**Реферат**

Курсовой проект: 50 страниц, 22 рисунка, 3 таблицы, 6 используемых источников, 4 приложения.

СИМПЛЕКС-МЕТОД, VISUAL STUDIO, C#, WINDOWS FORMS, МЕТОДЫ ОПТИМИЗАЦИИ, ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

Целью данного курсового проекта является разработка приложения для расчета прибыли производства предприятия, процесс деятельности которого можно квантизировать и перевести в математическую модель, доступную для решения симплекс-методом.

В результате выполнения курсовой работы разработано десктоп-приложение, решающее поставленную проблему.

Реализованное приложение позволяет повысить эффективность работы предприятия, на котором оно будет использоваться посредством оптимизации имеющихся ресурсов, например, с целью получения максимальной прибыли.

Содержание

[Введение 6](#_Toc165816253)

[Нормативные ссылки 8](#_Toc165816254)

[1Анализ технических требований и уточнение спецификаций 9](#_Toc165816255)

[1.1 Анализ задания и выбор технологии, языка и среды разработки 9](#_Toc165816256)

[1.2 Анализ процесса обработки информации и построение функциональных диаграмм 10](#_Toc165816257)

[1.3 Анализ хранимой информации и выбор структур данных для ее представления 13](#_Toc165816258)

[1.4 Выбор методов и алгоритмов решения задачи 14](#_Toc165816259)

[2 Проектирование структуры и компонентов программного продукта 17](#_Toc165816260)

[2.1 Разработка интерфейса пользователя 17](#_Toc165816261)

[2.1.1 Разработка структурной схемы интерфейса 17](#_Toc165816262)

[2.1.2 Построение графа (диаграммы) состояний интерфейса 18](#_Toc165816263)

[2.1.3 Разработка форм ввода-вывода информации 19](#_Toc165816264)

[2.2 Разработка алгоритма основной программы и структурной схемы программного продукта 20](#_Toc165816265)

[2.2.1 Описание структуры приложения и схема связности модулей 20](#_Toc165816266)

[2.2.2 Схема движения информационных потоков 21](#_Toc165816267)

[2.3 Разработка основных алгоритмов программного продукта 22](#_Toc165816268)

[2.4 Разработанные классы программы 26](#_Toc165816269)

[3 Тестирование программы 32](#_Toc165816270)

[3.1 Разработка плана тестирования 32](#_Toc165816271)

[3.2 Разработка алгоритма процедуры тестирования 33](#_Toc165816272)

[3.3 Оценка результатов тестирования 33](#_Toc165816273)

[4 Сопровождение 35](#_Toc165816274)

[4.1 Руководство пользователя 35](#_Toc165816275)

[4.2 Обслуживание модели, алгоритма, программы и их эксплуатация 37](#_Toc165816276)

[Заключение 38](#_Toc165816277)

[Список используемой литературы 40](#_Toc165816278)

[Приложение А – Планы тестирования интерфейса 41](#_Toc165816279)

[Приложение Б – Листинг приложения 44](#_Toc165816280)

[Приложение В – Листинг модульных тестов 73](#_Toc165816281)

[Приложение Г - Антиплагиат 77](#_Toc165816282)

# Введение

Современное информационное общество неразрывно связано с обработкой данных и оптимизацией процессов. В этом контексте методы линейного программирования, в частности симплекс-метод, играют ключевую роль в решении различных задач оптимизации, от экономических до технических.

Предметная область, к которой относится данная курсовая работа, охватывает проблематику оптимизации процессов производства, распределения ресурсов или планирования бюджета. Симплекс-метод является одним из наиболее эффективных инструментов для решения таких задач, благодаря своей способности находить оптимальное решение в многообразии ограничений и переменных.

Цель настоящей курсовой работы заключается в исследовании и применении симплекс-метода для решения конкретной задачи оптимизации в предметной области.

Задачи, поставленные перед курсовой работой, включают в себя:

* Изучение теоретических основ линейного программирования и симплекс-метода.
* Формализация конкретной задачи оптимизации в рамках выбранной предметной области.
* Программная реализация симплекс-метода для решения данной задачи с использованием предполагаемых средств проектирования.
* Анализ полученных результатов и их применение в реальной предметной области.

Актуальность данной работы обусловлена постоянным развитием технологий и растущей потребностью в оптимизации процессов в различных сферах деятельности. Эффективное использование симплекс-метода позволяет существенно сократить временные и материальные затраты, повысить производительность и эффективность бизнес-процессов.

Целью данной курсовой работы является исследование и применение симплекс-метода для решения конкретной задачи оптимизации в выбранной предметной области. В результате работы будет получено практическое решение, демонстрирующее применимость и эффективность симплекс-метода в решении реальных задач оптимизации.

Для разработки приложения, реализующего симплекс-метод для решения задачи оптимизации в выбранной предметной области, предполагается использовать следующие средства проектирования:

* Язык программирования: В качестве основного языка программирования будет использован C#. Он предоставляет широкие возможности для создания приложений под платформу .NET и имеет богатые инструменты для работы с математическими вычислениями.
* Интегрированная среда разработки (IDE): Для комфортной работы над проектом предполагается использовать Visual Studio 2022, которая обладает всеми необходимыми инструментами для разработки приложений на C#, включая удобный редактор кода, отладчик и инструменты для создания пользовательского интерфейса.
* CASE-средство: Для моделирования структуры приложения, визуализации классов и взаимодействия между компонентами будет использовано средство моделирования Star UML. Оно позволяет создавать диаграммы классов, диаграммы последовательностей и другие типы диаграмм для проектирования программного обеспечения.

Выбор этих средств проектирования обеспечит эффективную разработку приложения, сочетающего в себе удобство использования, высокую производительность и возможности визуализации результатов.

