Санкт-Петербургский Государственный Университет

Кафедра технологии программирования

Исхаков Александр Александрович

Выпускная квалификационная работа бакалавра

Алгоритмы нахождения оптимальных траекторий в целочисленной модели Неймана и их реализация на языке C++

Направление 010300

Фундаментальная информатика и информационные технологии

Научный руководитель:

ассистент

Парфенов А. П.

Санкт-Петербург

2016

Содержание

Введение

Экономическая модель, представленная Джоном фон Нейманом в опубликованной в 1937 году статье[1] (перевод на английский язык вышел в 1945 году [1]), оказала значимое влияние на развитие математической экономики, послужив источником большому количеству исследований и породив множество интерпретаций. Модель Неймана является обобщением известной модели Леонтьева, допускающим производство каждым технологическим процессом более одного вида продуктов. Значительное число учебных пособий по математической экономике [2] [3] [4], в том числе современных [5], содержит выведенные в оригинальной статье фон Неймана описание модели и исследования динамического равновесия как сбалансированного роста без изменения структуры, расширенные и дополненные другими авторами. Обзор описанной в этой литературе классической модели Неймана, без углубления в условия существования и свойства равновесия, приведен в соответствующем разделе. Важным следствием исследований динамического равновесия являются теоремы о магистралях, которые не рассматриваются в рамках этой работы — для ознакомления можно обратиться к [6].

Модель Неймана наглядна и широко применима для моделирования экономических систем разного масштаба, и эта работа акцентирует внимание на системах малого масштаба (уровень отдельных предприятий), для которых дискретность изменения интенсивности технологических процессов не позволяет считать погрешность от использования непрерывной модели незначительной. Автору не удалось найти литературу, посвященную этому вопросу, при этом, по мнению автора, исследование оптимизационной задачи для целочисленной модели может помочь понять, как быстро

[1] *Neumann J. von.* Über ein ökonomisches Gleichungssystem und eine Verallgemeinerung des Brouwerschen Fixpunktsatzes // Ergebnisse eines Mathematischen Kolloquiums, 1937. No 8. P. 73-83.

растет оптимальное решение для таких систем.

В работе применяются несколько различных методов решения оптимизационной задачи для целочисленной модели, представляющие различные подходы к задаче (статический и динамический, с нахождением точного и приближенного решения), выясняются их достоинства, недостатки и ограничения. Также в рамках данной работы выполнена программная реализация этих методов, позволяющая проверить эффективность и провести более детальное сравнение.

Классическая модель Неймана в литературе

Этот раздел посвящен формальному описанию классической модели Неймана — динамической линейной модели производства, какой она представлена в экономической литературе. Эта модель «представляет собой существенный шаг вперед по сравнению с моделью Леонтьева» [4] — леонтьевская модель тоже допускает динамическую формулировку, однако в модели Неймана технологические процессы могут производить более одного продукта, что позволяет оперировать не только понятием «чистой» отрасли.

Рассмотрим экономическую модель, в которой существует $n \in \mathbb{N}$ *продуктов* и $m \in \mathbb{N}$ линейных *технологических процессов*, перерабатывающих некоторые количества продуктов в другие количества этих же продуктов.

Под продуктами подразумеваются первичные факторы производства (земля, труд, капитал), сырье, товары и услуги — все, что необходимо для производства и все, что в результате оказывается произведено [4, § 2.1].

Линейность технологических процессов понимается в значении линейной зависимости количества произведенных товаров от количества ресурсов.

Каждый i-тый процесс можно описать при помощи двух неотрицательных векторов: $a_i = \{a_{i1}, a_{i2}, ..., a_{in}\}$, который выражает количество затрачиваемых товаров, и $b_i = \{b_{i1}, b_{i2}, ..., b_{in}\}$, который выражает количество производимых.

Определение. Неотрицательная матрица $A = (a_{ij})_{i=1,j=1}^{m,n}$ называется *матрицей затрат*, а неотрицательная матрица $B = (b_{ij})_{i=1,j=1}^{m,n}$ — *матрицей выпуска*.

Матрицы A и B неотрицательны, поскольку элементы матриц представляют количества товаров. Также эти матрицы по своей природе являются

разреженными, так как подавляющее число процессов задействует лишь определенное число разных продуктов, обычно сильно меньшее чем общее число n, представленное в модели.

Чтобы «модель функционировала желательным образом» [3, § 9.5] — всякий продукт выпускался хотя бы одним процессом и во всяком процессе хотя бы один продукт затрачивался — наложим на матрицы A и B следующие ограничения:

$$\begin{aligned} &\forall i\ \exists j\ a_{ij} > 0, \\ &\forall j\ \exists i\ b_{ij} > 0. \end{aligned} \tag{1}$$

Динамику модели мы будем рассматривать на промежутке времени $[0, T]$. И хотя производство — непрерывный процесс, для существенного упрощения работы с моделью удобно допустить дискретность времени.

Другое важное допущение: продолжительность всех технологических процессов составляет одну единицу времени. В статье фон Неймана утверждается, что процессы с большей продолжительностью можно разбить на процессы единичной продолжительности, при необходимости вводя промежуточные ресурсы как дополнительные продукты [1]. Докажем это, сформулировав утверждение в следующей форме:

Утверждение 1. *Модель с различной продолжительностью технологических процессов можно свести к модели Неймана.*

Доказательство. Докажем утверждение для модели, в которой два технологических процесса Q_1 и Q_2 и n продуктов.

Пусть Q_1 длится r ед. времени, а Q_2 — s, при этом НОД$(r, s) = p$, $r/p = k$, $s/p = l$. Если мы примем за новую единицу времени временной промежуток в p ед., то Q_1 будет занимать время k.

Разобьем Q_1 на k последовательных процессов: для этого заменим его на процессы Q_1^1, Q_1^2, ..., Q_1^k, длящихся 1 единицу времени, и введем $k - 1$

дополнительных ресурсов. Для Q_1^1 вектор затрат будет таким же, какой был у Q_1, а вектор выпуска будет иметь единственное ненулевое значение только для элемента, соответствующего первому дополнительному ресурсу. Вектор затрат Q_1^2 равен вектору выпуска Q_1^1, а единственное ненулевое значение вектора выпуска Q_1^2 будет соответствовать второму дополнительному ресурсу, и так далее. Последний процесс Q_1^k будет иметь вектор выпуска, равный вектору выпуска Q_1. Вместе все нововведенные процессы за время k затрачивают и производят те же продукты, что и Q_1, а требование специальных дополнительных ресурсов не позволит запустить какой-либо из внутренних процессов отдельно.

Q_2 в новых единицах времени будет занимать время l. Так же, как и Q_1, разобьем Q_2 на l последовательных процессов, длящихся 1 единицу времени.

В конечном счете мы получили модель с $n+k+l-2$ продуктами и $k+l$ процессами с одинаковой продолжительностью, причем по своему смыслу она полностью совпадает с исходной.

Аналогичным образом можно свести модель с произвольным количеством процессов m продолжительностью t_1, t_2, ..., t_m. После преобразований мы получим $\frac{t_1+t_2+...+t_m}{\text{НОД}(t_1,t_2,...,t_m)}$ процессов и $n+\frac{t_1+t_2+...+t_m}{\text{НОД}(t_1,t_2,...,t_m)}-m$ ресурсов. □

Исходные m технологических процессов называются *базисными*. Соответствующие им строки матриц затрат и выпуска определяют технологический потенциал, заложенный в каждом процессе. Для того, чтобы выразить, насколько процесс участвует в общем производстве в конкретный момент времени, используют понятие *интенсивности*.

Представим производство в момент времени t как технологический процесс, представляющий из себя линейную комбинацию базисных процессов. Векторы затрат и выпуска этого процесса будут иметь вид:

$$a_t = \{\sum_{i=1}^{m} z_i a_{i1}, \sum_{i=1}^{m} z_i a_{i2}, ..., \sum_{i=1}^{m} z_i a_{in}\} = zA,$$

$$b_t = \{\sum_{i=1}^{m} z_i b_{i1}, \sum_{i=1}^{m} z_i b_{i2}, ..., \sum_{i=1}^{m} z_i b_{in}\} = zB,$$

где $z_1, z_2, ..., z_n$ — некоторые характерные для момента времени t коэффициенты, называемые интенсивностями.

Определение. *Вектором интенсивностей* в момент времени t называется неотрицательный вектор $z_t = \{z_1, z_2, ..., z_m\}$.

Определение. Вектор x *больше или равен (меньше или равен)* вектора y той же размерности, если каждый элемент x_i вектора x больше или равен (меньше или равен) соответствующему элементу y_i вектора y. Вектор x *больше (меньше)* вектора y той же размерности, если существует хотя бы один элемент x_i вектора x, строго больший (меньший) соответствующего элемента y_i вектора y, а все остальные элементы x больше или равны (меньше или равны) соответсвующим элементам y.

Определение. Экономическая модель называется *замкнутой*, если в ней отсутствует поступление ресурсов извне и экспорт продуктов.

Наложим на рассматриваемую модель условие замкнутости. Более того, введем более строгое ограничение: в период $(t, t+1)$ мы можем тратить только произведенные в период $(t-1, t)$ продукты (в пособии Ашманова это ограничение и названо условием замкнутости [4, § 2.1], однако необходимо сказать о продуктах, произведенных на предыдущих промежутках времени, что будет сделано ниже). Так как векторы затрат и выпуска в момент времени t выражаются как $z_t A$ и $z_t B$, то из этих условий мы получаем следующие неравенства:

$$\begin{aligned} &z_{t+1} A \leqslant z_t B,\ t \in \{1, ..., T-1\}, \\ &z_1 A \leqslant y_0, \end{aligned} \tag{2}$$

где y_0 — вектор запасов на начало рассматриваемого периода времени $[0, T]$.

Условие об использовании только произведенного на предыдущем промежутке времени не означает, однако, что данная экономическая модель не подразумевает хранения продуктов. Хранение без потерь можно представить как технологический процесс, вектор затрат которого равен вектору выпуска. При этом модель Неймана «хорошо приспособлена, в частности, к анализу моделей с капиталом, изнашиваемым при использовании» [2, § 10.3] Зная срок годности или вероятность поломки того или иного продукта, в технологический процесс хранения можно заложить и износ продукта, сделав соответствующий элемент вектора выпуска меньше, чем в векторе затрат. Так же и условие замкнутости означает не столько полную изоляцию системы, сколько то, что импорт и экспорт необходимо выражать через технологические процессы.

В своей модели фон Нейман также рассматривал *цены* на товары. Точно так же, как и интенсивность для технологических процессов, цены на продукты изменяются во времени. При этом векторы цен двойственны векторам интенсивности [1].

Определение. *Вектором цен* в момент времени t называется неотрицательный вектор $p_t = \{p_1, p_2, ..., p_m\}$.

Условием, двойственным (2), является *условие неприбыльности базисных процессов*:

$$Ap_t \geqslant Bp_{t+1},\ t \in \{1, ..., T-1\}. \tag{3}$$

Это условие обусловлено тем, что основной интерес для фон Неймана представляло состояние *равновесия*, при котором вектор интенсивностей растет во времени с постоянным коэффициентом, а цены остаются неизменными. Если же какой-то процесс является прибыльным, то это приводит к росту цен и нарушает равновесие [1]. Можно вспомнить условие экономического равновесия в условиях совершенной конкуренции — равенство спроса и предложения: при нем тоже не создается никакой прибыли [5, § 5.3].

Для денежной массы в модели существуют следующие условия: она не изменяется со временем и постоянно находится в обращении. Это означает, что цены на продукты на промежутке $(t, t+1)$ устанавливаются таким образом, чтобы общая выручка от реализации товаров, произведенных на промежутке $(t-1, t)$, равнялась денежным затратам на ресурсы на этом же промежутке $(t-1, t)$. При этом вся эта выручка должна быть полность потрачена на ресурсы для производства на промежутке $(t, t+1)$. Математически через матрицы затрат и выпуска и векторы интенсивности и цен это можно записать как:

$$z_t A p_t = z_t B p_{t+1},$$

$$z_t B p_{t+1} = z_{t+1} A p_{t+1},\ t \in \{1, ..., T-1\},$$

Определение. *Моделью Неймана* называется динамическая модель с дискретным временем и линейной технологией, в которой определены n продуктов и m базисных технологических процессов и действует следующая система ограничений:

$$\begin{cases} z_{t+1} A \leqslant z_t B, \\ z_1 A \leqslant y_0, \\ A p_t \geqslant B p_{t+1}, \\ z_t A p_t = z_t B p_{t+1}, \\ z_t B p_{t+1} = z_{t+1} A p_{t+1},\ t \in \{1, ..., T-1\}, \end{cases} \tag{4}$$

где A и B — матрицы затрат и выпуска, удовлетворяющие ограничениям (1), z_t и p_t — векторы интенсивностей и цен в момент времени t, y_0 — вектор начальных запасов.

