Contents

Solver Reports

Programming the Solver

Using Frontline's Solvers

Thank you for using Frontline Systems's Solver products for Lotus 1-2-3 97 Edition. This Helpfile covers the features of the standard Solver for 1-2-3, which is offered on a trial basis free to Lotus 1-2-3 97 users.

This Helpfile provides information about the Solver's features and options, and describes how to use the Solver most effectively. It will help you set up and solve problems, and interpret the results of running the Solver such as the completion messages and reports.

You can obtain a low-cost permanent license to use the standard Solver directly from Frontline Systems. When you're ready, you can upgrade to more powerful versions of the Solver, as they become available. When you upgrade, your Solver models and macros developed with the standard version will work as-is with the enhanced Solver products.

The Enhanced Solver Products

The Premium Solver

The Premium Solver is Frontline's basic upgrade to the standard Solver, and is the foundation upon which our other enhanced Solvers are built. Please contact Frontline Systems for the latest news on availability of the Premium Solver for Lotus 1-2-3 97 Edition.

Once the Premium Solver is installed, you can solve nonlinear optimization problems (NLPs) of up to 400 variables (or changing cells) and 200 constraints in addition to bounds on the variables, and linear optimization problems (LPs) of up to 800 variables with no limit on the number of constraints. User interface improvements make it easier to set up your models and give you better control of various aspects of the solution process. Solution speed is improved for all types of problems, especially for problems with all linear and integer constraints.

The Quadratic Solver

The Quadratic Solver includes all of the features of the Premium Solver, plus a new, much faster algorithm for solving quadratic optimization problems (QPs). These problems have a quadratic objective function (in which variables can be squared or multiplied by each other) and all linear constraints; they are often used to find "efficient portfolios" of securities using the Markowitz or Sharpe methods. Where the standard Solver employs the much slower nonlinear GRG method on problems of this type, the Quadratic Solver employs methods specially suited for these problems to find more accurate optimal solutions in a small fraction of the time that otherwise would have been required. Please contact Frontline Systems for the latest news on availability of the Quadratic Solver for Lotus 1-2-3 97 Edition.

The Large-Scale LP Solver

The Large-Scale LP Solver combines the power of the most advanced LP solution methods with the features of the Premium Solver. You can solve "industrial strength" LP models in a form that encourages rapid development and modification, and present results in a format that your management or client will understand.

Because the Large-Scale LP Solver represents problems in an internal "sparse matrix" form, it can handle LP models of many thousands of variables and constraints, which would require megabytes of memory and hours of solution time using the standard Solver, or even the Premium Solver.

Due to the intrinsic finite precision of computer arithmetic, which becomes an important factor in large LP models, a standard Simplex method such as that used in the standard Solver or the Premium Solver may be foiled by numerical "instability" and fail to find the optimal solution. The Large-Scale LP Solver uses sophisticated methods of scaling, matrix factorization and pivoting to overcome the problems of finite precision arithmetic; it can find solutions to LP models which would have been considered numerically intractable a few years ago. The same methods contribute to solution speed on problems that formerly took many "pivots" or iterations to solve.

Please contact Frontline Systems for the latest news on availability of the Large-Scale LP Solver for Lotus 1-2-3 97 Edition.

How to Use This Helpfile

"Custom Installation" describes steps you can take to install the Solver for 1-2-3 97 Edition if you had trouble with the automatic installation program. If you have successfully installed the Solver, you can skip this section.

"Solver Basics" reviews the basic framework of the optimization problems which can be handled by the Solver. It describes how a model is made up of variables, constraints and an objective, and covers the major categories of optimization problems (linear and quadratic programming, nonlinear programming, and integer programming) handled by the Solver.

"Building Solver Models" is an introduction to the process of building optimization models in Lotus 1-2-3, translating from algebraic notation to spreadsheet formulas and Solver dialog choices. It also covers multiple selections for decision variables, use of the Variables button, the possible forms of constraint left- and right-hand sides, and use of the "bin" dropdown to specify binary integer variables.

"Diagnosing Solver Problems" helps you determine what is wrong if you don't get the solution you expect from the Solver, or if you encounter a message other than "Solver found a solution." It outlines the most common problems that users have, based on our technical support experience with the Solver.

"Solver Options" documents in depth the advanced options and tolerances which can be set using the Solver Options dialog. The effect of each option and situations where you would likely choose it are described.

"Solver Reports" describes the contents of the reports which may be chosen from the Solver Results dialog. It shows you how to interpret the numbers in the reports, and how to use the Sensitivity Report to predict changes in the optimal solution in response to certain kinds of changes in your input data.

"Programming the Solver" describes how you can control the Solver from LotusScript and create "turn-key" applications using the Solver.

Telephone Support and the World Wide Web

Frontline Systems offers telephone technical support for the Solver on a fee basis, and also maintains an extensive World Wide Web support site which you can access free of charge. Most questions can be answered from this Helpfile or by consulting our Web site -- both are based on our telephone support experience. We encourage you to use our free resources first!

To learn about Solver licensing and telephone technical support, click on the Help button in the Solver Parameters, Solver Options or Solver Results dialog. You will see a dialog box which reports the status of your Solver license: If you are using the trial version, this dialog shows how many problem solution attempts (clicks of the Solve button) remain before the trial license expires. If you have purchased a permanent license, it shows your user name and registration serial number. To purchase a permanent license now, click the Register button. Also included in this dialog is telephone support and Web site information. Click the Help button in this dialog to reach the Solver Helpfile.

If you have access to the World Wide Web via the Internet, you will find a wealth of current information about the Solver -- over 65 pages at this writing -- at Frontline Systems' Web site, **http://www.frontsys.com**. Since this site is frequently updated, it is well worth while to check it periodically for the latest news about the standard and enhanced Solvers. The World Wide Web is currently our best way of delivering technical support assistance for both our standard and our enhanced Solvers. If you can't find the answer you need, you can send email to Frontline Systems directly from your Web browser with a single mouse click (or, if you're using a separate email system, send it to **info@frontsys.com**).

To reach Frontline Systems by phone, fax, or postal mail, please use:

Frontline Systems, Inc.
P.O. Box 4288
Incline Village, NV 89450, USA
Tel (702) 831-0300
Fax (702) 831-0314

Further Reading

Although this Helpfile will provide many valuable hints for making effective use of the Solver, it does not attempt to teach you how to formulate Solver models or apply linear and quadratic programming, nonlinear programming or integer programming techniques. To make the most of the Solver, we strongly recommend that you consult one of the books cited below, or discuss your problem with someone in your firm or at your local university with a background in operations research and/or management science. There is a vast literature on problems of various types and for various industries and business situations which have been solved successfully with the methods available in the Solver. Don't reinvent the wheel -- find out how others have solved problems similar to yours!

Spreadsheet Modeling and Decision Analysis by Cliff T. Ragsdale, published by Course Technology. ISBN 1-56527-277-3. This book uses the Microsoft Excel Solver, which is quite similar to the 1-2-3 97 Solver, for all of its examples. You'll find a discussion of linear, nonlinear and integer programming; an explanation of sensitivity analysis and how to use the Solver's reports; topics like goal programming and multi-objective optimization; and additional coverage of regression, time series analysis, queuing, project management, decision analysis, and other topics.

Practical Management Science: Spreadsheet Modeling and Applications by S. Christian Albright and Wayne L. Winston, published by Duxbury Press. ISBN 0-534-21774-5. This is a new book based on Winston's well-regarded textbook *Operations Research: Applications and Algorithms* (see below), updated to use spreadsheet Solvers throughout. It is notable for its extensive, in-depth coverage of classic optimization problems, including many in the exercises. It also includes coverage of decision making under uncertainty, inventory models, queuing, simulation and forecasting.

Managerial Spreadsheet Modeling and Analysis by Rick Hesse, published by Richard D. Irwin. ISBN 0-256-21530-8. This book teaches you how to formulate a model from a complex business situation, using a four-step process: Picture and paraphrase, verbal model, algebraic model and spreadsheet model. It covers types of models ranging from simple goalseeking and unconstrained problems to linear, nonlinear and integer programming problems.

Management Science: Modeling, Analysis and Interpretation by Jeffrey D. Camm and James R. Evans, published by South-Western College Publishing. ISBN 0-538-82738-6. This book focuses on modeling and covers both optimization and simulation. It uses the Excel Solver and the (earlier) Lotus Solver in many examples, though some examples use the older LINDO optimizer which has its own language for expressing LP models. It covers multiobjective LP models, integer models and network models, but does not cover nonlinear optimization models.

Model Building in Mathematical Programming, Third Edition by H.P. Williams, published by John Wiley. ISBN 0-471-92580-2 (HC), 0-471-94111-5 (SC). Written before the advent of spreadsheet optimizers, this book is still valuable for its explanation of model-building methods, especially if you are building larger-scale optimization models. It focuses on linear and integer programming, mentioning nonlinear models only briefly, but it offers a unique treatment of large-scale model structure and decomposition methods. It also includes a complete discussion of 20 models drawn from various industries.

Operations Research: Applications and Algorithms, Third Edition by Wayne L. Winston, published by Duxbury Press. ISBN 0-534-20971-8 (with DOS software), 0-534-20973-4 (with Mac software). This popular textbook, also written before the advent of spreadsheet optimizers, covers many classic

optimization problems and also includes a discussion of some of the algorithms used in the Solver, such as the Simplex method for linear programming, the Branch & Bound method for integer programming, and selected methods for nonlinear programming -- as well as many other topics in operations research. An edition with (non-spreadsheet-based) Windows software is due in the Fall of 1996, with ISBN 0-534-52020-0.

Management Science by Andrew S. Shogan, published by Prentice-Hall, Inc. ISBN 0-13-551219-0. Another popular textbook written before the advent of spreadsheet optimizers, with in-depth coverage of the Simplex and Branch & Bound methods as well as many classic optimization problems. Also includes a detailed discussion of sensitivity analysis, goal programming, and piecewise linear programming -- as well as many other management science topics -- but does not cover nonlinear programming.

For a technical description of the nonlinear GRG solver included with the standard and enhanced Solvers, please consult the following academic papers:

L.S. Lasdon, A. Waren, A. Jain and M. Ratner. Design and Testing of a Generalized Reduced Gradient Code for Nonlinear Programming. *ACM Transactions on Mathematical Software* 4:1 (1978), pp. 34-50.

L.S. Lasdon and S. Smith. Solving Sparse Nonlinear Programs Using GRG. *ORSA Journal on Computing* 4:1 (1992), pp. 2-15.

Extracting Files

Read this section ONLY if you have had trouble with automatic installation of the Solver. It explains how to manually install the Solver files and configure 1-2-3 to use the Solver. It assumes that you have experience working with Windows 95 or Windows NT files, directories and the Registry.

To install the Solver automatically, simply run the program SOLVE123.EXE on the computer where you have already installed 1-2-3 97 Edition. The automatic installation process performs all of the steps outlined in the sections "Extracting Files," "Modifying the Registry" and "Using the Files Add-Ins Menu."

To install the Solver manually, first create a **solver** subdirectory within the **\lotus\123** directory which was created when 1-2-3 97 Edition was installed. Then open the file SOLVE123.EXE using a archiver program such as PKZIP or WinZip, and extract all of the archived files into the newly created **solver** subdirectory.

Modifying the Registry

To perform this step, you must use the Registry Editor. From the **Start** Menu, choose **Run** and type or select the program name **regedit**. The Registry is organized hierarchically, much like the Windows file system. Double-click on each key level to display:

HKEY_LOCAL_MACHINE\SOFTWARE\Lotus\Components\LotusScriptExtensions\2.0

Choose **Edit New String Value**. For the name, type **SOLVER** (this must be all capitals). For the value, type the complete path to the Solver DLL, such as **c:\lotus\123\solver\solve123.dll**. Now, move to:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Help

Choose **Edit New String Value**. For the name, type **solver.hlp**. For the value, type the complete path to the *directory containing* the Solver Helpfile, such as **c:\lotus\123\solver**. Finally, choose **Registry Exit** to dismiss the Registry Editor.

Using the Files Add-Ins Menu

The final step is to *register* the Solver add-in with 1-2-3 97 Edition. To do this, start 1-2-3 and select the **Add-Ins** option from the **File** menu. Then select **Manage Add-Ins...** from the cascading submenu. A dialog box should appear, containing a list of add-ins (which may be empty). You will add the Solver add-in to this list, and specify that it should be auto-loaded.

Click on the **Register...** button. In the Register Add-In dialog, which operates like a File Open dialog, navigate to the directory containing **solver.12a** (normally **c:\lotus\123\solver**). Click on **solver.12a** in the file list, then click on the **Open** button.

The Manage Add-Ins dialog should reappear, this time listing the Solver add-in's path, such as **c:\lotus\123\solver\solver.12a**. Now click just to the left of the drive letter on this line, so that a check mark appears next to the full pathname. Then click on the **Done** button.

To verify that the Solver add-in has been registered, select the **Analyze** option from the **Range** menu. On the cascading submenu, you should see a new **Solver...** option. You select this menu option to activate the Solver and display the Solver Parameters dialog.

Now you can open **solvsamp.123**, which should be in the directory **c:\lotus\123\solver**, and try out the six Solver examples on tabs B through G of this workbook. You can also set up and solve a problem manually as described step-by-step in sheet tab A.

Elements of Solver Models

The basic purpose of the Solver is to find a *solution* -- that is, values for the *variables* or Changing Cells in your model -- which satisfies the *constraints* and which maximizes or minimizes the *objective* or Set Cell value. Let us examine this framework more closely.

The model you create for use with the Solver is no different from any other spreadsheet model. It consists of input values; formulas which calculate values based on the input values or on other formulas; and other elements such as formatting. You can practice "what if" with a Solver model just as easily as with any other spreadsheet model. This familiar concept can be very useful when you wish to present your results to managers or clients, who are usually "spreadsheet literate" even if they are unfamiliar with Solvers or optimization.

Related Topics:

Decision Variables and Parameters

The Objective Function

Constraints

Feasible and Optimal Solutions

Linear and Nonlinear Functions

Quadratic Functions

Functions to Avoid: Discontinuities

Decision Variables and Parameters

The input values may be fixed numbers associated with the particular problem. We'll call these values *parameters* of the model. Often you will have several "cases" or variations of the same problem to solve, and the parameter values will change in each problem variation.

Alternatively, the input values may be quantities which are variable, or under the control of the decision maker. We'll refer to these as the variables, *decision variables*, or Changing Cells. Very often, the same cell values you use to play "what if" are the ones for which you'll want the Solver to find solution values. These cells are listed in the Changing Cells edit box of the Solver Parameters dialog.

The Objective Function

The quantity you want to maximize or minimize is called the *objective function* or Set Cell. This cell is listed in the Set Cell edit box of the Solver Parameters dialog.

