МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование кафедры)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование
(наименование профиля, специализации)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему: «Построение алгоритма и программная реализация решения задачи управления запасами на основе динамического программирования»

Обучающийся С. В. Сегеда (И.О. Фамилия) _____ (личная подпись)

Руководитель к.т.н., доцент, Н.А. Сосина
(ученая степень, звание, И.О. Фамилия)

Консультант к.п.н., доцент, А.В. Егорова
(ученая степень, звание, И.О. Фамилия)

Тольятти 2024

Аннотация

Тема выпускной квалификационной работы: «Построение алгоритма и программная реализация решения задачи управления запасами на основе динамического программирования».

Работа выполнена бакалавром Тольяттинского Государственного Университета, кафедры "Прикладная математика и информатика", группы ПМИб-2002б, Сегедой Сергеем Викторовичем.

Актуальность исследования данной темы подчеркивается необходимостью повышения эффективности управления запасами в условиях динамично меняющейся экономической среды. Традиционные методы управления запасами не всегда способны обеспечить оптимальные результаты в условиях переменных рыночных условий, изменяющихся потребительских предпочтений и технологических инноваций.

Выпускная квалификационная работа посвящена решению задачи управления запасами на основе динамического программирования, а также программной реализации решения алгоритма.

Целью выпускной квалификационной работы является построение алгоритма решения задачи управления запасами на основе динамического программирования и выполнение программной реализации разработанного алгоритма.

Объект исследования – математическое моделирование управления запасами.

Предмет исследования – алгоритм метода динамического программирования.

Задачи работы:

а) изучить метод динамического программирования;

б) изучить методы управления запасами;

в) построить математическую модель производственной задачи управления запасами;

г) разработать алгоритм решения сформулированной задачи методом динамического программирования;

д) выполнить программную реализацию разработанного алгоритма.

Результатом работы является готовое программное решение производственной задачи управления запасами с применением метода динамического программирования.

Бакалаврская работа содержит пояснительную записку объемом 0 страниц, включая 0 иллюстраций, 0 таблиц и список литературы.

Abstract

The title of the graduation work is "Construction of the algorithm and program implementation of the solution of the problem of inventory management on the basis of dynamic programming".

The graduate qualification work is devoted to a general formulation of the problem of inventory management based on dynamic programming.

The purpose of the graduate qualification work is to study the method of dynamic programming, to solve a specific production problem of inventory management by this method and to implement the program code.

The object of the study - problems of inventory management in production.

The subject of the study - algorithm of the dynamic programming method.

In the first part, what is production inventory management, the general formulation of the problem of inventory management by dynamic programming and other methods that could be used to solve the problem are discussed.

The second part solves the problem of inventory management by the method of dynamic programming, and tells about the optimality principle formulated by R. Bellman.

The third part shows the program implementation of the inventory control problem solved by the dynamic programming method.

The bachelor's work contains an explanatory note with a volume of 0 pages, including 0 illustrations, 0 tables, a list of references.

Оглавление

Введение

В современном мире управление запасами играет ключевую роль в эффективной деятельности предприятий, организаций и даже государственных структур. Эффективное управление запасами позволяет минимизировать издержки, оптимизировать производственные процессы и обеспечить непрерывность поставок товаров и услуг. Однако, несмотря на значимость этой задачи, существует ряд нерешенных проблем, связанных с оптимизацией управления запасами. К примеру, поиск максимума классическими методами часто приводит к тому, что на грядущем этапе решения приходится решать новые задачи, более сложные, чем исходные. Также для значительной части производственных задач управления запасами, классические методы не применимы или вовсе неэффективны. К таким производственным задачам, как нахождение максимума целевой функции, легче всего подойти поэтапно, а не решать систему уравнений аналитически, у которой к тому же имеются ограничения поставленной задачи.

Актуальность исследования данной темы подчеркивается необходимостью повышения эффективности управления запасами в условиях динамично меняющейся экономической среды. Традиционные методы управления запасами не всегда способны обеспечить оптимальные результаты в условиях переменных рыночных условий, изменяющихся потребительских предпочтений и технологических инноваций.

Объект исследования – производственная задача управления запасами, решаемая методом динамического программирования.

Предмет исследования – алгоритм метода динамического программирования.

Целью выпускной квалификационной работы является аналитическое решение задачи управления запасами на основе динамического программирования, обоснование данного метода и его программная реализация.

Для выполнения поставленной цели необходимо решить следующие задачи:

а) изучить общий подход метода динамического программирования;

б) изучить существующие подходы и методы управления запасами;

в) составить производственную задачу управления запасами, решаемую методом динамического программирования;

г) показать применение метода динамического программирования при решении производственной задачи управления запасами аналитически;

д) выполнить программную реализацию разработанного алгоритма.

Выпускная квалификационная работа состоит из введения, трех глав, заключения и списка используемых источников.

В главе 1 рассматривается общая постановка задачи управления запасами методом динамического программирования.

В главе 2 рассказывается про принцип оптимальности, сформулированный Р. Беллманом, и решается поставленная задача управления запасами методом динамического программирования.

В главе 3 демонстрируется алгоритм программы.

В заключении представлены результаты и выводы о выполненной работе.

Глава 1 Общая постановка задачи управления запасами методом динамического программирования

1.1 Метод динамического программирования

Термин "динамическое программирование" обязан своим появлением и утверждением в научном сообществе Ричарду Беллману, который впервые употребил его в 1940 году для задач, решаемых шаг за шагом с применением результатов каждого вычисления на следующем шаге. Впоследствии Беллман усовершенствовал это понятие, предложив общепринятую в настоящее время формулировку. Как следствие, Беллману потребовалось много времени, чтобы выбрать подходящее название. Вместо слова "планирование", которое ассоциировалось с тем, что происходило в то время в Советском Союзе, он предложил вариант "программирование". И слово "динамический" оказалось удачным не только потому, что передавало суть методологии, но и потому, что было понятным и его трудно было заменить чем-то другим. Для Беллмана было важно, чтобы его термин не мог быть неправильно истолкован, чтобы его смысл не был искажен. Так появилось понятие "динамическое программирование".

Динамическое программирование — это особый подход к решению задач. Единого определения динамического программирования не существует, но идея заключается в том, что оптимальное решение часто можно найти, рассмотрев все возможные способы решения задачи и выбрав лучший из них.

Проще говоря, суть динамического программирования заключается в том, что при решении задачи она разбивается на несколько более мелких подзадач, которые напрямую зависят друг от друга.

Работа динамического программирования очень похожа на рекурсию с запоминанием промежуточных решений - такую рекурсию еще называют мемоизацией. Рекурсивные алгоритмы обычно также разбивают задачу.

“Динамические методы строят решения шаг за шагом, избегая избыточных вычислений. Таким образом, можно рассматривать их как рекурсивные методы, начиная с самых низов. Общая постановка задачи динамического программирования выглядит так: имеется физическая система S, заданная определенными параметрами. Требуется найти оптимальное управление u^* из доступного множества управлений, чтобы перевести систему из начального состояния x_0 в конечное состояние x_N, при этом достигнув экстремума целевой функции (показателя эффективности управления)” [5].

“Алгоритм динамического программирования начинается с разбора исходной задачи на более мелкие подзадачи. Для каждой подзадачи определяется рекурсивное соотношение, позволяющее выразить ее решение через решения более мелких подзадач. Затем определяются базовые случаи, для которых решение известно напрямую, без рекурсивных вычислений. Далее реализуется рекурсивный алгоритм, который вычисляет решения подзадач, начиная с базовых случаев и двигаясь к исходной задаче. Полученные результаты сохраняются, чтобы избежать повторных вычислений. Наконец, результаты подзадач комбинируются для получения решения исходной задачи. При необходимости алгоритм может быть оптимизирован для повышения его эффективности” [6].

“Это означает, что при динамическом программировании исходная задача разбивается на более мелкие подзадачи, каждая из которых оптимизируется, а затем объединяется для получения оптимального решения исходной задачи” [19].

“Если большие задачи могут быть решены путем комбинирования решений их меньших подзадач, то методы динамического программирования могут быть использованы эффективно. Кроме того, существует важная связь между значениями задачи и значениями их подзадачами. В литературе по оптимизации это соотношение часто называют уравнением Беллмана” [2].

В таких методах, как "разделяй и властвуй", подзадачи могут решаться несколько раз. Динамическое программирование решает любую из этих

подзадач только один раз, уменьшая количество вычислений. Результаты сохраняются, чтобы избежать повторения вычислений на более поздних этапах, когда эти подзадачи необходимо решить.

“Динамическое программирование используется для решения оптимизационных задач, таких как поиск кратчайшего пути, когда существует множество возможных решений, но основное внимание уделяется только поиску оптимального решения, и эти задачи обычно характеризуются перекрывающимися подзадачами, это означает, что решение каждой подзадачи может быть использовано несколько раз” [5]. Вместо того чтобы создавать новые подзадачи, рекурсивный алгоритм решает одну и ту же подзадачу многократно, примерами чего являются вычисление чисел Фибоначчи и поиск оптимальной подструктуры. В результате оптимальное решение исходной задачи может быть построено на основе оптимальных решений ее подзадач.

Задачи, решаемые методом динамического программирования, характеризуются наличием определенных элементов, которые помогают структурировать и эффективно решать задачу. Ключевые элементы в задаче, это состояния, которые описывают текущее состояние системы в каждый момент времени. Состояние должно включать всю информацию, необходимую для принятия окончательных решений, и должно быть достаточно полным, чтобы отражать влияние принятых решений. Решения (средства управления), которые могут быть приняты в каждом состоянии. Эти решения влияют на переход из текущего состояния в следующее, а также на затраты или выгоды, связанные с каждым решением. Функция перехода, описывает, как текущее состояние и выбранное действие влияют на следующее состояние системы. Функция перехода моделирует динамику системы и является важным элементом динамического программирования. Функция затрат или выгод, которая оценивает каждое действие в каждом состоянии. Это могут быть затраты, которые необходимо минимизировать, или выгоды, которые необходимо максимизировать. Она используется для

оценки эффективности всех типов действий в контексте достижения конечной цели. Критерий оптимальности, это как строится решение проблемы в целом. Это может быть формулировка, направленная на минимизацию общих затрат или максимизацию общей выгоды. В зависимости от проблемы может потребоваться найти подходящую стратегию с точки зрения ожидаемой дисконтированной стоимости или достичь состояния за наименьшее количество времени. Граничные условия, необходимы для того, чтобы начать решение задачи и проверить, когда оно должно закончиться.

Метод динамического программирования эффективен для решения задач, в которых решение на одном этапе влияет на возможности и затраты на последующих этапах, и часто используется в управлении запасами, планировании ресурсов, оптимизации расписания, управлении инвестициями и многих других областях.

1.2 Управления запасами на предприятии

Управление запасами — важнейший компонент логистической системы любой организации, включающий планирование, организацию и контроль запасов с целью максимизации прибыли и минимизации затрат. Эффективное управление запасами позволяет компаниям поддерживать оптимальный уровень запасов, обеспечивая бесперебойное производство и удовлетворение спроса клиентов. Модель управления запасами, используемая организацией, должна в первую очередь учитывать специфику конкретного производства, поэтому невозможно говорить об универсальном алгоритме для всех. Можно выделить несколько основных этапов.

