

# Содержание

<b>От издательства</b> .....	12
<b>Предисловие</b> .....	13
<b>Глава 1. Введение</b> .....	17
1.1. Языки моделирования для оптимизации .....	17
1.2. Моделирование на Pyomo .....	19
1.2.1. Простые примеры .....	19
1.2.2. Пример раскраски графа .....	21
1.2.3. Ключевые особенности Pyomo .....	24
Python .....	24
Настраиваемые возможности .....	24
Командные инструменты и скрипты .....	24
Определение конкретных и абстрактных моделей .....	24
Объектно ориентированный дизайн .....	25
Выразительные возможности моделирования .....	25
Интеграция с решателями .....	25
Открытый исходный код .....	25
1.3. Подготовительные действия .....	26
1.4. Краткий обзор книги .....	26
1.5. Обсуждение .....	27
<b>Часть I. ВВЕДЕНИЕ В PYOMO</b> .....	28
<b>Глава 2. Математическое моделирование и оптимизация</b> .....	29
2.1. Математическое моделирование .....	29
2.1.1. Общие сведения .....	29
2.1.2. Пример моделирования .....	30
2.2. Оптимизация .....	32
2.3. Моделирование в Pyomo .....	34
2.3.1. Конкретная формулировка .....	34
2.4. Линейные и нелинейные модели оптимизации .....	36
2.4.1. Определение .....	36
2.4.2. Линейная версия .....	37
2.5. Решение модели Pyomo .....	37
2.5.1. Решатели .....	37
2.5.2. Python-скрипты .....	37

<b>Глава 3. Обзор Puomo</b> .....	39
3.1. Введение .....	39
3.2. Задача о расположении складов .....	40
3.3. Модели Puomo.....	41
3.3.1. Переменные, целевые функции и ограничения .....	41
3.3.2. Индексированные компоненты .....	42
3.3.3. Правила конструирования .....	44
3.3.4. Конкретная модель для задачи о расположении складов .....	45
3.3.5. Компоненты моделирования для множеств и параметров .....	48
<b>Глава 4. Модели Puomo и их компоненты: введение</b> .....	51
4.1. Объектно ориентированный AML .....	51
4.2. Общие парадигмы компонентов .....	53
4.2.1. Индексированные компоненты .....	53
4.3. Переменные .....	54
4.3.1. Объявления Var .....	54
4.3.2. Работа с объектами Var .....	57
4.4. Целевые функции.....	57
4.4.1. Объявление Objective .....	58
4.4.2. Работа с объектами Objective .....	59
4.5. Ограничения.....	59
4.5.1. Объявление Constraint .....	60
4.5.2. Работа с объектами Constraint .....	62
4.6. Множества .....	62
4.6.1. Объявление Set.....	63
4.6.2. Работа с объектами Set.....	66
4.7. Параметры .....	68
4.7.1. Объявление Param.....	68
4.7.2. Работа с объектами Param .....	71
4.8. Именованные выражения.....	72
4.8.1. Объявление Expression.....	73
4.8.2. Работа с объектами Expression .....	74
4.9. Суффиксы .....	74
4.9.1. Объявления Suffix .....	75
4.9.2. Работа с суффиксами.....	76
4.10. Другие компоненты моделирования .....	77
<b>Глава 5. Программирование нестандартных технологических процессов</b> .....	79
5.1. Введение .....	79
5.2. Опрос модели.....	82
5.2.1. Функция value.....	83
5.2.2. Доступ к атрибутам индексированных компонентов.....	84
5.2.2.1. Срезы индексов компонентов .....	84
5.2.2.2. Обход всех объектов Var в модели .....	84

5.3. Модификация структуры модели Pyomo.....	85
5.4. Типичные примеры программирования .....	86
5.4.1. Цикл по местоположениям складов и построение диаграммы .....	86
5.4.2. Решатель судoku .....	88

<b>Глава 6. Взаимодействие с решателями .....</b>	<b>94</b>
6.1. Введение .....	94
6.2. Использование решателей.....	95
6.3. Исследование решения .....	97
6.3.1. Результаты решателя.....	97

## **Часть II. ДОПОЛНИТЕЛЬНЫЕ ТЕМЫ .....**

99

<b>Глава 7. Нелинейное программирование в Pyomo.....</b>	<b>100</b>
7.1. Введение .....	100
7.2. Задачи нелинейного программирования в Pyomo .....	101
7.2.1. Нелинейные выражения .....	101
7.2.2. Задача Розенброка .....	102
7.3. Решение задач нелинейного программирования.....	104
7.3.1. Нелинейные решатели .....	105
7.3.2. Дополнительные советы по нелинейному программированию .....	105
Инициализация переменных .....	105
Неопределенные вычисления .....	106
Сингулярности модели и масштабирование задачи .....	106
7.4. Примеры нелинейного программирования.....	107
7.4.1. Инициализация переменных для мультимодальной функции .....	107
7.4.2. Оптимальные квоты для неистощительной добычи оленей .....	108
7.4.3. Оценка моделей инфекционных заболеваний.....	112
7.4.4. Проектирование реактора .....	115

<b>Глава 8. Структурное моделирование с помощью блоков .....</b>	<b>119</b>
8.1. Введение .....	119
8.2. Блочные структуры.....	121
8.3. Блоки как индексированные компоненты.....	123
8.4. Правила конструирования внутри блоков .....	124
8.5. Извлечение значений из иерархических моделей .....	125
8.6. Пример использования блоков: оптимальный многопериодный размер партии .....	126
8.6.1. Формулировка без блоков .....	127
8.6.2. Формулировка с блоками .....	129

<b>Глава 9. Производительность: конструирование модели и интерфейсы с решателями.....</b>	<b>131</b>
9.1. Выявление узких мест с помощью профилирования.....	131
9.1.1. Хронометраж.....	133
9.1.2. TicTocTimer .....	133

9.1.3. Профилировщики .....	134
9.2. Повышение производительности конструирования модели с помощью класса <code>LinearExpression</code> .....	137
9.3. Многократное решение с применением хранимых решателей .....	138
9.3.1. Когда использовать хранимый решатель .....	138
9.3.2. Основы использования .....	139
9.3.3. Работа с индексированными переменными и ограничениями .....	141
9.3.4. Повышение производительности .....	142
9.3.5. Пример .....	142
9.4. Разреженные множества индексов .....	143

## **Глава 10. Абстрактные модели и их решение**..... 145

10.1. Общие сведения .....	145
10.1.1. Абстрактные и конкретные модели .....	145
10.1.2. Абстрактная формулировка модели (H) .....	147
10.1.3. Абстрактная модель для задачи о расположении складов .....	148
10.2. Команда <code>pyomo</code> .....	150
10.2.1. Подкоманда <code>help</code> .....	151
10.2.2. Подкоманда <code>solve</code> .....	152
10.2.2.1. Задание объекта модели .....	154
10.2.2.2. Выбор данных с помощью пространств имен .....	155
10.2.2.3. Настройка технологического процесса <code>Pyomo</code> .....	158
<code>pyomo_preprocess</code> .....	159
<code>pyomo_create_model</code> .....	159
<code>pyomo_create_modeldata</code> .....	159
<code>pyomo_print_model</code> .....	159
<code>pyomo_modify_instance</code> .....	160
<code>pyomo_print_instance</code> .....	160
<code>pyomo_save_instance</code> .....	160
<code>pyomo_print_results</code> .....	160
<code>pyomo_save_results</code> .....	161
<code>pyomo_postprocess</code> .....	161
10.2.2.4. Настройка поведения решателя .....	161
10.2.2.5. Анализ результатов решателя .....	162
10.2.2.6. Управление диагностической печатью .....	162
10.2.3. Подкоманда <code>convert</code> .....	164
10.3. Команды данных для <code>AbstractModel</code> .....	165
10.3.1. Команда <code>set</code> .....	166
10.3.1.1. Простые множества .....	166
10.3.1.2. Множество кортежей .....	167
10.3.1.3. Массивы множеств .....	168
10.3.2. Команда <code>param</code> .....	168
10.3.2.1. Одномерные параметрические данные .....	169
10.3.2.2. Многомерные параметрические данные .....	171
10.3.3. Команда <code>include</code> .....	173
10.3.4. Пространства имен данных .....	173