You may have a Solver model which has nothing to maximize or minimize, in which case the Set Cell edit box will be blank. In this situation the Solver will simply find a solution which satisfies the constraints.

The Solver also permits you to enter a specific value which you want the objective function or Set Cell to achieve. This feature was included for compatibility with the **Range Analyze Backsolver...** option, which allows you to seek a specific value for a cell by adjusting the values of other cells on which it depends. In fact, entering a specific value for the Solver's Set Cell is exactly the same as leaving the Set Cell blank and entering an equality constraint for the Set Cell in the Constraint List Box.

There is rarely a good reason to use the Set Cell Value of edit box in the Solver Parameters dialog. If you have a simple "goalseeking" or "backsolving" problem, you will find it convenient to use the **Range Analyze Backsolver...** option. If you have nothing to maximize or minimize, we recommend that you leave the Set Cell blank and enter any constraints you need in the Constraint List Box.

Constraints

Constraints are relations such as A1 >= 0. A constraint is *satisfied* if the condition it specifies is true *within a small tolerance.* This is a little different from a logical formula such as +A1>=0 evaluating to 1 or 0 which you might enter in a cell. In this example, if A1 were -0.0000001, the logical formula would evaluate to 0, but with the default Solver Precision setting, the constraint would be satisfied. Because of the numerical methods used to find solutions to Solver models and the finite precision of computer arithmetic, it would be unrealistic to require that constraints like A1 >= 0 be satisfied exactly -- such solutions would rarely be found.

In the 1-2-3 97 Solver, constraints are specified by giving a cell reference such as A1 or A1..A10 (the "left hand side"), a relation (<=, = or >=), and an expression for the "right hand side." Although you can enter any numeric expression on the right hand side, we encourage you to use only *constants*, or references to cells which contain *constant values* on the right hand side. (A constant value to the Solver is any value which does not depend on any of the decision variables.) Using constant right hand sides in constraints will simplify your model, and allows the Solver to handle the constraints more efficiently.

Another type of constraint is of the form A1 = integer, where A1 is one of the decision variables. This specifies that the solution value for A1 must be an integer or whole number such as -1, 0 or 2 *to within a small tolerance.* A common special case, also supported by the Solver, is a constraint such as A1 = binary which specifies that A1 must be either 0 or 1 at the solution. The presence of even one such integer constraint in a Solver model makes the problem an integer programming problem (discussed below), which may be much more difficult to solve than the equivalent problem without the integer constraint.

Feasible and Optimal Solutions

A solution (values for the decision variables) for which all of the constraints in the Solver model are satisfied is called a *feasible solution.* The Solver proceeds by first finding a feasible solution, and then seeking to improve upon it, changing the decision variables to move from one feasible solution to another feasible solution until the objective function has reached its maximum or minimum. This is called an *optimal solution.*

How does the Solver know (more important, how do *you* know) that the solution is optimal? The answer to this question depends on the type of problem (linear/quadratic, nonlinear or integer) you are trying to solve, which is discussed at some length in the next section.

Linear and Nonlinear Functions

The objective function in a Solver problem is some calculated value that depends on the decision variable cells; the job of the Solver is to find some combination of values for the decision variables which maximizes or minimizes the objective function. During the optimization process, *only the decision variable cells are changed*; all other "input" cells are held constant. If you analyze the chain of formulas which calculates the objective function value, you will find that parts of those formulas (those which refer to non-decision variable cells) are unchanging in value and could be replaced by a numeric constant for the purposes of the optimization.

If you follow the suggestion above and use only constant values on the right hand sides of constraints, then the same observation applies to the left hand sides of the constraints: Parts of the constraint formulas (those which refer to non-decision variable cells) are unchanging in value, and only the parts dependent on the decision variables "count" during the optimization.

The mathematical form of the relationship between the objective function and constraint cells and the decision variables has important implications for the difficulty of the problem, the Solver "engine" that can be used, and the speed of solution. The simplest and most common case is where the objective function is a *linear* function of the variables. This means that the objective can be written as a sum of terms, where each term consists of one decision variable multiplied by a (positive or negative) constant. Algebraically, we can write:

$$\max \text{ (or min)} \quad a_1x_1 + a_2x_2 + \ldots + a_nx_n$$

where the a_is, which are called the *coefficients*, stand for constant values and the x_is stand for the decision variables. Remember that the a_is need only be *constant in the optimization problem*, i.e. not dependent on any of the decision variables. As an example, suppose that the objective function is +B1/B2*C1+(D1*2+E1)*C2, where only C1 and C2 are decision variables, and the other cells contain constants (or formulas that don't depend on any of the decision variables). This would still be a linear function, where a_1 = B1/B2 and a_2 = (D1*2+E1) are the coefficients, and x_1 = C1 and x_2 = C2 are the variables.

Note that the @SUMPRODUCT function computes exactly the algebraic expression shown above. If we were to place the formula +B1/B2 in cell A1, and the formula +(D1*2+E1) in cell A2, then we could write the example objective function as:

@SUMPRODUCT(A1..A2,C1..C2)

A *nonlinear* function, as its name implies, is any function of the decision variables which is not linear, i.e. which cannot be written in the algebraic form shown above. Examples would be +1/C1, @LOG(C1), +C1^2 or +C1*C2 where both C1 and C2 are decision variables. If the objective function or any of the constraints are nonlinear functions of the variables, then the problem cannot be solved with an LP Solver.

Quadratic Functions

The last two examples above, +C1^2 or +C1*C2, are simple instances of *quadratic* functions of the variables. A more complex example would be:

+2*C1^2+3*C2^2+4*C1*C2+5*C1

A quadratic function is a sum of terms, where each term is a (positive or negative) constant (again called a *coefficient*) multiplied by a single variable or the product of two variables. Common uses for quadratic functions are to compute the mean squared error in a curve-fitting application, or the variance or standard deviation of security returns in a portfolio optimization application.

What if you have already created a complex spreadsheet model without using functions like @SUMPRODUCT, and you aren't sure whether your objective function and constraints are linear or nonlinear functions of the variables? A quick way to find out is to try solving the model with Assume Linear Model box checked in the Solver Options dialog. Although this test is not 100% foolproof, it will in most cases return an error message if the problem contains nonlinear functions of the variables. In any case, the next step is to look closely at the formulas in your spreadsheet and to try rewriting them using the @SUMPRODUCT product. Your effort will be rewarded in most cases by a better understanding of your model, as well as a problem which Frontline's enhanced Solvers can handle efficiently.

Functions to Avoid: Discontinuities

There are many formulas and built-in functions which you can use in a 1-2-3 spreadsheet, but which cause difficulties for both linear and nonlinear solvers. These functions share the property that they are non-smooth or *discontinuous* at some point. The most common example is the IF function. For example:

@IF(A1>10,B1,2*B1)

is discontinuous around A1=10 because its value "jumps" from whatever value B1 has to twice that value. A nonlinear solver relies on information from partial derivatives to guide it towards a feasible and optimal solution; since it is unable to compute the partial derivatives of a function at points where that function is discontinuous, it cannot guarantee that any solution it finds is truly optimal. In practice, the nonlinear GRG algorithm included with the standard Solver can sometimes deal with discontinuities which are "incidental" to the problem, but as a general statement, the Solver cannot handle problems where the objective function or some of the constraints are discontinuous.

A partial list of the most common 1-2-3 functions which are discontinuous at certain points would include the ones listed below.

@ABS
@MIN
@MAX
@INT
@ROUND
@IF
@CHOOSE
@CEILING
@FLOOR
@COUNT

If you aren't sure about a particular function, try graphing it (by hand or in 1-2-3) over the expected range of the decision variables; this will usually reveal whether the function is smooth or discontinuous.

Linear and Nonlinear Programming

A model in which the objective function and *all* of the constraints (other than integer constraints) are linear functions of the decision variables is called a *linear programming* (LP) problem. If the objective function or any of the constraints is not a linear function of the decision variables, the model is called a *nonlinear programming* (NLP) problem. (The term "programming" dates from the 1940s and the discipline of "planning and programming" where these solution methods were first used; it has nothing to do with computer programming.) If the problem includes integer constraints, it is called an *integer linear* or *integer nonlinear* programming problem, respectively. A linear programming problem with some "regular" (continuous) decision variables, and some variables which are constrained to integer values, is called a *mixed-integer programming* (MIP) problem.

A *quadratic programming* (QP) problem can be thought of as a generalization of a linear programming problem, or as a restricted case of a nonlinear problem. Its objective is a quadratic function of the decision variables, and *all* of its constraints must be *linear* functions of the variables. A QP problem cannot be solved with a linear programming "engine" such as the Large-Scale LP Solver. Since a QP problem is a special case of an NLP problem, it *can* be solved with the standard GRG nonlinear solver in 1-2-3, but this may take far more time than solving an LP of the same size. Frontline's Quadratic Solver includes special methods for efficiently solving QP problems.

Related Topics:

Linear Programming

Quadratic Programming

Nonlinear Programming

Linear Programming

Linear programming problems are intrinsically easier to solve than nonlinear problems. In an NLP there may be more than one feasible region and the optimal solution might be found at any point within any such region. In contrast, an LP has at most one feasible region with "flat faces" (i.e. no curves) on its outer surface, and the optimal solution will always be found at a "corner point" on the surface where the constraints intersect. (In some problems there may be multiple optimal solutions, all of them lying along a line between corner points, with the *same* objective function value.) This means that an LP Solver needs to consider many fewer points than an NLP Solver, and it is always possible to determine (subject to the limitations of finite precision computer arithmetic) that an LP problem (i) has no feasible solution, (ii) has an unbounded objective, or (iii) has an optimal solution (either a single point or multiple equivalent points along a line).

Related Topics:

Problem Size and Numerical Stability

The Simplex Method

Problem Size and Numerical Stability

Because of their structural simplicity, the main limitations on the size of LP problems which can be solved are time, memory, and the possibility of numerical "instabilities" which are the cumulative result of the small errors intrinsic to finite precision computer arithmetic. The larger the model, the more likely it is that numerical instabilities will be encountered in solving it.

Most large LP models are *sparse* in nature: While they may include thousands of decision variables and constraints, the typical constraint will depend upon only a few of the variables. This sparsity can be exploited to save memory and gain speed in solving the problem.

The Simplex Method

LP problems are generally solved via the Simplex method. The standard Solver uses a straightforward implementation of the Simplex method to solve LP problems, when the Assume Linear Model box is checked in the Solver Options dialog. The Premium and Quadratic Solvers use an improved implementation of the Simplex method, when the Simplex or LP/Quadratic Solver is chosen from the Solver "engine" dropdown list in the Solver Parameters dialog. The memory required by this Simplex code increases with the number of variables *times* the number of constraints, regardless of the model's sparsity.

The Large-Scale LP Solver uses a far more sophisticated implementation of the Simplex method, which fully exploits sparsity in the LP model to save time and memory. To cope with potential numerical instabilities, the Large-Scale LP Solver uses techniques such as automatic scaling, matrix factorization using the LU decomposition with the Bartels-Golub update, steepest-edge pivoting strategies, and dynamic Markowitz refactorization. These same techniques often result in much faster solution times -- making it practical to solve LP problems with thousands of variables and constraints.

Quadratic Programming

Quadratic programming problems are more complex than LP problems, but simpler than general NLP problems. Such problems have only one feasible region with "flat faces" on its surface, but the optimal solution may be found anywhere within the region or on its surface. Frontline's Quadratic Solver uses the Simplex method to determine the feasible region, then uses special methods based on the properties of quadratics to find the optimal solution.

Large QP problems are subject to many of the same considerations as large LP problems: When using a straightforward or "dense" representation as in Frontline's current Quadratic Solver, the amount of memory required increases with the number of variables *times* the number of constraints, regardless of the model's sparsity. Numerical instabilities can arise in QP problems and may cause *more* difficulty than in similar-size LP problems. To deal with these issues, Frontline hopes to offer a large-scale sparse quadratic solver in the future.

Nonlinear Programming

As outlined above, nonlinear programming (NLP) problems are intrinsically more difficult to solve than LP and QP problems. Because of the possibility of multiple feasible regions and multiple locally optimal points within such regions, there is no known way to determine with certainty that the problem is infeasible, the objective unbounded, or that an optimal solution is the "global optimum" across all feasible regions. However, many common NLP problems have a simpler structure than this general description, and are more amenable to solution.

It is important to realize that an NLP Solver, like the one in 1-2-3, applies the same method to *all* problems, even those that are really LPs or QPs. If you don't check the Assume Linear Model box in the Solver Options dialog, or (in the enhanced Solvers) select another solver from the dropdown list box in the Solver Parameters dialog, the default GRG Nonlinear Solver will be used. This solver may have difficulty with LP or QP problems that could have been solved easily with one of the other solvers.

Related Topics:

The GRG Method

The GRG Method

In the standard Solver, NLP problems are solved with the GRG (Generalized Reduced Gradient) method as implemented in Lasdon and Waren's GRG2 code. This method and specific implementation have been proven in use over many years as one of the most robust and reliable approaches to solving difficult NLP problems.

The GRG method is subject to the intrinsic limitations cited above on its ability to find the globally optimal solution. However, limited guarantees can be made about the GRG method's ability to find a "local optimum," in particular where the objective function and all of the constraints are twice continuously differentiable. When these are combined with your knowledge of problem structure in a specific case, the result will often be a definitive "optimal solution." For more information on this topic, please consult the references cited in the Using Frontline's Solvers.

As with the Simplex method, the GRG method in the standard Solver uses a "dense" problem representation, and its memory and solution time increases with the number of variables *times* the number of constraints. It is also subject to problems of numerical instability, which may be even more severe than for LP and QP problems. In the future, Frontline Systems plans to release a Large-Scale NLP Solver based on Lasdon and Waren's LSGRG code, which uses sparse storage methods and more sophisticated numerical techniques specific to nonlinear models.

Integer Programming

When a Solver model includes integer constraints, it is called an integer programming problem. Solving such problems may require *far* more computing time than the same problem without the integer constraints.

The standard Solver, the Premium and Quadratic Solvers, and the Large-Scale LP Solver all use the Branch and Bound method to find optimal solutions to such problems. However, refinements to this method in the enhanced Solver products will often result in considerably faster solutions than those obtained with the standard Solver.

Related Topics:

The Branch & Bound Method

The Branch & Bound Method

The Branch & Bound method begins by finding the optimal solution in the absence of the integer constraints. If it happens that in this solution, the decision variables whose values are constrained to be integers already have integer values, then no further work is required.

If one or more integer variables have non-integral solutions, the Branch & Bound method chooses one such variable and "branches," creating two new subproblems where the value of that variable is more tightly constrained. For example, if integer variable A1 has the value 3.45 at the solution, then one subproblem will have the additional constraint A1 <= 3 and the other subproblem will add the constraint A1 >= 4. These subproblems are solved and the process is repeated, "branching" as needed on each of the integer decision variables, until a solution is found where all of the integer variables have integer values (to within a small tolerance).