1. Прогнозирование спроса. Прогнозирование осуществляется для каждой группы выпускаемой продукции, исходя из текущих объемов продаж и наличия неудовлетворенного покупательского спроса.

2. Планирование реализации. На основе прогноза спроса необходимо определить, сколько продукции необходимо произвести для удовлетворения

потребностей покупателей.

3. Прогнозирование производства. На этом этапе прогнозируется производство с учетом площади, оборудования, количества сотрудников, возможных потерь, брака и наличия готовой продукции на складе.

4. Расчет количества запасов. Оптимальное количество зависит от плана производства, так как необходимо, прежде всего, обеспечить непрерывность и ритмичность производственного процесса, но не допустить образования излишков, превышающих необходимый страховой запас.

5. Составление графика закупок. При составлении оптимального графика закупок учитываются условия работы с поставщиком, в том числе минимальный размер платежа, стоимость доставки, порядок расчетов и т. д..

6. Контроль. Важно выстроить эффективную систему контроля за движением сырья, материалов и полуфабрикатов, позволяющую не отклоняться от оптимальных показателей.

Необходимость управлять запасами для экономии средств породила множество моделей и методов. Можно назвать наиболее популярные модели.

1. Модель Уилсона. Считается самой простой, поскольку не предполагает существования неопределенности. Модель позволяет определить оптимальный уровень запасов для всего ассортимента, а параметрами являются уровень спроса, стоимость размещения и затраты на хранение. Предполагается, что склад пополняется с определенной периодичностью, равными партиями. Эта формула подходит для компаний, выпускающих продукцию со стабильным спросом, когда нет риска возникновения непредвиденных ситуаций.

2. Модель «точно в срок». Эта модель подразумевает своевременную поставку сырья и материалов, то есть в тот момент, когда они необходимы на производственной линии, и в том количестве, которое требуется на момент поставки. Для снижения затрат эта модель практически исключает складские запасы, но ее использование увеличивает зависимость организации от поставщиков. Любой сбой в поставках может привести к нехватке сырья и

вынужденному простою оборудования.

3. Модель ABC. Резервы делятся на три группы в зависимости от метода анализа объема и затрат. В группу А входят запасы, необходимые для самых дорогих продуктов с длительным циклом использования. На них приходится большая часть денежных средств, поэтому группа А строго контролируется, а необходимый объем и затраты точно определяются. Группа В включает сырье и материалы для продукции средней ценовой категории. Группа С — это самые дешевые запасы с высокой оборачиваемостью и самой низкой нормой прибыли.

Эффективное управление запасами влияет на общую операционную эффективность организации, снижая риски, связанные с излишками и дефицитом, и обеспечивая высокий уровень удовлетворенности клиентов.

1.3 Математические методы решения задачи управления запасами

Помимо метода динамического программирования, для решения задачи управления производственными запасами можно использовать всевозможные альтернативные методы. Некоторые из методов могут быть использованы в зависимости от деталей проблемы и имеющихся данных.

1. Линейное программирование — это набор математических и вычислительных инструментов для поиска конкретного решения системы, которое соответствует максимуму или минимуму другой линейной функции. Также это фундаментальный метод оптимизации, который уже несколько десятилетий используется в областях, требующих большого количества математических вычислений.

2. Имитационное моделирование — это процесс замены одного объекта другим с целью получения информации о наиболее важных характеристиках исходного объекта через объект модели. Процесс моделирования включает в себя такие этапы, как создание модели,

экспериментирование с моделью, обработка и интерпретация результатов моделирования. Имитация позволяет моделировать всевозможные операционные сценарии и их влияние на систему управления запасами. Это особенно полезно в условиях неопределенности и для тестирования различных стратегий управления запасами.

3. Метод ветвей и границ. “Его суть заключается в последовательном переборе вариантов, рассмотрении только наиболее перспективных по определенным характеристикам и отбрасывании бесперспективных вариантов. При использовании метода ветвей и границ область допустимых решений (ОДР) исходной задачи определенным образом разбивается на непересекающиеся подмножества, а подзадачи, то есть задачи на этих подмножествах, решаются с одинаковой ЦФ и без учета условия целочисленности (например, задачи ЛП). Если в результате получается неправильное оптимальное решение, то ОДР подзадачи разбивается снова, и этот процесс продолжается до тех пор, пока не будет найдено правильное оптимальное решение исходной задачи. Его можно использовать для поиска минимальной стоимости производства и хранения при определенных ограничениях на объем производства и хранения” [14].

4. Генетические алгоритмы предназначены для решения задач оптимизации и моделирования путем последовательного выбора, комбинирования и изменения нужных параметров с помощью механизмов, напоминающих биологическую эволюцию. Генетический алгоритм основан на методе случайного поиска. Основной недостаток случайного поиска заключается в том, что мы не знаем, сколько времени потребуется для решения задачи. Они могут помочь оптимизировать процессы управления запасами, находя решения, которые минимизируют затраты и при этом удовлетворяют всем требованиям производственного процесса.

“Это всевозможные методы, которые можно применить к задаче управления запасами, но все же более эффективным остается метод динамического программирования по ключевым моментам таким, как

идентификация и последовательность принятия решений, минимизация общих затрат, сложные функции затрат, решение дискретных задач, расчет ограничений на запасы и производство и возможность включения начальных и граничных условий” [14]. Исходя из этих особенностей, метод динамического программирования представляется наиболее подходящим для решения задачи оптимизации управления запасами, поскольку он обеспечивает точное и обоснованное решение с учетом всех производственных и логистических аспектов.

Сравним эффективность методов динамического программирования и линейного программирования при решении задач управления запасами.

Линейное программирование отлично подходит для решения задач, в которых связи между переменными могут быть выражены линейными уравнениями, однако линейное программирование не подходит для ситуаций с многоэтапными решениями и взаимосвязанными этапами, требующими чередования решений. Динамическое программирование лучше подходит для задач с несколькими этапами и сложностью, поскольку позволяет решать каждый этап отдельно, опираясь на результаты предыдущих этапов, и идеально подходит для задач с нелинейными затратами и решениями.

Сравним эффективность методов динамического программирования и имитационного моделирования при решении задач управления запасами.

Имитация позволяет моделировать различные сценарии и анализировать их возможные результаты без необходимости точно определять все аспекты системы математически, это полезно для анализа и тестирования, но не позволяет получить четкое или оптимальное решение. Динамическое программирование предоставляет специальный алгоритм для поиска оптимального решения на основе минимизации затрат или максимизации выгоды, что делает его предпочтительным для задач, требующих точного и экономически обоснованного решения.

Сравним эффективность метода динамического программирования и методы принятия решений в условиях неопределенности при решении задач

управления запасами.

Методы принятия решений в условиях неопределенности (такие как стохастическое программирование) нужны для работы с неопределенностью, но они идеальны, когда исходные данные (спрос и затраты) подвержены большим колебаниям. Динамическое программирование предполагает достаточно точную информацию о системе и ее параметрах для каждого шага, что делает его менее гибким в отношении параметров неопределенности, если только не вносятся изменения, включающие вероятностные элементы.

Сравним эффективность метода динамического программирования с методом ветвей и границ при решении задач управления запасами.

Метод ветвей и границ хорошо подходит для решения оптимизационных задач с дискретными переменными, где необходимо найти глобальное оптимальное решение, но этот метод может быть дорогим со стороны вычислительных ресурсов. ДП более эффективно по времени и ресурсам для большинства многошаговых задач, поскольку разбивает задачу на подзадачи и решает их последовательно, что зачастую более эффективно.

Сравним эффективность методов динамического программирования и генетических алгоритмов при решении задач управления запасами.

Генетические алгоритмы подходят для решения сложных оптимизационных задач, где традиционные методы неэффективны, они используют методы, основанные на эволюции и естественном отборе, и могут находить решения в сложных пространствах поиска. Динамическое программирование дает более стабильные и предсказуемые результаты в задачах с четко определенной пошаговой структурой и известными затратами, что делает их более контролируемыми и предсказуемыми.

Динамическое программирование остается самым мощным инструментом для решения задач управления запасами из-за своей способности точно моделировать и оптимизировать последовательные процессы и решения.

Глава 2 Решение производственной задачи управления запасами методом динамического программирования

2.1 Уравнение Беллмана

“В основе вычислительных алгоритмов динамического программирования лежит следующий принцип оптимальности, сформулированный Р. Беллманом: каково бы не было состояние системы S в результате k −1 шагов, управление на k -ом шаге должно выбираться так, чтобы оно в совокупности с управлениями на всех последующих шагах с (k +1) -го до N -го включительно доставляло экстремум целевой функции” [2].

“Первоначально уравнение Беллмана нашло свое применение в теории управления, технических системах и различных областях прикладной математики. В конечном итоге оно стало ключевым инструментом экономической теории. Многие задачи, решаемые с помощью теории оптимальных уравнений, находят свое решение в ее аналоге - уравнении Беллмана. Этим термином обозначается уравнение динамического программирования, связанное с проблемами оптимизации в дискретном времени. В сценариях непрерывной оптимизации уравнение, выраженное в частных производных, носит название уравнения Гамильтона-Якоби-Беллмана. Понимание уравнения Беллмана требует понимания фундаментальных концепций. Каждая проблема оптимизации вращается вокруг определенной цели, будь то минимизация времени в пути, сокращение затрат, максимизация прибыли, повышение полезности и так далее. Эта математическая функция, отражающая суть задачи, называется целевой функцией” [2].

“Динамическое программирование упрощает сложное многопериодное планирование, разбивая его на управляемые шаги в различных временных интервалах. Такой подход требует отслеживания эволюции обстоятельств принятия решений во времени” [5]. Важнейшие данные, необходимые для

принятия обоснованных решений в любой момент времени, называются "состоянием". Например, определяя уровень потребления и расходов с течением времени, люди должны учитывать такие факторы, как их первоначальное богатство. Следовательно, богатство становится основной переменной состояния, хотя, скорее всего, будут и другие переменные, которые необходимо учитывать.

Управляющие переменные, выбранные в определенный момент, часто называют детерминантами действий. Рассмотрим следующее: исходя из текущих обстоятельств, индивиды могут определить текущий уровень потребления. Выбор этих управляющих переменных может по сути сформировать последующее состояние. Однако на переход к следующему состоянию часто влияют факторы, выходящие за рамки непосредственного контроля. Например, в базовом сценарии сегодняшнее богатство (текущее состояние) и потребление (управляющая переменная) могут напрямую влиять на завтрашнее богатство (обновленное состояние), хотя обычно другие элементы также играют роль в формировании будущего богатства.

В сфере динамического программирования ее подход описывает оптимальный план, находя правило, которое сообщает, какими должны быть элементы управления, учитывая любое возможное значение состояния. Рассмотрим такую ситуацию, если потребление (c) зависит исключительно от богатства (W), то необходимо найти директиву, которая определяет потребление в зависимости от богатства. Эта директива, определяющая средства управления в зависимости от состояния, обозначается как функция политики.

