. Во первых можно выделить интуитивно понятное цветное меню в верхней части формы, которое присутствует на каждой экранной форме. С помощью этого меню можно легко осуществлять переход по семи формам и это будет отображено цветом кнопок меню. Дополнительно семь кнопок, которые было бы сложно отобразить в одну линию на экране телефона, разделены по двум группам: для того что связано с обработкой данных и того что связано с вычислениями. При работе с данными добавление происходит переходом на отдельную экранную форму через кнопку добавить, а удаление происходит в 2 этапа: установка флажков для определенных записей и нажатие соответствующей кнопки.

Отдельно стоит выделить что элементы расположены по прямой проходящей слева на право, сверху в низ, что рекомендоваться для интерфейса мобильных устройств.

Разобравшись в структуре данных и тем как будет осуществляться взаимодействие с приложением через его интерфейс можно перейти к рассмотрению того как будет решаться задача. Для начала будет необходимо решить задачу предсказания нового элемента в числовой последовательности. Числовой последовательностью называется упорядоченный набор пронумерованных чисел [1].

2.2 Предсказание значения в последовательности

Перед тем как приступать к решению следует проанализировать данные. Так-как в приложении фигурируют продажи следует учесть что спрос потребителя на продукцию обычно зависит от определенных ежегодных событий. Например цветочный магазин чаще всего посещают по праздникам, а в зимнее время года продажи в цветочных магазинах резка падают. Поэтому для того чтобы не создавать сложные предсказательные модели учитывающие сезонность спроса множество исторических данных будет выбираться по данным заданного месяца. Множество - набор, совокупность каких-либо

объектов - элементов этого множества [2]. Тогда определить новое значение последовательности можно будет нахождением общего тренда последовательности.

Для того чтобы найти общий тренд последовательности A необходимо сначала найти разность соседних элементов последовательности.

$$y_j = a_{j+1} - a_j, j = \overrightarrow{0, n-1} \quad (1)$$

где a_j — элементы входной последовательности A;

y_i — разности соседних элементов входной последовательности;

n — кол-во элементов в входной последовательности.

Затем по полученной новой последовательности (1) строиться линейная регрессия. Линейная регрессия — используемая в статистике регрессионная модель зависимости одной переменной y от одной или нескольких других переменных x с линейной функцией зависимости [5]. Далее линейная регрессия (2) будет вычислена по методу наименьших квадратов, для этого достаточно посчитать параметры линейной регрессии (3.1), (3.2) подставив вместо y разность соседних элементов, а вместо x множество индексов этих элементов

$$f(x) = b_0 + b_1 x \quad (2)$$

$$b_0 = \frac{\sum_{i=1}^{n} y_i * \sum_{i=1}^{n} i^2 - \sum_{i=1}^{n} i * \sum_{i=1}^{n} i y_i}{n * \sum_{i=1}^{n} i^2 - \left(\sum_{i=1}^{n} i\right)^2} \quad (3.1)$$

$$b_1 = \frac{n * \sum_{i=1}^{n} i y_i - \sum_{i=1}^{n} i * \sum_{i=1}^{n} y_i}{n * \sum_{i=1}^{n} i^2 - \left(\sum_{i=1}^{n} i\right)^2} \quad (3.2)$$

где $f(x)$ — получаемое уравнение регрессии с коэффициентами b_0, b_1;

x — момент времени, в который необходимо сделать прогноз.

Полученная таким образом линейная регрессия будет описывать общий тренд изменений всего множества. Последнее значение разность соседних элементов может быть найдено подстановкой n в функцию линейной

регрессии. Добавление новой разности элементов к последнему элементу последовательности даст следующий элемент этой последовательности.

Таким образом к последовательности с выраженной общей тенденцией может быть добавлен прогнозированный новый элемент, пример добавленного к последовательности элемента показан на следующем рисунке.

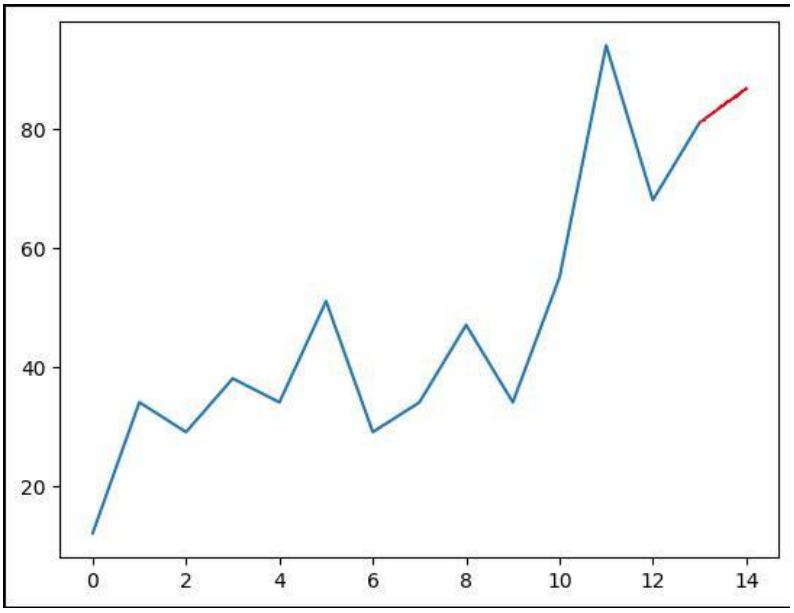

Рисунок 3 - Пример прогноза значения элемента

Например рассматривая рисунок 3 в последовательности можно заметить общую тенденцию к увлечению значений в последовательности, что и показано при добавлении в неё нового элемента по этому методу.

Новый элемент обозначен на рисунке красным цветом. Его значение больше значения предыдущего элемента последовательности, что соответствует общему тренду на увлечение новых значений элементов последовательности.

Разобравшись в том как работает прогноз новых значений можно перейти к решению задачи условной оптимизации.

Задача об отыскании об отыскании предельного значения заданной функции, определенной на заданном множестве называется задачей условной оптимизации [8].

Далее рассмотрим решение задачи условной оптимизации, эта задача будет применяться в расчете значения количества продукции дающего максимальную прибыль на заданных запасах ресурсов.

2.3 Решение задачи условной оптимизации

В самой простой форме задача условной оптимизации может быть записана следующим образом.

$$f(x_1 \dots x_n) = PX \rightarrow max \tag{4}$$

$$G(X) = \begin{cases} A_1 X \leq b_1 \\ \dots \\ A_{m+n} X \leq b_{m+n} \end{cases} \tag{5}$$

где $x_1 \dots x_n = X$ — выбранное количество производимой продукции в соответствии с индексами продуктов;

P — актуальная цена продукции в виде вектора, с соответствием индексам продуктов;

A — расширенная матрица затрат ресурсов на производство продукции;

B — приведенный к матрице A вектор запасов ресурсов на складе.

Рассматривая минимальную задачу выделяется функция оптимальности решения (4) и система ограничений (5). Следует учитывать что функция (4) — это функция от многих переменных — скалярная функция векторного аргумента размерности n [11]. Ограничениями в задаче условной оптимизации принято считать систему линейных неравенств, решение задачи которых должно выполнять [3]

Обычно эту задачу используют для нахождения оптимального количества производимой продукции X, дающего прибыль f(X) и ограниченное запасами ресурсов b при матрице расхода ресурсов A. Дополнительно задача ограничена на минимум производимых продуктов. [12]

Простейшим способом решения этой задачи является генетический метод. Генетический алгоритм является одним из методов случайного поиска

[10]. Случайны поиск - .методы поиска решения задачи путем случайного перебора пространства решений [14]. Суть этого метода заключается в формировании множества решений, сортируемых в соответствии с целевой функцией. Новый элемент множества формируется из двух других его элементов путем формирования случайной выборки параметров этих элементов с добавлением случайной мутации [13].

$$X_{l+1}^{g} = x_{i+k_{1v2},1}^{g} + mutation_{?}, \ldots, x_{i+k_{1v2},n}^{g} + mutation_{?}. k_1, k_2 = ? \qquad (6)$$

где $mutation_{?}$ — случайный выбранный элемент из вектора размеров мутации;

l — размер популяции в момент добавления нового элемента;

k_1, k_2 — случайное смешение вниз по популяции $k_1 < k_2$;

X^{g} — популяция полученных во время работы алгоритма решений.

Полученные новые элементы (6) дописываются на свое место, определенное функцией. До тех пор пока среднее квадратическое отклонение не достигнет целевого уровня или не будет превышено время жизни алгоритмы. Мутации это множество элементов сплюснутой параболы (7).

$$mutation_x = \begin{cases} 0.001x^4 |x| \geq 0.5 \\ 0, |x| < 0.5 \end{cases}, x \in [-1,1] \qquad (7)$$

где $mutation_x$ — множество из мутаций заданных параболой, обнуленной в центре.

Этот метод хорошо подходит для решения задачи условной оптимизации, потому что из-за линейности ограничений формируется область решений без локальных максимумов. Двигаясь вдоль ограничений формируемое множество всегда придет в область решения задачи.

В зависимости от начальной выборки этот алгоритм может работать очень долго поэтому ему может потребоваться поддержка дополнительных методов для формирования начальной выборки. Обычно начальная выборка может быть легко сформирована из значений вдоль вектора градиента целевой функции, но часто решение может находиться далеко от места пересечения

вектора и границы области решений, поэтому требуется быстро получить приближенное решение и использовать генетический метод начиная от приближенного решения.

На этом этапе следует учесть что чаще всего пользователь будет считать уже решенную до этого задачу с небольшими изменениями, поэтому для формирования начальной выборки можно использовать переобученные модели, которые стремятся выдать прошлые результаты хотя бы для части параметров, которые будут задействованы при генетическом решении.

Такой, переобученной, моделью может быть самая обычная нейронная сеть. Эта модель не даст близких результатов на совершенно новых данных, но если часть входных параметров будет неизменной то она может вычислить часть выходных правильно.

Прежде чем приступать к работе с нейронной сетью надо подготовить правильные данные. Нейронная сеть работает на фиксированном количестве входных и выходных данных, а в задаче количество и продуктов и ресурсов может быть разным. Для того чтобы решить эту проблему надо дополнить данные пустыми продуктами и ресурсами, которые не повлияют на решение. Преобразованная задача будет выгладить следующим образом.

$$f(0 \dots x_1 \dots x_n) = PX \rightarrow max \qquad (4^*)$$

$$G(X) = \begin{cases} 0X \leq 0 \\ \dots \\ A_1 X \leq b_1 \\ \dots \\ A_{m+n} X \leq b_{m+n} \end{cases} \qquad (5^*)$$

Таким образом это не повлияет на область решений и результаты. Когда была решена группа задач и когда эта группа была выровнена нулевыми значениями можно приступать к формированию нейронной сети. Для решения задачи подойдет сеть, слои которой показаны на следующей схеме.

