КРАТКОЕ ВВЕДЕНИЕ В AMPL - СОВРЕМЕННЫЙ АЛГЕБРАИЧЕСКИЙ ЯЗЫК МОДЕЛИРОВАНИЯ

(препринт)

О.А. Щербина

2012 г.

1. Общие сведения

1.1. *Автоматизация построения математических моделей оптимизации.*

В настоящее время методология математического моделирования интенсивно развивается, охватывая все новые сферы – от разработки больших технических систем и управления ими до анализа сложнейших экономических и социальных процессов, поднимая общий уровень теоретических исследований, давая возможность проводить их в более тесной связи с экспериментальными исследованиями.

Математическое моделирование может рассматриваться как новый общий способ исследования объектов реального мира - метод познания, конструирования, проектирования, который сочетает в себе многие достоинства как теории, так и эксперимента [1]. Работа не с самим объектом (явлением, процессом), а с его моделью дает возможность без ущерба для моделируемого объекта, относительно быстро и без существенных затрат исследовать его свойства и поведение в интересующих исследователя ситуациях. В то же время вычислительные (компьютерные, имитационные) эксперименты с моделями объектов позволяют, опираясь на мощь современных вычислительных методов и технических инструментов информатики, изучать объекты в достаточной полноте, недоступной чисто теоретическим подходам.

Математические модели применяются в системах поддержки принятия решений, позволяющих непрофессиональному пользователю компьютера использовать современные математические модели и методы при поиске оптимального решения[1].

---

[1] Сараев А. Д., Щербина О. А. Системный анализ и современные информационные технологии // Труды Крымской академии наук. - Симферополь: СОНАТ, 2006. - С. 47-59.

Основу математического моделирования составляет триада модель - алгоритм – программа [1]. Рассмотрим подробнее указанные компоненты. На **первом этапе** вычислительного эксперимента выбирается (или строится) модель исследуемого объекта, отражающая в математической форме важнейшие его свойства - законы, которым он подчиняется, связи между его составляющими элементами, и т. д. Под моделью при этом понимается «"эквивалент" объекта, отражающий в математической форме важнейшие его свойства - законы, которым он подчиняется, связи, присущие составляющим его частям, и т.д.» ([1], с.8). Математическая модель (ее основные фрагменты) исследуется традиционными аналитическими средствами вычислительной математики для получения предварительных (обычно качественных) знаний об объекте.

Второй этап связан с выбором (или разработкой) вычислительного алгоритма для реализации модели на компьютере. Необходимо получить искомые величины с заданной точностью на имеющейся вычислительной технике. Вычислительные алгоритмы должны не искажать основные свойства модели и, следовательно, исходного объекта, они должны быть адаптирующимися к особенностям решаемых задач и используемых вычислительных средств.

На **третьем этапе** создается программное обеспечение (решатель) для реализации модели и алгоритма на компьютере. Программный продукт должен учитывать специфику математического моделирования, связанную с использованием системы математических моделей, многовариантностью расчетов.

Таким образом, разработка программ, переводящих модель и алгоритм на понятный компьютеру язык, завершает создание рабочего инструмента исследователя.

Готовая триада тестируется в «пробных» экспериментах. На этом этапе посредством цепочки усложнений (иерархии все более полных моделей) обеспечивается ее адекватность. После этого можно переходить к «опытам», дающим все требуемые качественные и количественные свойства и характеристики объекта.

Само по себе построение моделей представляет собой «применение фундаментальных законов природы, вариационных принципов, аналогий, иерархических цепочек» ([1], с.11), а процесс построения моделей включает в себя следующие этапы ([1], с.25-26):

1. «Словесно-смысловое описание объекта или явления» («формулировка предмодели»);
2. «Завершение идеализации объекта» и упрощение описания;
3. Переход «к выбору или формулировке закона (вариационного принципа, аналогии и т.п.)» и его записи в математической форме;
4. Завершает формулировку модели ее «оснащение» (задание начального состояния и параметров объекта). Этот этап особенно важен, поскольку: «...наконец, формулируется цель исследования модели ...»;
5. Модель изучается всеми доступными методами (в том числе с применением различных подходов и вычислительных методов);
6. В результате исследования математической модели достигается поставленная цель. При этом «должна быть установлена всеми возможными способами (сравнением с практикой, сопоставлением с другими подходами) ее адекватность - соответствие объекту и сформулированным предположениям...».

Заметим, что описанные выше этапы 1-2 моделирования часто называются в литературе построением *концептуальной модели*[2].

[2] Рыжаков А.Н., Щербина О.А. Современные проблемы математического моделирования в исследовании операций // Динамические системы. - 2006. - Вып. 21. - С. 115-129.