Hence, the Branch & Bound method may solve many subproblems, *each one* a "regular" Solver problem. The number of subproblems may grow **exponentially**; the "bounding" part of the Branch & Bound method is designed to eliminate sets of subproblems that do not need to be explored because the resulting solutions cannot be better than the solutions already obtained. Obviously this may take a great deal of computing time.

Integer programming has many important applications. Many of them involve the use of integer decision variables that are further constrained to have values of either 0 or 1; these are called *binary* or *0-1* integer variables. Such variables may be specified in one step with the "bin" dropdown choice in the Add/Change Constraint dialogs. Binary integer variables can be used to represent yes/no decisions, such as whether a pipeline is open, or whether a facility is built at a certain location. For a discussion of applications of integer programming, please consult the references cited in the Introduction.

From Algebra to Spreadsheets

Optimization problems are often described in algebraic terms. In this topic, we'll show how you can translate from the algebraic statement of a problem to a spreadsheet model which the Solver can optimize.

A spreadsheet, however, can do much more than accept and compute values for algebraic expressions. It can help you organize and display the structure of the model you are trying to optimize, through tools such as defined names, formatting and outlining. As models become larger, the problems of managing data for constraints, coefficients, and so on become more significant, and a properly organized spreadsheet model can help manage this complexity.

Related Topics:

Setting Up a Model

Setting Up a Model

To set up an optimization model as a 1-2-3 spreadsheet, you will follow these essential steps:

1. Reserve a cell to hold the value of each decision variable.
2. Pick a cell to represent the objective function, and enter a formula that calculates the objective function value in this cell.
3. Pick other cells and use them to enter the formulas that calculate the left hand sides of the constraints.
4. The constraint right hand sides can be entered as numbers in other cells, or entered directly in the Solver's Add Constraint dialog box.

Within this overall structure, you have a great deal of flexibility in how you lay out and format the cells which represent variables and constraints, and which formulas and built-in functions you use. For example, the formulas needed for a linear programming problem can always be specified with the @SUMPRODUCT function.

Cells for decision variables, the objective function, and the left hand sides of constraints must be on the active worksheet. Constraint right hand sides which are simple cells or cell ranges must also be on the active worksheet. To use a cell on another worksheet, enter a *formula* referencing this cell on the right hand side of a constraint.

Decision Variables and Constraints

You have a great deal of flexibility in how you specify the decision variables and the constraints in the Solver dialogs. In the previous section, we discussed the simplest forms. In this section, we'll cover more general forms of specifying both variables and constraints.

Related Topics:

Variables and Multiple Selections

Using the Variables Button

Constraint Left and Right Hand Sides

Variables and Multiple Selections

The Solver Parameters dialog initially displays just one Changing Cells edit box to specify the decision variables in a model. This edit box accepts cell ranges, which may be typed in as cell coordinates (or as range names equivalent to cell coordinates), or entered by clicking the Range Selector button and selecting the desired cells in the spreadsheet. However, in many Solver models it is convenient to lay out the decision variable cells in several different ranges on the spreadsheet An example is the "Maximizing Income" worksheet in the SOLVSAMP.123 workbook.

Using the Variables Button

If you click on the Variables button, the dialog box changes to show the variable selections in a list box, as shown below. The Variables button is now labeled Constraints, and clicking on it returns you to the original display.

You can now use the Add, Change and Delete buttons to add, modify or remove variable cells, just as you would for constraints. Each row in the Variable Cells list box can contain a multiple selection; however we recommend that you use a simple selection for clarity, since there is no limit on the number of rows in the list box.

Constraint Left and Right Hand Sides

In the simplest cases, constraint left hand sides are single cell references, and the right hand sides are constants entered in the Add Constraint dialog. But the Solver permits more general forms for both the left and right hand sides of constraints.

The constraint *left* hand side, entered in the Cell Reference edit box, may be any cell range, such as a column, row, or rectangular area of cells. The cells you reference must be on the active worksheet. In the example shown earlier, we could have entered all five constraint left hand sides at once, as B1..B5 (as long as we entered the five right hand sides at the same time).

The constraint *right* hand side may be any of the following:

1. A numeric constant such as 1.
2. A cell reference such as C1.
3. A cell range such as C1..C5.
4. An arbitrary formula such as +A:C1+1 or +A:C2/B:D2.
5. Either "integer" or "binary" for integer constraints

Option 5 is for integer constraints *only* and is discussed below under "Using Integer Constraints." When entering a formula (Option 4), be sure to include an explicit sheet reference such as A:C1. This is the only context where you can refer to cells on sheets other than the active worksheet. If you use option 3 -- a selection of more than one cell -- the number of cells selected must match the number of cells you selected for the constraint left hand side. The two selections need not have the same "shape:" For example, the left hand side could be a column and the right hand side a row. You may also use rectangular areas of cells. In any case, when you use this form you are specifying several constraints at once, and the constraint left hand sides correspond element-by-element to the right hand sides. In the example shown earlier, you could have entered the right hand side values 400, 200, 800, 400 and 600 into cells C1 to C5, and entered a single constraint such as B1..B5 <= C1..C5. You can see examples of this form in nearly all of the sample worksheets included with the Solver, as well as throughout this Helpfile. It is by far the most useful form.

If the constraint right hand side is a cell reference, cell selection or formula, the Solver needs to know whether the contents of those cells, or the value of the formula is *constant* in the problem, or *variable* (i.e. dependent on the values of the decision variables). If the right hand side depends on any of the decision variables, the Solver transforms a constraint such as "LHS >= RHS" into "LHS - RHS >= 0" internally. Both the linear and nonlinear Solvers work internally with constant bounds on the constraint functions.

Related Topics:

Implicit Non-Negativity Constraints

Efficiency of Constraint Forms

Using Integer Constraints

Implicit Non-Negativity Constraints

Many Solver problems -- and perhaps *most* LP problems -- have "non-negativity" constraints, or lower bounds of zero on the decision variables. To save you the trouble of entering these constraints explicitly in the Constraints list box, the Solver provides an Assume Non-Negative check box in the Solver Options dialog. When this box is checked, all variables which do not have explicit lower bounds in the Constraint list box are automatically given lower bounds of zero. You can enter constraints such as A1 >= 2 or A1 >= -3 for certain variables, overriding the implicit lower bound, and check the Assume Non-Negative box to give all other variables a lower bound of zero.

Efficiency of Constraint Forms

The Solver recognizes the case where the constraint left hand side is a decision variable, or a set of decision variables. As long as the corresponding right hand sides are constant (i.e. not dependent on any of the variables), these constraints are specially treated as bounds on the variables. The most common instance of a bound on a variable is a non-negativity constraint such as A1 >= 0, but any sort of constant bounds are efficiently handled by both the linear and nonlinear Solvers.

There is no difference in terms of efficiency between a constraint entered (for example) as A1 <= 100 or as A1 <= B1 where B1 contains 100; the Solver recognizes that B1 is equivalent to a constant. The form A1 <= B1 is usually better from the standpoint of maintainability of your optimization model.

On the other hand, a constraint right hand side which is a formula -- even a simple one like 2+2 -- will incrementally increase the solution time for the model. Because the Solver doesn't have the facilities to recognize the right hand side "on the fly," it treats any formula as a RHS potentially dependent on the variables, and internally creates a constraint "LHS - RHS >= 0" -- even if the formula really was a constant bound on a variable. It is better to place whatever formula you need into a cell, and reference that cell as the constraint right hand side.

Using Integer Constraints

Integer constraints can only be applied to cells which are decision variables; hence the cells selected on the left hand side of the constraint must be a subset (or all) of the cells in the By Changing Cells edit box, or the Variable Cell list box. Integer constraints specify that the selected variable cells must have solution values which are integers, such as -1, 0 or 2, to within a small tolerance. Variable cells which have *binary* integer constraints must be either 0 or 1 at the optimal solution.

You specify an integer or binary constraint by selecting the "int" or "bin" choice from the Relation dropdown list in the Add/Change Constraint dialog. The Solver displays such constraints in the Constraint list box in the form "A1..A10 = integer" or "A1..A10 = binary". A binary integer constraint is exactly equivalent to three other constraints: One of the form "A1..A10 = integer", another of the form "A1..A10 >= 0" and a third of the form "A1..A10 <= 1".

Be sure that you select "int" or "bin" from the Relation dropdown list. If you select = from the dropdown list and *type* the word "integer" or "binary" on the right hand side, the Solver will not recognize this as an integer constraint, and clicking on Solve will probably result in the error message "Solver encountered an error value in a target or constraint cell".

If You Aren't Getting the Solution You Expect

This topic is intended to help you diagnose problems with your Solver models. *The most important step you can take* to deal with potential Solver problems is to start out with a clear idea of the type of optimization model you are creating, how it relates to well-known problem types, and whether yours is a linear, nonlinear or integer programming problem -- as discussed in previous topics. If you then build your model in a *well-structured, readable and efficient form* -- as outlined elsewhere in this Helpfile -- diagnosing problems should be relatively easy. But at times you may be "surprised" by the results you get from the Solver.

If the Solver stops with a solution (set of values for the decision variables or changing cells) which is different from what you expect, or what you believe is correct, follow the suggestions below. You can usually narrow down the problem to one of a few possibilities.

Check the Solver Completion Message shown in the Solver Results dialog. Users sometimes contact Frontline Systems about "wrong solutions", but they don't know which Solver Completion Message they received. This is crucial to diagnosing the problem.

Consider carefully the possibility that the solution found by the Solver is correct, and that your expectation is wrong. This may mean that what your model actually says is different from what you intended.

Check the "Show Iteration Results" box in the Solver Options dialog and re-solve. The Solver will pause with the message "Solver paused, current solution values displayed on worksheet." Click Continue to see the path towards the solution taken by the Solver.

Consider the possibilities for *poorly scaled models*, role of the *Tolerance option* for integer problems, limitations on *nonlinear problems*, and problems with *discontinuous functions* outlined below.

Solver Completion Messages

One of the Solver Completion messages described on the following pages will appear in the Solver Results dialog whenever the Solver stops running.

These messages are numbered with the result codes which can be returned when you invoke the Solver in a user-written program in LotusScript. It's a good idea to test for these codes in your programs, as outlined in the topic "Programming the Solver."

Some of these messages have a slightly different interpretation depending on which Solver "engine" you are using -- the linear Simplex Solver, nonlinear GRG Solver, or the Branch & Bound method for problems with integer constraints. See the explanations of each message, particularly the first one, "Solver found a solution."

Related Topics:

0. Solver found a solution. All constraints and optimality conditions are satisfied.

This means that the Solver has found the optimal or "best" solution under the circumstances. The exact meaning depends on whether you are solving a linear, nonlinear, or integer programming problem:

If you are solving a *linear* programming problem with no integer constraints, the Simplex Solver has found the *globally* optimal solution: There is *no* other solution satisfying the constraints which has a better value for the objective (Target Cell). It is possible that there are other solutions with the *same* objective value, but all such solutions are linear combinations of the current decision variable values.

If you are solving a *nonlinear* programming problem with no integer constraints, the GRG Solver has found a *locally* optimal solution: There is no other set of values for the decision variables close to the current values and satisfying the constraints which yields a better value for the objective (Target Cell). In general, there *may* be other sets of values for the variables, far away from the current values, which yield better values for the objective.

If you are solving a mixed-integer programming problem (any problem with integer constraints), the Solver has found a solution which was within the range of the true integer optimal solution allowed by the Tolerance value in the Solver Options dialog (5% by default). If the problem is linear or quadratic, the true integer optimal solution (within the Tolerance) has been found. If the problem is nonlinear, the Branch & Bound process has found the best of the locally optimal solutions found for subproblems by the GRG method.

1. Solver has converged to the current solution. All constraints are satisfied.

This message appears only when the GRG Solver is used. It means that the Solver stopped because the objective function value is changing very slowly for the last few iterations or trial solutions. More precisely, the GRG Solver stops if the absolute value of the *relative* change in the objective function is less than the value of the Convergence box in the Solver Options dialog. A *poorly scaled* model is more likely to trigger this stopping condition, even if the Use Automatic Scaling box in the Solver Options dialog is checked. If you are sure that your model is well scaled, you should consider why it is that the objective function is changing so slowly. For more information, see the discussion of "GRG Solver Stopping Conditions" below.

2. Solver cannot improve the current solution. All constraints are satisfied.

This message appears only when the GRG Solver is used, and occurs only rarely. It means that the model is *degenerate* and the Solver is probably *cycling*. One possibility worth checking is that some of your constraints are redundant, and should be removed. For more information, see the discussion of "GRG Solver Stopping Conditions" below.

3. Stop chosen when the maximum iteration limit was reached.

This message appears when (i) the Solver has completed the maximum number of iterations, or trial solutions, allowed in the Iterations box in the Solver Options dialog *and* (ii) you clicked on the Stop button when the Solver displayed the Show Trial Solution dialog. You may increase the value in the Iterations box (to a maximum of 32767) or click on Continue instead of Stop in the Show Trial Solution dialog. But you should also consider whether re-scaling your model or adding constraints might reduce the total number of iterations required.

4. The Set Target Cell values do not converge.

This message appears when the Solver is able to increase (if you are trying to Maximize) or decrease (for Minimize) without limit the value calculated by the objective or Target Cell, while still satisfying the constraints. Remember that, if you've selected Minimize, the Target Cell may take on negative values without limit unless this is prevented by constraints on the Target Calls or other cells.

If the objective is a linear function of the decision variables, it can *always* be increased or decreased without limit (picture it as a straight line), so the Solver will seek the extreme value which still satisfies the constraints. If the objective is a nonlinear function of the variables, it may have a "natural" maximum or minimum (for example, +A1*A1 has a minimum at zero), or no such limit (for example, @LOG(A1) increases without limit).

If you receive this message, you may have forgotten a constraint, or failed to anticipate values for the variables that allow the objective to increase or decrease without limit. The final values for the adjustable cells, the constraint left hand sides and the objective should provide a strong clue about what happened.

5. Solver could not find a feasible solution.

This message appears when the Solver could not find *any* combination of values for the decision variables which allows all of the constraints to be satisfied *simultaneously*. If you are using the linear Simplex Solver, and the model is well scaled, the Simplex method has determined for *certain* that there is no feasible solution. If you are using the nonlinear GRG Solver, the GRG method was unable to find a feasible solution, starting from the initial values of the variables; however it is possible that there is a feasible solution far away from these initial values, which the Solver might find if you run it with different initial values for the variables. In either case, you should first look for conflicting constraints, i.e. conditions which cannot be satisfied simultaneously. Most often this is due to choosing the wrong relation (e.g. <= instead of >=) on an otherwise appropriate constraint.