Постановка задачи

Классическая модель Неймана с системой ограничений (4), описанная в предыдущем пункте, удобна для исследования экономического равновесия теоретической экономики с совершенной конкуренцией. Однако модель Неймана в целом имеет куда более широкое применение. Например, мы можем не рассматривать двойственные ограничения, связанные с вектором цен — ведь в реальных экономиках, далеких от идеальной экономики с совершенной конкуренцией, не соблюдаются неприбыльность процессов и неизменность денежной массы. Тогда можно вообще не рассматривать вектор цен.

Модель Неймана применима для описания реальных объектов самого разного масштаба: от экономики целых стран до производства одного предприятия. Однако на малых масштабах непрерывность значений вектора интенсивностей не соответствует реальному положению дел, поэтому для задач такого типа удобнее рассматривать модель Неймана с целочисленными интенсивностями.

Применив вышеуказанные замечания к классической модели Неймана, мы получим модификацию модели, которая в дальнейшем в работе будет называться *целочисленной моделью Неймана*.

Определение. *Целочисленной моделью Неймана* будем называть динамическую модель с дискретным временем и линейной технологией, в которой определены n продуктов и m базисных технологических процессов и действует следующая система ограничений:

$$\begin{cases} z_{t+1}A \leqslant z_tB, \ t \in \{1, ..., T-1\}, \\ z_1A \leqslant y_0, \\ z_t \in \mathbb{Z}_+^m, \ t \in \{1, ..., T\}, \end{cases} \tag{5}$$

где A и B — матрицы затрат и выпуска, удовлетворяющие ограничениям (1), z_t — вектор интенсивностей в момент времени t, y_0 — вектор начальных запасов.

Теперь, когда рассматриваемая модель определена, формализуем задачу оптимизации. *Допустимым множеством* задачи будет являться множество наборов из T векторов интенсивностей в $\mathbb{N}^m$, удовлетворяющих ограничениям (5). Непосредственно задача оптимизации заключается в том, чтобы выбрать из допустимого множества некоторый определенный набор векторов $z_t,\ t \in \{1, ..., T\}$ (этот набор векторов мы будем называть *оптимальным решением* или *оптимальной траекторией*), максимизирующих некоторую функцию, называемую целевой. Теперь определимся с *целевой функцией*, выбор которой зависит от подхода к понятию *полезности*.

Определение. *Полезность* — мера удовлетворения потребностей индивидов при потреблении продуктов. *Общая полезность* представляет собой совокупную полезность при потреблении всех единиц продукта, а *предельная* — полезность при потреблении еще одной единицы продукта.

Хотя в определении полезности фигурирует слово «потребление», данное понятие применимо не только к потребителям, но и к производителям — в контексте модели Неймана мерой удовлетворения потребностей производителя может выступать количество произведенных продуктов.

Из *закона убывающей полезности* [5, § 3.2] известно, что предельная полезность с увеличением количества единиц продукта постепенно снижается. Однако на достаточно близком к началу интервале зависимость общей полезности от количества продуктов практически линейна, что позволяет рассматривать упрощенную *линейную* полезность.

Тогда общая полезность некоторого количества одного продукта будет линейно зависеть от количества, поэтому полезность продукта удоб-

но выразить через некоторый коэффициент. Для вычисления общей полезности всей сокупности продуктов введем вектор коэффициентов $c = \{c_1, c_2, ..., c_n\}$, каждый из n элементов которого будет выражать влияние каждого продукта на общую полезность.

Если мы ставим перед собой задачу к концу заданного периода добиться максимально возможного прироста производства, то речь идет о *терминальной* полезности. Такая задача формализуется просто — необходимо максимизировать общую полезность всей произведенной продукции в конечный момент времени T.

Поскольку полезность продуктов мы определяем с помощью вектора коэффициентов, а вектор количества продуктов в момент T равен $z_T B$, *задачу оптимизации терминальной полезности* можно сформулировать как:

$$\max_{z_t,\ t \in \{1..T\}} c(z_T B)^T \tag{6}$$

В целом в данной работе рассматривается терминальная полезность, однако один из методов оптимизации допускает некоторое обощение и позволяет перейти к более общему понятию интегральной полезности.

При оптимизации интегральной полезности учитываются показатели на всей протяженности рассматриваемого временного промежутка. Однако возникает вопрос — полезность чего имеет смысл оптимизировать? Рассматривать полезность всего объема продукции на каждом шаге не имеет смысла, поскольку некоторая часть продуктов на каждом шаге будет затрачиваться на выпуск продуктов на следующем шаге. К тому же такую задачу можно свести к ряду последовательных задач оптимизации терминальной полезности, и не будет необходимости рассматривать отдельный вид задач.

Задача оптимизации интегральной полезности оказывается актуальной

в контексте поддержания высокого уровня потребления. Введем понятие «потребление» для модели Неймана.

Определение. *Потреблением* в момент t называется разность между количеством произведенных на шаге t продуктов и количеством продуктов, затрачиваемых на производство на шаге $t+1$: $z_tB - z_{t+1}A$. *Начальным потреблением* (потреблением на нулевом шаге) тогда будет считаться разница между начальными запасами и затратами на производство на первом шаге: $y_0 - z_1A$. На последнем шаге T будем считать потребленными все произведенные продукты: z_TB.

Полезность потребления может быть неоднородной относительно времени, поэтому введем коэффициенты k_0, k_1, ..., k_T, которые будут выражать эту неоднородность. С учетом вектора коэффициентов полезности продуктов c, задача оптимизации интегральной полезности формулируется следующим образом:

$$\max_{z_t,\ t\in\{1..T\}} c\left(k_0y_0 + \sum_{i=1}^{T}(k_i(z_iB)^T - k_{i-1}(z_iA)^T)\right) \tag{7}$$

Можно заметить, что задача терминальной полезности является частным случаем задачи интегральной полезности при $k_0 = 0$, ..., $k_{T-1} = 0$, $k_T = 1$.

Итак, задача оптимизации определена, можно перейти к выбору методов решения.

Задача развернута во времени, т.е. является динамической, однако если рассматривать набор из T векторов размерности m как вектор размерности Tm, то задачу можно считать статической, что позволяет применять к ней методы решения статических задач. Такой подход называется *сведением к статической задаче*, и данный подход может оказаться весьма плодотворным, если получаемая статическая задача хорошо изучена и для нее имеются эффективные алгоритмы решения. Однако если временная

эффективность алгоритмов экспоненциально зависит от размерности, то их использование может быть нерациональным при достаточно больших T.

Другой подход к динамической задаче — представить ее как последовательность подзадач, находящих каждый из T векторов оптимального решения на основании уже найденных. Соответственно, для этих подзадач из оценки временной эффективности уходит T — что само по себе уже является большим преимуществом. Такой подход называется *динамическим программированием*.

Методы можно разделить на точные и приближенные. Точные методы, как это следует из названия, будут давать значение оптимального решения с нулевой погрешностью. Погрешность решений приближенных методов будет ненулевой, однако временная эффективность таких методов может быть значительно выше.

В данной работе будут рассмотрены два конкретных метода решения оптимизационной задачи: приближенный метод континуализации, сводящий целочисленную задачу к непрерывной статической, и точный метод динамического программирования.

Задача работы: подробное формальное описание этих методов, выделение их свойств, оценка временной эффективности, а также их программная реализация и сравнительный анализ на основе реализации.

Цель работы: рассмотреть различные подходы к решению оптимизационной задачи для целочисленной модели Неймана, понять границы применимости конкретных методов решения задачи и выявить их достоинства и недостатки.

Глава 1. Методы решения задачи оптимизации модели

Выпишем целиком оптимизационные задачи, методы решения которых мы будем рассматривать в этой главе:

1. Задача терминальной максимизации:

$$\max_{z_t,\ t\in\{1..T\}} c(z_T B)^T,$$
$$\begin{cases} z_{t+1}A \leqslant z_t B,\ t \in \{1,...,T-1\}, \\ z_1 A \leqslant y_0, \\ z_t \in \mathbb{Z}_+^m,\ t \in \{1,...,T\}. \end{cases} \tag{8}$$

2. Задача интегральной максимизации:

$$\max_{z_t,\ t\in\{1..T\}} c\left(k_0 y_0 + \sum_{i=1}^{T}(k_i(z_i B)^T - k_{i-1}(z_i A)^T)\right),$$
$$\begin{cases} z_{t+1}A \leqslant z_t B,\ t \in \{1,...,T-1\}, \\ z_1 A \leqslant y_0, \\ z_t \in \mathbb{Z}_+^m,\ t \in \{1,...,T\}. \end{cases} \tag{9}$$

1.1. Метод континуализации

В экономической литературе используется непрерывная модель Неймана, даже несмотря на то, что суть многих технологических процессов подразумевает целочисленность. И действительно, для расширяющихся экономик, в контексте которых модель Неймана и рассматривается в литературе, значения векторов z_t будут расти в геометрической прогрессии от t и допущение о непрерывности не должно сильно повлиять на значение решения оптимизационной задачи. Однако конкретный метод сведения к

непрерывной модели (*континуализации*) и погрешность не оговариваются. В этом параграфе будет рассмотрен один из таких методов, установлены налагаемые им ограничения и оценка погрешности оптимального решения.

Последующие рассуждения применимы для обеих задач терминальной (8) и интегральной (9) максимизации — далее в тексте задача с целочисленными ограничениями будет называться исходной. Поскольку в рассматриваемых далее модификациях задачи вид целевой функции остается неизменным, понятия «ограничения задачи» и «задача» используются как эквивалентные.

Откажемся от ограничения на целочисленность, получив непрерывную модель. Тогда вместо динамической задачи проще рассматривать статическую — выписав ограничения со всех моментов времени из $[0, T]$ в единую систему ограничений, мы получим задачу линейного программирования с ограничениями:

$$\begin{cases} z_1 A \leqslant y_0, \\ z_2 A \leqslant z_1 B, \\ ... \\ z_T A \leqslant z_{T-1} B, \\ z_t \geqslant \{0, 0, ..., 0\},\ t \in \{1, ..., T\}. \end{cases} \tag{10}$$

Что нам даст нахождение оптимального решения такой задачи? Очевидно, что полученное решение может быть вне множества допустимых значений исходной задачи. Но даже округление вниз не обязательно даст допустимое значение: неравенство $z_{t+1}A \leqslant z_t B$ не гарантирует выполнения $\lfloor z_{t+1} \rfloor A \leqslant \lfloor z_t \rfloor B$.

Пример 1. $z = (1, 0.5)$, $A = \begin{pmatrix} 2 & 2 \\ 50 & 50 \end{pmatrix}$, $B = \begin{pmatrix} 1 & 1 \\ 100 & 100 \end{pmatrix}$.

$$z_{t+1}A = \begin{pmatrix} 27 & 27 \end{pmatrix} \leqslant \begin{pmatrix} 51 & 51 \end{pmatrix} = z_t B,$$

$$\lfloor z_{t+1} \rfloor A = \begin{pmatrix} 2 & 2 \end{pmatrix} > \begin{pmatrix} 1 & 1 \end{pmatrix} = \lfloor z_t \rfloor B.$$

Возникает необходимость построить задачу, для которой округление оптимального решения будет давать допустимое решение исходной задачи.

