
LINDO

API 6.0

User Manual

LINDO Systems, Inc.



1415 North Dayton Street, Chicago, Illinois 60642

Phone: (312)988-7422 Fax: (312)988-9065

E-mail: info@lindo.com

COPYRIGHT

LINDO API and its related documentation are copyrighted. You may not copy the LINDO API software or related documentation except in the manner authorized in the related documentation or with the written permission of LINDO Systems, Inc.

TRADEMARKS

LINDO is a registered trademark of LINDO Systems, Inc. Other product and company names mentioned herein are the property of their respective owners.

DISCLAIMER

LINDO Systems, Inc. warrants that on the date of receipt of your payment, the disk enclosed in the disk envelope contains an accurate reproduction of LINDO API and that the copy of the related documentation is accurately reproduced. Due to the inherent complexity of computer programs and computer models, the LINDO API software may not be completely free of errors. You are advised to verify your answers before basing decisions on them. NEITHER LINDO SYSTEMS INC. NOR ANYONE ELSE ASSOCIATED IN THE CREATION, PRODUCTION, OR DISTRIBUTION OF THE LINDO SOFTWARE MAKES ANY OTHER EXPRESSED WARRANTIES REGARDING THE DISKS OR DOCUMENTATION AND MAKES NO WARRANTIES AT ALL, EITHER EXPRESSED OR IMPLIED, REGARDING THE LINDO API SOFTWARE, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR OTHERWISE. Further, LINDO Systems, Inc. reserves the right to revise this software and related documentation and make changes to the content hereof without obligation to notify any person of such revisions or changes.

Copyright ©2009 by LINDO Systems, Inc. All rights reserved.

Published by



LINDO SYSTEMS INC.

1415 North Dayton Street
Chicago, Illinois 60642
Technical Support: (312) 988-9421
E-mail: tech@lindo.com
<http://www.lindo.com>

TABLE OF CONTENTS

TABLE OF CONTENTS	iii
Preface	vii
Chapter 1:.....	1
Introduction.....	1
What Is LINDO API?	1
Linear Solvers.....	2
Mixed-Integer Solver.....	2
Nonlinear Solver	3
Global Solver	3
Stochastic Solver.....	3
Installation	3
Windows Platforms	4
Unix-Like Platforms.....	4
Updating License Keys.....	6
Solving Models from a File using Runlindo	7
Sample Applications.....	9
Array Representation of Models.....	9
Sparse Matrix Representation	10
Simple Programming Example	13
Chapter 2:.....	17
Function Definitions.....	17
Common Parameter Macro Definitions	18
Structure Creation and Deletion Routines.....	21
License and Version Information Routines	23
Input-Output Routines	25
Parameter Setting and Retrieving Routines.....	42
Available Parameters.....	53
Available Information	99
Model Loading Routines.....	114
Solver Initialization Routines	135
Optimization Routines	139
Solution Query Routines	144
Model Query Routines.....	160
Model Modification Routines	191
Model and Solution Analysis Routines.....	210
Error Handling Routines	219
Advanced Routines	221
Callback Management Routines	227
Memory Management Routines	238
Random Number Generation Routines.....	241
Sampling Routines	245
Chapter 3:.....	259
Solving Linear Programs.....	259
A Programming Example in C.....	259
A Programming Example in Visual Basic.....	269

VB and Delphi Specific Issues:	277
Chapter 4: Solving.....	279
Mixed-Integer Programs.....	279
Staffing Example Using Visual C++	280
Staffing Example Using Visual Basic	287
Chapter 5: Solving Quadratic Programs	295
Setting up Quadratic Programs	296
Loading Quadratic Data via Extended MPS Format Files.....	296
Loading Quadratic Data via API Functions	297
Sample Portfolio Selection Problems.....	300
Example 1. The Markowitz Model:	300
Example 2. Portfolio Selection with Restrictions on the Number of Assets Invested:..	304
Chapter 6: Solving Second-Order Cone Programs.....	311
Setting up Second-Order Cone Programs	314
Loading Cones via Extended MPS Format Files.....	314
Loading Cones via API Functions	316
Example 3: Minimization of Norms:	316
Converting Models to SOCP Form.....	321
Example 4: Ratios as SOCP Constraints:	322
Quadratic Programs as SOCP	326
Chapter 7: Solving Nonlinear Programs.....	329
Black-Box Style Interface	330
Loading Model Data.....	331
Evaluating Nonlinear Terms via Callback Functions	333
Instruction-List Style Interface	337
Postfix Notation in Representing Expressions	337
Supported Operators and Functions	339
Grey-Box Style Interface	346
Instruction Format.....	348
Example 1.....	348
Example 2.....	348
Example 3.....	349
Differentiation	349
Solving Non-convex and Non-smooth models	350
Linearization	350
Multistart Scatter Search for Difficult Nonlinear Models	352
Global Optimization of Difficult Nonlinear Models	354
Sample Nonlinear Programming Problems.....	355
Example 1: Black-Box Style Interface:	355
Example 2: Instruction-List Style Interface	361
Example 3: Multistart Solver for Non-Convex Models.....	371
Example 4: Global Solver with MPI Input Format.....	375
Example 5: Grey-Box Style Interface	381
Chapter 8:.....	389
Stochastic Programming	389
Multistage Decision Making Under Uncertainty	389
Recourse Models	391
Scenario Tree	391
Setting up SP Models:.....	393
Loading Core Model:	394

Loading the Time Structure:	397
Loading the Stochastic Structure:.....	399
Monte Carlo Sampling.....	406
Sample SP Problems	411
An Investment Model to Fund College Education:	411
An American Put-Options Model:	413
Appendix 8a: Correlation Specification.....	415
Appendix 8b: Random Number Generation	415
Appendix 8c: Variance Reduction	416
Appendix 8d: The Costs of Uncertainty, EVMU and EVPI	416
Chapter 9:.....	421
Using Callback Functions.....	421
Specifying a Callback Function	421
A Callback Example Using C	424
A Callback Example Using Visual Basic	429
Integer Solution Callbacks.....	430
Chapter 10: Analyzing Models and Solutions	433
Sensitivity and Range Analysis of an LP.....	433
Diagnosis of Infeasible or Unbounded Models.....	435
Infeasible Models.....	435
Unbounded Linear Programs	437
Infeasible Integer Programs	438
Infeasible Nonlinear Programs	438
An Example for Debugging an Infeasible Linear Program.....	438
Block Structured Models	444
Determining Total Decomposition Structures.....	446
Determining Angular Structures	447
Chapter 11: mxLINDO.....	449
A MATLAB Interface.....	449
Introduction.....	449
Setting up MATLAB to Interface with LINDO	449
Using the mxLINDO Interface	450
Calling Conventions	452
mxLINDO Routines	452
Structure Creation and Deletion Routines.....	452
License Information Routines	455
Input-Output Routines	456
Error Handling Routines	464
Parameter Setting and Retrieving Routines	466
Model Loading Routines.....	473
Solver Initialization Routines	486
Optimization Routines	490
Solution Query Routines.....	491
Model Query Routines	498
Model Modification Routines	517
Model and Solution Analysis Routines	534
Advanced Routines.....	541
Callback Management Routines.....	546
Auxiliary Routines.....	552
Sample MATLAB Functions	554

M-functions using mxLINDO.....	554
Chapter 12:.....	557
An Interface to Ox	557
Introduction.....	557
Setting up Ox Interface.....	557
Calling Conventions	558
Example. Portfolio Selection with Restrictions on the Number of Assets Invested.....	560
Appendix A: Error Codes	565
Appendix B:	573
MPS File Format	573
Integer Variables	575
Semi-continuous Variables.....	576
SOS Sets.....	577
SOS2 Example	578
Quadratic Objective.....	579
Quadratic Constraints.....	580
Second Order Cone Constraints	581
Appendix C:.....	585
LINDO File Format	585
Flow of Control	585
Formatting	585
Optional Modeling Statements	587
FREE Statement.....	587
GIN Statement.....	588
INT Statement.....	588
SUB and SLB Statements	589
TITLE Statement.....	589
Appendix D:.....	591
MPI File Format.....	591
Appendix E:	593
SMPS File Format	593
CORE File.....	593
TIME File	593
STOCH File	595
Appendix F:	601
SMPI File Format	601
References	605
INDEX	607

Preface

LINDO Systems is proud to introduce LINDO API 6.0. The general features include a) stochastic optimization b) global and multistart solvers for global optimization, c) nonlinear solvers for general nonlinear optimization, d) simplex solvers for linear optimization e) barrier solvers for linear, quadratic and second-order-cone optimization f) mixed-integer solvers for linear-integer and nonlinear-integer optimization, g) tools for analysis of infeasible linear, integer and nonlinear models, h) interfaces to other systems such as MATLAB, Ox, Java and .NET and i) support of more platforms (see below). The primary solvers in LINDO API 6.0 are:

❑ **Stochastic Solver:**

The stochastic programming solver provides the opportunity of decision making under uncertainty through multistage stochastic models with recourse. The user is required to express the uncertainty by providing distribution functions, either built-in or user-defined, and the stochastic solver will optimize the model to minimize the cost of the initial stage plus the expected value of recourse over the planning horizon. Advanced sampling modes are also available to approximate stochastic parameters from parametric distributions.

❑ **General Nonlinear Solver:**

LINDO API is the first full-featured solver callable library to offer general nonlinear and nonlinear/integer capabilities. This unique feature allows developers to incorporate a single general purpose solver into their custom applications. As with its linear and integer capabilities, LINDO API provides the user with a comprehensive set of routines for formulating, solving, and modifying nonlinear models. The Nonlinear license option is required in order to use the nonlinear capabilities with LINDO API.

❑ **Global Solver:**

The global solver combines a series of range bounding (e.g., interval analysis and convex analysis) and range reduction techniques (e.g., linear programming and constraint propagation) within a branch-and-bound framework to find proven global solutions to non-convex NLPs. Traditional nonlinear solvers can get stuck at suboptimal, local solutions. This is no longer the case when using the global solver.

❑ **Multistart Solver:**

The multistart solver intelligently generates a sequence of candidate starting points in the solution space of NLP and mixed integer NLPs. A traditional NLP solver is called with each starting point to find a local optimum. For non-convex NLP models, the quality of the best solution found by the multistart solver tends to be superior to that of a single solution from a traditional nonlinear solver. A user adjustable parameter controls the maximum number of multistarts to be performed. See Chapter 7, *Solving Nonlinear Models*, for more information.

❑ **Barrier (Interior-Point) Solver:**

Barrier solver is an alternative way for solving linear and quadratic programming problems. LINDO API's state-of-the-art implementation of the barrier method offers great speed advantages for large scale sparse models. LINDO API 6.0 also includes a special variant of the barrier solver specifically designed to solve *Second-Order-Cone* problems. See Chapter 6, *Solving Second-Order-Cone Models*, for more information.

❑ **Simplex Solvers:**

LINDO API 6.0 offers two advanced implementations of the primal and dual simplex methods as the primary means for solving linear programming problems. Its flexible design allows the users to fine tune each method by altering several of the algorithmic parameters.

❑ **Mixed Integer Solver:**

The mixed integer solver's capabilities of LINDO API 6.0 extend to linear, quadratic, and general nonlinear integer models. It contains several advanced solution techniques such as a) cut generation b) tree reordering to reduce tree growth dynamically, and c) advanced heuristic and presolve strategies.

❑ **Model and Solution Analysis Tools:**

LINDO API 6.0 includes a comprehensive set of analysis tools for a) debugging of infeasible linear, integer and nonlinear programs using series of advanced techniques to isolate the source of infeasibilities to smaller subset of the original constraints, b) performing sensitivity analysis to determine the sensitivity of the optimal basis to changes in certain data components (e.g. objective vector, right-hand-side values etc..).

❑ **Quadratic Recognition Tools:**

The QP recognition tool is a useful algebraic pre-processor that automatically determines if an arbitrary NLP is actually a quadratic model. QP models may then be passed to the faster quadratic solver, which is available as part of the barrier solver option.

❑ **Linearization Tools:**

Linearization is a comprehensive reformulation tool that automatically converts many non-smooth functions and operators (e.g., max and absolute value) to a series of linear, mathematically equivalent expressions. Many non-smooth models may be entirely linearized. This allows the linear solver to quickly find a global solution to what would have otherwise been an intractable nonlinear problem.

❑ **Decomposition Tools:**

Many large scale linear and mixed integer problems have constraint matrices that are totally decomposable into a series of independent block structures. A user adjustable parameter can be set, so the solver checks if a model can be broken into smaller independent models. If total decomposition is possible, it will solve the independent problems sequentially to reach a solution for the original model. This may result in dramatic speed improvements. Refer to the *Block Structured Models* section in Chapter 10, *Analyzing Models and Solutions*, for more information.

❑ **Java Native Interface:**

LINDO API includes Java Native Interface (JNI) support for Windows, Solaris, and Linux platforms. This new feature allows users to call LINDO API from Java applications, such as applets running from a browser.

❑ **MATLAB Interface:**

The Matlab interface allows using LINDO API functions from within MATLAB. Using MATLAB's modeling and programming environment, you can build and solve linear, nonlinear, quadratic, and integer models and create custom algorithms based upon LINDO API's routines and solvers.

❑ **.NET Interface:**

LINDO API includes C# and VB.NET interfaces that allow it to be used from within .NET's distributed computing environment (including Windows Forms, ADO.NET, and ASP.NET).

The interfaces are in the form of classes that allow managed .NET code to interact with unmanaged LINDO API code via the "System.Runtime.InteropServices" namespace.

❑ **Ox Interface:**

This interface provides users of the Ox statistical package, the ability to call LINDO API's functions the same way they call native Ox functions. This offers greater flexibility in developing higher-level Ox routines that can set up and solve different kinds of large-scale optimization problems, testing new algorithmic ideas or expressing new solution techniques.

❑ **Platforms:**

LINDO API 6.0 is currently available on Sparc Solaris 32/64 bit, Windows 32/64 bit, Linux 32/64-bit, Mac Intel 32-bit and Mac PowerPC 32-bit. For availability of LINDO API 6.0 on all other platforms, you may wish to contact LINDO Systems, Inc.

LINDO Systems, Inc
1415 N. Dayton
Chicago, Illinois
(312) 988 9421

info@lindo.com
http://www.lindo.com

January 2009

Chapter 1:

Introduction

What Is LINDO API?