10.4. Компоненты построения .....	174
-----------------------------------	-----

### **Часть III. РАСШИРЕНИЯ МОДЕЛИРОВАНИЯ .....**

176
-----

#### **Глава 11. Обобщенное дизъюнктивное программирование .....**

177
-----

11.1. Введение .....	177
11.2. Моделирование ОДП в Pyomo .....	180
11.3. Выражение логических ограничений .....	182
11.4. Решение моделей ОДП .....	183
11.4.1. Преобразование типа «М большое» .....	183
11.4.2. Оболочечное преобразование .....	184
11.5. Смешанная задача с полунепрерывными переменными .....	185

#### **Глава 12. Дифференциальные алгебраические уравнения .....**

187
-----

12.1. Введение .....	187
12.2. Компоненты моделирования ДАУ в Pyomo .....	188
12.3. Решения моделей Pyomo с ДАУ .....	190
12.3.1. Конечно-разностное преобразование .....	191
12.3.2. Преобразование коллокации .....	192
12.4. Дополнительные возможности .....	193
12.4.1. Применение нескольких дискретизаций .....	193
12.4.2. Ограничение формы управляющих входов .....	194
12.4.3. Построение графиков .....	194

#### **Глава 13. Математические программы с ограничениями равновесия .....**

196
-----

13.1. Введение .....	196
13.2. Моделирование условий равновесия .....	197
13.2.1. Условия дополнителности .....	197
13.2.2. Выражения дополнителности .....	198
13.2.3. Моделирование смешанных условий дополнителности .....	198
13.3. Преобразования МПОР .....	202
13.3.1. Преобразование standard_form .....	202
13.3.2. Преобразование simple_nonlinear .....	203
13.3.3. Преобразование simple_disjunction .....	203
13.3.4. Интерфейс с AMPL-решателями .....	204
13.4. Интерфейсы с решателями и метарешатели .....	205
13.4.1. Нелинейные переформулирования .....	205
13.4.2. Дизъюнктивные переформулирования .....	206
13.4.3. PATH и интерфейс с ASL-решателем .....	206
13.5. Обсуждение .....	207

#### **Приложение А. Краткое руководство по Python .....**

208
-----

A.1. Обзор .....	208
A.2. Установка и выполнение Python .....	209
A.3. Формат строки в Python .....	210

---

А.4. Переменные и типы данных.....	211
А.5. Структуры данных.....	212
А.5.1. Строки.....	212
А.5.2. Списки.....	212
А.5.3. Кортежи.....	213
А.5.4. Множества.....	214
А.5.5. Словари.....	214
А.6. Условные предложения.....	214
А.7. Итерации и циклы.....	215
А.8. Генераторы и списковые включения.....	216
А.9. Функции.....	216
А.10. Объекты и классы.....	218
А.11. Присваивание, сору и деерсору.....	219
А.11.1. Ссылки.....	219
А.11.2. Копирование.....	220
А.12. Модули.....	220
А.13. Ресурсы, посвященные Python.....	221
<b>Литература.....</b>	<b>222</b>
<b>Предметный указатель.....</b>	<b>225</b>

# Предисловие

В этой книге описывается инструмент математического моделирования – программа Python Optimization Modeling Objects (Pyomo). Pyomo поддерживает создание и анализ математических моделей для сложных приложений оптимизации. Обычно эта способность ассоциируется с языками алгебраического моделирования (algebraic modeling languages – AML) высокого уровня. Большинство AML реализованы как самостоятельные языки моделирования, но объекты моделирования Pyomo встроены в Python, полноценный язык программирования высокого уровня, для которого имеется обширный набор библиотек. Pyomo получил награды организации R&D100 и INFORMS Computing Society.

Моделирование – фундаментальный процесс во многих аспектах научных исследований, техники и бизнеса, а благодаря широкому распространению вычислительной техники численный анализ математических моделей стал обыденностью. Ко всему прочему основным принципом языков AML стала робастная формулировка больших моделей для сложных приложений, встречающихся на практике [37]. AML облегчили процесс формулирования моделей, упростив управление разреженными данными и добавив поддержку естественного выражения компонентов модели. Дополнительно AML типа Pyomo поддерживают написание скриптов, включающих объекты модели, что дает возможность проводить нестандартный анализ сложных задач.

Основой Pyomo является объектно ориентированное представление моделей оптимизации. Также Pyomo содержит пакеты для определения расширений и переформулирования модели. Кроме того, в состав Pyomo входят пакеты, определяющие интерфейсы с такими решателями, как CPLEX и Gurobi, и службами решения типа NEOS.

## Цели этой книги

В третье издание включено переработанное описание средства моделирования Pyomo. Основная цель книги – дать общее описание Pyomo, которое позволило бы пользователям создавать и оптимизировать модели. Поэтому в книге много примеров, иллюстрирующих различные методы формулирования моделей.

Другая цель книги – проиллюстрировать богатство возможностей Pyomo, в том числе формулирование и анализ типичных моделей оптимизации, включая линейное программирование, смешанно-целочисленное линейное программирование, нелинейное программирование, смешанно-целочисленное нелинейное программирование, математическое программирова-

ние с ограничениями равновесия, ограничения и целевые функции, основанные на дифференциальных уравнениях, обобщенное дизъюнктивное программирование. Кроме того, Pyomo включает интерфейсы к различным распространенным пакетам программ оптимизации, как то CBC, CPLEX, GLPK и Gurobi. Модели Pyomo можно оптимизировать с помощью таких программ, как IPOPT, в которой используется интерфейс к библиотеке AMPL Solver Library.

Наконец, книга призвана облегчить знакомство с Pyomo даже тем пользователям, которые мало что знают о Python. Приложение A содержит краткое введение в Python, но мы были поражены тем, насколько справочники по Python полезны новым пользователям Pyomo. Хотя в Pyomo используются объекты Python, выражение моделей на Pyomo следует ясному и лаконичному синтаксису Python.

Однако в нашем обсуждении продвинутых средств моделирования Pyomo предполагается некоторое знакомство с объектно ориентированным проектированием и возможностями языка Python. Например, мы проводим различие между определением класса и его экземплярами. Мы не пытались объяснить эти средства Python в книге. Поэтому от читателя ожидается желание хотя бы немного изучить Python, если он хочет понимать и эффективно использовать продвинутые средства моделирования.

## Для кого предназначена эта книга

Книга представляет собой справочное пособие для студентов, научных работников и инженеров-практиков. Структура Pyomo настолько проста, что программу можно использовать в курсах для студентов и аспирантов. Однако предполагается, что читатель знаком с основами оптимизации и математического моделирования. В книге нет глоссария, но мы рекомендуем использовать для справки глоссарий математического программирования [32].

Pyomo также является ценным инструментом для теоретиков и практиков. При разработке Pyomo много внимания было уделено способности поддерживать описание и анализ реальных приложений. А значит, производительность и надежные интерфейсы с решателями занимали не последнее место.

Кроме того, мы надеемся, что исследователи найдут в Pyomo эффективный каркас для разработки высокоуровневых средств оптимизации и анализа. Например, Pyomo лежит в основе пакета оптимизации в условиях неопределенности `mpi-sppy`, в котором используется тот факт, что объекты моделирования встроены в полнофункциональный высокоуровневый язык программирования. Это позволяет прозрачно распараллеливать подзадачи с помощью соответствующих библиотек Python. Этот механизм разработки обобщенных решателей для сложных моделей очень мощный, и мы надеемся, что его можно будет использовать в сочетании со многими другими методами оптимизации.



## Благодарности

Мы ценим усилия многих людей, способствовавших выходу этого и предыдущих изданий книги. Мы благодарны Элизабет Лоев из издательства Springer, которая сопровождала книгу от идеи до последних этапов производства; ее энтузиазм заразителен. Мы также благодарим Маделинн Фарбер из Sandia National Laboratories за помощь в юридических нюансах выпуска программного обеспечения с открытым исходным кодом и публикации книги. Наконец, мы признательны Дугу Проуту за разработку логотипов Pyomo, PySP, Pyomo.DAE и Coopr.

Мы в долгу перед всеми, кто тратил свое время и силы на рецензирование книги. Не будь их, книга содержала бы много опечаток и программных ошибок. Итак, спасибо Джеку Инголлу, Зеву Фридману, Харви Гринбергу, Шону Леггу, Анжелике Вонг, Дэниэлю Уорду, Деанне Гарсиа, Эллис Озакиол и Флориану Мадеру. Отдельное спасибо Эмбер Грей-Фенне и Рэнди Бросту.