Концептуальное моделирование определяет, анализирует и описывает необходимые понятия и ограничения области приложения с помощью некоторого (обычно диаграммного) языка моделирования, основанного на небольшом числе мета-понятий (образующих мета-модель). Цель концептуального моделирования - сделать объекты реального мира доступными на абстрактном уровне, а также помочь исследователю в исключении не относящихся к делу структур путем построения отношений между идеализированными элементами, фокусируясь лишь на существенных связях. Концептуальная модель определяет структуру моделируемой системы, свойства ее элементов, связи между элементами, существенные для моделируемой системы (с точки зрения цели системы или цели моделирования). Как правило, разработка концептуальной модели производится перед разработкой более детализированной математической модели.

Опираясь на триаду модель - алгоритм - программа, исследователь получает в руки универсальный, гибкий и относительно недорогой инструмент, который вначале отлаживается, тестируется и калибруется на решении содержательного набора пробных задач. После этого проводится широкомасштабное исследование математической модели для получения необходимых качественных и количественных свойств и характеристик исследуемого объекта с помощью вычислительного эксперимента, который по своей природе имеет междисциплинарный характер. В совместных исследованиях участвуют специалисты в прикладной области, прикладной и вычислительной математике, по прикладному и системному программному обеспечению. Вычислительный эксперимент проводится с использованием самых разных методов и подходов.

Следует отметить эволюционный процесс «смены» парадигм моделирования. Первый этап моделирования характеризовался математической записью отдельных моделей реальных объектов. Для них характерна простота описаний, типична линейность уравнений и малая размерность (часто воспроизводится всего одна или две переменных). Методы анализа связаны в основном с получением аналитических решений.
Второй этап характеризуется попытками возможно более полного описания объектов реального мира, причем объект представлен в модели в виде большой или сложной системы, причем учитывается нелинейность зависимостей, возрастает размерность моделей, достигая нескольких десятков. При этом для записи моделей в компьютере и решения соответствующих задач вычислительной математики используются языки программирования. Чрезвычайно важную роль на этом этапе приобретает вычислительный эксперимент. В настоящее время начинается переход к третьему поколению математических моделей, в которых используются методы автоматизации формирования моделей и данных, возникают языки моделирования, существенно отличающиеся своей декларативной природой от обычных языков программирования.

В связи с вышеизложенным чрезвычайно актуальны вопросы автоматизации различных аспектов математического моделирования: выбора и построения моделей, записи моделей в форме, близкой к обычной математической записи, создание библиотек решателей[3].

[3] Решатель - программное обеспечение, предназначенное для решения рассматриваемой математической задачи.

1.2. О методологических вопросах моделирования

В течение последних десятилетий интенсивно развивались различные парадигмы[4] моделирования. При этом каждая парадигма моделирования включает систему накопленных знаний, опыта, методологии и средств моделирования, предназначенных для решения соответствующих этой парадигме задач. Парадигмы моделирования различаются по типу моделей (например, статические, динамические, непрерывные, дискретные, детерминированные, стохастические, нечеткие, модели с «мягкими ограничениями»), по различным методам анализа моделей (например, имитация, оптимизация, многокритериальная оптимизация, нечеткая оптимизация). Кроме этого, в рамках тех или иных парадигм моделирования для решения соответствующих задач разрабатываются специализированные программные продукты - решатели.

Остановимся на методологических вопросах моделирования. В ряде случаев не совсем понятно, о каких именно моделях и задачах идет речь. Так, в русскоязычной литературе часто пишут «В результате формализации данных поставленной задачи получим модель, которая представляет собой задачу линейного программирования. Решим полученную задачу с помощью симплекс-метода». В этом предложении дважды встречалось слово «задача», имеющее совершенно различный смысл. Попробуем описать возможные различия в степени абстракции моделей.

Джеффрион (Geoffrion)[5] выделяет 4 уровня абстракции для моделей. На самом нижнем уровне находится «**конкретная модель»** (model instance), которая является **конкретизацией модели**, получающейся путем присвоения

---

[4] **Парадигма** (от греч. *paradeigma* пример, образец) — общие принципы, концептуальная схема постановки проблем и их решения, методов исследования, общепринятых в научном сообществе. При этом научная парадигма состоит из теорий, законов, правил, моделей, концепций и определений, общепринятых в научном сообществе.

[5] Geoffrion A.M. Integrated modeling system // Comp.Sci.Econ.Manag. – 1989. – V. 2. - P. 3-15.

параметрам модели их числовых значений (или, согласно [1], путем «оснащения» модели данными). Следующий, более высокий уровень абстракции - **класс моделей**, который является множеством подобных моделей с известной математической формулировкой без конкретных числовых данных. Третьим уровнем абстракции является **парадигма моделирования**, которая является множеством классов подобных моделей. И, наконец, на четвертом уровне абстракции находится **направление моделирования** (или **подход моделирования**), которая является группой схожих парадигм моделирования и может быть разделом прикладной математики. Например, исследование операций - направление моделирования, в которое входят такие парадигмы моделирования, как математическое программирование, имитационное моделирование, теория массового обслуживания, теория марковских процессов.