Рассмотрим задачу с ограничениями вида:

$$\begin{cases} z_1 A \leqslant y_0, \\ z_2 A \leqslant (z_1 - \{1, 1, ..., 1\})B, \\ ... \\ z_T A \leqslant (z_{T-1} - \{1, 1, ..., 1\})B, \\ z_t \geqslant \{0, 0, ..., 0\},\ t \in \{1, ..., T\}. \end{cases} \tag{11}$$

Утверждение 2. *Округление вниз оптимального решения задачи* (11) *даст допустимое решение исходной задачи.*

Доказательство. Обозначим оптимальные значения векторов интенсивностей в задаче (11) как $z_t^*,\ t \in \{1, ..., T\}$. Тогда верно, что

$$z_{t+1}^* A \leqslant (z_t^* - \{1, 1, ..., 1\})B,\ t \in \{1, ..., T-1\}.$$

Из следующей цепочки очевидных неравенств:

$$\lfloor z_{t+1}^* \rfloor A \leqslant z_{t+1}^* A \leqslant (z_t^* - \{1, 1, ..., 1\})B \leqslant \lfloor z_t^* \rfloor B,$$

получаем выражение вида

$$\lfloor z_{t+1}^* \rfloor A \leqslant \lfloor z_t^* \rfloor B,\ t \in \{1, ..., T-1\}.$$

Так как $z_1^* A \leqslant y_0$, то $\lfloor z_1^* \rfloor A \leqslant y_0$. Условие целочисленности и неотрицательности $\lfloor z_t^* \rfloor \in \mathbb{Z}_+^m$ следует из неотрицательности $z_t^*,\ t \in \{1, ..., T\}$.

Таким образом, для $\lfloor z_t^* \rfloor,\ t \in \{1, ..., T\}$ выполняются все ограничения исходной задачи, а значит, оно является допустимым. □

Заметим, что задача (11) может не иметь решений — если $(z_t - \{1,1,...,1\})B \leqslant \{0,0,...,0\}$ для какого-либо $t \in \{1,...,T-1\}$, то в силу неотрицательности матрицы A не существует такого неотрицательного z_{t+1}, что $z_{t+1}A \leqslant (z_t - \{1,1,...,1\})B$. Допущения, достаточные для существования решений, будут обозначены немного позже.

Обозначим как $\varepsilon_t,\ t \in \{1,...,T\}$ некоторые конкретные векторы, для которых выполняется

$$\begin{cases} \varepsilon_{t+1}A \geqslant (\varepsilon_t + \{1,1,...,1\})B,\ t \in \{1,...,T-1\}, \\ \varepsilon_t \geqslant 0,\ t \in \{1,...,T\}. \end{cases} \tag{12}$$

Заметим, что такие векторы существуют, если каждый столбец матрицы A содержит хотя бы один ненулевой элемент — одно из ограничений в этом методе решения.

Утверждение 3. *Если для набора векторов $\widetilde{z}_t,\ t \in \{1,...,T\}$, являющегося допустимым решением исходной задачи, выполняются неравенства $\varepsilon_t \leqslant \widetilde{z}_t,\ t \in \{1,...,T\}$, где ε_t удовлетворяют ограничениям (12), то существует набор $z_t^*,\ t \in \{1,...,T\}$, являющийся допустимым решением задачи (11), такой, что $z_t^* = \widetilde{z}_t - \varepsilon_t$.*

Доказательство. Итак, предположим, что неравенства $\varepsilon_t \leqslant \widetilde{z}_t,\ t \in \{1,...,T\}$ выполняются. Из ограничений исходной задачи:

$$(\widetilde{z}_{t+1} - \varepsilon_{t+1})A = \widetilde{z}_{t+1}A - \varepsilon_{t+1}A \leqslant \widetilde{z}_tB - \varepsilon_{t+1}A,\ t \in \{1,...,T-1\}.$$

Тогда из этого неравенства и ограничений (12):

$$-\varepsilon_{t+1}A \leqslant -(\varepsilon_t + \{1,1,...,1\})B,$$

$$\widetilde{z}_tB - \varepsilon_{t+1}A \leqslant \widetilde{z}_tB - (\varepsilon_t + \{1,1,...,1\})B,$$

$$(\widetilde{z}_{t+1} - \varepsilon_{t+1})A \leqslant (\widetilde{z}_t - \varepsilon_t - \{1,1,...,1\})B.$$

Заменив в последнем выражении $\widetilde{z}_t - \varepsilon_t$ на z_t^*, получим

$$z_{t+1}^* A \leqslant (z_t^* - \{1, 1, ..., 1\})B,\ t \in \{1, ..., T-1\}.$$

а поскольку $\varepsilon_t \leqslant \widetilde{z}_t$, то

$$z_t^* \geqslant \{0, 0, ..., 0\},\ t \in \{1, ..., T\}.$$

Известно, что $\widetilde{z}_1 A \leqslant y_0$, поэтому

$$z_1^* A = (\widetilde{z}_1 - \varepsilon_1)A \leqslant y_0.$$

Таким образом, $z_t^*,\ t \in \{1, ..., T\}$ удовлетворяют ограничениям задачи (11), а, значит, являются допустимым решением. □

Следствие 3.1. *Существование допустимых решений исходной задачи, для которых выполняется* $\varepsilon_t \leqslant \widetilde{z}_t,\ t \in \{1, ..., T\}$ *является достаточным условием существования допустимых решений задачи* (11).

Следствие 3.2. *Множество допустимых решений задачи* (11) *является* ε*-сетью для множества допустимых решений исходной задачи, для которых выполняется* $\varepsilon_t \leqslant \widetilde{z}_t,\ t \in \{1, ..., T\}$.

С помощью следующих утверждений найдем взаимосвязь между ε_t и оценкой погрешности метода. Для начала рассмотрим целевую функцию исходной задачи. Она имеет вид $c(z_T B)^T$ в случае терминальной максимизации и $c\left(k_0 y_0 + \sum_{i=1}^{T}(k_i(z_i B)^T - k_{i-1}(z_i A)^T)\right)$ в случае интегральной. Поскольку A, B, c, $k_i, i = \overline{0, T}$ заданы, то обе эти целевые функции можно выразить как линейную комбинацию компонент векторов $z_i,\ t \in \{1, ..., T\}$ (плюс константа в случае интегральной максимизации). Поэтому, если ввести вектор $z = \{z_t\}_{t=1}^T$, в обоих случаях целевую функцию можно выразить как

$$vz^T + const, \tag{13}$$

где v — вектор размерности Tm. Т.е. в обеих задачах целевая функция линейно зависит от векторов интенсивностей.

Утверждение 4. *Если $X^* \in X$ является ε-сетью $\widetilde{X} \in X$, то для функции f, определенной на X, выполняется неравенство:*

$$\max_{x \in X^*} f(x) \geqslant \max_{x \in \widetilde{X}} f(x) - \omega_f(\varepsilon),$$

где ω_f — модуль непрерывности f.

Доказательство. По определению модуля непрерывности:

$$\omega_f(\varepsilon) = \sup\{|f(x_1) - f(x_2)| : x_1, x_2 \in X, |x_1 - x_2| \leqslant \varepsilon\}$$

Пусть максимум по $\widetilde{X}$ достигается в $\widetilde{x}$, тогда из определения ε-сети существует $x^* \in X^*$, для которого выполняется $|\widetilde{x} - x^*| \leqslant \varepsilon$. Поэтому

$$f(\widetilde{x}) - \omega_f(\varepsilon) \leqslant f(x^*) \leqslant f(\widetilde{x}) + \omega_f(\varepsilon).$$

Поскольку $\max\limits_{x \in X^*} f(x) \geqslant f(x^*)$, из левой части двойного неравенства следует:

$$\max_{x \in X^*} f(x) \geqslant \max_{x \in \widetilde{X}} f(x) - \omega_f(\varepsilon).$$

□

Следствие 4.1. *Если существует вектор $\varepsilon = \{\varepsilon_t\}_{t=1}^T$, для которого выполняются ограничения (12), а для оптимального решения $\widetilde{z} = \{\widetilde{z}_t\}_{t=1}^T$ исходной задачи выполняется $\varepsilon \leqslant \widetilde{z}$, то*

$$v(z^*)^T \geqslant v(\widetilde{z} - \varepsilon)^T,$$

где $z^* = \{z_t^*\}_{t=1}^T$ — оптимальное решение задачи (11), v — вектор коэффициентов при переменных в целевой функции из (13).

Из ограничений (12) видно, что ε_t склонен расти с увеличением t. Найдем оценку минимально возможных ε_t, $t \in \{1, ..., T\}$ сверху, обозначив ее как $\overline{\varepsilon_t}$. На первом шаге оценка очевидна: $\varepsilon_1 = \overline{\varepsilon_1} = \{0, 0, ..., 0\}$.

В неравенстве $\varepsilon_{t+1} A \geqslant (\varepsilon_t + \{1, 1, ..., 1\})B$ мы не можем выразить ε_{t+1}, поскольку A^{-1} может не существовать. Поэтому для последующих шагов

t мы будем искать оценку компонент вектора ε_t. Пусть $e_t^i, i = \overline{1,m}$ — компоненты, а $\overline{e_t}$ — их верхняя оценка. Распишем неравенство подробнее:

$$\begin{cases} e_{t+1}^1 a_{11} + ... + e_{t+1}^m a_{m1} \geqslant (e_t^1 + 1)b_{11} + ... + (e_t^m + 1)b_{m1}, \\ e_{t+1}^1 a_{12} + ... + e_{t+1}^m a_{m2} \geqslant (e_t^1 + 1)b_{12} + ... + (e_t^m + 1)b_{m2}, \\ ... \\ e_{t+1}^1 a_{1n} + ... + e_{t+1}^m a_{mn} \geqslant (e_t^1 + 1)b_{1n} + ... + (e_t^m + 1)b_{mn}. \end{cases} \tag{14}$$

Предположим, что мы знаем $\overline{e_t}$. Возьмем такой $\overline{e_{t+1}}$, чтобы для каждого $i = \overline{1,n}$ выполнялось

$$\overline{e_{t+1}} \sum_{j=1}^{m} a_{ji} \geqslant (\overline{e_t} + 1) \sum_{j=1}^{m} b_{ji}.$$

$\sum_{j=1}^{m} a_{ji}$ выражает общее количество продукта i, затрачиваемое во всех технологических процессах — неравенство этой суммы нулю следует из допущения, которое мы сделали для существования векторов из (12) (о том, что каждый столбец матрицы A содержит хотя бы один ненулевой элемент). Т. к. сумма не равна нулю, на нее можно разделить:

$$\overline{e_{t+1}} \geqslant \frac{(\overline{e_t} + 1) \sum\limits_{j=1}^{m} b_{ji}}{\sum\limits_{j=1}^{m} a_{ji}}.$$

Для этого будет достаточно выполнения следующего равенства:

$$\overline{e_{t+1}} = \max_{i=\overline{1,n}} \frac{(\overline{e_t} + 1) \sum\limits_{j=1}^{m} b_{ji}}{\sum\limits_{j=1}^{m} a_{ji}};$$

$$\overline{e_{t+1}} = (\overline{e_t} + 1) \max_{i=\overline{1,n}} \frac{\sum\limits_{j=1}^{m} b_{ji}}{\sum\limits_{j=1}^{m} a_{ji}}.$$

Из этой рекуррентной формулы, зная значение $\overline{\varepsilon_1}$, мы можем выразить $\overline{e_t},\ t \in \{1, ..., T\}$:

$$\overline{e_t} = \sum_{k=1}^{t-1} \left(\max_{i=\overline{1,n}} \frac{\sum\limits_{j=1}^{m} b_{ji}}{\sum\limits_{j=1}^{m} a_{ji}} \right)^k.$$

Введем замену $q = \max\limits_{i=\overline{1,n}} (\sum\limits_{j=1}^{m} b_{ji} / \sum\limits_{j=1}^{m} a_{ji})$ и свернем выражение по формуле суммы геометрической прогрессии:

$$\overline{e_t} = \begin{cases} \frac{q(q^{t-1}-1)}{q-1},\ \text{если } q \neq 1; \\ t-1,\ \text{если } q = 1. \end{cases}$$

Оценку вектора можно записать как вектор из оценок его компонент:

$$\overline{\varepsilon_t} = \{\overline{e_t}, \overline{e_t}, ..., \overline{e_t}\}.$$

Для растущей экономики $q > 1$, поскольку хотя бы один продукт суммарно всеми процессами производится больше, чем потребляется. Поэтому ε_t (а, следовательно, и погрешность метода) экспоненциально растет по t. Таким образом, этот метод пригоден для решения только тех задач, у которых оптимальное значение вектора интенсивностей по t растет экспоненциально.