Особенно мы благодарны растущему сообществу пользователей Pyomo. Ваш интерес к Pyomo и ваш энтузиазм стали самым важным фактором, повлиявшим на наше решение написать эту книгу. Мы благодарим первых приверженцев Pyomo, которые поделились с нами подробными отзывами о структуре и полезности программы, в том числе Фернандо Бадилла, Стивена Чена, Неда Дмитрова. ЮэЮэ Фана, Эрика Хонга, Аллена Холдера, Андреса Ироуме, Дэррила Меландера, Кэрол Мейерс, Пьера Нансель-Пенарда, Мехула Рангвала, Еву Уормингхаус и Дэвида Алдерсона. Ваши отзывы продолжают оказывать важное влияние на дизайн и возможности Pyomo.

Мы также благодарны нашим друзьям из проекта COIN-OR за поддержку Pyomo. Хотя разработка Pyomo ведется в основном на GitHub, наше партнерство с COIN-OR – ключевая часть нашей стратегии, благодаря которой Pyomo остается жизнеспособным проектом с открытым исходным кодом.

Особая благодарность – нашим коллегам, создававшим пакеты для Pyomo: Франсиско Муньосу, Тимоти Эклу, Кэвину Хантеру, Патрику Стилу и Дэниэлю Уорду. Мы также признательны Тому Браунстейну, Дэйву Гэю и Нику Бенеvidасу за помощь в разработке модулей Python документации по Pyomo.

Авторы выражают благодарность за поддержку, оказанную при создании этой книги: Национальному научному фонду (гранты СВЕТ#0941313 и СВЕТ#0955205), Управлению передовых научных исследований в области вычислений при отделении по науке Министерства энергетики США, проекту ARPA-E Министерства энергетики США в рамках программы интеграции зеленых электрических сетей, Институту проектирования передовых энергетических систем (IDAES) с финансированием от отдела технических систем на основе имитационного моделирования, междисциплинарную исследовательскую программу отделения ископаемого топлива Министерства энергетики США, программу моделирования передовых сетей электроснабжения (AGM) Министерства энергетики США и Лабораторию целенаправленных исследований и разработок в Sandia National Laboratories.

И наконец, мы благодарны своим семьям и друзьям за их терпеливое отношение к нашей страсти к программам оптимизации.

## Замечания и вопросы

В этой книге документирована версия Pyomo 6.0. Дополнительную информацию, включая опечатки, смотрите на сайте Pyomo: <http://www.pyomo.org>.

Исходный код Pyomo размещен на GitHub, а примеры, приведенные в этой книге, находятся в каталоге `pyomo/examples/doc/pyomobook`: <https://github.com/Pyomo/pyomo>.

На многие вопросы, касающиеся Pyomo, есть ответы на сайте Stack Overflow: <https://stackoverflow.com/>.

Мы приветствуем отзывы читателей. Направляйте их непосредственно авторам или на форум Pyomo: [pyomo-forum@googlegroups.com](mailto:pyomo-forum@googlegroups.com).

Удачи!

Альбукерк, Нью-Мексико, США  
Энн Арбор, Мичиган, США  
Альбукерк, Нью-Мексико, США  
Альбукерк, Нью-Мексико, США  
Альбукерк, Нью-Мексико, США  
Альбукерк, Нью-Мексико, США  
Ливермор, Калифорния, США  
Дэвис, Калифорния, США

*Майкл Бинум  
Гэйб Хакебейл  
Уильям Харт  
Карл Лэрд  
Бетани Николсон  
Джон Сиурола  
Жан-Поль Уотсон  
Дэвид Л. Вудраф*

5 января 2021 г.

# Глава 1

## Введение

В этой главе мы познакомимся с Pyomo, инструментом для моделирования и решения задач оптимизации на основе Python, и расскажем, зачем он создавался. Моделирование – фундаментальный процесс во многих аспектах научных исследований, техники и бизнеса. Языки алгебраического моделирования типа Pyomo – это высокоуровневые языки для формулирования и решения математических задач оптимизации. Pyomo – гибкий и расширяемый каркас моделирования, который вбирает в себя и обобщает главные идеи языков алгебраического моделирования, организуя их в контексте широко распространенного языка программирования.

### 1.1. ЯЗЫКИ МОДЕЛИРОВАНИЯ ДЛЯ ОПТИМИЗАЦИИ

В этой книге описывается инструмент математического моделирования: программный пакет Python Optimization Modeling Objects (Pyomo). Pyomo поддерживает формулирование и анализ математических моделей для сложных задач оптимизации. Обычно эта возможность ассоциируется с коммерческими языками алгебраического моделирования (AML), такими как AIMMS [1], AMPL [2] и GAMS [22]. Pyomo реализует развитый набор средств моделирования и анализа и предоставляет доступ к этим средствам из полнофункционального высокоуровневого языка программирования Python, для которого написано множество библиотек.

Модели оптимизации определяют целевые функции для рассматриваемой системы. Модели можно использовать для исследования компромиссов между различными целевыми функциями, нахождения экстремальных состояний и худших случаев, а также идентификации основных факторов, от которых зависит поведение системы. Поэтому модели оптимизации применяются для анализа широкого круга научных, деловых и технических задач.

Благодаря широкой доступности вычислительных ресурсов численный анализ моделей оптимизации стал обыденным явлением. Для вычислительного анализа модели оптимизации необходимо описать модель и передать ее программе-решателю. В отсутствие языка спецификации таких моделей процесс написания входных файлов, выполнения решателя и получения ре-

зультатов от него оказывается утомительным и чреват ошибками. На это накладывает тот факт, что реальные приложения велики и сложны, так что отлаживать их трудно. Ко всему прочему для решателей имеется много разных форматов ввода, но лишь малая их часть стандартизирована. Таким образом, применение нескольких решателей для анализа одной модели оптимизации влечет за собой дополнительные сложности. Кроме того, верифицировать модель (т. е. проверить, что переданная решателю модель точно выражает намерения разработчика) исключительно трудно, не имея высокоуровневого языка для выражения модели.

AMPL – это высокоуровневые языки для описания и решения задач оптимизации [26, 37]. Они сводят к минимуму трудности, связанные с анализом моделей, благодаря высокоуровневой спецификации модели. Кроме того, AMPL предоставляют интерфейсы к внешним программам-решателям, которые используются для анализа задач, и позволяют пользователю взаимодействовать с результатами решателя в контексте этой спецификации.

Специализированные AMPL, такие как AIMMS [1], AMPL [2, 21] и GAMS [22], представляют собой языки описания модели оптимизации с интуитивно понятным и лаконичным синтаксисом для определения переменных, ограничений и целевых функций. Кроме того, они поддерживают абстрактные понятия: разреженные множества, индексы и алгебраические выражения, без которых не обойтись при описании крупномасштабных реальных задач с тысячами или миллионами ограничений и переменных. Эти AMPL могут представлять самые разные модели оптимизации и реализуют интерфейсы с различными решателями. В последнее время AMPL стали поддерживать новые скриптовые возможности, которые позволяют выражать высокоуровневые алгоритмы анализа вместе со спецификацией модели оптимизации.

Альтернативная стратегия – использовать AMPL, который расширяет какой-нибудь стандартный высокоуровневый язык программирования (в противоположность специализированному проприетарному языку) для формулирования моделей оптимизации, которые затем анализируются решателями, написанными на низкоуровневых языках. Такой двухязыковый подход сочетает гибкость высокоуровневого языка для формулирования задач оптимизации и эффективность низкоуровневого для численных расчетов. Этот подход завоевывает все большую популярность в программном обеспечении для научных расчетов. Среда оптимизации TOMLAB, написанная на Matlab [57], – один из наиболее зрелых пакетов, в которых он применяется; в Rуomo также используется этот подход. Есть также примеры расширения стандартных языков программирования типа Java и C++ путем включения AMPL-конструкций. Так, библиотеки моделирования FlopC++ [19], Optim [47] и JuMP [13] поддерживают описание моделей оптимизации с использованием объектно ориентированного проектирования на C++, Java и Julia соответственно. Эти библиотеки частично приносят в жертву интуитивно понятный математический синтаксис специализированного AMPL, зато позволяют воспользоваться всей гибкостью современных языков программирования высокого уровня. Их дополнительное преимущество заключается в том, что они непосредственно компонируются с высокопроизводительными библио-

теками оптимизации и решателями, что для некоторых приложений может быть существенно.

