

Design and Use of the Microsoft Excel Solver

Daniel Fylstra

*Frontline Systems Inc., PO Box 4288,
Incline Village, Nevada 89450*

Leon Lasdon

*Department of Management Science and
Information Systems, College of Business
Administration, University of Texas,
Austin, Texas 78712*

John Watson

*Software Engines, 725 Magnolia Street,
Menlo Park, California 94025*

Allan Waren

*Computer and Information Science Department,
Cleveland State University, Cleveland, Ohio 44115*

In designing the spreadsheet optimizer that is bundled with Microsoft Excel, we and Microsoft made certain choices in designing its user interface, model processing, and solution algorithms for linear, nonlinear, and integer programs. We describe some of the common pitfalls users encounter and remedies available in the latest version of Microsoft Excel. The Solver has many applications and great impact in industry and education.

Since its introduction in February 1991, the Microsoft Excel Solver has become the most widely distributed and almost surely the most widely used general-purpose optimization modeling system. Bundled with every copy of Microsoft Excel and Microsoft Office shipped during the last eight years, the Excel Solver is in the hands of 80 to 90 percent of the 35 million users of office-productivity software worldwide. The remaining 10 to 20 percent of this audience use either Lotus 1-2-3 or Quattro Pro, both of which now include very similar spreadsheet solvers,

based on the same technology used in the Excel Solver.

This widespread availability has spawned many applications in industry and government. In education, increasing numbers of MBA and undergraduate business instructors have adopted the Excel Solver as their tool for introducing students to optimization: most management science textbooks now include coverage of the Excel Solver, and several recent texts use it exclusively in the optimization chapters.

We review the background and design

philosophy of the Excel Solver. We seek to explain why the Excel Solver works the way it does, to clear up some common misunderstandings and pitfalls, and to suggest ideas for good modeling practice when using spreadsheet optimization. We also briefly survey applications of the Excel Solver in industry and education and describe how practitioners who are not affiliated with the OR/MS community use it. The example models in this paper are available on Practice Online at (<http://silmaril.smeal.psu.edu/pol.html>) and at <http://www.frontsys.com/interfaces.htm>. Much more information—over 200 web pages at this writing—is available on Frontline Systems' World Wide Web site (<http://www.frontsys.com>).

The Microsoft Excel Solver combines the functions of a graphical user interface (GUI), an algebraic modeling language like GAMS [Brooke, Kendrick, and Meeraus 1992] or AMPL [Fourer, Gay, and Kernighan 1993], and optimizers for linear, nonlinear, and integer programs. Each of these functions is integrated into the host spreadsheet program as closely as possible. Many of the decisions we and Microsoft made in designing the Solver were motivated by this goal of seamless integration.

Optimization in Microsoft Excel begins with an ordinary spreadsheet model. The spreadsheet's formula language functions as the algebraic language used to define the model. Through the Solver's GUI, the user specifies an objective and constraints by pointing and clicking with a mouse and filling in dialog boxes. The Solver then analyzes the complete optimization model and produces the matrix form re-

quired by the optimizers in much the same way that GAMS and AMPL do.

The optimizers employ the simplex, generalized-reduced-gradient, and branch-and-bound methods to find an optimal solution and sensitivity information. The Solver uses the solution values to update the model spreadsheet and provides sensitivity and other summary information on additional report spreadsheets.

Background and Design Philosophy of the Excel Solver

The Microsoft Excel Solver and its counterparts in Lotus 1-2-3 97 and Corel Quattro Pro were not the first spreadsheet optimizers; that distinction belongs to What's Best!, conceived by Sam Savage, Linus Schrage, and Kevin Cunningham in 1985 and marketed by General Optimization Inc. for the Lotus 1-2-3 Release 2 spreadsheet [Savage 1985]. What'sBest! is still available in versions for each of the major spreadsheets and is now sold and supported by Lindo Systems Inc. Other early spreadsheet optimizers included Frontline Systems' What-If Solver [Frontline Systems 1990], Enfin Software's Optimal Solutions [Enfin Software 1988], and Lotus Development's Solver in earlier versions of 1-2-3 [Lotus Development 1990].

The design approach of What-If Solver, implemented in the graphical user interface of Excel, was chosen by Microsoft over several alternatives including What'sBest!; by Borland (the original developers of Quattro Pro) over an earlier solver developed internally by that company; and later by Lotus over their own internally developed solver. A major reason for this outcome, we believe, is that the Excel Solver had as its design goal

“making optimization a feature of spreadsheets,” whereas other packages, such as What’sBest!, “use the spreadsheet to do optimization.” In many small ways, the Excel Solver caters to the tens of millions of spreadsheet users, rather than to the tens of thousands of OR/MS professionals.

Although OR/MS professionals readily learn to use the Excel Solver, they often find certain aspects of its design puzzling or at least different from their expectations. In most cases, the differences are due to (1) the architecture of spreadsheet programs, (2) the expectations of the majority of spreadsheet users who are not OR/MS professionals, or (3) the desires of the spreadsheet vendors (Microsoft in the case of the Excel Solver).

The Architecture of Spreadsheet Programs

Because of the architecture of spreadsheet programs, it is easy to create spreadsheet models that contain discontinuous functions or even nonnumeric values. These models usually cannot be solved with classical optimization methods. The spreadsheet’s formula language is designed for general computations and not just for optimization. Indeed, Excel supports a rich variety of operators and several hundred built-in functions, as well as user-written functions. In contrast, GAMS, AMPL, and similar modeling languages include only a small set of operators and functions sufficient for expressing linear, smooth nonlinear, and integer optimization models.