Подытожим суть решения методом сведения к непрерывной модели:

Теорема 5. *Пусть в исходной задаче каждый столбец матрицы A содержит хотя бы один ненулевой элемент и оптимальное решение $\widetilde{z}_t,\ t = \overline{1,T}$ удовлетворяет неравенствам $\widetilde{z}_t \geqslant \{\overline{e_t}, \overline{e_t}, ..., \overline{e_t}\},\ t = \overline{1,T}$, где*

$$\overline{e_t} = \begin{cases} \frac{q(q^{t-1}-1)}{q-1},\ \textit{если } q \neq 1; \\ t-1,\ \textit{если } q = 1; \end{cases}$$

$$q = \max_{i=\overline{1,n}} \frac{\sum\limits_{j=1}^{m} b_{ji}}{\sum\limits_{j=1}^{m} a_{ji}}$$

Выразим целевую функцию исходной задачи в виде $\sum\limits_{t=1}^{T} v_t z_t^T + const.$

Для задачи с такой же целевой функцией и ограничениями вида

$$\begin{cases} z_1 A \leqslant y_0, \\ z_2 A \leqslant (z_1 - \{1, 1, ..., 1\})B, \\ ... \\ z_T A \leqslant (z_{T-1} - \{1, 1, ..., 1\})B, \\ z_t \geqslant \{0, 0, ..., 0\},\ t = \overline{1, T} \end{cases}$$

существует оптимальное решение $z_t^*,\ t = \overline{1, T}$*, при этом имеет место следующее неравенство:*

$$\sum_{t=1}^{T} v_t (z_t^*)^T \geqslant \sum_{t=1}^{T} v_t (\widetilde{z}_t - \{\overline{e_t}, \overline{e_t}, ..., \overline{e_t}\})^T.$$

Округление вниз $\lfloor z_t^* \rfloor,\ t = \overline{1, T}$ *даст допустимое решение исходной задачи. Таким образом,* $\lfloor z_t^* \rfloor,\ t = \overline{1, T}$ *можно считать оптимальным решением исходной задачи с погрешностью, не превышающей* $\sum\limits_{t=1}^{T} v_t (\{\overline{e_t}, \overline{e_t}, ..., \overline{e_t}\}^T + \{1, 1, ..., 1\}^T)$.

Одно из самых главных преимуществ этого метода — высокая временная эффективность. Согласно [7], битовая сложность (учитывающая трудоемкость операций над числами в компьютерном представлении) решения задачи линейного программирования с Tm переменных и Tn неравенств методом внутренней точки Кармаркара при заданной цифровой точности приближенного решения составляет

$$O\left(T^{3,5} \max(n^{3,5}, m^{3,5})[l + \log(T \max(n, m))]\right),$$

где l — число разрядов, отводимых на запись числа во входных данных. Необходимость составления необходимой нам ЗЛП и округление оптимального решения перед вычислением целевой функции не изменяют эту оцен-

ку. Таким образом, мы можем найти приближенное решение нашей задачи за полиномиальное время.

1.2. Метод динамического программирования

Теперь рассмотрим подход к решению терминальной задачи (8), для которого не требуется сводить задачу к статической, — динамическое программирование. В этом параграфе мы будем строить метод, дающий точное решение.

Поскольку задача является целочисленной, она дискретна и, с учетом ограничений, конечна — поэтому мы можем решить её методом полного перебора всех элементов: последовательно строить все цепочки векторов интенсивностей, удовлетворяющие ограничениям, а затем из всех цепочек выбирать ту, которая максимизирует целевую функцию. Однако очевидно, что такой метод будет крайне неэффективен и потому непригоден для практического использования. Возникает потребность в сокращении количества перебираемых на каждом шаге вариантов, а также отсечения цепочек, заведомо не ведущих к оптимальному решению. Для этого можно использовать понятие Парето-оптимальности.

Определение. Вектор $x \in X$ *Парето-оптимален на множестве* X, если не существует такого $y \in X$, что $y > x$. Множество Парето-оптимальных на X векторов обозначим $P(X)$.

Следующие утверждения касательно Парето-оптимальных векторов интенсивностей лягут в основу описываемого в этом разделе метода для терминальной задачи.

Утверждение 6. *Существует оптимальное решение терминальной задачи, состоящее из векторов интенсивностей, каждый из которых Парето-оптимален на множестве достижимых значений на соответствующем шаге.*

Доказательство. Число достижимых значений вектора z_T конечно, а значит, существует как минимум одно Парето-оптимальное значение. Целевую функцию можно выразить как линейную комбинацию компонентов z_T с неотрицательными коэффициентами, поэтому максимизирующим целевую функцию будет какой-то из Парето-оптимальных векторов. Обозначим этот Парето-оптимальный вектор как $\widetilde{z}_T$.

Теперь, начиная от T и двигаясь в обратном направлении, строим оптимальное решение $\widetilde{z}_t,\ t = \overline{1,T}$.

Пусть достижимый $\widetilde{z}_t$ является Парето-оптимальным — докажем, что среди достижимых значений z_{t-1} существует Парето-оптимальный вектор, который можно включить в наше решение. $\widetilde{z}_t$ является достижимым, поэтому существует некоторый z^*_{t-1}, для которого $\widetilde{z}_t A \leqslant z^*_{t-1} B$. Если он Парето-оптимален, то включим его в наше решение: $\widetilde{z}_{t-1} = z^*_{t-1}$. Иначе в качестве $\widetilde{z}_{t-1}$ возьмем доминирующий его Парето-оптимальный вектор z^{**}_{t-1}: мы можем это сделать, т. к. из $z^{**}_{t-1} > z^*_{t-1}$ следует, что $\widetilde{z}_t A \leqslant z^{**}_{t-1} B$.

Таким образом, по индукции мы можем построить оптимальное решение $\widetilde{z}_t,\ t = \overline{1,T}$, состоящее из векторов, Парето-оптимальных на множестве достижимых значений на соответсвующих шагах. □

Утверждение 7. *Если* $X = X_1 \cup X_2 ... \cup X_k$, *то* $P(X) = P(P(X_1) \cup P(X_2) ... \cup P(X_k))$

Доказательство. Если $x \in X$ принадлежит к $P(X)$, то не существует $y \in X$, который будет больше x. Существует X_i, $i = \overline{1,k}$, что $x \in X_i$, притом $y > x$ не существует ни в одном X_i, поэтому $x \in P(X_1) \cup P(X_2) ... \cup P(X_k)$. $P(X_1) \cup P(X_2) ... \cup P(X_k)$ является подмножеством X, и поэтому $y > x$ также не существует в $P(X_1) \cup P(X_2) ... \cup P(X_k)$, поэтому $x \in P(P(X_1) \cup P(X_2) ... \cup P(X_k))$. Таким образом,

$$P(X) \subset P(P(X_1) \cup P(X_2) ... \cup P(X_k)).$$

Предположим, что существует $x \in P(P(X_1) \cup P(X_2)... \cup P(X_k))$, который не принадлежит $P(X)$. Тогда существует такой $y \in X$, что $y > x$. Так как x находится среди Парето-оптимальных векторов множества $P(X_1) \cup P(X_2)... \cup P(X_k)$, то $y \notin P(X_1) \cup P(X_2)... \cup P(X_k)$. Но поскольку $y \in X$, то y принадлежит хотя бы некоторому X_i, $i = \overline{1,k}$. Поэтому в множестве существует такой $z \in P(X_1) \cup P(X_2)... \cup P(X_k)$, что $z > y$. Тогда z также будет больше x, и мы получаем, что в множестве $P(X_1) \cup P(X_2)... \cup P(X_k)$ существует вектор, больше Парето-оптимального, что противоречит определению Парето-оптимального вектора. Значит, наше предположение не верно и

$$P(P(X_1) \cup P(X_2)... \cup P(X_k)) \subset P(X).$$

Таким образом, $P(X) = P(P(X_1) \cup P(X_2)... \cup P(X_k))$. □

Из утверждения (6) следует, что для поиска оптимального решения достаточно рассматривать Парето-оптимальные векторы интенсивностей. Возникает новая задача: необходимо наиболее эффективно определять такие векторы.

Допустим, что мы знаем векторы, Парето-оптимальные на множестве допустимых значений на шаге $t-1$, а нужно узнать для шага t. Разобьем задачу на подзадачи: для начала найдем векторы, Парето-оптимальные среди допустимых для каждого конкретного Парето-оптимального z_{t-1}, объединим их в один общий список и выберем Парето-оптимальные среди них. То, что таким образом будут найдены все векторы, которые являются Парето-оптимальными на множестве допустимых на шаге t, следует из утверждения (7) и доказательства утверждения (6).

При заданном z_{t-1} будет генерировать векторы z_t в порядке, обратном лексикографическому, т. к. это будет эффективнее по сравнению с обычным перебором от нулевого вектора. Поскольку лексикографический порядок расширяет отношение $>$ на векторах (которое является отношением

частичного порядка), то при переборе можно сразу начать составлять окончательный список Парето-оптимальных векторов — если сгенерированный вектор не доминируется ни одним вектором из уже присутствующих в списке, то он Парето-оптимален и его нужно в этот список добавить. К тому же, после генерации вектора $\{z_t^1, ..., z_t^{m-1}, z_t^m\}$ можно не рассматривать векторы $\{z_t^1, ..., z_t^{m-1}, x\}$, где x принимает значения от $z_t^m - 1$ до 0, поскольку они будут заведомо меньше уже рассмотренного вектора.

Исходя из рассуждений выше, алгоритм нахождения оптимального решения терминальной задачи оптимизации методом динамического программирования выглядит следующим образом:

1. Задаем $t = 1$. В первый момент времени у нас нет векторов z_{t-1}, поэтому вместо $z_{t-1}B$ на следующем шаге используем вектор y_0;

2. Генерируем вектор z_t, наибольший в смысле лексикографического порядка, удовлетворяющий ограничению $z_tA \leqslant z_{t-1}B$: устанавливаем элементы, начиная с первого, в максимально допустимое значение (с учетом уже установленных элементов). Добавляем его в список Парето-оптимальных среди допустимых для текущего z_{t-1};

3. В последнем сгенерированном векторе находим среди индексов с 1 по $m - 1$ наибольший индекс, соответствующий ненулевому элементу: если такого индекса нет, то переходим к шагу 5, иначе же обозначим этот индекс как i и переходим к шагу 4;

4. Генерируем новый вектор на основе предыдущего: уменьшаем z_t^i на 1 и задаем новые значения элементам с индексами больше чем i: начиная с i+1-го, устанавливаем элементы в максимально допустимое значение (с учетом уже установленных элементов). Проверяем, доминируется ли он каким-нибудь вектором из списка Парето-оптимальных среди допустимых для текущего z_{t-1}: если нет, то добавляем его в

этот список. Переходим к шагу 3;

5. Берем следующий вектор z_{t-1} из списка Парето-оптимальных в момент $t-1$ и переходим к шагу 2. Если таких векторов больше не осталось (или $t=1$ и списка для $t-1$ нет), то переходим к шагу 6;

6. Собираем списки Парето-оптимальных векторов среди допустимых для каждого z_{t-1} в один список, при этом для каждого вектора сохраняем ссылку на z_{t-1}, для которого он является допустимым (при конфликте выбираем любой). При слиянии сортируем список в порядке, обратном лексикографическому;

7. Используем алгоритм типа «решета Эратосфена», чтобы в полученном списке отобрать Парето-оптимальные: выбираем в качестве «опорного» первый вектор из списка и удаляем все последующие вектора, которые меньше его. Затем в качестве «опорного» выбираем следующий вектор в списке и снова удаляем все последующие вектора, которые меньше его, и так далее, пока новый «опорный» не окажется последним в списке;

8. Если $t=T$, переходим к шагу 9. Иначе увеличиваем t на 1, полученный на предыдущем шаге список есть не что иное как список Парето-оптимальных векторов в момент $t-1$, берем из него первый вектор z_{t-1} переходим к шагу 2;

9. Среди векторов z_T в списке найдем тот, что максимизирует целевую функцию (при конфликте выбираем любой). По ссылке найдем вектор z_{T-1}, для которого этот максимизирующий является допустимым, затем аналогично по ссылке находим вектор z_{T-2} и так далее до z_1. Найденные вектора $z_t,\ t=\overline{1,T}$ являются искомым оптимальным решением.