The LINDO Application Programming Interface (API) provides a means for software developers to incorporate optimization into their own application programs. LINDO API is designed to solve a wide range of optimization problems, including linear programs, mixed integer programs, quadratic programs, and general nonlinear non-convex programs. These problems arise in areas of business, industry, research, and government. Specific application areas where LINDO API has proven to be of great use include product distribution, ingredient blending, production and personnel scheduling, inventory management... The list could easily occupy the rest of this chapter.

Optimization helps you find the answer that yields the best result; attains the highest profits, output, or happiness; or achieves the lowest cost, waste, or discomfort. Often these problems involve making the most efficient use of your resources—including money, time, machinery, staff, inventory, and more. Optimization problems are often classified as linear or nonlinear, depending on whether the relationships in the problem are linear with respect to the variables.

The most fundamental type of optimization problems is the *linear program* (LP) of the form:

Minimize (or maximize) $c_1x_1 + c_2x_2 + \dots + c_nx_n$

Such that

$$A_{11}x_1 + A_{12}x_2 + \dots + A_{1n}x_n \ ? \ b_1$$

$$A_{21}x_1 + A_{22}x_2 + \dots + A_{2n}x_n \ ? \ b_2$$

$$\vdots \quad \dots \quad \vdots$$

$$A_{m1}x_1 + A_{m2}x_2 + \dots + A_{mn}x_n \ ? \ b_m$$

$$L_1 \leq x_1 \leq U_1$$

$$L_2 \leq x_2 \leq U_2$$

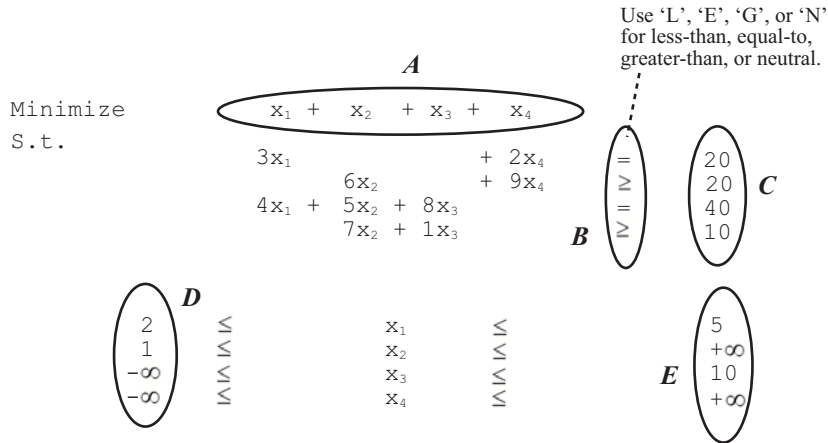
$$\vdots$$

$$L_n \leq x_n \leq U_n$$

where A_{ij} , c_j , b_i , L_j , U_j are known real numbers; ? is one of the relational operators ‘≤’, ‘=’, or ‘≥’; and x_1, x_2, \dots, x_n are the decision variables (unknowns) for which optimal values are sought.

The expression being optimized is called the objective function and c_1, c_2, \dots, c_n are the objective coefficients. The relationships whose senses are expressed with ? are the constraints; $A_{i1}, A_{i2}, \dots, A_{in}$ are the coefficients; and b_i is the right-hand side value for the i^{th} constraint. L_j and U_j represent lower and upper bounds for the j^{th} decision variable and can be finite or infinite.

The diagram below shows how each component of LP data, except the coefficients of the constraint matrix, can be trivially represented by vectors (arrays). The circled elements labeled $A, B, C, D,$ and E in the following figure symbolize these components and refer to *objective coefficients*, *constraint senses*, *right-hand sides*, *lower-bounds*, and *upper-bounds*, respectively.



In this small example, these vectors translate to the following:

$$\begin{aligned} A &= [1 \quad 1 \quad 1 \quad 1]. \\ B &= [E \quad G \quad E \quad G]. \\ C &= [20 \quad 20 \quad 40 \quad 10]. \\ D &= [2 \quad 1 \quad -LS_INFINITY \quad -LS_INFINITY]. \\ E &= [5 \quad LS_INFINITY \quad 10 \quad LS_INFINITY]. \end{aligned}$$

Each of these vectors can be represented with an array of appropriate type and passed to LINDO API via model loading routines. Although it is also possible to represent the coefficients of the constraint matrix with a single vector, a different representation, called the *sparse matrix representation*, has been adopted. This is discussed in more detail below.

Sparse Matrix Representation

LINDO API uses a sparse matrix representation to store the coefficient matrix of your model. It represents the matrix using three (or optionally four) vectors. This scheme is utilized, so it is unnecessary to store zero coefficients. Given that most matrix coefficients in real world math programming models are zero, this storage scheme proves to be very efficient and can drastically reduce storage requirements. Below is a brief explanation of the representation scheme.

We will use the coefficients of the constraint matrix in our sample LP from above. These are as follows:

$$\begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ \left[\begin{array}{cccc} 3 & 0 & 0 & 2 \\ 0 & 6 & 0 & 9 \\ 4 & 5 & 8 & 0 \\ 0 & 7 & 1 & 0 \end{array} \right] \end{array}$$

Three Vector Representation

Three vectors can represent a sparse matrix in the following way. One vector will contain all of the nonzero entries from the matrix, ordered by column. This is referred to as the *Value* vector. In our example, this vector has 9 entries and looks like:

$$\text{Value} = [3 \ 4 \ 6 \ 5 \ 7 \ 8 \ 1 \ 2 \ 9].$$

Note that all of the entries from the first column appear first, then the entries from the second column, and so on. All of the zeros have been stripped out.

In the second vector, which we call the *Column-start* vector, we record which points in the *Value* vector represent the start of a new column from the original matrix. The n^{th} entry in the *Column-start* vector tells us where in the *Value* vector to find the beginning of the n^{th} column. For instance, the column starts for the *Value* vector of our small example are underlined in the following diagram. Note that LINDO API uses zero-based counting, so the *Column-start* vector is as follows:

$$\begin{array}{cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \text{Value} = [& \underline{3} & 4 & \underline{6} & 5 & 7 & \underline{8} & 1 & \underline{2} & 9] . \\ & \swarrow & & \swarrow & & \swarrow & \uparrow & & \swarrow & \uparrow \\ \text{Column-start} = [& 0 & 2 & 5 & 7 & 9] . \end{array}$$

Note that the *Column-start* vector has one more entry than there are columns in our matrix. The extra entry tells LINDO where the last column ends. It will always be equal to the length of the *Value* vector.

From the *Column-start* vector, we can deduce which column is associated with each entry in our *Value* vector. The only additional information that we need is the row numbers of the entries. We store this information in a third vector, the *Row-index* vector. This vector is the same length as the *Value* vector. Each entry in the *Row-index* vector tells which row the corresponding entry from the *Value* vector belongs to. In our example, the number 3 belongs to the first row, which we call row 0, so the first entry in the *Row-index* vector is 0. Similarly, the second entry in the *Value* vector (4), belongs to the third row (row 2 when starting from zero), so the second entry of the *Row-index* vector is 2. Continuing in this way through the rest of the entries of the *Value* vector, the resulting *Row-index* vector appears as follows:

$$\begin{array}{cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \text{Row-index} = [& 0 & 2 & 1 & 2 & 3 & 2 & 3 & 0 & 1] . \end{array}$$

In summary, our transformation from a matrix into 3 vectors is:

$$\begin{bmatrix} 3 & 0 & 0 & 2 \\ 0 & 6 & 0 & 9 \\ 4 & 5 & 8 & 0 \\ 0 & 7 & 1 & 0 \end{bmatrix} \Rightarrow \begin{array}{ll} \text{Column-starts:} & [0 \ 2 \ 5 \ 7 \ 9] \\ \text{Value:} & [3 \ 4 \ 6 \ 5 \ 7 \ 8 \ 1 \ 2 \ 9] \\ \text{Row-index:} & [0 \ 2 \ 1 \ 2 \ 3 \ 2 \ 3 \ 0 \ 1] \end{array}$$

Four Vector Representation

The four vector representation allows more flexibility than the three vector representation. Use it when you expect to add rows to your original matrix (i.e., if you will be adding additional constraints to your model).

The four vector representation uses the same three vectors as above. However, it allows you to have “blanks” in your *Value* vector. Because of this, you must also pass a vector of column lengths, since the solver doesn’t know how many blanks there will be.

For example, suppose we wish to leave room for one additional row. Then, our *Value* vector becomes:

$$\text{Value} = [3 \ 4 \ X \ 6 \ 5 \ 7 \ X \ 8 \ 1 \ X \ 2 \ 9 \ X]$$

where the X ’s represent the blanks. The blanks may be nulls or any other value, since they will be ignored for the time being.

Our *Column-start* vector becomes:

$$\begin{array}{ccccccccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ \text{Value} = [& 3 & 4 & X & 6 & 5 & 7 & X & 8 & 1 & X & 2 & 9 & X] . \\ & \nearrow & & \nearrow & & \nearrow & & \nearrow & & \nearrow & & \nearrow & & \nearrow \\ \text{Column-start} = [& 0 & 3 & 7 & 10 & 13] . \end{array}$$

Our new vector is the *Column-length* vector. It will contain the length of each column (i.e., the number of nonzeros in each column). This allows the solver to skip the blanks (X ’s) in the *Value* vector. In our small example, since the first column contains two nonzero and nonblank entries, the first element of the *Column-length* vector will be 2. Continuing through the remaining columns, the *Column-length* vector and its corresponding entries from the *Value* vector are as follows:

$$\begin{array}{l} \text{Column-length} = [2 \ 3 \ 2 \ 2] . \\ \text{Value} = [\underline{3} \ \underline{4} \ X \ \underline{6} \ \underline{5} \ \underline{7} \ X \ \underline{8} \ \underline{1} \ X \ \underline{2} \ \underline{9} \ X] . \end{array}$$

Our *Row-index* vector is as before, except we add a blank for each blank in the *Value* vector. As with the *Value* vector, these blanks will be ignored, so they can contain any value. Thus, the *Row-index* vector becomes:

$$\text{Row-index} = [0 \ 2 \ X \ 1 \ 2 \ 3 \ X \ 2 \ 3 \ X \ 1 \ 2 \ X] .$$

In summary, the four vector transformation is:

$$\begin{bmatrix} 3 & 0 & 0 & 2 \\ 0 & 6 & 0 & 9 \\ 4 & 5 & 8 & 0 \\ 0 & 7 & 1 & 0 \end{bmatrix} \Rightarrow \begin{array}{l} \text{Column lengths:} \ [2 \ 3 \ 2 \ 2] \\ \text{Column starts:} \ [0 \ 3 \ 7 \ 10 \ 13] \\ \text{Values:} \ [3 \ 4 \ X \ 6 \ 5 \ 7 \ X \ 8 \ 1 \ X \ 2 \ 9 \ X] \\ \text{Row indexes:} \ [0 \ 2 \ X \ 1 \ 2 \ 3 \ X \ 2 \ 3 \ X \ 0 \ 1 \ X] \end{array}$$

Simple Programming Example

Up to this point, we have seen that the objective function coefficients, right-hand side values, constraint senses, and variable bounds can be stored in vectors of appropriate dimensions and the constraint matrix can be stored in three or four vectors using the sparse matrix representation. In this section, we show how these objects should be declared, assigned values, and passed to LINDO API to complete the model setup phase and invoke optimization.

Recall the small LP example model from the array representation section above:

$$\begin{array}{rllll}
 \text{Minimize} & x_1 + x_2 + x_3 + x_4 & & & \\
 \text{S.t.} & & & & \\
 & 3x_1 & & + 2x_4 & = 20 \\
 & & 6x_2 & & + 9x_4 \geq 20 \\
 & 4x_1 + 5x_2 & + 8x_3 & & = 40 \\
 & & 7x_2 & + 1x_3 & \geq 10 \\
 & 2 & \leq x_1 & \leq & 5 \\
 & 1 & \leq x_2 & \leq & +\infty \\
 & -\infty & \leq x_3 & \leq & 10 \\
 & -\infty & \leq x_4 & \leq & +\infty
 \end{array}$$

It is easy to verify that the model has 4 variables, 4 constraints, and 7 nonzeros. As determined in the previous section, its constraint matrix has the following (three-vector) sparse representation:

$$\begin{array}{rll}
 \text{Column-start} & = & [0 \ 2 \ 5 \ 7 \ 9] \\
 \text{Values} & = & [3.0 \ 4.0 \ 6.0 \ 5.0 \ 7.0 \ 8.0 \ 1.0 \ 2.0 \ 9.0] \\
 \text{Row-index} & = & [0 \ 2 \ 1 \ 2 \ 3 \ 2 \ 3 \ 0 \ 1]
 \end{array}$$

Other components of the LP data, as described above, are:

$$\begin{array}{rll}
 \text{Right-hand side values} & = & [20 \ 20 \ 40 \ 10] . \\
 \text{Objective coefficients} & = & [1 \ 1 \ 1 \ 1] . \\
 \text{Constraint senses} & = & [E \ G \ E \ G] . \\
 \text{Lower bounds} & = & [2 \ 1 \ -\text{LS_INFINITY} \ -\text{LS_INFINITY}] . \\
 \text{Upper bounds} & = & [5 \ \text{LS_INFINITY} \ 10 \ \text{LS_INFINITY}] .
 \end{array}$$

Create an Environment and Model

Before any data can be input to LINDO API, it is necessary to request LINDO API to initialize the internal solvers by checking the license this user has and to get handles of the required resources (e.g., pointers to internal memory areas). This is achieved by creating a LINDO environment object and creating a model object within the environment. These reside at the highest level of LINDO API's internal object oriented data structure. In this structure, a model object belongs to exactly one environment object. An environment object may contain zero or more model objects.

The following code segment does this:

```

/* declare an environment variable */
pLSEnv pEnv;

/* declare a model variable */
pLSmodel pModel;

/* Create the environment./
pEnv = LScreateEnv ( &nErrorCode, MY_LICENSE_KEY);

/* Create the model./
pModel = LScreateModel ( pEnv, &nErrorCode);

```

The environment data type, *pLSenv*, and the model data type, *pLSmodel*, are both defined in the *lindo.h* header file. A call to *LScreateEnv()* creates the LINDO environment. Finally, the model object is created with a call to *LScreateModel()*. For languages other than C/C++ *pLSenv* and *pLSmodel* objects refer to integer types. The associated header files are located in the ‘lindoapi/include’ directory.

Load the Model

The next step is to set up the LP data and load it to LINDO API. This is generally the most involved of the steps.

