Introduction

As our title suggests, there are two aspects to the subject of this book. The first is mathematical programming, the optimization of a function of many variables subject to constraints. The second is the AMPL modeling language, which we designed and implemented to help people use computers to develop and apply mathematical programming models.

We intend this book as an introduction both to mathematical programming and to AMPL. For readers already familiar with mathematical programming, it can serve as a user's guide and reference manual for the AMPL software. We assume no previous knowledge of the subject, however, and hope that this book will also encourage the use of mathematical programming models by those who are new to the field.

Mathematical programming

The term ''programming'' was in use by 1940 to describe the planning or scheduling of activities within a large organization. ''Programmers'' found that they could represent the amount or level of each activity as a *variable* whose value was to be determined. Then they could mathematically describe the restrictions inherent in the planning or scheduling problem as a set of equations or inequalities involving the variables. A solution to all of these *constraints* would be considered an acceptable plan or schedule.

Experience soon showed that it was hard to model a complex operation simply by specifying constraints. If there were too few constraints, many inferior solutions could satisfy them; if there were too many constraints, desirable solutions were ruled out, or in the worst case no solutions were possible. The success of programming ultimately depended on a key insight that provided a way around this difficulty. One could specify, in addition to the constraints, an *objective*: a function of the variables, such as cost or profit, that could be used to decide whether one solution was better than another. Then it didn't matter that many different solutions satisfied the constraints — it was sufficient to find one such solution that minimized or maximized the objective. The term *mathematical programming* came to be used to describe the minimization or maximization of an objective function of many variables, subject to constraints on the variables.

In the development and application of mathematical programming, one special case stands out: that in which all the costs, requirements and other quantities of interest are terms strictly proportional to the levels of the activities, or sums of such terms. In mathematical terminology, the objective is a linear function, and the constraints are linear equations and inequalities. Such a problem is called a *linear program*, and the process of setting up such a problem and solving it is called *linear programming.* Linear programming is particularly important because a wide variety of problems can be modeled as linear programs, and because there are fast and reliable methods for solving linear programs even with thousands of variables and constraints. The ideas of linear programming are also important for analyzing and solving mathematical programming problems that are not linear.

All useful methods for solving linear programs require a computer. Thus most of the study of linear programming has taken place since the late 1940's, when it became clear that computers would be available for scientific computing. The first successful computational method for linear programming, the simplex algorithm, was proposed at this time, and was the subject of increasingly effective implementations over the next decade. Coincidentally, the development of computers gave rise to a now much more familiar meaning for the term ''programming.''

In spite of the broad applicability of linear programming, the linearity assumption is sometimes too unrealistic. If instead some smooth nonlinear functions of the variables are used in the objective or constraints, the problem is called a *nonlinear program.* Solving such a problem is harder, though in practice not impossibly so. Although the optimal values of nonlinear functions have been a subject of study for over two centuries, computational methods for solving nonlinear programs in many variables were developed only in recent decades, after the success of methods for linear programming. The field of mathematical programming is thus also known as *large scale optimization,* to distinguish it from the classical topics of optimization in mathematical analysis.

The assumptions of linear programming also break down if some variables must take on whole number, or integral, values. Then the problem is called *integer programming,* and in general becomes much harder. Nevertheless, a combination of faster computers and more sophisticated methods have made large integer programs increasingly tractable in recent years.

The AMPL modeling language

Practical mathematical programming is seldom as simple as running some algorithmic method on a computer and printing the optimal solution. The full sequence of events is more like this:

- Formulate a model, the abstract system of variables, objectives, and constraints that represent the general form of the problem to be solved.
- Collect data that define a specific problem instance.
- Generate a specific objective function and constraint equations from the model and data.

- Solve the problem instance by running a program, or *solver*, to apply an algorithm that finds optimal values of the variables.
- Analyze the results.
- Refine the model and data as necessary, and repeat.

If people could deal with mathematical programs in the same way that solvers do, the formulation and generation phases of modeling might be relatively straightforward. In reality, however, there are many differences between the form in which human modelers understand a problem and the form in which solver algorithms work with it. Conversion from the ''modeler's form'' to the ''algorithm's form'' is consequently a time-consuming, costly, and often error-prone procedure.

In the special case of linear programming, the largest part of the algorithm's form is the constraint coefficient matrix, which is the table of numbers that multiply all the variables in all the constraints. Typically this is a very sparse (mostly zero) matrix with anywhere from hundreds to hundreds of thousands of rows and columns, whose nonzero elements appear in intricate patterns. A computer program that produces a compact representation of the coefficients is called a matrix generator. Several programming languages have been designed specifically for writing matrix generators, and standard computer programming languages are also often used.