## 1.2. МОДЕЛИРОВАНИЕ НА PYOMO

Цель Pyomo – предоставить платформу для описания моделей оптимизации, которая воплощает идеи современных AML в контексте, обеспечивающем гибкость, расширяемость, переносимость, открытость и удобство сопровождения. Pyomo – это AML, расширяющий Python путем включения объектов для моделирования оптимизации [30]. Эти объекты можно использовать для задания моделей оптимизации и их преобразования в различные форматы, понятные внешним решателям.

Теперь мы приведем несколько примеров, иллюстрирующих использование Pyomo для описания моделей оптимизации.

### 1.2.1. Простые примеры

Рассмотрим следующую линейную программу (ЛП):

$$\begin{aligned} \min \quad & x_1 + 2x_2 \\ \text{при условиях} \quad & 3x_1 + 4x_2 \geq 1 \\ & 2x_1 + 5x_2 \geq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Эту ЛП легко выразить на Pyomo следующим образом:

```
import pyomo.environ as pyo

model = pyo.ConcreteModel()
model.x_1 = pyo.Var(within=pyo.NonNegativeReals)
model.x_2 = pyo.Var(within=pyo.NonNegativeReals)
model.obj = pyo.Objective(expr=model.x_1 + 2*model.x_2)
model.con1 = pyo.Constraint(expr=3*model.x_1 + 4*model.x_2 >= 1)
model.con2 = pyo.Constraint(expr=2*model.x_1 + 5*model.x_2 >= 2)
```

Первая строка – стандартное предложение импорта в Python, которое инициализирует окружение Pyomo и загружает библиотеку базовых компонентов моделирования. В следующей строке конструируется объект модели и определяются его атрибуты. В этом примере описывается *конкретная модель*. Компоненты модели – объекты, являющиеся атрибутами объекта модели, а объект ConcreteModel инициализирует каждый компонент модели при добавлении. Переменные, ограничения и целевая функция модели определяются с помощью *компонентов модели*.

Редко бывает так, что создается единственный экземпляр некоторой модели оптимизации, подлежащей решению. Чаще имеется общая модель оп-

тимизации, а затем создается конкретный экземпляр этой модели с определенными данными. Например, следующие уравнения представляют ЛП со скалярными параметрами  $n$  и  $m$ , векторными параметрами  $b$  и  $c$  и матричным параметром  $a$ :

$$\begin{aligned} \min \quad & \sum_{i=1}^n c_i x_i \\ \text{при условиях} \quad & \sum_{i=1}^n a_{ji} x_i \geq b_j \quad \forall j = 1 \dots m \\ & x_i \geq 0 \quad \forall i = 1 \dots n \end{aligned}$$

Эту ЛП можно выразить в виде конкретной модели на Pyomo следующим образом:

```
import pyomo.environ as pyo
import mydata

model = pyo.ConcreteModel()

model.x = pyo.Var(mydata.N, within=pyo.NonNegativeReals)

def obj_rule(model):
    return sum(mydata.c[i]*model.x[i] for i in mydata.N)
model.obj = pyo.Objective(rule=obj_rule)

def con_rule(model, m):
    return sum(mydata.a[m,i]*model.x[i] for i in mydata.N) \
           >= mydata.b[m]
model.con = pyo.Constraint(mydata.M, rule=con_rule)
```

Этот скрипт требует, чтобы при конструировании каждого компонента модели были доступны соответствующие данные. В нашем случае необходимые данные находятся в файле `mydata.py`:

```
N = [1,2]
M = [1,2]
c = {1:1, 2:2}
a = {(1,1):3, (1,2):4, (2,1):2, (2,2):5}
b = {1:1, 2:2}
```

Эту ЛП можно также рассматривать как абстрактную математическую модель, в которой неспецифицированные символические значения параметров определяются позже, на этапе инициализации модели. Например, эту ЛП можно следующим образом выразить в виде абстрактной модели на Pyomo:

```
import pyomo.environ as pyo

model = pyo.AbstractModel()

model.N = pyo.Set()
model.M = pyo.Set()
```

```

model.c = pyo.Param(model.N)
model.a = pyo.Param(model.M, model.N)
model.b = pyo.Param(model.M)
model.x = pyo.Var(model.N, within=pyo.NonNegativeReals)

def obj_rule(model):
    return sum(model.c[i]*model.x[i] for i in model.N)
model.obj = pyo.Objective(rule=obj_rule)

def con_rule(model, m):
    return sum(model.a[m,i]*model.x[i] for i in model.N) \
           >= model.b[m]
model.con = pyo.Constraint(model.M, rule=con_rule)

```

Этот пример включает компоненты модели, которые предоставляют абстрактные, или символические, определения множества и значений параметров. Объект `AbstractModel` откладывает инициализацию компонентов модели до момента создания *экземпляра модели*, когда передаются заданные пользователем данные множества и параметры. И конкретные, и абстрактные модели можно инициализировать данными из различных источников, включая файлы команд данных, например:

```

param : N : c :=
1 1
2 2 ;
param : M : b :=
1 1
2 2 ;

param a :=
1 1 3
1 2 4
2 1 2
2 2 5 ;

```

## 1.2.2. Пример раскраски графа

Мы продолжим иллюстрировать возможности моделирования Pyomo на примере простой хорошо известной задачи оптимизации: минимальной раскраски графа (известной также как раскраска вершин). В задаче о раскраске графа требуется назначить цвета вершинам графа, так чтобы никакие две смежные вершины не были окрашены в один цвет. У раскраски графов есть много практических применений, включая распределение регистров в компиляторах, планирование ресурсов и сопоставление с образцами, а кроме того, она лежит в основе развлекательных головоломок типа судоку.

Обозначим  $G = (V, E)$  граф с множеством вершин  $V$  и множеством ребер  $E \subseteq V \times V$ . Для данного  $G$  целью в задаче минимальной раскраски графа является нахождение допустимой раскраски минимальным числом цветов. Для

простоты предположим, что ребра из множества  $E$  упорядочены таким образом, что если  $(v_1, v_2) \in E$ , то  $v_1 < v_2$ . Обозначим  $k$  максимальное число цветов и определим множество возможных цветов  $C = \{1, \dots, k\}$ .

Задачу минимальной раскраски графа можно представить в виде следующей целочисленной программы (ЦП):

$$\begin{aligned}
 & \min y \\
 & \text{при условиях } \sum_{c \in C} x_{v,c} = 1 \quad \forall v \in V \\
 & \quad x_{v_1,c} + x_{v_2,c} \leq 1 \quad \forall (v_1, v_2) \in E \\
 & \quad y \geq c \cdot x_{v,c} \quad \forall v \in V, c \in C \\
 & \quad x_{v,c} \in \{0, 1\} \quad \forall v \in V, c \in C
 \end{aligned} \tag{1.1}$$

В этой формулировке переменная  $x_{v,c}$  равна 1, если вершина  $v$  окрашена цветом  $c$ , и 0 в противном случае, а  $y$  – число использованных цветов. Первое ограничение требует, чтобы каждая вершина была окрашена ровно одним цветом. Второе ограничение означает, что вершины, соединенные ребром, должны быть окрашены разными цветами. Третье ограничение определяет нижнюю границу  $y$  и гарантирует, что  $y$  не меньше числа цветов, использованных для раскраски. Четвертое, и последнее, ограничение означает, что  $x_{v,c}$  могут принимать только два значения.

На рис. 1.1 приведена формулировка описанной задачи раскраски графа в Pyomo с использованием конкретной модели; пример взят из работы Gross and Yellen [27]. Эта спецификация состоит из предложений Python, определяющих объект ConcreteModel, и последующего определения различных атрибутов этого объекта, включая переменные, ограничения и целевую функцию оптимизации. В строках 10–24 определены данные модели. Строка 28 – стандартное предложение импорта Python, которое вносит все символы (например, классы и функции), определенные в pyomo.environ, в текущее пространство имен. В строке 31 создается объект модели – экземпляр класса ConcreteModel. В строках 34 и 35 определены переменные модели. Отметим, что  $y$  – скалярная переменная, а  $x$  – двумерный массив переменных. В остальных строках определяются ограничения и целевая функция модели. Класс Objective определяет единственную целевую функцию с помощью именованного параметра expr. Класс ConstraintList представляет список ограничений, которые добавляются по одному.