Objective

The following code segment is used to enter the direction of the objective. The possible values for the direction of the objective are *LS_MAX* and *LS_MIN*, which are predefined macros that stand for maximize or minimize. For our sample problem, the objective direction is given as maximization with the following code:

```
int nDir = LS_MIN;
```

The constant terms in the objective function are stored in a double scalar with the following:

```
double dObjConst = 0.0;
```

Finally, the objective coefficients are placed into an array with the following:

```
double adC[4] = { 1., 1., 1., 1.};
```

Constraints

The following code segment is used to enter the number of constraints:

```
int nM = 4;
```

The constraint right-hand sides are placed into an array with the following:

```
double adB[4] = { 20., 20., 40., 10. };
```

The constraint types are placed into an array with the following:

```
char acConTypes[4] = { 'E', 'G', 'E', 'G' };
```

The number of nonzero coefficients in the constraint matrix is stored:

```
int nNZ = 9;
```

Finally, the length of each column in the constraint matrix is defined. This is set to NULL in this example, since no blanks are being left in the matrix:

```
int *pnLenCol = NULL;
```

The nonzero coefficients, column-start indices, and the row indices of the nonzero coefficients are put into arrays with the following:

```
int anBegCol[5] = { 0 , 2 , 5 , 7 , 9 };
double adA[9] = { 3.0, 4.0, 6.0, 5.0, 7.0, 8.0, 1.0, 2.0, 9.0 };
int anRowX[9] = { 0 , 2 , 1 , 2 , 3 , 2 , 3 , 0 , 1 };
```

Note: Refer to the section *Sparse Matrix Representation* above for more information on representing a matrix with three or four vectors.

Variables

The following code segment is used to declare the number of variables:

```
int nN = 4;
```

The upper and lower bounds on the variables are defined with the following:

```
double pdLower[4] = {2, 1, -LS_INFINITY, -LS_INFINITY};
double pdUpper[4] = {5, LS_INFINITY, 10, LS_INFINITY};
```

Then, the variable types are placed into an array with the following:

```
char acVarTypes[4] = {'C', 'C', 'C', 'C'};
```

The variable types could actually be omitted and LINDO API would assume that the variables were continuous.

We have now assembled a full description of the model and pass this information to LINDO API with the following:

```
nErrorCode = LSloadLPData( pModel, nM, nN, nDir, dObjConst, adC, adB,
acConTypes, nNZ, anBegCol, pnLenCol, adA, anRowX, pdLower, pdUpper);
```

All LINDO API functions return an error code indicating whether the call was successful or not. If the call was successful, then the error code is zero. Otherwise, an error has occurred and its type could be looked up in Appendix A, *Error Codes*. It is imperative that the error code returned is always checked to verify that the call was successful.

Note: If there is a nonzero error code, the application program should stop, since the results would be unpredictable and it may cause the program to crash.

Solve

Since the model is an LP, a linear solver, such as the primal simplex method, can be used. The model is solved with the following call:

```
nErrorCode = LSoptimize( pModel, LS_METHOD_PSIMPLEX, &nSolStat);
```

Alternative solvers available for linear models include dual simplex and barrier (if licensed). When the second argument in the function call is set to `LS_METHOD_FREE`, LINDO API will decide the solver to use by examining its structure and mathematical content. See the *Common Macro Definitions* section of Chapter 2, *Function Definitions*, for more information on the predefined macros `LS_METHOD_PSIMPLEX` and `LS_METHOD_FREE`.

Retrieve the Solution

The next step is to retrieve the solution using solution query functions. Many of the LINDO API query functions need to have space allocated before calling the routine. You must be sure to allocate sufficient space for query routines that include a pointer to a string, an integer vector, a double precision vector, or character vector. If sufficient memory is not initially allocated, the application will crash once it is built and executed. See *Solution Query Routines* in Chapter 2, *Function Definitions*, for more information on which routines require space to be allocated for them. Refer to Chapter 3, *Solving Linear Programs*, for more details on building and solving the model and a programming example in Visual Basic.

Here, the objective value and optimal variable values will be displayed. The objective value is retrieved and printed with the following:

```
double adX[4];
nErrorCode = LSgetInfo( pModel, LS_DINFO_POBJ, &dObj);
printf( "Objective Value = %g\n", dObj);
```

See the context of the *LSgetInfo()* function in Chapter 2, *Function Definitions*, for more information on the predefined macro `LS_DINFO_POBJ`. It tells LINDO API to fetch the value of the primal objective value via the *LSgetInfo()* function. The optimal variable values are retrieved and printed with the following:

```
nErrorCode = LSgetPrimalSolution ( pModel, adX);
printf ("Primal values \n");
for (i = 0; i < nN; i++) printf( " x[%d] = %g\n", i, adX[i]);
printf ("\n");
```

The output of this program would appear as follows:

```
Objective Value = 10.44118
Primal values
x[0] = 5
x[1] = 1.176471
x[2] = 1.764706
x[3] = 2.5
```

Clear Memory

A last step is to release the LINDO API memory by deleting the LINDO environment with the following call:

```
nErrorCode = LSdeleteEnv( &pEnv);
```

This frees up all data structures LINDO API allocated to the environment and all of the environment's associated models.

References

Birge, J. and F. Louveaux(1997), *Introduction to Stochastic Programming*, Springer.

L'Ecuyer, P., R. Simard, E. Chen, and W. Kelton(2002), "An Object-Oriented Random-Number Package with Many Long Streams and Substreams", *Operations Research*, vol. 50, no. 6, pp. 1073-1075.



INDEX

1

100% rule, 434

A

absolute optimality tolerance, 73
 absolute value, 339, 350, 368
 Add Module command, 269, 430
 adding
 constraints, 194, 518
 variables, 194, 195, 520
 addition, 339
 AddressOf operator, 233, 235, 429
 advanced routines, 221, 541
 algebraic reformulation, 86
 algorithm
 barrier, 295
 generalized reduced gradient, 3
 alternate optima, 158
 analysis routines, 210, 534
 analyzing models and solutions, 433
 AND function, 339, 343
 angular block structure, 445, 534
 annuity, 340
 antithetic variate, 98, 249, 407
 antithetic variates, 416
 API
 callback functions, 227
 error messages, 565
 examples, 259
 function definitions, 17
 arc sine, 340
 arc tangent, 340
 arguments, right-hand side, 452
 ASCII text format, 26
 asset investing, 304
 asymmetric Q matrix, 300
 automatic differentiation, 349
 auxiliary routines, 552
 available parameters, 53, 99
 average, 342

B

backward transformation, 221, 541
 barrier algorithm, 295
 barrier solver, 77, 94

iterations, 227
 license, 92
 solving, 53, 58, 60, 80, 139, 265
 basis, 542
 crossover, 58
 cuts, 227
 forward transformation, 222
 getting, 144, 147, 491
 loading, 486
 MIPs, 493
 parameters, 73
 warm start, 135
 Beasley, J., 304
 Beta distribution, 416
 Big M, 70, 351
 binary variables, 26, 167, 504, 587, 588
 binomial distribution, 341
 Binomial distribution, 416
 Birge, J., 411
 black-box interface, 330, 353
 example, 355
 blanks, 573, 574
 block structured models, 444
 finding, 210, 534
 getting, 213, 537
 loading, 137, 488
 parameters, 55
 bounds
 assets invested, 304
 best bounds, 212, 536
 defaults, 286, 293, 587, 588, 589
 free variables, 573, 587
 global optimization, 87
 MATLAB, 475, 476, 503, 520, 552, 553
 modifying, 206, 209, 529, 533
 MPS files, 573
 name, 119, 168, 477, 505
 objective bounds, 167, 432, 504
 ranges, 214, 433, 538
 real bound, 78
 risk of loss, 300
 running time, 76
 SUB/SLB, 587, 589
 type, 575
 variable upper/lower, 118, 166, 196, 276, 587
 branch-and-bound
 cuts, 70
 limits, 69, 79, 82, 83, 89
 solver, 142, 287, 293, 375, 491
 solver status, 230, 547

-
- branching
 - branch count, 227, 432
 - branch direction, 73, 75, 85, 86
 - global optimization, 82
 - priorities, 71, 124, 136, 138, 486, 489
 - strong branching, 78
 - variable branching, 78, 486
 - BTRAN, 221, 541
 - building an application, 267
- C**
- C example, 259, 421, 424
 - C++ example, 280
 - debugging, 435–44
 - callback functions, 355, 421
 - definitions, 227, 546
 - double precision, 423
 - examples, 421–32
 - frequency, 53
 - MIPs, 235, 279, 550
 - query routines, 227, 546
 - callback management routines, 227, 546
 - callback.bas, 429
 - CALLBACKTYPE, 422
 - calling conventions, 452, 558
 - capitalization, 6, 567, 574
 - cardinality constraints, 304
 - Cauchy distribution, 416
 - cdecl protocol, 422
 - CheckErr(), 273
 - Chisquare distribution, 416
 - Chi-squared distribution, 341
 - Cholesky decomposition, 326
 - class module, 429
 - ClassWizard, 282
 - clique cuts, 227
 - coefficients
 - adding, 194, 196, 520
 - backward transformation, 221, 541
 - C++ example, 264, 285, 286
 - coefficient matrix, 10, 166, 265
 - forward transformation, 542
 - getting, 164, 165, 167, 502, 503, 504, 552
 - left-hand side, 587
 - linearization, 351
 - loading, 117, 118, 473, 476, 553
 - modifying, 204, 527
 - number of, 164, 167, 265, 502, 504
 - quadratic, 121, 197
 - range of, 56
 - reduction, 61, 69, 76, 81
 - right-hand side, 275, 518
 - sparse format, 114
 - storing, 276
 - Visual Basic example, 291, 292
 - column
 - column length, 12, 165, 196, 285, 293
 - column start, 11, 12, 285, 291
 - file format, 195, 196, 520
 - MATLAB, 556
 - names, 119, 477
 - nonlinear, 120, 170, 478, 506, 507, 508, 509
 - comments, 586
 - compiling, 266
 - complement function, 339
 - complementarity, 352
 - cone optimization, 311
 - conjunction, 339
 - constant term, 115, 117, 165, 205, 206, 335, 474, 476, 503, 530, 552, 553
 - constraints, 291, 585
 - adding, 194, 518
 - C++ example, 264, 286
 - cardinality, 304
 - complementarity, 352
 - cuts, 72
 - deleting, 200, 524
 - equal to, 116, 474
 - errors, 565
 - forcing, 351
 - get, 160, 162, 164, 165, 498, 500, 502, 503, 552
 - greater than, 116, 152, 474
 - GUB, 69
 - index of, 127, 164, 223, 224, 543, 544
 - internal index, 501
 - left-hand sides, 587
 - less than, 116, 152, 474
 - limit, 589
 - loading, 118, 476, 553
 - matrix, 118, 166, 265, 476, 503
 - modifying, 204, 205, 207, 527, 528
 - names, 119, 168, 501, 585
 - nonlinear data, 169, 506, 508, 509
 - number of, 91, 115, 117, 152, 167, 473, 474, 476, 503, 504
 - Pluto Dogs example, 285
 - quadratic, 121, 174, 197, 295
 - ranges, 215, 433, 538
 - right-hand sides, 264, 531, 586
 - selective evaluation, 60
 - splitting, 586
 - status, 144
 - storing, 276
 - violated, 55, 63
 - Visual Basic example, 292
 - continuous model, 139, 152, 490
 - continuous variables, 139, 148, 158, 352, 589
 - priorities, 136
 - contra cuts, 227
-

converting models to SOCP form, 321
 convex models, 54, 64, 350, 353, 354, 371
 convexification, 83
 core file, 34, 36
 core model, 131, 132, 133, 155, 157, 187, 191, 393, 411
 correlation, 415
 cosine, 340
 covariance, 295
 crashing, 60, 64
 creating
 environment, 453
 model, 453
 creation routines, 21
 crossover, 58, 140
 cutoff value, 54, 58, 72, 77, 81
 cuts
 depth, 70
 frequency, 70
 max passes, 71
 total generated, 227
 types of, 69, 70, 72

D

data
 fields, 282
 formulation, 164, 165, 167, 502, 503, 504
 getting, 162, 421, 500
 global, 21, 421, 428, 431
 lines, 575
 loading, 124
 name, 119, 168, 477, 505
 passing, 276
 quadratic, 510, 511
 storing, 21
 structures, 21, 262, 266, 428
 types, 17, 42, 263
 debug, 435
 example, 438
 parameters, 93
 decision variables, 285, 291
 decomposition, viii, 445
 angular structures, 447
 Dantzig-Wolfe, 556
 finding, 211, 534
 getting, 488, 537
 loading, 138
 parameters, 55, 86
 total, 446
 default bounds, 286, 293, 587, 588, 589
 definitions, 17
 deletion routines, 21, 200, 203, 524
 examples, 266, 277
 MATLAB, 454

 nonlinear programming, 330
 variables, 526
 Delphi, 277
 delta tolerance, 70, 83, 351
 derivatives, 335
 accuracy, 349
 calculating, 224, 226, 336, 544, 545
 discontinuous, 337
 examples, 355
 finite differences, 60, 63
 getting, 170, 171, 507
 setting, 234, 549
 deterministic equivalent, 3, 20, 37, 97
 Devex pricing, 57, 58
 differentiation, 349
 dimensions of model, 42, 275
 direction
 of constraints, 205
 of objective, 264, 285, 291
 to branch, 73, 75
 disaggregation, 69, 227
 discontinuous derivatives, 337
 discrete variables, 352
 disjunction, 339
 Distribution Function Macros, 256
 division, 339
 double precision, 348, 368
 callback functions, 423, 432
 getting parameters, 44, 46, 466, 468
 parameters, 42
 setting parameters, 48, 50, 469, 471
 dual
 models, 29, 30, 54, 459
 objective, 227
 reductions, 61, 76, 81
 simplex, 57, 60, 69, 77, 80, 94, 139, 265
 solution, 148
 values, 145, 423, 492, 493
 writing, 459
 dual angular structure, 210, 445, 534

E

e, 340
 ector Push, 345
 educational license, 92
 eigenvalue, 295
 embedded blanks, 573, 574
 END, 585
 engineering design, 311
 enumeration solver, 74
 environment, 22
 creating, 21, 259, 273, 453
 deleting, 21, 22, 454
 space, 21