6. Solver stopped at user's request.

This message appears only if you press ESC to display the Show Trial Solution dialog, and then click on the Stop button. If you are writing a LotusScript program, the user of your program may do this, so be sure to test for this return code value (6) and take action appropriate for your application.

7. The conditions for Assume Linear Model are not satisfied.

This message appears if you have selected the linear Simplex Solver "engine," but the Solver's numeric test to ensure that the objective and constraints were indeed linear functions of the decision variables was not satisfied. (The wording of this message may be slightly different in the enhanced Solver products, which don't use the Assume Linear Model check box.) To understand exactly what is meant by a linear model, read the topic "Solver Basics."

If you receive this message, examine the formulas for the objective (Target Cell) and constraints for nonlinear or discontinuous functions or operators applied to the decision variables or adjustable cells. You can always write a linear function using only @SUM and @SUMPRODUCT.

8. The problem is too large for Solver to handle.

This message appears when the Solver determines that there are too many decision variables or constraints in your model. In many cases you will first see a message such as "Too many adjustable cells" or "Too many constraints" when you set up the model using the Solver Parameters dialog. This message appears after you click Solve and the Solver analyzes the model for cases not checked earlier.

9. Solver encountered an error value in a target or constraint cell.

This message appears when the Solver recalculates your worksheet using a new set of values for the decision variables (adjustable or changing cells), and discovers an error value (such as ERR or NA) in the cell calculating the objective (Target Cell) or one of the constraints. Inspecting the worksheet for error values like these will usually indicate the source of the problem. If you have entered formulas for the right hand sides of certain constraints, the error might have occurred in one of these formulas rather than in a cell on the worksheet. For this and other reasons, we recommend that you use only constants and cell references on the right hand sides of constraints. If you use a formula, be sure to include explicit sheet letters on any cell references, such as 1+A:C1.

Look for names or cell references to rows or columns that you have deleted, or for unanticipated values of the decision variables which lead to arguments outside the domains of your functions -- such as a negative value supplied to @SQRT. You can often add constraints to avoid such domain errors; if you have trouble with a constraint such as A1 >= 0, try a constraint such as A1 >= 0.00001 instead.

10. Stop chosen when the maximum time limit was reached.

This message appears when (i) the Solver has run for the maximum time (number of seconds) allowed in the Max Time box in the Solver Options dialog *and* (ii) you clicked on the Stop button when the Solver displayed the Show Trial Solution dialog. You may increase the value in the Max Time box (to a maximum of 32767 seconds) or click on Continue instead of Stop in the Show Trial Solution dialog. But you should also consider whether re-scaling your model or adding constraints might reduce the total solution time required.

11. There is not enough memory available to solve the problem.

This message appears when the Solver could not allocate the memory it needs to solve the problem. In addition to situations where the Solver problem itself requires too much memory, this message can appear if you have too many workbooks open in 1-2-3 or if you have too many open applications besides 1-2-3. Close these workbooks or applications and try again.

12. Another 1-2-3 instance is using SOLVE123.DLL. Try again later.

This message should appear only if you are running multiple instances of 1-2-3, and you click on Solve while another 1-2-3 instance is also running the Solver. Wait for the other 1-2-3 instance to finish solving and then try again. If this message appears under any other circumstances (most likely due to previous problems with 1-2-3), you should restart Windows and then try again.

13. Error in model. Please verify that all cells and constraints are valid.

This message means that the internal "model" (information about the adjustable cells, target cell, constraints, Solver options, etc.) which is created by the SOLVER.12A add-in and passed to SOLVE123.DLL is not in a valid form. You might receive this message if you are using the wrong version of either SOLVER.12A or SOLVE123.DLL, or if you have modified cells on the "hidden sheet" used by the Solver, either interactively or in a LotusScript or macro program.

14. Your free trial license to use the Solver has expired.

You attempted to solve a problem (by clicking the Solve button, or calling the LotusScript function SolverSolve or the macro {solver-define}) more than 100 times with the free trial version of the Solver. The trial version is limited to 100 problem solution attempts. You can upgrade your trial version to a permanent license (allowing an unlimited number of problem solution attempts) at any time: Just click the Help button in any of the Solver dialogs, read the message displayed in the initial Help dialog, then click the Register button and follow the instructions to purchase a license. The cost is $50 and you can use any of several credit cards. If you have either a modem or an Internet connection, you can complete the purchase instantly and automatically, with full security for your credit card and other information. Otherwise, you can call an 800 number 24 hours a day, 7 days a week and obtain an enablement code from the operator, or you can complete the purchase by fax or mail.

Problems with Poorly Scaled Models

A poorly scaled model is one in which the typical values of the objective and constraint functions differ by several orders of magnitude. A classic example is a financial model with some dollar amounts in millions, and other rate of return figures in percent. Poorly scaled models often cause difficulty for both linear and nonlinear Solver algorithms; the effect is often more severe for the nonlinear GRG Solver.

The Solver must perform many calculations where quantities derived from the values of the objective and constraints must be divided into and subtracted from one another. Because of the finite precision of computer arithmetic, when these calculations are performed with values of very different magnitudes, roundoff error builds up to the point where the Solver can no longer reliably find the optimal solution.

When the Use Automatic Scaling box in the Solver Options dialog is checked, the Solver will attempt to scale the values of the objective and constraint functions internally in order to minimize the effects of a poorly scaled model. ***We recommend that you check the Use Automatic Scaling box for most of your Solver problems***.

The Tolerance Option and Integer Constraints

When you solve a mixed-integer programming problem (any problem with integer constraints), the solution process is governed by the Tolerance option in the Solver Options dialog. Since the *default setting of the Tolerance option* is 0.05, the Solver stops when it has found a solution satisfying the integer constraints whose objective is within 5% of the true integer optimal solution. Therefore, you may know of or be able to discover an integer solution which is "better" than the one found by the Solver.

The reason that the default setting of the Tolerance option is 0.05 is that the solution process for integer problems -- which can take a great deal of time in any case -- often finds a near-optimal solution (sometimes *the* optimal solution) relatively quickly, and then spends far more time exhaustively checking other possibilities to find (or verify that it has found) the very best integer solution. The Tolerance option default setting is a compromise value that often saves a great deal of time, and still ensures that a solution returned by the Solver will be within 5% of the true integer optimal solution.

To ensure that the Solver finds the true integer optimal solution -- possibly at the expense of far more solution time -- set the Tolerance option to a value of zero.

Limitations on Nonlinear Problems

Nonlinear problems are intrinsically more difficult to solve than linear problems, and there are fewer guarantees about what the Solver (or *any* optimization method) can do. Whenever the "GRG Solver" choice appears in the Solver dropdown list in the Solver Parameters dialog, the GRG (Generalized Reduced Gradient) algorithm is used to solve the problem -- *even if it is actually a linear model* that could be solved by the (faster and more reliable) Simplex method. The GRG method will *usually* find the optimal solution to a linear problem -- but occasionally you will receive a Solver Completion Message indicating some uncertainty about the status of the solution -- especially if the model is poorly scaled, as discussed above. So you should always ensure that you have selected the right Solver "engine" for your problem.

When dealing with a nonlinear problem, it is a good idea to run the Solver starting from several different sets of initial values for the decision variables. Since the Solver follows a path from the starting values (guided by the direction and curvature of the objective function and constraints) to the final solution values, it will normally stop at a peak or valley closest to the starting values you supply. By starting from more than one point -- ideally chosen based on your own knowledge of the problem -- you can increase the chances that you have found the best possible "optimal solution."

If your model has certain mathematical properties, such as convexity, it is possible to make stronger guarantees about the Solver's ability to find the true optimal solution. For more information on this topic consult the books recommended in the Introduction, particularly the titles by Wayne Winston.

Related Topics:

GRG Solver Stopping Conditions

GRG Solver Stopping Conditions

It is important to understand what the nonlinear GRG Solver can and cannot do, and what each of the possible Solver Completion Messages means for this Solver "engine." At best, the GRG Solver -- like virtually all nonlinear optimization algorithms -- can find a *locally optimal* solution to a reasonably *well-scaled* model. At times, the Solver will stop *before* finding a locally optimal solution, when it is making very slow progress (the objective function is changing very little from one trial solution to another) or for other reasons.

Related Topics:

Locally Versus Globally Optimal Solutions

When Solver has Converged to the Current Solution

When Solver Cannot Improve the Current Solution

Locally Versus Globally Optimal Solutions

When the first message ("Solver found a solution") appears, it means that the GRG Solver has found a *locally optimal* solution -- there is no other set of values for the decision variables close to the current values which yields a better value for the objective. Figuratively, this means that the Solver has found a "peak" (if maximizing) or "valley" (if minimizing) -- but there may be other taller peaks or deeper valleys far away from the current solution. Mathematically, this message means that the Karush - Kuhn - Tucker (KKT) conditions for local optimality have been satisfied (to within a certain tolerance, related to the Precision setting in the Solver Options dialog).

Although there is ongoing research in globally optimal nonlinear programming methods which we are monitoring, the best that current nonlinear methods can guarantee is to find a *locally* optimal solution. We recommend that you run the GRG Solver starting from several different sets of initial values for the decision variables -- ideally chosen based on your own knowledge of the problem. In this way you can increase the chances that you have found the best possible "optimal solution."

When Solver has Converged to the Current Solution

When the GRG Solver's second stopping condition is satisfied (*before* the KKT conditions are satisfied), the second message ("Solver converged to the current solution") appears. This means that the objective function value is changing very slowly for the last few iterations or trial solutions. More precisely, the GRG Solver stops if the absolute value of the *relative* change in the objective function is less than the value in the Convergence box in the Solver Options dialog for the last 5 iterations. While the default value of 1E-4 (0.0001) is suitable for most problems, it may be too large for some models, causing the GRG Solver to stop prematurely when this test is satisfied, instead of continuing for more iterations until the KKT conditions are satisfied.

A *poorly scaled* model is more likely to trigger this stopping condition, even if the Use Automatic Scaling box in the Solver Options dialog is checked. So it pays to design your model to be reasonably well scaled in the first place: The typical values of the objective and constraints should not differ from each other, or from the decision variable values, by more than three or four orders of magnitude.

If you are getting this message when you are seeking a locally optimal solution, you can change the setting in the Convergence box to a smaller value such as 1E-5 or 1E-6; but you should also consider why it is that the objective function is changing so slowly. Perhaps you can add constraints or use different starting values for the variables, so that the Solver does not get "trapped" in a region of slow improvement.

When Solver Cannot Improve the Current Solution

The third stopping condition, which yields the message "Solver cannot improve the current solution," occurs only rarely. It means that the model is *degenerate* and the Solver is probably *cycling*. The issues involved are beyond the level of this User Guide, as well as most of the books recommended in the Introduction. One possibility worth checking is that some of your constraints are redundant, and should be removed. If this suggestion doesn't help and you cannot reformulate the problem, you will probably need specialized consulting assistance.

Problems with Discontinuous Functions

A *discontinuous* function is one whose graph shows sudden "breaks" instead of a smooth (straight linear, or curved nonlinear) line. The most common example is the IF function. For example:

@IF(A1>10,B1,2*B1)

is discontinuous around A1=10 because its value "jumps" from whatever value B1 has to twice that value. A model with a discontinuous function does *not* meet the conditions required by either the linear or the nonlinear solver in 1-2-3 97 Edition.

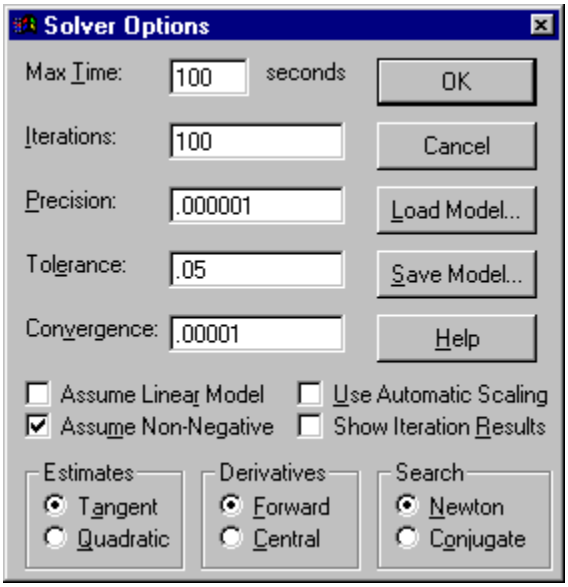
A nonlinear solver relies on partial derivative information to guide it towards a feasible and optimal solution; since it is unable to compute the partial derivatives of a function at points where that function is discontinuous, it cannot guarantee that any solution it finds is truly optimal. In practice, the nonlinear GRG algorithm included with the standard Solver can sometimes deal with discontinuities which are "incidental" to the problem, but as a general statement, the Solver cannot handle problems where the objective function or some of the constraints are discontinuous.

If you try to solve a problem with discontinuous functions with the linear Simplex Solver, it is possible -- though very unlikely -- that the linearity test performed by the Solver will not detect the discontinuities and will proceed to try to solve the problem. (This probably means that the functions *were* linear over the range considered by the linearity test -- but there are no guarantees at all that the solution found is optimal!)

You *can* use discontinuous functions such as @IF and @CHOOSE in calculations on the worksheet which are *not dependent on the decision variables*, and are therefore constant in the optimization problem. But any discontinuous functions that do depend on the variables will likely cause problems for the Solver. Users sometimes fail to realize that certain functions, such as @ABS and @ROUND, are discontinuous at certain points. For more information on this subject, read the section "Functions to Avoid: Discontinuous Functions" in the topic "Solver Basics."

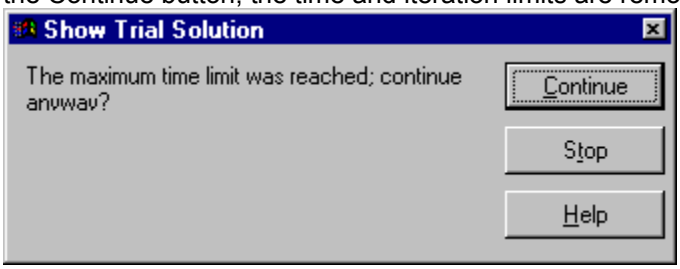
Introduction

This topic describes the options available in the Solver Options dialog for the standard Solver. Bear in mind that the options which control numerical tolerances and solution strategies are pre-set to the choices which are applicable to the great majority of problems; you should only change these settings in exceptional circumstances. The options you will use most often are common to all the Solver products, and control features like the display of iteration results or the upper limits on solution time or Solver iterations.