# Нормативные ссылки

В данном курсовом проекте использованы следующие нормативные ссылки:

* ГОСТ Р 1.5-2004 Стандартизация в Российской Федерации. Стандарты национальные Российской Федерации. Правила построения, изложения, оформления и обозначения.
* ГОСТ Р 1.12-2004 Стандартизация в Российской Федерации. Термины и определения.
* ГОСТ Р 7.0.5-2008 СИБИД. Библиографическая ссылка. Общие требования и правила составления.
* ГОСТ Р ИСО 9000-2008 Системы менеджмента качества. Основные положения и словарь.
* ГОСТ Р ИСО/МЭК 12207-99 Информационная технология. Процессы жизненного цикла программных средств.
* ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных продуктов.
* ГОСТ 19.105-78 ЕСПД. Обозначение программ и программных продуктов.
* ГОСТ 19.401-78 ЕСПД. Текст программы. Требования к содержанию и оформлению.
* ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов.

# 1Анализ технических требований и уточнение спецификаций

## 1.1 Анализ задания и выбор технологии, языка и среды разработки

Выбор использования объектно-ориентированной парадигмы программирования в реализации приложения для данной курсовой работы обосновывается следующими причинами:

* Модульность и структурирование: Объектно-ориентированное программирование позволяет структурировать код приложения на основе объектов, которые соответствуют реальным или абстрактным сущностям предметной области. Это обеспечивает лучшую модульность, упрощает понимание и поддержку кода, а также способствует повторному использованию компонентов.
* Инкапсуляция и сокрытие данных: ООП позволяет скрыть внутреннюю реализацию объектов и предоставить только необходимый интерфейс для работы с ними. Это повышает уровень абстракции, улучшает безопасность и облегчает сопровождение кода[6].
* Наследование и полиморфизм: Механизмы наследования и полиморфизма позволяют создавать иерархии классов и использовать общие свойства и методы, что способствует повторному использованию кода и уменьшает его объем.

Теперь, учитывая преимущества объектно-ориентированного программирования, были выбраны следующие средства программирования:

* C# - это мощный и современный язык программирования, который полностью поддерживает объектно-ориентированную парадигму. Он обладает богатыми возможностями для создания объектов, классов, наследования и полиморфизма, что делает его прекрасным выбором для разработки приложений, основанных на ООП.
* Windows Forms - это фреймворк для создания графических пользовательских интерфейсов в приложениях под платформу Windows. Он предоставляет широкий набор элементов управления и инструментов для создания интерактивных и интуитивно понятных интерфейсов. Windows Forms хорошо интегрирован с языком C# и предоставляет возможности для быстрой и удобной разработки приложений.
* Visual Studio 2022 - это мощная интегрированная среда разработки, которая обладает широкими возможностями для работы с языком C#, Windows Forms и другими технологиями Microsoft. Она предоставляет удобный редактор кода, инструменты для отладки, дизайнера форм, а также интеграцию с другими инструментами разработки и системами контроля версий.

## 1.2 Анализ процесса обработки информации и построение функциональных диаграмм

Вводимую и выводимую информацию можно представить в виде диаграммы вариантов использования, отражающей действия, которые пользователь будет предпринимать во время работы с приложением (рисунок 1).

#### 

#### Рисунок 1 – Диаграмма вариантов использования приложения

Таким образом были выявлены подлежащие реализации функции приложения, обрабатывающие получаемую информацию и выводящие результирующие значения.

Процесс обработки информации для разрабатываемого приложения включает следующие основные шаги:

1. Считывание введённых пользователем данных:

* Пользователь вводит данные, необходимые для постановки задачи оптимизации, такие как коэффициенты целевой функции, ограничения и параметры задачи.
* Приложение считывает эти данные и формирует математическую модель задачи оптимизации.

1. Пошаговое решение задачи:

* Применяется симплекс-метод для решения поставленной задачи оптимизации.
* На каждом шаге алгоритма происходит выбор базисных переменных, проверка на оптимальность текущего решения, вычисление и выбор направления движения и пересчёт базисных переменных.
* Каждый шаг алгоритма и соответствующие изменения в состоянии задачи фиксируются для последующего анализа.

1. Запись каждого шага в ходе решения:

* Приложение сохраняет каждый шаг алгоритма, включая значения переменных, оценки, базисные переменные и другие параметры.
* Эти данные могут быть сохранены в виде структуры или объекта для дальнейшего использования и отображения пользователю.

1. Вывод полученных результатов и записанного хода решения задачи:

* По завершении решения задачи оптимизации приложение выводит результаты оптимизации, такие как оптимальное значение целевой функции и значения переменных.
* Также выводится записанный ход решения, который позволяет пользователю понять, каким образом было получено оптимальное решение.
* Результаты могут быть представлены в текстовом или графическом виде, в зависимости от требований к интерфейсу приложения.

Такой процесс обработки информации обеспечивает пользователю возможность ввода данных, получения подробного отчёта о ходе решения задачи и вывода результата оптимизации.

Данный процесс можно представить в виде функциональной диаграммы (рисунок 2) и её декомпозиции (рисунок 3).

#### 

#### Рисунок 2 – Функциональная диаграмма приложения, обобщенная

#### 

#### Рисунок 3 – Функциональная диаграмма приложения, декомпозиция

Также была составлена концептуальная диаграмма классов, отображающая подлежащие к реализации модули приложения (рисунок 4).

#### 

#### Рисунок 4 – Концептуальная диаграмма классов приложения

## 1.3 Анализ хранимой информации и выбор структур данных для ее представления

Основной информацией, которая необходима для написания программного кода симплекс-метода, является базис, а также сама пересчетная таблица. Информация такого рода будет хранится в течении всего времени работы программы, однако, есть необходимость хранить также промежуточные данные, использующиеся в циклах программы, пока ответ не будет найден[2].