Although matrix generators can successfully automate some of the work of translation from modeler's form to algorithm's form, they remain difficult to debug and maintain. One way around much of this difficulty lies in the use of a *modeling language* for mathematical programming. A modeling language is designed to express the modeler's form in a way that can serve as direct input to a computer system. Then the translation to the algorithm's form can be performed entirely by computer, without the intermediate stage of computer programming. Modeling languages can help to make mathematical programming more economical and reliable; they are particularly advantageous for development of new models and for documentation of models that are subject to change.

Since there is more than one form that modelers use to express mathematical programs, there is more than one kind of modeling language. An *algebraic* modeling language is a popular variety based on the use of traditional mathematical notation to describe objective and constraint functions. An algebraic language provides computer-readable equivalents of notations such as $x_j + y_j$, $\Sigma_{j=1}^{n} a_{ij} x_j$, $x_j \geq 0$, and $j \in S$ that would be familiar to anyone who has studied algebra or calculus. Familiarity is one of the major advantages of algebraic modeling languages; another is their applicability to a particularly wide variety of linear, nonlinear and integer programming models.

While successful algorithms for mathematical programming first came into use in the 1950's, the development and distribution of algebraic modeling languages only began in the 1970's. Since then, advances in computing and computer science have enabled such languages to become steadily more efficient and general.

This book describes AMPL, an algebraic modeling language for mathematical programming; it was designed and implemented by the authors around 1985, and has been evolving ever since. AMPL is notable for the similarity of its arithmetic expressions to customary algebraic notation, and for the generality and power of its set and subscripting

expressions. AMPL also extends algebraic notation to express common mathematical programming structures such as network flow constraints and piecewise linearities.

AMPL offers an interactive command environment for setting up and solving mathematical programming problems. A flexible interface enables several solvers to be available at once so a user can switch among solvers and select options that may improve solver performance. Once optimal solutions have been found, they are automatically translated back to the modeler's form so that people can view and analyze them. All of the general set and arithmetic expressions of the AMPL modeling language can also be used for displaying data and results; a variety of options are available to format data for browsing, printing reports, or preparing input to other programs.

Through its emphasis on AMPL, this book differs considerably from the presentation of modeling in standard mathematical programming texts. The approach taken by a typical textbook is still strongly influenced by the circumstances of 30 years ago, when a student might be lucky to have the opportunity to solve a few small linear programs on any actual computer. As encountered in such textbooks, mathematical programming often appears to require only the conversion of a ''word problem'' into a small system of inequalities and an objective function, which are then presented to a simple optimization package that prints a short listing of answers. While this can be a good approach for introductory purposes, it is not workable for dealing with the hundreds or thousands of variables and constraints that are found in most real-world mathematical programs.

The availability of an algebraic modeling language makes it possible to emphasize the kinds of general models that can be used to describe large-scale optimization problems. Each AMPL model in this book describes a whole class of mathematical programming problems, whose members correspond to different choices of indexing sets and numerical data. Even though we use relatively small data sets for illustration, the resulting problems tend to be larger than those of the typical textbook. More important, the same approach, using still larger data sets, works just as well for mathematical programs of realistic size and practical value.

We have not attempted to cover the optimization theory and algorithmic details that comprise the greatest part of most mathematical programming texts. Thus, for readers who want to study the whole field in some depth, this book is a complement to existing textbooks, not a replacement. On the other hand, for those whose immediate concern is to apply mathematical programming to a particular problem, the book can provide a useful introduction on its own.

In addition, AMPL software is readily available for experiment: the AMPL web site, `www.ampl.com`, provides free downloadable ''student'' versions of AMPL and representative solvers that run on Windows, Unix/Linux, and Mac OS X. These can easily handle problems of a few hundred variables and constraints, including all of the examples in the book. Versions that support much larger problems and additional solvers are also available from a variety of vendors; again, details may be found on the web site.

Outline of the book

The second edition, like the first, is organized conceptually into four parts. Chapters 1 through 4 are a tutorial introduction to models for linear programming:

    1. Production Models: Maximizing Profits
    2. Diet and Other Input Models: Minimizing Costs
    3. Transportation and Assignment Models
    4. Building Larger Models

These chapters are intended to get you started using AMPL as quickly as possible. They include a brief review of linear programming and a discussion of a handful of simple modeling ideas that underlie most large-scale optimization problems. They also illustrate how to provide the data that convert a model into a specific problem instance, how to solve a problem, and how to display the answers.

The next four chapters describe the fundamental components of an AMPL linear programming model in detail, using more complex examples to examine major aspects of the language systematically:

    5. Simple Sets and Indexing
    6. Compound Sets and Indexing
    7. Parameters and Expressions
    8. Linear Programs: Variables, Objectives and Constraints

We have tried to cover the most important features, so that these chapters can serve as a general user's guide. Each feature is introduced by one or more examples, building on previous examples wherever possible.