The Expectations of Spreadsheet Users

The Excel Solver was designed to meet the expectations of spreadsheet users—in particular, users of earlier versions of Ex-

cel—rather than traditional OR/MS professionals. An example is the terminology it uses in dialog boxes, such as “Target Cell” (for the objective) and “Changing Cells” (for the decision variables). We used these terms—at Microsoft’s request—to mirror the terms used in the Goal Seek feature, which predated the Solver in Excel and in other spreadsheet programs. The Goal Seek feature, which spreadsheet users often describe as “what-if in reverse,” solves a nonlinear function of one variable for a specified value. Spreadsheet users see the Excel Solver as a more powerful successor to the Goal Seek feature [Person 1997]. Figure 1 shows Excel’s Goal Seek dialog box, and Figure 2 shows the Solver Parameters dialog box with its similar terminology.

The Desires of the Spreadsheet Vendors

The influence of the spreadsheet vendors’ desires is reflected in the way the Solver determines whether the model is linear or nonlinear. By default, the Solver assumes that the model is nonlinear. The user must select the Assume Linear Model check box in the Solver Options dialog box

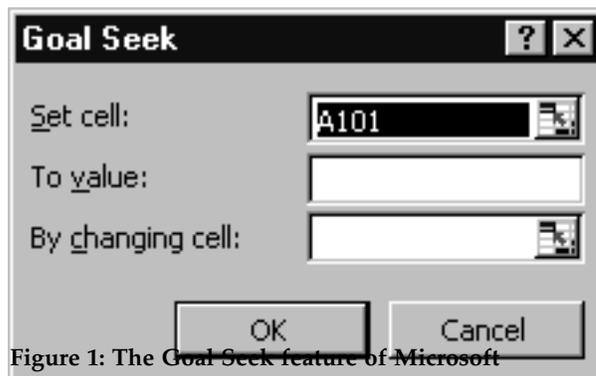


Figure 1: The Goal Seek feature of Microsoft Excel predated the Solver. This feature uses iterative methods to solve a simple equation (formula in the “set cell” equal to the “value”) in one variable (the “changing cell”).

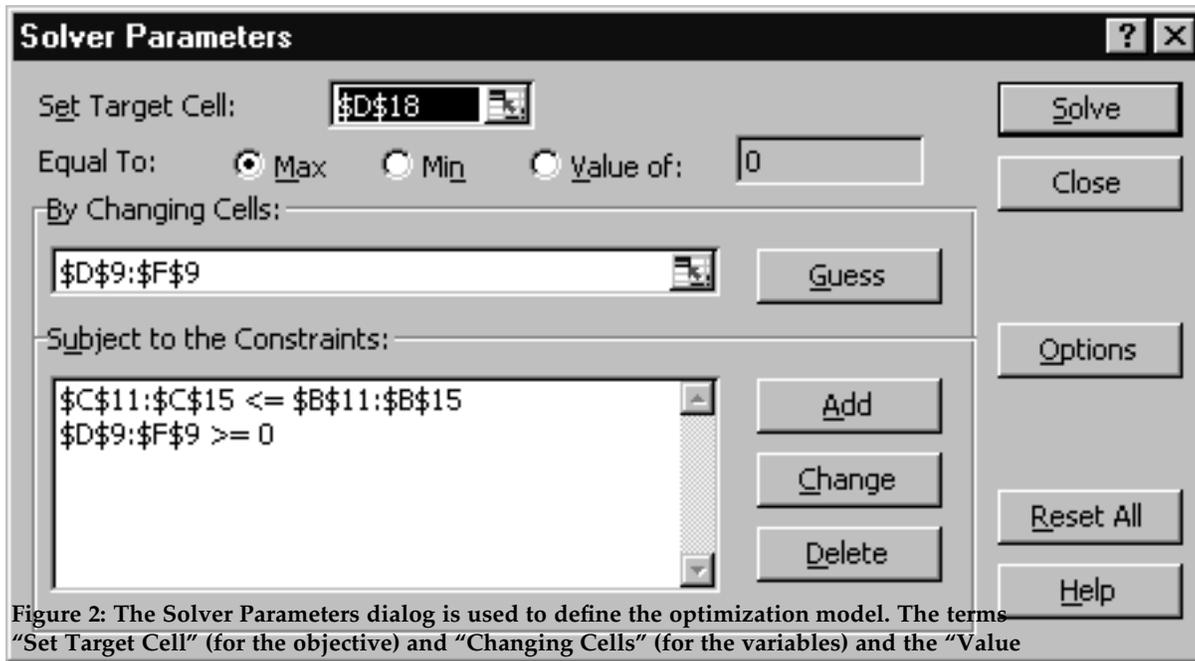


Figure 2: The Solver Parameters dialog is used to define the optimization model. The terms “Set Target Cell” (for the objective) and “Changing Cells” (for the variables) and the “Value of” option were derived from the earlier Goal Seek feature.

to override this assumption; the Solver does not attempt to automatically determine whether the model is linear by inspecting the formulas making up the model. Most of Excel’s several hundred built-in functions and all user-written functions would have to be treated as “not linear” (smooth nonlinear or discontinuous over their full domains) in an automatic test. But users sometimes create models using these functions and then add constraints that result in a linear model over the feasible region. Microsoft wanted a general approach that would support such cases and specified the use of the check box, as well as the use of the nonlinear solver as the default choice.

The Role of Bundled Spreadsheet Solvers

The “free” bundled version of the Excel Solver described in this paper and similar

products, such as What’sBest! Personal Edition, represent the low end of the range of spreadsheet solver functionality, capacity, and performance. More powerful versions are available and these versions are most often used to solve problems in industry. For example, where the standard Excel Solver supports just 200 decision variables, Frontline Systems’ Large-Scale LP Solver (a component of the Premium Solver Platform) supports up to 16,000 variables, and Lindo Systems’ What’sBest! Extended Edition supports up to 32,000 variables. Table 1 summarizes the characteristics of the Premium Solver products offered by Frontline Systems.