При выборе структуры данных для представления базиса и пересчетной таблицы, стоит обратить внимание на следующее: структуры имеют четко определённые, заранее заданные размеры, имеется необходимость обращаться к элементам по индексам для использования в циклах, а также для изменения значения элемента. Для таких целей наилучшим выбором будет массив, который имеет все вышеперечисленные функции. Базис содержит лишь одно измерение, в то время как таблица содержит два, и будет хранится в многомерном массиве.

Для определения новой базисной переменной, точнее указания столбца с наибольшим коэффициентом целевой функции, есть необходимость хранить только лишь индекс выбранного столбца. Для таких целей достаточно целочисленной переменной.

## 1.4 Выбор методов и алгоритмов решения задачи

Поставленная в условиях варианта курсовой работы задача предрасполагает к использованию симплекс-метода, поскольку её очень просто перевести математическую модель, подлежащую решению данным способом. Также можно отметить другие причины выбора именно этого варианта решения поставленной задачи:

1. Эффективность: Симплекс-метод является одним из наиболее эффективных методов решения линейных задач оптимизации. Он хорошо работает даже для задач с большим количеством переменных и ограничений.
2. Гарантированная сходимость: Симплекс-метод обеспечивает гарантированную сходимость к оптимальному решению в конечном числе шагов для линейных задач оптимизации.
3. Широкое применение: Симплекс-метод применим для решения разнообразных задач оптимизации, таких как задачи линейного программирования в экономике, производстве, логистике и других областях.
4. Доступность: Симплекс-метод широко изучен и реализован во многих библиотеках и программных средствах для решения задач оптимизации. Это обеспечивает доступность и удобство его использования в различных прикладных задачах.
5. Понятность и интерпретируемость: Решение задачи симплекс-методом обычно сопровождается последовательностью шагов, каждый из которых имеет понятную геометрическую или алгебраическую интерпретацию. Это позволяет легче понять процесс оптимизации и результаты его работы.

Исходя из этих факторов, симплекс-метод является логичным и эффективным выбором для решения задачи оптимизации в рамках данной курсовой работы.

Симплекс-метод начинается с выбора начального базисного допустимого решения, которое обеспечивает нулевые значения переменных, не являющихся базисными[3]. Затем алгоритм оценивает, как изменение базиса может улучшить целевую функцию, используя метод разрешающего столбца. После выбора разрешающего столбца определяется разрешающая строка, которая ограничивает шаг в направлении улучшения. Далее происходит пересчет базиса и переменных, чтобы найти новое допустимое решение. Алгоритм продолжает эти шаги, двигаясь от одного базисного допустимого решения к другому, пока не достигнет оптимального решения или определит, что проблема не ограничена или недопустима[1].

Описанный выше алгоритм также можно представить в виде блок-схемы (рисунок 5).

#### 

#### Рисунок 5 – Блок-схема алгоритма симплекс-метода

В данном разделе был проведён анализ технических требований и уточнение спецификаций продукта. Сначала был проведён анализ задания и выбор технологий, языка программирования и среды разработки. Их результатом стал выбор объектно-ориентированной парадигмы программирования, выбор языка программирования C# и использование среды разработки Visual Studio 2022. Затем был проанализирован процесс обработки информации, на основе которого были построены функциональные диаграммы программы. Далее был проведён анализ хранимой информации и выбор структур данных для её представления – двумерные массивы.

# 2 Проектирование структуры и компонентов программного продукта

## 2.1 Разработка интерфейса пользователя

### 2.1.1 Разработка структурной схемы интерфейса

При разработке интерфейса программы должны быть учтены все выполняемые ею функции:

* Визуализация таблицы симплекс-метода в отчете;
* Таблица для ввода решения;
* Область для вывода решения;
* Функция ввода экстремума;
* Вспомогательная функция ввода задачи-примера;
* Поддержка произвольного количества условий и переменных;

Учитывая все функции приложения была построена структурная схема интерфейса (рисунок 6).

#### 

#### Рисунок 6 – Структурная схема интерфейса главного окна

Вариант приложения, состоящего из одного окна, был выбран по следующим причинам:

* Удобство: Пользователю не нужно переключаться между различными окнами или страницами, чтобы выполнить различные задачи. Все функции доступны в одном месте, что упрощает навигацию и улучшает пользовательский опыт.
* Простота: Единая форма уменьшает сложность интерфейса, так как пользователь видит только необходимые элементы управления и информацию для выполнения задачи. Это снижает вероятность запутанности и ошибок при взаимодействии с системой.
* Минимализм: Одно окно может быть спроектировано с учетом минималистичного дизайна, что делает интерфейс более эстетичным и приятным для использования. Минимизация лишних элементов также помогает сосредоточить внимание пользователя на основных функциях.

В целом, единая форма интерфейса может обеспечить более эффективное взаимодействие с системой, улучшая удобство, простоту и моду на минимализм. Это особенно важно в контексте современных требований к пользовательским интерфейсам, где пользователи ценят интуитивность и эффективность.

### 2.1.2 Построение графа (диаграммы) состояний интерфейса

При построении диаграммы состояний интерфейса следует учитывать специфику самого условия задачи. Оно состоит из всего одного базового варианта. Наличие этого условия добавляет необходимость реализации способа автоматического ввода начальных условий задачи.

Также в целях создания более гибкого приложения должна быть реализована функция редактирования и очищения таблицы условий задачи. Все вышеперечисленные состояния могут быть отражены диаграммой (рисунок 7).

#### 

#### Рисунок 7 – Диаграмма состояний интерфейса

### 2.1.3 Разработка форм ввода-вывода информации

При использовании данного приложения, большое значения имеет ввод-вывод информации, так как работа симплекс-метода без входных табличных значений, на основе которых выдается решение, невозможна. Тот же исход будет и при вводе некорректных данных, что тоже должно быть учтено.

Приложение имеет несколько состояний:

* Заполнение таблицы условий;
* Заполнение таблицы целевой функции;
* Изменение количества условий;
* Изменение количества переменных;
* Изменение искомого экстремума;
* Вывод решения;
* Вывод отчета;
* Вывод моделирования прибыли;
* Сообщение об ошибке.