Представленный алгоритм реализует «динамическое программирование вперед» — для данной задачи такой подход предпочтительней «динамического программирования назад», поскольку начальное состояние системы (вектор начальных запасов y_0) задан единственным образом, в отличие от конечного состояния, характеризуемого для растущей модели довольно большим множеством допустимых векторов интенсивностей на последнем шаге.

Оценим временную сложность метода. Пусть $T > 1$. При $t = 1$ перебор векторов осуществляется за $O(L_1^{m-1}R)$, где R — максимальное число ненулевых элементов в строке матрицы A (т. к. матрица разрежена, то $R << n$), L_1 — максимальное допустимое значение элемента вектора интенсивностей на первом шаге, равное $\max\limits_i \min\limits_j y_0^j/a_{ij}$.

При t от 2 до T мы для начала вычисляем $z_{t-1}B$ за $O(Cn)$, где C — максимальное число ненулевых элементов в столбце матрицы B (т. к. матрица разрежена, то $C << m$). Затем, как и на первом шаге, перебираем векторы, но уже для каждого Парето-оптимального вектора на предыдущем шаге, поэтому перебор осуществляется за $O(L_t^{m-1}RP_{t-1})$, где P_{t-1} — количество Парето-оптимальных векторов на шаге $t-1$, а L_t — максимальное допустимое значение элемента вектора интенсивностей на шаге t. Далее полученные векторы объединяем в один список, при этом время на объединение линейно зависит от количества векторов во всех списках, это количество можно грубо оценить сверху общим числом перебираемых векторов, откуда получаем оценку времени слияния $O(L_t^{m-1}P_{t-1})$. Время нахождения Парето-оптимальных векторов из общего списка можно оценить как $O(L_t^{m-1}P_{t-1}P_t)$, поскольку точно будет меньше времени сравнения каждого вектора общего списка с каждым Парето-оптимальным.

После получения списка Парето-оптимальных векторов на шаге T мы ищем вектор, максимизирующий целевую функцию за $O(CnP_T)$, а потом за $O(T)$ находим оптимальное решение.

Пусть $L = \max\limits_t L_t$, $P = \max\limits_t P_t$, тогда общая оценка времени исполнения будет иметь вид:

$$O(L^{m-1}R + (T-1)(Cn + L^{m-1}RP + L^{m-1}P + L^{m-1}P^2) + CnP + T) =$$

$$= O((T-1)L^{m-1}P(R+P) + Cn(T+P)).$$

Из-за несколько грубых оценок отдельных частей алгоритма и фигурирующих величин (а также того факта, что L и P трудно оценить до начала работы алгоритма), полученная оценка не годится для вычисления примерного времени нахождения решения реализацией алгоритма, однако отражает, например, экспоненциальную зависимость от размера входных данных. Несмотря на то, что формально из оценки следует линейная зависимость от T, для растущих моделей вектора интенсивностей с каждым шагом растут в геометрической прогрессии, что приводит к быстрому росту L, который в оценке фигурирует как основание степени, поэтому для больших T выполнение алгоритма может занять довольно большое количество времени.

Глава 2. Реализация методов и их сравнительный анализ

В рамках данной выпускной квалификационной работы осуществлена программная реализация описанных в предыдущем разделе методов на языке C++. Как язык программирования высокого уровня, C++ поддерживает необходимый уровень абстракции для удобной платформонезависимой разработки алгоритмов рассматриваемых методов, при этом предоставляя большое количество стандартных структур данных, дополнительных подключаемых библиотек алгебры матриц и линейного программирования, а также обеспечивая высокое время исполнения скомпилированной программы.

2.1. Программная реализация

Этот параграф посвящен описанию структуры программного кода, реализованных методов и использованной подключаемой библиотеки. С частичным листингом кода можно ознакомиться в Приложении.

В основе реализации лежит класс `NMProblem`, предназначенный для создания объектов типа задачи оптимизации целочисленной модели Неймана. Непосредственно методы оптимизации реализованы public-методами `solveByContinualizationMethod()` и `solveByDynamicProgramming()`, при этом последняя задействует ряд private-методов.

Для решения задачи линейного программирования в методе континуализации используются возможности библиотеки Clp (COIN-OR linear programming). Это проект с открытым исходным кодом для решения задач линейного программирования, написанный на C++. Библиотека поддерживает работу с разреженными матрицами (что является крайне полезным свойством, поскольку матрицы A и B в модели Неймана по своей

природе являются разреженными), причем библиотека содержит отдельный класс `CoinPackedMatrix` для разреженных матриц [8], что позволяет использовать ее в реализации не только метода континуализации, но и метода динамического програмирования.

Таким образом, суть работы `solveByContinualizationMethod()` заключается в преобразовании свойств объекта класса `NMProblem` в формат статической задачи линейного программирования, определенной в параграфе 1.1, решении ее при помощи средств библиотеки Clp и сохранении значения оптимального решения и целевой функции. Для решения ЗЛП используем библиотечный класс `ClpPredictorCorrector`, применяющий метод прогноза и коррекции Мехротра — одну из реализаций метода внутренней точки, широко используемую на практике. В среднем для нахождения решения ему требуется примерно на 35-50% меньше шагов, чем алгоритму Кармаркара [9].

Метод `solveByDynamicProgramming()` реализует алгоритм, приведенный в параграфе 1.2. Для хранения Парето-оптимальных на шаге векторов используется список из объектов специальной структуры `VectorAndParent` — в поле `elem` этой структуры хранится указатель на массив элементов вектора, а в поле `parent` хранится ссылка на Парето-оптимальный на предыдущем шаге вектор, для которого вектор со значениями из `elem` является допустимым. В своей работе метод использует другие методы: `fillParetoOptimalVectorsList()` — для составления списка Парето-оптимальных векторов из допустимых для конкретного значения вектора интенсивностей на предыдущем шаге, `maximizeVector()` — для построения максимального в лексикографическом смысле вектора интенсивностей с учетом матрицы коэффициентов и вектора ограничений, `addIfNotDominated()` — для добавления в список недоминируемых векторов, `transferToGeneralList()` — для переноса векторов из списка для конкретного вектора на предыдущем шаге в общий список Парето-оптимальых

векторов на шаге.

2.2. Сравнительный анализ

Благодаря программной реализации мы можем провести более детальный анализ методов оптимизации, применив их к конкретным задачам. Для этого сгенерируем наборы задач терминальной оптимизации с различными параметрами. В качестве параметров будут выступать такие величины как размерность (величины m и n) и число рассматриваемых промежутков времени T. Для построения графиков по рассчитанным для сгенерированных задач данным будет использовать утилиту gnuplot [10].

Перечислим параметры, по которым будем сравнивать работу методов. Во-первых, получаемое методами значение целевой функции — не только потому что ее максимизация является целью оптимизационной задачи, но и потому что она является линейной комбинацией значений элементов вектора интенсивностей на шаге T. Во-вторых, время исполнения — таким образом, можно проверить оценки временной эффективности на практике.

Немаловажным параметром задачи, который также стоит отметить, является рост модели — это понятие определяет, как увеличивается оптимальный вектор интенсивностей с течением времени. Эта величина непосредственно влияет на финальный вектор интенсивностей, а следовательно, и на значение целевой функции. При одинаковых для генерируемых задач порядках значений элементов соответствующих матриц и векторов (A, B, y_0, c) разница в оптимальных значениях целевой функции будет выражать разницу роста моделей. Поэтому оптимальное значение целевой функции в анализе будет фигурировать не только как зависимый параметр, но и как независимый.

Итак, на рисунке 1 мы видим график зависимости выдаваемых методами значений целевой функции от размера входной задачи (другие пара-

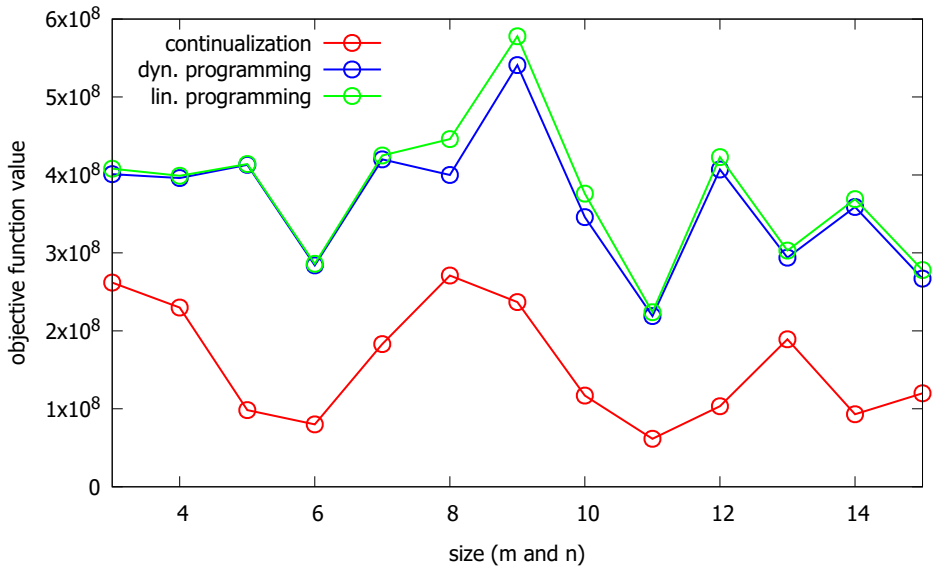

Рис. 1: График зависимости значений целевой функции, полученных методом динамического программирования, методом континуализации и сведением к ЗЛП отказом от целочисленных ограничений, от размерности задачи (величин m, n). $T = 5$, оптимальное значение целевых функций для задач имеет порядок 10^8

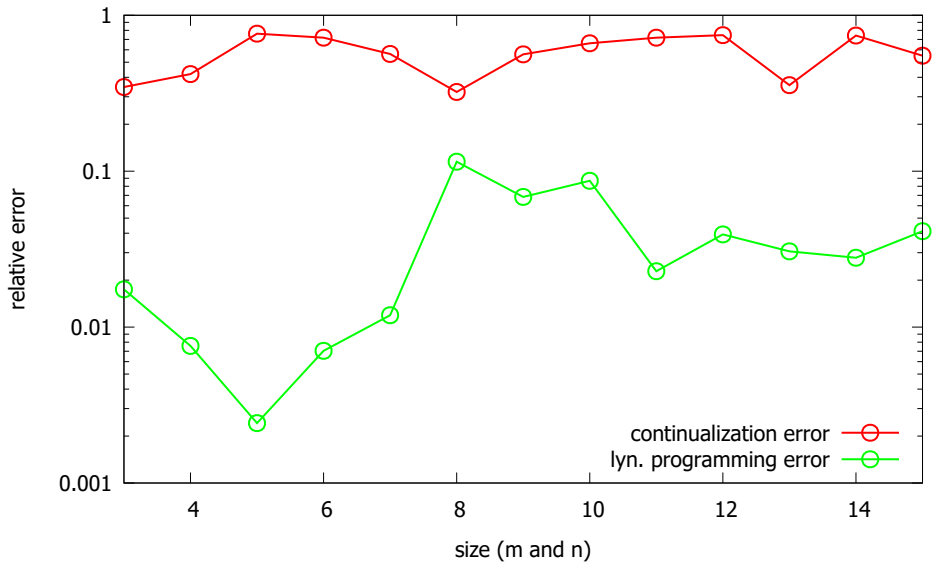

Рис. 2: График зависимости относительной погрешности значений целевой функции, полученных методом континуализации и сведением к ЗЛП отказом от целочисленных ограничений, от размерности задачи (величин m, n). $T = 5$, оптимальное значение целевых функций для задач имеет порядок 10^8

метры зафиксированы: $T = 5$, а оптимальное значение целевой функции имеет порядок 10^8), каждая точка которого представляет усредненные для 10 задач значения. Три линии на графике обозначают значения целевой функции, полученные методом континуализации, методом динамического программирования и сведением к задаче линейного программирования путем отбрасывания ограничения на целочисленность.

Значение, полученное методом динамического программирования, является точным (вернее, имеющим погрешность, не превышающую ошибку при вычислениях чисел с плавающей точкой), оно и использовалось для выбора задач одного порядка, поэтому на графике нас в первую очередь интересуют значения, полученные двумя другими методами, а точнее насколько эти значения отличаются от значений, полученных методом динамического программирования.