Like most optimization software, the Excel Solver has steadily improved in performance over the years. Although solution times are model dependent, in overall terms, the Solver in Excel 97 offers about

MICROSOFT EXCEL SOLVER

	Excel Built-In Solver	Premium Solver	Premium Solver Plus	Premium Solver Platform
NLP variables/constraints	200/100 + bounds	400/200 + bounds	400/200 + bounds	1000/1000 + bounds
LP variables/constraints	200/unlimited	800/unlimited	800/unlimited	2000/unlimited to 16,000/unlimited
Setup performance	1x	1–50x	1–50x	1–50x
NLP performance	1x	1x	1.5x	2–10x
LP performance	1x	2–3x	2–3x	Large scale
MIP performance	1x	5–10x	25–50x	25–50x
Selection of optimizers	Fixed set	Fixed set	Fixed set	Multiple choices, field-installable
LP/QP methods	Simplex w/bounds	Enhanced simplex w/bounds	Enhanced simplex, dual, quadratic	Sparse simplex, LU, Markowitz
MIP methods	B&B	Enhanced B&B	Enhanced B&B, P&P, dual, simplex	Enhanced B&B, P&P, dual, simplex
NLP methods	GRG2	GRG2	Enhanced GRG2	LSGRG, SQP, etc.
Reports	Standard: Answer, Limits, Sensitivity	Standard + Linearity, Feasibility	Standard + Linearity, Feasibility	Standard + Linearity, Feasibility

Table 1: The characteristics of the enhanced Excel Solvers are summarized in this table. For integer problems, “B&B” refers to branch and bound and “P&P” refers to preprocessing and probing. For nonlinear problems, “GRG” refers to the generalized reduced gradient method and “SQP” refers to sequential quadratic programming.

five times the performance of that in Excel 5.0 and perhaps 20 times the performance of the earliest version in Excel 3.0 (assuming a constant hardware platform). The Premium Solver further improves mixed-integer problem solution times by a factor of 25 to 50 over the Excel 97 Solver (Table 1). While spreadsheet solvers are unlikely to compete with dedicated optimizers, such as CPLEX and OSL, they do provide a practical platform for solving

real-world optimization problems.

User Interface and Selection of Objectives, Decision Variables, and Constraints

In the Excel Solver, as in an algebraic modeling system, the optimization model is defined by algebraic formulas (which appear in spreadsheet cells). Excel’s formula language can express a wide range of mathematical relationships, but Excel has no facilities for distinguishing decision

variables from other variables or objectives or constraints from other formulas. Hence, the Excel Solver provides both interactive and user-programmable ways to specify which spreadsheet cells are to serve each of these roles.

In interactive use, the user selects Tools Solver . . . from the Excel menu bar, displaying the Solver Parameters dialog box (Figure 2). As noted earlier, this dialog box is patterned after the Goal Seek feature (Figure 1). The “Value of” option offers a way to directly solve goal-seeking problems using the Solver; when the user selects this option and enters a target value, an equality constraint is added to the optimization model, and there is no objective to be maximized or minimized. (Alternatively, one may simply leave the Set Target Cell edit box blank and enter an equality constraint in the Constraint list box.) In either case, the problem is solved with a (constant) dummy objective, and the Solver stops when the first feasible solution is found. In this way, the Excel Solver fulfills spreadsheet users’ expectations of a more powerful Goal Seek capability that can be used to find solutions for systems of equations and inequalities.

Decision Variables and the Guess Button

Model decision variables are entered in the By Changing Cells edit box. Excel allows one to enter a so-called multiple selection, which consists of up to 16 ranges (rectangles, rows or columns, or single cells) separated by commas. Alternatively, one may press the Guess button to obtain an initial entry in the By Changing Cells edit box. This feature often puzzles OR/MS professionals; Ragsdale [1997] includes a sidebar saying that the “Solver

usually guesses wrong” and advising students not to use it, but many spreadsheet users find it useful. When one presses the Guess button, the Solver places a selection in the By Changing Cells edit box that includes all input (nonformula) cells on which the objective formula depends. This selection will usually include the actual decision variables as a subset and may be edited to remove ranges of cells that are not decision variables (for example, those that are fixed parameters in the model).

Constraints

The key issue in a spreadsheet solver’s user interface is the method of specifying constraints. What’sBest! originally used a “Rule of Constraints” that required every formula cell dependent on the variables to be nonnegative—but this form was not intuitive for typical spreadsheet users and was not acceptable to the spreadsheet vendors. (More recent versions of What’sBest! use a new constraint representation.) In the earlier Lotus-developed solver for 1-2-3, Lotus used logical expressions in the spreadsheet’s formula language, including the relational operators \leq , $=$, and \geq , to represent constraints. The solver dialog box simply offered an edit box in which a range of cells containing such logical formulas could be entered—thereby taking full advantage of an existing spreadsheet feature.

In the Excel Solver, in consultation with Microsoft, we chose a different way of specifying constraints, for several reasons. First, spreadsheet logical formulas (expressions that evaluate to TRUE or FALSE in Excel, or 1 or 0 in Lotus 1-2-3) are more general than constraints. They allow such relations as $<$, $>$, and $<>$ (not equal),

MICROSOFT EXCEL SOLVER

which are not easily handled by current optimization methods, as well as such logical operators as AND, OR, and NOT. Second, relations such as $A1 \geq 0$, are evaluated by the spreadsheet as strictly satisfied or unsatisfied, whereas an optimization algorithm evaluates constraints within a tolerance. For example, if $A1 = -0.0000005$, the Excel Solver would treat $A1 \geq 0$ as satisfied (using the default Precision setting of 10^{-6} or 0.000001), but the logical formula $= A1 \geq 0$ in a cell would display as FALSE. Third, constraints almost always come in blocks or indexed sets, such as $A1:A10 \geq 0$, and it is very advantageous for users to be able to enter such constraints and later view and edit them in block form. Hence, the Excel Solver provides a Constraint list box in the Solver Parameters dialog box where users can add, change, or delete blocks of

constraints by clicking the corresponding buttons.