Основное окно приложения также является и формой ввода и вывода информации. На основании данного факта и перечисленных в данном и предыдущем пунктах требований была составлена диаграмма состояний приложения (рисунок 8).

#### 

#### Рисунок 8 – Диаграмма состояний формы интерфейса

## 2.2 Разработка алгоритма основной программы и структурной схемы программного продукта

### 2.2.1 Описание структуры приложения и схема связности модулей

При проектировании структуры будущего программного продукта стоит описать основные функциональные элементы продукта, а также характер их взаимодействия.

Основные функциональные элементы приложения касаются ввода информации и ее очистки, получения выходных данных, а также выведения отчета, составленного на основе этих данных. Все вышеперечисленные элементы можно отобразить на структурной схеме, которая в полной мере отражает их взаимодействие (рисунок 9).

#### 

#### Рисунок 9 – Структурная схема программного продукта

### 2.2.2 Схема движения информационных потоков

Схема движения информационных поток, содержит несколько уровней детализации. Главный уровень содержит абстракцию, которая заключает в себе весь алгоритм программы, а также входной и выходной потоки. Вследствие того, что абстракция содержит в себе весь программный код, табличные данные – будут являться входными данными, а отчет – выходными (рисунок 10).

#### 

#### Рисунок 10 – Диаграмма потоков данных (уровень А0)

Далее главный уровень (А0) следует детализировать согласно состояниям программы и структурной схеме программного продукта. Диаграмма, приведенная на рисунке 9, отражает информационные потоки на уровне работы алгоритмов программы. Алгоритм получает входные данные в табличном формате, удобном для пользователя, затем заполняет структуры данных, заранее определенных программой, после чего управление передается алгоритму симплекс-метода. Результаты работы алгоритма передаются далее в форматируемый отчет (рисунок 11).

#### 

#### Рисунок 11 – Диаграмма потоков данных, декомпозиция уровня A0

## 2.3 Разработка основных алгоритмов программного продукта

Основной алгоритм использования программы можно представить в виде блок-схемы. Он основан на относительной независимости функционала пользовательских возможностей друг от друга (рисунок 12).

#### 

#### Рисунок 12 – Блок-схема основного алгоритма программы

Основной алгоритм программы содержит в себе несколько подпрограмм, каждая из которых выполняет одно определенное действие.

1. Заполнить.

Используемые переменные: глобальная таблица типа DataGridView (constraintsGridView), глобальная таблица типа DataGridView (functionGridView).

Схема алгоритма (рисунок 13).

#### 

#### Рисунок 13 – Алгоритм подпрограммы автоматического заполнения условий задачи

Функциональное назначение алгоритма: алгоритм предназначен для автоматического заполнения условий задачи из заданного варианта.

Входные данные: отсутствуют.

Выходные данные: заполненные таблицы условий задачи и целевой функции задачи в соответствии с заданным вариантом курсовой работы.

2. Алгоритм очистки данных таблиц.

Используемые переменные: глобальная таблица типа DataGridView (constraintsGridView), глобальная таблица типа DataGridView (functionGridView).

Схема алгоритма (рисунок 14).

#### 

#### Рисунок 14 – Схема алгоритма очищения данных таблиц

Функциональное назначение алгоритма: алгоритм предназначен для полной очистки данных двух таблиц: constraintsGridView и functionGridView. После очищения предыдущие данные не будут доступны в течении работы программы.

Входные данные: отсутствуют.

Выходные данные: пустые таблицы, ранее содержавшие некоторые данные.

3. Алгоритм получения решения.

Используемые переменные: глобальная таблица типа DataGridView (constraintsGridView), глобальная таблица типа DataGridView (functionGridView), массив вещественного типа данных для хранения значений симплекс-таблицы, две целочисленные переменные для хранения индексов разрешающего элемента.

Схема алгоритма (рисунок 15).

#### 

#### Рисунок 15 – Алгоритм подпрограммы получения решения

Функциональное назначение алгоритма: алгоритм предназначен для решения задачи симплекс-методом, полученные значения также используются для расчета других значений.

Входные данные: обработанные данные из таблиц constraintsGridView и functionGridView.

Выходные данные: набор промежуточных симплекс-таблиц, двумерный массив со значениями готового решения.

## 2.4 Разработанные классы программы

Классы рабочей части программы создавались на основе изученной предметной области решения задач оптимизации симплекс-методом. В результате анализа процесс решения была составлена диаграмма классов (рисунок 16), подлежащих реализации для успешного применения симплекс-метода в поставленных в работе задачах. Конкретно были выделены сущности:

* «Simplex» - класс, в котором производятся расчеты в соответствии с методом.
* «Function» - класс, содержащий целевую функцию математической модели.
* «Constraint» - класс, содержащий условия математической модели.
* «SimplexIndexResult» - класс, отвечающий за переходы между шагами симплекс-алгоритма.
* «SimplexSnap» - класс, сохраняющий результаты работы симплекс-метода в удобном для работы программы формате.

#### 

#### Рисунок 16 – Диаграмма рабочих классов программы

Также были разработаны классы интерфейса программы и их внутренние компоненты. Так, были созданы классы «MainForm» и «ReportForm», отвечающие за основное окно и за окно отчета соответственно. Отчет отделён от остальной части решения задачи из-за его объёма. Его внедрение в основное окно вылилось бы в большую площадь зачастую неиспользуемого места на экране и неудобство использования. Для этих классов также была составлена диаграмма (рисунок 17).

#### 

#### Рисунок 17 – Диаграмма классов форм программы

Также была составлена диаграмма последовательностей, поясняющая переход между окнами при выведении отчета ( рисунок 18).

#### 

#### Рисунок 18 – Диаграмма последовательностей перехода между окнами

Итогом создания классов программы стала диаграмма компоновки программного продукта (рисунок 19), показывающая связи между файлами разработанного продукта.