По сравнению со специализированными AML, модели Pyomo, очевидно, более многословны (см., например, Hart et al. [30]). Однако этот пример иллюстрирует, что синтаксис Python все же позволяет выразить математические идеи лаконично и интуитивно понятно. Оставляя в стороне классы Pyomo, в этом примере нет ничего, кроме стандартного синтаксиса и методов Python. Например, в строке 4 используются генераторы Python для обхода всех элементов множества colors и применения к ним функции sum. В Pyomo имеются некоторые служебные функции, позволяющие упростить конструирование выражений, но никаких хитроумных расширений базовой функциональности Python не требуется.



```

1 #
2 # Пример раскраски графа, написанный на основе программы из книги
3 #
4 # Jonathan L. Gross and Jay Yellen,
5 # "Graph Theory and Its Applications, 2nd Edition",
6 # Chapman & Hall/CRC, Boca Raon, FL, 2006.
7 #
8
9 # Определить данные графа
10 vertices = set (['Ar', 'Bo', 'Br', 'Ch', 'Co', 'Ec',
11                 'FG', 'Gu', 'Pa', 'Pe', 'Su', 'Ur', 'Ve'])
12
13 edges = set ((['FG', 'Su'), ('FG', 'Br'), ('Su', 'Gu'),
14               ('Su', 'Br'), ('Gu', 'Ve'), ('Gu', 'Br'),
15               ('Ve', 'Co'), ('Ve', 'Br'), ('Co', 'Ec'),
16               ('Co', 'Pe'), ('Co', 'Br'), ('Ec', 'Pe'),
17               ('Pe', 'Ch'), ('Pe', 'Bo'), ('Pe', 'Br'),
18               ('Ch', 'Ar'), ('Ch', 'Bo'), ('Ar', 'Ur'),
19               ('Ar', 'Br'), ('Ar', 'Pa'), ('Ar', 'Bo'),
20               ('Ur', 'Br'), ('Bo', 'Pa'), ('Bo', 'Br'),
21               ('Pa', 'Br')])
22
23 ncolors = 4
24 colors = range(1, ncolors+1)
25
26
27 # Предложение импорта Python
28 import pyomo.environ as pyo
29
30 # Создать объект модели Pyomo
31 model = pyo.ConcreteModel()
32
33 # Определить переменные модели
34 model.x = pyo.Var(vertices, colors, within=pyo.Binary)
35 model.y=pyo.Var()
36
37 # Каждая вершина окрашивается одним цветом
38 model.node_coloring = pyo.ConstraintList()
39 for v in vertices:
40     model.node_coloring.add(
41         sum(model.x[v,c] for c in colors) == 1)
42
43 # Вершины, соединенные ребром, нельзя окрашивать одним цветом
44 model.edge_coloring = pyo.ConstraintList()
45 for v,w in edges:
46     for c in colors:
47         model.edge_coloring.add(
48             model.x[v,c] + model.x[w,c] <= 1)
49
50 # Задать нижнюю границу минимального числа потребных
51 # цветов
52 model.min_coloring = pyo.ConstraintList()
53 for v in vertices:
54     for c in colors:
55         model.mincoloring.add(
56             model.y>=c_model.x[v,c])
57
58 # Минимизировать число потребных цветов
59 model.obj = pyo.Objective(expr=model.y)

```

**Рис. 1.1** ❖ Конкретная модель Pyomo для задачи минимальной раскраски графа

## 1.2.3. Ключевые особенности Pyomo

### *Python*

Понятный синтаксис Python позволяет Pyomo выражать математические идеи интуитивно понятно и лаконично. Кроме того, выразительные средства программирования Python можно использовать для формулирования сложных моделей и определения высокоуровневых решателей, обращающихся к высокопроизводительным библиотекам оптимизации. Python предоставляет развитые возможности написания скриптов и позволяет пользователям анализировать модели и решения Pyomo, привлекая весь потенциал сторонних библиотек (например, `numpy`, `scipy` и `matplotlib`). Наконец, поскольку Pyomo погружена в Python, пользователи могут изучать ее базовый синтаксис, пользуясь обширной документацией по Python.

### *Настраиваемые возможности*

Pyomo спроектирована с учетом модели разработки по принципу «каши из топора», когда каждый разработчик делает то, что ему хочется. Ключевым элементом такого дизайна является каркас на основе плагинов, который Pyomo использует для интегрирования компонентов модели, преобразований модели, решателей и диспетчеров решателей. Каркас управляет регистрацией этих возможностей. Таким образом, пользователи могут настраивать Pyomo модульным образом, не опасаясь дестабилизировать базовую функциональность.

### *Командные инструменты и скрипты*

Модели Pyomo можно анализировать с помощью командных утилит или скриптов на Python. Командная утилита `pyomo` предоставляет обобщенный интерфейс к большинству возможностей моделирования Pyomo и реализует обобщенный процесс оптимизации. Этот процесс можно легко реализовать Python-скриптом и настроить под конкретные потребности пользователя.

### *Определение конкретных и абстрактных моделей*

Примеры в разделе 1.2.1 иллюстрируют поддержку конкретных и абстрактных моделей Pyomo. Разница между этими подходами к моделированию заключается в том, когда именно инициализируются компоненты модели: в конкретных моделях компоненты инициализируются сразу, а в абстрактных инициализация откладывается до момента инициализации модели. Следовательно, оба подхода эквивалентны, а выбор зависит от контекста, в котором используется Pyomo, и от вкусов пользователя. Модели обоих типов легко инициализировать данными из самых разных источников (например, файлов в форматах `csv`, `json`, `yaml`, `excel` и баз данных).

## **Объектно ориентированный дизайн**

В Pyomo применяется объектно ориентированный подход к проектированию библиотек. Модель – это объект Python, а компоненты модели – атрибуты этого объекта. Такой дизайн позволяет Pyomo автоматически управлять именованием компонентов модели и естественно разделяет компоненты объектов разных моделей. Структуру моделей Pyomo можно уточнять с помощью блоков, т. е. поддерживается иерархическая вложенность компонентов модели. Многие продвинутые средства моделирования Pyomo основаны на таком структурном моделировании.

## **Выразительные возможности моделирования**

Компоненты моделей Pyomo можно использовать для формулирования широкого круга задач оптимизации, в том числе:

- линейные программы;
- квадратичные программы;
- нелинейные программы;
- смешанно-целочисленные линейные программы;
- смешанно-целочисленные квадратичные программы;
- обобщенные дизъюнктивные программы;
- смешанно-целочисленные стохастические программы;
- динамические задачи с дифференциальными алгебраическими уравнениями;
- математические программы с ограничениями равновесия.

## **Интеграция с решателями**

Pyomo поддерживает тесно и слабо связанные интерфейсы с решателями. Тесно связанные инструменты моделирования обращаются к библиотечным решателям задач оптимизации непосредственно (например, путем статической или динамической компоновки), а слабо связанные вызывают внешние исполняемые файлы (например, посредством системных вызовов). Многие решатели задач оптимизации читают данные задач в хорошо известных форматах (например, в формате AMPL n1 [24]); они слабо связаны с Pyomo. Решатели, имеющие интерфейсы к Python (например, Gurobi и CPLEX), могут быть тесно связаны, что позволяет избежать записи данных во внешние файлы.

## **Открытый исходный код**

Pyomo разрабатывается как проект с открытым исходным кодом, чтобы сделать процесс проектирования и реализации ПО максимально прозрачным. Pyomo распространяется на условиях лицензии BSD [8], которая налагает меньше ограничений на использование в коммерческих или государственных организациях. Управление исходным кодом Pyomo осуществляется с по-

мощью GitHub [53] и проекта COIN-OR [9]. Списки рассылки для разработчиков и пользователей ведутся с помощью Google Groups. Растет количество свидетельств в пользу того, что надежность ПО с открытым исходным кодом не уступает надежности закрытого коммерческого ПО [3, 59], и при разработке Puomo ведется тщательный контроль для обеспечения стабильности и надежности программы.