In accord with the GUI conventions used throughout Excel, one can select blocks of cells for decision variables and for left-hand sides and right-hand sides of constraints by typing coordinates or by clicking and dragging with the mouse. The latter method is far more often used. Excel also allows the user to define symbolic names for individual cells or ranges of cells (through the Insert Name menu option). The Excel Solver will recognize any names the user has defined for the objective, variables, and blocks of constraints and will display them in the Solver Parameters dialog box (Figure 3).

For those who prefer to use spreadsheet logical formulas for constraints, the Excel Solver will read and write constraints in this form, when the Load Model and Save

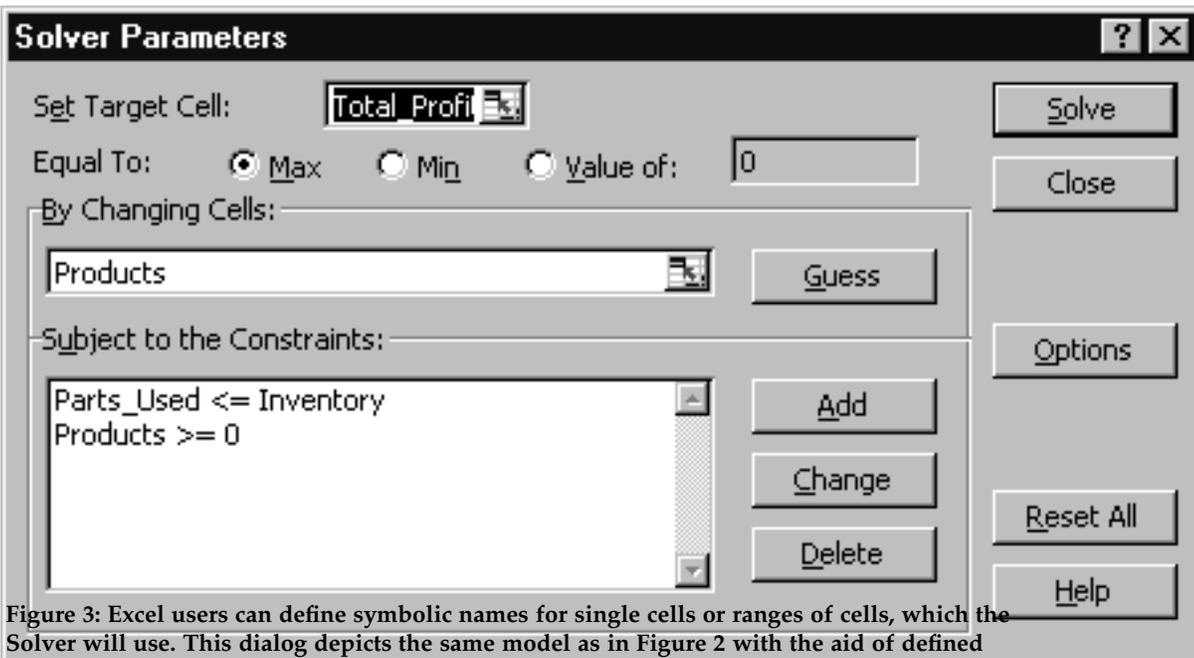


Figure 3: Excel users can define symbolic names for single cells or ranges of cells, which the Solver will use. This dialog depicts the same model as in Figure 2 with the aid of defined names, resulting in a much more readable model.

Model buttons in the Solver Options dialog box are used.

Solver Options

The user can control several options and tolerances used by the optimizers through the Solver Options dialog box (Figure 4). In the standard Excel Solver, all such options appear in one dialog box; in the Premium Solver products, where many more options and tolerances are available, each optimizer has a separate dialog box.

The Max Time and the Iterations edit boxes control the Solver’s running time. The Show Iteration Results check box instructs the Solver to pause after each major iteration and display the current “trial solution” on the spreadsheet. In lieu of these options, however, the user can simply press the ESC key at any time to interrupt the Solver, inspect the current iterate, and decide whether to continue or to stop.

The Assume Linear Model check box determines whether the simplex method or the GRG2 nonlinear programming algo-

rithm will be used to solve the problem. The Use Automatic Scaling check box causes the model to be rescaled internally before solution. The Assume Non-Negative check box places lower bounds of zero on any decision variables that do not have explicit bounds in the Constraints list box.

The Precision edit box is used by all of the optimizers and indicates the tolerance within which constraints are considered binding and variables are considered integral in mixed-integer-programming (MIP) problems. The Tolerance edit box (a somewhat unfortunate name, but Microsoft’s choice) is the integer optimality or MIP-gap tolerance used in the branch-and-bound method. The GRG2 algorithm uses the Convergence edit box and Estimates, Derivatives, and Search option button groups.

Modeling Practice

Excel, including the Solver, offers many convenient ways to select and manipulate blocks of cells for variables and constraints. Modelers should take advantage of this feature by laying out optimization models with indexed sets (for example, products, regions, or time periods) along the columns and rows of tables or blocks of cells. We also highly recommend the practice of defining names for indexed sets of variables and constraints and even for single cells. For example, the structure of the model with names defined as shown in Figure 3 is far more easily grasped than the same model with cell coordinate ranges as shown in Figure 2. Blocks of constraint values can often be computed more easily with Excel’s array of formulas, which provide some of the high-level fea-

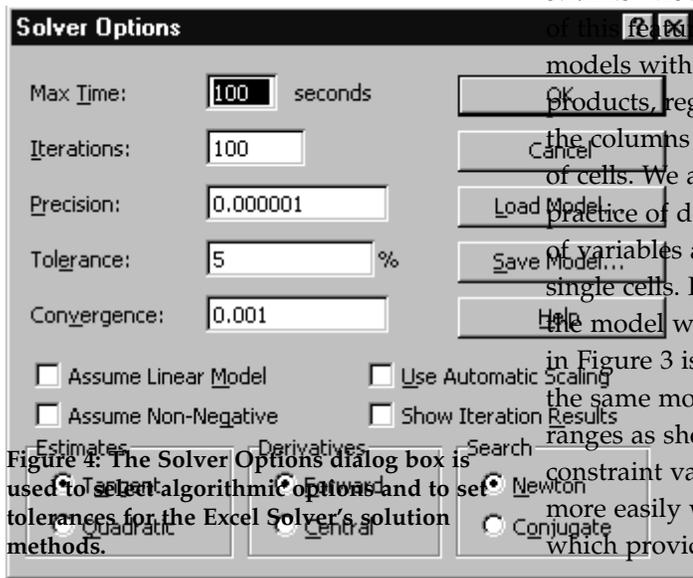


Figure 4: The Solver Options dialog box is used to select algorithmic options and to set tolerances for the Excel Solver’s solution methods.

tures of algebraic modeling languages, though without all of the flexibility of such languages.