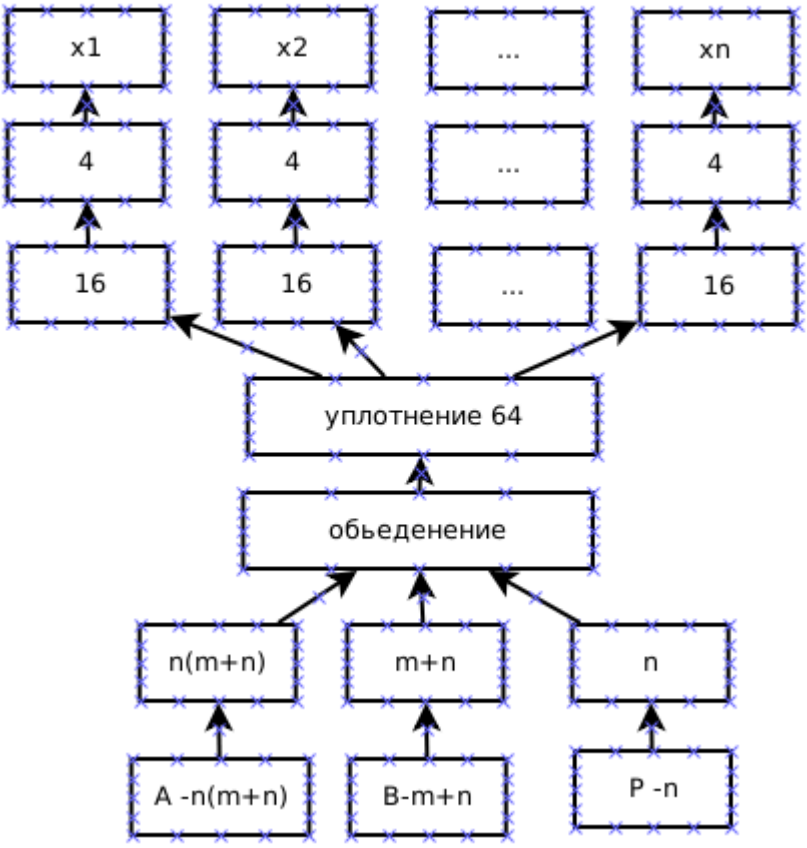

Рисунок 4 - Структура сети

На рисунке 4 показана возможная реализация слоев подходящей нейронной сети. На практике эта структура нейронной сети представляет собой n нейронных сетей с общими входными слоями [15]. В качестве метрики и функции потерь для этой сети можно использовать среднюю квадратическую ошибку и при ограниченном наборе данных для обучения будет получен эффект от работы переобученной модели, что ускорит решение похожих задач генетическим алгоритмом.

На вход этой сети идет вектор из значений входных данных задачи. Как и было указано ранее входные значения меньшей задачи приводятся к необходимому масштабу путем записывания перед значениями нулей.

Закончив описывать все используемые методы решения задач их можно реализовать программно и интегрировать в приложение.

2.4 Используемые в разработке инструменты

Так как работа предполагает обработку больших данных и работу с нейронными сетями алгоритмы стоит реализовать на языке программирования python.

Python - высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью[16]. Важным фактором в выборе Python является то что он пор тирован и работает почти на всех известных платформах [9]

Выбрав python в качестве языка для разработки, интерфейс андроид приложения можно реализовать при помощи фреймворка kivy. Kivy - это фреймворк, разработанный организацией Kivy, наряду с Python для Android, Kivy для iOS, и несколькими другими библиотеками, предназначенными для использования на всех платформах. Kivy — свободное программное обеспечение с открытым исходным кодом. [19]

Для того чтобы преобразовать kivy приложения в apk файл также понадобится buildozer. Buildozer - это инструмент, предназначенный для простой упаковки мобильных приложений. Он автоматизирует весь процесс сборки, загружает необходимые компоненты, такие как python для Android, Android SDK, NDK. [17]

Выбрав инструменты можно приступать непосредственно к реализации.

Глава 3. Реализация разработанного программного продукта

3.1 Реализация предсказательного алгоритма

Для того чтобы реализовать предсказание следующего элемента последовательности сначала необходимо реализовать метод для получения коэффициентов (3.1), (3.2) линейной регрессии (2) по методу наименьших квадратов.

$$y_j = a_{j+1} - a_j, j = \overrightarrow{0, n-1} \quad (1)$$

$$f(x) = b_0 + b_1 x \quad (2)$$

$$b_0 = \frac{\sum_{i=1}^{n} y_i * \sum_{i=1}^{n} i^2 - \sum_{i=1}^{n} i * \sum_{i=1}^{n} i y_i}{n * \sum_{i=1}^{n} i^2 - \left(\sum_{i=1}^{n} i\right)^2} \quad (3.1)$$

$$b_1 = \frac{n * \sum_{i=1}^{n} i y_i - \sum_{i=1}^{n} i * \sum_{i=1}^{n} y_i}{n * \sum_{i=1}^{n} i^2 - \left(\sum_{i=1}^{n} i\right)^2} \quad (3.2)$$

Программная реализация вычисления коэффициентов будет выглядеть следующим образом в соответствии с уравнениями (3.1), (3.2). Далее идет фрагмент кода с алгоритмом расчета коэффициентов уравнения регрессии.

```
def lr(x,y):
    s = [0,0,0,0,0,0]
    n = len(x)
    for i in range(0, n):
        s[0]+=y[i]
        s[1]+=x[i]*x[i]
        s[2]+=x[i]
        s[3]+=x[i]*y[i]
    return ((s[0]*s[1])-(s[2]*s[3]))/((n*s[1])-(s[2]*s[2])), ((n*s[3])-(s[0]*s[2]))/((n*s[1])-(s[2]*s[2]))
```

Рисунок 5 - Коэффициенты линейной регрессии

На рисунке 5 показана функция расчета коэффициентов регрессии. Далее необходимо рассчитать разность соседних элементов, для этой разности найти общий тренд, нахождением линейной регрессии по методу наименьших квадратов и получить следующее значение последовательности на основе общего тренда. Реализация этого алгоритма представлена на следующей картинке.

```
def predict(Y):
    trend = [Y[i+1]-Y[i] for i in range(0,len(Y)-1)]
    b0, b1 = lr(x=[i for i in range(0,len(trend))], y=trend)
    f = lambda x: b1*x+b0
    return Y[len(Y)-1]+f(len(trend))
```

Рисунок 6 - предсказание следующего значения последовательности

На рисунке 6 показана функция для расчета нового значения последовательности. Применение этого алгоритма можно рассмотреть на ранее иллюстрированной последовательности.

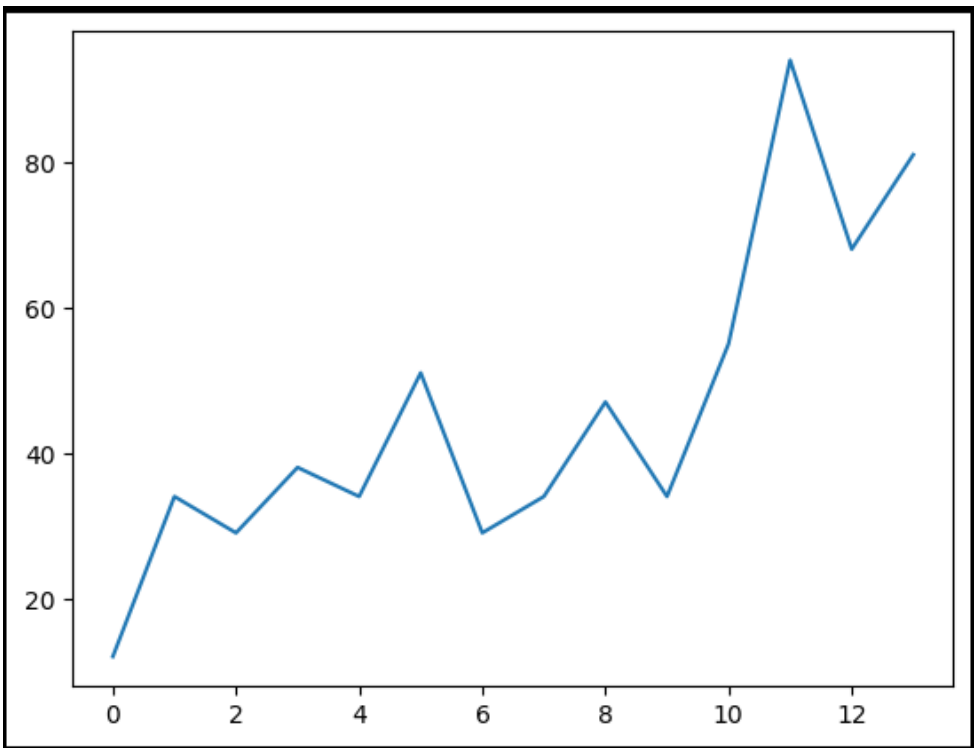

Рисунок 7 - начальная последовательность

На рисунке 7 показан график на основе значений в последовательности значений созданной для демонстрации работы алгоритма. Сначала для этой последовательности необходимо найти разность соседних элементов, которая показана следующей картинкой

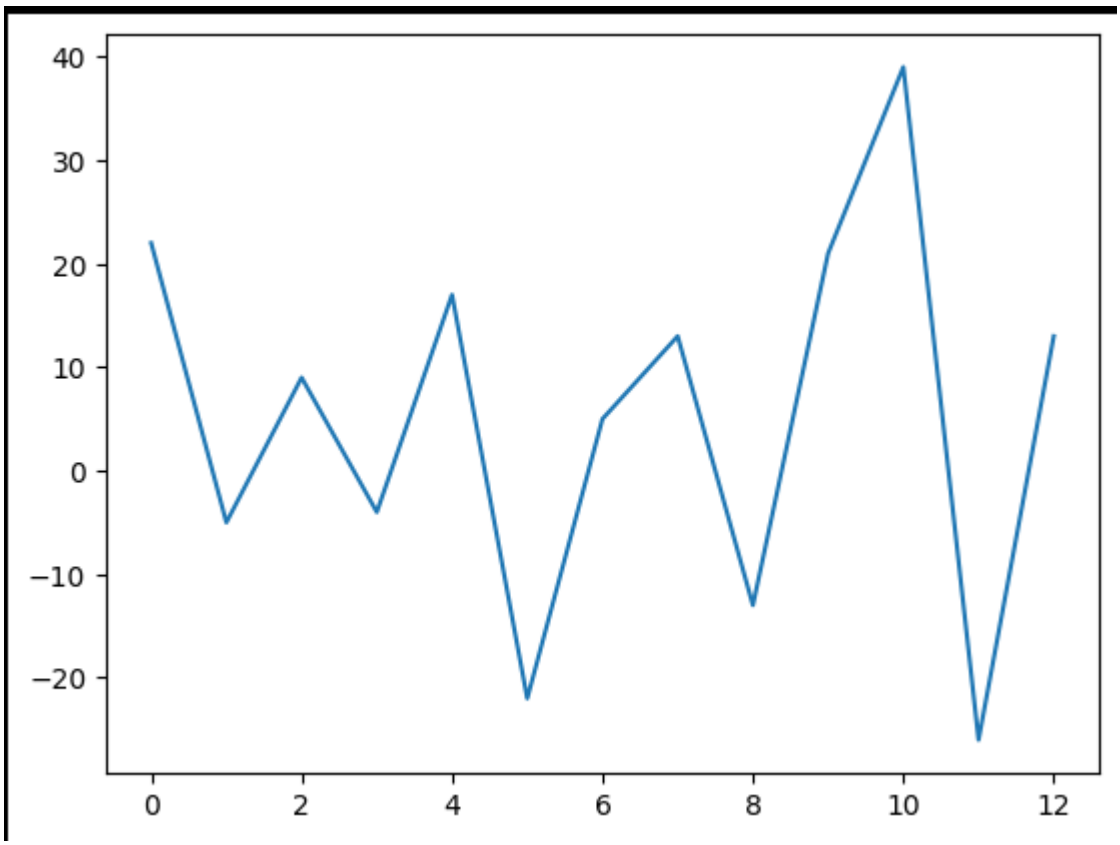

Рисунок 8 - Разность соседних элементов

На рисунке 8 показана новая последовательность на основе локальных тренеров при изменении значений каждой пары элементов подаваемой последовательности. Для этой новой последовательности необходимо найти и построить уравнение линейной регрессии.