#### 

#### Рисунок 19 – Диаграмма компоновки программного продукта

Программный продукт создавался по восходящей методологии, так как симплекс-метод – это математический алгоритм, состоящий из повторяющихся действий. Поэтому сначала были реализованы базовые функции, на основании которых был собрана программная реализация алгоритма[4]. Также можно отметить и несколько второстепенных преимуществ выбранной методологии:

* Постепенное улучшение функциональности: Восходящий подход позволяет начать с минимальной жизнеспособной версии (MVP) приложения, которая решает базовые задачи оптимизации. Затем функциональность приложения может постепенно расширяться и улучшаться на основе обратной связи от пользователей и изменений в требованиях.
* Быстрое время выхода на рынок: Разработка MVP позволяет быстрее запустить продукт на рынок и начать собирать обратную связь от пользователей. Это особенно важно в области оптимизации, где компании постоянно сталкиваются с вызовами и нуждаются в эффективных решениях как можно скорее.
* Гибкость и адаптивность: Восходящий подход делает приложение более гибким и адаптивным к изменяющимся требованиям и условиям рынка. Вместо того чтобы пытаться предусмотреть все возможные сценарии заранее, команды разработчиков могут быстро реагировать на изменения, внедряя новые функции и оптимизируя существующие.
* Минимизация рисков: Разработка поэтапно позволяет уменьшить риски, связанные с разработкой большого и сложного приложения. Путем начала с простой версии MVP можно быстро выявить проблемы и исправить их, минимизируя риски больших инвестиций в разработку.
* Легкое масштабирование и оптимизация: Постепенное улучшение приложения позволяет более эффективно масштабировать и оптимизировать его. После того как базовая функциональность установлена и протестирована, можно сконцентрироваться на улучшении производительности, интерфейса пользователя и других аспектов приложения.

Подводя итоги, в данном разделе было проведено проектирование структуры и компонентов программного продукта. Был разработан интерфейс – его структурная схема, диаграмма его состояний и формы ввода-вывода информации. Был разработан алгоритм основной программы и структурная схема продукта. Произведено описание структуры приложения и составлена схема связности модулей. Разработаны основные алгоритмы программного продукта.

# 3 Тестирование программы

## 3.1 Разработка плана тестирования

Целью данного плана тестирования является обеспечение высокого качества разрабатываемого продукта путем проверки его функциональности, производительности, безопасности и удобства использования. Основные задачи включают:

* Проверку корректности работы отдельных компонентов системы (модульные тесты).
* Проверку работоспособности и пользовательского опыта взаимодействия с интерфейсом (тестирование интерфейса).

Модульные тесты будут проводиться для каждого отдельного модуля системы с целью проверки их функциональности в изоляции от других компонентов. Этот вид тестирования выбран в первую очередь потому, что при разработке использовалась восходящая методология разработки. План модульного тестирования включает в себя:

* Создание тестовых случаев для каждого модуля на основе его спецификации и требований.
* Определение критериев приемки для каждого теста.
* Разработку средств автоматизации для проведения модульных тестов.
* Оценку покрытия кода модульными тестами.
* Регулярное обновление и поддержание тестовых случаев в соответствии с изменениями в коде.

Тестирование интерфейса будет проводиться для проверки удобства использования, соответствия дизайну и функциональности пользовательского интерфейса. План тестирования интерфейса включает в себя:

* Создание тестовых сценариев, охватывающих различные аспекты взаимодействия пользователя с интерфейсом.
* Проверку адаптивности интерфейса к различным разрешениям экранов.
* Оценку времени отклика и общей производительности интерфейса.

## 3.2 Разработка алгоритма процедуры тестирования

Учитывая этапы, описанные выше, можно составить следующий алгоритм проверки программы:

1. Определение функциональных требований (пункт 1).

2. Разработка модульных тестов (Приложение Г).

3. Проведение модульного тестирования (рисунок 20).

4. Тестирование интерфейса (приложение А).

5. Документирование тестирования (таблицы приложения А, рисунок 20).

Модульные тесты будут спроектированы и реализованы на основе фреймворка xUnit. Он предоставляет программистам инструменты для написания и запуска тестов для различных типов приложений на разных языках программирования. Каждый фреймворк XUnit предоставляет базовые функциональности для организации тестов, а также генерации отчетов о выполненных тестах, включая информацию о количестве успешных и неудачных тестов, времени выполнения и другие статистические данные[5].

При тестировании интерфейса используется ручное тестирование за неимением возможности автоматизации данного процесса. Процесс, условия и алгоритм тестирований записан в единообразных планах в формате таблиц.

## 3.3 Оценка результатов тестирования

Ниже приведены результаты выполнения разработанных для приложения модульных тестов (рисунок 20).

#### 

#### Рисунок 20 – Результат выполнения модульных тестов приложения

Планы и результаты тестирования интерфейса представлены в приложении А.

В данном разделе было проведено разноплановое тестирование программы. Были разработаны планы тестирования, разработан алгоритм процедуры тестирования и проведена оценка результатов. Основываясь на результатах тестирования приложения, можно сделать вывод о том, что оно полностью готово к эксплуатации, а ошибки, которые возможно обнаружатся в будущем будут несущественны и некритичны.

# 4 Сопровождение

## 4.1 Руководство пользователя

Ниже представлено пошаговое описание типичного способа использования приложения с пояснениями по каждой функции приложения.

1. Вход в приложение. При запуске программы открывается главное окно приложения (рисунок 21).

#### 

#### Рисунок 21 – Главное окно приложения

2. Заполнение таблицы. Для удобства использования в программе предоставлена возможность автоматического заполнение таблиц (кнопка «OK»).

3. Получение отчета. При нажатии на кнопку «Go» появляется окно отчета, содержащее результаты решения задачи оптимизации (рисунок 22).