For further suggestions on modeling practice for spreadsheet optimization, we encourage readers to consult Conway and Ragsdale [1997].

User Programmability

The user-programmable interface offered by the Excel Solver—a feature rarely found in other optimization modeling systems—is critically important to the many commercial users who are using Excel and Microsoft Office as a platform for developing custom applications. Every interactive, GUI-based action supported by the Excel Solver has a counterpart function call in Visual Basic for Applications (VBA), Excel's built-in programming language. (The earlier Excel macro language is also supported for backward compatibility.) All components of Excel share this feature, making it a flexible platform for decision support applications. For example, the new marketing textbook [Lilien and Rangaswamy 1997] includes a number of Excel Solver models that are controlled by VBA programs.

Model Extraction and Evaluation of the Jacobian Matrix

Like an algebraic modeling system such as GAMS or AMPL, the Excel Solver extracts the optimization problem from the spreadsheet formulas and builds a representation of the model suitable for an optimizer. For a linear programming (LP) problem, the focus of this model representation is the LP coefficient matrix. In more general terms, this is the Jacobian matrix of partial derivatives of the problem functions (objective and constraints) with re-

spect to the decision variables. In LP problems, the matrix entries are constant and need to be evaluated only once at the start of the optimization. In nonlinear programming (NLP) problems, the Jacobian matrix entries are variable and must be recomputed at each new trial point.

The Jacobian matrix could be obtained either analytically by symbolic differentiation of the spreadsheet formulas [Ng and Char 1979]; or during function evaluation through so-called automatic differentiation methods [Griewank and Corliss 1991]; or it could be approximated by finite differences [Gill, Murray, and Wright 1981]. This choice is a major design decision in any optimization modeling system, with many trade-offs. What'sBest! can be regarded as using the symbolic algebraic approach; systems such as GAMS and AMPI use automatic differentiation; and the Excel Solver uses finite differences.

The most important reason for choosing the finite difference approach for the Excel Solver was the requirement, set by Microsoft, that it support all of Excel's built-in functions as well as user-written functions. Symbolic differentiation would have been difficult for many of Excel's several hundred functions (and in fact, What'sBest! rejects most of them) and impossible for user-written functions. To use automatic differentiation we would have had to modify the Excel recalcuator and require user-written functions (often coded in other languages) to supply both function and derivative values, neither of which was possible. On the other hand, finite differences could be efficiently calculated using the finely tuned Excel recalcuator as is.

The Solver is concerned only with those formulas that relate the objective and constraints to the decision variables; it treats all other formulas on the spreadsheet as constant in the optimization problem. Excel, 1-2-3, and Quattro Pro all implement a form of minimal recalculation in which only those formulas that are dependent on the cell values that have changed need to be recalculated.

In calculating finite differences, the $[i,j]$ th element of the Jacobian matrix is approximated by the formula

$$\frac{f_i(x + \delta e_j) - f_i(x)}{\delta} \text{ where } \delta = \text{eps} \mid 1 = x_j \mid.$$

In this formula, e_j is the j th unit vector and eps is a perturbation factor, typically 10^{-8} approximately equal to the square root of the machine precision [Gill, Murray, and Wright 1981]. After an initial recalculation to evaluate $f(x)$, the Solver perturbs each variable in turn, recalculates the spreadsheet, and obtains values for the j th column of the Jacobian matrix. Hence the process requires $n + 1$ recalculations for an n variable problem; each recalculation after the first perturbs just one variable and resets another, thereby taking advantage of the spreadsheet's minimal recalculation feature.

Modeling Practice

The use of finite differences in the Excel Solver has a number of implications for spreadsheet modelers. The Solver's model processing allows users to employ any of Excel's several hundred built-in functions, as well as user-written functions, in constructing the spreadsheet. While many of these functions have nonlinear or non-smooth values, they can be used freely to

compute parameters of the model that do not depend on the decision variables, even if the optimization model is an LP. Indeed, it is often convenient to use IF, CHOOSE, and table LOOKUP functions in calculating parameters, and we frequently see these functions in models created by commercial users of Frontline Systems' Premium Solver products.

Computing finite differences does, however, take time to recalculate the spreadsheet. Bearing in mind that Excel will recalculate every formula on the current worksheet that depends on the decision variables—even those not involved in the optimization model—modelers can minimize this time by keeping auxiliary calculations on a separate worksheet. Because of the significant overhead in recalculating multiple worksheets, the Excel Solver currently requires that cells for the decision variables, the objective, and the left-hand sides of constraints appear on the active sheet, although model formulas and right-hand sides of constraints can refer to other sheets.

For users with models that take a long time to recalculate, we strongly recommend an upgrade to Excel 97, the latest version of Excel at this writing. Recalculation performance is greatly improved in this version, and the Solver is correspondingly faster on the majority of models. Frontline Systems' Premium Solver products offer additional ways to speed up evaluation of the Jacobian matrix (Table 1), and we plan further improvements in this area.

Solving Linear Problems

When a user checks the Assume Linear Model box (Figure 4) the Excel Solver uses

a straightforward implementation of the simplex method with bounded variables to find the optimal solution. This code operates directly on the LP coefficient matrix (that is, the Jacobian), which is determined using finite differences. The standard Excel Solver stores the full matrix, including zero entries; however, no matrix rows are required for simple variable bounds. Frontline Systems' Large-Scale LP Solver (Table 1) relies on a sparse representation of the matrix and of the LU factorization of the basis with dynamic Markowitz re-factorization, yielding better memory usage and improved numerical stability on large-scale problems.