Графовый подход к управлению моделями, например, использует методы теории графов для представления математической модели. Далее используется к**онцептуальная основа (схема) моделирования -** специализация в подходе моделирования. Чисто **концептуальное представление** проблемы строится внутри **концептуальной основы.**

Язык моделирования - формальная компьютерная исполнимая нотация (система обозначений), которая может быть использована для выражения абстрактных понятий концептуальной основы.

Система моделирования - это компьютерная система, которая принимает модель пользователя, транслирует ее в формат, воспринимаемый соответствующим решателем, вызывает, переводит результат работы решателя в форму, понятную пользователю. Решатели обычно не являются частью системы моделирования, система с ними лишь взаимодействует, причем решатели обычно разрабатываются независимо.

Типичный процесс моделирования включает построение модели, выбор метода решения, перевод прикладной модели в модель поиска решения, выбор решателя, перевод модели поиска решения в формат решателя, вызов решателя, интерпретация решения.

Система моделирования взаимодействует с пользователем и решателями. Хорошая система моделирования должна осуществлять это взаимодействие достаточно легко, т.е. интерфейс пользователя должен быть ориентирован на потребности пользователя, а интерфейс решателя должен быть ориентирован на потребности решателей.

1.3. Подходы к представлению и созданию моделей

Подход баз данных. В этом подходе модели представляются с помощью некоторой модели данных, в том числе с использованием сетевой модели данных CODASYL DBTG, модели сущность-объект, реляционной модели.

Графовый подход. Согласно этому подходу, математическая модель представляется с помощью одного или более графов или орграфов. Использование графов при представлении знаний имеет много преимуществ, таких как концептуальная ясность, легкость программирования и простота манипулирования. Граф состоит из вершин и дуг, отражающих семантику модели, причем вершины интерпретируются как объекты. Представление сложных математических отношений посредством графов приводит к более эффективному взаимодействию между специалистами по моделированию и заказчиками.

Современной концептуальной основой моделирования, использующей графы как часть представления моделей, является *структурное*

моделирование[6], предложенное Джеффрионом. Представление проблемы в виде структурной модели является ациклическим графом, описывающим все компоненты проблемы и соотношения между ними.

Подход, основанный на знаниях. Подход использует средства теории искусственного интеллекта для управления моделями. Посредством различных схем знаний, могут быть представлены синтаксическое знание структуры проблемы, семантическое знание о различных компонентах системы, а также (процедурное) знание о том, как манипулировать моделями. При этом в качестве схем представления знаний используются такие концептуальные основы, как

- семантические сети,
- исчисление предикатов первого порядка,
- продукционные правила.

Манипулирование моделями

Манипулирование моделями включает возможности выбора модели, установления последовательности (графика) вызова моделей, интеграции моделей, вызова моделей, генерации компонент новой модели.

- **Выбор модели**. Эта функция пытается найти те модели, которые могут использованы при формировании адекватной модели и затем автоматически выбирает либо позволяет пользователю выбрать подходящую модель. Для пользователя важно то, какой тип модели является подходящим, но не то, как работает процедура поиска решения. Обычно для выбора типа модели необходимы различные виды информации, при этом невозможно использовать всю эту информацию одновременно. Поэтому целесообразно использовать пошаговое

[6] Рыжаков А.Н., Щербина О.А. Современные проблемы математического моделирования в исследовании операций // Динамические системы. - 2006. - Вып. 21. - С. 115-129.

улучшение процедуры выбора модели, осуществляя систематическое применение этой информации.

- **Установление последовательности (графика) вызова моделей.** График моделей является последовательностью моделей, заданной в порядке их выполнения, основанной в основном на связях вход-выход. Таким образом, если выходы модели А являются входами модели В, то модель А выполняется до модели В и результат модели А передается модели В.
- **Интеграция моделей.** Эта функция состоит в идентификации и правильном сочетании моделей и других необходимых для моделирования компонент (таких как файлы данных, процедуры анализа данных), необходимых для подготовки ответа по конкретному запросу. Интеллектуальные системы, разработанные с этой целью, основаны на логическом программировании, графах связей, эвристическом поиске, правилах, семантических сетях, фреймах.
- **Вызов модели.** Эта функция включает активацию выбранных или интегрированных моделей для реализации конкретного процесса моделирования проблемы пользователя.
    - **Генерация новых компонент модели.** Возможна ситуация, что ни одна из существующих моделей не способна решить проблему. Это может возникнуть в двух случаях: 1) Основные компоненты модели уже имеются в базе моделей, но требуемые функции или тесты не построены; 2) нет ни основных компонент модели, ни функций и тестов.
    - **Интерпретация результатов.** Эта фаза состоит в проверке допущений модели, проведении анализа на чувствительность и, если нужно, корректировке модели,. Существующие в настоящее время решатели имеют возможности проведения анализа на

чувствительность в виде, который трудно понять неспециалистам. Дружественный интерфейс должен показывать результаты анализа на чувствительность в виде графов или других простых представлений. Примером реализации системы с помощью пользователям при работе с моделями ЛП является ANALYZE[7]. Эта система не только помогает помощью пользователям проводить анализ на чувствительность, но также и в документировании модели, верификации, отладке и интерпретации результатов.