-
- variables, 267
 - EP_ABS, 339
 - EP_ACOS, 340
 - EP_ACOSH*, 346
 - EP_AND, 339
 - EP_ASIN, 340
 - EP_ASINH*, 345
 - EP_ATAN, 340
 - EP_ATAN2, 340
 - EP_ATANH*, 346
 - EP_AVG, 342
 - EP_COS, 340
 - EP_COSH*, 345
 - EP_DIVIDE, 339
 - EP_EQUAL, 339
 - EP_EXP, 340
 - EP_EXPOINV, 343
 - EP_EXT_AND, 343
 - EP_FALSE, 340
 - EP_FLOOR, 340
 - EP_FPA, 340
 - EP_FPL, 341
 - EP_GTHAN, 339
 - EP_GTOREQ, 339
 - EP_IF, 340
 - EP_LGM, 340
 - EP_LN, 339
 - EP_LNX*, 346
 - EP_LOG, 339
 - EP_LOGB*, 346
 - EP_LOGX*, 346
 - EP_LTHAN, 339
 - EP_LTOREQ, 339
 - EP_MAX, 343
 - EP_MIN, 342
 - EP_MINUS, 339
 - EP_MOD, 340
 - EP_MULTINV, 344
 - EP_MULTIPLY, 339
 - EP_NEGATE, 339
 - EP_NO_OP, 339
 - EP_NORMDENS, 343
 - EP_NORMINV, 343
 - EP_NORMSINV*, 346
 - EP_NOT, 339
 - EP_NOT_EQUAL, 339
 - EP_NPV, 343
 - EP_OR, 339
 - EP_PBN, 341
 - EP_PCX, 341
 - EP_PEB, 341
 - EP_PEL, 341
 - EP_PERCENT, 339
 - EP_PFD, 342
 - EP_PFS, 342
 - EP_PHG, 342
 - EP_PI, 339
 - EP_PLUS, 339
 - EP_POWER, 339
 - EP_PPL, 341
 - EP_PPS, 341
 - EP_PSL, 340
 - EP_PSN, 340
 - EP_PTD, 341
 - EP_PUSH_NUM, 343
 - EP_PUSH_OR, 343
 - EP_PUSH_VAR, 343
 - EP_RAND, 342
 - EP_SIGN, 340
 - EP_SIN, 340
 - EP_SINH*, 345
 - EP_SQR*, 345
 - EP_SQRT, 339
 - EP_SUM, 342
 - EP_SUMIF, 344
 - EP_SUMPROD, 344
 - EP_TAN, 340
 - EP_TANH*, 345
 - EP_TRIAINV, 343
 - EP_TRUE, 340
 - EP_TRUNC*, 346
 - EP_UNIFINV, 344
 - EP_USER, 342
 - EP_USER operator, 346
 - EP_USRCOD, 344
 - EP_VLOOKUP, 344
 - EP_VMULT*, 345
 - EP_VPUSH_NUM, 345
 - EP_VPUSH_VAR, 345
 - EP_WRAP, 341
 - equal to
 - constraints, 118, 164, 165, 194, 502
 - error messages, 565
 - operators, 339, 586
 - quadratic programs, 295
 - Erlang loss, 341
 - error codes, 219, 220, 464, 565
 - error handling routines, 53, 219, 273, 464
 - EVMU, 109, 417
 - EVPI, 97, 109, 417
 - examples
 - callback functions, 421–32
 - debugging, 438
 - linear programs, 259
 - MATLAB, 554
 - programming in C, 259, 421
 - Visual Basic, 287
 - exclamation mark, 586
 - exp() function, 340
 - expiration, 58, 79, 91, 568
-

exponential distribution, 343, 416

F

F distribution, 342, 416
 false, 340
 feasibility tolerance, 55, 63
 fields, 282
 file formats, 25

- ASCII text format, 26
- column format, 195, 196, 520
- error messages, 565
- LINDO, 585
- LINGO, 25, 32, 461
- MPI, 27, 375, 457, 565, 591, 593, 601
- MPS, 25, 26, 169, 573
- row format, 194, 585

 file input, 7
 finance, 300
 finite differences, 335

- black-box interface, 355
- coefficients, 121
- derivatives, 60, 63, 234, 549
- gradients, 235, 336
- instruction-list interface, 349

 finite source queue, 342
 first order approximations, 59
 fixed variables, 72, 81, 573
 floating point tolerance, 82
 FLOOR function, 340
 flow cover, 69, 227
 forcing constraints, 351
 form module, 429
 formatted MPS file, 26
 formulation data, 164, 165, 167, 502, 503, 504
 forward transformation, 222, 542
 four vector representation, 12
 FREE, 587
 free variables, 54, 573, 587, 588
 frequency of callbacks, 53
 frequency of cuts, 70
 frontend, 329
 FTRAN, 222, 542
 full rank, 295
 Funcalc(), 334
 functions

- definitions, 17
- objective, 57, 291, 585, 586
- postfix notation, 339
- prefixes, 17
- prototypes, 262

 functions to callback, 355, 421

- definitions, 227, 546
- frequency, 53
- MIPs, 235, 279, 550

G

Gamma distribution, 416
 gamma function, 340
 gaussian distributions, 353
 GCD cuts, 69, 227
 general integers, 167, 279, 504, 587, 588
 generalized upper bound, 69
 Geometric distribution, 416
 getting

- constraints, 162, 164, 500, 502
- data, 162, 421, 500
- parameters, 43, 44, 127, 466, 467, 468
- variable types, 516

 GIN, 167, 279, 504, 587, 588
 global data, 21, 421, 428, 431
 global optimality, 62
 global optimization

- cuts, 227
- non-convex models, 353
- nonlinear models, 350, 354
- parameters, 82, 91
- quadratic programs, 295
- solving, 141, 490

 global solver, vii, 3, 92, 354, 375
 Gomory cuts, 69, 227
 Gradcalc(), 336
 gradient, 3, 63, 120, 329, 336, 371, 478
 greater than, 118, 164, 165, 194, 502

- constraints, 152
- errors, 565
- example, 264, 285, 291
- operator, 586
- postfix notation, 339

 grey-box interface, 330, 346

- example, 381

 GUB cuts, 69, 227
 Gumbel distribution, 416

H

handler code, 282
 hashing, 238
 header file, 21, 53, 262, 263, 268, 452
 heuristic, 72, 74
 histogram, 191
 Hungarian notation, 17, 452
 Hypergeometric distribution, 416
 hypergeometric probability, 342

I

IF() function, 340
 IIS, 31, 435, 460

- finding, 211

- getting, 216, 539
 - MATLAB, 535
 - parameters, 93
 - incumbent solution, 83, 84, 227, 430
 - indefinite, 295
 - independent block structure, 444
 - index
 - of a row, 166, 176, 196, 276, 504, 513, 520
 - of constraints, 127, 164, 223, 224, 543, 544
 - inequality operators, 586
 - infeasibilities, 227
 - MATLAB, 535, 539
 - primal infeasibility, 423, 546
 - rounded solutions, 588, 589
 - solver status, 433
 - infeasible solution, 31, 216, 435, 460
 - infinity, 587
 - infix notation, 337
 - inheriting, 42
 - initial values, 135, 139, 179, 486, 487, 489, 515
 - initialization of solver, 135, 486
 - inner product, 344
 - Input/Output, of models, 25
 - instruction-list interface, 115, 116, 330, 337, 474
 - example, 361
 - instruction format, 348
 - INT, 587, 588
 - integer part, 340
 - integer programming. *See also* mixed-integer programming
 - callback functions, 235, 430, 432, 550
 - constraint cuts, 72
 - cut level, 69, 70
 - examples, 279
 - getting, 44, 46
 - heuristics, 74
 - internal index, 178, 501, 514
 - loading, 124
 - optimality tolerance, 73
 - setting, 49, 50
 - slack values, 150, 152, 497
 - integer variables
 - binary, 587, 588
 - block structure, 137
 - bounded, 573
 - branching priorities, 136, 138, 489
 - general, 167, 279, 504, 587, 588
 - integer feasible tolerance, 73, 77
 - limit, 91
 - parameters, 42
 - solving for, 139, 148, 158
 - variable status, 144, 147, 213
 - integrality, 69, 287, 293
 - interface, 329, 421
 - black-box, 330, 353, 355
 - callback function, 428
 - grey-box, 346, 381
 - instruction list, 330, 337, 361
 - java, viii
 - MATLAB, viii, 449
 - nonlinear, 329
 - interior point algorithm, 295
 - interior point solver, 58, 60, 77, 80, 92, 94, 139, 265
 - Interior-Point Solver Programs
 - parameters, 65
 - internal error, 566
 - internal index
 - constraints, 501
 - getting, 164, 167
 - variables, 178, 179, 514, 515
 - interrupt solver, 53, 422, 430, 568
 - inverse of standard Normal, 343, 346
 - inverse transform of cdf, 416
 - investing, 304
 - irreducibly inconsistent set, 31, 435, 460
 - finding, 211
 - getting, 216, 539
 - MATLAB, 535
 - parameters, 93
 - irreducibly unbounded set, 31, 437, 460
 - finding, 211
 - getting, 217, 540
 - MATLAB, 535
 - parameters, 93
 - iterations, 227
 - barrier, 432
 - callback functions, 428, 430
 - iteration limit, 56, 63, 65, 567
 - nonlinear, 432
 - simplex, 432
 - IUS, 31, 435, 437, 460
 - finding, 211
 - getting, 217, 540
 - MATLAB, 535
 - parameters, 93
- ## J
- Jacobian, 169, 170, 172, 349, 478, 506, 507, 509
 - java interface, viii
 - JNI, viii
- ## K
- K-best solutions, 158
 - Kendall tau, 415
 - knapsack cuts, 69, 227
 - knapsack solver, 74
-

L

- Laplace distribution, 416
- Latin hypercube sampling, 98, 249, 407, 408, 409, 416
- Latin square sampling, 98, 249, 407, 408, 409, 416
- lattice cuts, 69, 227
- leading blanks, 573
- left-hand sides, 587
 - arguments, 452
- length of column, 12, 196, 285, 293
- length of objective, 369
- less than, 118, 164, 165, 194, 502
 - constraints, 152
 - errors, 565
 - example, 264, 275
 - operator, 586
 - postfix notation, 339
- license
 - barrier, 92, 265, 297, 315
 - C++ example, 6
 - educational, 92
 - error messages, 567, 568
 - expiration, 91
 - global, 92
 - license key, 21, 23
 - MATLAB, 453, 455
 - nonlinear, 92, 297, 315
 - reading, 24
 - runtime, 92
 - trial, 91
- license key, 6
- LIFO stack, 592
- lifting cuts, 69
- limits
 - branch-and-bound, 79, 82
 - constraints, 589
 - integer variables, 91
 - iteration, 56, 63, 65, 567
 - license expiration, 91
 - time limit, 58, 70, 79, 80, 89, 91, 568
 - variables, 91
- LINDO contact information, ix
- LINDO format, 25, 585
 - reading, 25, 456
 - writing, 29, 32, 458, 461
- lindo.bas, 269
- lindo.h, 262, 268, 269, 431
- lindo.par, 8
- linear loss function, 340, 341
- linear models, 353
- linear programming, 1, 78, 259
 - getting data, 503
 - loading, 476
- linear solver, 2
- linearity, 61, 62, 337, 351
- linearization, viii, 3, 70, 350, 369
- LINGO format, 25, 32
 - writing, 461
- linking, 266
- Linux, 8
- LMBinPack.m, 556
- LMreadf.m, 555
- Indapi40.lic, 6, 24
- loading
 - models, 117, 476
 - variables, 482, 483, 484, 485
- Loading Core Model, 394
- Loading the Stochastic Structure, 399
- Loading the Time Structure, 397
- locally optimal, 352, 371
- location, 422
- logarithm, 335, 339
- Logarithmic distribution, 416
- logical operators, 350
- Logistic distribution, 416
- Lognormal distribution, 416
- long variable, 269
- looping, 286, 292
- loose inequality operators, 586
- Louveaux, F., 411
- lower bounds
 - adding, 196, 520
 - best, 212
 - getting, 166, 475, 503, 552
 - LINDO files, 587
 - loading, 118, 476, 553
 - MIPs, 71
 - modifying, 206, 529
 - MPS files, 573
 - nonlinear programs, 117, 336
 - objective, 167, 504
 - SLB, 587, 589
 - Visual Basic example, 276
- LS_BASTYPE_ATLO, 18, 19
- LS_BASTYPE_ATUP, 18, 19
- LS_BASTYPE_BAS, 18, 19
- LS_BASTYPE_FNUL, 18, 19
- LS_BASTYPE_SBAS, 18, 19
- LS_CONETYPE_QUAD, 19
- LS_CONETYPE_RQUAD, 19
- LS_CONTYPE_EQ, 19
- LS_CONTYPE_FR, 19
- LS_CONTYPE_GE, 19
- LS_CONTYPE_LE, 19
- LS_DERIV_BACKWARD_DIFFERENCE, 63
- LS_DERIV_CENTER_DIFFERENCE, 63
- LS_DERIV_FORWARD_DIFFERENCE, 63
- LS_DERIV_FREE, 63
- LS_DINFO_MIP_OBJ, 432