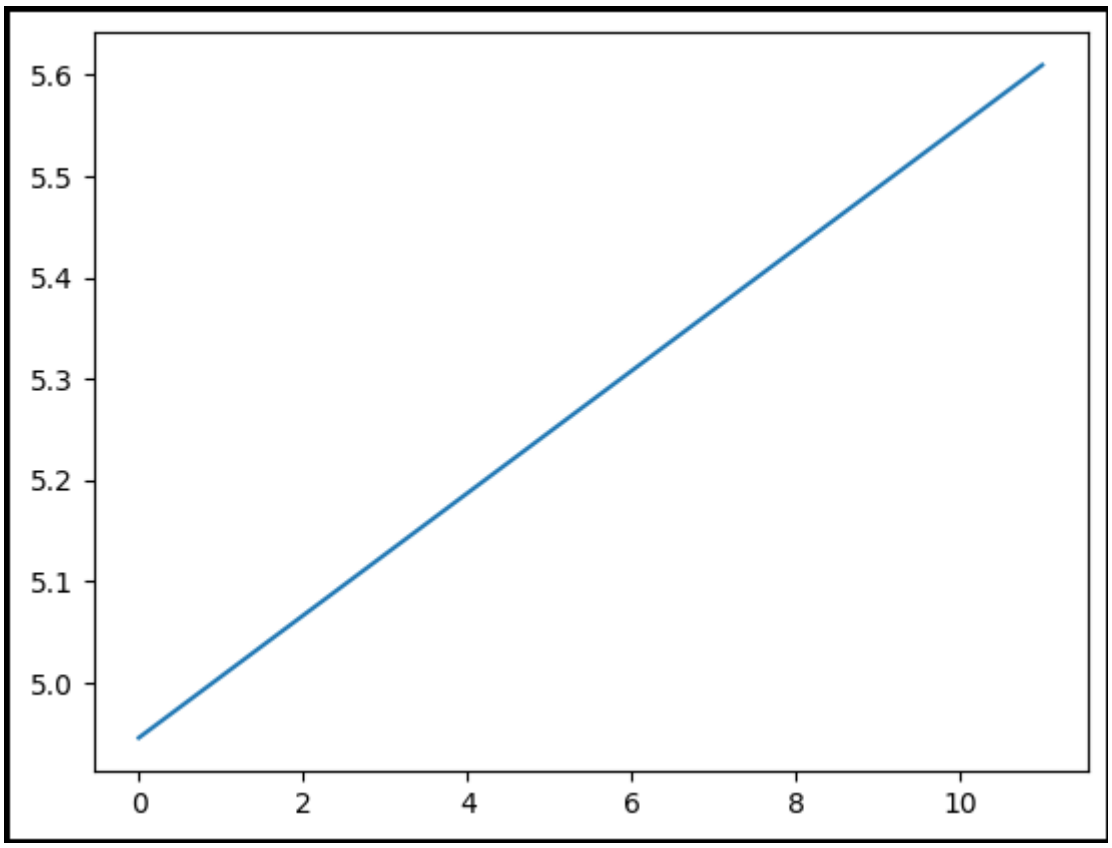

Рисунок 9 - Линейная регрессия

На рисунке 9 показана линия общего тренда для всех изменений значений в последовательности.

Для разности соседних элементов эта линейная регрессия отражает общий тренд изменения значений последовательности.

Рассчитав значение нового элемента, с помощью уравнения линейной регрессии можно получить значение нового элемента последовательности.

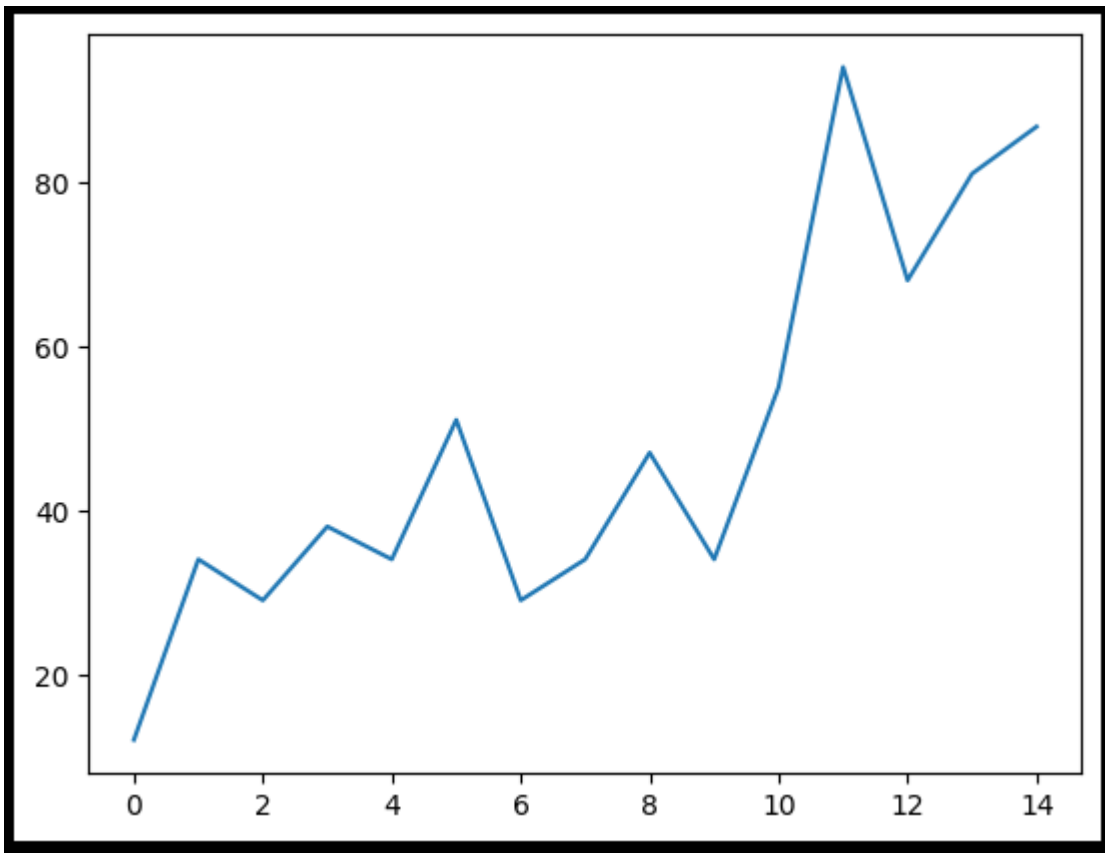

Рисунок 10 - Новое значение последовательности

На рисунке 10 показан новый добавленный, крайний правый, элемент на подаваемой на вход последовательности, новое значение которого необходимо было предсказать.

Рассматривая значение нового элемента последовательности он действительно соответствует общему тренду изменения значений в последовательности.

Также следует учесть что использование этого метода для нахождения большего количества элементов на основе добавленных, будет линейно, и может привести к расхождению полученных и фактических значений.

Таким образом можно приступать к реализации следующих алгоритмов.

3.2 Реализация генетического алгоритма

Для того чтобы реализовать генетический алгоритм достаточно знать принцип его работы:

-Формируется список начальных значений. Например список может быть сформирован элементами вдоль вектора градиента, целевой функции.

-Элементы списка сортируются по значению целевой функции.

-Выбираются 2 элемента: лучший и любой другой элемент

-На основе полученных элементов формируется новый элемент и добавляется в список.

-Если не превышено заданное количество итераций и среднее квадратическое отклонение верхних элементов списка превышает целевое то возвращаемся к шагу 2

-Среди получены решений в списке выбирается самое верхнее и принимается за лучшее решение

$$X_{i+1}^{g} = x_{i+k_{1v2},1}^{g} + mutation_{?}, \ldots, x_{i+k_{1v2},n}^{g} + mutation_{?}. k_1, k_2 = ? \qquad (6)$$

$$mutation_x = \begin{cases} 0.001x^4 |x| \geq 0.5 \\ 0, |x| < 0.5 \end{cases}, x \in [-1,1] \qquad (7)$$

Этот алгоритм будет иметь следующую программную реализацию, показанную следующим фрагментом кода.

```python
mutation_range = [0.08*pow(x/100,4) if abs(x)>0.5 else 0. for x in range(-100,100)]

def genetic(A,B,P,values,ttl,alpha,p,m):
    n = len(P)
    m = len(B)
    def G(X):
        result = 1.
        for j in range(0,m):
            sum = 0.
            for i in range(0,n):
                sum+=A[j][i]*X[i]
            result*=int(sum<=B[j])
            if result==0:
                break
        return result
    def F(X):
        result = 0.
        if G(X)!=0:
            for i in range(0,n):
                result+=P[i]*X[i]
        return result

    for i in range(0,len(values)):
        values[i].insert(0,F(values[i]))
    while(ttl>0)and(alpha<abs(sum([values[i][0]*values[i][0]-values[i+1][0]*values[i+1][0] for i in range(0,4)])/5)):
        ttl-=1
        values = values[:m]
        values.sort(key=lambda x: x[0], reverse=True)
        if -values[1][0]+values[0][0]<alpha:
            break
        for i in range(0,p):
            j = choice(range(i+1,len(values)))
            new_value = [choice([values[i][k],values[j][k]])+choice(mutation_range) for k in range(0,n)]
            new_value.insert(0,F(new_value))
            values.append(new_value)
    values.sort(key=lambda x: x[0], reverse=True)
    return values[0:p]
```

Рисунок 11 - Реализация генетического алгоритма

На рисунке 11 показана реализация генетического алгоритма для выполнения расчетов. Реализованный алгоритм отличается от описанного тем что на каждой итерации формирования новых элементов по формуле (6) осуществляется для лучших p элементов с другими случайными элементами.

Далее необходимо метод протестировать, для этого можно рассмотреть задачу условной оптимизации в её геометрической форме, где ограничения отображаются с помощью барьерных линий или плоскостей, а решение в виде точки в области решения задачи [6].

Применение этого алгоритма может быть иллюстрировано на задаче с двумя продуктами. Пунктирными линиями будут обозначены различные вводимые барьеры, а красными точками различные решения задействованные в алгоритме.