Automatic Scaling and Related Pitfalls

Earlier versions of the standard Excel Solver had no provision for automatic scaling of the coefficient matrix; they used values directly from the user's spreadsheet. Since it is easy to rescale the objective and constraint values on the spreadsheet itself, we did not think that automatic scaling would be needed, especially for linear problems. We were wrong. Over the years, we have received many spreadsheet models from users—including business school instructors—that did not seem to solve correctly. In virtually all of these cases, the model was very poorly scaled—for example, with dollar amounts in millions for some constraints and return figures in percentages for others—yet none of these users identified scaling as a problem. It seems that in the widespread move to emphasize modeling over algorithms, such issues as scaling (still important in using software) have been de-emphasized or forgotten.

To improve performance of the nonlin-

ear solver in Excel 4.0, we added the Use Automatic Scaling check box to the Solver Options dialog box. But this dug a deeper pitfall for users with linear problems, since this automatic scaling option had no effect on the linear solver—and users often overlooked the documentation of this fact in Excel's online Help.

In Excel 97, the Use Automatic Scaling box applies to both linear and nonlinear problems. If the user checks this box and the Assume Linear Model box, the Solver rescales columns, rows, and right-hand sides to a common magnitude before beginning the simplex method. It unscales the solution values before storing them into cells on the spreadsheet. With this enhancement, the simplex solver is able to handle most poorly scaled models without any extra effort by the user.

Linearity Test and Related Pitfalls

For the reasons outlined earlier, the Excel Solver asks the user to specify whether the model is linear, but it does perform a simple numerical test to check the linearity assumption for reasonableness. This linearity test gave rise to another pitfall, again for poorly scaled models. Prior to Excel 97, the Solver performed this test after it had obtained a solution using the simplex method. It used these solution values x^* and the initial values x^0 for the variables to check that the objective and each constraint function $f_i(x)$, evaluated by recalculating the spreadsheet, satisfied the following condition:

$$|f_i(x^*) - (f_i(x^0) + \nabla f_i(x^0)(x^* - x^0))| \leq tol.$$

Here $\nabla f_i(x^0)$ is the function gradient, that is, the appropriate row of the LP coefficient matrix, and tol is the Precision value

in the Solver Options dialog box with a default value of 10^{-6} .

Given that the model might contain any of the hundreds of Excel built-in functions as well as user-written functions and that the test is performed at discrete points, this test cannot be perfect; very occasionally, a model with nonlinear, or even discontinuous functions, will pass the linearity test. In practice, however, this linearity test almost always detects situations in which the user has accidentally set up a model that doesn't satisfy the linearity assumption—and truly linear models will always pass the linearity test, as long as they are well scaled.

Unfortunately, linear models that are poorly scaled will sometimes fail this test. Since the resulting error message is "The conditions for Assume Linear Model are not satisfied," the user who is not conscious of the effect of poor scaling may not realize that this is the problem. (The only saving grace is that very poorly scaled models, which might otherwise yield incorrect answers in the absence of automatic scaling, almost always give this error message instead.)

In Excel 97, we have substantially revised the linearity test. The Solver performs a quick check before solving the problem by verifying that the problem functions, evaluated at several multiples of the initial variable values, satisfy the above condition. If the problem fails this test, the user is warned against using the simplex method. When the Solver finds an optimal solution using the simplex method it performs a further check. It verifies that the objective function and constraint slacks, obtained by recalculating

the spreadsheet at the optimal point, match the values provided by the LP solution within the Precision value in the Solver Options dialog. As long as the user selects the Use Automatic Scaling box, so that the values in the LP matrix are well scaled internally, this test should be robust even for poorly scaled models.

Modeling Practice

Students (and instructors) who use Excel 97, with its automatic scaling and its improved linearity test, can avoid the pitfalls described earlier. We strongly encourage business school instructors to upgrade to Excel 97 as soon as possible. Schools still using Windows 3.1 can obtain an academic version of Frontline Systems' Premium Solver for Excel 5.0 with the same enhancements, but support for this 16-bit version will be limited in the future. Still, we emphasize that, while we have used scaling methods favored in the literature [Gill, Murray, and Wright 1981], no automatic scaling method is perfect. It will always be possible to create examples that cause problems in spite of automatic scaling, and we suggest that instructors devote at least some time to explaining the limitations of finite precision computer arithmetic to students. Ragsdale [1997] addresses scaling briefly but effectively, for instance. The example model in Figure 5, which is available for download on Practice Online, is a poorly scaled variant of the Working Capital Management worksheet distributed with Excel. It will yield a nonoptimal solution (of all zeroes) in Excel 5.0 and 7.0 and in Excel 97 if the Use Automatic Scaling box is cleared. It yields the correct solution in Excel 97 if the user checks the Use Automatic Scaling box.