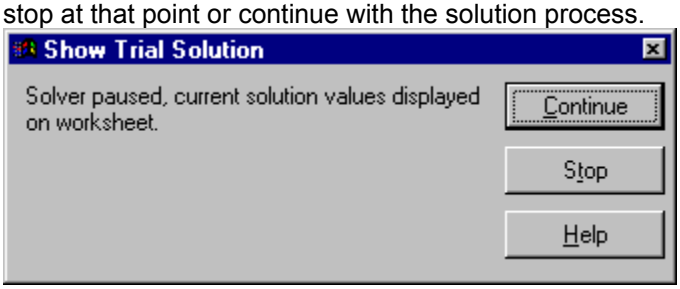


Max Time and Iterations

The Max Time and Iterations settings determine how much time, or computing effort, the Solver will expend on a problem before asking you whether you want to stop. You may need to increase these settings from their default values, in order to solve larger problems. Bear in mind that if the maximum time or maximum number of iterations is exceeded, the Solver stops and displays a dialog like the one below: You will have the option to stop at that point or to continue the solution process. If you click on the Continue button, the time and iteration limits are removed, and you will not be prompted again.



There is really no downside to specifying quite a large value for the Max Time or Iterations setting: If you ever get impatient, you can simply press the ESC key while the Solver is running. If your model is large enough to take some time to recalculate even once, you should hold down the ESC key for a second or two. After a momentary delay, the dialog box shown below will appear, and you will have the option to stop at that point or continue with the solution process.



Precision

The number entered here determines how closely the calculated values of the constraint left hand sides must match the right hand sides in order for the constraint to be satisfied. Recall from Elements of Solver Models that a constraint is satisfied if the relation it represents is true *within a small tolerance*; the Precision value is that tolerance. With the default setting of 1.0E-6 (0.000001), a calculated left hand side of -1.0E-7 would satisfy a contraint such as A1 >= 0.

Related Topics:

Precision and Regular Constraints

Precision and Integer Constraints

Precision and Regular Constraints

Use caution in making this number much smaller, since the finite precision of computer arithmetic virtually ensures that the values calculated by 1-2-3 and the Solver will differ from the expected or "true" values by a small amount. On the other hand, setting the Precision to a much larger value would cause constraints to be satisfied too easily. If your constraints are not being satisfied because the values you are calculating are in units such as millions of dollars, consider checking the Use Automatic Scaling box instead of altering the Precision setting.

Precision and Integer Constraints

Another use of Precision is determining whether an integer constraint, such as A1 = integer or A1 = binary, is satisfied. If the difference between the decision variable's value and the closest integer value is less than the Precision, the variable value is treated as an integer.

Integer Tolerance

This number affects only the solution process for integer programming problems; it has no impact on "regular" optimization problems. When you solve an integer programming problem, it often happens that the Branch & Bound method will find a good solution fairly quickly, but will require a great deal of computation time to find (or verify that it has found) the optimal integer solution. The Integer Tolerance setting may be used to tell the Solver to stop if the best solution it has found so far is "close enough."

The Branch & Bound process starts by finding the optimal solution without considering the integer constraints (this is called the *relaxation* of the integer programming problem). The objective value of the relaxation forms the initial "best bound" on the objective of the optimal *integer* solution, which can be no better than this. During the optimization process, the Branch & Bound method finds "candidate" integer solutions, and it keeps the best solution so far as the "incumbent." By eliminating alternatives as its proceeds, the B&B method also tightens the "best bound" on how good the integer solution can be.

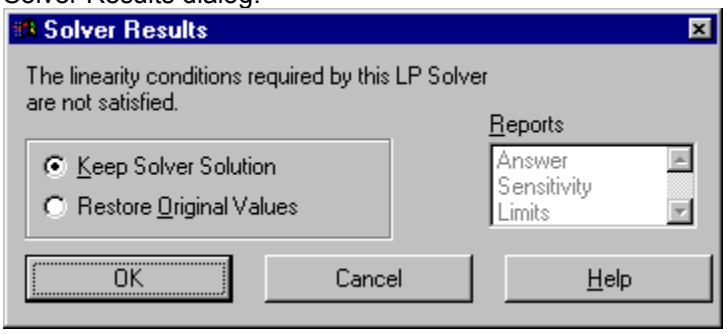
Each time the Solver finds a new "incumbent" -- an improved all-integer solution -- it computes the maximum percentage difference between the objective of this solution and the current "best bound" on the objective:

$$\frac{\text{Objective of incumbent} - \text{Objective of best bound}}{\text{Objective of best bound}}$$

If the absolute value of this maximum percentage difference is equal to or less than the Integer Tolerance, the Solver will stop and report the current integer solution as the optimal result. If you set the Integer Tolerance to zero, the Solver will continue searching until all alternatives have been explored and the optimal integer solution has been found. This may take a great deal of computing time.

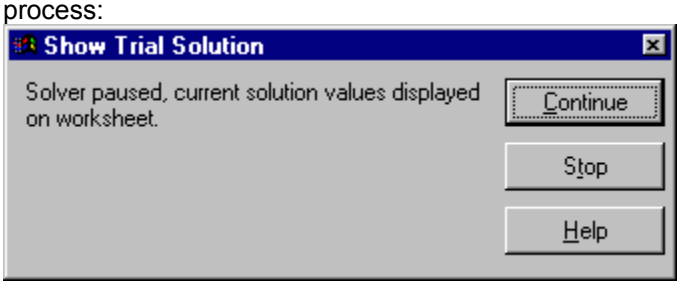
Assume Linear Model

In the standard Solver, the Assume Linear Model check box controls the choice of the linear Simplex Solver or the nonlinear GRG Solver: When it is checked, the Simplex Solver is used; otherwise the GRG Solver is used. The box is labeled "Assume Linear Model" because the Solver initially *assumes* that your model is made up entirely of *linear* functions for the objective and constraints. It solves the problem using the Simplex Solver, and then *checks the solution* by comparing the values for the objective and constraints calculated from the Simplex method with the values for these cells obtained when the solution values are placed in the decision variable cells and the spreadsheet is recalculated. If these values do not agree to within a close tolerance, the Solver displays the Completion Message shown below in the Solver Results dialog:



Show Iteration Results

When this box is checked, a dialog like the one below will appear on every iteration during the solution process:



This is the same dialog which appears when you press ESC at any time during the solution process, but when the Show Iteration Results box is checked it appears automatically on every iteration. When this dialog appears, the best values so far for the decision variables appear on the spreadsheet, which is recalculated to show the values of the objective function and the constraints. You may click on the Continue button to go on with the solution process, or on the Stop button to stop immediately.

Use Automatic Scaling

When this box is checked, the Solver will attempt to scale the values of the objective and constraint functions internally in order to minimize the effects of a poorly scaled model. A *poorly scaled model* is one in which the typical values of the objective and constraint functions differ by several orders of magnitude. A classic example is a financial model with some dollar amounts in millions, and other rate of return figures in percent. Poorly scaled models may cause difficulty in the solution process, again due to the effects of finite precision computer arithmetic.

Poorly scaled models are one of the most common causes of problems in which the Solver appears to stop prematurely without reaching the true optimal solution; it is a good idea to keep this box checked for all of your Solver models.

If your model is nonlinear and you do check the Use Automatic Scaling box, *make sure that the initial values for the decision variables are "reasonable,"* i.e. of roughly the same magnitudes that you expect for those variables at the optimal solution. The effectiveness of the Automatic Scaling option depends on how well these starting values reflect the values encountered during the solution process.

Assume Non-Negative

When this box is checked, any decision variables without explicit lower bounds (>= constraints) in the Constraints list box of the Solver Parameters dialog will be given a lower bound of zero when the problem is solved. This option has no effect for decision variables which *do* have explicit >= constraints, even if those constraints allow the variables to assume negative values. For example, a constraint A1 >= -3 specifies that A1 can assume values from -3 to 0, even if the Assume Non-Negative box is checked.

GRG Solver Options

These options affect only the nonlinear GRG Solver; they are not used by the linear Simplex Solver. The default choices are suitable for the vast majority of problems; although it generally won't hurt to change these options, you should first consider other alternatives such as improved scaling before attempting to fine-tune these options. In some scientific and engineering applications, alternative choices may improve the solution process.

Related Topics:

Convergence

Other Nonlinear Options

Estimates

Derivatives

Search

Convergence

As discussed under "GRG Solver Stopping Conditions" GRG Solver Stopping Conditions, the GRG Solver will stop and display the message "Solver converged to the current solution" when the objective function value is changing very slowly for the last few iterations or trial solutions. More precisely, the GRG Solver stops if the absolute value of the *relative* change in the objective function is less than the value in the Convergence edit box for the last 5 iterations. While the default value of 1E-4 (0.0001) is suitable for most problems, it may be too large for some models, causing the GRG Solver to stop prematurely when this test is satisfied, instead of continuing for more iterations until the optimality conditions are satisfied.

If you are getting this message when you are seeking a locally optimal solution, you can change the setting in the Convergence box to a smaller value such as 1E-5 or 1E-6; but you should also consider why it is that the objective function is changing so slowly. Perhaps you can add constraints or use different starting values for the variables, so that the Solver does not get "trapped" in a region of slow improvement.

Other Nonlinear Options

The following general background may assist you in understanding how the Estimates, Derivatives and Search options are used. For more information, consult the academic papers on the GRG method listed under "Further Reading."

On each major iteration, the GRG Solver must compute values for the first partial derivatives of the objective and constraints. The Derivatives option is concerned with how these partial derivatives are computed.

The GRG (Generalized Reduced Gradient) solution process proceeds by first "reducing" the problem to an unconstrained optimization problem, by solving a set of nonlinear equations for certain variables (the "basic" variables) in terms of others (the "nonbasic" variables). Then a search direction (a vector in N-space, where N is the number of nonbasic variables) is chosen along which an improvement in the objective function will be sought. The Search option is concerned with how this search direction is determined.

Once a search direction is chosen, a one-dimensional "line search" is carried out along that direction, varying a step size in an effort to improve the reduced objective. The initial estimates for values of the variables which are being varied has a significant impact on the effectiveness of the search. The Estimates option is concerned with how these estimates are obtained.

Estimates

This option determines the approach used to obtain initial estimates of the basic variable values at the outset of each one-dimensional search. The Tangent choice uses linear extrapolation from the line tangent to the reduced objective function. The Quadratic choice extrapolates the minimum (or maximum) of a quadratic fitted to the function at its current point. If the current reduced objective is well modeled by a quadratic, then the Quadratic option can save time by choosing a better initial point, which requires fewer subsequent steps in each line search. If you have no special information about the behavior of this function, the Tangent choice is "slower but surer." **Note:** the Quadratic choice here has no bearing on quadratic programming problems.

Derivatives

The first partial derivatives of the problem functions are computed through "finite differencing," which involves perturbing the current values of the decision variables, observing how the problem functions change, and performing a "rise over run" calculation. With Forward differencing (the default choice), the point from the previous iteration (where the problem function values are already known) is used in conjunction with the current point. This reduces the recalculation time required for finite differencing, which can account for up to half of the total solution time. Central differencing relies only on the current point and perturbs the decision variables in opposite directions from that point. Although this involves more recalculation time, it may result in a better choice of search direction when the derivatives are rapidly changing, and hence fewer total iterations.

Search

It would be far too expensive to determine a search direction using the pure form of Newton's method, by computing the Hessian matrix of *second* partial derivatives of the problem functions. (This would roughly square the number of spreadsheet recalculations required to solve the problem.) Instead, an direction is chosen through an estimation method. The default choice Newton uses a quasi-Newton (or BFGS) method, which maintains an *approximation* to the Hessian matrix; this requires more storage (an amount proportional to the square of the number of currently binding constraints) but performs very well in practice. The alternative choice Conjugate uses a conjugate gradient method, which does not require storage for the Hessian matrix and still performs well in most cases. The choice you make here is not crucial, since the GRG solver is capable of switching *automatically* between the quasi-Newton and conjugate gradient methods depending on the available storage.

Introduction

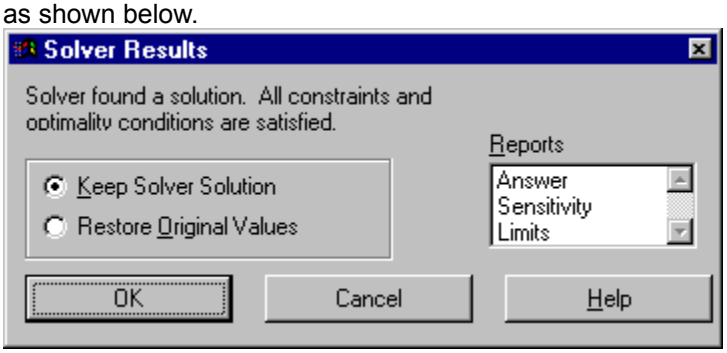
This topic will help you use the information in the Solver Reports, which can be produced whenever the Solver finds a solution. We'll explain how to interpret the values in the Answer, Sensitivity and Limits reports, using as an example the Product Mix problem, which appears as sheet tab B in the workbook SOLVSAMP.123.

All three types of reports can be useful, but we recommend that you focus on the Sensitivity Report. When properly interpreted, this report will tell you a great deal about your model and its optimal solution, that you could not easily determine by simply inspecting the final solution values on the worksheet. Using the Sensitivity report, you can determine what would happen if you changed your model in various ways and re-ran the Solver, without your having to actually carry out these steps.

All of the reports are *1-2-3 worksheets*, with grid lines turned off. You can turn the grid lines back on, if you wish, through the **Sheet Properties...** menu option (using the View tab in the dialog box). Because the reports are worksheets, you can copy and edit the report information, perform calculations on the numbers in the reports, or create graphs directly from the report data. This makes the 1-2-3 97 Solver's reports considerably more useful than those produced by standalone optimization software packages.

Selecting the Reports

When the Solver finds the solution to an optimization problem, or when the solution process is terminated prematurely due to some error condition (or your own intervention), the Solver Results dialog is displayed, as shown below.