LS_DINFO_MIP_SOLOBJVAL_LST_BRANCH, 432
LS_DINFO_MIPBESTBOUND, 432
LS_DINFO_SAMP_KURTOSIS, 112
LS_DINFO_SAMP_MEAN, 112
LS_DINFO_SAMP_SKEWNESS, 112
LS_DINFO_SAMP_STD, 112
LS_DINFO_STOC_ABSOPT_GAP, 109
LS_DINFO_STOC_DINFEAS, 109
LS_DINFO_STOC_EVOBJ, 109
LS_DINFO_STOC_NUM_COLS_DETEQE, 111
LS_DINFO_STOC_NUM_COLS_DETEQI, 111
LS_DINFO_STOC_NUM_NODES, 110
LS_DINFO_STOC_NUM_NODES_STAGE, 110
LS_DINFO_STOC_NUM_ROWS_DETEQE, 111
LS_DINFO_STOC_NUM_ROWS_DETEQI, 111
LS_DINFO_STOC_NUM_SCENARIOS, 110
LS_DINFO_STOC_PINFEAS, 109
LS_DINFO_STOC_RELOPT_GAP, 109
LS_DINFO_STOC_TOTAL_TIME, 110
LS_DPARAM_CALLBACKFREQ, 53, 422
LS_DPARAM_GOP_BNDLIM, 83, 87
LS_DPARAM_GOP_BOXTOL, 82
LS_DPARAM_GOP_DELTATOL, 83
LS_DPARAM_GOP_FLTTOL, 82
LS_DPARAM_GOP_OPTTOL, 82, 84
LS_DPARAM_GOP_WIDTOL, 83, 84
LS_DPARAM_IPM_BASIS_REL_TOL_S, 66
LS_DPARAM_IPM_BASIS_TOL_S, 66
LS_DPARAM_IPM_BASIS_TOL_X, 66
LS_DPARAM_IPM_BI_LU_TOL_REL_PIV, 66
LS_DPARAM_IPM_TOL_DSAFE, 66
LS_DPARAM_IPM_TOL_INFEAS, 65
LS_DPARAM_IPM_TOL_MU_RED, 66
LS_DPARAM_IPM_TOL_PATH, 65
LS_DPARAM_IPM_TOL_PFEAS, 65
LS_DPARAM_MIP_ABSOPTTOL, 73
LS_DPARAM_MIP_ADDCUTOBJTOL, 72
LS_DPARAM_MIP_ADDCUTPER, 72
LS_DPARAM_MIP_ADDCUTPER_TREE, 72
LS_DPARAM_MIP_AOPTTIMLIM, 72
LS_DPARAM_MIP_BIGM_FOR_INTTOL, 67
LS_DPARAM_MIP_CUTOFFOBJ, 76
LS_DPARAM_MIP_CUTOFFVAL, 77
LS_DPARAM_MIP_CUTTIMLIM, 70
LS_DPARAM_MIP_DELTA, 70, 351
LS_DPARAM_MIP_FP_TIMLIM, 69
LS_DPARAM_MIP_FP_WEIGHTH, 68
LS_DPARAM_MIP_HEUMINTIMLIM, 72, 74
LS_DPARAM_MIP_INTTOL, 73
LS_DPARAM_MIP_LBIGM, 70, 351
LS_DPARAM_MIP_LSOLTIMLIM, 80
LS_DPARAM_MIP_MINABSBJSTEP, 80
LS_DPARAM_MIP_PEROPTTOL, 72, 73
LS_DPARAM_MIP_PSEUDOCOST_WEIGHT, 81
LS_DPARAM_MIP_REDCOSTFIX_CUTOFF, 72
LS_DPARAM_MIP_REDCOSTFIX_CUTOFF_T
REE, 81
LS_DPARAM_MIP_RELINTTOL, 73, 77
LS_DPARAM_MIP_RELOPTTOL, 73
LS_DPARAM_MIP_TIMLIM, 79
LS_DPARAM_NLP_FEASTOL, 63
LS_DPARAM_NLP_PSTEP_FINITEDIFF, 60
LS_DPARAM_NLP_REDGTOL, 63
LS_DPARAM_OBJPRINTMUL, 57
LS_DPARAM_SOLVER_CUTOFFVAL, 54, 58
LS_DPARAM_SOLVER_FEASTOL, 44, 55
LS_DPARAM_SOLVER_IUSOL, 56
LS_DPARAM_SOLVER_OPTTOL, 55
LS_DPARAM_STOC_ABSOPTTOL, 98
LS_DPARAM_STOC_RELOPTTOL, 97
LS_DPARAM_STOC_TIME_LIM, 97
LS_FORMATTED_MPS, 26, 457
LS_IINFO_DIST_TYPE, 111
LS_IINFO_ITER, 432
LS_IINFO_MIP_ACTIVENODES, 432
LS_IINFO_MIP_BRANCHCOUNT, 432
LS_IINFO_MIP_LPCOUNT, 432
LS_IINFO_MIP_LTYPE, 432
LS_IINFO_MIP_NEWIPSOL, 432
LS_IINFO_MIP_SOLSTATUS_LAST_BRANCH,
432
LS_IINFO_MIP_STATUS, 432
LS_IINFO_NUM_STOCPAR_AIJ, 110
LS_IINFO_NUM_STOCPAR_INSTR_CONS, 110
LS_IINFO_NUM_STOCPAR_INSTR_OBJS, 110
LS_IINFO_NUM_STOCPAR_LB, 110
LS_IINFO_NUM_STOCPAR_OBJ, 110
LS_IINFO_NUM_STOCPAR_RHS, 110
LS_IINFO_NUM_STOCPAR_UB, 110
LS_IINFO_SAMP_SIZE, 111
LS_IINFO_STOC_BAR_ITER, 109
LS_IINFO_STOC_NLP_ITER, 109
LS_IINFO_STOC_NUM_BENDERS_FCUTS, 111
LS_IINFO_STOC_NUM_BENDERS_OCUTS,
111
LS_IINFO_STOC_NUM_COLS_BEFORE_NODE
, 111
LS_IINFO_STOC_NUM_COLS_CORE, 111
LS_IINFO_STOC_NUM_COLS_DETEQE, 111
LS_IINFO_STOC_NUM_COLS_DETEQI, 111
LS_IINFO_STOC_NUM_COLS_NAC, 111
LS_IINFO_STOC_NUM_COLS_STAGE, 111
LS_IINFO_STOC_NUM_NODE_MODELS, 110
LS_IINFO_STOC_NUM_NODES, 110
LS_IINFO_STOC_NUM_NODES_STAGE, 110
LS_IINFO_STOC_NUM_QCP_CONS_DETEQE,
112
LS_IINFO_STOC_NUM_ROWS_BEFORE_NOD
E, 111

-
- LS_IINFO_STOC_NUM_ROWS_CORE, 111
 - LS_IINFO_STOC_NUM_ROWS_DETEQE, 111
 - LS_IINFO_STOC_NUM_ROWS_DETEQI, 111
 - LS_IINFO_STOC_NUM_ROWS_NAC, 111
 - LS_IINFO_STOC_NUM_ROWS_STAGE, 111
 - LS_IINFO_STOC_NUM_SCENARIOS, 110
 - LS_IINFO_STOC_NUM_STAGES, 110
 - LS_IINFO_STOC_STAGE_BY_NODE, 110
 - LS_IINFO_STOC_STATUS, 110
 - LS_IIS_ADD_FILTER, 20, 436
 - LS_IIS_DEFAULT, 20, 436
 - LS_IIS_DEL_FILTER, 20, 436
 - LS_IIS_DFBS_FILTER, 20, 436
 - LS_IIS_ELS_FILTER, 20, 437
 - LS_IIS_FSC_FILTER, 20, 436
 - LS_IIS_GBS_FILTER, 20, 436
 - LS_IIS_NORM_FREE, 20, 437
 - LS_IIS_NORM_INFINITY, 20, 437
 - LS_IIS_NORM_ONE, 20, 437
 - LS_IMAT_AIJ, 20
 - LS_INFINITY, 18, 118, 166, 196
 - LS_IPARAM_MIP_USECUTOFFOBJ, 77
 - LS_IPARAM_ALLOW_CNTRLBREAK, 53
 - LS_IPARAM_BARRIER_PROB_TO SOLVE, 54
 - LS_IPARAM_BARRIER_SOLVER, 53
 - LS_IPARAM_CHECK_FOR_ERRORS, 53
 - LS_IPARAM_DECOMPOSITION_TYPE, 55
 - LS_IPARAM_GOP_ALGREFORMMD, 86
 - LS_IPARAM_GOP_BBSRCHMD, 86
 - LS_IPARAM_GOP_BRANCH_LIMIT, 88
 - LS_IPARAM_GOP_BRANCHMMD, 85
 - LS_IPARAM_GOP_CORELEVEL, 89
 - LS_IPARAM_GOP_DECOMPTMD, 86
 - LS_IPARAM_GOP_HEU_MODE, 89
 - LS_IPARAM_GOP_LPSOPT, 89
 - LS_IPARAM_GOP_LSOLBRANLIM, 89
 - LS_IPARAM_GOP_MAXWIDMD, 83, 84
 - LS_IPARAM_GOP_OPT_MODE, 87
 - LS_IPARAM_GOP_OPTCHKMD, 84
 - LS_IPARAM_GOP_POSTLEVEL, 85
 - LS_IPARAM_GOP_PRELEVEL, 85
 - LS_IPARAM_GOP_PRINTLEVEL, 86
 - LS_IPARAM_GOP_RELBRNDMD, 87
 - LS_IPARAM_GOP_SUBOUT_MODE, 89
 - LS_IPARAM_GOP_TIMLIM, 83
 - LS_IPARAM_GOP_USE_NLPSOLVE, 89
 - LS_IPARAM_GOP_USEBNDLIM, 87
 - LS_IPARAM_GOP_ANALYZE_LEVEL, 93
 - LS_IPARAM_IIS_INFEAS_NORM, 94
 - LS_IPARAM_IIS_ITER_LIMIT, 94
 - LS_IPARAM_IIS_METHOD, 93
 - LS_IPARAM_IIS_PRINT_LEVEL, 94
 - LS_IPARAM_IIS_REOPT, 94
 - LS_IPARAM_IIS_TIME_LIMIT, 95
 - LS_IPARAM_IIS_USE_EFILTER, 93
 - LS_IPARAM_IIS_USE_GOP, 93
 - LS_IPARAM_IIS_USE_SFILTER, 94
 - LS_IPARAM_INSTRUCT_LOADTYPE, 53
 - LS_IPARAM_IPM_MAX_ITERATIONS, 66
 - LS_IPARAM_IPM_NUM_THREADS, 67
 - LS_IPARAM_IPM_OFF_COL_TRH, 66
 - LS_IPARAM_IUS_ANALYZE_LEVEL, 93
 - LS_IPARAM_IUS_TOPOPT, 94
 - LS_IPARAM_LIC_BARRIER, 92
 - LS_IPARAM_LIC_CONSTRAINTS, 91
 - LS_IPARAM_LIC_DAYSTOEXP, 91
 - LS_IPARAM_LIC_DAYSTOTRIALEXP, 91
 - LS_IPARAM_LIC_EDUCATIONAL, 92
 - LS_IPARAM_LIC_GLOBAL, 92
 - LS_IPARAM_LIC_GOP_INTEGERS, 91
 - LS_IPARAM_LIC_GOP_NONLINEARVARS, 91
 - LS_IPARAM_LIC_INTEGERS, 91
 - LS_IPARAM_LIC_NONLINEAR, 92
 - LS_IPARAM_LIC_NONLINEARVARS, 91
 - LS_IPARAM_LIC_NUMUSERS, 92
 - LS_IPARAM_LIC_PLATFORM, 91
 - LS_IPARAM_LIC_RUNTIME, 92
 - LS_IPARAM_LIC_VARIABLES, 91
 - LS_IPARAM_LP_PRELEVEL, 59
 - LS_IPARAM_LP_PRINTLEVEL, 57, 62
 - LS_IPARAM_LP_SCALE, 56
 - LS_IPARAM_MAXCUTPASS_TREE, 71
 - LS_IPARAM_MIP_AGGCUTLIM_TOP, 80
 - LS_IPARAM_MIP_AGGCUTLIM_TREE, 80
 - LS_IPARAM_MIP_ANODES_SWITCH_DF, 78
 - LS_IPARAM_MIP_BRANCH_LIMIT, 79
 - LS_IPARAM_MIP_BRANCH_PRIO, 71
 - LS_IPARAM_MIP_BRANCHDIR, 73
 - LS_IPARAM_MIP_BRANCHRULE, 75
 - LS_IPARAM_MIP_CUTDEPTH, 70
 - LS_IPARAM_MIP_CUTFREQ, 70
 - LS_IPARAM_MIP_CUTLEVEL_TOP, 69
 - LS_IPARAM_MIP_CUTLEVEL_TREE, 70
 - LS_IPARAM_MIP_DUAL_SOLUTION, 80
 - LS_IPARAM_MIP_FP_ITRLIM, 69
 - LS_IPARAM_MIP_FP_MODE, 68
 - LS_IPARAM_MIP_FP_OPT_METHOD, 69
 - LS_IPARAM_MIP_HEU_MODE, 68
 - LS_IPARAM_MIP_HEULEVEL, 72, 74
 - LS_IPARAM_MIP_ITRLIM, 79
 - LS_IPARAM_MIP_KEEPINMEM, 73
 - LS_IPARAM_MIP_MAKECUT_INACTIVE_CO
UNT, 65, 68
 - LS_IPARAM_MIP_MAXCUTPASS_TOP, 71
 - LS_IPARAM_MIP_MAXNONIMP_CUTPASS,
71
 - LS_IPARAM_MIP_NODESELRULE, 75
 - LS_IPARAM_MIP_PRE_ELIM_FILL, 65, 68
 - LS_IPARAM_MIP_PRELEVEL, 76
 - LS_IPARAM_MIP_PRELEVEL_TREE, 81
-