	A	B	C	D	E	F	G	H	I
1	Example 4: Working Capital Management.								
2	Determine how to invest excess cash in 1-month, 3-month and 6-month CDs so as to								
3	maximize interest income while meeting company cash requirements (plus safety margin).								
4									
5		<i>Yield</i>	<i>Term</i>				<i>Purchase CDs in months:</i>		
6	<i>1-mo CDs:</i>	1.0%	1				1, 2, 3, 4, 5 and 6		<i>Interest</i>
7	<i>3-mo CDs:</i>	4.0%	3				1 and 4		<i>Yield:</i>
8	<i>6-mo CDs:</i>	9.0%	6				1	<i>Total</i>	2%
9									
10	<i>Month:</i>	<i>Month 1</i>	<i>Month 2</i>	<i>Month 3</i>	<i>Month 4</i>	<i>Month 5</i>	<i>Month 6</i>	<i>End</i>	
11	<i>Init Cash:</i>	\$400,000	\$205,000	\$216,000	\$237,000	\$158,400	\$109,400	\$125,400	
12	<i>Matur CDs:</i>		100,000	100,000	110,000	100,000	100,000	120,000	
13	<i>Interest:</i>		1,000	1,000	1,400	1,000	1,000	2,300	
14	<i>1-mo CDs:</i>	100,000	100,000	100,000	100,000	100,000	100,000		
15	<i>3-mo CDs:</i>	10,000			10,000				
16	<i>6-mo CDs:</i>	10,000							
17	<i>Cash Uses:</i>	75,000	(10,000)	(20,000)	80,000	50,000	(15,000)	60,000	
18	<i>End Cash:</i>	\$205,000	\$216,000	\$237,000	\$158,400	\$109,400	\$125,400	\$187,700	

Figure 5: This spreadsheet, which can be downloaded from Practice Online as FIGURE5.XLS, is a poorly scaled model that “fools” the linearity test in earlier Excel versions, yielding the message “The conditions for Assume Linear Model are not satisfied.”

Solving Nonlinear Problems

When the Assume Linear Model box in the Solver Options dialog is cleared, the Excel Solver uses the generalized reduced gradient method, as implemented in the GRG2 code [Lasdon et al. 1978], to solve the problem. Like other gradient-based methods, GRG2 is guaranteed to find a local optimum only on problems with continuously differentiable functions and then only in the absence of numerical difficulties (such as degeneracy or ill conditioning). However, GRG2 has a reputation for robustness, compared with other nonlinear optimization methods, on difficult problems where these conditions are not fully satisfied.

Problem Representation

GRG2 requires function values and the Jacobian matrix (which is not constant for nonlinear models). The Excel Solver approximates the Jacobian matrix using finite

differences as described earlier and re-evaluates it at the start of each major iteration.

Automatic Scaling

A poorly scaled model can cause even more problems for GRG2 than for the simplex method. The earliest version of the Excel Solver used variable and constraint values directly from the spreadsheet, but as of Excel 4.0 (released in 1992), the Solver rescales both variable and function values internally if the user checks the Use Automatic Scaling box in the Solver Options dialog box. Unlike the simplex code, which uses gradient values for scaling (as of Excel 97), the GRG2 algorithm in Excel uses typical-value scaling. In this approach GRG2 rescales the decision variables and problem functions by dividing by their initial values at the beginning of the solution process. (We chose this approach because our tests showed that gradient-based scal-

ing was not very effective on typical nonlinear spreadsheet models where scaling was a problem.)

GRG2 Stopping Conditions

Like the simplex method, the GRG2 algorithm will stop when it finds an optimal solution, when the objective appears to be unbounded, when it can find no feasible solution, or when it reaches the time limit or maximum number of iterations. For nonlinear models, an “optimal solution” means that the Solver has found a local optimum where the Kuhn-Tucker conditions are satisfied to within the convergence tolerance; the message displayed is “Solver found a solution.” GRG2 also stops when the current solution meets a “slow progress” test: The relative change in the objective is less than the convergence tolerance for the last five iterations. In this case, the message displayed is “Solver converged to the current solution.” In previous Excel versions, the convergence tolerance was fixed at 10^{-4} or 10^{-5} (depending on the version) and could not be changed by the user. In Excel 97, there is new Convergence edit box (Figure 4) that sets this tolerance.

The message “Solver could not find a feasible solution” occurs when the GRG2 algorithm terminates with a positive sum of infeasibilities. This almost always indicates a truly infeasible model, but with nonlinear problems this can occur (rarely) in feasible problems if GRG2 finds a local optimum of the phase one objective (the sum of the infeasibilities) or if GRG2 simply terminates in phase one due to slow progress. Remedies available through the Solver Options dialog box (Figure 4) include using automatic scaling, increasing

the feasibility tolerance (Precision option), decreasing the convergence tolerance to make it more difficult to terminate in phase one, trying central differences, and trying other starting points.

Nonsmooth Functions

The convergence results for gradient-based methods, such as GRG2, depend on differentiability of the problem functions. The spreadsheet formula language is designed to express arbitrary calculations, and users can easily create optimization models that include nonsmooth functions, that is, functions with discontinuous values or first partial derivatives at one or more points. Examples of such functions are ABS, MIN and MAX, INT and ROUND, CEILING and FLOOR, and the commonly used IF, CHOOSE, and LOOKUP functions. Expressions involving relations (outside the context of Solver-recognized constraints) and such Boolean operators as AND, OR, and NOT are discontinuous at their points of transition between FALSE and TRUE values.

The presence of any of these (or many other) functions in a spreadsheet does not necessarily mean that the optimization model is nonsmooth. For example, an IF function whose conditional expression is independent of the decision variables and whose result expressions are smooth is itself smooth. Similar statements apply to the other functions mentioned above.

Even if the problem is nonsmooth, GRG2 may never encounter a point of discontinuity. This depends on the path that the algorithm takes, which depends on the starting point. GRG2 may simply skip over a discontinuity or may never encounter a region where discontinuities occur.

MICROSOFT EXCEL SOLVER

Problems occur when the finite difference process (which approximates partial derivatives) spans both sides of a discontinuity, for then the estimated derivatives are likely to be very large. If GRG2 is converging to a local solution where the objective is nonsmooth, inaccurate derivative estimates near the solution are likely to cause it to oscillate about that point and to terminate because of a small fractional change in the objective.