1.4. Основные черты языков моделирования

Разработка языков моделирования началась в конце 70-х годов, когда в Мировом банке был создан GAMS (General Algebraic Modeling System)[8].

В языке моделирования модель записывается в форме, близкой к математической записи, что является важной чертой алгебраических языков моделирования.

Запись модели стала независимой от формата данных решателя. К языку моделирования могут быть подключены различные решатели, при этом процесс перевода модели и данных в соответствующий входной формат решателя выполняется компьютером автоматически, что позволяет сделать ошибки перевода невозможными (разумеется, после тщательной отладки). В языке моделирования модели и данные хранятся раздельно, т.е. имеется четкое разделение модели и данных для нее. Это позволяет многократно использовать одну и ту же модель для решения различных задач, изменяя только данные.

Многие системы используют интерфейс ODBC (open database connectivity) для автоматического доступа к данным, а также интерфейс к

[7] Greenberg H. Intelligent analysis support for linear programs // Computers and Chemical Engineering. – 1992. - 16(7). - P.659-674.
[8] Brooke A., Kendrick D., Meeraus A. GAMS: A User's Guide. - Redcliff City: The Scietific Press, - 1992.

большинству табличных процессоров (таких, как Excel). Это освобождает пользователя от трудоемкой обязанности поиска соответствующих данных при каждом использовании модели. Вторым преимуществом концепции разделения модели и данных является возможность использования в период разработки модели «игрушечных» моделей с небольшими искусственными массивами данных, а в дальнейшем отлаженная на этих простых примерах модель без каких-либо изменений может быть использована для решения больших производственных задач с реальными объемами информации. Вычисление производных данных может быть автоматизировано с помощью использования автоматического дифференцирования.

После того, как модель запрограммирована, далее используется решатель, способный решать описанные выше задачи оптимизации. *Алгебраические языки моделирования* (АЯМ) берут на себя задачу перевода математической постановки задачи в форму, понятную для соответствующего решателя (т.е. программы, решающей оптимизационную задачу). После решения задачи решателем АЯМ выдают отчет в удобной для пользователя форме.

АЯМ позволяют работать с моделями, заданными во входном файле в виде алгебраических уравнений и неравенств. АЯМ транслирует входной файл в понятный для решателя вид и осуществляет вызов этого решателя. В данном случае решатель рассматривается как «черный ящик» (известно, какие исходные данные требуются для него и что будет получено в результате его работы). В ряде случаев решатель может запросить дополнительную информацию, необходимую для его работы. Как только решатель решит задачу, решение будет возвращено АЯМ и результаты будут сообщены пользователю.

АЯМ позволяют специалисту в области математического моделирования записать модель в математической форме, использующей множества индексов, параметры, переменные и константы. Очень важна возможность АЯМ записывать подобные ограничения в виде множеств, к которым затем можно обращаться, используя соответствующие индексы. Это позволяет компактно записывать модели в виде, близком к естественной математической записи.

2. Алгебраический язык моделирования AMPL.

2.1. Основные особенности программирования на AMPL

AMPL (**A** **M**athematical **P**rogramming **L**anguage) [2] – это язык высокого уровня для описания задач математического программирования, использующий декларативно-алгебраический стиль представления моделей математического программирования, близкий к традиционной математической нотации. Вместе с тем AMPL дает возможность описать и сложные модели оптимизации с различными логическими условиями, с использованием сложных систем индексации переменных и ограничений. AMPL позволяет задать модель математического программирования независимо от данных, используемых для конкретного примера модели. В данном разделе приведены лишь базовые сведения об AMPL, которые могут понадобиться для описания моделей и выполнения лабораторных работ этого курса.

AMPL требует также задания исходных данных (оснащения модели). Модель и один (или более) файлов данных направляются в систему AMPL. AMPL работает подобно компилятору: модель и данные соединяются в промежуточный файл, который передается решателю. Решатель фактически находит оптимальное решение задачи, используя промежуточный файл, построенный AMPL, и применяя соответствующий алгоритм. Решатель выдает решение в виде текстового файла.

Работа с AMPL из командной строки

Модели записываются в виде текстового файла <*имя файла*>.mod. При написании модели на языке AMPL можно использовать любой текстовый редактор.