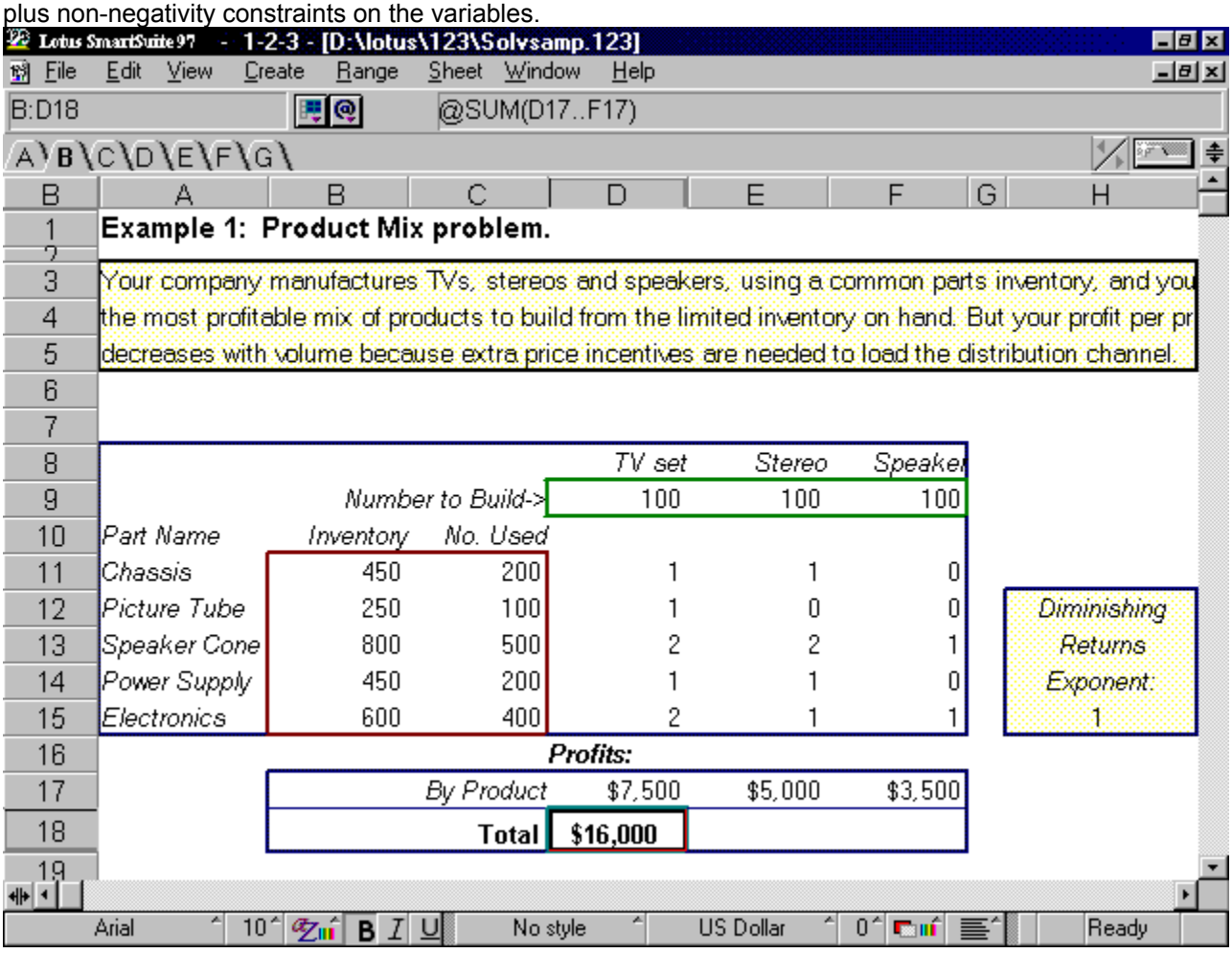
If the solution process was terminated prematurely, the Reports list box in the dialog above will be greyed, and you will not be able to select any reports; you will need to re-solve and obtain an optimal solution in order to produce the reports.

When the Reports list box is available, you can select one, two, or all three reports. Simply click on the report names to select the reports you want, or press Alt-R and then down-arrow from the keyboard. In this setting, you don't need to hold down the SHIFT or CTRL key while you click on the report names with the mouse.

Once the reports are selected, you can choose one of the options "Keep Solver Solution" or "Restore Original Values." When you click on OK, the reports will be generated. Clicking on Cancel instead will cancel generation of the reports. The reports are 1-2-3 worksheets which are inserted in the current workbook, just after the sheet containing the Solver model.

An Example Model

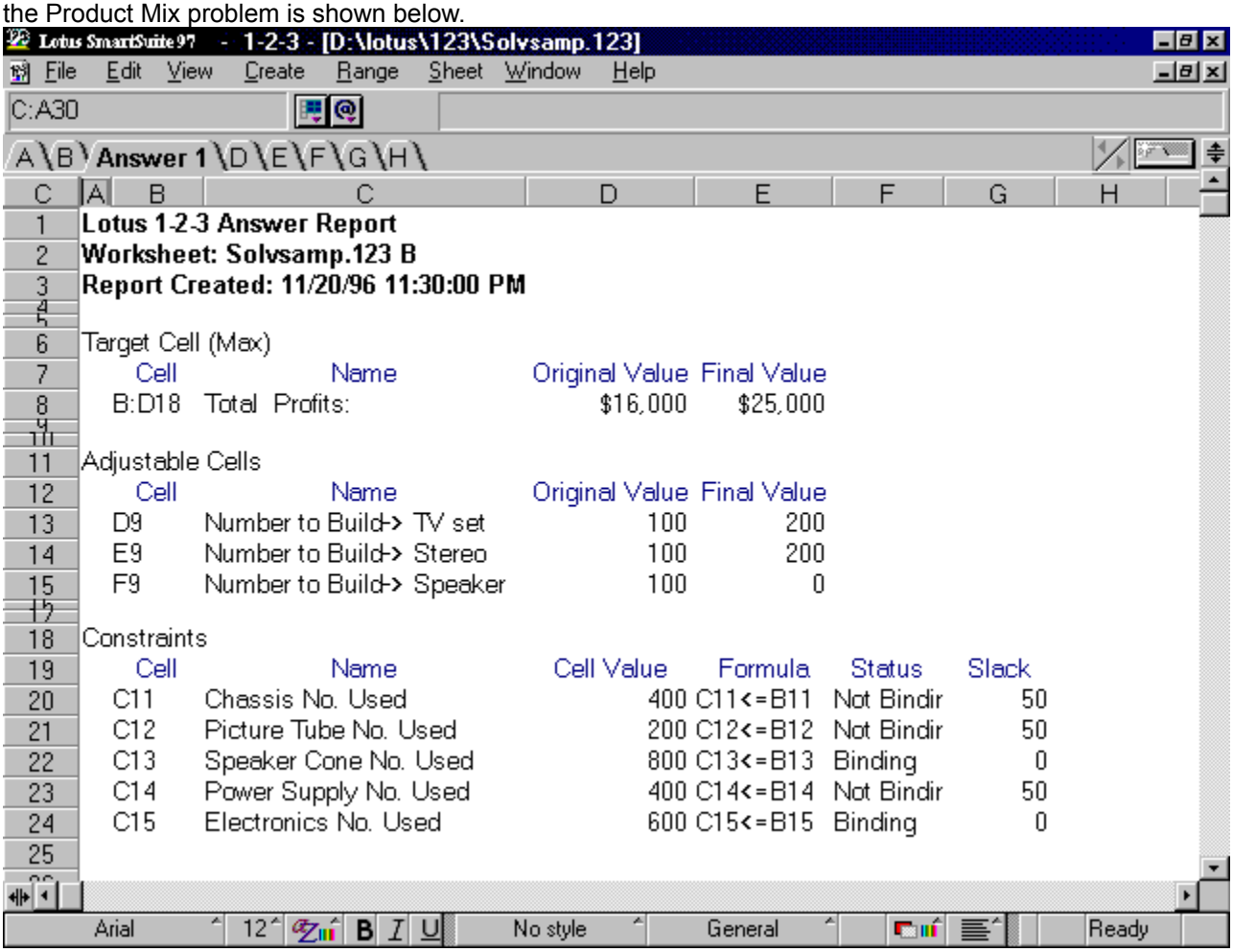
To illustrate the Solver Reports, we will solve the Product Mix problem on sheet tab B of SOLVSAMP.123. This model has three decision variables at D9..F9 and five constraints of the form C11..C15 <= B11..B15, plus non-negativity constraints on the variables.



In brief, the Answer Report summarizes the original and final values of the decision variables and constraints, with information about which constraints are "binding" at the solution. The Sensitivity Report provides information about how the solution would change for small changes in the constraints or the objective function. And the Limits Report shows you the largest and smallest value each decision variable can assume and still satisfy the constraints, while all other variables are held fixed at their solution values.

The Answer Report

The Answer Report provides basic information about the decision variables and the constraints, with their original and final values. It also gives you a quick way to determine which constraints are "binding" or satisfied with equality at the solution, and which constraints have slack. An example Answer Report for the Product Mix problem is shown below.



Lotus SmartSuite 97 - 1-2-3 - [D:\lotus\123\Solvsamp.123]

File Edit View Create Range Sheet Window Help

C:A30

A B **Answer 1** D E F G H

	A	B	C	D	E	F	G	H
1	**Lotus 1-2-3 Answer Report**							
2	**Worksheet: Solvsamp.123 B**							
3	**Report Created: 11/20/96 11:30:00 PM**							
6	Target Cell (Max)							
7		Cell	Name	Original Value	Final Value			
8		B:D18	Total Profits:	$16,000	$25,000			
11	Adjustable Cells							
12		Cell	Name	Original Value	Final Value			
13		D9	Number to Build-> TV set	100	200			
14		E9	Number to Build-> Stereo	100	200			
15		F9	Number to Build-> Speaker	100	0			
18	Constraints							
19		Cell	Name	Cell Value	Formula	Status	Slack	
20		C11	Chassis No. Used	400	C11<=B11	Not Bindir	50	
21		C12	Picture Tube No. Used	200	C12<=B12	Not Bindir	50	
22		C13	Speaker Cone No. Used	800	C13<=B13	Binding	0	
23		C14	Power Supply No. Used	400	C14<=B14	Not Bindir	50	
24		C15	Electronics No. Used	600	C15<=B15	Binding	0	
25								

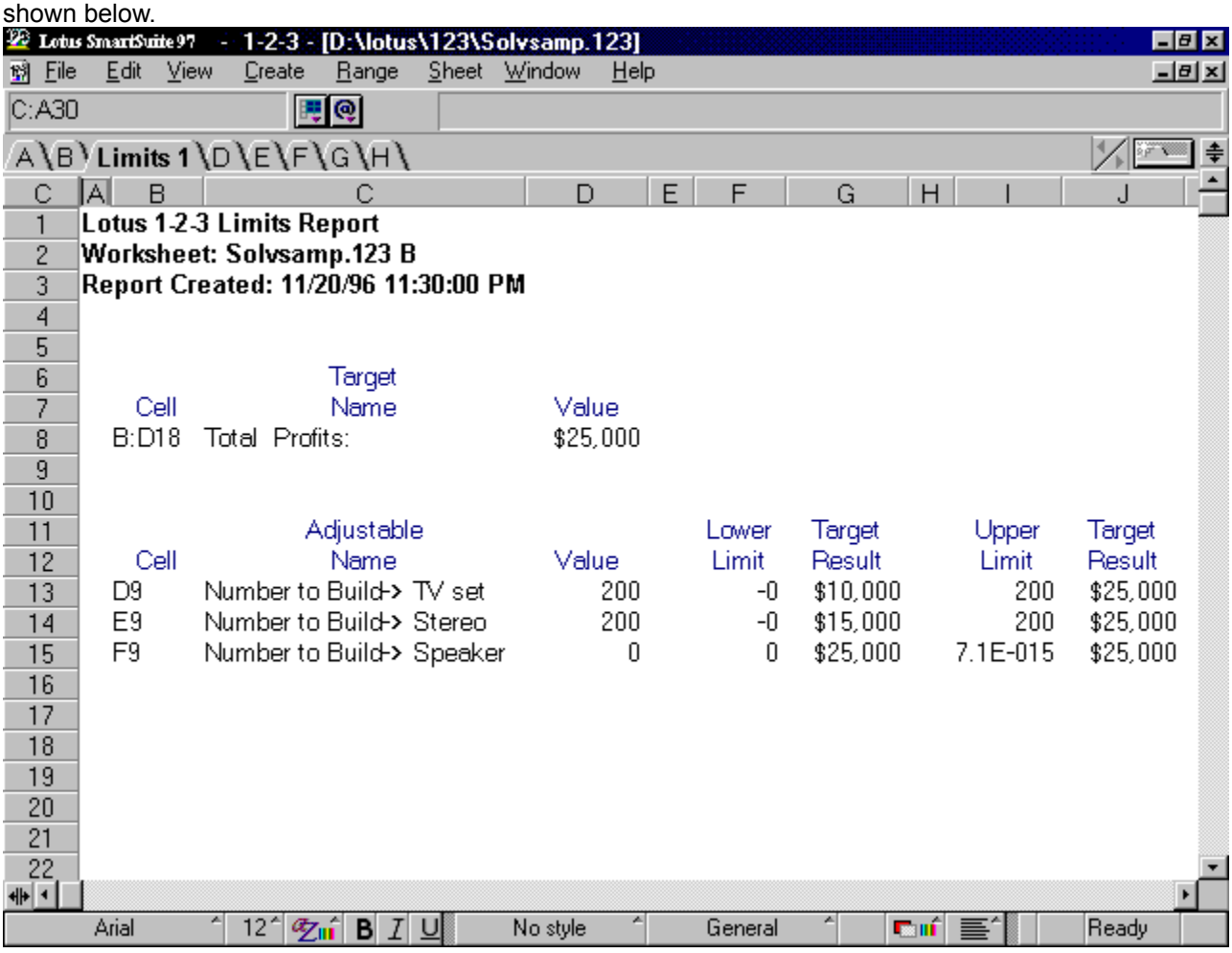
Arial 12 B I U No style General Ready

First shown are the objective function (target cell) and decision variables (adjustable cells), with their original value and final values. Next are the constraints, with their final cell values; a formula representing the constraint; a "status" column showing whether the constraint was binding or non-binding at the solution; and the *slack* value -- the difference between the final value and the lower or upper bound imposed by that constraint. A binding constraint, which is satisfied with equality, will always have a slack of zero.

Note: When creating a report, the Solver constructs the entries in the Name column by searching for the first text cell to the left and the first text cell above each adjustable (changing) cell and each constraint cell. If you lay out your Solver model in tabular form, with text labels in the leftmost column and topmost row, these entries will be most the most useful -- as in the example above. Also note that the *formatting* for the Original Value, Final Value and Cell Value is "inherited" from the formatting of the corresponding cell in the Solver model.

The Limits Report

The Limits Report was designed to provide a different kind of "sensitivity analysis" information. It is created by re-running the optimization model with each decision variable (or changing cell) in turn as the objective (both maximizing and minimizing), and all other variables held fixed. Hence, it shows a "lower limit" for each variable, which is the smallest value that a variable can take while satisfying the constraints and holding all of the other variables constant, and an "upper limit," which is the largest value the variable can take under these circumstances. An example of the Limits Report for the Product Mix problem is shown below.



The Sensitivity Report

The Sensitivity Report provides classical sensitivity analysis information for both linear and nonlinear programming problems, including dual values (in both cases) and range information (for linear problems only). The dual values for (nonbasic) variables are called Reduced Costs in the case of linear programming problems, and Reduced Gradients for nonlinear problems. The dual values for (binding) constraints are called Shadow Prices for linear programming problems, and Lagrange Multipliers for nonlinear problems.