#### 

#### Рисунок 22 – Окно с результатами

5. Моделирование прибыли. При нажатии на кнопку «Смоделировать» появляется в текстовом окне, где появляется решение задачи, появляется её же решение, но с условием того, что на каждой следующей итерации количество человеческого ресурса, доступного для каждого цеха, увеличивается на единицу.

В случае ручного заполнения условий задачи нужно воспользоваться переключателями рядом с надписями «Условия» и «Переменные», выбрав необходимое количество переменных. После чего ввести в получившуюся пустую таблицу условия из математической модели задачи. Знак неравенства выбирается из 3 вариантов: «<= (меньше или равно)», «>=» (больше или равно), «=» (равно). После чего необходимо ввести коэффициенты целевой функции и выбрать необходимый экстремум.

## 4.2 Обслуживание модели, алгоритма, программы и их эксплуатация

При использовании встроенной модели, заполняющейся автоматически, не стоит менять местами уже имеющиеся условия и добавлять новые только снизу. Иначе могут возникнуть ошибки в дополнительной информации, выдаваемой отчетом.

Алгоритм не нуждается в доработках в функциональном плане, однако может быть подвергнут процессу оптимизации его внутренней реализации. Уже имеющиеся модульные тесты будут хорошим подспорьем в процессе рефакторинга алгоритма.

Обслуживание программы и её доработка должны проводиться только с целью реализации новых функций или исправления ошибок старой реализации, но не с целью удаления оригинальных функций, так как они и так являются минимально необходимыми.

В результате данного раздела было составлено руководство пользователя, описывающее и разъясняющее работу с разработанным продуктом. Были определены и описаны процессы обслуживания модели, алгоритма и программы, а также их эксплуатация.

Заключение

В результате выполнения курсовой работы было спроектировано и реализовано приложение по решению задачи оптимизации используя симплекс-метод.

В первом разделе был проведён анализ технических требований и уточнение спецификаций продукта. Сначала был проведён анализ задания и выбор технологий, языка программирования и среды разработки. Их результатом стал выбор объектно-ориентированной парадигмы программирования, выбор языка программирования C# и использование среды разработки Visual Studio 2022. Затем был проанализирован процесс обработки информации, на основе которого были построены функциональные диаграммы программы. Далее был проведён анализ хранимой информации и выбор структур данных для её представления – двумерные массивы.

Во втором разделе было проведено проектирование структуры и компонентов программного продукта. Был разработан интерфейс – его структурная схема, диаграмма его состояний и формы ввода-вывода информации. Был разработан алгоритм основной программы и структурная схема продукта. Произведено описание структуры приложения и составлена схема связности модулей. Разработаны основные алгоритмы программного продукта.

В третьем разделе было проведено разноплановое тестирование программы. Были разработаны планы тестирования, разработан алгоритм процедуры тестирования и проведена оценка результатов. Основываясь на результатах тестирования приложения, можно сделать вывод о том, что оно полностью готово к эксплуатации, а ошибки, которые возможно обнаружатся в будущем будут несущественны и некритичны.

В четвертом разделе было составлено руководство пользователя, описывающее и разъясняющее работу с разработанным продуктом. Были определены и описаны процессы обслуживания модели, алгоритма и программы, а также их эксплуатация.

После поэтапного решения задач был получен готовый продукт, соответствующий поставленным требованиям, однако он готов к дальнейшим изменениям и усовершенствованиям при необходимости, например в плане дизайна интерфейса или добавления возможности решения других типов задач оптимизации. Продукт был протестирован в соответствии с методологиями, изученными в процессе обучения. Также важно отметить полученные посредством изучения предметной области в процессе работы знания и опыт разработки.

Список используемой литературы

1. Аттетков А. В. Методы оптимизации: Учебное пособие / А.В. Аттетков, В.С. Зарубин, А.Н. Канатников. - М.: ИЦ РИОР: НИЦ Инфра-М, 2019. - 270 с.: ил.; - (Высшее образование: Бакалавриат). - ISBN 978-5-16-103309-8. - Текст : электронный. - URL: https://znanium.com/bookread2.php?book=1002733

2. Балдин К. В. Математическое программирование / Балдин К.В., Брызгалов Н.А., Рукосуев А.В., - 2-е изд. - Москва :Дашков и К, 2018. - 218 с.: ISBN 978-5-394-01457-4. - Текст : электронный. - URL: https://znanium.com/bookread2.php?book=415097

3. Карманов В. Г. Математическое программирование [Электронный ресурс] : Учебное пособие / В. Г. Карманов. - 6-е изд., испр. - Москва : ФИЗМАТЛИТ, 2018. - 264 с. - ISBN 978-5-9221-0983-3. - Текст : электронный. - URL: https://znanium.com/bookread2.php?book=544747

4. Каштанов В. А. Исследование операций (линейное программирование и стохастические модели) : учебник / В.А. Каштанов, О.Б. Зайцева. — Москва : КУРС, 2017. - 256 с. - ISBN 978-5-906818-78-2. - Текст : электронный. - URL: https://znanium.com/bookread2.php?book=1017099

5. Морозова, Ю. В. Тестирование программного обеспечения : учебное пособие / Ю. В. Морозова. - Томск : Эль-Контент, 2019. - 120 с. - ISBN 978-5-4332-0279-5. - Текст : электронный. - URL: https://znanium.com/catalog/product/1845910

6. Хорев, П. Б. Объектно-ориентированное программирование с примерами на С# : учебное пособие / П.Б. Хорев. — Москва : ФОРУМ : ИНФРА-М, 2023. — 200 с. — (Среднее профессиональное образование). - ISBN 978-5-00091-713-8. - Текст : электронный. - URL: https://znanium.com/catalog/product/1895650