## 1.3. ПОДГОТОВИТЕЛЬНЫЕ ДЕЙСТВИЯ

Для выполнения примеров, приведенных в книге, необходимо установить следующее программное обеспечение:

- Python версии 3.6 или старше (хотя почти все примеры работают и с более ранними версиями Python). В настоящее время Puomo опирается на CPython; для Jython и PyPy поддерживается лишь часть функциональности;
- Puomo 6.0, именно эта версия используется в книге;
- решатель GLPK [25], который используется для генерирования вывода большинства примеров. Можно использовать и другие решатели для задач линейного и смешанно-целочисленного линейного программирования, но GLPK легко устанавливается и широко доступен;
- решатель IPOPT [34], который используется для генерирования вывода в примерах нелинейных моделей. Можно использовать и другие нелинейные оптимизаторы, если они собраны с библиотекой AMPL Solver Library [23];
- решатель CPLEX [11], который используется для генерирования вывода в примерах стохастического программирования. Этот коммерческий решатель предоставляет возможности, которые отсутствуют в решателях задач оптимизации с открытым исходным кодом (например, оптимизацию квадратичных целочисленных программ);
- Python-пакет `matplotlib` для построения графиков.

Инструкции по установке Puomo имеются на сайте Puomo по адресу [www.puomo.org](http://www.puomo.org). В приложении А приведено краткое пособие по языку программирования Python; в различных онлайн-источниках можно найти более полные пособия и документацию.

## 1.4. КРАТКИЙ ОБЗОР КНИГИ

Оставшаяся часть книги разделена на три части. Первая часть – введение в Puomo. Главу 2 можно рассматривать как очень краткий курс оптимизации и математического моделирования, включающий примеры применения Puomo к формулированию и решению алгебраических моделей оптимизации. Глава 3 иллюстрирует возможности моделирования Puomo на примере

простых конкретных и абстрактных моделей, а в главе 4 описываются базовые компоненты моделирования `Pyomo`. Основы вложения моделей `Pyomo` в скрипты на Python излагаются в главе 5. И завершается первая часть главой 6, в которой описывается взаимодействие с решателями.

Во второй части книги документированы продвинутые возможности и расширения. В главе 7 описываются средства нелинейного программирования `Pyomo`, а в главе 8 – построение иерархических моделей. Глава 9 содержит рекомендации по повышению производительности. В главе 10 описан класс `AbstractModel`, синтаксис командных файлов данных и командный интерфейс к `Pyomo`.

Третья часть книги посвящена расширениям моделирования. В главе 11 приведен обзор обобщенного дизъюнктивного программирования. В главе 12 описаны динамические модели, выражаемые дифференциальными и алгебраическими уравнениями, а в главе 13 – программы с ограничениями равновесия.

**ПРИМЕЧАНИЕ** Эта книга не является полным справочным руководством по `Pyomo`. Наша цель – обсудить базовую функциональность, имеющуюся в версии `Pyomo` 6.0.

## 1.5. ОБСУЖДЕНИЕ

Многие разработчики пришли к выводу, что понятный синтаксис Python и богатый набор библиотек – отличная почва для моделирования оптимизации [30]. Средства моделирования реализованы в различных пакетах программ на Python, например `PuLP` [49], `AMPLpy` [4] и `OpenOpt` [46]. Кроме того, существует немало пакетов, реализующих интерфейсы с решателями, в т. ч. такие пакеты с открытым исходным кодом, как `PyGlpk` [50] и `pyipopt` [51], в дополнение к Python-интерфейсам таких коммерческих решателей, как `CPLEX` [11] и `Gurobi` [28].

У `Pyomo` есть несколько отличительных особенностей. Во-первых, `Pyomo` предоставляет механизмы для расширения базовой функциональности моделирования и оптимизации без внесения изменений в код самой `Pyomo`. Во-вторых, `Pyomo` поддерживает определение конкретных и абстрактных моделей. Это дает пользователю большую гибкость при решении вопроса о том, насколько тесно данные должны быть интегрированы с определением модели. Наконец, `Pyomo` поддерживает широкий класс моделей оптимизации, включая стандартные линейные программы, общие нелинейные модели, обобщенные дизъюнктивные программы, задачи, описываемые дифференциальными уравнениями, и математические программы с условиями равновесия.

Часть **I**



# **ВВЕДЕНИЕ В РУОМО**

# Глава 2

## Математическое моделирование и оптимизация

В этой главе объясняются начала оптимизации и математического моделирования. Вы не найдете здесь полного описания этих тем, а лишь информацию, необходимую для чтения книги. Более подробное обсуждение методов оптимизации см. в работе Williams [58]. Приведены реализации простых моделей, чтобы читатель получил первое представление о том, как используется Puomo.

### 2.1. МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ

#### 2.1.1. Общие сведения

Моделирование – фундаментальный процесс во многих аспектах научных исследований, техники и бизнеса. Под моделированием понимается создание упрощенного представления системы или объекта материального мира. Такое упрощение позволяет структурировать знания об исходной системе, чтобы проанализировать результирующую модель. Шихль (Schichl [56]) отмечает, что модели используются для следующих целей:

- **объяснить явления**, возникающие в системе;
- **предсказать** будущие состояния системы;
- **оценить ключевые факторы**, влияющие на происходящее в системе;
- **идентифицировать экстремальные состояния** системы, возможно, представляющие худшие случаи или планы с минимальной стоимостью;
- **проанализировать компромиссы** и предложить их лицам, принимающим решения.

Кроме того, структура представления модели упрощает передачу ассоциированных с моделью знаний. Например, одним из ключевых аспектов модели является уровень детализации, который отражает знания о системе, необходимые для использования модели в приложении.

Математика всегда играла важнейшую роль в представлении и формулировании наших знаний. Математическое моделирование становилось все более формальным по мере разработки новых каркасов для описания сложных систем. В современном моделировании центральными являются следующие математические понятия:

- **переменные:** представляют *неизвестные* или изменяющиеся части модели (например, принимаемые решения или характеристики выхода системы);
- **параметры:** символические представления реальных данных, могут варьироваться для различных экземпляров задачи или сценариев;
- **отношения:** *равенства, неравенства* и другие математические связи, описывающие, как различные части модели соотносятся между собой.

Модели оптимизации – это математические модели, в которых функции представляют цели, ради достижения которых моделируется система. Модели оптимизации можно анализировать, исследуя различные компромиссы, позволяющие находить решения, оптимизирующие целевые функции системы. Следовательно, эти модели можно использовать в разнообразных научных, технических и деловых приложениях.

## 2.1.2. Пример моделирования

*Модель* в том смысле, в котором мы употребляем это слово, представляет объекты, абстрагируя некоторые свойства. Все мы знакомы с физическими моделями, например игрушечными железными дорогами или моделями автомобилей. Но нас интересуют математические модели, в которых для представления аспектов системы или объекта материального мира используются символы.

Например, человеку интересно определить, сколько шариков мороженого лучше всего купить. Обозначим символом  $x$  число шариков, а символом  $c$  – цену одного шарика. Тогда полную стоимость можно смоделировать как произведение  $c$  на  $x$ , обычно обозначаемое  $cx$ .

Может понадобиться более сложная модель, если имеются скидки за объем заказа или наценки за неполные шарик. Скорее всего, эта модель не годится для отрицательных  $x$ . Редко удастся продать шарик мороженого за ту же цену, за которую купил.

Труднее математически смоделировать счастье от покупки шариков мороженого в вафельном рожке. Один из возможных подходов – измерять счастье пропорционально. Определим единицу счастья, ассоциированную с одним шариком, и обозначим ее  $h$ . Тогда можно смоделировать счастье от покупки  $x$  шариков как произведение  $h$  на  $x$ , или  $hx$ . Для кого-то такая модель является неплохим приближением для значений  $x$  от  $\frac{1}{2}$  до 3, но вряд ли кто-то станет



в сто раз счастливее, получив в рожке 100 шариков мороженого, когда ему достаточно одного. Для некоторых людей модель счастья для значений  $x$  от 0 до 10 может выглядеть так:

$$h \cdot (x - (x/5)^2).$$

Заметим, что в этой модели счастье становится отрицательным, когда в рожке больше 25 шариков, поэтому не для всех она хороша.

Обычно хотят моделировать сразу несколько вещей. Например, к шарикам мороженого могут прилагаться орешки. Поскольку можно купить несколько вкусовостей, купленное представляется вектором  $x$  (т. е. символ  $x$  теперь представляет список). *Элементы* списка обозначаются  $x_i$ , где  $i$  – индекс элемента. Например, если мы примем, что первый элемент – число шариков мороженого, то на это число можно ссылаться с помощью  $x_1$ . При большем числе измерений в качестве индекса используется *кортеж*, обозначаемый  $i, j$  или  $(i, j)$ .