По сути, принцип оптимального принятия решений — это путь, ведущий к наиболее благоприятному исходу в соответствии с поставленной целью.

“Например, если индивид выбирает потребление в зависимости от уровня благосостояния, чтобы оптимизировать счастье (при условии, что счастье может быть представлено математической функцией, такой как функция полезности, и определяется чем-то богатством), то каждому уровню

богатства будет соответствовать максимальный потенциальный уровень счастья. Это оптимальное значение цели, выраженное как функция состояния, называется функцией значения” [2].

Ричард Беллман продемонстрировал, что задача динамической оптимизации в дискретном времени может иметь рекурсивную, пошаговую структуру, называемую обратной индукцией. Это предполагает установление связи между функцией стоимости в текущем периоде и ее аналогом в последующем периоде. Эта связь между двумя функциями стоимости называется уравнением Беллмана. В рамках этой структуры оптимальная стратегия на последний период предопределяется как функция от значения переменной состояния на данный момент, таким образом выражая результирующее оптимальное значение функции цели на основе этого значения переменной состояния.

“Принцип оптимальности Беллмана можно записать в математической форме:

$$F_k(x_{k-1}, u_k) = \underset{u_k}{\operatorname{extr}}\big(z_k(x_{k-1}, u_k) + F_{k+1}(x_k)\big), \qquad (1)$$

где $F_k(x_{k-1}, u_k)$ − условно-оптимальное значение целевой функции на интервале от k -го до N -го шага включительно;

$z_k(x_{k-1}, u_k)$ − значение целевой функции на k -м шаге;

$F_{k+1}(x_k)$ условно-оптимальное значение целевой функции на интервале от (k+1) -го до N -го шага включительно;

x_{k-1} - множество состояний, в которых система S может находиться перед k -м шагом;

u_k - множество управлений, которые могут быть выбраны на k -м шаге для перевода системы S в одно из состояний множества x_k;

x_k - множество состояний, в которых система S может находиться в конце k -го шага.

Равенство (1) - основное функциональное уравнение динамического программирования.

При вычислении $F_N(x_{N-1}, u_N)$ на последнем шаге значение $F_{N+1}(x_N)$ можно положить равным нулю" [2].

"В соответствии с уравнением (1) и с учетом множеств состояний системы x_{k-1}, x_k и множеств управлений u_k, которые могут быть построены на k -м шаге, вычислительная процедура метода ДП распадается на два этапа: условную и безусловную оптимизацию" [2].

"На основании равенства (1) условная оптимизация осуществляется поэтапно при движении из конечного состояния x_n системы S в первоначальное состояние x_0 путем построения на каждом этапе условно-оптимального управления и нахождения условно-оптимального значения функции цели для каждого шага. Безусловная оптимизация осуществляется в обратном направлении: от первого шага к последнему, в результате чего находятся уже оптимальные управления на каждом шаге с точки зрения всего процесса. Первый этап сложнее и длительнее второго, на втором этапе отрабатываются рекомендации, полученные на первом этапе" [5].

Следует отметить, что понятия «конец» и «начало» можно поменять местами и разворачивать процесс оптимизации в другом направлении. С какого конца начать – диктуется удобством выбора этапов и возможных состояний на их начало.

2.2 Решение производственной задачи

Представлена система управления запасами графита с тремя этапами производства и дискретным спросом, который изменяется со временем и является детерминированным. На первом этапе спрос на графит равен 5 единицам, на втором - 2, а на третьем - 7. Начальный запас графита на складе отсутствует. Стоимость хранения одной единицы графита составляет 60 на первом этапе, 30 на втором и 50 на третьем. Затраты на производство графита

на каждом этапе определяются функцией $\varphi_j(x_j) = \mathrm{a}x_j{}^2 + \mathrm{b}x_j + \mathrm{c}, \mathrm{j} = 1,2,3$. Необходимо определить оптимальное количество единиц графита, которое следует производить на каждом из трех этапов, чтобы удовлетворить потребности потребителей, при этом минимизировать общие затраты на производство и хранение продукции на всех этапах.

Обозначения:

d_j – заявки потребителей на графит;

y_j – единицы графита на складе;

h_j – затраты на хранение единиц графита;

x_j – затраты на производство графита.

Таблица 1 – Исходные данные

d_1	d_2	d_3	a	b	c	h_1	h_2	h_3	y_1
5	2	7	8	1	4	60	30	50	0

Решение:

применяя рекуррентное соотношение, учитывая исходные данные из таблицы 1, последовательно вычисляем:

$F_1 = (\zeta = y_2)$, $F_2 = (\zeta = y_3)$, $F_3 = (\zeta = y_4)$, и находим: $x_1^*(\zeta = y_2)$, $x_2^*(\zeta = y_3)$, $x_3^*(\zeta = y_4)$.

1 этап. Пусть k=1. Тогда по формуле $F_1(\zeta = y_2) = \min\{\mathrm{a}x_j{}^2 + \mathrm{b}x_j + \mathrm{c} + h_1 y_2\}$ в итоге получаем: $F_1(\zeta = y_2) = \min\{8x_j{}^2 + \mathrm{b}x_j + 4 + 60y_2\}$.

Параметр состояния y_2 принимает целые значения на промежутке от 0 до $d_2 + d_3 = 2 + 7 = 9$, т.е. $y_2 = 0,1,2,\ldots,9$.

Каждому значению параметра состояния должна отвечать определённая область изменения переменной x_1, характеризуемая условием $0 \le x_1 \le d_1 + y_2$.

На начальном этапе получаем $0 \le x_1 \le 5 + y_2$.

Однако на первом этапе объем производства x1 не может быть меньше пяти, так как спрос $d_1 = 5$, а исходный запас $y_1 = 0$.

Из балансового уравнения $x_1 + y_1 - d_1 = y_2$ следует, что объем производства связан со значением параметра состояния $\zeta = y_2$ соотношением $x_1 = y_2 + d_1 - y_1 = y_2 + 5 - 0 = y_2 + 5$.

При условии, что задан уровень запаса к началу первого этапа, получается, что каждому значению y_2 отвечает единственное значение x_1, и поэтому $F_1(\zeta = y_2) = \Omega_1(x_1, x_2)$.

Придавая y_2 значения в диапазоне от 0 до 9 и учитывая, что $x_1 = y_2 + 5$, находим:

$y_2 = 0, x_1 = 5, \Omega_1(5,0) = 8 \cdot 5^2 + 5 + 4 + 60 \cdot 0 = 209;$

$y_2 = 1, x_1 = 6, \Omega_1(6,1) = 8 \cdot 6^2 + 6 + 4 + 60 \cdot 1 = 358;$

$y_2 = 2, x_1 = 7, \Omega_1(7,2) = 8 \cdot 7^2 + 7 + 4 + 60 \cdot 2 = 523;$

$y_2 = 3, x_1 = 8, \Omega_1(8,3) = 8 \cdot 8^2 + 8 + 4 + 60 \cdot 3 = 704;$

$y_2 = 4, x_1 = 9, \Omega_1(9,4) = 8 \cdot 9^2 + 9 + 4 + 60 \cdot 4 = 901;$

$y_2 = 5, x_1 = 10, \Omega_1(10,5) = 8 \cdot 10^2 + 10 + 4 + 60 \cdot 5 = 1114;$

$y_2 = 6, x_1 = 11, \Omega_1(11,6) = 8 \cdot 11^2 + 11 + 4 + 60 \cdot 6 = 1343;$

$y_2 = 7, x_1 = 12, \Omega_1(12,7) = 8 \cdot 12^2 + 12 + 4 + 60 \cdot 7 = 1588;$

$y_2 = 8, x_1 = 13, \Omega_1(13,8) = 8 \cdot 13^2 + 13 + 4 + 60 \cdot 8 = 1849;$

$y_2 = 9, x_1 = 14, \Omega_1(14,9) = 8 \cdot 14^2 + 14 + 4 + 60 \cdot 9 = 2126.$

Таблица 2 – Результаты вычисления функции состояния $F_1(\zeta)$

$\zeta = y_2$	0	1	2	3	4	5	6	7	8	9
$F_1(\zeta = y_2)$	209	358	523	704	901	1114	1343	1588	1849	2126
$x_1^*(\zeta = y_2)$	5	6	7	8	9	10	11	12	13	14

2 этап. Пусть k = 2.

Табулируем функцию $F_2(\zeta = y_3)$: $F_2(\zeta = y_3) = \min \Omega_2(x_2, y_3) = \min \{\mathrm{a}{x_2}^2 + \mathrm{b}x_2 + \mathrm{c} + h_2 y_3 + \mathrm{F}_1(\mathrm{y}_2)\} = \min \{8{x_2}^2 + x_2 + 4 + 30y_3 + F_1(y_2)\}$.

Переменная x_2 принимает значения в пределах $0 \le x_2 \le d_2 + y_3 = y_3 + 2$, а y3 принимает значения в диапазоне $0 \le y_3 \le d_3 = 7$, т.е. $y_3 = 0,1,2,\dots,7$.

Из балансового уравнения $x_2 + y_2 - d_2 = y_3$ следует, что $y_2 = y_3 + d_2 - x_2 = y_3 + 2 - x_2$.

Приписывая для y_3 разные целые значения от 0 до 7, будем поочередно вычислять $\Omega_2(x_2, \zeta)$, а затем определять $F_2(\zeta)$ и $x_2^*(\zeta)$.

Весь процесс табулирования приведен в таблице 3.

Таблица 3 – Процесс табулирования

$\zeta = y_3$	$0 \le x_2 \le y_3 + 2$	x_2	$y_2 = y_3 + 2 - x_2$	$\Omega_2(x_2, y_3) = 8\text{x}2^2 + \text{x}2 + 4 + 30\text{y}3 + F_1(\text{y}2)$
$y_3 = 0$	$0 \le x_2 \le 2$	0	$y_2 = 0 + 2 - 0 = 2$	$\Omega_2(0,0) = 8 \cdot 0^2 + 0 + 4 + 30 \cdot 0 + F_1(2) = 4 + 523 = 527$
—	—	1	$y_2 = 0 + 2 - 1 = 1$	$\Omega_2(1,0) = 8 \cdot 1^2 + 1 + 4 + 30 \cdot 0 + F_1(1) = 13 + 358 = 371$
—	—	2	$y_2 = 0 + 2 - 2 = 0$	$\Omega_2(2,0) = 8 \cdot 2^2 + 2 + 4 + 30 \cdot 0 + F_1(0) = 38 + 209 = 247$
$y_3 = 1$	$0 \le x_2 \le 3$	0	$y_2 = 1 + 2 - 0 = 3$	$\Omega_2(0,1) = 8 \cdot 0^2 + 0 + 4 + 30 \cdot 1 + F_1(3) = 34 + 704 = 738$
—	—	1	$y_2 = 1 + 2 - 1 = 2$	$\Omega_2(1,1) = 8 \cdot 1^2 + 1 + 4 + 30 \cdot 1 + F_1(2) = 43 + 523 = 556$
—	—	2	$y_2 = 1 + 2 - 2 = 1$	$\Omega_2(2,1) = 8 \cdot 2^2 + 2 + 4 + 30 \cdot 1 + F_1(1) = 68 + 358 = 426$
—	—	3	$y_2 = 1 + 2 - 3 = 0$	$\Omega_2(3,1) = 8 \cdot 3^2 + 3 + 4 + 30 \cdot 1 + F_1(0) = 109 + 209 = 318$
$y_3 = 2$	$0 \le x_2 \le 4$	0	$y_2 = 2 + 2 - 0 = 4$	$\Omega_2(0,2) = 8 \cdot 0^2 + 0 + 4 + 30 \cdot 2 + F_1(4) = 64 + 901 = 965$