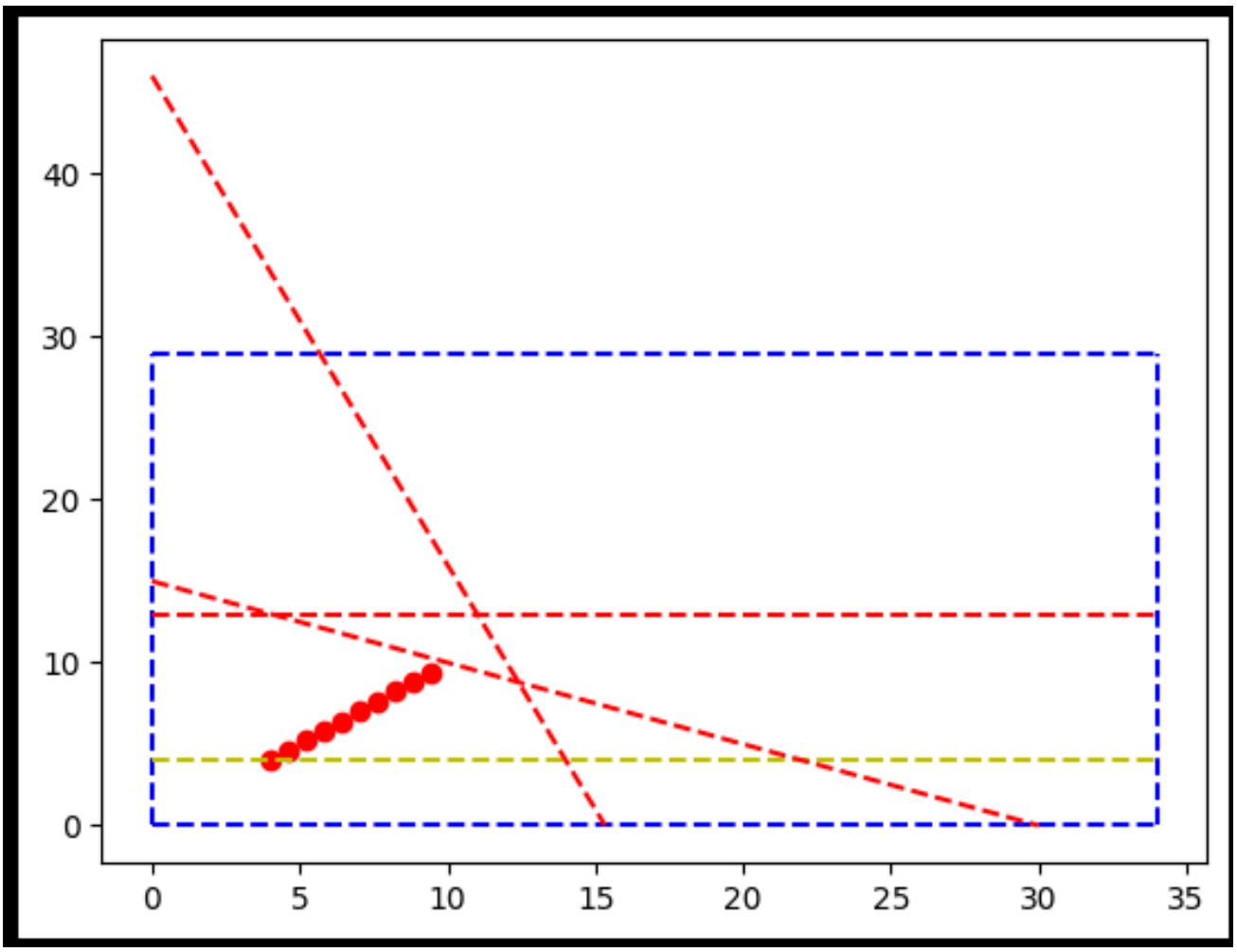

Рисунок 12 - Начальные данные

На рисунке 12 выделена, различными барьерам, область решений, внутри которой вдоль вектора градиента (вектора наискорейшего возрастания целевой функции) зафиксированы начальные значения.

Далее алгоритм начинает свою работу. Сначала будут формироваться новые значения по параметрам уже созданных.

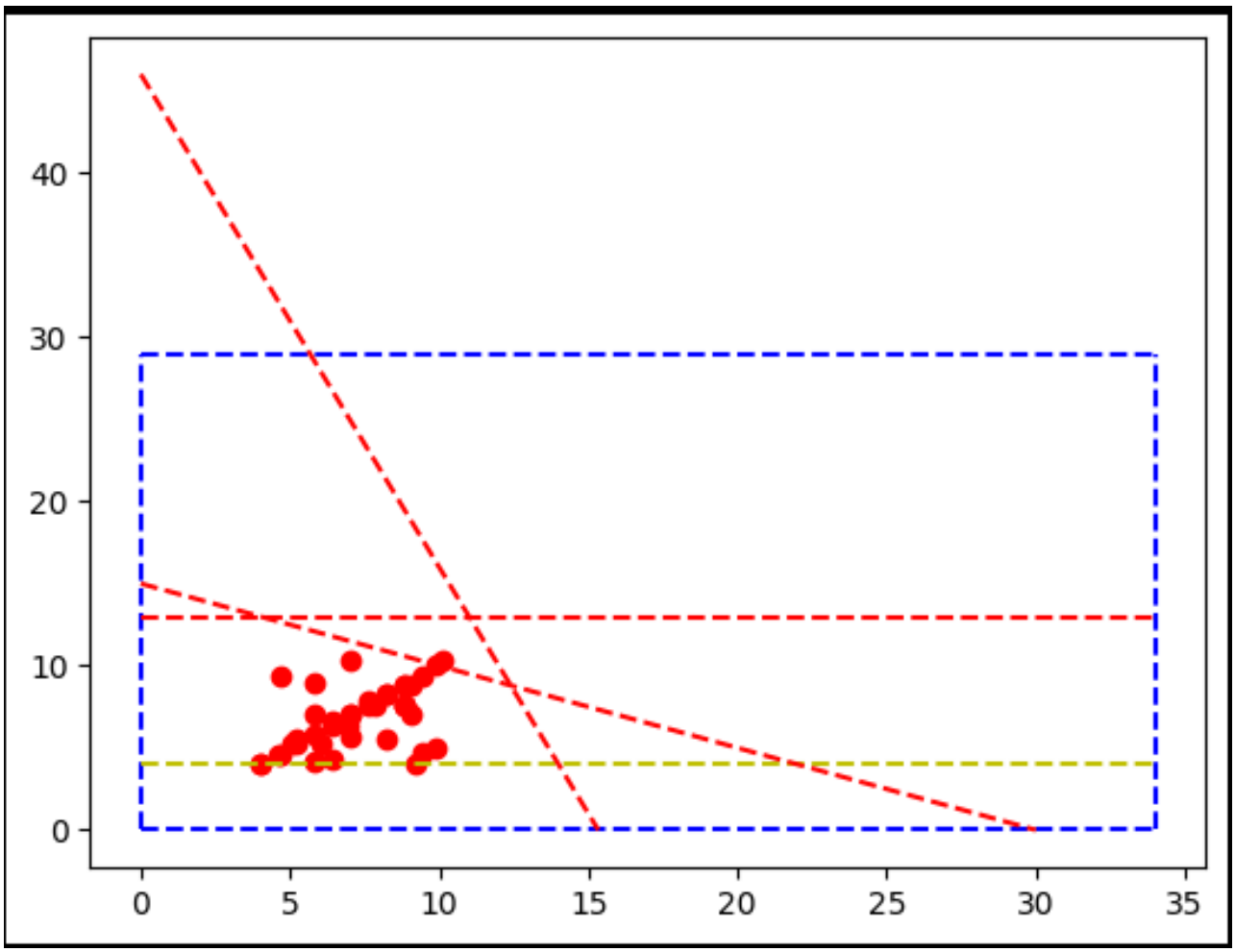

Рисунок 13 - 1й этап формирования решения

На рисунке 13 показаны возможные точки решения для 1го этапа формирования решения. До тех пор пока в списке не заполнены свободные места решение будет заполнять область прямоугольника, куба, … до тех пор пока не будет заполнен весь список.

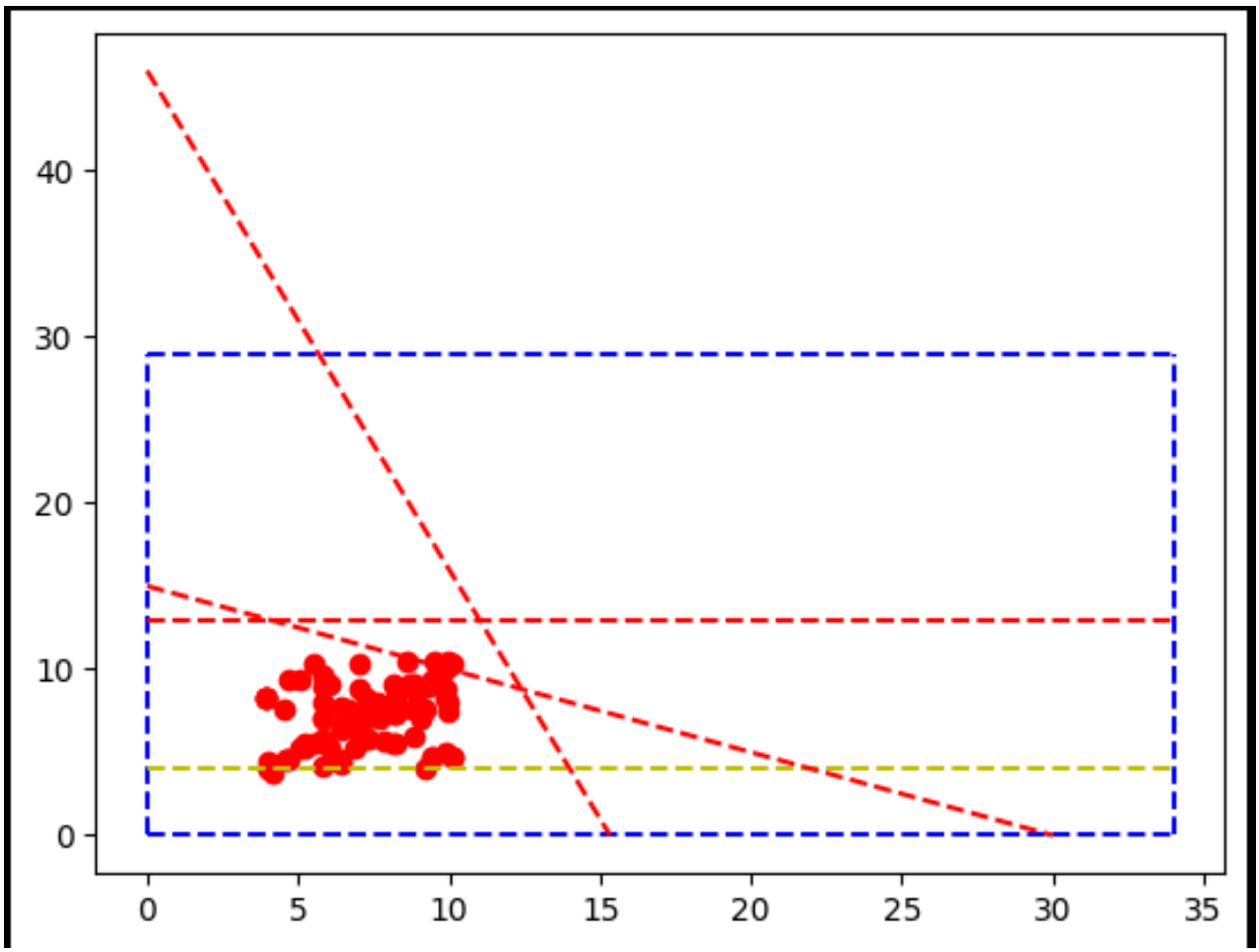

Рисунок 14 - заполнение пробелов списка

На рисунке 14 показаны возможные точки решения для 1го этапа формирования решения в тот момент когда начинается 2й этап. Затем область решения начинает стягиваться к лучшим значениям.