Constraints which are simple *upper and lower bounds* on the variables, that you enter in the Constraints list box of the Solver Parameters dialog, are handled specially (for efficiency reasons) by both the linear and nonlinear Solver algorithms, and will *not* appear in the Constraints section of the Sensitivity report. When an upper or lower bound on a variable is *binding* at the solution, a nonzero Reduced Cost or Reduced Gradient for that variable will appear in the "Adjustable Cells" section of the report; this is normally the same as a Lagrange Multiplier or Shadow Price for the upper or lower bound.

Note: The *formatting* of cells in the Sensitivity Report is "inherited" from the corresponding cells in the Solver model. This can make a significant difference in how the Reduced Gradient, Lagrange Multiplier, Reduced Cost and Shadow Prices are displayed. In the example Sensitivity Reports to follow, the various cells in the Product Mix model were formatted to display as integers (0 decimal places), so the entries in the report are formatted the same way. If you select the cell displayed as -2 below (the Reduced Gradient for Speakers), you'll see that the actual value is -2.4999... which has been rounded *down* to -2. Bear this in mind when designing your model and when reading the report. Since the report is a *worksheet*, you can always change the cell formatting.

Related Topics:

Interpreting Dual Values

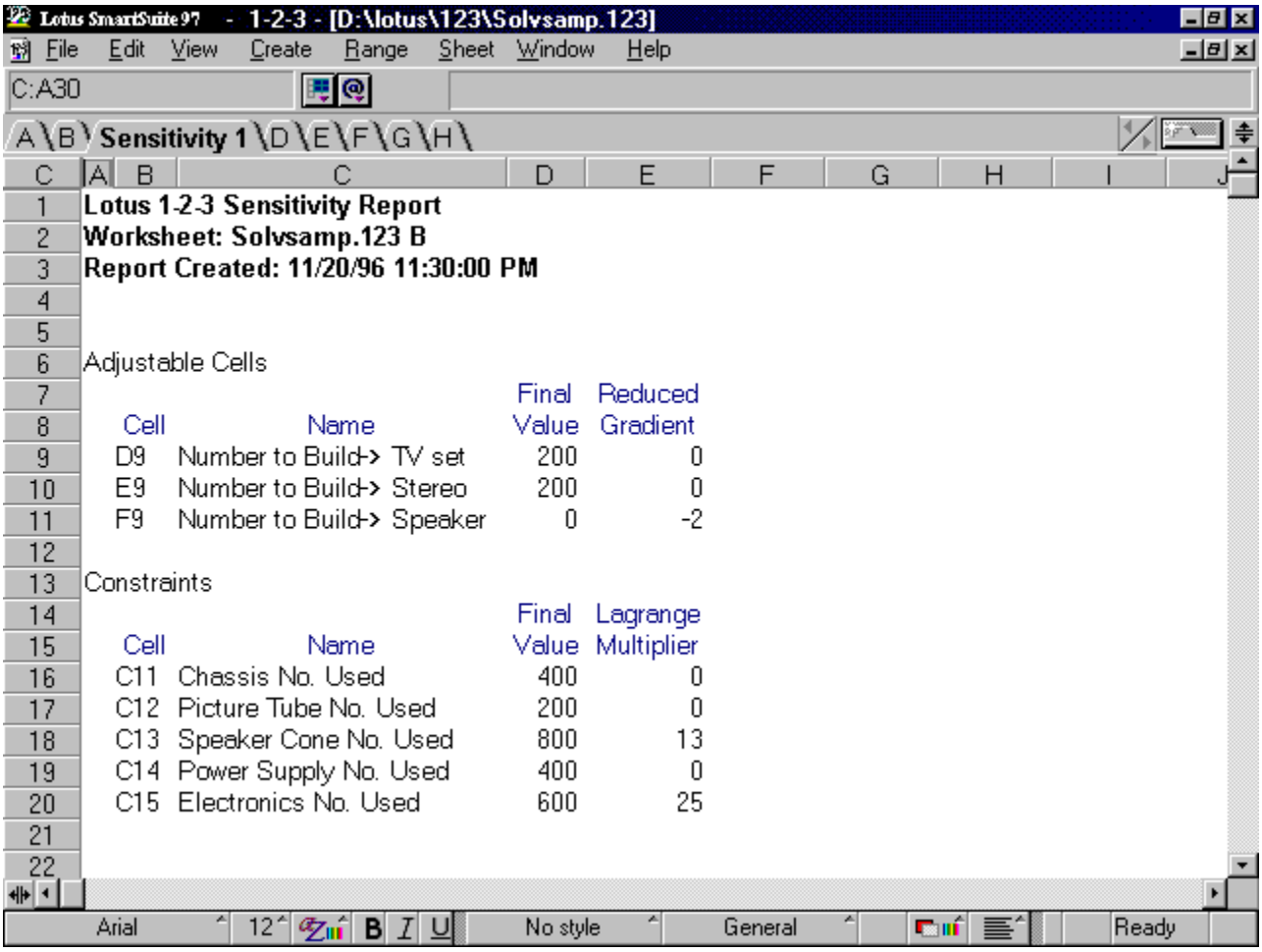
Interpreting Range Information

Interpreting Dual Values

Dual values are the most basic form of sensitivity analysis information. The dual value for a variable is nonzero only when the variable's value is equal to its upper or lower bound (usually zero) at the optimal solution. This is called a *nonbasic* variable, and its value was driven to the bound during the optimization process. Moving the variable's value away from the bound will *worsen* the objective function's value; conversely, "loosening" the bound will *improve* the objective. The dual value measures the increase in the objective function's value *per unit increase* in the variable's value. In the example Sensitivity Report below, the dual value for producing speakers is -2.499, meaning that if we were to build one speaker (and therefore less of something else), our total profit would decrease by $2.50.

The dual value for a constraint is nonzero only when the constraint is equal to its bound. This is called a *binding* constraint, and its value was driven to the bound during the optimization process. Moving the constraint left hand side's value away from the bound will *worsen* the objective function's value; conversely, "loosening" the bound will *improve* the objective. The dual value measures the increase in the objective function's value *per unit increase* in the constraint's bound. In the example Sensitivity Report below, increasing the number of electronics units from 600 to 601 will allow the Solver to increase total profit by $25.

An example of a Sensitivity Report generated for the Product Mix problem when the Solver "engine" is the nonlinear GRG solver is shown below. Note that it includes only the solution values and the dual values: Reduced Gradients for variables and Lagrange Multipliers for constraints. If you solve a *quadratic programming* problem with the LP/Quadratic "engine" in either the Quadratic or Large-Scale LP Solvers, the report will also appear in this format.



If you are not accustomed to analyzing sensitivity information for nonlinear problems, you should bear in mind that the dual values are valid only at the single point of the optimal solution -- if there is any curvature involved, the dual values begin to change (along with the constraint gradients) as soon as you move away from the optimal solution. In the case of linear problems, the dual values remain constant over the range of Allowable Increases and Decreases in the variables' objective coefficients and the constraints' right hand sides, respectively.

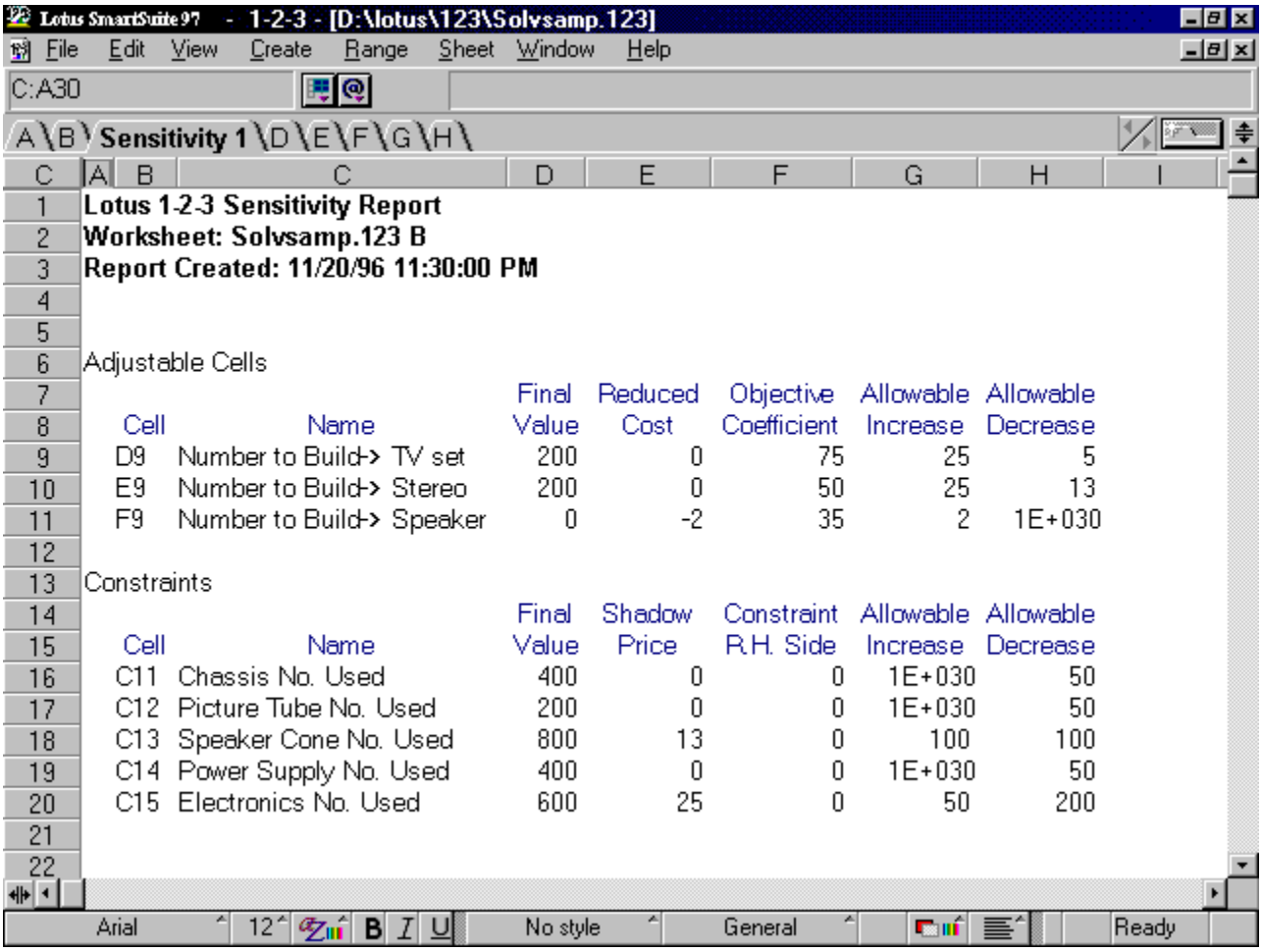
Interpreting Range Information

In linear programming problems, unlike nonlinear problems, the dual values are *constant* over a range of possible changes in the objective function coefficients and the constraint right hand sides. The Sensitivity Report for linear programming models includes this range information.

A Sensitivity Report for the Product Mix problem when the Solver "engine" is the standard Simplex or the Large-Scale LP Solver is shown below. In addition to the dual values (Reduced Costs for variables, Shadow Prices for constraints), this report provides information about the range over which these values will remain valid.

For each decision variable, the report shows its coefficient in the objective function, and the amount by which this coefficient could be increased or decreased without changing the dual value. In the example below, we'd still build 200 TV sets even if the profitability of TV sets decreased up to $5 per unit. Beyond that point, or if the unit profit of speakers increased by more than $2.50 -- *rounded* below for display purposes to $3 -- we'd start building speakers.

For each constraint, the report shows the constraint right hand side, and the amount by which the RHS could be increased or decreased without changing the dual value. In this example, we could use up to 50 more electronics units, which we'd use to build more TV sets instead of stereos, increasing our profits by $25 per unit. Beyond 650 units, we would switch to building speakers at an incremental profit of $20 per unit (a new dual value). A value of 1.00E+30 in these reports represents "infinity:" in the example below, we wouldn't build any speakers regardless of how much the profit per speaker were *decreased*.


Lotus SmartSuite 97 - 1-2-3 - [D:\lotus\123\Solvsamp.123]
File Edit View Create Range Sheet Window Help
C:A30
A B Sensitivity 1 D E F G H
Lotus 1-2-3 Sensitivity Report
Worksheet: Solvsamp.123 B
Report Created: 11/20/96 11:30:00 PM
Adjustable Cells
Cell Name Final Value Reduced Cost Objective Coefficient Allowable Increase Allowable Decrease
D9 Number to Build-> TV set 200 0 75 25 5
E9 Number to Build-> Stereo 200 0 50 25 13
F9 Number to Build-> Speaker 0 -2 35 2 1E+030
Constraints
Cell Name Final Value Shadow Price Constraint R.H. Side Allowable Increase Allowable Decrease
C11 Chassis No. Used 400 0 0 1E+030 50
C12 Picture Tube No. Used 200 0 0 1E+030 50
C13 Speaker Cone No. Used 800 13 0 100 100
C14 Power Supply No. Used 400 0 0 1E+030 50
C15 Electronics No. Used 600 25 0 50 200
Arial 12 No style General Ready

Controlling the Solver's Operation

You can control every aspect of the Solver's operation programmatically. You can display or completely hide the Solver dialog boxes, create or modify the choices of objective (target cell), variables (changing cells) and constraints, check whether an optimal solution was found and produce reports. You do this by calling a set of Solver-specific functions from a program you write in LotusScript for 1-2-3 97 Edition.

Controlling the Solver can be as simple as *adding one line* to your LotusScript code! Each worksheet in a workbook may have a Solver problem defined, which is saved automatically with the workbook. You can create this Solver model interactively if you wish. If you distribute such a workbook, with a worksheet containing a Solver model and a LotusScript program, all you need to do in your code is activate the worksheet and call the function **SolverSolve**.

***Note*:** In order to use any of the Solver-specific functions, you must include a LotusScript **Use** statement referencing the Solver add-in in your Global Declarations section. For example:

```
Use "c:\lotus\123\solver\solver.12a"
```

LotusScript Function Reference

SolverAdd (Form 1)

Equivalent to choosing **Range Analyze Solver...** and pressing the Add button in the Solver Parameters dialog box. Adds a constraint to the current problem.

Syntax

SolverAdd(CellRef, Relation, FormulaText)

CellRef is a reference to a cell or a range of cells on the active worksheet and forms the left hand side of the constraint.

Relation specifies the arithmetic relationship between the left and right sides, or whether **CellRef** must have an integer value at the solution.

Relation	Relationship
1	<=
2	=
3	>=
4	Int (CellRef is an integer variable)
5	Bin (CellRef is a binary integer variable)

FormulaText is the right hand side of the constraint and will often be a single number, but it may be a formula or a reference to a range of cells. If it is a formula, any cell references should have explicit sheet letters, such as "1+A:C1".

If **Relation** is 4 or 5, **FormulaText** is ignored, and **CellRef** must be a subset of the By Changing Cells.