При написании моделей используются основные команды AMPL. AMPL-модель содержит описания объектов модели, т.е. множеств, переменных, параметров, целевой функции и ограничений. Для описания объектов используются служебные слова **set, var, param, minimize/maximize, subject to** (или кратко **s.t.**). При этом AMPL-модель содержит несколько типов элементов, подробнее описываемых ниже: декларации с ключевыми словами: **set** (множество индексов), **param** (параметр), **var** (переменная), **arc** (дуга – для описания сетевых моделей); из целевых функций вида **maximize/minimize,** ограничений, **subject to** (при ограничениях), **node** (вершина – для описания сетевых моделей).

Во многом синтаксис команд AMPL очень подобен C. AMPL поддерживает такие функции, как abs(), cos(), sin(), log(), sqrt(), exp() с использованием основных операций +, -, *, /, ^ или **. Все команды оканчиваются точкой с запятой «;». К командам вывода относятся **display**, а также команды **write** и **print**. К другим полезным элементам относятся **option** – для изменения опций AMPL или решателя, **include –** чтение из отдельного файла, а также **quit** – для выхода из AMPL.

Имена (идентификаторы) состоят из латинских букв (прописных и строчных), цифр и знаков подчеркивания. Символ # означает начало комментария. Все, что находится за этим символом, игнорируется AMPL. Комментарии могут быть также ограничены символами /* и */, причем они могут отделены друг от друга несколькими строками.

Команды AMPL используют простой синтаксис:

Переменные описываются с использованием служебного слова **var**.

Параметры описываются с использованием служебного слова **param**.

Суммирование $\sum_{i=1}^{n}$ записывается так: `sum{i in 1..n}`.

Служебные слова AMPL (такие как **var**, **param**, **solve**, **maximize** и др.), а также имена функций (например, **sum**, **log**, **sin**) зарезервированы и не могут использоваться для имен объектов. К служебным зарезервированным словам относятся также **for**, **if**, **elseif**, **else**, **while**, **file**, **system**.

Индексы переменных и ограничений заключаются в квадратные скобки (например, a[i]).

Числа могут записываться в разных форматах. Так, 0.0123, 1.23D-2, 1.23e-2, 1.23E-2 - это эквивалентные записи одного и того же числа 0,0123.

Литералы – это строки, заключенные в кавычки (одинарные или двойные). Например, ‘abc’, ‘x’, ‘y’, “ABC”.

Команда вида

include <имя файла>

вставляет указанный файл.

Команды

model; include < имя файла >

data; include < имя файла >

могут быть сокращены до следующих:

model < имя файла >;

data < имя файла >;

Команда **commands** аналогична команде **include,** но это оператор и должен оканчиваться точкой с запятой.

В AMPL модель и данные разделены. Множества, описанные в модели, не имеют размеров или заданных элементов. Использование в модели

множества индексов означает лишь то, что используются элементы этого множества, причем неважно, сколько их имеется. Задание величин делается только в файле данных, который имеет вид <*имя файла*>.dat. Здесь элементы множеств и параметры определяются явно, повторно записывая служебное слово, имя объекта и перечисляя значения после ":=".

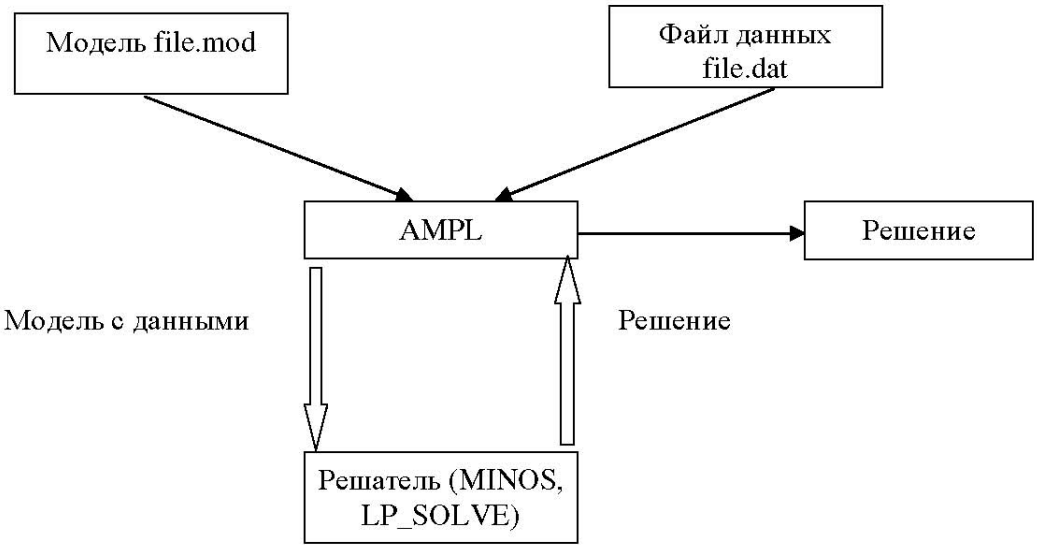

Рис.1. Схема работы системы моделирования AMPL.