Скорость стягивания решения к максимальной точке определяется пределами мутаций каждой новой появляющейся точки, здесь разумно уменьшать пределы мутации с каждой новой итерацией предполагая что значения ближе к целевому решению и не должны выйти за пределы области решения.

Начала процесса перемещения решения к максимуму показано на следующем рисунке.

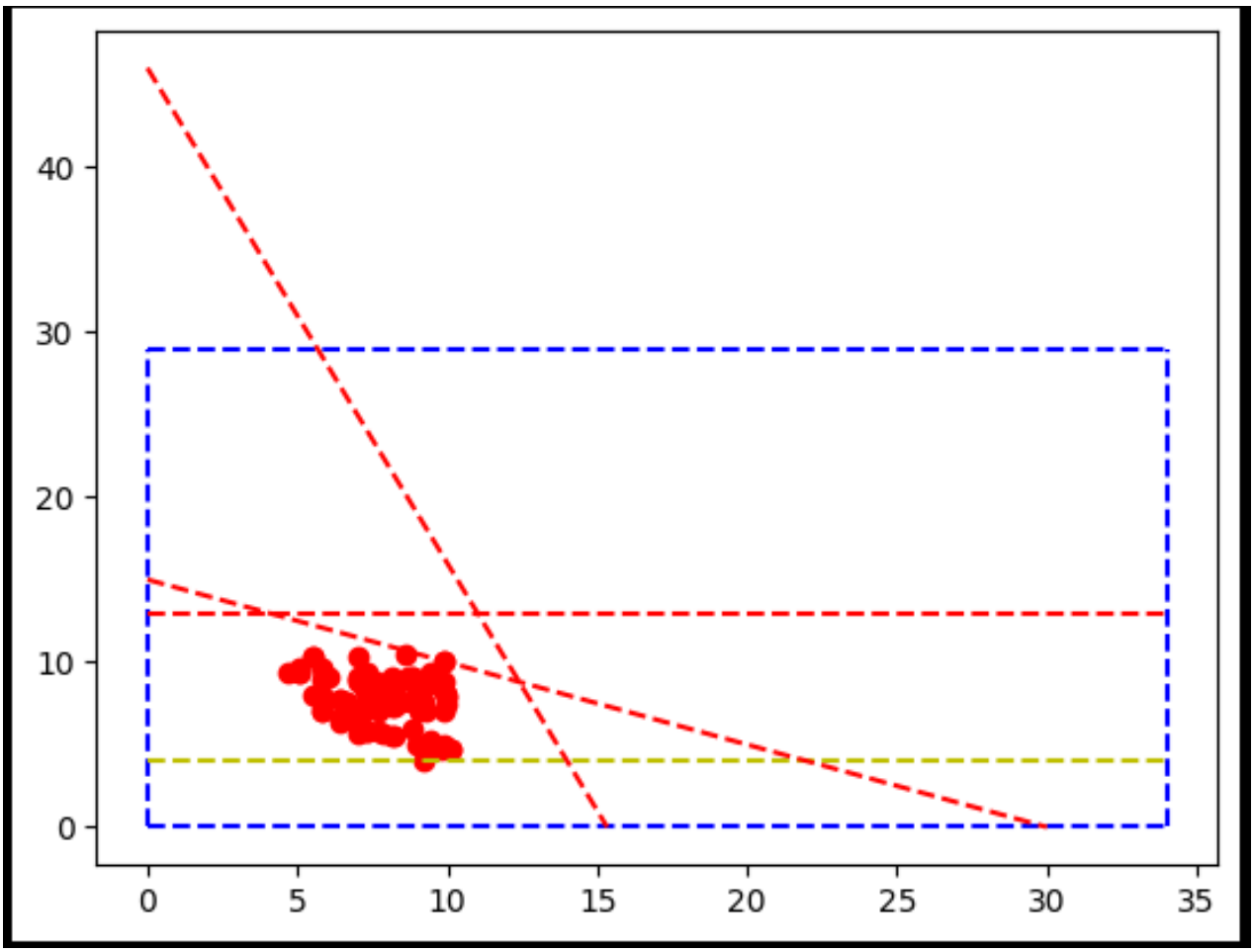

Рисунок 15 - 2й этап формирования решения

На рисунке 15 показаны возможные точки решения для 2го этапа формирования решения. На этом этапе начинает влиять случайная мутация и список решений начнет упираться в барьер.

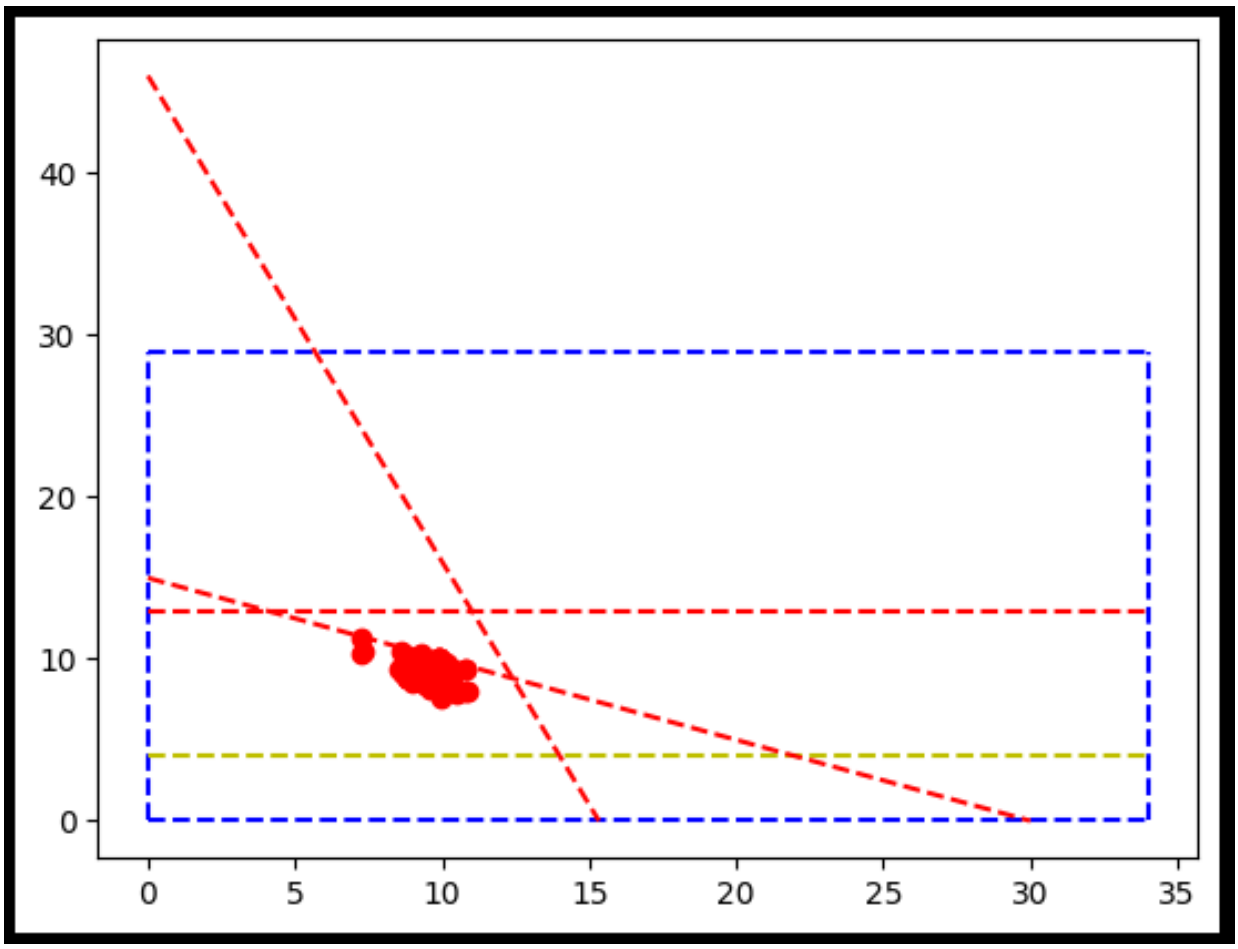

Рисунок 16 - 3й этап формирования решения

На рисунке 16 показаны возможные точки решения для 3го этапа формирования решения. На этом этапе вступает в силу факт того что барьеры часто не перпендикулярны градиенту. Что позволит заметить что решения в правой части области решений лучше чем в левой. Это позволит значениям скользить вдоль барьера в точку максимума функции.