If **FormulaText** is a reference to a range of cells, the number of cells in the range must match the number of cells in **CellRef**, although the shape of the areas need not be the same. For example, **CellRef** could be a row and **FormulaText** could refer to a column, as long as the number of cells is the same.

Remarks

The SolverAdd, SolverChange and SolverDelete functions correspond to the Add, Change, and Delete buttons in the Solver Parameters dialog box. You use these functions to define constraints. For many macro applications, however, you may find it more convenient to load the problem in a single step using the SolverLoad function.

Each constraint is uniquely identified by the combination of the cell reference on the left and the relationship (<=, =, >=, int or bin) between its left and right sides. This takes the place of selecting the appropriate constraint in the Solver Parameters dialog box. You can manipulate constraints with SolverChange and SolverDelete.

SolverAdd (Form 2)

Equivalent to choosing **Range Analyze Solver...**, pressing the Variables button, and then pressing the Add button in the Solver Parameters dialog box. Adds a set of decision variable cells to the current problem.

Syntax

SolverAdd(CellRef, 0, 0)

CellRef is a reference to a cell or a range of cells on the active worksheet and forms a set of decision variables.

Remarks

The SolverAdd, SolverChange and SolverDelete functions correspond to the Add, Change, and Delete buttons in the Solver Parameters dialog box. In this form, you can use these functions to add or change sets of decision variables. For many macro applications, however, you may find it more convenient to load the problem in a single step using the SolverLoad function.

Note that SolverOk defines the *first* entry in the By Changing Variable Cells list box. Use SolverAdd to define additional entries in the Variables Cells list box. Do *not* call SolverOk with a different **ByChange** argument *after* you have defined more than one set of variable cells.

SolverChange (Form 1)

Equivalent to choosing **Range Analyze Solver...** and pressing the Change button in the Solver Parameters dialog box. Changes the right hand side of an existing constraint.

Syntax

SolverChange(CellRef, Relation, FormulaText)

For an explanation of the arguments and constraints, see SolverAdd.

Remarks

If the combination of **CellRef** and **Relation** does not match any existing constraint, the function returns the value 4 and no action is taken.

To change the **CellRef** or **Relation** of an existing constraint, use SolverDelete to delete the old constraint and then use SolverAdd to add the constraint in the form you want.

SolverChange (Form 2)

Equivalent to choosing **Range Analyze Solver...**, pressing the Variables button, and then pressing the Change button in the Solver Parameters dialog box. Changes a set of decision variable cells.

Syntax

SolverChange(CellRef, Relation, 0)

CellRef is a reference to a cell or a range of cells on the active worksheet, currently defined in the By Changing Variable Cells list box as variable cells.

Relation is a reference to a different cell or range of cells on the active worksheet, which will replace **CellRef** as a new set of variable cells.

Remarks

If **CellRef** does not match any existing set of variable cells, the function returns the value 1 and no action is taken.

SolverDelete (Form 1)

Equivalent to choosing **Range Analyze Solver...** and pressing the Delete button in the Solver Parameters dialog box. Deletes an existing constraint.

Syntax

SolverDelete(CellRef, Relation, FormulaText)

For an explanation of the arguments and constraints, see SolverAdd.

Remarks

If the combination of **CellRef** and **Relation** does not match any existing constraint, the function returns the value 4 and no action is taken. If the constraint is found, it is deleted, and the function returns the value 0.

SolverDelete (Form 2)

Equivalent to choosing **Range Analyze Solver...**, pressing the Variables button, and then pressing the Delete button in the Solver Parameters dialog box. Deletes an existing set of variable cells.

Syntax

SolverDelete(CellRef, 0, 0)

CellRef is a reference to a cell or a range of cells on the active worksheet, currently defined in the By Changing Variable Cells list box as variable cells.

Remarks

If **CellRef** does not match any existing set of variable cells, the function returns the value 1 and no action is taken. If the variable cells are found, they are deleted, and the function returns the value 0.

SolverFinish

Equivalent to making selections and clicking OK in the Solver Results dialog box that appears when the solution process is finished. The dialog box will not be displayed.

Syntax

SolverFinish(KeepFinal, ReportArray)

KeepFinal is the number 1 or 2 and specifies whether to keep or discard the final solution. If **KeepFinal** is 1 or omitted, the final solution values are kept in the changing cells. If **KeepFinal** is 2, the final solution values are discarded and the former values of the changing cells are restored.

ReportArray is an array argument specifying what reports to create when Solver is finished.

If ReportArray is	1-2-3 creates
1	An Answer Report
2	A Sensitivity Report
3	A Limits Report

A combination of these values produces multiple reports. For example, if **ReportArray**(1) = 1 and **ReportArray**(2) = 3, 1-2-3 creates an Answer Report and a Limits Report.

SolverFinishDialog

Equivalent to making selections in the Solver Results dialog box that appears when the solution process is finished. The dialog box will be displayed, and the user will be able to change the options you initially specify.

Syntax

SolverFinishDialog(KeepFinal, ReportArray)

For an explanation of the arguments, see SolverFinish.

SolverGet

Returns information about the current Solver problem. The settings are specified in the Solver Parameters and Solver Options dialog boxes, or with the other Solver functions described in this section.

Syntax

SolverGet(TypeNum, SheetName)

TypeNum is a number specifying the type of information you want. The following settings are specified in the Solver Parameters dialog box.

```
TypeNum   Returns

1         The reference in the Set Cell box, or 0 if
          Solver has not been used on the active document

2         A number corresponding to the Equal To option
          1 = Max
          2 = Min
          3 = Value Of

3         The value in the Value Of box

4         The reference in the By Changing Cells box (only
          the first entry in the Variables list box)

5         The number of entries in the Constraints list box

6         An array of the left sides of the constraints as text

7         An array of numbers corresponding to the relationships
          between the left and right sides of the constraints:
          1  =  <=
          2  =  =
          3  =  >=
          4  =  int
          5  =  bin

8         An array of the right sides of the constraints as text
```

The following settings are specified in the Solver Options dialog box:

```
TypeNum   Returns

9         The Max Time value (as a number in seconds)

10        The Iterations value (max number of iterations)

11        The Precision value (as a decimal number)
```

```
12        The integer Tolerance value (as a decimal number)

13        1 if the Assume Linear Model check box is selected;
          0 otherwise.

14        1 if Show Iteration Result check box is selected;
          0 otherwise

15        1 if Use Automatic Scaling check box is selected;
          0 otherwise

16        A number corresponding to the type of Estimates:
          1  =  Tangent
          2  =  Quadratic

17        A number corresponding to the type of Derivatives:
          1  =  Forward
          2  =  Central

18        A number corresponding to the type of Search:
          1  =  Newton
          2  =  Conjugate

19        The Convergence value (as a decimal number) in the nonlinear
          GRG Solver

20        1 if the Assume Non-Negative check box is selected;
          0 otherwise

21        An array of the entries in the Variables list box as text
```

SheetName is the name of a worksheet that contains the Solver problem for which you want information. If **SheetName** is omitted, it is assumed to be the active sheet.

SolverLoad

Equivalent to choosing **Range Analyze Solver...**, choosing the Options button from the Solver Parameters dialog box, and choosing the Load Model button in the Solver Options dialog box. Loads Solver problem specifications that you have previously saved on the worksheet.

Syntax

SolverLoad(LoadArea)

LoadArea is a reference on the active worksheet to a range of cells from which you want to load a complete problem specification.

The first cell in **LoadArea** contains a formula for the Set Cell Box; the second cell contains a formula for the changing cells; subsequent cells contain constraints in the form of logical formulas. The last cell optionally contains an array of Solver option values (see SolverOptions).

Although **LoadArea** must be on the active worksheet, it need not be the current selection.

SolverOk

Equivalent to choosing **Range Analyze Solver...** and making entries and selections in the Solver Parameters dialog box. Specifies basic Solver options. The dialog box will not be displayed.

Syntax

SolverOk(SetCell, MaxMinVal, Valueof, ByChange)

SetCell corresponds to the Set Cell box in the Solver Parameters dialog box (the objective function in the optimization problem). **SetCell** must be a reference to a cell on the active worksheet. If you enter a cell, you must enter a value for **MaxMinVal**. If you do not enter a cell, you must include three zeroes (0, 0, 0) before the **ByChanging** value.

MaxMinVal corresponds to the options Max, Min and Value Of in the Solver Parameters dialog box. Use this option only if you entered a reference for **SetCell**.

MaxMinVal	Option specified
1	Maximize
2	Minimize
3	Value Of

ValueOf is the number that becomes the target for the cell in the Set Cell box if **MaxMinVal** is 3. **ValueOf** is ignored if the cell is being maximized or minimized.

ByChanging indicates the changing cells (decision variables), as entered in the By Changing Cells box. **ByChanging** must be a cell reference (usually a cell range or multiple reference) on the active worksheet. You can add more changing cell references using Form 2 of the SolverAdd function.

SolverOkDialog

Equivalent to choosing **Range Analyze Solver...** and making entries and selections in the Solver Parameters dialog box. The Solver Parameters dialog box will be displayed, and the user will be able to change the options you initially specify.

Syntax

SolverOkDialog(SetCell, MaxMinVal, Valueof, ByChange)

For an explanation of the arguments, see SolverOK.

SolverOptions

Equivalent to choosing **Range Analyze Solver...** and then choosing the Options button in the Solver Parameters dialog box. Specifies the Solver algorithmic options.

Syntax

SolverOptions(MaxTime, Iterations, Precision, AssumeLinear, StepThru, Estimates, Derivatives, SearchOption, IntTolerance, Scaling, Convergence, AssumeNonneg)

The arguments correspond to the options in the Solver Options dialog box. If any of the arguments are of the wrong type, LotusScript will report an error. If all arguments are of the correct type, but an argument has an invalid value, the function returns a positive integer corresponding to its position. A zero indicates that all options were accepted.

MaxTime must be an integer greater than zero. It corresponds to the Max Time edit box.

Iterations must be an integer greater than zero. It corresponds to the Iterations edit box.

Precision must be a number between zero and one, but not equal to zero or one. It corresponds to the Precision edit box.

AssumeLinear is a value 1 or 0 corresponding to the Assume Linear Model check box.

StepThru is a value 1 or 0 corresponding to the Show Iteration Results check box. If 1, Solver pauses at each trial solution; if 0 it does not. If you have supplied SolverSolve with a valid LotusScript function reference, your function or macro will be called each time Solver pauses; otherwise the standard Show Trial Solution dialog box will appear.

Estimates is the number 1 or 2 and corresponds to the Estimates option: 1 for Tangent and 2 for Quadratic.

Derivatives is the number 1 or 2 and corresponds to the Derivatives option: 1 for Forward and 2 for Central.

SearchOption is the number 1 or 2 and corresponds to the Search option: 1 for Newton and 2 for Conjugate.

IntTolerance is a number between zero and one, corresponding to the Tolerance edit box. This argument applies only if integer constraints have been defined.

Scaling is a value 1 or 0 corresponding to the Use Automatic Scaling check box. If 1, then Solver rescales the constraints internally to similar orders of magnitude during computation. If 0, Solver calculates normally.

Convergence is a number between zero and one, but not equal to zero or one. It corresponds to the Convergence box.

AssumeNonneg is a value 1 or 0 corresponding to the Assume Non-Negative check box. If 1, Solver supplies a lower bound of zero for all variables without explicit lower bounds in the Constraint list box.

SolverReset

Equivalent to choosing **Range Analyze Solver...** and choosing the Reset All button in the Solver Parameters dialog box. Erases all cell selections and constraints from the Solver Parameters dialog box and restores all the settings in the Solver Options dialog box to their defaults. The SolverReset function is automatically performed when you call SolverLoad.

Syntax

SolverReset()

SolverSave

Equivalent to choosing **Range Analyze Solver...**, choosing the Options button from the Solver Parameters dialog box, and choosing the Save Model button in the Solver Options dialog box. Saves the problem specifications on the worksheet.

Syntax

SolverSave(SaveArea)

SaveArea is a reference on the active worksheet to a range of cells or to the upper-left corner of a column of cells into which you want to save the current problem specification.

Remarks

If you specify only one cell for **SaveArea**, the area is extended downwards for as many cells as are required to hold the problem specifications (3 plus the number of constraints).

If you specify more than one cell and the area is too small for the problem, the problem specifications will not be saved, and the function will return the value 2.

SaveArea must be on the active worksheet, but it need not be the current selection.

SolverSolve

Equivalent to choosing **Range Analyze Solver...** and choosing the Solve button in the Solver Parameters dialog box. If successful, returns an integer value indicating the condition that caused the Solver to stop, as described below.

Syntax

SolverSolve(UserFinish, ShowRef)

UserFinish is a value 0 or 1, specifying whether to show the standard Solver Results dialog box.

If **UserFinish** is 1, SolverSolve returns its integer value without displaying anything. Your LotusScript program should decide what action to take (for example, by examining the return value or presenting its own dialog box); it *must* call SolverFinish in any case to return the worksheet to its proper state.

If **UserFinish** is 0, Solver displays the standard Solver Results dialog box, allowing the user to keep or discard the final solution values, and optionally produce reports.

ShowRef is either "", or a string representing a LotusScript function to be called instead of displaying the Show Trial Solution dialog box during the solution process. It is used when you want to regain control whenever Solver finds a new intermediate solution value. The function can inspect the current solution values on the worksheet, or take other actions such as saving or charting the intermediate values, as required by your application. It may determine on each call whether Solver should continue or stop.

In you are not using a custom function, **ShowRef** should be "". To write a custom ShowRef function, you must be familiar with and use certain advanced features of LotusScript. For technical details and examples of the ShowRef function, please visit the 1-2-3 Solver technical support pages and/or the discussion forum on the Frontline Systems World Wide Web site, http://www.frontsys.com.

Remarks

If a Solver problem has not been completely defined, SolverSolve returns -1. Otherwise the Solver "engine" is started, and the problem specifications are passed to it. When the solution process is complete, SolverSolve returns an integer value indicating the stopping condition:

Value	Stopping Condition
0	Solver found a solution. All constraints and optimality conditions are satisfied.
1	Solver has converged to the current solution. All constraints are satisfied.
2	Solver cannot improve the current solution. All constraints are satisfied.
3	Stop chosen when the maximum iteration limit was reached.
4	The Set Cell values do not converge.
5	Solver could not find a feasible solution.

6	Solver stopped at user's request.
7	The conditions for Assume Linear Model are not satisfied.
8	The problem is too large for Solver to handle.
9	Solver encountered an error value in a target or constraint cell.
10	Stop chosen when maximum time limit was reached.
11	There is not enough memory available to solve the problem.
12	Another 1-2-3 instance is using SOLVE123.DLL. Try again later.
13	Error in model. Please verify that all cells and constraints are valid.
14	Your free trial license to use the Solver has expired.