Продолжение таблицы 3

$\zeta = y_3$	$0 \le x_2 \le y_3 + 2$	x_2	$y_2 = y_3 + 2 - x_2$	$\Omega_2(x_2, y_3) = 8\text{x}2^2 + \text{x}2 + 4 + 30\text{y}3 + F_1(\text{y}2)$
—	—	1	$y_2 = 2 + 2 - 1 = 3$	$\Omega_2(1,2) = 8 \cdot 1^2 + 1 + 4 + 30 \cdot 2 + F_1(3) = 72 + 704 = 776$
—	—	2	$y_2 = 2 + 2 - 2 = 2$	$\Omega_2(2,2) = 8 \cdot 2^2 + 2 + 4 + 30 \cdot 2 + F_1(2) = 98 + 523 = 621$
—	—	3	$y_2 = 2 + 2 - 3 = 1$	$\Omega_2(3,2) = 8 \cdot 3^2 + 3 + 4 + 30 \cdot 2 + F_1(1) = 139 + 358 = 497$
—	—	4	$y_2 = 2 + 2 - 4 = 0$	$\Omega_2(4,2) = 8 \cdot 4^2 + 4 + 4 + 30 \cdot 2 + F_1(0) = 196 + 209 = 405$
$y_3 = 3$	$0 \le x_2 \le 5$	0	$y_2 = 3 + 2 - 0 = 5$	$\Omega_2(0,3) = 8 \cdot 0^2 + 0 + 4 + 30 \cdot 3 + F_1(5) = 94 + 1114 = 1208$
—	—	1	$y_2 = 3 + 2 - 1 = 4$	$\Omega_2(1,3) = 8 \cdot 1^2 + 1 + 4 + 30 \cdot 3 + F_1(4) = 103 + 901 = 1004$
—	—	2	$y_2 = 3 + 2 - 2 = 3$	$\Omega_2(2,3) = 8 \cdot 2^2 + 2 + 4 + 30 \cdot 3 + F_1(3) = 128 + 704 = 832$
—	—	3	$y_2 = 3 + 2 - 3 = 2$	$\Omega_2(3,3) = 8 \cdot 3^2 + 3 + 4 + 30 \cdot 3 + F_1(2) = 169 + 523 = 692$
—	—	4	$y_2 = 3 + 2 - 4 = 1$	$\Omega_2(4,3) = 8 \cdot 4^2 + 4 + 4 + 30 \cdot 3 + F_1(1) = 226 + 358 = 584$
—	—	5	$y_2 = 3 + 2 - 5 = 0$	$\Omega_2(5,3) = 8 \cdot 5^2 + 5 + 4 + 30 \cdot 3 + F_1(0) = 299 + 209 = 508$
$y_3 = 4$	$0 \le x_2 \le 6$	0	$y_2 = 4 + 2 - 0 = 6$	$\Omega_2(0,4) = 8 \cdot 0^2 + 0 + 4 + 30 \cdot 4 + F_1(6) = 124 + 1343 = 1467$
—	—	1	$y_2 = 4 + 2 - 1 = 5$	$\Omega_2(1,4) = 8 \cdot 1^2 + 1 + 4 + 30 \cdot 4 + F_1(5) = 133 + 1114 = 1247$

Продолжение таблицы 3

$\zeta = y_3$	$0 \le x_2 \le y_3 + 2$	x_2	$y_2 = y_3 + 2 - x_2$	$\Omega_2(x_2, y_3) = 8\text{x}2^2 + \text{x}2 + 4 + 30\text{y}3 + F_1(\text{y}2)$
—	—	2	$y_2 = 4 + 2 - 2 = 4$	$\Omega_2(2,4) = 8 \cdot 2^2 + 2 + 4 + 30 \cdot 4 + F_1(4) = 158 + 901 = 1059$
—	—	3	$y_2 = 4 + 2 - 3 = 3$	$\Omega_2(3,4) = 8 \cdot 3^2 + 3 + 4 + 30 \cdot 4 + F_1(3) = 199 + 704 = 903$
—	—	4	$y_2 = 4 + 2 - 4 = 2$	$\Omega_2(4,4) = 8 \cdot 4^2 + 4 + 4 + 30 \cdot 4 + F_1(2) = 256 + 523 = 779$
—	—	5	$y_2 = 4 + 2 - 5 = 1$	$\Omega_2(5,4) = 8 \cdot 5^2 + 5 + 4 + 30 \cdot 4 + F_1(1) = 329 + 358 = 687$
—	—	6	$y_2 = 4 + 2 - 6 = 0$	$\Omega_2(6,4) = 8 \cdot 6^2 + 6 + 4 + 30 \cdot 4 + F_1(0) = 418 + 209 = 627$
$y_3 = 5$	$0 \le x_2 \le 7$	0	$y_2 = 5 + 2 - 0 = 7$	$\Omega_2(0,5) = 8 \cdot 0^2 + 0 + 4 + 30 \cdot 5 + F_1(7) = 154 + 1588 = 1742$
—	—	1	$y_2 = 5 + 2 - 1 = 6$	$\Omega_2(1,5) = 8 \cdot 1^2 + 1 + 4 + 30 \cdot 5 + F_1(6) = 163 + 1343 = 1506$
—	—	2	$y_2 = 5 + 2 - 2 = 5$	$\Omega_2(2,5) = 8 \cdot 2^2 + 2 + 4 + 30 \cdot 5 + F_1(5) = 188 + 1114 = 1302$
—	—	3	$y_2 = 5 + 2 - 3 = 4$	$\Omega_2(3,5) = 8 \cdot 3^2 + 3 + 4 + 30 \cdot 5 + F_1(4) = 229 + 901 = 1130$
—	—	4	$y_2 = 5 + 2 - 4 = 3$	$\Omega_2(4,5) = 8 \cdot 4^2 + 4 + 4 + 30 \cdot 5 + F_1(3) = 286 + 704 = 990$
—	—	5	$y_2 = 5 + 2 - 5 = 2$	$\Omega_2(5,5) = 8 \cdot 5^2 + 5 + 4 + 30 \cdot 5 + F_1(2) = 359 + 523 = 882$
—	—	6	$y_2 = 5 + 2 - 6 = 1$	$\Omega_2(6,5) = 8 \cdot 6^2 + 6 + 4 + 30 \cdot 5 + F_1(1) = 448 + 358 = 806$

Продолжение таблицы 3

$\zeta = y_3$	$0 \le x_2 \le y_3 + 2$	x_2	$y_2 = y_3 + 2 - x_2$	$\Omega_2(x_2, y_3) = 8\text{x}2^2 + \text{x}2 + 4 + 30\text{y}3 + F_1(\text{y}2)$
—	—	7	$y_2 = 5 + 2 - 7 = 0$	$\Omega_2(7,5) = 7 \cdot 3^2 + 7 + 4 + 30 \cdot 5 + F_1(0) = 553 + 209 = 762$
$y_3 = 6$	$0 \le x_2 \le 8$	0	$y_2 = 6 + 2 - 0 = 8$	$\Omega_2(0,6) = 8 \cdot 0^2 + 0 + 4 + 30 \cdot 6 + F_1(8) = 184 + 1849 = 2033$
—	—	1	$y_2 = 6 + 2 - 1 = 7$	$\Omega_2(1,6) = 8 \cdot 1^2 + 1 + 4 + 30 \cdot 6 + F_1(7) = 193 + 1588 = 1781$
—	—	2	$y_2 = 6 + 2 - 2 = 6$	$\Omega_2(2,6) = 8 \cdot 2^2 + 2 + 4 + 30 \cdot 6 + F_1(6) = 218 + 1343 = 1561$
—	—	3	$y_2 = 6 + 2 - 3 = 5$	$\Omega_2(3,6) = 8 \cdot 3^2 + 3 + 4 + 30 \cdot 6 + F_1(5) = 259 + 1114 = 1373$
—	—	4	$y_2 = 6 + 2 - 4 = 4$	$\Omega_2(4,6) = 8 \cdot 4^2 + 4 + 4 + 30 \cdot 6 + F_1(4) = 316 + 901 = 1217$
—	—	5	$y_2 = 6 + 2 - 5 = 3$	$\Omega_2(5,6) = 8 \cdot 5^2 + 5 + 4 + 30 \cdot 6 + F_1(3) = 389 + 704 = 1093$
—	—	6	$y_2 = 6 + 2 - 6 = 2$	$\Omega_2(6,6) = 8 \cdot 6^2 + 6 + 4 + 30 \cdot 6 + F_1(2) = 478 + 523 = 1001$
—	—	7	$y_2 = 6 + 2 - 7 = 1$	$\Omega_2(7,6) = 8 \cdot 7^2 + 7 + 4 + 30 \cdot 6 + F_1(1) = 583 + 358 = 941$
—	—	8	$y_2 = 6 + 2 - 8 = 0$	$\Omega_2(8,6) = 8 \cdot 8^2 + 8 + 4 + 30 \cdot 6 + F_1(0) = 704 + 209 = 913$
$y_3 = 7$	$0 \le x_2 \le 9$	0	$y_2 = 7 + 2 - 0 = 9$	$\Omega_2(0,7) = 8 \cdot 0^2 + 0 + 4 + 30 \cdot 7 + F_1(9) = 214 + 2126 = 2340$
—	—	1	$y_2 = 7 + 2 - 1 = 8$	$\Omega_2(1,7) = 8 \cdot 1^2 + 1 + 4 + 30 \cdot 7 + F_1(8) = 223 + 1849 = 2072$