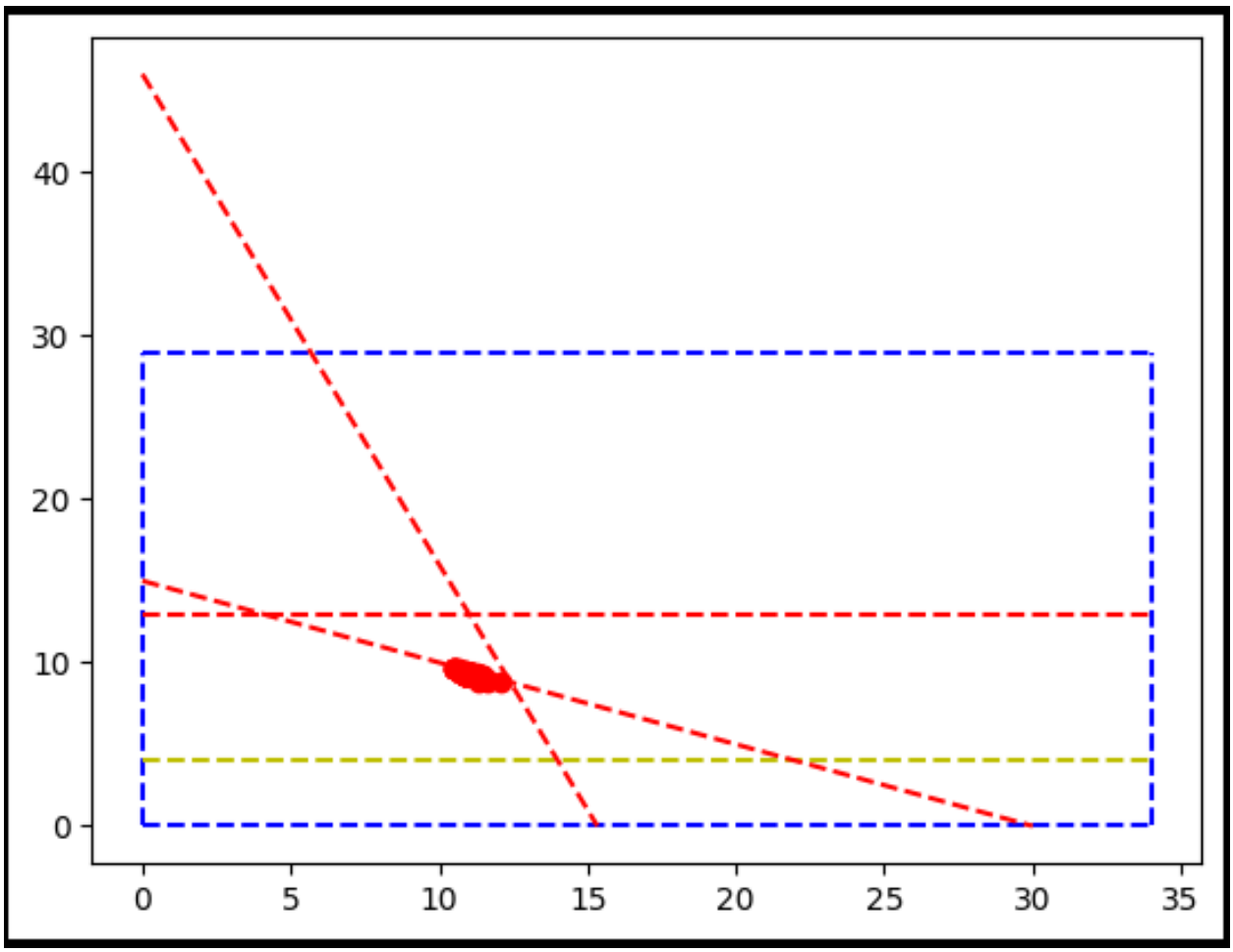

Рисунок 17 - 4й этап формирования решения

На рисунке 17 показаны возможные точки решения для 4го этапа формирования решения. На последнем этапе вступает в силу фактор того что решения начинают застревать в точке максимума функции, а то что с помощью линейных ограничений невозможно получить локальные максимумы в области решений говорит о том что, единственным максимумом будет решение.

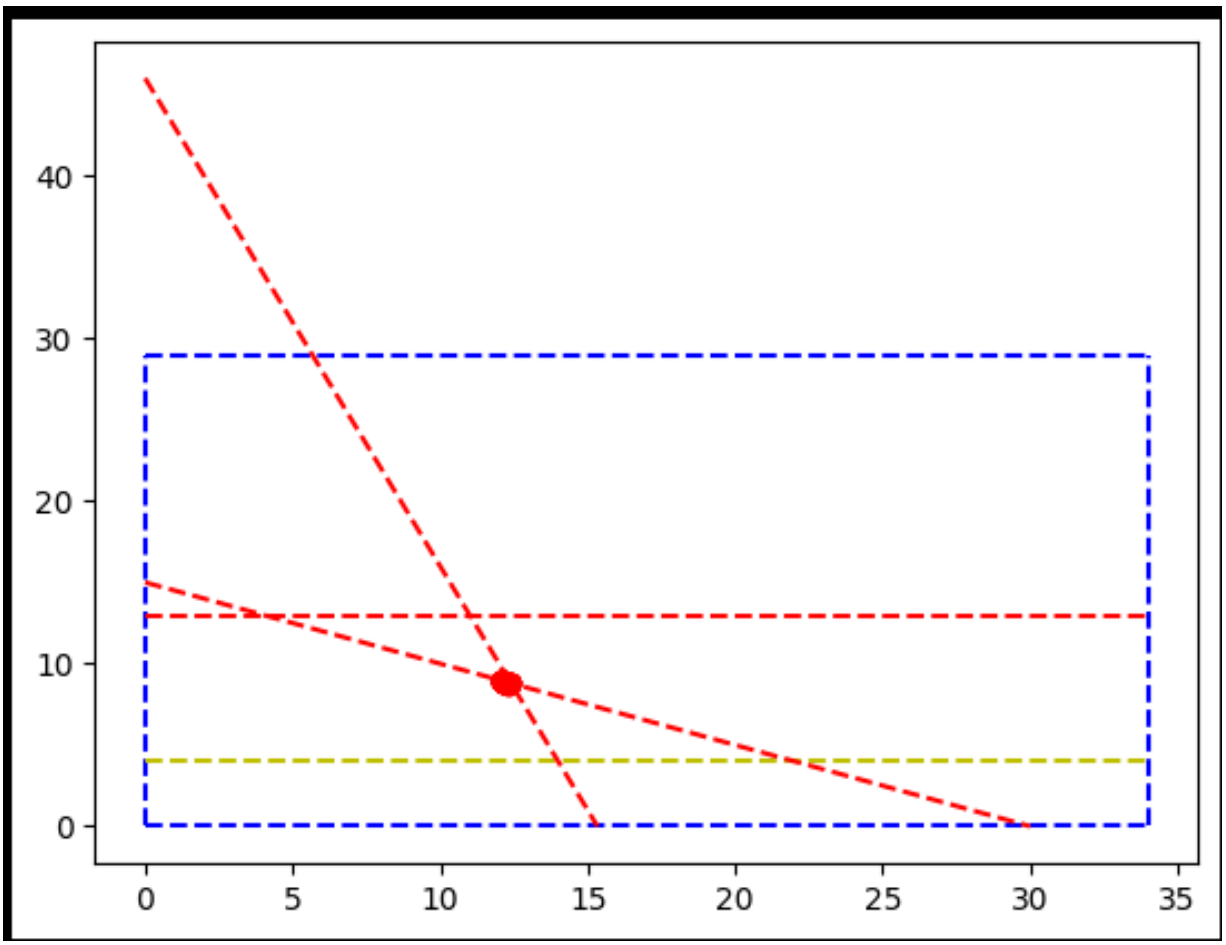

Рисунок 18 - заключительный этап формирования решения

На рисунке 18 показаны возможные точки решения для заключительного этапа формирования решения. На последнем этапе большинство точек собралось в области решения что говорит о том что среднее квадратическое отклонение среди лучших значений не превышает целевой и цикл поиска решений можно завершать.

В том случае если барьер перпендикулярен вектору градиента то любая точка этого барьера будет глобальным максимумом и подходит в качестве решения.

Таким образом можно гарантировано получить решения на любом числе параметров, но это может потребовать очень долгих вычислений. Решить эту проблему призвана интеграция следующего алгоритма в генетический алгоритм.

3.3 Реализация запоминающей нейронной сети

Возможность данного решения заключается в предположение что в приложении пользователь будет повторно решать такую-же или похожую задачу. На основе этого можно утверждать что переобученная нейронная сеть выдаст близкое значение для большинства параметров будет близким к решению.

Передача такого значения в генетический алгоритм заблокирует расхождение по этим параметрам, стягивая значения к решению.

Далее будет представлено применение нейронной сети для запоминания демонстрационной с генерированной задачи. Сеть будет соответствовать следующей схеме слоев нейронной сети.

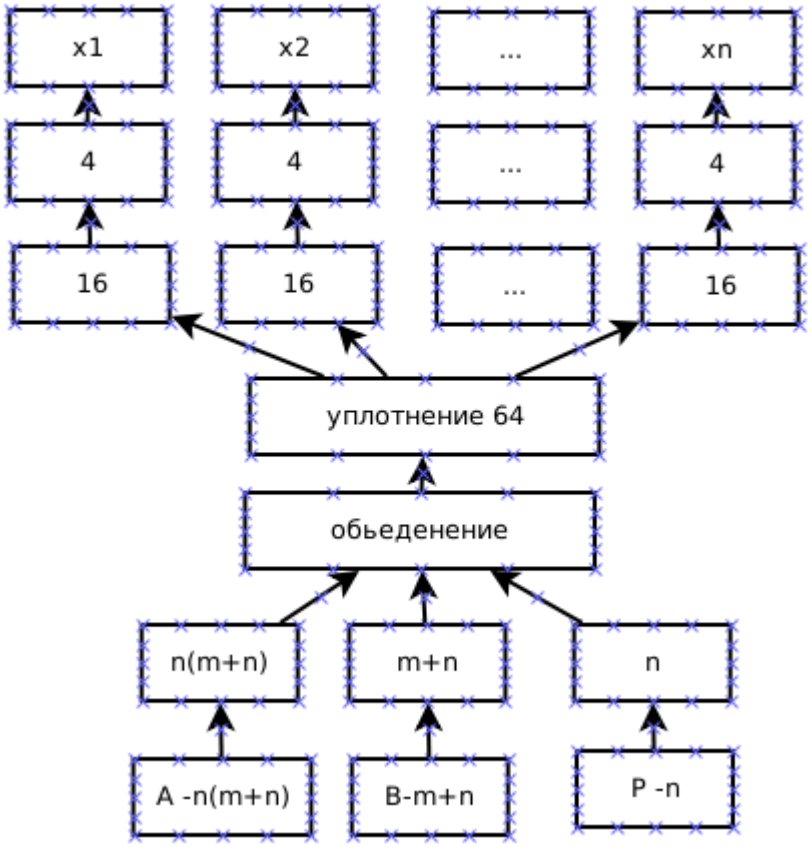

Рисунок 19 - Структура нейронной сети

На рисунке 19 показана структура слоев нейронной сети которая будет реализована. Для того чтобы реализовать структуру нейронной сети как на

рисунке 4,19 необходимо преобразовать входные и выходные данные тренировочной выборки, записав соответствующие значения нулями формулы (4*), (5*). Реализация метода будет следующей.

```
def to_vector(n_max,m_max,A,B,P):
    n = len(P)
    m = len(B)

    vec = [0 for i in range(0,n_max*m_max+m_max+n_max)
    i = n_max*m_max+m_max+n_max
    for j in range(n-1,-1,-1):
        i-=1
        vec[i]=P[j]
    i = n_max*m_max+m_max
    for j in range(m-1,-1,-1):
        i-=1
        vec[i]=B[j]
    i = n_max*m_max
    for j in range(m-1,-1,-1):
        for k in range(n-1,-1,-1):
            i-=1
            vec[i]=A[j][k]
    return array(vec)
```

```
def resize(lst, size):
  res = [0 for i in range(0,size)]
  i = size
  for j in range(len(lst)-1,-1,-1):
    i-=1
    res[i]=lst[j]
  return array(res)
```

Рисунок 20 - Изменение размера задачи

На рисунке 20 показаны функции для преобразования вычисляемой задачи под размер входа нейронной сети. Преобразовав задачу таким образом входные и выходные параметры сети будут подаваться как вектор. В дальнейшем они будут дополнительно преобразованы под структуру сети.

```
__in_A = Input(100, name="iA")
__in_B = Input(10, name="iB")
__in_P = Input(10, name="iP")
_s1_A = Dense(100, activation="relu")(__in_A)
_s1_B = Dense(10, activation="relu")(__in_B)
_s1_P = Dense(10, activation="relu")(__in_P)
_s2 = concatenate([_s1_A,_s1_B,_s1_P])
_s3 = Dense(64, activation="relu")(_s2)
_s4 = [Dense(4, activation="relu", name=f"s4_{i}")(_s3) for i in range(0,10)]
_s6 = [Dense(4, activation="relu", name=f"s6_{i}")(_s4[i]) for i in range(0,10)]
_out = [Dense(1, activation="linear", name=f"x{i}")(_s6[i]) for i in range(0,10)]
model_4 = Model([__in_A,__in_B,__in_P], _out)
model_4.summary()
```