LS_IPARAM_MIP_PREPRINTLEVEL, 76
LS_IPARAM_MIP_PRINTLEVEL, 76
LS_IPARAM_MIP_PSEUDOCOST_RULE, 81
LS_IPARAM_MIP_REOPT, 77
LS_IPARAM_MIP_SCALING_BOUND, 71
LS_IPARAM_MIP_SOLVERTYPE, 74
LS_IPARAM_MIP_STRONGBRANCHDONUM,
68
LS_IPARAM_MIP_STRONGBRANCHLEVEL,
78
LS_IPARAM_MIP_SWITCHFAC_SIM_IPM, 78
LS_IPARAM_MIP_TOPOPT, 80
LS_IPARAM_MIP_TREEREORDERLEVEL, 78
LS_IPARAM_MIP_USE_CUTS_HEU, 67
LS_IPARAM_MIP_USE_ENUM_HEU, 81
LS_IPARAM_MIP_USE_INT_ZERO_TOL, 67
LS_IPARAM_MPS_OBJ_WRITESTYLE, 54
LS_IPARAM_NLP_AUTODERIV, 61, 349
LS_IPARAM_NLP_AUTOHESS, 64
LS_IPARAM_NLP_CONVEX, 64
LS_IPARAM_NLP_CONVEXRELAX, 63
LS_IPARAM_NLP_CR_ALG_REFORM, 63
LS_IPARAM_NLP_DERIV_TYPE, 63
LS_IPARAM_NLP_FEASCHK, 62
LS_IPARAM_NLP_ITRLMT, 63, 65
LS_IPARAM_NLP_LINEARITY, 62, 351
LS_IPARAM_NLP_LINEARZ, 61, 351
LS_IPARAM_NLP_MAXLOCALSEARCH, 64,
375
LS_IPARAM_NLP_PRELEVEL, 61
LS_IPARAM_NLP_QUADCHK, 64
LS_IPARAM_NLP_SOLVE_AS_LP, 59
LS_IPARAM_NLP_SOLVER, 59
LS_IPARAM_NLP_STALL_ITRLMT, 64
LS_IPARAM_NLP_STARTPOINT, 63
LS_IPARAM_NLP_SUBSOLVER, 60
LS_IPARAM_NLP_USE_CRASH, 60
LS_IPARAM_NLP_USE_LINDO_CRASH, 64
LS_IPARAM_NLP_USE_SELCONVEVAL, 60
LS_IPARAM_NLP_USE_SLP, 60
LS_IPARAM_NLP_USE_STEEPEDGE, 60
LS_IPARAM_OBJSENSE, 57
LS_IPARAM_PROB_TO_SOLVE, 58
LS_IPARAM_SOL_REPORT_STYLE, 53
LS_IPARAM_SOLVER_IPMSOL, 58, 140
LS_IPARAM_SOLVER_PRE_ELIM_FILL, 59
LS_IPARAM_SOLVER_RESTART, 57
LS_IPARAM_SOLVER_TIMLMT, 58
LS_IPARAM_SOLVER_USE_CONCURRENT_O
PT, 67
LS_IPARAM_SOLVER_USECUTOFFVAL, 58
LS_IPARAM_SPLEX_DPRICING, 57
LS_IPARAM_SPLEX_DUAL_PHASE, 59
LS_IPARAM_SPLEX_ITRLMT, 56
LS_IPARAM_SPLEX_PPRICING, 57
LS_IPARAM_SPLEX_REFACFRQ, 53
LS_IPARAM_STOC_BUCKET_SIZE, 97
LS_IPARAM_STOC_CALC_EVPI, 97
LS_IPARAM_STOC_DEBUG_LEVEL, 97
LS_IPARAM_STOC_DETEQ_TYPE, 97
LS_IPARAM_STOC_ITER_LIM, 96
LS_IPARAM_STOC_MAX_NUMSCENS, 97
LS_IPARAM_STOC_METHOD, 96
LS_IPARAM_STOC_NODELP_PRELEVEL, 97
LS_IPARAM_STOC_PRINT_LEVEL, 96
LS_IPARAM_STOC_REOPT, 96
LS_IPARAM_STOC_RG_SEED, 96
LS_IPARAM_STOC_SAMP_CONT_ONLY, 97
LS_IPARAM_STOC_SAMP_SIZE_NODE, 96
LS_IPARAM_STOC_SAMP_SIZE_SPAR, 96
LS_IPARAM_STOC_SHARE_BEGSTAGE, 97
LS_IPARAM_STOC_TOPOPT, 96
LS_IPARAM_TIMLMT, 58
LS_IPARAM_VER_NUMBER, 58
LS_IROW_OBJ, 20
LS_IROW_VFX, 20
LS_IROW_VLB, 20
LS_IROW_VUB, 20
LS_JCOL_INST, 20
LS_JCOL_RHS, 20
LS_JCOL_RLB, 20
LS_JCOL_RUB, 20
LS_LINK_BLOCKS_BOTH, 55
LS_LINK_BLOCKS_COLS, 55
LS_LINK_BLOCKS_FREE, 55
LS_LINK_BLOCKS_NONE, 55
LS_LINK_BLOCKS_ROWS, 55
LS_MAX, 18, 57, 117, 165, 276
LS_MAX_ERROR_MESSAGE_LENGTH, 219
LS_METHOD_BARRIER, 19, 60, 139
LS_METHOD_DSIMPLEX, 19, 60, 139
LS_METHOD_FREE, 19, 139
LS_METHOD_NLP, 19, 139
LS_METHOD_PSIMPLEX, 19, 60, 139, 265
LS_METHOD_STOC_ALD, 20
LS_METHOD_STOC_DETEQ, 20
LS_METHOD_STOC_FREE, 20
LS_METHOD_STOC_NBD, 20
LS_MIN, 18, 57, 115, 117, 165, 264, 285, 291, 474
LS_MIP_SET_CARD, 20
LS_MIP_SET_SOS1, 19
LS_MIP_SET_SOS2, 19
LS_MIP_SET_SOS3, 19
LS_NMETHOD_CONOPT, 59
LS_NMETHOD_FREE, 59
LS_NMETHOD_MSW_GRG, 59
LS_PROB_SOLVE_DUAL, 54
LS_PROB_SOLVE_FREE, 54
LS_PROB_SOLVE_PRIMAL, 54
LS_SOLUTION_MIP, 19

-
- LS_SOLUTION_MIP_OLD, 19
 - LS_SOLUTION_OPT, 19
 - LS_SOLUTION_OPT_IPM, 19
 - LS_SOLUTION_OPT_OLD, 19
 - LS_STATUS_CUTOFF, 18
 - LS_STATUS_INFORUNB, 18
 - LS_STATUS_LOADED, 18
 - LS_STATUS_LOCAL_INFEASIBLE, 18
 - LS_STATUS_LOCAL_OPTIMAL, 18
 - LS_STATUS_NEAR_OPTIMAL, 18
 - LS_STATUS_NUMERICAL_ERROR, 18
 - LS_STATUS_UNKNOWN, 18
 - LS_STATUS_UNLOADED, 18
 - LS_UNFORMATTED_MPS, 26
 - LS_VARTYPE_BIN, 19
 - LS_VARTYPE_CONT, 19
 - LS_VARTYPE_INT, 19
 - L SaddCones(), 193, 517
 - L SaddConstraints(), 194, 195, 518, 520
 - L SaddContinuousIndep (), 131
 - L SaddDiscreteBlocks (), 132
 - L SaddDiscreteIndep (), 130
 - L SaddNLPaj(), 198
 - L SaddNLPobj(), 199
 - L SaddQCterms(), 197
 - L SaddScenario (), 133
 - L SaddSETS(), 195, 519, 521, 522, 523
 - L SaddVariables(), 194, 195, 520
 - L SbuildStringData(), 126
 - L ScaleConFunc(), 223, 543
 - L ScalConGrad(), 224, 544
 - L ScalObjFunc(), 225, 543
 - L ScalObjGrad(), 226, 545
 - L ScreateEnv(), 21, 263, 269, 330, 453, 567
 - quadratic programming, 297, 316
 - L ScreateModel(), 21, 22, 263, 269, 273, 330, 453
 - quadratic programming, 297, 316
 - L ScreateRG (), 241
 - L SdeleteAj(), 202
 - L SdeleteCones(), 199, 523
 - L SdeleteConstraints(), 200, 524
 - L SdeleteEnv(), 21, 22, 266, 277, 297, 316, 454
 - nonlinear programming, 330
 - L SdeleteModel(), 22, 23, 454
 - L SdeleteNLPobj(), 201
 - L SdeleteQCterms(), 200, 524
 - L SdeleteSemiContVars(), 202, 525
 - L SdeleteSETS(), 203, 525, 526, 527
 - L SdeleteString(), 127
 - L SdeleteStringData(), 126
 - L SdeleteVariables(), 203, 526
 - L SdisposeRG (), 243
 - L SdoBTRAN(), 221, 541
 - L SdoFTRAN(), 222, 542
 - L Senv()
 - creating, 21, 453
 - deleting, 22, 454
 - error messages, 219
 - getting, 43, 44, 466, 467
 - setting, 47, 48, 49, 469, 470
 - LSERR_ARRAY_OUT_OF_BOUNDS, 571
 - LSERR_BAD_CONSTRAINT_TYPE, 565
 - LSERR_BAD_DECOMPOSITION_TYPE, 565
 - LSERR_BAD_DISTRIBUTION_TYPE, 571
 - LSERR_BAD_LICENSE_FILE, 565
 - LSERR_BAD_MODEL, 565
 - LSERR_BAD_MPI_FILE, 565
 - LSERR_BAD_MPS_FILE, 565
 - LSERR_BAD_OBJECTIVE_SENSE, 565
 - LSERR_BAD_SMPI_CORE_FILE, 569
 - LSERR_BAD_SMPI_STOC_FILE, 569
 - LSERR_BAD_SMPS_CORE_FILE, 569
 - LSERR_BAD_SMPS_STOC_FILE, 569
 - LSERR_BAD_SMPS_TIME_FILE, 569
 - LSERR_BAD_SOLVER_TYPE, 565
 - LSERR_BAD_VARIABLE_TYPE, 565
 - LSERR_BASIS_BOUND_MISMATCH, 565
 - LSERR_BASIS_COL_STATUS, 565
 - LSERR_BASIS_INVALID, 565
 - LSERR_BASIS_ROW_STATUS, 565
 - LSERR_BLOCK_OF_BLOCK, 566
 - LSERR_BOUND_OUT_OF_RANGE, 566
 - LSERR_CANNOT_OPEN_CORE_FILE, 569
 - LSERR_CANNOT_OPEN_FILE, 566
 - LSERR_CANNOT_OPEN_STOC_FILE, 569
 - LSERR_CANNOT_OPEN_TIME_FILE, 569
 - LSERR_CHECKSUM, 566
 - LSERR_COL_BEGIN_INDEX, 566
 - LSERR_COL_INDEX_OUT_OF_RANGE, 566
 - LSERR_COL_NONZCOUNT, 566
 - LSERR_CORE_BAD_NUMSTAGES, 570
 - LSERR_CORE_BAD_STAGE_INDEX, 570
 - LSERR_CORE_BAD_STRUCTURE, 570
 - LSERR_CORE_INVALID_SPAR_INDEX, 569
 - LSERR_CORE_NOT_IN_TEMPORAL_ORDER, 571
 - LSERR_CORE_SPAR_COUNT_MISMATCH, 569
 - LSERR_CORE_SPAR_NOT_FOUND, 569
 - LSERR_CORE_SPAR_VALUE_NOT_FOUND, 570
 - LSERR_CORE_TIME_MISMATCH, 570
 - LSERR_DATA_TERM_EXIST, 568
 - LSERR_DIST_BAD_CORRELATION_TYPE, 572
 - LSERR_DIST_INVALID_NUMPARAM, 571
 - LSERR_DIST_INVALID_PARAMS, 571
 - LSERR_DIST_INVALID_PROBABILITY, 571
 - LSERR_DIST_INVALID_SD, 571
 - LSERR_DIST_INVALID_X, 571
-

LSERR_DIST_NO_DERIVATIVE, 571
LSERR_DIST_NO_PDF_LIMIT, 571
LSERR_DIST_ROOTER_ITERLIM, 571
LSERR_DIST_SCALE_OUT_OF_RANGE, 571
LSERR_DIST_SHAPE_OUT_OF_RANGE, 571
LSERR_DIST_TRUNCATED, 571
LSERR_EMPTY_COL_STAGE, 572
LSERR_EMPTY_ROW_STAGE, 572
LSERR_ERRMSG_FILE_NOT_FOUND, 566
LSERR_ERROR_IN_INPUT, 115, 118, 123, 474,
481, 566
LSERR_GOP_BRANCH_LIMIT, 566
LSERR_GOP_FUNC_NOT_SUPPORTED, 566
LSERR_ILLEGAL_NULL_POINTER, 566
LSERR_INDEX_DUPLICATE, 566
LSERR_INDEX_OUT_OF_RANGE, 118, 123,
481, 566
LSERR_INFO_NOT_AVAILABLE, 566
LSERR_INFO_UNAVAILABLE, 570
LSERR_INST_INVALID_BOUND, 568
LSERR_INST_MISS_ELEMENTS, 568
LSERR_INST_SYNTAX_ERROR, 568
LSERR_INST_TOO_SHORT, 568
LSERR_INSTRUCT_NOT_LOADED, 566
LSERR_INTERNAL_ERROR, 566
LSERR_INVALID_ERRORCODE, 567
LSERR_ITER_LIMIT, 567
LSERR_LAST_ERROR, 567, 568
LSERR_MIP_BRANCH_LIMIT, 567
LSERR_MISSING_TOKEN_NAME, 570
LSERR_MISSING_TOKEN_ROOT, 570
LSERR_MODEL_ALREADY_LOADED, 567
LSERR_MODEL_NOT_LINEAR, 567
LSERR_MODEL_NOT_LOADED, 567
LSERR_NO_ERROR, 567
LSERR_NO_LICENSE_FILE, 567
LSERR_NO_METHOD_LICENSE, 567
LSERR_NO_VALID_LICENSE, 567
LSERR_NOT_CONVEX, 567
LSERR_NOT_SORTED_ORDER, 568
LSERR_NOT_SUPPORTED, 567
LSERR_NUMERIC_INSTABILITY, 567
LSERR_OLD_LICENSE, 567
LSERR_OUT_OF_MEMORY, 567
LSERR_PARAMETER_OUT_OF_RANGE, 568
LSERR_RG_ALREADY_SET, 572
LSERR_RG_NOT_SET, 571
LSERR_RG_SEED_NOT_SET, 572
LSERR_ROW_INDEX_OUT_OF_RANGE, 568
LSERR_SCEN_INDEX_OUT_OF_SEQUENCE,
569
LSERR_STEP_TOO_SMALL, 568
LSERR_STOC_BAD_ALGORITHM, 570
LSERR_STOC_BAD_PRECISION, 570
LSERR_STOC_BLOCK_SAMPLING_NOT_SUP
PORTED, 572
LSERR_STOC_EVENTS_NOT_LOADED, 572
LSERR_STOC_INVALID_CDF, 571
LSERR_STOC_INVALID_SAMPLE_SIZE, 571
LSERR_STOC_INVALID_SCENARIO_CDF, 569
LSERR_STOC_MISSING_BNDNAME, 570
LSERR_STOC_MISSING_OBJNAME, 570
LSERR_STOC_MISSING_PARAM_TOKEN, 571
LSERR_STOC_MISSING_RHSNAME, 570
LSERR_STOC_MISSING_RNGNAME, 570
LSERR_STOC_MODEL_ALREADY_PARSED,
569
LSERR_STOC_MODEL_NOT_LOADED, 569
LSERR_STOC_NODE_INFEASIBLE, 570
LSERR_STOC_NODE_UNBOUNDED, 570
LSERR_STOC_NOT_DISCRETE, 571
LSERR_STOC_NULL_EVENT_TREE, 570
LSERR_STOC_OUT_OF_SAMPLE_POINTS, 572
LSERR_STOC_SAMPLE_ALREADY_GENERA
TED, 572
LSERR_STOC_SAMPLE_ALREADY_LOADED,
572
LSERR_STOC_SAMPLE_NOT_GENERATED,
572
LSERR_STOC_SAMPLE_SIZE_TOO_SMALL,
572
LSERR_STOC_SCENARIO_LIMIT, 571
LSERR_STOC_SCENARIO_SAMPLING_NOT_S
UPPORTED, 572
LSERR_STOC_SPAR_NOT_FOUND, 569
LSERR_STOC_TOO_MANY_SCENARIOS, 570
LSERR_STOC_TREE_ALREADY_INIT, 572
LSERR_TIME_BAD_NUMSTAGES, 570
LSERR_TIME_BAD_TEMPORAL_ORDER, 570
LSERR_TIME_LIMIT, 568
LSERR_TIME_NUMSTAGES_NOT_SET, 572
LSERR_TIME_SPAR_COUNT_MISMATCH, 569
LSERR_TIME_SPAR_NOT_EXPECTED, 569
LSERR_TIME_SPAR_NOT_FOUND, 569
LSERR_TOO_SMALL_LICENSE, 568
LSERR_TOTAL_NONZCOUNT, 568
LSERR_TRUNCATED_NAME_DATA, 568
LSERR_UNABLE_TO_SET_PARAM, 568
LSERR_USER_FUNCTION_NOT_FOUND, 568
LSERR_USER_INTERRUPT, 568
LSERR_VARIABLE_NOT_FOUND, 568
LSfindBlockStructure(), 210, 446, 534
LSfindIIS(), 211, 435, 535
LSfindIUS(), 211, 535
LSfreeGOPSolutionMemory(), 238
LSfreeHashMemory(), 238
LSfreeMIPSolutionMemory(), 239
LSfreeSolutionMemory(), 239
LSfreeSolverMemory(), 240