Продолжение таблицы 3

$\zeta = y_3$	$0 \le x_2 \le y_3 + 2$	x_2	$y_2 = y_3 + 2 - x_2$	$\Omega_2(x_2, y_3) = 8\text{x}2^2 + \text{x}2 + 4 + 30\text{y}3 + F_1(\text{y}2)$
—	—	2	$y_2 = 7 + 2 - 2 = 7$	$\Omega_2(2,7) = 8 \cdot 2^2 + 2 + 4 + 30 \cdot 7 + F_1(7) = 248 + 1588 = 1836$
—	—	3	$y_2 = 7 + 2 - 3 = 6$	$\Omega_2(3,7) = 8 \cdot 3^2 + 3 + 4 + 30 \cdot 7 + F_1(6) = 289 + 1343 = 1632$
—	—	4	$y_2 = 7 + 2 - 4 = 5$	$\Omega_2(4,7) = 8 \cdot 4^2 + 4 + 4 + 30 \cdot 7 + F_1(5) = 346 + 1114 = 1460$
—	—	5	$y_2 = 7 + 2 - 5 = 4$	$\Omega_2(5,7) = 8 \cdot 5^2 + 5 + 4 + 30 \cdot 7 + F_1(4) = 419 + 901 = 1320$
—	—	6	$y_2 = 7 + 2 - 6 = 3$	$\Omega_2(6,7) = 8 \cdot 6^2 + 6 + 4 + 30 \cdot 7 + F_1(3) = 508 + 704 = 1212$
—	—	7	$y_2 = 7 + 2 - 7 = 2$	$\Omega_2(7,7) = 8 \cdot 7^2 + 7 + 4 + 30 \cdot 7 + F_1(2) = 613 + 523 = 1136$
—	—	8	$y_2 = 7 + 2 - 8 = 1$	$\Omega_2(8,7) = 8 \cdot 8^2 + 8 + 4 + 30 \cdot 7 + F_1(1) = 734 + 358 = 1092$
—	—	9	$y_2 = 7 + 2 - 9 = 0$	$\Omega_2(9,7) = 8 \cdot 9^2 + 9 + 4 + 30 \cdot 7 + F_1(0) = 871 + 209 = 1080$

Наименьшие подходящие из полученных значений Ω_2 представлены в таблице 3. Итого получаем: $F_2(0) = \Omega_2(2,0) = 247$, $F_2(1) = \Omega_2(3,1) = 318, F_2(2) = \Omega_2(4,2) = 405, F_2(3) = \Omega_2(5,3) = 508, F_2(4) = \Omega_2(6,4) = 627, F_2(5) = \Omega_2(7,5) = 762, F_2(6) = \Omega_2(8,6) = 913, F_2(7) = \Omega_2(9,7) = 1080.$

3 этап.

k = 3. Табулируем функцию $F_3(\zeta = y_4)$: $F_3(\zeta = y_4) = \min \Omega_3(x_3, y_4) = \min \{\text{a}{x_3}^2 + \text{b}x_3 + \text{c} + h_3 y_4 + \text{F}_2(\text{y}_3)\} = \min \{8{x_3}^2 + x_3 + 4 + 50y_4 + F_2(y_3)\}$. Вычисляем значение функции состояния лишь для одного значения аргумента

$\zeta = y_4 = 0$, так как не хотим оставлять продукцию на хранение в конце высчитываемого периода. Весь процесс вычисления приведен в таблице 4.

Таблица 4 – Процесс вычисления функции состояния

$\zeta = y_4$	$0 \le x_3 \le d_3 + y_4$	x_3	$y_3 = y_4 + d_3 - x_3$	$\Omega_3(x_3, y_4) = 8{x_3}^2 + x_3 + 4 + 50y_4 + F_2(y_3)$
$y_4 = 0$	$0 \le x_3 \le 7$	0	$y_3 = 0 + 7 - 0 = 7$	$\Omega_3(0,0) = 8 \cdot 0^2 + 0 + 4 + 50 \cdot 0 + F_2(7) = 4 + 1080 = 1084$
—	—	1	$y_3 = 0 + 7 - 1 = 6$	$\Omega_3(1,0) = 8 \cdot 1^2 + 1 + 4 + 50 \cdot 0 + F_2(6) = 13 + 913 = 926$
—	—	2	$y_3 = 0 + 7 - 2 = 5$	$\Omega_3(2,0) = 8 \cdot 2^2 + 2 + 4 + 50 \cdot 0 + F_2(5) = 38 + 762 = 800$
—	—	3	$y_3 = 0 + 7 - 3 = 4$	$\Omega_3(3,0) = 8 \cdot 3^2 + 3 + 4 + 50 \cdot 0 + F_2(4) = 79 + 627 = 706$
—	—	4	$y_3 = 0 + 7 - 4 = 3$	$\Omega_3(4,0) = 8 \cdot 4^2 + 4 + 4 + 50 \cdot 0 + F_2(3) = 186 + 508 = 694$
—	—	5	$y_3 = 0 + 7 - 5 = 2$	$\Omega_3(5,0) = 8 \cdot 5^2 + 5 + 4 + 50 \cdot 0 + F_2(2) = 279 + 405 = 684$
—	—	6	$y_3 = 0 + 7 - 6 = 1$	$\Omega_3(6,0) = 8 \cdot 6^2 + 6 + 4 + 50 \cdot 0 + F_2(1) = 298 + 318 = 666$
—	—	7	$y_3 = 0 + 7 - 7 = 0$	$\Omega_3(7,0) = 8 \cdot 7^2 + 7 + 4 + 50 \cdot 0 + F_2(0) = 403 + 247 = 650$

Наименьшее правильное из полученных значений из таблицы 4 $F_3(\zeta = y_4) = min\Omega_3(x_3, y_4) = 650$, заметим, что минимум достигается при значении x_3, равном $x_3^*(\zeta = y_4 = 0)$. Получили, что на крайнем этапе необходимо выпускать 7 единиц продукции $x_3^* = 7$. Вычисляем предпоследнюю компоненту оптимального плана:

$y_3 = y_4 + d_3 - x_3 = 0 + 7 - 7 = 0$. В первой части таблицы 4 предыдущего этапа находим выделенную строку для $y_3 = 0$. Имеем: $x_2^*(\zeta = y_3 = 0) = 2$.

Находим x_1^*, учитывая таблицу 2 и $y_2 = y_3 + d_2 - x_2 = 2 + 2 - 4 = 0$. Имеем: $x_1^* = x_1^*(\zeta = y_2 = 0) = 5$. Из всех вычислений следует, что оптимальный план производства имеет вид: $x_1^* = 5,\ x_2^* = 2,\ x_3^* = 7$. Минимальные общие затраты составляют 614 единиц.

Таблица 5 – Результат вычислений

Этап	1	2	3	Итого за 3 этапа
Имеем продукцию к началу этапа, шт.	$y_1 = 0$	$y_2 = 0$	$y_3 = 0$	—
Производим в течение этапа, шт.	$x_1 = 5$	$x_2 = 2$	$x_3 = 7$	$x_1 + x_2 + x_3 = 14$
Отдаем заказчикам, шт.	$d_1 = 5$	$d_2 = 2$	$d_3 = 7$	$d_1 + d_2 + d_3 = 14$
Остаток к концу этапа (Храним в течение текущего этапа), шт.	$y_2 = 0$	$y_3 = 0$	$y_4 = 0$	—
Затраты на производство, руб.	$\varphi(x_1) = 209$	$\varphi(x_2) = 196$	$\varphi(x_3) = 245$	$\varphi(x1) + \varphi(x2) + \varphi(x3) = 650$
Затраты на хранение, руб.	$h_1 y_2 = 0$	$h_2 y_3 = 0$	0	$h_1 y_2 + h_2 y_3 = 0$

Видим, что потребность заказчиков из таблицы 5 на каждом из этапов выполняются: $y_1 + x_1 \geq d_1$, $y_2 + x_2 \geq d_2$, $y_3 + x_3 \geq d_3$, $0 + 5 \geq 5$, $0 + 2 \geq 2$, $0 + 7 \geq 7$. Полный объем производства за все периоды и то сколько продукции мы имели к началу первого этапа равно суммарной потребности: $x_1 + x_2 + x_3 = 14$ и $d_1 + d_2 + d_3 = 14$.

При данном плане производства не требуется долгосрочное хранение продукции, так как оптимальные общие затраты достигаются за счет немедленного потребления всей произведенной продукции.

Глава 3 Программная реализация алгоритма решения задачи управления запасами

3.1 Выбор языка и его среды программирования

Для программной реализации задачи управления запасами с применением метода динамического программирования, мной был выбран широко известный всем язык Python. Если говорить про его историю, то язык программирования Python был создан в 1989–1991 годах голландским программистом Гвидо ван Россумом. Изначально это был любительский проект: разработчик начал работу над ним, просто чтобы занять себя на рождественских каникулах. Хотя сама идея создания нового языка появилась у него двумя годами ранее. Язык программирования он и выбрал — Python, что это означало название комик-группы. История развития этого языка происходила в несколько этапов, каждый из которых означал выход новой версии языка.

1. “Первый этап развития был положен в 1991 году, где Гвидо опубликовал первую версию (0.9.0) языка, которая включала в себя базовые возможности, корректировку ошибок и работу с данными различных типов” [7].

2. “Далее, через 3 года, была выпущена версия (1.0). Она включала в себя дополнения в виде обработки списков данных, таких как систематизация, сокращение, фильтрация и сопоставление” [7].

3. “В 2000 году была опубликована версия (2.0). В ней были исправлены недочеты прежних версий и были добавлены полезные функции для программистов” [7].

4. “В 2008 году представлена версия Python (3.0), включившая возможность печати, расширенное исправление ошибок и поддержку деления чисел” [7].

“Язык программирования Python, начавшийся как проект одного человека, по сей день развивается и поддерживается командой разработчиков”

[2].

“Данный язык программирования особенно хорошо подходит для реализации задач, таких как оптимизация производственных процессов, по таким ключевым причинам, как: богатая экосистема библиотек, простота и читаемость, интеграция и масштабируемость, гибкость и поддержка сообщества” [7].

Для работы с языком Python, была выбрана бесплатная среда VSCode. Он является одним из наиболее популярных редакторов кода, который предоставляет широкий набор функций и возможностей для работы с Python. VSCode имеет низкий порог входа и быстро запускается, также есть поддержка IntelliSense, что обеспечивает автозаполнение, подсказки и быстрый доступ к документации. Большое количество расширений позволяет настроить среду разработки под свои потребности, включая поддержку различных фреймворков и инструментов для Python. Встроенная поддержка Git упрощает работу с контролем версий, а также есть возможность интеграции с другими инструментами и сервисами. Ну и поскольку VSCode является популярным инструментом, у него большое сообщество пользователей и активная поддержка разработчиков.

3.2 Алгоритм программы

Код состоит из нескольких классов, каждый из которых выполняет определенную функцию в рамках процесса производства и управления затратами. Программа рассчитана на взаимодействие с пользователем через консоль. Пользователь задает количество этапов, а затем для каждого этапа указывает спрос и стоимость хранения. Далее программа оптимизирует производственный процесс с учетом данных параметров и выводит результаты.

В программе было использовано несколько основных принципов объектно-ориентированного программирования (ООП), которые

способствуют его масштабируемости и поддерживаемости.

```
import tkinter as tk
from tkinter import ttk, messagebox
```

Рисунок 1 — Импорт модулей

На рисунке 1 импортируются необходимые модули для создания графического интерфейса. ‘tkinter’ используется для создания основного окна и виджетов, ‘ttk’ для улучшенных виджетов, а ‘messagebox’ для отображения сообщений об ошибках.

```
# Базовый класс для управления производственными этапами
class ProductionStage:
    def __init__(self, demand, storage_cost):
        self.demand = demand  # спрос на этом этапе
        self.storage_cost = storage_cost  # стоимость хранения на этапе

    # Метод для расчета стоимости хранения
    def calculate_storage_cost(self, y_prev):
        if y_prev > 0:
            return self.storage_cost * y_prev
        return 0
```

Рисунок 2 — Определение класса ‘ProductionStage’

На рисунке 2 блок задает основу для представления производственных этапов, устанавливая структуру для хранения данных о спросе и стоимости хранения, а также базовую логику для их обработки и представляет собой отдельный этап производства. Этот класс инкапсулирует свойства и методы, связанные со спросом и стоимостью хранения на этапе производства, а также ‘ProductionStage’ абстрагирует концепцию производственного этапа, упрощая обращение с ним в коде.