Значения для ЗЛП без ограничений на целочисленность, как видно на графике, могут превышать максимальное значение целевой функции — это происходит из-за того, что вектора интенсивностей на последнем шаге могут принимать недопустимые для исходной задачи значения, и именно поэтому мы не рассматривали этот метод. Однако эти данные можно рассматривать как верхнюю оценку оптимального значения целевой функции для нашей задачи.

Растущая в геометрической прогрессии ошибка метода континуализации приводит к тому, что погрешность значения целевой функции для этого метода может достигать тех же порядков, что и само значение целевой функции. Тем не менее, эти данные можно использовать как нижнюю оценку, и к тому же выводимые векторы интенсивностей будут принадлежать допустимому множеству.

Рассмотрим также график на рисунке 2, где на оси ординат в логарифмическом масштабе теперь отображена относительная погрешность приближенных методов. Мы видим слабую зависимость погрешностей от раз-

мерности — куда больше на значение ошибки влияет само оптимальное значение целевой функции. Тем не менее, можно заметить небольшой рост погрешности ЗЛП при росте размерности.

Теперь рассмотрим график на рисунке 3. На оси абсцисс все так же расположена размерность, а на оси ординат в логарифмическом масштабе теперь время работы методов континуализации и динамического программирования. Здесь зависимость от размерности более ярко выражена, особенно для метода континуализации — график подтверждает степенную зависимость от размера, пусть и с явно меньшим показателем. Время выполнения метода динамического программирования на порядки выше, но зависимость от размерности слабее — трудно сказать, степенная она или линейная. Указанная в оценке экспоненциальная зависимость написана в расчете на почти полный перебор допустимых векторов интенсивностей на каждом шаге при поиске Парето-оптимальных, чего в среднем случае не происходит.

Теперь исследуем зависимость от T. Сгенерируем 10 задач с $m = n = 5$

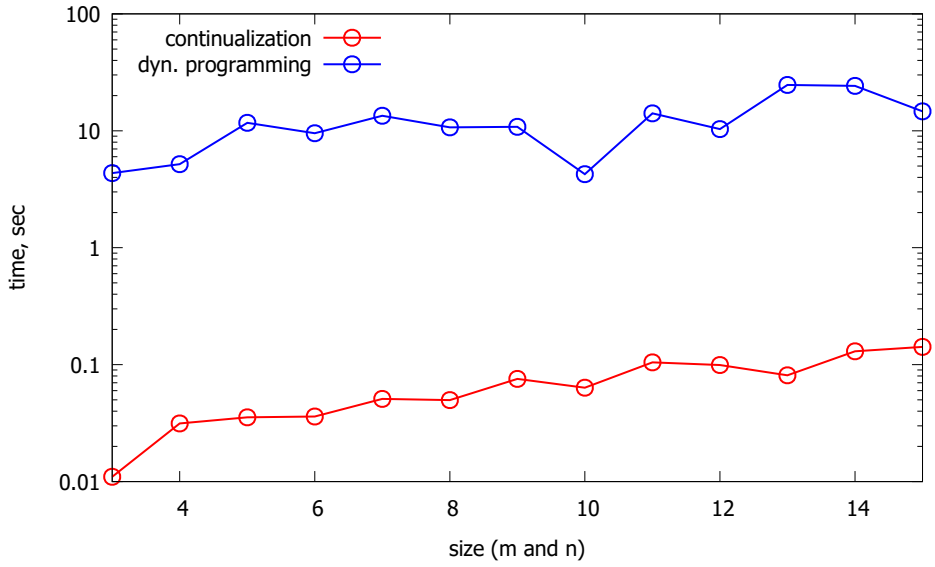

Рис. 3: График зависимости времени выполнения методов континуализации и динамического программирования от размерности задачи (величин m, n). $T = 5$, оптимальное значение целевых функций для задач имеет порядок 10^8

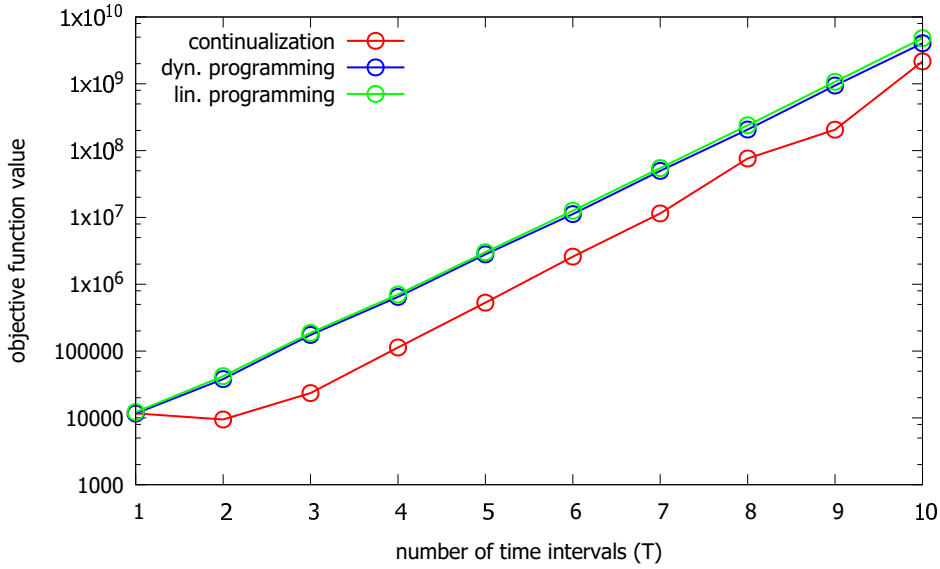

Рис. 4: График зависимости значений целевой функции, полученных методом динамического программирования, методом континуализации и сведением к ЗЛП отказом от целочисленных ограничений, от числа рассматриваемых временных промежутков (величины T). $m = n = 5$, оптимальное значение целевых функций для задач имеет порядок 10^6 для $T = 5$

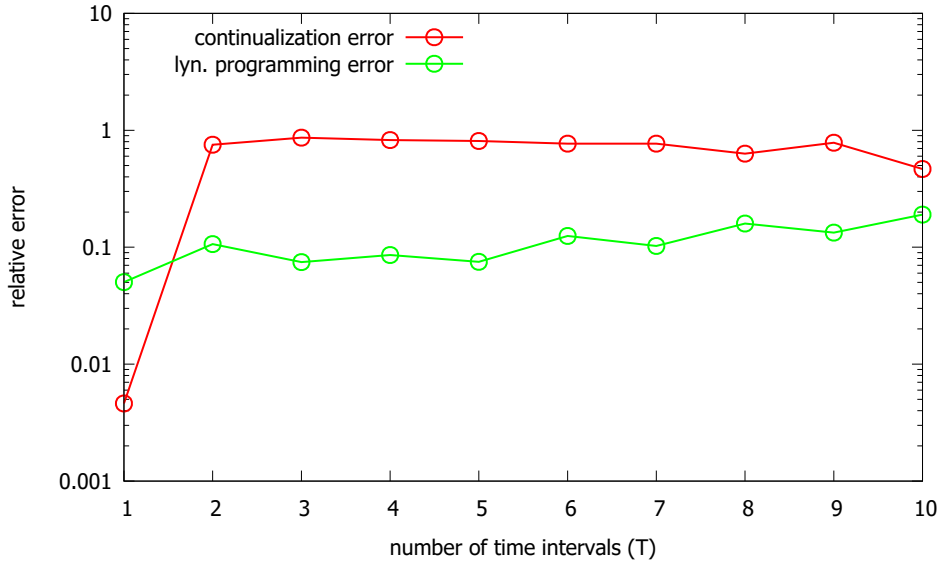

Рис. 5: График зависимости относительной погрешности значений целевой функции, полученных методом континуализации и сведением к ЗЛП отказом от целочисленных ограничений, от числа рассматриваемых временных промежутков (величины T). $m = n = 5$, оптимальное значение целевых функций для задач имеет порядок 10^6 для $T = 5$

и оптимальными значениями целевых функций при $T = 5$ порядка 10^6, и будем их решать, изменяя T. Среднее значений целевых функций, выдаваемых методами значений целевых функций, расположена на оси ординат рисунка 4. Видно, что для рассматриваемых задач ярко выражен экспоненциальный рост значений целевой функции с увеличением T, такой же рост наблюдается и для значений, получаемых от всех методов.

При этом относительная погрешность, изображенная на графике на рисунке 5, изменяется по-разному. Для сведения к ЗЛП отбрасыванием ограничений на целочисленность медленно нелинейно растет (но этот рост может быть вызван ростом значений целевой функции, поскольку на предыдущем графике для относительной погрешности этот метод показывал зависимость от размерности). Относительная погрешность метода континуализации начинает от близкого к нулю значения при $T = 1$ (методическая погрешность в этом случае равна 0, значит, это погрешность вычислений и округлений), а затем становится почти постоянной — видимо, абсолютная погрешность и значение целевой функции растут примерно с одной скоростью. Важно лишь отметить, что такой баланс соблюдается не для всех задач — вектора интенсивностей на оптимальной траектории должны увеличиваться в геометрической прогрессии и достаточно быстро.

Следующий график на логарифмической шкале демонстрирует зависимость времени выполнения методов от числа рассматриваемых временных промежутков (рисунок 6). Время метода континуализации демонстрирует степенную зависимость, что соответствует оценке. Время метода динамического программирования растет экспоненциально, что не соответствует оценке, которая зависит от T линейно — сказывается рост значения целевой функции, как это было упомянуто в комментарии к оценке.

На многих рассмотренных графиках можно видеть корреляцию между зависимыми величинами и оптимальным значением целевой функции задачи. Рисунок 7 содержит точки с значениями для 200 задач с зафик-

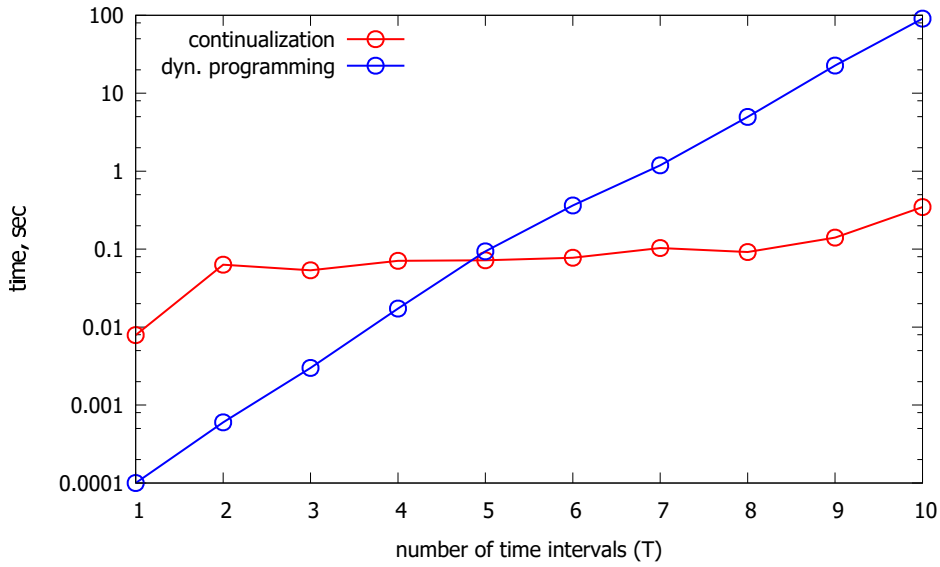

Рис. 6: График зависимости времени выполнения методов континуализации и динамического программирования от числа рассматриваемых временных промежутков (величины T). $m = n = 5$, оптимальное значение целевых функций для задач имеет порядок 10^6 для $T = 5$

сированными параметрами $m = n = 7$, $T = 4$ и без ограничения на порядок значений целевой функции. Ось абсцисс в логарифмическом масштабе отображает значение целевой функции методом динамического программирования, а ординат — значение целевой функции методом континуализации, тоже в логарифмическом мастабе.