Приложение А – Планы тестирования интерфейса

Таблица 1 – План тестирования функции решения задачи

| Наименование | Описание |
| --- | --- |
| Наименование проекта | Приложение решения задач оптимизации |
| Номер версии | 1.0 |
| Имя тестера | Карданов Артур Ауесович |
| Даты тестирования | 28.04.2024 |
| Test Case # | TC\_RS\_1 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка корректности выведения решения |
| Резюме испытания | Корректное решение, соответствующее действительному результату |
| Шаги тестирования | 1. Запуск приложения. 2. Нажатие на кнопку «Заполнить таблицы».   3. Нажатие на кнопку «Решить». |
| Данные тестирования | Первоначальные данные полученного на выполнение варианта курсовой работы. |
| Ожидаемый результат | Вывод на экран в нижнем текстовом разделе корректного решения задачи. |
| Фактический результат | Корректное решение выведено. |
| Предпосылки | Отсутствуют. |
| Постусловия | Отсутствуют. |
| Статус | Pass |
| Комментарии | Отсутствуют. |

Таблица 2 – План тестирования функции создания отчета по решению задачи

| Наименование | Описание |
| --- | --- |
| Наименование проекта | Приложение решения задач оптимизации |
| Номер версии | 1.0 |
| Имя тестера | Карданов Артур Ауесович |
| Даты тестирования | 28.04.2024 |
| Test Case # | TC\_RT\_1 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка создания отчета по решению поставленной задачи |
| Резюме испытания | Готовый правильный отчет. |
| Шаги тестирования | 1. Запуск приложения.  2. Нажатие на кнопку «Заполнить таблицы».  3. Нажатие на кнопку «Решить».  4. Нажатие на кнопку «Отчет». |
| Данные тестирования | Первоначальные данные полученного на выполнение варианта курсовой работы. |
| Ожидаемый результат | Вывод на экран нового окна с отчетом по нахождении решения задачи, содержащего ход решения задачи в виде промежуточных таблиц и их текстового описания |
| Фактический результат | Окно отчета выведено, промежуточные таблицы когерентны с решением вручную, качество текстового описания удовлетворительное. |
| Предпосылки | Отсутствуют. |
| Постусловия | Отсутствуют. |
| Статус | Pass |
| Комментарии | Отсутствуют |

Таблица 3 – План тестирования функции моделирования повышения прибыли

| Наименование | Описание |
| --- | --- |
| Наименование проекта | Приложение решения задач оптимизации |
| Номер версии | 1.0 |
| Имя тестера | Карданов Артур Ауесович |
| Даты тестирования | 28.04.2024 |
| Test Case # | TC\_MD\_1 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка функции моделирования роста прибыли |
| Резюме испытания | Адекватная работа функции моделирования прибыли |
| Шаги тестирования | 1. Запуск приложения.  2. Нажатие на кнопку «Заполнить таблицы».  3. Нажатие на кнопку «Смоделировать». |
| Данные тестирования | Первоначальные данные полученного на выполнение варианта курсовой работы. |
| Ожидаемый результат | Вывод на экран в нижнем текстовом разделе корректного решения задачи для 10 её вариантов, отличающихся постепенно увеличивающимся количеством доступной для цехов рабочей силы.. |
| Фактический результат | Корректное решение выведено, функция моделирования работает удовлетворительно. |
| Предпосылки | Отсутствуют |
| Постусловия | Отсутствуют |
| Статус | Pass |
| Комментарии | Отсутствуют |

Приложение Б – Листинг приложения

##### Б.1 Модуль Constraint

using System;

namespace SimplexMethod

{

public class Constraint

{

public double[] variables;

public double b;

public string sign;

public Constraint(double[] variables, double b, string sign)

{

if (sign == "=" || sign == "<=" || sign == ">=")

{

this.variables = variables;

this.b = b;

this.sign = sign;

} else

{

throw new ArgumentException("Wrong sign");

}

}

}

}

##### Б.2 Модуль Function

namespace SimplexMethod

{

public class Function

{

public double[] variables;

public double c;

public bool isExtrMax;

public Function(double[] variables, double c, bool isExtrMax)

{

this.variables = variables;

this.c = c;

this.isExtrMax = isExtrMax;

}

}

}

##### Б.3 Модуль SimplexIndexResult

using System;

namespace SimplexMethod

{

public class SimplexIndexResult

{

public Tuple<int, int> index;

public SimplexResult result;

public SimplexIndexResult(Tuple<int, int> index, SimplexResult result)

{

this.index = index;

this.result = result;

}

}

}

##### Б.4 Модуль SimplexSnap

using System.Linq;

namespace SimplexMethod

{

public class SimplexSnap

{

public double[] b;

public double[][] matrix;

public double[] M;

public double[] F;

public int[] C;

public double fValue;

public double[] fVars;

public bool isMDone;

public bool[] m;

public SimplexSnap(double[] b, double[][] matrix, double[] M, double[] F, int[] C, double[] fVars, bool isMDone, bool[] m)

{

this.b = Copy(b);

this.matrix = Copy(matrix);

this.M = Copy(M);

this.F = Copy(F);

this.C = Copy(C);

this.isMDone = isMDone;

this.m = Copy(m);

this.fVars = Copy(fVars);

fValue = 0;

for (int i = 0; i < C.Length; i++)

{

fValue += fVars[C[i]] \* b[i];

}

}

T[] Copy<T>(T[] array)

{

T[] newArr = new T[array.Length];

for (int i = 0; i < array.Length; i++)

{

newArr[i] = array[i];

}

return newArr;

}

T[][] Copy<T>(T[][] matrix)

{

T[][] newMatr = new T[matrix.Length][];

for (int i = 0; i < matrix.Length; i++)

{

newMatr[i] = new T[matrix.First().Length];

for (int j = 0; j < matrix.First().Length; j++)

{

newMatr[i][j] = matrix[i][j];

}

}

return newMatr;