Пример создания модели AMPL

Решить следующую задачу, используя решатель MINOS[9]

$z = x_1 + x_2 \to \min$

при ограничениях

$x_1 \cdot x_2 \geq 4$,

$0 \leq x_1, x_2 \leq 3$.

Создадим следующий текстовый файл с именем ex1.mod

```
# ex1.mod - Model
```

[9] MINOS – решатель для задач нелинейного программирования.

```
      var x1 >= 0, <= 3;   # Задание нижней и верхней границ
переменной x1
      var x2 >= 0, <= 3;   # Задание нижней и верхней границ
переменной x2
     minimize z:  x1 + x2;           # Целевая функция z
     s.t. con1:  x1 * x2 >= 4; # Ограничение 1 – ему присвоено
имя con1
     data;          # Задание начальной точки для решателя
     var x1    := 1;
     var x2    := 1;
```

Далее наберите >ampl (находясь в папке, где есть исполнимый файл ampl). Появится командное окно AMPL с подсказкой **ampl:**

В окне AMPL наберите следующие команды для решения этой задачи (не забудьте о точке с запятой ";" в конце каждой команды) для решения этой задачи с помощью решателя MINOS (по умолчанию).

```
ampl: include ex1.mod;
ampl: solve;
ampl: display x1, x2, z;
```

AMPL выдает следующую информацию о решении x1 = 2, x2 = 2 с оптимальным значением целевой функции z = 4.

```
MINOS 5.5: optimal solution found.
15 iterations, objective 4
Nonlin evals: constrs = 38, Jac = 37.
x1 = 2
x2 = 2
z = 4
```

Для редактирования текстового файла (в связи с обнаруженными ошибками или необходимостью изменения модели), внесите необходимые изменения в текстовом файле модели и сохраните его. Затем в окне AMPL наберите следующие команды для ввода измененной модели:

```
ampl:  reset;
ampl:  model ex1.mod;
```

Запишем на AMPL модель целочисленного линейного программирования (ЦЛП) вида:

$$\sum_{j=1}^{n} c_j x_j \to \max$$

при ограничениях

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad i = 1,\ldots, m,$$

$$x_j = 0, 1, \quad j = 1,\ldots, n.$$

Описание параметра m в AMPL выглядит так:

```
param m;
```

Аналогично опишем параметр n:

```
param n;
```

Опишем множества индексов $i = 1,\ldots, m$ и $j = 1,\ldots, n$:

```
set I=1..m;
set J=1..n;
```

Для описания параметров c_j, b_i, a_{ij} используем запись:

```
param c{J};
param b{I};
param a{I, J};
```

Для описания переменных используем запись:

```
var x {J} binary;
```

которая означает, что x_j - бинарная переменная, принимающая только значения 0 или 1.

Описание целевой функции $\sum_{j=1}^{n} c_j x_j \to \max$ выглядит следующим образом:

```
maximize z: sum {j in 1..n} c[j] * x[j];
```

Ограничения $\sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad i = 1,\ldots, m$ запишутся так:

```
con{i in 1..m}: sum {j in 1..n} a[i,j] * x[j] <= b[i];
```

Таким образом, модель ЦЛП записывается на AMPL в виде ip.mod:

```
param m;
param n;
set I:=1..m;
set J:=1..n;
param c {J};
param b {I};
param a {I, J};
var x {J} binary;
maximize z:  sum{j in J}c[j]*x[j];
subject to res {i in I}:
sum{j in J}a[i,j]*x[j]<= b[i];
```

Для решения конкретной задачи ЦЛП вида

$z = 2x_1 + 3x_2 + x_3 \to \max$

при ограничениях

$x_1 + 2x_2 + x_3 \le 2,$

$2x_1 + 3x_2 + 2x_3 \le 4,$

$x_1, x_2, x_3 = 0, 1$

можно использовать описанный выше файл модели ip.mod вместе с файлом данных ip.dat следующего вида

```
 param m:=2;
 param n:=3;
 param c [1] 2 [2] 3 [3] 1;
 param b [1] 2 [2] 4;
 param a: 1   2   3:=
  1       1   2   1
  2       2   3   2;
```

В окне AMPL набираем операторы:

```
model ip.mod;

data ip.mod;

option solver lpsolve;

solve;

display x, obj;
```

В результате получим:

```
LP_SOLVE 4.0.1.0: optimal, objective 3

5 simplex iterations

3 branch & bound nodes: depth 2

x [*] :=

1  1

2  0

3  1

;



obj = 3
```

Замечание. Запись `option solver lpsolve;` означает явное задание решателя задач смешанного целочисленного линейного программирования ***lp_solve (****http:// lpsolve.sourceforge.net/).*

Общие синонимы для переменных, ограничений и целевых функций

В ряде случаев бывает нужно распечатать либо проверить все имеющиеся в модели переменные, ограничения, целевые функции. С этой целью AMPL использует общие синонимы всех компонентов:

_nvars – число переменных в рассматриваемой модели;

_ncons – число ограничений в рассматриваемой модели;

_nobjs – число целевых функций в рассматриваемой модели;

Соответственно параметры с индексами содержат имена всех компонент модели AMPL:

_varname{1.._nvars} – имена переменных в рассматриваемой модели;

_conname{1.._ncons} – имена ограничений в рассматриваемой модели;

_objname{1.._nobjs} – имена целевых функций в рассматриваемой модели;

И, наконец, имеются следующие синонимы для компонент:

_var{1.._nvars} – синонимы переменных в рассматриваемой модели;

_con{1.._ncons} – синонимы ограничений в рассматриваемой модели;

_obj{1.._nobjs} – синонимы целевых функций в рассматриваемой модели.

Графический интерфейс AMPL

Имеется графический интерфейс AMPL, называемый AMPL Studio, который доступен на Web-странице ***http://www.optirisk-systems.com***.

2.2. Декларации компонент модели

В дальнейшем для формального описания операторов используем **форму Бэкуса—Наура** (сокр. БНФ, Бэкуса—Наура форма) — формальную систему описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие категории. БНФ-конструкция состоит из предложений вида

<определяемый символ> ::= <посл.1> | <посл.2> | . . . | <посл.n>,

описывающих правила. Такое правило означает, что символ <определяемый символ> может заменяться на одну из последовательностей посл.1, посл.2, ..., посл.n. Знак определения обычно выглядит как ::=. Кроме того, используются квадратные скобки: [A] — элемент A входит или не входит.

Декларации объектов модели AMPL имеют следующий общий вид

<объект> <имя объекта> [<алиас>] [<индексное выражение>] [<тело>]

<объект>::= set | param | var | arc | minimize | maximize | subject to | node

Декларации параметров

Параметрами могут быть скалярные величины, векторы, матрицы и массивы известных данных. Декларация параметра имеет вид:

param <*имя_переменной*> <*алиас*> <*индексное_выражение*> <*атрибуты*>;

<*алиас*> *::=<литерал>*

<*индексное выражение*>*::=<set_expression_list*

<*арифметический_оператор*>*::=*+ | **-** | *less* | ***** | */* | *mod* | *div* | ^| ****** |

<*унарный оператор*>*::=* + | **-**

<*оператор редукции*>*::= sum* | *prod* | *max* | *min*

Необязательные атрибуты деклараций параметров могут разделяться запятыми, ниже приведены указанные атрибуты:

Атрибут:	Примечание
binary	Ограничивает возможные значения переменной 0 или 1.
integer	Ограничивает возможные значения переменной целочисленными значениями
symbolic	Если задано **symbolic**, то должно также быть заданы **in sexpr,** а также исключаются атрибуты, требующие задания числовых значений, такие как **>=**<выражение>
>= <выражение>	>= задает нижнюю границу
<= < выражение >	<= задает верхнюю границу
:= < выражение >	:= задает начальное значение
default < выражение >	Задает значение по умолчанию для начальных значений, которые могут быть заданы в разделе данных **data**
= < выражение >	Задает определенную переменную

AMPL позволяет определять параметры через другие, ранее определенные, параметры. Параметры могут также вычисляться и определяться рекурсивно. Например, для числа сочетаний C_n^m по ***m*** из ***n*** предметов может быть определено, используя следующую декларацию

```
param comb {i in 0..n, k in 0..m} :=

if k = 0 or k = i then 1 else comb[i-1,k-1] + comb[i-1,k];
```

Декларации переменных

Декларации переменных начинаются со служебного слова ***var:***

var <*имя_переменной*> <*алиас*> <*индексное_выражение*> <*атрибуты*>

где все элементы декларации такие же, как в декларации параметра (см. выше).

Декларации ограничений

Ограничения записываются в следующей общей форме[10]:

<декларация ограничения> :: =
[**subject to**] <*имя ограничения*> [<*алиас*>] [<*индексное выражение*>] <*атрибуты*>
[: <выражение ограничения>];

< выражение ограничения >*::=* <*выражение*> <= <*выражение*> |

< *выражение* > = <*выражение* > | < *выражение* > >= < *выражение* > |

Декларации целевой функции

Декларация (описание) целевой функции имеет вид:

<декларация целевой функции> :: = **maximize** <имя> [<алиас>] [<*индексное выражение*>] [: выражение]; |
minimize < имя > [< алиас >] [<*индексное выражение* >] [: выражение];