Метод ‘calculate_storage_cost’ рассчитывает стоимость хранения на

этапе в зависимости от количества единиц, оставшихся с предыдущего этапа.

```
# Производный класс, модифицирующий расчет стоимости хранения
class SpecialProductionStage(ProductionStage):
    # Переопределенный метод для расчета стоимости хранения с модификацией
    def calculate_storage_cost(self, y_prev):
        base_cost = super().calculate_storage_cost(y_prev)  # вызов метода базового класса
        return base_cost * 2  # удваиваем стоимость хранения
```

Рисунок 3 — Определение класса ‘SpecialProductionStage’

На рисунке 3 создается производный класс, который изменяет поведение базового класса ‘ProductionStage’, а именно метод расчета стоимости хранения, делая его удвоенным. Переопределение метода ‘calculate_storage_cost’ демонстрирует полиморфизм, позволяя объектам ‘SpecialProductionStage’ иметь другую реализацию этого метода, чем в базовом классе. Также ‘SpecialProductionStage’ наследует от ‘ProductionStage’, расширяя его функциональность.

Переопределенный метод ‘calculate_storage_cost’ удваивает стоимость хранения по сравнению с базовым классом, используя вызов родительского метода через super().

```
# Класс для расчета стоимости производства
class ProductionCost:
    def __init__(self, a, b, c):
        self.a = a  # параметр a в формуле затрат
        self.b = b  # параметр b в формуле затрат
        self.c = c  # параметр c в формуле затрат

    # Метод для расчета стоимости производства
    def calculate(self, x):
        return self.a * x**2 + self.b * x + self.c
```

Рисунок 4 — Определение класса ‘ProductionCost’

На рисунке 4 блок задает класс для расчета стоимости производства,

используя заданную математическую формулу, что позволяет в последующем применять этот расчет при оптимизации производственных процессов. Класс инкапсулирует данные, необходимые для расчета стоимости производства, скрывая сложные вычисления за простым интерфейсом.

Метод ‘calculate’ использует квадратичную формулу для определения стоимости производства на основе заданного объема.

```
# Оптимизатор производства, управляющий расчетом оптимального количества и стоимости
class ProductionOptimizer:
    def __init__(self, stages, production_cost):
        self.stages = stages  # список этапов производства
        self.production_cost = production_cost  # объект для расчета затрат на производство
        self.optimal_production = [0] * len(stages)  # инициализация списка оптимального производства
        self.total_cost = None  # общая стоимость производства

    # Расчет оптимального производства
    def calculate_optimal_production(self):
        F = [{} for _ in range(len(self.stages))]  # таблица для хранения минимальных стоимостей

        # Перебор всех этапов
        for stage_index, stage in enumerate(self.stages):
            # Перебор возможных остатков на текущем этапе
            for y_next in range(sum(stage.demand for stage in self.stages[stage_index:]) + 1):
                best_cost = float('inf')
                best_x = 0
```

Рисунок 5 — Определение класса ‘ProductionOptimizer’

```
# Перебор возможных объемов производства
for x in range(stage.demand, sum(stage.demand for stage in self.stages[:stage_index + 1]) + 1):
    y_prev = x - stage.demand + (0 if stage_index == 0 else sum(stage.demand for stage in self.stages[:stage_index]) - sum(self.optimal_production[:stage_index]))
    cost = self.production_cost.calculate(x)  # расчет затрат на производство
    cost += stage.calculate_storage_cost(y_prev)  # добавление стоимости хранения
    if stage_index > 0:
        cost += min(F[stage_index - 1].values())  # добавление минимальной предыдущей стоимости
```

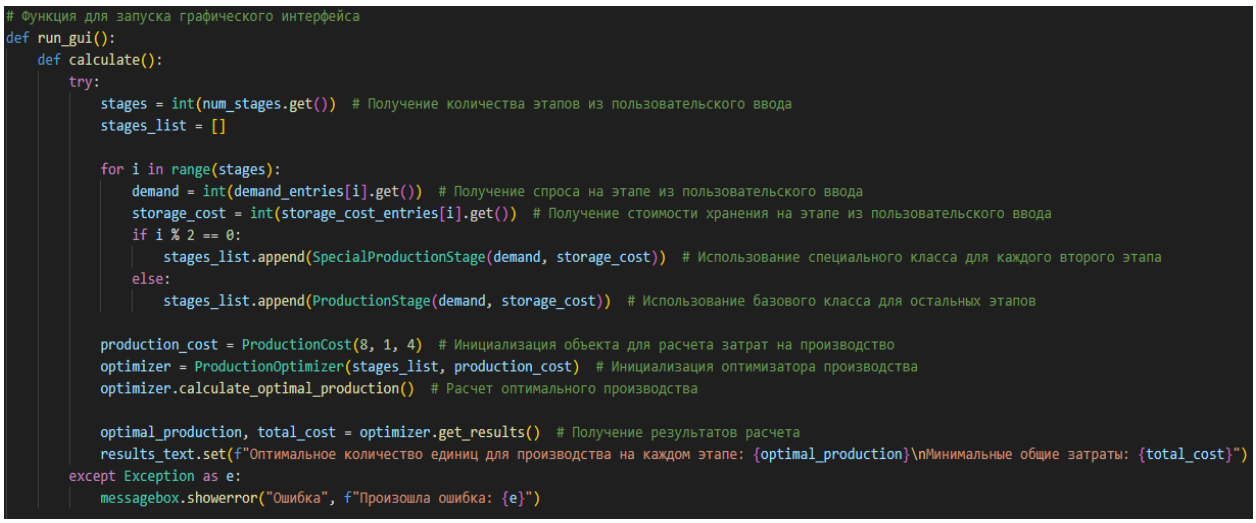
Рисунок 6 — Определение класса ‘ProductionOptimizer’

Рисунок 7 — Определение класса ‘ProductionOptimizer’

Здесь реализуется логика для оптимизации производства, включая расчеты оптимальных объемов производства на каждом этапе и итоговой стоимости. ‘ProductionOptimizer’ абстрагирует сложный процесс оптимизации в набор простых вызовов методов, таких как ‘calculate_optimal_production’ и ‘get_results’. Класс управляет всем процессом оптимизации производства, инкапсулируя данные, методы для расчета оптимальных объемов и затрат.

На рисунках 5-6 метод ‘calculate_optimal_production’ проводит расчеты для определения оптимальных объемов производства на каждом этапе, минимизируя общие затраты.

На рисунке 7 метод ‘get_results' возвращает оптимальные объемы производства и минимальную общую стоимость.

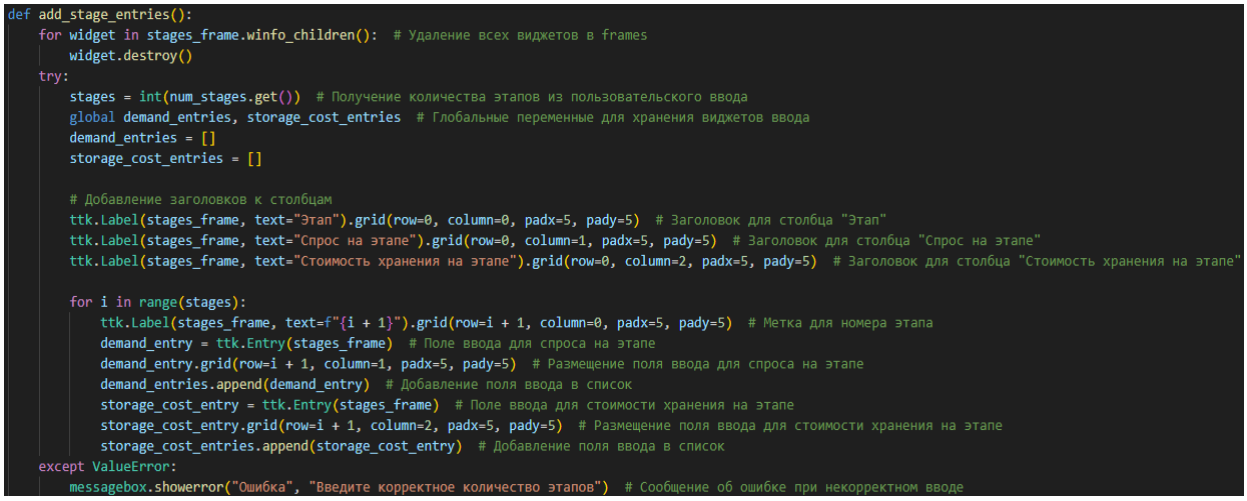


```
# Функция для запуска графического интерфейса
def run_gui():
    def calculate():
        try:
            stages = int(num_stages.get())  # Получение количества этапов из пользовательского ввода
            stages_list = []

            for i in range(stages):
                demand = int(demand_entries[i].get())  # Получение спроса на этапе из пользовательского ввода
                storage_cost = int(storage_cost_entries[i].get())  # Получение стоимости хранения на этапе из пользовательского ввода
                if i % 2 == 0:
                    stages_list.append(SpecialProductionStage(demand, storage_cost))  # Использование специального класса для каждого второго этапа
                else:
                    stages_list.append(ProductionStage(demand, storage_cost))  # Использование базового класса для остальных этапов

            production_cost = ProductionCost(8, 1, 4)  # Инициализация объекта для расчета затрат на производство
            optimizer = ProductionOptimizer(stages_list, production_cost)  # Инициализация оптимизатора производства
            optimizer.calculate_optimal_production()  # Расчет оптимального производства

            optimal_production, total_cost = optimizer.get_results()  # Получение результатов расчета
            results_text.set(f"Оптимальное количество единиц для производства на каждом этапе: {optimal_production}\nМинимальные общие затраты: {total_cost}")
        except Exception as e:
            messagebox.showerror("Ошибка", f"Произошла ошибка: {e}")
```

Рисунок 8 — Определение функции ‘run_gui’ и ‘calculate’

run_gui - главная функция для создания и запуска графического интерфейса.