Сразу заметно большое количество задач, для которых метод континуализации выдает в качестве значения целевой функции ноль, в то время как метод динамического программирования находит ненулевой оптимум — причем с ростом значения этого оптимума число таких задач уменьшается. Нулевое решение в задаче тоже является допустимым, однако вряд ли применимым в качестве полезной оценки, что говорит о том, что метод континуализации пригоден преимущественно для задач с достаточно быстрорастущим решением. Если вынести на ось ординат относительную погрешность метода континуализации (рисунок 8), то видно, что чем больше оптимальное значение целевой функции, тем меньше относительная погрешность, причем зависимость либо степенная с довольно маленькой

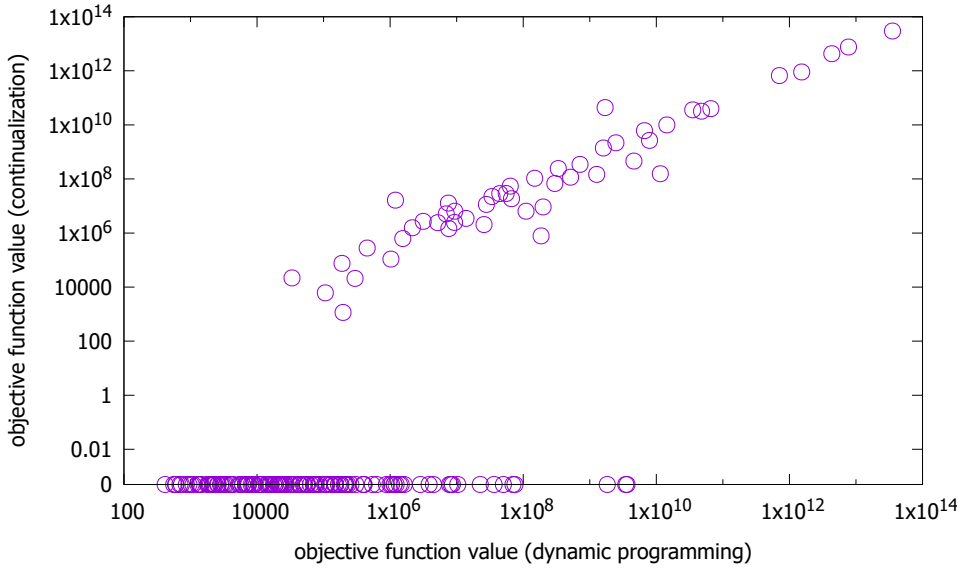

Рис. 7: Значения целевой функции, полученные методом динамического программирования, и значения для тех же задач, полученные методом континуализации. $T = 4$, $m = n = 7$

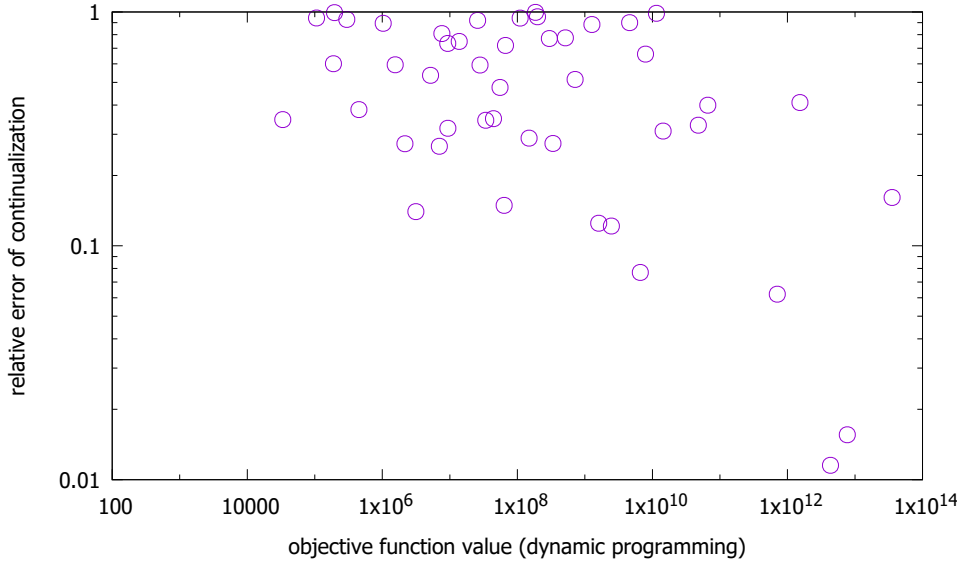

Рис. 8: Значения целевой функции, полученные методом динамического программирования, и относительная погрешность значений для тех же задач, полученные методом континуализации. $T = 4$, $m = n = 7$

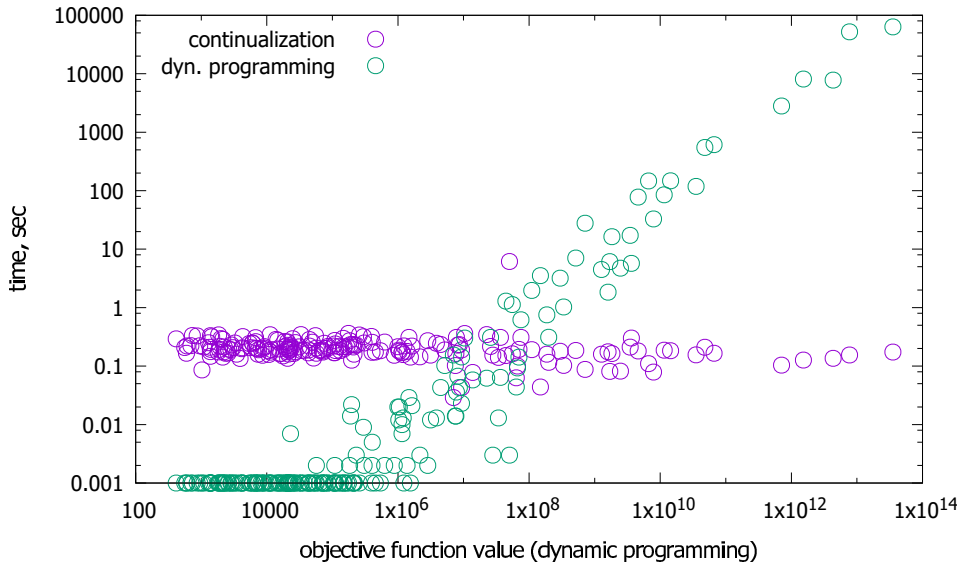

Рис. 9: Значения целевой функции, полученные методом динамического программирования, и время поиска соответсвующих им оптимальных траекторий двумя методами. $T = 4$, $m = n = 7$

отрицательным показателем, либо экспоненциальная с обратным показателем.

Теперь поставим на ось ординат время и отобразим время работы обоих методов 9. В то время как с ростом значения целевой функции время для метода континуализации остается практически неизменным, время для метода динамического программирования оказывается в степенной зависимости (что соответсвует оценке, поскольку для этих задач m зафиксирован) — до определенных значений он оказывается даже быстрее метода континуализации, но затем быстрый рост приводит к тому, что его временная эффективность становится очень низкой.

В результате анализа графиков, можно прийти к следующим выводам: метод динамического программирования довольно эффективен, однако с очень быстрым ростом вектора интенсивностей быстро растет время выполнения метода и поиск решения может занять существенное время. Метод континуализации дает допустимое решение, однако погрешность с течением времени растет в геометрической прогрессии и для недостаточно

быстро растущих моделей получаемое допустимое решение может быть бесполезно даже в качестве нижней оценки (в крайнем случае может давать тривиальное нулевое решение).

Таким образом, эффективность рассмотренных методов сильно зависит от скорости роста модели. Для предварительной оценки роста можно использовать решение задачи линейного программирования без ограничения целочисленности, которое дает верхнюю оценку.

Следует также заметить, что представленная в работе реализация методов представляет простую демонстрацию описанных в работе алгоритмов в програмном коде и может быть оптимизирована, например, с помощью применения многопоточности.

Заключение

В работе знакомая экономистам модель Неймана рассмотрена в новом ключе — как объект целочисленной оптимизационной задачи. Исследованы методы нахождения оптимальной траектории, представляющие различные подходы к решению задачи оптимизации.

Метод континуализации представляет из себя сведение новой задачи к хорошо изученному классу задач линейного программирования. Построена модификация задачи, представляющая из себя задачу линейного программирования, сформулированы и доказаны утверждения о том, что решение этой модификации является допустимым решением исходной задачи, установлены ограничения применения, произведена оценка погрешности и временной сложности.

Динамическое программирование является одним из основных и универсальных подходов к динамической задаче. Второй метод, представленный в работе, применяет этот подход к задаче оптимизации целочисленной модели Неймана. Доказаны утверждения о том, что данный метод дает точное решение оптимизационной задачи, составлен подробный алгоритм нахождения оптимальной траектории, произведена оценка временной сложности.

Для обоих методов написана программная реализация на языке C++ и проведено тестирование на наборах сгенерированных задач с различными параметрами. Анализ результатов тестов подтвердил и уточнил теоретические выводы. Эффективность использования методов зависит от скорости роста экономической модели: при быстром экспоненциальном росте может быть предпочтительней найти приближенное решение с помощью метода континуализации, а при умеренном росте можно за приемлемое время найти точное решение с помощью метода динамического программирования.

Исследование методов не только показало их применимость к реше-

нию задачи, но также и наглядно продемонстрировало различия подходов, определяющих эти методы. Однако подходы к динамической оптимизационной задаче разнообразны, и даже в рамках подходов методы решения могут быть различны. Поэтому исследование задачи оптимизации целочисленной модели Неймана может быть продолжено — можно отметить такие возможные направления исследований, как обобщение описанного в данной работе метода динамического программирования на случай интегральной полезности, реализация приближенного метода динамического программирования, дающего более точную нижнюю оценку, чем метод континуализации, а также применение к задаче метода ветвей и границ.

Список литературы

[1] *Neumann J. von.* A Model of general economic equilibrium // The Review of Economic Studies, 1945. Vol. 13, No 1. P. 1-9.

[2] *Ланкастер К.* Математическая экономика. М.: Советское радио, 1972. 464 с.

[3] *Гейл Д.* Теория линейных экономических моделей. М.: Изд-во иностранной литературы, 1963. 418 с.

[4] *Ашманов С. А.* Математические модели и методы в экономике. М.: Изд-во Московского университета, 1980. 199 с.

[5] *Данилов Н. Н.* Курс математической экономики. М.: Высшая школа, 2006. 407 с.

[6] *Моришима М.* Равновесие, устойчивость, рост. М: Наука, 1972. 280 с.

[7] *Хачиян Л. Г.* Сложность задач линейного программирования // Новое в жизни, науке, технике. Сер. « Математика, кибернетика». М.: Знание, 1987. №10. 32 с.

[8] Clp Documentation. http://coin-or.org/Doxygen/Clp/index.html

[9] *Mehrotra S.* On the implementation of a primal–dual interior point method // SIAM Journal on Optimization, 1992. Vol. 2, No 4. P. 575-601.