Рисунок 21 - Реализация сетевой модели

На рисунке 21 показан код, необходимый для получения целевой модели нейронной сети. Структура сети показана на следующей картинке.

```
Model: "model_9"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 iA (InputLayer)             [(None, 100)]             0         []

 iB (InputLayer)             [(None, 10)]              0         []

 iP (InputLayer)             [(None, 10)]              0         []

 dense_38 (Dense)            (None, 100)               10100     ['iA[0][0]']

 dense_39 (Dense)            (None, 10)                110       ['iB[0][0]']

 dense_40 (Dense)            (None, 10)                110       ['iP[0][0]']

 concatenate_9 (Concatenate  (None, 120)               0         ['dense_38[0][0]',
 )                                                                  'dense_39[0][0]',
                                                                    'dense_40[0][0]']

 dense_41 (Dense)            (None, 64)                7744      ['concatenate_9[0][0]']

 s4_0 (Dense)                (None, 4)                 260       ['dense_41[0][0]']

 s4_1 (Dense)                (None, 4)                 260       ['dense_41[0][0]']

 s4_2 (Dense)                (None, 4)                 260       ['dense_41[0][0]']

 s4_3 (Dense)                (None, 4)                 260       ['dense_41[0][0]']

 s4_4 (Dense)                (None, 4)                 260       ['dense_41[0][0]']

 s4_5 (Dense)                (None, 4)                 260       ['dense_41[0][0]']

 s4_6 (Dense)                (None, 4)                 260       ['dense_41[0][0]']

 s4_7 (Dense)                (None, 4)                 260       ['dense_41[0][0]']

 s4_8 (Dense)                (None, 4)                 260       ['dense_41[0][0]']

 s4_9 (Dense)                (None, 4)                 260       ['dense_41[0][0]']

 s6_0 (Dense)                (None, 4)                 20        ['s4_0[0][0]']

 s6_1 (Dense)                (None, 4)                 20        ['s4_1[0][0]']

 s6_2 (Dense)                (None, 4)                 20        ['s4_2[0][0]']

 s6_3 (Dense)                (None, 4)                 20        ['s4_3[0][0]']

 s6_4 (Dense)                (None, 4)                 20        ['s4_4[0][0]']

 s6_5 (Dense)                (None, 4)                 20        ['s4_5[0][0]']

 s6_6 (Dense)                (None, 4)                 20        ['s4_6[0][0]']

 s6_7 (Dense)                (None, 4)                 20        ['s4_7[0][0]']

 s6_8 (Dense)                (None, 4)                 20        ['s4_8[0][0]']

 s6_9 (Dense)                (None, 4)                 20        ['s4_9[0][0]']

 x0 (Dense)                  (None, 1)                 5         ['s6_0[0][0]']

 x1 (Dense)                  (None, 1)                 5         ['s6_1[0][0]']

 x2 (Dense)                  (None, 1)                 5         ['s6_2[0][0]']

 x3 (Dense)                  (None, 1)                 5         ['s6_3[0][0]']

 x4 (Dense)                  (None, 1)                 5         ['s6_4[0][0]']

 x5 (Dense)                  (None, 1)                 5         ['s6_5[0][0]']

 x6 (Dense)                  (None, 1)                 5         ['s6_6[0][0]']

 x7 (Dense)                  (None, 1)                 5         ['s6_7[0][0]']

 x8 (Dense)                  (None, 1)                 5         ['s6_8[0][0]']

 x9 (Dense)                  (None, 1)                 5         ['s6_9[0][0]']
```

Рисунок 22 - Слои в нейронной сети

На рисунке 22 показана полученная структура слоев нейронной сети. Скомпилировав эту нейронную сеть с квадратической функцией потерь можно получить нейронную сеть для решения задачи регрессии на каждый из выходных параметров. Затем сеть можно обучить.

```
_A, _B, _C, _X = [],[],[], [[] for i in range(0,10)]
for e in train:
  b = resize(e['res'],10)
  for i in range(0,len(b)):
    _X[i].append(b[i])
  b = to_vector(n_max=10,m_max=10,A=e['A'],B=e['B'],P=e['P'])
  _A.append(b[0:100])
  _B.append(b[100:110])
  _C.append(b[110:120])
_X = [array(x) for x in _X]
_A = array(_A)
_B = array(_B)
_C = array(_C)

model_4.compile(optimizer='rmsprop', loss = ['mean_squared_error' for i in range(0,10)], metrics=['mean_squared_error'])
model_4.fit(
    x=[_A, _B, _C],
    y=_X,
    epochs=90
)
```

Рисунок 23 - Компиляция и обучение нейронной сети

На рисунке 23 показан фрагмент кода предназначенный для обучения сети на тренированных данных.

В дальнейшем применяя эту сеть на значениях по структуре подобных одним из тех которые сеть запомнила будет происходить получение результата.

```
[model_4.predict([_A, _B, _C])[i][6] for i in range(0,10)]
```

```
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
[array([0.00903481], dtype=float32),
 array([0.0132869], dtype=float32),
 array([0.02203238], dtype=float32),
 array([0.05106506], dtype=float32),
 array([0.07330011], dtype=float32),
array([0.0540435], dtype=float32),
array([0.11686443], dtype=float32),
array([0.28646225], dtype=float32),
array([0.17978448], dtype=float32),
array([0.1665784], dtype=float32)]
```

```
[_X[i][6] for i in range(0,10)]
```

```
[0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.052020000000000004,
 0.11716,
 0.3474,
 0.19310000000000002,
 0.17978000000000002]
```

Рисунок 24 - Восстановление запомненного сетью результата

На рисунке 24 показан пример того как для выбранного решения задачи были восстановлены значения на основе входных значений этой задачи. Таким образом все необходимые алгоритмы реализованы и можно приступать к реализации интерфейса и приложения.

3.4 Создание интерфейса приложения

Теперь необходима разработать само приложение и соединить его с алгоритмами. Структура kivy приложения будет построена в соответствии с следующей диаграммой классов.

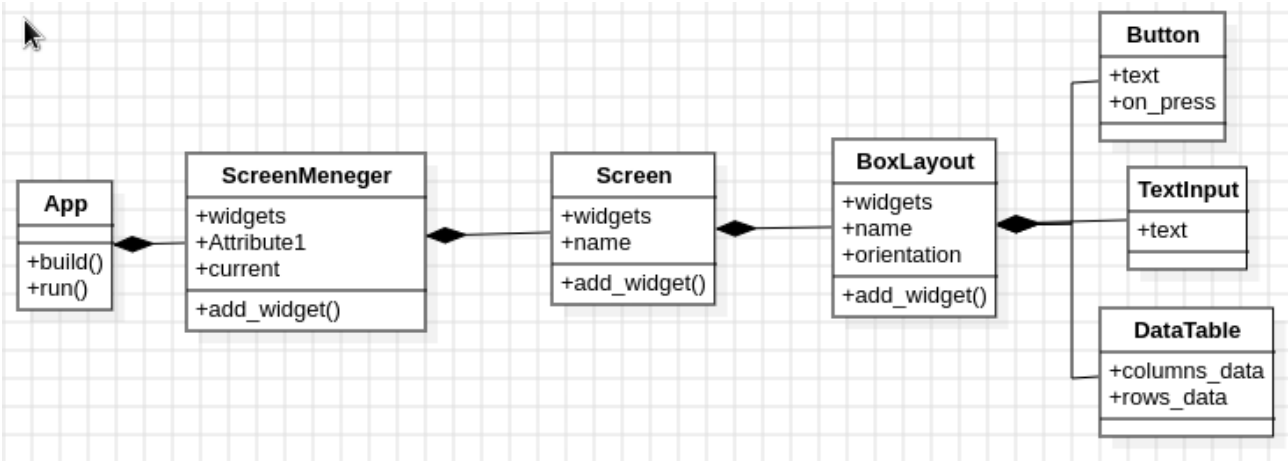

Рисунок 25 - Структура классов в kivy приложении

На рисунке 25 показана возможная схема структуры компонентов приложения созданного на kivy. На этой схеме показано что на самом верхнем уровне будет объект приложение, которое может быть создано или запущено. Объект — это абстрактная сущность, наделенная характеристиками объектов реального мира, описанная классом [20]. Все данные в программе на Python представлены объектами или отношениями между объектами [18]. Класс - это определяемый пользователем тип или структура данных, который содержит данные и функции, которыми будет обладать созданный на его основе объект [4]. Приложение в своем корне будет содержать менеджер экранов, который позволит переключаться между содержащимися в нем экранными формами. Экранная форма состоит как минимум из одного слоя, который может содержать различные интерфейсные элементы такие как кнопки, поля ввода и таблицы.

Для того чтобы создать kivy приложение необходимо создать и запустить объект приложения.

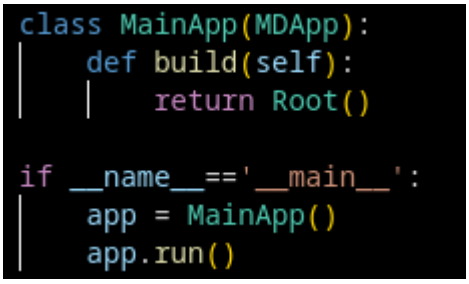

Рисунок 26 - Объект приложения

На рисунке 26 показан фрагмент кода на котором задается верхний уровень приложения и выполняется запуск приложения.

При создании объект приложения будет построен используя корневой элемент интерфейса, наследуемый от менеджера экранов или слоя. Далее идет инициализация корневого элемента и создания экранной формы отображающей список продуктов.

```
class Root(ScreenManager):

    def go(self, name): self.current=name

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
# данные
        data = load(open(file="data.json", mode="r"))
        # продукты
        продукты = Screen(name="продукты")
        _ = BoxLayout(orientation="vertical")
        __ = BoxLayout(size_hint_y=0.1)
        __.add_widget(Button(text='данные', on_press=lambda b: self.go("продукты"), background_color= (0.4, 1.0, 0.0)))
        __.add_widget(Button(text='расчеты', on_press=lambda b: self.go("выбор"), background_color= (0.0, 0.8, 0.0)))
        _.add_widget(__)
        __ = BoxLayout(size_hint_y=0.1)
        __.add_widget(Button(text='продукты', on_press=lambda b: self.go("продукты"), background_color= (0.4, 1.0, 0.0)))
        __.add_widget(Button(text='ресурсы', on_press=lambda b: self.go("ресурсы"), background_color= (0.0, 0.8, 0.0)))
        __.add_widget(Button(text='затраты', on_press=lambda b: self.go("затраты"), background_color= (0.0, 0.8, 0.0)))
        __.add_widget(Button(text='продажи', on_press=lambda b: self.go("продажи"), background_color= (0.0, 0.8, 0.0)))
        _.add_widget(__)
        __ = BoxLayout(size_hint_y=0.8)
        self.список_продуктов = MDDataTable(
            use_pagination=True,
            check=True,
            column_data=[
                ("id",20),
                ("название",70)
            ],
            row_data=data["продукты"]
        )
        __.add_widget(self.список_продуктов)
        _.add_widget(__)
        __=BoxLayout(size_hint_y=0.1)
        def del_продукты():
            for e in self.список_продуктов.get_row_checks():
                self.список_продуктов.remove_row([int(e[0]),e[1]])
                self.выбор_продуктов.remove_row([int(e[0]),e[1]])
        __.add_widget(Button(text="удалить", background_color= (0.3, 0.3, 1.0),  on_press=lambda b:del_продукты()))
        __.add_widget(Button(text="добавить", background_color= (0.2, 0.2, 1.0),  on_press=lambda b: self.go("новый_продукт")))
        _.add_widget(__)
        продукты.add_widget(_)
        self.add_widget(продукты)
```

Рисунок 27 - инициализация корневого элемента

На рисунке 27 показан фрагмент кода описывающий одну из экранных форм приложения.