На рисунке 8 функция ‘calculate’ - выполняет расчет оптимального производства на основе введенных данных. Получает количество этапов и их параметры из полей ввода. Создает объекты ‘ProductionStage’ и ‘SpecialProductionStage’ в зависимости от этапа. Инициализирует и запускает оптимизатор производства 'ProductionOptimizer'. Получает и отображает результаты оптимизации. В случае ошибки отображает сообщение об ошибке.



```
def add_stage_entries():
    for widget in stages_frame.winfo_children():  # Удаление всех виджетов в frames
        widget.destroy()
    try:
        stages = int(num_stages.get())  # Получение количества этапов из пользовательского ввода
        global demand_entries, storage_cost_entries  # Глобальные переменные для хранения виджетов ввода
        demand_entries = []
        storage_cost_entries = []

        # Добавление заголовков к столбцам
        ttk.Label(stages_frame, text="Этап").grid(row=0, column=0, padx=5, pady=5)  # Заголовок для столбца "Этап"
        ttk.Label(stages_frame, text="Спрос на этапе").grid(row=0, column=1, padx=5, pady=5)  # Заголовок для столбца "Спрос на этапе"
        ttk.Label(stages_frame, text="Стоимость хранения на этапе").grid(row=0, column=2, padx=5, pady=5)  # Заголовок для столбца "Стоимость хранения на этапе"

        for i in range(stages):
            ttk.Label(stages_frame, text=f"{i + 1}").grid(row=i + 1, column=0, padx=5, pady=5)  # Метка для номера этапа
            demand_entry = ttk.Entry(stages_frame)  # Поле ввода для спроса на этапе
            demand_entry.grid(row=i + 1, column=1, padx=5, pady=5)  # Размещение поля ввода для спроса на этапе
            demand_entries.append(demand_entry)  # Добавление поля ввода в список
            storage_cost_entry = ttk.Entry(stages_frame)  # Поле ввода для стоимости хранения на этапе
            storage_cost_entry.grid(row=i + 1, column=2, padx=5, pady=5)  # Размещение поля ввода для стоимости хранения на этапе
            storage_cost_entries.append(storage_cost_entry)  # Добавление поля ввода в список
    except ValueError:
        messagebox.showerror("Ошибка", "Введите корректное количество этапов")  # Сообщение об ошибке при некорректном вводе
```

Рисунок 9 — Определение функции ‘add_stage_entries’

На рисунке 9 функция ‘add_stage_entries’ - добавляет поля ввода для каждого этапа производства на основе количества этапов. Очищает предыдущие виджеты, если они существуют. Добавляет метки для каждого этапа, а также поля ввода для спроса и стоимости хранения. В случае ошибки отображает сообщение об ошибке.

```
    root = tk.Tk()  # Создание главного окна приложения
    root.title("Production Optimizer")  # Заголовок окна

    main_frame = ttk.Frame(root, padding="10")  # Основной фрейм для размещения виджетов
    main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))  # Размещение основного фрейма

    ttk.Label(main_frame, text="Количество этапов:").grid(row=0, column=0, padx=5, pady=5)  # Метка для ввода количества этапов
    num_stages = ttk.Entry(main_frame, width=5)  # Поле ввода для количества этапов
    num_stages.grid(row=0, column=1, padx=5, pady=5)  # Размещение поля ввода для количества этапов
    ttk.Button(main_frame, text="Задать", command=add_stage_entries).grid(row=0, column=2, padx=5, pady=5)  # Кнопка для задания количества этапов

    stages_frame = ttk.Frame(main_frame, padding="10")  # Фрейм для размещения полей ввода спроса и стоимости хранения
    stages_frame.grid(row=1, column=0, columnspan=3, sticky=(tk.W, tk.E, tk.N, tk.S))  # Размещение фрейма для полей ввода

    ttk.Button(main_frame, text="Рассчитать", command=calculate).grid(row=2, column=0, columnspan=3, padx=5, pady=10)  # Кнопка для запуска расчета

    results_text = tk.StringVar()  # Переменная для хранения текста результатов
    results_label = ttk.Label(main_frame, textvariable=results_text)  # Метка для отображения результатов
    results_label.grid(row=3, column=0, columnspan=3, padx=5, pady=10)  # Размещение метки для отображения результатов

    root.mainloop()  # Запуск основного цикла приложения

# Запуск графического интерфейса
run_gui()
```

Рисунок 10 — Шаги создания графического интерфейса.

На рисунке 10 блок создает графический интерфейс для ввода данных о производственных этапах и отображения результатов оптимизации производства. Он включает в себя поля для ввода количества этапов, спроса и стоимости хранения на каждом этапе, кнопку для запуска расчета и отображения результатов, а также место для вывода результатов. ‘root.mainloop()’ - запускает основной цикл приложения, чтобы окно оставалось открытым и реагировало на действия пользователя.

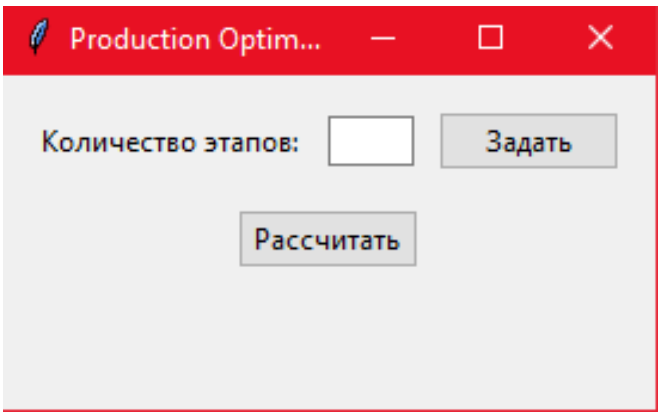


Рисунок 11 — Окно ввода количества этапов

На рисунке 11 показан интерфейс, где можно нажать на кнопку «Задать», при помощи которой задается требуемое количество этапов. Также есть кнопка «Рассчитать» для перехода на следующий интерфейс.

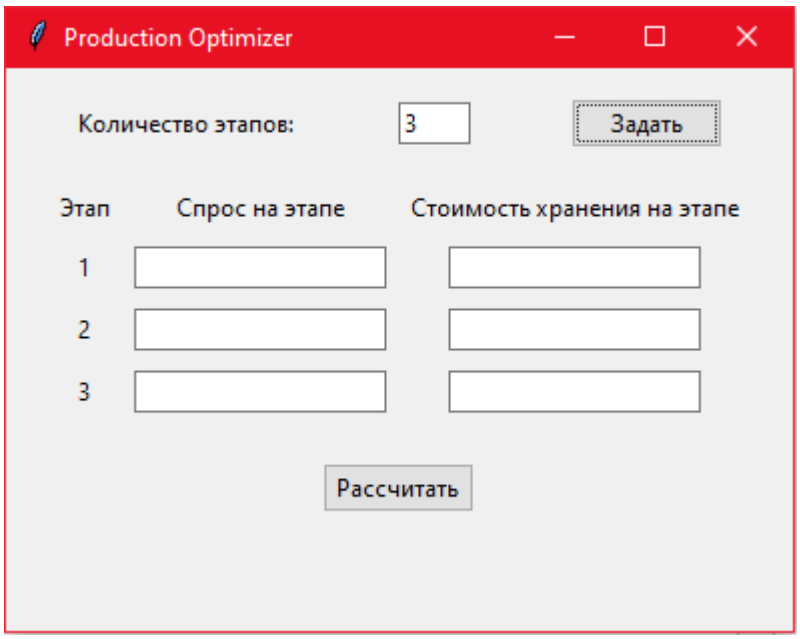


Рисунок 12 — Окно ввода требуемых единиц для каждого этапа

На рисунке 12 показан интерфейс с полями, которые обязательно нужно заполнить, для расчета минимальных общих затрат. Также можно поменять количество этапов.

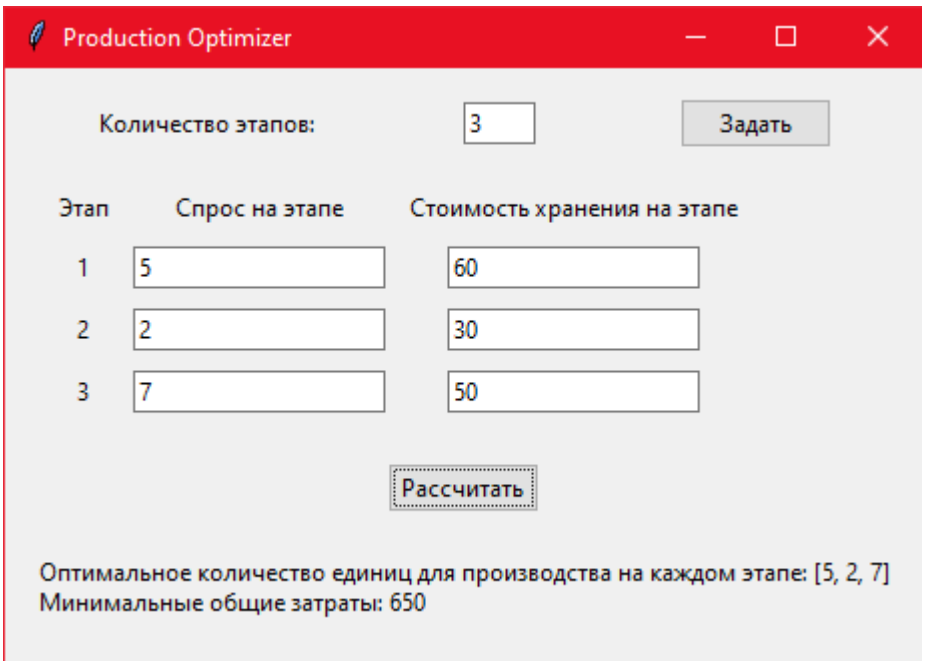


Рисунок 13 — Окно с результатами

На рисунке 13 продемонстрирован интерфейс с окном вывода итоговой информации. Из результатов тут можно увидеть минимальные общие затраты и оптимальное количество единиц для производства на каждом этапе.

Заключение

Тема бакалаврской работы посвящена актуальной проблеме реализации алгоритма решения производственной задачи управления запасами на основе метода динамического программирования.

В ходе выполнения бакалаврской работы была проделана следующая работа:

а) исследован общий подход метода динамического программирования к производственной задачи управления запасами;

б) продемонстрировано аналитическое решение конкретной производственной задачи управления запасами с помощью динамического программирования;

в) составлен и выполнен программный алгоритм для сформулированной задачи управления запасами.

Были проанализированы всевозможные математические методы, с помощью которых можно было бы решить задачу управления запасами. Из анализа следует вывод, что динамическое программирование остается самым мощным инструментом для решения задач управления запасами из-за своей способности точно моделировать и оптимизировать последовательные процессы и решения.

Для удобства пользования в программе был реализован простой интерфейс. Программа позволяет добавлять дополнительные сложности в расчеты, а также улучшать интерфейс, если это необходимо.

Основным результатом выполненной выпускной квалификационной работы является построение математической модели задачи управления запасами, разработка алгоритма решения задачи оптимизации на основе динамического программирования, программная реализация разработанного алгоритма.

Результаты работы могут быть полезны для решения проблем, связанных с управлением запасами на производстве.