Пусть  $c$  – вектор цен с такими же индексами, как у  $x$  (т. е.  $c_1$  – цена одного шарика мороженого, а  $c_2$  – цена одной порции орешков). Тогда полная стоимость мороженого и орешков равна

$$c_1 x_1 + c_2 x_2 = \sum_{i=1}^2 c_i x_i.$$

И эта модель стоимости, вероятно, годится не для всех возможных значений элементов  $x$ , но для каких-то целей она достаточно хороша.

Часто бывает полезно брать индексы из некоторого множества. Для рассмотренного выше примера можно было бы взять множество  $\{1, 2\}$  и записать полную стоимость в виде

$$\sum_{i \in \{1, 2\}} c_i x_i,$$

но обычно используется более абстрактное выражение вида

$$\sum_{i \in \mathcal{A}} c_i x_i,$$

где под  $\mathcal{A}$  понимается множество индексов элементов векторов  $c$  и  $x$  (в нашем примере  $\mathcal{A}$  будет равно  $\{1, 2\}$ ).

Помимо суммирования по множеству индексов, нам иногда нужны условия, выполняющиеся для всех индексов. В этом случае используется запятая. Например, если мы хотим, чтобы никакой элемент  $x$  не был отрицательным числом, то можем написать:

$$x_i \geq 0, i \in \mathcal{A}$$

– это читается как « $x$   $i$ -е больше либо равно нулю для всех  $i$ , принадлежащих  $\mathcal{A}$ ».

Ни в математике, ни в математическом моделировании нет такого закона, чтобы использовать только однобуквенные символы, например  $x, c$  или  $i$ . Никто не запрещает в качестве элементов  $\mathcal{A}$  использовать изображения рожка мороженого и чашки орешков, но работать с такими обозначениями было бы трудно. Множество может быть и таким –  $\{Scoops, Cups\}$ , но в книгах так

обычно не делается, потому что занимает слишком много места и пришлось бы переносить строки. Точно так же  $x$  можно было бы заменить словом *Quantity* или чем-то подобным. Длинные имена поддерживаются такими языками моделирования, как Ruoто, и, вообще говоря, использовать в моделях осмысленные имена – здравая мысль. Пробелы или знаки минуса в именах часто приводят к проблемам и путанице, поэтому обычно вместо них используются знаки подчеркивания.

## 2.2. ОПТИМИЗАЦИЯ

Символ  $x$  часто используется как *переменная* при моделировании оптимизации. Иногда он называется *переменной решения*, потому что модели оптимизации и строятся для того, чтобы принимать решения. Это может запутать людей, знакомых со статистическим моделированием. Там символ  $x$  часто используется для обозначения данных, и компьютеру передаются значения  $x$ , чтобы он вычислил статистики, тогда как в моделях оптимизации компьютеру передаются другие данные, а требуется вычислить хорошие значения  $x$ . Конечно, никто не заставляет использовать именно букву  $x$ , но в учебниках и вводных курсах обычно выбирают ее.

Такие значения, как стоимость (мы обозначали ее символом  $c$ ), называются *данными*, или *параметрами*. Модель оптимизации можно описать, не определяя значений параметров, но в конкретном оптимизируемом экземпляре значения данных должны быть определены, иногда мы называем их *данными экземпляра*.

Чтобы можно было произвести оптимизацию, в модели должна быть определена *целевая функция*. При оптимальных значениях переменных решения получается наилучшее возможное значение целевой функции. Важно отметить, что мы говорим «оптимальные значения» (во множественном числе), потому что часто бывает так, что наилучшее значение целевой функции достигается при нескольких наборах значений переменных. Эта функция обычно записывается в очень абстрактной форме, например  $f(x)$ . Что понимать под «наилучшим» значением – наибольшее или наименьшее, определяется типом оптимизации: *максимизация* или *минимизация*.

Предположим, к примеру, что  $x$  – не вектор, а скаляр, равный числу шариков мороженого. Если воспользоваться описанной выше моделью счастья, то

$$f(x) \equiv h \cdot (x - (x/5)^2),$$

где  $h$  – данные. (На самом деле в этом конкретном примере значение  $h$  совершенно не важно для нахождения  $x$ , максимизирующего счастье.) Смоделированная нами задача оптимизации записывается в виде

$$\max h \cdot (x - (x/5)^2),$$

но склонные к строгости авторы пишут так:

$$\max_x h \cdot (x - (x/5)^2),$$

чтобы было понятно, что  $x$  – переменная решения. В данном случае существует всего одно наилучшее значение  $x$ , которое можно найти методами численной оптимизации. Это значение оказывается дробным, т. е. придется купить нецелое число шариков. Такую модель не назовешь полезной для типичной лавки мороженого, где купить можно только целое неотрицательное число шариков. Чтобы включить это требование, мы добавим в модель оптимизации *ограничение*:

$$\begin{aligned} & \max_x h \cdot (x - (x/5)^2) \\ & \text{при условии} \\ & \quad x \in \text{множеству целых неотрицательных чисел.} \end{aligned}$$

Предположим, что эта модель используется не в лавке, а дома, где мороженое подает родитель пользователя модели. Если родитель готов делить шарики на части, но не подает больше двух шариков, то ограничение

$$x \in \text{множеству целых неотрицательных чисел}$$

следует заменить таким:

$$0 \leq x \leq 2.$$

Эта модель неидеальна, потому что не все дробные значения  $x$  осмыслены.

Чтобы проиллюстрировать рассмотренные до сих пор аспекты модели, вернемся к нескольким продуктам, которые описываются множеством индексов  $\mathcal{A}$ , так что  $x$  – вектор. Воспользуемся следующей моделью счастья для продукта с индексом  $i$ :

$$h_i \cdot (x_i - (x_i/d_i)^2),$$

где  $h$  и  $d$  – векторы данных с таким же множеством индексов, как у  $x$ . Пусть далее  $c$  – вектор цен, а  $u$  – вектор максимальных количеств продуктов, которые можно приобрести. Предположим временно, что для любого продукта купленное количество может быть дробным. И наконец, предположим, что бюджет ограничен величиной  $b$ . Тогда задачу оптимизации можно записать в виде:

$$\begin{aligned} & \max_x \sum_{i \in \mathcal{A}} h_i \cdot (x_i - (x_i/d_i)^2) \quad (H) \\ & \text{при условии} \quad \sum_{i \in \mathcal{A}} c_i x_i \leq b \\ & \quad 0 \leq x_i \leq u_i, i \in \mathcal{A} \end{aligned}$$

Иногда последнее ограничение записывают в виде двух:

$$\begin{aligned} & \max_x \sum_{i \in \mathcal{A}} h_i \cdot (x_i - (x_i/d_i)^2) \quad (H) \\ & \text{при условии} \quad \sum_{i \in \mathcal{A}} c_i x_i \leq b \\ & \quad x_i \leq u_i, i \in \mathcal{A} \\ & \quad x_i \geq 0, i \in \mathcal{A} \end{aligned}$$

Принято помещать сокращенное имя модели в скобках в одной строке с целевой функцией. Имя (P) очень распространенное, но мы использовали (H) как сокращение от «happiness» (счастье). По имени (H) мы будем ссылаться на эту модель далее в данной главе, когда покажем, как реализовать ее в Pyomo и решить.

## 2.3. МОДЕЛИРОВАНИЕ В PYOMO

Теперь рассмотрим различные стратегии формулирования и оптимизации алгебраических моделей в Pyomo. Детальное объяснение моделей Pyomo мы отложим до главы 3, а здесь приведем только примеры использования Pyomo для модели (H).

### 2.3.1. Конкретная формулировка

*Конкретная модель* Pyomo инициализирует компоненты по мере их конструирования. Это позволяет использовать встроенные в Python структуры данных при определении экземпляра модели. Существует много способов реализовать нашу модель в виде конкретной модели Pyomo, и начнем мы с использования списков и словарей Python.