Помимо инициализации на рисунке 27 показан метод смены текущей экранной формы.

Подгрузив данные создается экранная форма для отображения списка продуктов. В верхней части слоя этой формы создаются 6 кнопок для навигации по меню, как и на каждой следующей экранной форме.

После создания меню создается таблица считанных данных продуктов, объект записывается в корень для обновления данных. После таблицы добавляются кнопки: кнопка удалить, для удаление объектов с установленным флажком и кнопка добавить которая переводит на другую экранную форму где будет добавляться продукт.

Таким образом разрабатываются и все остальные экранные формы. После того как приложение сделано оно упаковывается в apk файл и его отображение можно проверить на телефоне Android.

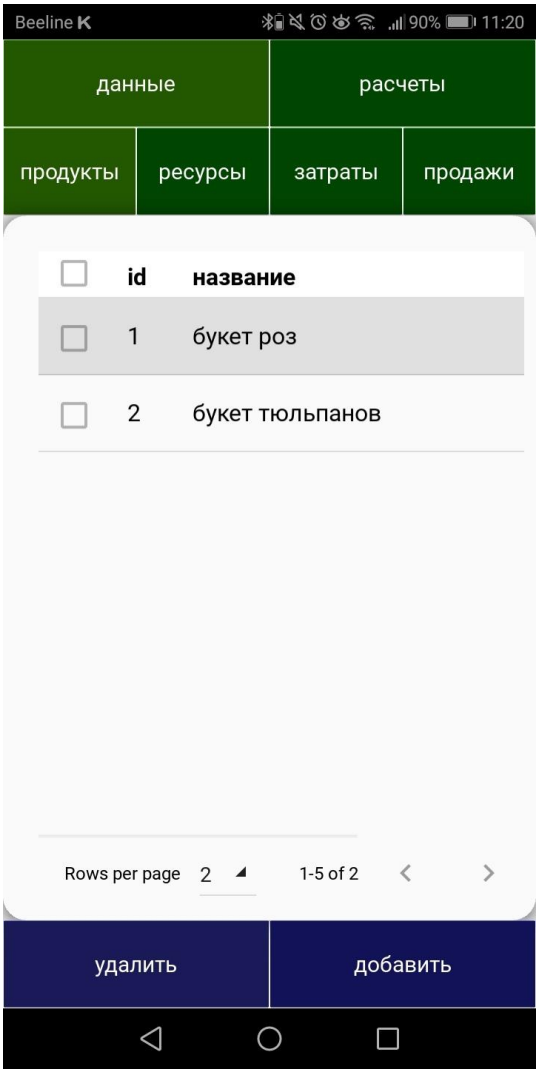

Рисунок 28 - Демонстрация отображаемой формы продуктов

На рисунке 28 продемонстрировано отображение одной из экранных форм в работающем приложении.

Так как kivy кросс платформенный упаковка для проверки работоспособности необязательна и работа программы может быть протестирована на любой платформе.

Для переноса созданной программы с интерфейсом сделанном на фреймворке kivy необходимо использовать сборщик buildozer. Этот

инструмент позволит без дополнительных усилий собрать программу под android.

Buildozer управляет файлом с именем buildozer.spec в каталоге вашего приложения с описанием требований и настроек вашего приложения, таких как заголовок, значок, включенные модули и т.д. Файл спецификации будет использоваться для создания пакета для Android, iOS и других устройств [17].

Первая компиляция проекта в builodzer занимает много времени, но все необходимые инструменты будут установлены в папку проекта, для сборки android приложений с помощью этого инструмента лучше всего завести отдельный проект для сборки и перемешать исходный код приложений в этот проект при каждой сборке.

Кроме android этот инструмент позволяет собирать программу для установки и на других платформах таких как iOS, Windows, OSX и т. д.

3.5 Работа в разработанной программе

После того как приложение сделано и собрано в нем можно провести вычисления для каких-либо данных. Идеальным примером могут быть данные цветочного магазина, потому что цветы на месте собираются в букет из материалов в магазине и сразу продаются.

Работа программы будет проверяться на примере данных продаж в цветочном магазине в январе.

```
{
    "продукты":[[1,"букет роз"], [2,"букет тюльпанов"]],
    "ресурсы":[[1,"ящик роз",2],[2,"ящик тюльпанов",1],[3,"рулон пленки",2],[4,"лента",3]],
    "затраты":[[1,1,0.125 ],[1,2,0     ],[2,2,0.125 ],[2,1,0     ],[1,3,0.0125],[1,4,0.0125],[2,3,0.0125],[2,4,0.0125]],
    "продажи":[
        [1,"12/01/2019",120],
        [2,"13/01/2019",100],
        [1,"12/01/2019",120],
        [2,"13/01/2019",100],
        [1,"15/02/2019",120],
        [1,"20/03/2019",120],
        [2,"13/04/2019",100],
        [1,"08/06/2019",120],
        [1,"13/09/2019",120],
        [2,"13/10/2019",100],
        [1,"12/01/2020",120],
        [1,"13/01/2020",120],
        [1,"12/01/2021",120],
        [1,"12/01/2022",120],
        [1,"12/01/2023",120],
        [1,"12/01/2023",120],
        [1,"12/01/2023",120],
        [1,"12/01/2023",120]
    ]
```


Рисунок 29 - Демонстрационные данные

На рисунке 29 показаны данные которые будут задействованы в расчете демонстрационной задачи.

В этих данных есть 2 категории товаров, эти товары собираются из ресурсов четырех видом и для этих товаров есть история продаж. Анализируя эту историю видно что на тюльпаны в январе обычно нет спроса, в то время как розы покупают. В тоже время в текущий период на складе есть запасы, которые позволят в этот момент произвести товары обоих видов. Запустив вычисления в программе будет получена следующая картина.

Рисунок 30 - Результаты вычислений

На рисунке 30 показано результаты решения задачи. По этим результатам видно что спроса на тюльпаны в январе не ожидается а запасы на складе позволяют произвести большое количество продукции.

Таким образом приложение работоспособно и его разработку можно считать завершенной.

Заключение

В процессе выполнения курсовой балкарской работы было выполнено множество задач:

-был спроектирован интерфейс программного продукта;

-были сформулированы методы для решения задач предсказания элементов последовательности и решения задачи условной оптимизации;

-был зафиксирован подход для воспроизведения решенной задачи с использованием нейронных сетей;

-сформулированные методы были реализованы программно и интегрированы в разработанный программный продукт, соответствующий спроектированному интерфейсу;

-была проверена работоспособность программного продукта на демонстрационной задаче.

Как результат проделанной работы был получено работоспособное android приложение, позволяющее пользователю решить целевую задачу и предоставляющее для этого все необходимые интерфейсные элементы.

Таким образом поставленная задача может быть выполнена в полном объеме — получено работающее мобильное приложения с простым, интуитивно понятным, интерфейсом, дающее простые для понимания и анализа результаты.

Использование фрэймворка kivy связано с распространенными проблемами при сборке приложения под платформу android: быстрое увеличение размера приложения по занимаемой им памяти и расхождение в порядке обработки экранных форм и таблиц данных при адаптации их к python-for-android.

Несмотря на эти недостатки, в приложении можно работать и его распространение возможно на таких платформах как github и playmarket.

Список используемой литературы

1. Атманов С.А. Линейное программирование. М.: “Наука”, 1981. — 340 с.
2. Бахвалов Н.С. Численные методы. – М.: “Наука”, 1993 — 636 с.
3. Березин И.С., Жидков Н.П. Методы вычислений. Том 1 и 2. – М.:“Наука”, 1994 — 620 с.
4. Васильев А.Н., ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА C++ — СПб.: Наука и Техника, 2016. — 544 с.
5. Зенков А.В. Численные методы: учеб. пособие — Екатеринбург : Издво Урал. ун-та, 2016. — 124 с.
6. Зуховицкий С.И., Авдеева Л.И. Линейное и выпуклое программирование. – М.: “Наука”, 1994.
7. Зюзев А.М. Объектно ориентированное программирование: учеб.-метод.пособие — Екатеринбур : Изд-во Урал. ун-та, 2019.— 116 с.
8. Измаилов А.Ф., Солодов М.В. Численные методы оптимизации: Учеб.пособие. – М.: ФИЗМАТЛИТ, 2005. – 304 с.
9. Кольцов Д.М., Дубовик Е.В. СПРлвочник PYTHON. Кратко, быстро, под рукой - СПб.: Наука и Техника, 2021. - 288 с.
10. Лесин В.В., Лисовец Ю.П. Основы методов оптимизации. – М.: Изд-во МАИ, 1995. – 344 с.
11. Моисеев Н. Н., Иванилов Ю. П., Столярова Е. М., Методы оптимизации. М.: – Наука, 1978.
12. Рейзлин, В. И. ЧИСЛЕННЫЕ МЕТОДЫ ОПТИМИЗАЦИИ / В. И. Рейзлин — 1-е изд. — Томск: Издательство Национального исследовательского Томского политехнического университета, 2013 — 105 с.
13. Растригин Л.А. Случайный поиск в задачах оптимизации многопараметрических систем. Рига, Зинатне,1965. 212 с

14. Растригин Л.А. Случайный поиск. М.: Знание, 1979. 64 с.
15. Шолле Ф. Глубокое обучение на Python. — СПб.: Питер, 2018. — 400 с
16. Barry P. Head First Python / Barry P. — 2nd edition. — : O'Reilly Media, 2016 — 622 с.
17. Buildozer documentation [Электронный ресурс]]. URL: https://buildozer.readthedocs.io/en/latest/index.html
18. Downey A. Think Python, 2e: How to Think Like a Computer Scientist / Downey A. — 2nd edition. — : O'Reilly Media, 2016 — 289 с.
19. Kivy documentation [Электронный ресурс]. URL: https://kivy.org/doc/stable/
20. Lutz M. Learning Python: Powerful Object-Oriented Programming / Lutz M. — 5th edition. — : O'Reilly Media, 2013 — 1643 с.