-
- LSgetBasis(), 144, 491
 - LSgetBestBounds(), 212, 536
 - LSgetBlockStructure(), 213, 537
 - LSgetBoundRanges(), 214, 433, 435, 538
 - LSgetCallback, 279
 - LSgetCallbackInfo(), 227, 422, 428, 430, 431, 546
 - LSgetConeDatai(), 160, 498
 - LSgetConeIndex(), 161, 499
 - LSgetConeNamei(), 161, 499
 - LSgetConstraintDatai(), 162, 500
 - LSgetConstraintIndex(), 163, 501
 - LSgetConstraintNamei(), 163, 501
 - LSgetConstraintRanges(), 215, 433, 434, 538
 - LSgetDeteqModel (), 184
 - LSgetDiscreteBlockOutcomes, 188
 - LSgetDiscreteBlocks, 187
 - LSgetDiscreteIndep, 189
 - LSgetDistrRV (), 242
 - LSgetDoubleRV (), 241
 - LSgetDualMIPSolution(), 279
 - LSgetDualSolution(), 145, 492
 - nonlinear programming, 330
 - quadratic programming, 297, 316
 - LSgetEnvDouParameter(), 44, 466
 - LSgetEnvIntParameter(), 44, 467
 - LSgetEnvParameter(), 43, 466
 - LSgetErrorMessage(), 219, 273, 464
 - LSgetErrorRowIndex, 220
 - LSgetErrorRowIndex(), 464
 - LSgetFileError(), 220, 465
 - LSgetHistogram, 191
 - LSgetIIS(), 216, 539
 - LSgetInfo(), 146, 279, 492
 - nonlinear programming, 330
 - quadratic programming, 297, 316
 - LSgetInitSeed (), 242
 - LSgetInt32RV (), 242
 - LSgetIUS(), 217, 540
 - LSgetLPConstraintDatai(), 164, 502
 - LSgetLPData(), 165, 503
 - LSgetLPVariableDataj(), 167, 504
 - LSgetMIPBasis(), 147, 493
 - LSgetMIPCallbackInfo(), 230, 279, 431, 547
 - LSgetMIPDualSolution(), 148, 493
 - LSgetMIPPrimalSolution(), 148, 158, 494
 - LSgetMIPReducedCosts(), 149, 279, 494
 - LSgetMIPSlacks(), 150, 279, 495
 - LSgetMIPSolution(), 279
 - LSgetModelDouParameter(), 46, 467, 468
 - LSgetModelIntParameter(), 46, 351, 468
 - LSgetModelParameter(), 45, 51, 467
 - LSgetNameData(), 168, 505
 - LSgetNextBestMIPSoln (), 158
 - LSgetNLPCConstraintDatai(), 169
 - LSgetNLPCConstraintDatai(), 506
 - LSgetNLPData(), 170, 507
 - LSgetNLPObjectiveData(), 171
 - LSgetNLPObjectiveData(), 508
 - LSgetNLPVariableDataj(), 172
 - LSgetNLPVariableDataj(), 509
 - LSgetNodeDualSolution, 156, 157, 158
 - LSgetNodeDualSolution (), 156
 - LSgetNodeListByScenario (), 185
 - LSgetNodePrimalSolution, 154
 - LSgetNodePrimalSolution (), 154
 - LSgetNodeReducedCost (), 38
 - LSgetNodeSlacks, 156
 - LSgetNodeSlacks (), 156
 - LSgetObjective(), 265, 277
 - LSgetObjectiveRanges(), 218, 433, 434, 540
 - LSgetPrimalSolution(), 150, 495
 - C++ example, 265
 - MATLAB, 495
 - nonlinear programming, 330
 - quadratic programming, 297, 316
 - Visual Basic example, 277
 - LSgetProbabilityByNode (), 184
 - LSgetProbabilityByScenario (), 183
 - LSgetQCData(), 173, 510
 - LSgetQCData(), 174, 511
 - LSgetReducedCosts(), 151, 496
 - LSgetReducedCostsCone(), 151, 496
 - LSgetSampleSizes, 190
 - LSgetScenarioDualSolution (), 157
 - LSgetScenarioIndex (), 183
 - LSgetScenarioModel, 192
 - LSgetScenarioName, 182
 - LSgetScenarioName (), 182
 - LSgetScenarioObjective, 154
 - LSgetScenarioObjective (), 154
 - LSgetScenarioPrimalSolution, 155
 - LSgetScenarioPrimalSolution (), 155
 - LSgetScenarioReducedCost (), 155
 - LSgetScenarioSlacks (), 158
 - LSgetSemiContData(), 175, 512
 - LSgetSETSDData(), 176, 513
 - LSgetSETSDData(), 177, 514
 - LSgetSlacks(), 152, 497
 - LSgetSolution(), 153, 497
 - LSgetStageIndex (), 181
 - LSgetStageName (), 180
 - LSgetStocParData (), 186
 - LSgetStocParIndex (), 181
 - LSgetStocParName (), 182
 - LSgetStocParOutcomes, 186
 - LSgetStocParOutcomes (), 185
 - LSgetStocParSample, 254
 - LSgetStringValue(), 127
 - LSgetVariableIndex(), 178, 514
 - LSgetVariableNamej(), 179, 515
-

-
- L\$getVarStages, 191
 - L\$getVarStartPoint(), 179, 515
 - L\$getVarType(), 180, 516
 - L\$getVersionInfo(), 23, 455
 - L\$getxxxxyyParameter(), 53
 - L\$loadBasis(), 135, 486
 - L\$loadBlockStructure(), 137, 139, 488
 - L\$loadConeData (), 114
 - L\$loadConeData(), 316, 473
 - L\$loadConstraintStages (), 129
 - L\$loadInstruct(), 115, 349, 369, 474
 - L\$loadLicenseString(), 23, 455
 - L\$loadLPData(), 117, 279
 - C++ example, 287
 - integer programming, 279
 - MATLAB, 476
 - nonlinear programming, 330
 - quadratic programming, 297, 316
 - Visual Basic example, 276, 292, 293
 - L\$loadNameData(), 33, 119, 477
 - L\$loadNLPData(), 120, 330, 478
 - L\$loadQCData(), 121, 297, 299, 479
 - L\$loadSampleSizes (), 128
 - L\$loadSemiContData(), 122, 480
 - L\$loadSETSData(), 123, 481
 - L\$loadStocParData (), 130
 - L\$loadStocParNames (), 134
 - L\$loadString(), 125
 - L\$loadStringData(), 125
 - L\$loadVariableStages (), 129
 - L\$loadVarPriorities(), 136, 486
 - L\$loadVarStartPoint(), 136, 487
 - L\$loadVarType(), 118, 124, 279, 287, 293, 482, 483, 484, 485
 - integer programming, 279
 - quadratic programming, 297, 316
 - L\$model
 - creating, 22, 453
 - deleting, 23, 454
 - getting, 467, 468
 - loading, 117, 119, 124, 476, 477, 482, 483, 484, 485
 - setting, 470, 471
 - L\$modifyAj(), 204, 527
 - L\$modifyCone(), 204, 528
 - L\$modifyConstraintType(), 205, 528
 - L\$modifyLowerBounds(), 206, 529
 - L\$modifyObjConstant(), 205, 206, 530
 - L\$modifyObjective(), 207, 530
 - L\$modifyRHS(), 207, 531
 - L\$modifySemiContVars(), 208, 531
 - L\$modifySET(), 208, 532
 - L\$modifyUpperBounds(), 209, 533
 - L\$modifyVariableType(), 209, 533
 - L\$optimize(), 139
 - C++ example, 265
 - MATLAB, 490
 - nonlinear programming, 330
 - quadratic programming, 297, 316
 - Visual Basic example, 276
 - L\$readBasis(), 28
 - L\$readEnvParameter(), 51, 472
 - L\$readLINDOFile(), 25, 456, 585, 586
 - L\$readModelParameter(), 51, 472, 473
 - L\$readMPIFile(), 27, 457
 - L\$readMPSFile(), 26, 457, 573
 - L\$readSMPIFile(), 35
 - L\$readSMPSFile (), 34
 - L\$readVarPriorities(), 138, 489
 - L\$readVarStartPoint(), 139, 489
 - L\$sampCreate (), 245
 - L\$sampDelete (), 245
 - L\$sampEvalDistr (), 248
 - L\$sampGenerate (), 249
 - L\$sampGetCIPoints (), 250
 - L\$sampGetCIPointsPtr (), 251
 - L\$sampGetCorrelationMatrix (), 251
 - L\$sampGetDiscretePdfTable (), 246
 - L\$sampGetDistrParam (), 247
 - L\$sampGetInfo (), 253
 - L\$sampGetPoints (), 249
 - L\$sampGetPointsPtr (), 250
 - L\$sampInduceCorrelation (), 252
 - L\$sampLoadDiscretePdfTable (), 246
 - L\$sampSetDistrParam (), 247
 - L\$sampSetRG (), 248
 - L\$sampSetUserDistr (), 247
 - L\$ssetCallback(), 227, 231, 279, 421, 422, 428
 - MATLAB, 546, 547
 - Visual Basic example, 429
 - L\$ssetDistrParamRG (), 244
 - L\$ssetDistrRG (), 244
 - L\$ssetEnvDouParameter(), 48, 422, 469
 - L\$ssetEnvIntParameter(), 49, 470
 - L\$ssetEnvLogFunc(), 232
 - L\$ssetEnvParameter(), 47, 469
 - L\$ssetFuncalc(), 233, 330, 334, 355, 548
 - L\$ssetGradcalc(), 234, 330, 336, 549
 - L\$ssetMIPCallback(), 235, 279, 430, 431
 - MATLAB, 547, 550
 - L\$ssetModelDouParameter(), 50, 351, 471
 - L\$ssetModelIntParameter(), 50, 349, 351, 471
 - L\$ssetModelLogFunc(), 236, 551
 - L\$ssetModelParameter(), 49, 470
 - L\$ssetNumStages (), 128
 - L\$ssetRGSeed (), 243
 - L\$ssetUsercalc(), 237, 347, 551
 - L\$setxxxxyyParameter(), 53
 - L\$solveGOP(), 139, 141, 490
 - L\$solveMIP(), 139, 142, 279, 491
-

C++ example, 287
 nonlinear programming, 370
 quadratic programming, 297, 316
 Visual Basic example, 293
 LSsolveSP (), 143
 LStocInfo
 LS_IINFO_STOC_SIM_ITER, 109
 LWriteBasis(), 28
 LWriteDeteqLINDOFile (), 37
 LWriteDeteqMPSFile (), 37
 LWriteDualLINDOFile(), 29, 458
 LWriteDualMPSFile(), 30, 459
 LWriteEnvParameter(), 52
 LWriteIIS(), 31, 460
 LWriteIUS(), 31, 460
 LWriteLINDOFile(), 32, 461, 586
 LWriteLINGOFile(), 32, 461
 LWriteModelParameter(), 52
 LWriteMPIFile(), 27
 LWriteMPSFile(), 33, 462, 573
 LWriteNodeSolutionFile (), 39
 LWriteScenarioLINDOFile (), 41
 LWriteScenarioMPIFile (), 40
 LWriteScenarioMPSFile (), 40
 LWriteScenarioSolutionFile (), 39
 LWriteSMPIFile(), 36
 LWriteSMPSFile (), 36
 LWriteSolution(), 34, 463
 LSXgetLPData(), 552
 LSXloadLPData(), 553
 lump sum, 341

M

Macintosh, 8
 macros, 262
 _LINDO_DLL_, 268
 APIERRORSETUP, 263
 LS_DINFO_POBJ, 265
 makefile.win, 267, 428
 Markowitz model, 300
 mathematical guarantee, 351
 MATLAB, viii, 449
 matrix, 10, 118, 166, 265, 286, 292, 476, 503
 block structured, 137, 445, 488
 covariance, 300
 nonlinear, 121
 quadratic, 121, 174, 197
 sparse, 329
 maximization, 57, 117, 165, 350, 476, 503, 552, 553
 memory, 230, 239, 240, 423, 567
 memory management routines, 238
 MEX-file, 449
 Microsoft Foundation Class, 280
 minimization, 57, 117, 165, 350, 476, 503, 552, 553
 minus, 586
 mixed-integer programs, 148, 158
 branch-and-bound, 142, 491
 callback functions, 235, 279, 550
 cut level, 69, 70
 data loading, 124
 example, 279
 parameters, 67
 query routines, 287, 294
 solution, 494
 mixed-integer solver, 2
 mod function(), 340, 341
 model
 analyzing, 433
 block structured, 210, 213, 444
 continuous, 139, 152, 490
 convex, 350, 371
 creating, 22, 259, 273, 453
 data, 21
 deleting, 22, 23, 454
 dimensions, 42, 275
 dual, 29, 30, 459
 I/O routines, 25
 loading routines, 114, 117, 473
 modification routines, 193, 517
 monitoring, 421, 431
 nonlinear, 329
 primal, 29
 query routines, 160, 498
 reading, 25
 smooth, 350
 writing, 25
 model and solution analysis routines, 534
 modification routines, 193, 517
 modifying variable types, 533
 modules, 429
 modulo, 286, 292
 Monte Carlo Sampling, 406
 MPI format, 27, 375, 457, 565, 591, 593, 601
 MPS format, 25, 573
 debugging, 435–44
 error messages, 565
 extended, 296
 LMreadf.m, 555
 names, 169
 reading, 26, 457
 SOCP, 314
 writing, 30, 33, 458, 462, 463
 MS Windows, 8
 multinomial distribution, 344
 multiple choice, 577
 multiplication, 339
 Multistage Decision Making Under Uncertainty, 389

multistart solver, vii, 3, 8, 59, 64, 227, 350, 352,
353, 371, 423
mxLINDO, 449
routines, 452