Список используемой литературы и используемых источников

1. Беллман Р. Динамическое программирование. − М.: Репринт оригинального издания иностранной литературы, 1960 год, 2012.
2. Введение в исследование операций. Линейное программирование. − Томск: НТЛ, 2009. − 200 с.
3. Доусон М. Программируем на Python. − СПб.: Питер, 2012. − 432 с.
4. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Глава 15. Динамическое программирование // Алгоритмы: построение и анализ = IntroductiontoAlgorithms / Под ред. И. В. Красикова. − 2-е изд. − М.: Вильямс, 2005. − 1296 с.
5. Лежнев, А.В. Динамическое программирование в экономических задачах [Текст]: учеб. пособие / Лежнев А.В. − Москва: Бином, 2010. − 176 с.
6. Маккинли У. Python и анализ данных. — Перевод с английского. − М.: ДМК Пресс, 2015. − 482 с. − SanjoyDasgupta, ChristosH. Papadimitriou, UmeshVazirani. Algorithms = Algorithms. − 1-е изд. − McGraw-Hill Science/Engineering/Math, 2006. − С. 336.
7. Марк Лутц. Программирование на Python / Пер. с англ. − 4-е изд. − СПб.: Символ-Плюс, 2011. - Т. I. − 992 с.
8. Марк Саммерфилд. Python на практике. — Перевод с английского. − М.: ДМК Пресс, 2014. − 338 с.
9. Некрасова М.Г. Методы оптимизации и теория управления: учебное пособие / М.Г. Некрасова. Комсомольск-на-Амуре: КНАГТУ, 2007. − 132 с.
10. Окулов С.М. Динамическое программирование. – М.: Бином. Лаборатория знаний, 2017. − 296 с.
11. Окулов С.М. Динамическое программирование / С.М. Окулов, О.А. Пестов. Москва: БИНОМ. Лаборатория знаний, 2012. − 260 с.

12. Фёдоров Д. Ю. Основы программирования на примере языка Python / Учебное пособие. − СПб.: Юрайт, 2018. − 167 с.

13. Чаплыгин А.Н. Учимся программировать вместе с Питоном. Учебник. − ревизия 226. − 135 с.

14. Ширяв В.И. Исследование операций и численные методы оптимизации: учебное пособие / В.И. Ширяв. Москва: Ленанд, 2017. − 224 с.

15. В.И. Струченкова, Д.А. Карпова «Динамическое программирование в прикладных задачах специального вида». Учебник – Синергия, 2020. – 14 с.

16. Р.Ф. Габасов, Ф.М. Кириллова «Основы динамического программирования». Учебник − Ленанд, 2020. – 264 с.

17. Вирт Никлаус. Алгоритмы и структуры данных. – Перевод с английского. – М.: Ткачев Ф.В., 2016. – 272 с.

18. А.В. Лежнев. Динамическое программирование в экономических задачах. – Лаборатория знаний, 2014. – 213с.

19. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Г. Алгоритмы: построение и анализ / Под ред. И. В. Красикова. − 3-е изд. − М.: Вильямс, 2020. − 1328 с.

20. Солтис Майкл. Введение в анализ алгоритмов – Перевод с английского. – М.: Логунов А.В., 2019. – 278 с.

21. Bertsekas, D. P. (2017), Dynamic Programming and Optimal Control (4th ed.), Athena Scientific.

22. Giegerich, R.; Meyer, C.; Steffen, P. (2004), "A Discipline of Dynamic Programming over Sequence Data" (PDF), Science of Computer Programming, 51 (3): 215–263, doi:10.1016/j.scico.2003.12.005.

23. Hamilton, Naomi (5 August 2008). "TheA-ZofProgrammingLanguages: Python". Computerworld. Archived from the originalon 29 December 2008. Retrieved 31 March 2010.

24. Lutz, Mark (2013). LearningPython (5thed.). O'ReillyMedia.

25. Meyn, Sean (2007), Control Techniques for Complex Networks, Cambridge University Press, archived from the original on 2010-06-19

Приложение А

Программная реализация разработанного алгоритма

```
import tkinter as tk
from tkinter import ttk, messagebox

# Базовый класс для управления производственными этапами
class ProductionStage:
    def __init__(self, demand, storage_cost):
        self.demand = demand  # спрос на этом этапе
        self.storage_cost = storage_cost  # стоимость хранения на этапе

    # Метод для расчета стоимости хранения
    def calculate_storage_cost(self, y_prev):
        if y_prev > 0:
            return self.storage_cost * y_prev
        return 0

# Производный класс, модифицирующий расчет стоимости хранения
class SpecialProductionStage(ProductionStage):
    # Переопределенный метод для расчета стоимости хранения с модификацией
    def calculate_storage_cost(self, y_prev):
        base_cost = super().calculate_storage_cost(y_prev)  # вызов метода базового класса
        return base_cost * 2  # удваиваем стоимость хранения

# Класс для расчета стоимости производства
class ProductionCost:
    def __init__(self, a, b, c):
        self.a = a  # параметр a в формуле затрат
        self.b = b  # параметр b в формуле затрат
        self.c = c  # параметр c в формуле затрат

    # Метод для расчета стоимости производства
    def calculate(self, x):
        return self.a * x**2 + self.b * x + self.c

# Оптимизатор производства, управляющий расчетом оптимального количества и стоимости
class ProductionOptimizer:
    def __init__(self, stages, production_cost):
        self.stages = stages  # список этапов производства
```

```
        self.production_cost = production_cost  # объект для расчета затрат на производство
        self.optimal_production = [0] * len(stages)  # инициализация списка оптимального производства
        self.total_cost = None  # общая стоимость производства

    # Расчет оптимального производства
    def calculate_optimal_production(self):
        F = [{} for _ in range(len(self.stages))]  # таблица для хранения минимальных стоимостей

        # Перебор всех этапов
        for stage_index, stage in enumerate(self.stages):
            # Перебор возможных остатков на текущем этапе
            for y_next in range(sum(stage.demand for stage in self.stages[stage_index:]) + 1):
                best_cost = float('inf')
                best_x = 0

                # Перебор возможных объемов производства
                for x in range(stage.demand, sum(stage.demand for stage in self.stages[:stage_index + 1]) + 1):
                    y_prev = x - stage.demand + (0 if stage_index == 0 else sum(stage.demand for stage in self.stages[:stage_index]) - sum(self.optimal_production[:stage_index]))
                    cost = self.production_cost.calculate(x)  # расчет затрат на производство
                    cost += stage.calculate_storage_cost(y_prev)  # добавление стоимости хранения
                    if stage_index > 0:
                        cost += min(F[stage_index - 1].values())  # добавление минимальной предыдущей стоимости

                    # Поиск минимальной стоимости
                    if cost < best_cost:
                        best_cost = cost
                        best_x = x

                F[stage_index][y_next] = best_cost
                self.optimal_production[stage_index] = best_x

        self.total_cost = min(F[-1].values())  # определение минимальной стоимости
```

```
    # Получение результатов расчета
    def get_results(self):
        return self.optimal_production, self.total_cost

# Функция для запуска графического интерфейса
def run_gui():
    def calculate():
        try:
            stages = int(num_stages.get())  # Получение количества этапов из пользовательского ввода
            stages_list = []

            for i in range(stages):
                demand = int(demand_entries[i].get())  # Получение спроса на этапе из пользовательского ввода
                storage_cost = int(storage_cost_entries[i].get())  # Получение стоимости хранения на этапе из пользовательского ввода
                if i % 2 == 0:
                    stages_list.append(SpecialProductionStage(demand, storage_cost))  # Использование специального класса для каждого второго этапа
                else:
                    stages_list.append(ProductionStage(demand, storage_cost))  # Использование базового класса для остальных этапов

            production_cost = ProductionCost(8, 1, 4)  # Инициализация объекта для расчета затрат на производство
            optimizer = ProductionOptimizer(stages_list, production_cost)  # Инициализация оптимизатора производства
            optimizer.calculate_optimal_production()  # Расчет оптимального производства

            optimal_production, total_cost = optimizer.get_results()  # Получение результатов расчета
            results_text.set(f"Оптимальное количество единиц для производства на каждом этапе: {optimal_production}\nМинимальные общие затраты: {total_cost}")
        except Exception as e:
            messagebox.showerror("Ошибка", f"Произошла ошибка: {e}")

    def add_stage_entries():
        for widget in stages_frame.winfo_children():  # Удаление всех виджетов в frames
            widget.destroy()
        try:
```

```
        stages = int(num_stages.get())  # Получение количества этапов из пользовательского ввода
        global demand_entries, storage_cost_entries  # Глобальные переменные для хранения виджетов ввода
        demand_entries = []
        storage_cost_entries = []

        # Добавление заголовков к столбцам
        ttk.Label(stages_frame, text="Этап").grid(row=0, column=0, padx=5, pady=5)  # Заголовок для столбца "Этап"
        ttk.Label(stages_frame, text="Спрос на этапе").grid(row=0, column=1, padx=5, pady=5)  # Заголовок для столбца "Спрос на этапе"
        ttk.Label(stages_frame, text="Стоимость хранения на этапе").grid(row=0, column=2, padx=5, pady=5)  # Заголовок для столбца "Стоимость хранения на этапе"

        for i in range(stages):
            ttk.Label(stages_frame, text=f"{i + 1}").grid(row=i + 1, column=0, padx=5, pady=5)  # Метка для номера этапа
            demand_entry = ttk.Entry(stages_frame)  # Поле ввода для спроса на этапе
            demand_entry.grid(row=i + 1, column=1, padx=5, pady=5)  # Размещение поля ввода для спроса на этапе
            demand_entries.append(demand_entry)  # Добавление поля ввода в список
            storage_cost_entry = ttk.Entry(stages_frame)  # Поле ввода для стоимости хранения на этапе
            storage_cost_entry.grid(row=i + 1, column=2, padx=5, pady=5)  # Размещение поля ввода для стоимости хранения на этапе
            storage_cost_entries.append(storage_cost_entry)  # Добавление поля ввода в список
    except ValueError:
        messagebox.showerror("Ошибка", "Введите корректное количество этапов")  # Сообщение об ошибке при некорректном вводе

root = tk.Tk()  # Создание главного окна приложения
root.title("Production Optimizer")  # Заголовок окна

main_frame = ttk.Frame(root, padding="10")  # Основной фрейм для размещения виджетов
main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))  # Размещение основного фрейма

ttk.Label(main_frame, text="Количество этапов:").grid(row=0, column=0,
```

```
padx=5, pady=5)  # Метка для ввода количества этапов
    num_stages = ttk.Entry(main_frame, width=5)  # Поле ввода для количества этапов
    num_stages.grid(row=0, column=1, padx=5, pady=5)  # Размещение поля ввода для количества этапов
    ttk.Button(main_frame, text="Задать", command=add_stage_entries).grid(row=0, column=2, padx=5, pady=5)  # Кнопка для задания количества этапов

    stages_frame = ttk.Frame(main_frame, padding="10")  # Фрейм для размещения полей ввода спроса и стоимости хранения
    stages_frame.grid(row=1, column=0, columnspan=3, sticky=(tk.W, tk.E, tk.N, tk.S))  # Размещение фрейма для полей ввода

    ttk.Button(main_frame, text="Рассчитать", command=calculate).grid(row=2, column=0, columnspan=3, padx=5, pady=10)  # Кнопка для запуска расчета

    results_text = tk.StringVar()  # Переменная для хранения текста результатов
    results_label = ttk.Label(main_frame, textvariable=results_text)  # Метка для отображения результатов
    results_label.grid(row=3, column=0, columnspan=3, padx=5, pady=10)  # Размещение метки для отображения результатов

    root.mainloop()  # Запуск основного цикла приложения

# Запуск графического интерфейса
run_gui()
```