[10] Official gnuplot documentation. http://gnuplot.info/documentation.html

Приложение

Метод континуализации

```
void NMProblem::solveByContinualizationMethod()
{
  //векторы для создания "упакованной" матрицы коэффициентов задачи
  std::vector<double> elem;
  std::vector<int> rowInd;
  std::vector<int> colInd;
  //вектор ограничений
  std::vector<double> lim(T*n, 0);
  //исходя из данных на входе, формируем вышеуказанные векторы
  for (int j = 0; j < n; ++j)
  {
    for (int i = 0; i < m; ++i)
    {
      if (A[i][j] != 0)
      {
        elem.push_back(A[i][j]);
        rowInd.push_back(j);
        colInd.push_back(i);
      }
    }
    lim[j] = y0[j];
  }
  for (int t = 1; t < T; ++t)
    for (int j = 0; j < n; ++j)
    {
      for (int i = 0; i < m; ++i)
      {
        if (B[i][j] != 0)
        {
          elem.push_back(-B[i][j]);
          rowInd.push_back(t*n + j);
          colInd.push_back((t - 1)*m + i);
          lim[t*n + j] -= B[i][j];
        }
      }
      for (int i = 0; i < m; ++i)
      {
        if (A[i][j] != 0)
        {
          elem.push_back(A[i][j]);
          rowInd.push_back(t*n + j);
          colInd.push_back(t*m + i);
        }
      }
    }
```

```
  //матрица коэф. задачи (Tn x Tm)
  CoinPackedMatrix coefPackedMatrix(false, &rowInd[0],
    &colInd[0], &elem[0], elem.size());

  //сформируем вектор коэф. (при z) целевой функции
  std::vector<double> objCoef(T*m, 0);
  for (int i = 0; i < m; ++i)
  {
    for (int j = 0; j < n; ++j)
    {
      for (int t = 0; t < T; ++t)
      {
        if (A[i][j] != 0)
          objCoef[t*m + i] -= c[j] * k[t] * A[i][j];
        if (B[i][j] != 0)
          objCoef[t*m + i] += c[j] * k[t + 1] * B[i][j];
      }
    }
  }
  //посчитаем значение свободного члена целевой функции
  double objFreeTerm = 0;
  for (int j = 0; j < n; ++j)
    objFreeTerm += k[0] * c[j] * y0[j];

  //создаем модель задачи, максимизирующую цел. ф-ю
  ClpPredictorCorrector model;
  model.loadProblem(coefPackedMatrix, 0, 0, &objCoef[0], 0, &lim[0]);
  model.setOptimizationDirection(-1);
  model.setMaximumIterations(5000);
  model.setMaximumBarrierIterations(5000);
  model.setMaximumSeconds(60);
  //решаем при помощи метода прогноза и коррекции Мехротра
  model.solve();

  //получаем оптимальное решение и значение цел. ф-ии
  cmSolution = new unsigned long int[T*m];
  for (int i = 0; i < T*m; ++i)
    cmSolution[i] = unsigned long
      int(floor(model.primalColumnSolution()[i]));
  cmObjective = objFreeTerm;
  for (int i = 0; i < T*m; ++i)
    cmObjective += objCoef[i] * double(cmSolution[i]);
  cmSolved = true;
}
```

Метод динамического программирования

```
void NMProblem::solveByDynamicProgramming()
{
  if (!terminal) return;
```

```
//сохраним разреженные A и B в "упакованном" формате
std::vector<double> elemA, elemB;
std::vector<int> rowIndA, rowIndB;
std::vector<int> colIndA, colIndB;
//A сохраним построчно
for (int i = 0; i < m; ++i)
  for (int j = 0; j < n; ++j)
  {
    if (A[i][j] != 0)
    {
      elemA.push_back(A[i][j]);
      rowIndA.push_back(i);
      colIndA.push_back(j);
    }
  }
CoinPackedMatrix packedA(false, &rowIndA[0],
  &colIndA[0], &elemA[0], elemA.size());
//B сохраним по столбцам
for (int j = 0; j < n; ++j)
  for (int i = 0; i < m; ++i)
  {
    if (B[i][j] != 0)
    {
      elemB.push_back(B[i][j]);
      rowIndB.push_back(i);
      colIndB.push_back(j);
    }
  }
CoinPackedMatrix packedB(true, &rowIndB[0],
  &colIndB[0], &elemB[0], elemB.size());

//в этом списке будут П.-опт. вектора, допустимые из опред. z_{t-1})
std::list<unsigned long int*> allowableParetoOptimalVectors;

//вектор ссылкок на списки П.-опт. на каждом шаге векторов
//при этом для каждого вектора хранится также
  //адрес П.-опт. на пред. шаге вектора, для которого он допустим
std::vector<std::list<VectorAndParent>*> paretoOptimalVectorsOnStep(T);
for (int t = 0; t < T; ++t)
  paretoOptimalVectorsOnStep[t] = new std::list<VectorAndParent>;
VectorAndParent* newVectorAndParent;

//получим список Парето-оптимальных векторов на первом шаге
fillParetoOptimalVectorsList(allowableParetoOptimalVectors, packedA, y0);
for (auto it = allowableParetoOptimalVectors.begin();
  it != allowableParetoOptimalVectors.end(); ++it)
{
  newVectorAndParent = new VectorAndParent(*it, NULL);
  (*paretoOptimalVectorsOnStep[0]).push_back(*newVectorAndParent);
}
allowableParetoOptimalVectors.clear();
```

```
//используя этот список, получим аналогичные списки для след шага и т.д.
double* zB = new double[n];
for (int t = 1; t < T; ++t)
{
  //для каждого П.-опт. вектора из прошлого шага
    //строим допустимые П.-опт. векторы и сливаем в один список
  for (auto prevStepListIt = (*paretoOptimalVectorsOnStep[t-1]).begin();
    prevStepListIt != (*paretoOptimalVectorsOnStep[t-1]).end();
    ++prevStepListIt)
  {
    //вычисление z_{t-1}*B
    for (int j = 0; j < n; ++j)
    {
      zB[j] = 0;
      for (int k = 0; k < packedB.getVector(j).getNumElements(); ++k)
        zB[j] += packedB.getVector(j).getElements()[k] *
          prevStepListIt->elem[packedB.getVector(j).getIndices()[k]];
    }

    fillParetoOptimalVectorsList(allowableParetoOptimalVectors,
      packedA, zB);
    transferToGeneralList(allowableParetoOptimalVectors,
      &(*prevStepListIt), (*paretoOptimalVectorsOnStep[t]));
  }

  //удалив доминируемые векторы, получим список П-опт. на шаге векторов
  bool isDominated;
  for (auto pivIt = (*paretoOptimalVectorsOnStep[t]).begin();
    pivIt != (*paretoOptimalVectorsOnStep[t]).end(); ++pivIt)
  {
    auto candidateIt = std::next(pivIt);
    while (candidateIt != (*paretoOptimalVectorsOnStep[t]).end())
    {
      isDominated = true;
      for (int i = 0; i < m; ++i)
        if (candidateIt->elem[i] > pivIt->elem[i])
          isDominated = false;
      if (isDominated)
        candidateIt = (*paretoOptimalVectorsOnStep[t]).erase(candidateIt);
      else ++candidateIt;
    }
  }
}

//определим, какой вектор на последнем шаге максимизирует целевую ф-ю
double maxObjVal = 0;
double objVal;
VectorAndParent* optimal;
for (auto it = (*paretoOptimalVectorsOnStep[T-1]).begin();
  it != (*paretoOptimalVectorsOnStep[T-1]).end(); ++it)
{
  objVal = 0;
```

```
    for (int j = 0; j < n; ++j)
      for (int i = 0; i < m; ++i)
        objVal += c[j]*B[i][j]*(it->elem[i]);
    if (objVal >= maxObjVal)
    {
      maxObjVal = objVal;
      optimal = &(*it);
    }
  }

  //с конца построим оптимальную цепочку векторов интенсивностей
  dpObjective = maxObjVal;
  dpSolution = new unsigned long int[T*m];
  for (int t = T - 1; t >= 0; --t)
  {
    for (int i = 0; i < m; ++i)
      dpSolution[t*m + i] = optimal->elem[i];
    optimal = optimal->parent;
  }
  dpSolved = true;

  delete[] zB;
  for (int t = 0; t < T; ++t)
    delete paretoOptimalVectorsOnStep[t];
}

//генерирует векторы в обратном лексикогр. порядке, сохраняя П.-оптимальные
void NMProblem::fillParetoOptimalVectorsList(
  std::list<unsigned long int*>& paretoOptimalVectors,
  CoinPackedMatrix& coef, double* bound)
{
  std::vector<unsigned long int>* candidate =
    new std::vector<unsigned long int>(m);
  std::vector<unsigned long int>* prev_candidate =
    new std::vector<unsigned long int>(m);
  bool isDominated;
  std::vector<double> boundVector(bound, bound + n);
  //каждую итерацию мы определяем некоторый элемент вектора как "опорный" -
    //набор рассматриваемых векторов-кандидатов мы получаем, уменьшая
    //этот элемент на 1 и генерируя элементы после него наибольшим образом
  maximizeVector(&(*candidate)[0], 0, coef, boundVector);
  for (int i = 0; i < m-1; ++i)
  {
    //проверяем вектор, от которого будем перебирать в этой итерации
    addIfNotDominated(&(*candidate)[0], paretoOptimalVectors);
    prev_candidate = candidate;
    candidate = new std::vector<unsigned long int>(*prev_candidate);

    //в цикле уменьшаем "опорный" элемент и проверяем векторы на П.-опт.
    while ((*candidate)[i] > 0)
    {
      --(*candidate)[i];
```

```
      maximizeVector(&(*candidate)[0], i+1, coef, boundVector);
      isDominated = !addIfNotDominated(&(*candidate)[0],
        paretoOptimalVectors);
      prev_candidate = candidate;
      candidate = new std::vector<unsigned long int>(*prev_candidate);
      if (isDominated) delete prev_candidate;
    }
  }
  //последний эл-т нет смысла делать "опорным", проверяем вектор на П.-опт.
  if (!addIfNotDominated(&(*candidate)[0], paretoOptimalVectors))
    delete candidate;
}

//начиная с нек. элемента генерирует вектор, наиб. в лексикогр. смысле
void NMProblem::maximizeVector(unsigned long int* subjVector, int startInd,
    CoinPackedMatrix& coef, std::vector<double> bound)
{
  CoinShallowPackedVector coefRow;
  unsigned long int max;
  for (int i = startInd; i < m; ++i)
  {
    //в coefRow сохраним ненулевые коэффициенты для i-го элемента
    coefRow = coef.getVector(i);
    //вычислим максимальное значение с учетом коэффициентов и ограничений
    if (coefRow.getNumElements() != 0)
    {
      max = unsigned long int(floor(bound[coefRow.getIndices()[0]] /
        coefRow.getElements()[0]));
      for (int k = 1; k < coefRow.getNumElements(); ++k)
      {
        if (unsigned long int(floor(bound[coefRow.getIndices()[k]] /
          coefRow.getElements()[k])) < max)
          max = unsigned long int(floor(bound[coefRow.getIndices()[k]] /
            coefRow.getElements()[k]));
      }
      //обновим значение ограничений
      if (max != 0 && i != m - 1)
        for (int k = 0; k < coefRow.getNumElements(); ++k)
          bound[coefRow.getIndices()[k]] -=
            max*(unsigned long int(coefRow.getElements()[k]));
      //присвоим вычисленное максимальное значение элементу
      subjVector[i] = max;
    }
    else //этот случай соотв. неправильному заполнению матрицы A
    {
      std::cout << "\nМатрица A содержит нулевую строку\n";
      subjVector[i] = 0;
    }
  }
}

//добавляет вектор в список и возвращает true,
```

```
  //если он не доминируется ни одним вектором из списка
bool NMProblem::addIfNotDominated(unsigned long int* subjVector,
  std::list<unsigned long int*>& listOfVectors)
{
  bool isDominated;
  for (auto it = listOfVectors.begin(); it != listOfVectors.end(); ++it)
  {
    isDominated = true;
    for (int i = 0; i < m; ++i)
      if (subjVector[i] > (*it)[i])
        isDominated = false;
    if (isDominated) return false;
  }
  listOfVectors.push_back(subjVector);
  return true;
}

//переносит элементы списка П.-опт. векторов, допустимых из опред. z_{t-1},
  //в список для шага, сохраняя ссылку на z_{t-1} и не нарушая порядок
void NMProblem::transferToGeneralList(std::list<unsigned long int*>& in,
  VectorAndParent* parent, std::list<VectorAndParent>& out)
{
  bool isDuplicate;
  VectorAndParent* newOutElement;
  auto itOut = out.begin();
  for (auto itIn = in.begin(); itIn != in.end(); ++itIn)
  {
    for (int i = 0; i < m; ++i)
    {
      while ((itOut != out.end()) && (itOut->elem[i] >(*itIn)[i]))
        ++itOut;
      if (itOut == out.end()) break;
      else
        if (itOut->elem[i] < (*itIn)[i]) break;
    }
    if (itOut == out.end())
    {
      newOutElement = new VectorAndParent(*itIn, parent);
      out.push_back(*newOutElement);
    }
    else
    {
      //необходимо проверить, не является ли добавляемый вектор дубликатом
      isDuplicate = true;
      for (int i = 0; i < m; ++i)
        if ((*itIn)[i] != itOut->elem[i])
          isDuplicate = false;
      if (!isDuplicate)
      {
        newOutElement = new VectorAndParent(*itIn, parent);
        out.insert(itOut, *newOutElement);
      }
```

```
      else delete[] (*itIn);
    }
  }
  //список, из которого перенесены элементы, очищается
  in.clear();
}
```