The following six chapters describe how to use AMPL in more sophisticated ways:

    9. Specifying Data
   10. Database Access
   11. Modeling Commands
   12. Display Commands
   13. Command Scripts
   14. Interactions with Solvers

The first two of these chapters explain how to provide the data values that define a specific instance of a model; Chapter 9 describes AMPL's text file data format, while Chapter 10 presents features for access to information in relational database systems. Chapter 11 explains the commands that read models and data, and invoke solvers; Chapter 12 shows how to display and save results. AMPL provides facilities for creating scripts of commands, and for writing loops and conditional statements; these are covered in Chapter 13. Chapter 14 goes into more detail on how to interact with solvers so as to make the best use of their capabilities and the information they provide.

Finally, we turn to the rich variety of problems and applications beyond purely linear models. The remaining chapters deal with six important special cases and generalizations:

15. Network Linear Programs
16. Columnwise Formulations
17. Piecewise-Linear Programs
18. Nonlinear Programs
19. Complementarity Problems
20. Integer Linear Programs

Chapters 15 and 16 describe additional language features that help AMPL represent particular kinds of linear programs more naturally, and that may help to speed translation and solution. The last four chapters cover generalizations that can help models to be more realistic than linear programs, although they can also make the resulting optimization problems harder to solve.

Appendix A is the AMPL reference manual; it describes all language features, including some not mentioned elsewhere in the text. Bibliography and exercises may be found in most of the chapters.

About the second edition

AMPL has evolved a lot in ten years, but its core remains essentially unchanged, and almost all of the models from the first edition work with the current program. Although we have made substantial revisions throughout the text, much of the brand new material is concentrated in the third part, where the original single chapter on the command environment has been expanded into five chapters. In particular, database access, scripts and programming constructs represent completely new material, and many additional AMPL commands for examining models and accessing solver information have been added.

The first edition was written in 1992, just before the explosion in Internet and web use, and while personal computers were still rather limited in their capabilities; the first student versions of AMPL ran on DOS on tiny, slow machines, and were distributed on floppy disks.

Today, the web site at `www.ampl.com` is the central source for all AMPL information and software. Pages at this site cover all that you need to learn about and experiment with optimization and the use of AMPL:

- Free versions of AMPL for a variety of operating systems.
- Free versions of several solvers for a variety of problem types.
- All of the model and data files used as examples in this book.

The free software is fully functional, save that it can only handle problems of a few hundred variables and constraints. Unrestricted commercial versions of AMPL and solvers are available as well; see the web site for a list of vendors.

You can also try AMPL without downloading any software, through browser interfaces at `www.ampl.com/TRYAMPL` and the NEOS Server (`neos.mcs.anl.gov`). The AMPL web site also provides information on graphical user interfaces and new AMPL language features, which are under continuing development.

Acknowledgements to the first edition

We are deeply grateful to Jon Bentley and Margaret Wright, who made extensive comments on several drafts of the manuscript. We also received many helpful suggestions on AMPL and the book from Collette Coullard, Gary Cramer, Arne Drud, Grace Emlin, Gus Gassmann, Eric Grosse, Paul Kelly, Mark Kernighan, Todd Lowe, Bob Richton, Michael Saunders, Robert Seip, Lakshman Sinha, Chris Van Wyk, Juliana Vignali, Thong Vukhac, and students in the mathematical programming classes at Northwestern University. Lorinda Cherry helped with indexing, and Jerome Shepheard with typesetting. Our sincere thanks to all of them.

Bibliography

E. M. L. Beale, ''Matrix Generators and Output Analyzers.'' In Harold W. Kuhn (ed.), *Proceedings of the Princeton Symposium on Mathematical Programming*, Princeton University Press (Princeton, NJ, 1970) pp. 25–36. A history and explanation of matrix generator software for linear programming.

Johannes Bisschop and Alexander Meeraus, ''On the Development of a General Algebraic Modeling System in a Strategic Planning Environment.'' Mathematical Programming Study **20** (1982) pp. 1–29. An introduction to GAMS, one of the first and most widely used algebraic modeling languages.

Robert E. Bixby, ''Solving Real-World Linear Programs: A Decade and More of Progress.'' Operations Reearch **50** (2002) pp. 3)–15. A history of recent advances in solvers for linear programming. Also in this issue are accounts of the early days of mathematical programming by pioneers of the field.

George B. Dantzig, ''Linear Programming: The Story About How It Began.'' In Jan Karel Lenstra, Alexander H. G. Rinnooy Kan and Alexander Schrijver, eds., *History of Mathematical Programming: A Collection of Personal Reminiscences.* North-Holland (Amsterdam, 1991) pp. 19–31. A source for our brief account of the history of linear programming. Dantzig was a pioneer of such key ideas as objective functions and the simplex algorithm.

Robert Fourer, ''Modeling Languages versus Matrix Generators for Linear Programming.'' ACM Transactions on Mathematical Software **9** (1983) pp. 143–183. The case for modeling languages.

C. A. C. Kuip, ''Algebraic Languages for Mathematical Programming.'' European Journal of Operational Research **67** (1993) 25–51. A survey.