N

names

column, 119, 477
constraints, 119, 168, 501, 585
data, 119, 168, 477, 505
getting, 168, 178, 179, 514, 515
hashing, 238
LINDO files, 585
loading, 119, 125
MATLAB, 514
MPS files, 573
row, 119, 477
natural logarithm, 339
necessary set, 436, 437
negation, 339, 349
negative semi-definite, 295
negative variables, 573, 587
Negativebinomial distribution, 416
New Project command, 287
newsvendor problem, 392, 394, 400
nmake, 266, 267, 428
node selection rule, 75, 86
non-convex models, 350, 354
nonlinear programs, vii, 62, 329
 constraint data, 169, 506, 508, 509
 getting data, 170, 507
 iterations, 227
 loading data, 120, 478
 objective data, 171
 optimization, 139
 parameters, 59, 91, 92
 variable data, 172
nonlinear solver, 3, 295
nonoptimal solutions, 588, 589
non-smooth models, viii, 350, 354
nonzero coefficients
 adding, 194, 502, 518
 C++ example, 286
 coefficient matrix, 166, 286, 292
 columns, 196, 285, 286, 291, 292, 520
 getting, 164, 503
 loading, 118, 476
 number of, 164, 167, 265, 476, 502, 504
 sparse format, 114, 473
 storing, 276
 variables, 167
 vectors, 221, 541, 542
 Visual Basic example, 292
norm minimization, 316

Normal cdf, 340
normal density, 343
Normal distribution, 416
not equal to, 339
notation
 Hungarian, 17, 452
 postfix, 337, 368, 592
 Reverse Polish, 337, 592
NP-hard, 354
numeric error, 87, 567

O

object oriented, 280
objective
 adding, 196, 520
 bounds, 167, 432, 504
 C++ example, 264, 285
 constant value, 115, 117, 165, 205, 206, 474,
 476, 503, 530, 552, 553
 cuts, 227
 direction, 264, 285, 291
 displaying, 265
 dual value, 227, 423
 function, 57, 291, 585, 586
 getting, 165, 167, 503, 504, 552
 integrality, 69
 length, 369
 loading, 117, 476, 553
 modifying, 207, 530
 name, 168, 505
 nonlinear data, 171
 parameters, 72
 primal value, 227
 printing, 57
 ranges, 218, 433, 540
 row, 285
 sense, 565
 Visual Basic example, 291
operators, 337, 586
 postfix notation, 339
optimal solution, 259, 287, 295, 494
optimality tolerance, 72, 73, 82
optimization, 139, 259, 421, 490
optimization method, LP, 80, 94
optimization routines, 139, 490
options, supported, 23
OR function, 343
order of precedence, 586
Ox statistical functions, 557
oxLINDO, 557

P

parameters, 42, 53, 99, 422, 568

getting, 45, 466, 467, 468
 setting, 48, 466
 parentheses, 338, 352, 586
 Pareto distribution, 416
 partial derivatives
 calculating, 224, 226, 336
 getting, 170, 171, 507
 setting, 234, 549
 partial pricing, 57
 passing data, 276
 password. See license key
 Pearson correlation, 415
 percent function, 339
 PI, 339
 piecewise linear, 578
 plant location, 69, 227
 plus, 586
 Pluto Dogs, 280
 Poisson, 341
 Poisson distribution, 416
 portfolio selection, 300, 560
 positive definite, 295
 positive semi-definite, 295, 584
 postfix notation, 337, 368, 592
 post-solving, 85
 power function, 339
 precedence order, 337, 352, 586
 prefixes, 17
 preprocessing, 59, 61, 73, 76, 81, 85
 present value, 340, 343
 pricing, 57
 primal
 infeasibility, 227, 423, 546
 model, 29, 30
 objective value, 227
 simplex, 57, 60, 69, 77, 80, 94, 139, 265
 solution, 54, 148, 150, 543
 values, 151, 495
 print level, 62, 76
 printing objective value, 57
 priorities, 136, 138, 486, 489
 probability, 340
 probing, 61, 76, 81
 product form inverse, 2
 product mix, 269
 progress of solver, 421
 protocol cdecl, 422
 prototypes, 262
 PSL, 340
 PUSH instruction, 343, 346
 put option, 413

Q

QMATRIX section, 296, 314

QSECTION, 296
 quadratic constraint, 580
 quadratic objective, 579
 quadratic program, 200, 326, 479, 510, 511, 524, 573
 constraints, 295
 data, 173, 174
 examples, 295
 loading, 121, 125
 MATLAB, 554
 multistart, 353
 quadratic programs as SOCP, 326
 quadratic recognition, vii, 64
 quadruplet entry for QC models, 297
 QUADS, 296
 query routines, 144
 callback functions, 227, 546
 errors, 566
 MIP models, 287, 294
 mxLINDO, 498
 solver status, 422

R

radians, 340
 random, 241
 random number, 342
 ranges
 analysis, 215, 218, 433, 538
 bounds, 214
 names, 168, 505
 vectors, 119, 477
rank correlation, 415
 reading
 LINDO format, 456
 MATLAB, 555
 models, 25
 MPS format, 457
 real bounds, 78
 real numbers, 115, 474
 Recourse Models, 391
 reduced costs, 72, 81, 151, 496
 reduced gradient solver, 59
 reduction, 63
 cuts, 227
 dual, 61, 76, 81
 of coefficients, 61, 69, 76, 81
 refactorization, 53
 reformulation, algebraic, 86
 relative optimality tolerance, 72, 73
 retrieving parameters, 42, 466
 Reverse Polish notation, 337, 592
 right-hand side
 adding, 194, 518
 arguments, 452

constraints, 264, 531, 586
 getting, 127, 164, 165, 502, 503
 increase/decrease, 215
 loading, 118, 476
 modifying, 207
 names, 168, 505
 values, 222
 vector, 119, 125, 477
 Visual Basic example, 275
 rotated quadratic cone, 581
 rounded solutions, 87, 588, 589
 routines
 auxiliary, 552
 callback management, 227, 546
 creation, 21
 deletion, 21, 200, 203, 524, 526
 errors, 566
 memory management, 238
 MIP models, 287, 294
 model loading, 114, 473
 model modification, 193, 517
 mxLINDO, 452
 optimization, 139, 490
 query, 144, 160, 498
 random number generation, 241
 sampling routines, 245
 solver status, 422
 row
 format, 194, 585
 index vector, 11, 12
 indices, 166, 176, 196, 276, 504, 513, 520
 names, 119, 477
 nonlinear, 120, 170, 478, 507
 objective, 285
 separable, 335
 runlindo, 7
 running an application, 267
 runtime license, 92

S

sampl.c, 267
 sampl.exe, 267, 268
 sampl.obj, 268
 Sample SP Problems, 411
 samplec.mak, 267
 samplevb.frm, 429
 sampling routines, 245
 SC bound type, 576
 scaling, 56
 scatter search, 352
 Scenario Tree, 391
 second order cone, 581
 second-order cone
 examples, 311
 second-order-cone optimization, 311
 selective constraint evaluation, 60
 semi-continuous variable, 576
 sense, of objective, 565
 sensitivity analysis, 433
 separable, 335
 serial number, 23
 setting parameters, 42, 48, 466
 Setting up SP Models, 393
 simple lower bound, 151
 simple lower/upper bound, 587, 589
 simplex method, 69, 77, 80, 94, 135, 486
 dual, 57, 139, 265
 iterations, 227
 primal, 57, 139, 265
 Simplex method, 2
 sine, 340
 size of version, 23, 91, 567, 568
 slack values, 150, 152, 497
 SLB, 587, 589
 SLP pricing, 60, 137
 smooth models, viii, 350, 354
 SOCP, 311
 MPS format, 314
 SOCP Constraints, 322
 SOCP Form, 321
 Solaris, 8
 solution, 259, 287, 494
 analyzing, 433
 dual, 148
 incumbent, 83, 84, 227, 430
 infeasible, 31, 216, 435, 460
 nonoptimal, 588, 589
 primal, 148, 150, 543
 query routines, 144, 491
 rounded, 588, 589
 unbounded, 31, 435, 437, 460
 writing, 34, 463
 solver, 329
 barrier, 53, 58, 60, 77, 80, 92, 94, 139, 265
 branch-and-bound, 142, 230, 287, 293, 375, 491,
 547
 enumeration, 74
 global solver, vii, 141, 354, 375, 490
 initialization, 135, 486
 interrupt, 53, 422, 430, 568
 knapsack, 74
 multistart, vii, 3
 multistart solver, 371
 nonlinear, vii, 62, 139, 295
 progress, 421
 quadratic, vii
 solver status, 279, 422, 432, 433
 type, 565
 SOS, 577

SOS2 set, 578
 sparse matrix representation, 10–12, 114, 329, 473
Spearman rank correlation, 415
 Special Ordered Sets, 577
 splitting lines, 586
 square root, 339
 stack based computer, 338
 staffing model, 279
 stage, 38, 39, 97, 110, 111, 128, 129, 130, 134, 154, 158, 180, 181, 186, 189, 191
 standard Normal cdf, 340
 standard Normal inverse, 343, 346
 start, column, 11, 12, 285, 291
 starting basis, 63, 135, 486
 starting points, 116, 139, 353, 475, 487, 489
 status of variables, 144, 147, 213
 steepest edge pricing, 57, 60
stochastic information, 109
 stochastic programming, 353, 389
 Stochastic Programming, 389
 stochastic solver, vii
 storing data, 21
 strong branching, 78
 structure creation/deletion routines, 21, 452
 student *t* distribution, 341
 Student-t distribution, 416
 SUB, 587, 589
 subtraction, 339
 successive linear programming, 3
 sufficient set, 216, 217, 436, 437, 539, 540
 summation, 342
 supported options, 23
 symmetric, 581
 symmetric matrix., 298
 syntax, 269, 585

T

t distribution, 341
 tangent, 340
 text format (ASCII), 26
 thread safe, 421, 431
 three vector representation, 11
 time limit, 58, 70, 79, 80, 89, 91, 95, 568
 title, 119, 168, 477, 505, 587, 589
 tolerances, 55, 63, 72, 73, 77, 82, 83
 transformation
 backward, 221, 541
 forward, 222, 542
 trial license, 91
 triangular distribution, 343
 true, 340
 types of constraints
 adding, 194, 518
 C++ example, 264

errors, 565
 getting, 160, 162, 164, 165, 498, 500, 502, 503, 552
 loading, 118, 476, 553
 modifying, 528
 types of cuts, 69, 70, 72
 types of data, 17, 42, 263

U

unbounded, 31, 433, 435, 437, 460, 589
 MATLAB, 535, 540
 unformatted MPS file, 26, 565
 uniform distribution, 344
 Uniform distribution, 416
 unsupported features, 567
 upper bounds
 adding, 196
 best, 212
 getting, 166
 LINDO files, 587
 loading, 118
 MATLAB, 475, 476, 503, 520, 552, 553
 MIPs, 71
 modifying, 209, 533
 MPS files, 573
 nonlinear programs, 117, 336
 objective, 167, 504
 SUB, 587, 589
 Visual Basic example, 276
 upper triangle, 298
 USER function, 342
 user interface, 329, 421, 428
 Usercalc(), 347
 user-defined function, 381

V

value vector, 11
 Value-At-Risk, 326
 variables
 adding, 194, 195, 520
 artificial, 55, 63
 binary, 26, 167, 504, 587, 588
 block structure, 137
 bounded, 118, 166, 196
 bounded, MATLAB, 475, 476, 503, 520, 552, 553
 branch-and-bound, 139
 branching on, 78, 124, 486
 branching priorities, 136, 138, 489
 coefficients, 167, 504
 continuous, 139, 148, 158, 352, 589
 decision, 285, 291
 defining, 367

- deleting, 203, 526
 - discrete, 352
 - displaying, 265
 - dual, 145, 492, 493
 - environment, 267
 - errors, 568
 - fixed, 72, 81, 573
 - free, 573, 587, 588
 - general integer, 167, 279, 504, 587, 588
 - getting, 167, 504
 - index of, 167
 - initial values, 139, 487, 489
 - integer, 148, 158, 573
 - integer feasible tolerance, 73, 77
 - internal index, 178, 179, 203, 514, 515, 526
 - left-hand sides, 587
 - limit, 91
 - loading, 482, 483, 484, 485
 - long, 269
 - MIPs, 279, 287
 - modifying, 209
 - name hashing, 238
 - names, 119, 125, 168, 178, 179, 505, 514, 515, 573, 585
 - negative, 573, 587
 - nonlinear, 91, 120, 170, 172, 478, 507
 - number of, 115, 117, 473, 474, 476, 503
 - primal, 151, 495
 - priorities, 136
 - quadratic, 121, 197
 - reduced costs, 151, 152, 496
 - slack/surplus values, 55, 63, 150, 152, 497
 - splitting lines, 586
 - status, 144, 147, 213
 - types of, 123, 124, 167, 180, 481, 482, 484, 485, 504, 516, 533
 - values, 277
 - variance reduction, 416
 - VB, 277
 - VB modules, 429
 - vcvars32.bat, 267
 - Vector AND, 343
 - Vector OR, 343
 - vector Push, 345
 - vectors, 11, 12, 119, 125, 221, 329, 337, 477, 520
 - versions, 23, 58, 91, 297, 315, 567, 568
 - violated constraints, 55, 63
 - Visual Basic, 233, 235
 - Visual Basic example, 269, 287, 429
 - Visual Basic for Applications, 429
 - Visual C++ 6, 266
 - Visual C++ example, 280
- ## W
- warm start, 239, 240, 353, *See also* initial values
 - Weibull distribution, 416
 - wizard, 282
 - wrap function, 341
 - wrapping, 286, 292
 - writing
 - dual, 458, 459
 - LINDO format, 461, 586
 - LINGO format, 461
 - models, 25
 - MPS format, 458, 462, 463
 - solutions, 34, 463
-