**ПРИМЕЧАНИЕ** Понимая, что нам часто придется создавать экземпляры моделей с разными данными, мы решили написать функцию, которая принимает требуемые данные в качестве аргументов и возвращает модель Pyomo. При таком подходе мы сможем повторно использовать общую модель Pyomo с разными данными.

```
import pyomo.environ as pyo

def IC_model(A, h, d, c, b, u):

    model = pyo.ConcreteModel(name = "(H)")

    def x_bounds(m, i):
        return (0,u[i])
    model.x = pyo.Var(A, bounds=x_bounds)

    def z_rule(model):
        return sum(h[i] * (model.x[i] - (model.x[i]/d[i])**2)
                   for i in A)
    model.z = pyo.Objective(rule=z_rule, sense=pyo.maximize)

    model.budgetconstr = pyo.Constraint(\
        expr = sum(c[i]*model.x[i] for i in A) <= b)

    return model
```

В объявлении `budgetconstr` мы определяем ограничение непосредственно с помощью именованного аргумента `expr`, хотя можно было бы использо-

вать и правило конструирования. Пример правила конструирования показан в объявлении целевой функции. Но при желании мы могли бы и там использовать именованный аргумент `exrg`.

**ПРИМЕЧАНИЕ** В Python символ обратной косой черты в конце строки означает, что предложение продолжается на следующей строке; мы пользуемся этим приемом, принимая во внимание ограничения на длину печатной строки в книге. В данном случае без символа продолжения можно было бы обойтись, потому что строка разрывается после круглой скобки.

Существуют и более элегантные способы написать функцию `IC_Model`, которые мы рассмотрим ниже. Имея конкретные данные, мы можем написать Python-программу, которая передает эти данные функции и получает полностью инициализированную модель `Pyomo`. Если на компьютере установлен решатель, то эта программа может отправить ему модель и в случае успеха запросить решение. Но прежде чем переходить к этим шагам, поговорим о других способах написать функцию `IC_model`.

Заметим, что `IC_model` – обычная функция Python. Модель `Pyomo` можно было бы реализовать просто как Python-программу, вызывающую решатель, или в виде функции `IC_model`, которая принимает в качестве единственного аргумента словарь, а не явный список аргументов. Программист, наверное, смог бы придумать и другие, более удачные способы написать такой код на Python.

Помимо рассмотренных выше компонентов моделирования, `Pyomo` предлагает класс множества (`Set`) и класс параметров (`Param`), которые мы обсудим в последующих главах. Ниже определена функция `IC_model_dict`, которая принимает словарь Python и определяет ту же самую модель, используя объекты типа `Set` и `Param`.

```
import pyomo.environ as pyo

def IC_model_dict(ICD):
    # ICD - словарь, содержащий данные для задачи

    model = pyo.ConcreteModel(name = "(H)")

    model.A = pyo.Set(initialize=ICD["A"])
    model.h = pyo.Param(model.A, initialize=ICD["h"])
    model.d = pyo.Param(model.A, initialize=ICD["d"])
    model.c = pyo.Param(model.A, initialize=ICD["c"])
    model.b = pyo.Param(initialize=ICD["b"])
    model.u = pyo.Param(model.A, initialize=ICD["u"])

    def xbounds_rule(model, i):
        return (0, model.u[i])
    model.x = pyo.Var(model.A, bounds=xbounds_rule)

    def obj_rule(model):
        return sum(model.h[i] * \
                   (model.x[i] - (model.x[i]/model.d[i])**2)\
                   for i in model.A)
```

```

model.z = pyo.Objective(rule=obj_rule,sense=pyo.maximize)

def budget_rule(model):
    return sum(model.c[i]*model.x[i]\
               for i in model.A) <= model.b
model.budgetconstr = pyo.Constraint(rule=budget_rule)

return model

```

## 2.4. ЛИНЕЙНЫЕ И НЕЛИНЕЙНЫЕ МОДЕЛИ ОПТИМИЗАЦИИ

### 2.4.1. Определение

Говорят, что выражение в модели оптимизации линейно, если оно включает только суммы переменных решения и произведения переменных решения на данные. Таким образом, линейное выражение – это непостоянная линейная функция от переменных решения. Пусть  $x$  – векторная переменная,  $c$  – вектор данных, и оба они индексированы множеством  $\mathcal{A}$ . И пусть 2 и 3 – элементы  $\mathcal{A}$ . Следующие выражения линейны:

$$\begin{aligned}
& \sum_{i \in \mathcal{A}} c_i x_i \\
& \sum_{i \in \mathcal{A}} x_i \\
& x_2 \\
& c_3 x_2 + c_2 x_3 \\
& c_3 x_2 + c_2 x_3 + 4
\end{aligned}$$

С другой стороны, следующие выражения нелинейны:  $x_i^2$ ,  $x_2 x_3$  и  $\cos(x_2)$ .

Линейные выражения часто приводят к задачам, для решения которых нужно значительно меньше вычислительных ресурсов, чем в случае нелинейных выражений. Поэтому многие авторы стремятся по возможности использовать линейные выражения, а некоторые из кожи вон лезут, чтобы не было ничего, кроме линейных выражений. Кроме того, часто ищут линейные аппроксимации нелинейных моделей, чтобы получить «достаточно хорошие» решения для исходной нелинейной модели.

Для иллюстрации рассмотрим следующую линейную аппроксимацию модели (H), в которой целевая функция заменена такой:

$$\max_x \sum_{i \in \mathcal{A}} h_i \cdot (1 - u_i/d_i^2) x_i, \quad (2.1)$$

где  $u_i$  – новый параметр модели. Это выражение действительно линейно, потому что переменные решения только умножаются на данные и складываются. Правда, параметр  $d$  возводится в квадрат, но это не переменная решения. Числовое значение всего выражения

$$h_i \cdot (1 - u_i/d_i^2)$$

вычисляется Pyomo до передачи экземпляра задачи решателю, а решатель должен найти оптимальные значения переменных решения.

## 2.4.2. Линейная версия

Если мы хотим модифицировать конкретную модель, приведенную на стр. 34, используя в ней выражение (2.1), то должны изменить правило выражения целевой функции следующим образом:

```
def obj_rule(model):
    return sum(h[i]*(1 - u[i]/d[i]**2) * model.x[i] \
              for i in A)
```

## 2.5. РЕШЕНИЕ МОДЕЛИ PYOMO

Pyomo предлагает методы для (1) объединения модели и данных, (2) передачи результирующего экземпляра модели решателю и (3) получения результатов для отображения и дальнейшего использования. Но Pyomo не занимается решением задач оптимизации самостоятельно. Это ответственность решателей.

### 2.5.1. Решатели

Pyomo можно установить вообще без решателей. Например, Pyomo может просто записывать экземпляры задач в файлы, которые затем будут переданы какому-то решателю. Так поступают, если решатель работает на другом компьютере. Но обычно решатель устанавливается вместе с Pyomo, и в большинстве примеров это предполагается.

Напомним, что целевая функция в модели (Н) не является линейной функцией переменной  $x$ , а ограничение на бюджет линейно. Многие решатели умеют решать задачу с квадратичной целевой функцией и линейными ограничениями, но не все. Если единственный установленный на компьютере решатель ограничен только линейными задачами, то (Н) следует аппроксимировать линейной моделью.

### 2.5.2. Python-скрипты

Python-скрипт запускается из командной строки или из среды разработки. Как и в случае модели, есть много способов написать скрипт и подать ему данные, мы опишем их в следующих разделах. Например, скрипт, определяющий конкретную модель из раздела 2.3.1, может выглядеть так:

```
A = ['I_C_Scoops', 'Peanuts']
h = {'I_C_Scoops': 1, 'Peanuts': 0.1}
d = {'I_C_Scoops': 5, 'Peanuts': 27}
c = {'I_C_Scoops': 3.14, 'Peanuts': 0.2718}
b = 12
u = {'I_C_Scoops': 100, 'Peanuts': 40.6}

model = IC_model_linear(A, h, d, c, b, u)

opt = pyo.SolverFactory('glpk')
results = opt.solve(model) # решает и обновляет модель
pyo.assert_optimal_termination(results)

model.display()
```

Если назвать этот файл `ConcHLinScript.py`, то из командной строки он запускается следующим образом:

```
python ConcHLinScript.py
```

Первые несколько строк, в которых переменным Python присваиваются значения, выглядят немного странно, но удивляться тут нечему. Обычно данные для задач оптимизации читаются из файлов или базы данных, но здесь мы присваиваем литеральные значения, чтобы не вводить дополнительных зависимостей. В последних строках создается решатель, находится решение задачи и отображается модель с найденными значениями. Функция `assert_optimal_termination` останавливает скрипт и выводит сообщение, если решатель не смог найти оптимальное решение. Родственная ей функция `check_optimal_termination` возвращает `True`, если решатель нашел оптимальное решение, и `False` в противном случае.