}

}

}

Приложение В – Листинг модульных тестов

##### В.1 Модуль UnitTest1

using FluentAssertions;

using Ref\_LR9;

using Ref\_LR9.Bills;

using Ref\_LR9.FileSource;

namespace SimplexTests

{

public class ResultTests

{

public Customer TestCustNoBonuses = new("Jej", 0);

public RegularGoods TestGoodRegular = new("TestGoodRegNoDisc", new AmountForQuantity(), new PercentForQuantity());

public SaleGoods TestGoodSale = new("TestGoodRegNoDisc", new AmountForQuantity(), new PercentForQuantity());

public SpecialGoods TestGoodSpecial = new("TestGoodRegNoDisc", new AmountForQuantity(), new PercentForQuantity());

IView view = new TxtView();

[Fact]

public void SimplexTableTest()

{

Bill bill = new Bill(TestCustNoBonuses);

Item item = new Item(TestGoodRegular, 2, 10);

bill.addGoods(item);

BillGenerator billGenerator = new(view, bill);

string[] str = billGenerator.GetBill().Split(new char[3] {' ', '\t', '\n' });

Assert.Equal(1, Convert.ToInt32(str[^3])); // bonus check

Assert.Equal(20, Convert.ToDouble(str[^6])); // price check

}

[Fact]

public void SimplexTablesTransitionTests()

{

Bill bill = new Bill(TestCustNoBonuses);

Item item = new Item(TestGoodRegular, 20, 10);

bill.addGoods(item);

BillGenerator billGenerator = new(view, bill);

string[] str = billGenerator.GetBill().Split(new char[3] { ' ', '\t', '\n' });

Assert.Equal(10, Convert.ToInt32(str[^3])); // bonus check

Assert.Equal(194, Convert.ToDouble(str[^6])); // price check

}

[Fact]

public void SimplexCalculationTest()

{

Bill bill = new Bill(TestCustNoBonuses);

Item item = new Item(TestGoodSpecial, 2, 10);

bill.addGoods(item);

BillGenerator billGenerator = new(view, bill);

string[] str = billGenerator.GetBill().Split(new char[3] { ' ', '\t', '\n' });

Assert.Equal(0, Convert.ToInt32(str[^3])); // bonus check

Assert.Equal(20, Convert.ToDouble(str[^6])); // price check

}

}

public class SimplexSnapsTests

{

public Customer TestCustWithBonuses = new("Jej", 100);

public RegularGoods TestGoodRegular = new("TestGoodRegNoDisc", new AmountForQuantity(), new PercentForQuantity());

public SaleGoods TestGoodSale = new("TestGoodRegNoDisc", new AmountForQuantity(), new PercentForQuantity());

public SpecialGoods TestGoodSpecial = new("TestGoodRegNoDisc", new AmountForQuantity(), new PercentForQuantity());

public IView view = new TxtView();

[Fact]

public void SnapCopyTest()

{

Bill bill = new Bill(TestCustWithBonuses);

Item item = new Item(TestGoodRegular, 2, 10);

bill.addGoods(item);

BillGenerator billGenerator = new(view, bill);

string[] str = billGenerator.GetBill().Split(new char[3] { ' ', '\t', '\n' });

Assert.Equal(1, Convert.ToInt32(str[^3])); // bonus check

Assert.Equal(20, Convert.ToDouble(str[^6])); // price check

}

[Fact]

public void SnapArrayTest()

{

Bill bill = new Bill(TestCustWithBonuses);

Item item = new Item(TestGoodRegular, 20, 10);

bill.addGoods(item);

BillGenerator billGenerator = new(view, bill);

string[] str = billGenerator.GetBill().Split(new char[3] { ' ', '\t', '\n' });

Assert.Equal(10, Convert.ToInt32(str[^3])); // bonus check

Assert.Equal(94, Convert.ToDouble(str[^6])); // price check

}

[Fact]

public void SimplexIndexTest()

{

Bill bill = new Bill(TestCustWithBonuses);

Item item = new Item(TestGoodSpecial, 2, 10);

bill.addGoods(item);

BillGenerator billGenerator = new(view, bill);

string[] str = billGenerator.GetBill().Split(new char[3] { ' ', '\t', '\n' });

Assert.Equal(0, Convert.ToInt32(str[^3])); // bonus check

Assert.Equal(0, Convert.ToDouble(str[^6])); // price check

}

}

public class ParsingTests

{

[Fact]

public void ConditionParsingTest()

{

string str = "CustomerName: Test\nCustomerBonus: 10\nGoodsTotalCount: 0\n#\nItemsTotalCount: 0\n#";

StringReader sr = new(str);

IFileSource fs = new YamlFileSource(sr);

Bill bill = BillFactory.CreateBill(fs, new DateTime(2024,12,01));

bill.\_customer.getName().Should().Be("Test");

bill.\_customer.getBonus().Should().Be(10);

}

[Fact]

public void FunctionParsingTest()

{

string str = "CustomerName: Test\nCustomerBonus: 10\nGoodsTotalCount: 0\n#\nItemsTotalCount: 0\n#";

StringReader sr = new(str);

IFileSource fs = new YamlFileSource(sr);

Bill bill = BillFactory.CreateBill(fs, new DateTime(2024, 12, 01));

bill.\_items.Count.Should().Be(0);

}

[Fact]

public void ExtremumParsingTest()

{

string str = "CustomerName: Test\nCustomerBonus: 10\nGoodsTotalCount: 1\n#\n1: Cola REG\nItemsTotalCount: 1\n#\n1: 1 65 6";

StringReader sr = new(str);

IFileSource fs = new YamlFileSource(sr);

Bill bill = BillFactory.CreateBill(fs, new DateTime(2024, 12, 01));

bill.\_items[0].getPrice().Should().Be(65);

bill.\_items[0].getQuantity().Should().Be(6);

bill.\_items[0].getGoods().getTitle().Should().Be("Cola");

}

}

}

Приложение Г – Антиплагиат

#### 