Обзор команд AMPL

Команда	Комментарий
call	Вызов импортированной функции
cd	Переход в другой каталог
check	Выполняет все команды check
close	Закрывает файл
commands	Чтение и интерпретация команд из файла
data	Переход к данным
delete	Удаление компонент модели
display	Печать компонент модели и выражений
drop	Исключение ограничения или целевой функции
end	Окончание ввода из текущего файла ввода
environ	Установить среду для модели
exit	Выйти из AMPL со значением статуса
expand	Показать детально компоненты модели
fix	Зафиксировать переменную на ее текущем значении
include	Включить содержимое файла
let	Изменить значения данных
load	Загрузить динамическую библиотеку
model	Загружает модель

[10] Приводится лишь простейшая форма ограничения.

objective	Выбрать целевую функцию для оптимизации
option	Установить или выдать значения опций
print	Неформатированная печать компонент модели и выражений
printf	Форматированная печать компонент модели и выражений
problem	Определить задачу или перейти к задаче
purge	Удаление компонент модели
quit	Окончить работу AMPL
read	Чтение из файла
read table	Чтение из таблицы данных
redeclare	Изменение декларации объекта
reload	Перезагрузка динамической библиотеки функций
remove	Удаление файла
reset	Сброс объектов, восстановление их исходного состояния
restore	Отменить действие команды **drop**
shell	Временный выход в операционную систему
show	Показать имена компонент модели
solexpand	Показать детальное расширение, видимое решателем
solution	Импортирует значения переменной из решателя
solve	Конкретный пример модели посылается решателю и возвращается найденное решение
update	Разрешить изменение данных
unfix	Отменить действие команды **fix**
unload	Выгрузить динамическую библиотеку
write	Выдача конкретного примера задачи
write table	Записать таблицу в таблицу данных
xref	Показать зависимости между компонентами модели

Файлы, упоминаемые в командах **include, model, data**, **commands**, имеющие простые имена (например, не содержащие слэш /), ищутся в папках, задаваемых опцией **ampl_include**: каждая непустая строка $**ampl_include** задает такую папку; если же $**ampl_include** пусто или является чистым**,** файлы ищутся в текущей папке.

Перенаправление вывода

Для перенаправления вывода в файл, достаточно добавить > и имя файла. Для открытия уже существующего файла и добавления к нему вывода нужно использовать >> вместо >.

Например,

ampl: model ip.mod;

ampl: data ip.dat;

ampl: solve;

```
LP_SOLVE 4.0.1.0: optimal, objective 3

5 simplex iterations

3 branch & bound nodes: depth 2
```

ampl: display x, obj > ip.out;

Скрипты и операторы потока управления

В AMPL предусмотрены операторы, подобные операторам потока управления в обычных языках программирования, которые дают возможность писать программу с помощью операторов, которая будет выполняться автоматически.

При работе с моделью пользователь часто применяет последовательности одних и тех же команд, которые нужно набирать на экране, Для ускорения процесса, возможно записать эти команды в специальный скриптовый файл <file>.run, вызывая его затем с помощью

```
include <file>.run
```

Так, для вызова файлов из примера , можно записать в файле ip.run

```
model ip.mod;

data ip.dat;

option solver lpsolve;

solve;

display x, z;
```

Для запуска скрипта на экране AMPL набирается `include ip.run`;

Стандартный формат данных

AMPL поддерживает стандартный формат, чтобы описать значения множеств и параметров, которые объединяются с моделью, чтобы получить конкретную задачу оптимизации.

Раздел данных состоит из последовательности токенов (лексем) (литералов и строк печатных символов), отделенный пробелами, символами табуляции и перевода строки. Лексемы включают ключевые слова, литералы, числа, и разделители () []:;: = *. Утверждение (оператор) - последовательность лексем, заканчивающаяся точкой с запятой. Комментарии могут использоваться так же, как в декларациях. Во всех случаях, расположение данных в четких строках и столбцах - для удобства чтения человеком; AMPL же игнорирует форматирование.

Раздел данных начинается с команды `data` и заканчивается концом-ввода или командой, которая возвращает в режим моделирования.

В разделе данных, объектам модели могут быть назначены значения в любом удобном порядке, независимом от порядка их декларирования.

Данные для множеств

Операторы, определяющие множество, состоят из ключевого слова **set** (множество), имени множества, опциональных: =, и членов. Одномерное

множество наиболее просто определить, задавая список его членов, опционально отделенных запятыми. Единичные или двойные кавычки литеральной строки могут быть опущены, если строка алфавитно-цифровая, но не определяет число.

Примеры задания данных для параметров приведены выше, в примере для модели ЦЛП.

Более подробные сведения о языке алгебраического моделирования AMPL можно найти в книге [2] и на сайте http://www.ampl.com.

Литература

1. Самарский А.А. Математическое моделирование. Идеи. Методы. Примеры / А.А. Самарский, А.П. Михайлов. - М.:ФИЗМАТЛИТ, 2005. – 320 с.

2. Fourer R. AMPL, A Modeling Language for Mathematical Programming, Second Edition / R. Fourer, D. Gay, B. Kernighan. - Belmont: Duxburry Press, 2003. – 517 p.