Modeling Practice

The path GRG2 takes and the scaling

factors it uses depend on the initial values of the variables. Users should take care to start the solution process with values for the variable cells that are representative of the values expected at the optimal solution, rather than with arbitrary values, such as all zeroes. The example spreadsheet in Figure 6, which is available for download on Practice Online, is an Excel version of a product-mix and pricing model from Fylstra [1992]. If the model is solved with initial values of zero for all four variables, GRG2 stops immediately,

	A	B	C	D	E	F
1	Product Mix and Pricing Problem					
2	Your company manufactures two products, AC Units and Heat Pumps, from a common					
3	set of resources including AC Compressors, HP Compressors, and labor hours. Your goal is					
4	to choose the best mix of products to make AND the best prices at which to sell them, in					
5	order to maximize profit. Demand for your products is inversely related to product price.					
6						
7				<i>AC Unit</i>	<i>Heat Pump</i>	
8		<i>Number to Build-></i>		1	1	
9		<i>Selling Price-></i>		101.00	151.00	
10	<i>Cost per Unit</i>			100.00	150.00	
11	<i>Profit per Unit</i>			1.00	1.00	
12	<i>Demand</i>			2238	1598	
13	<i>Unfilled Demand</i>			2237	1597	
14	<i>Part Name</i>	<i>Inventory</i>	<i>No. Used</i>			
15	<i>AC Compressors</i>	1195	1	1	0	
16	<i>HP Compressors</i>	980	1	0	1	
17	<i>Labor hours</i>	2400	3	1	2	
18				<i>Profits:</i>		
19		<i>By Product</i>		\$1	\$1	
20		Total		\$2		
21						

Figure 6: This spreadsheet, which can be downloaded from Practice Online as FIGURE6.XLS, causes the GRG2 nonlinear solver to stop at a nonoptimal solution if the initial values of all variables are zero. GRG2 finds the correct optimal solution for initial variable values that make the profits per unit positive.

declaring this point to be an “optimal solution” (in fact, this point is a Kuhn-Tucker point). With initial values that make each quantity to build and the profit per unit positive, GRG2 finds the correct optimal solution. Alternatively, if one changes the constraint that requires production to be less than or equal to demand to an equality constraint, GRG2 is able to find the correct solution even with initial values of zero, since it can solve for certain variables in terms of others.

We encourage users who encounter difficulty with slow progress or who receive the message “Solver converged to the current solution” to upgrade to Excel 97, which allows them to control the convergence tolerance. The example model in

Figure 7, also available for download on Practice Online, is a variant of the Quick Tour worksheet distributed with Excel. If this model is solved in Excel 97 with the default convergence tolerance of 10^{-4} , the Solver stops with the message “Solver converged to the current solution” and an objective value of \$79,705.55, just short of the true optimum. If the convergence tolerance is tightened to 10^{-5} , the Solver stops with “Solver found a solution” and an objective value of \$79,705.62. (In Excel 5.0 and 7.0, solving this model yields the optimal objective of \$79,705.62, because the convergence tolerance is hard-wired in these versions to 10^{-5} .)

GRG2 uses the value in the Precision edit box shown in Figure 4 (default 10^{-6})

	A	B	C	D	E	F	G
1	Quick Tour of Microsoft Excel Solver						
2	<i>Month</i>	<i>Q1</i>	<i>Q2</i>	<i>Q3</i>	<i>Q4</i>	<i>Total</i>	
3	<i>Seasonality</i>	0.9	1.1	0.8	1.2		
4							
5	<i>Units Sold</i>	3,592	4,390	3,192	4,789	15,962	
6	<i>Sales Revenue</i>	\$143,662	\$175,587	\$127,700	\$191,549	\$638,498	
7	<i>Cost of Sales</i>	89,789	109,742	79,812	119,718	399,061	
8	<i>Gross Margin</i>	53,873	65,845	47,887	71,831	239,437	
9							
10	<i>Salesforce</i>	8,000	8,000	9,000	9,000	34,000	
11	<i>Advertising</i>	10,000	10,000	10,000	10,000	40,000	
12	<i>Corp Overhead</i>	21,549	26,338	19,155	28,732	95,775	
13	<i>Total Costs</i>	39,549	44,338	38,155	47,732	169,775	
14							
15	<i>Prod. Profit</i>	\$14,324	\$21,507	\$9,732	\$24,099	\$69,662.10	
16	<i>Profit Margin</i>	10%	12%	8%	13%	11%	
17							
18	<i>Product Price</i>	\$40.00					
19	<i>Product Cost</i>	\$25.00					
20							
21							

Figure 7: This spreadsheet, which can be downloaded from Practice Online as FIGURE7.XLS, shows how the GRG2 nonlinear solver can stop with the message “Solver converged to the current solution.” With a tighter convergence tolerance, it stops at a slightly better, optimal point with the message “Solver found a solution.”

for its feasibility tolerance. Constraints are classified as active when they are within this (absolute) tolerance of one of their bounds and are violated when their bound violation exceeds this tolerance. The default value is rather tight for nonlinear problems, and users may find that they can solve some problems with nonlinear constraints faster or even to a better result if they increase this value. We recommend 10^{-4} for nonlinear problems but caution against using values greater than 10^{-2} . Users requiring high accuracy may prefer the default value. For nonlinear problems, maximum accuracy results from choosing central differences and the default feasibility tolerance.

When a model is nonsmooth or nonconvex, we recommend trying several different starting points. If GRG2 reaches roughly the same final point, one can be fairly confident that this is a global solution. If not, one can choose the best of the solutions obtained.

For further information on reduced gradient methods and the GRG2 solver, see Lasdon and Smith [1992].

Solving Problems with Integer Constraints

When a problem includes integer variables, the Excel Solver invokes a branch-and-bound (B&B) algorithm that can use either the simplex method or GRG2 to solve its subproblems. The user indicates which of the decision variables are integer by adding constraints, such as $A1:A10 = \text{integer}$ (or, in Excel 97, $A1:A10 = \text{binary}$), where $A1:A10$ is a range of variable cells. (One enters such constraints by selecting "int" or "bin" from the Relation list in the Add or Change Constraints dialog box.)

The branch-and-bound algorithm starts by solving the relaxed problem (without the integer constraints) using either GRG2 or the simplex method, yielding an initial best bound for the problem including the integer constraints. The algorithm then begins branching and solving subproblems with additional (or tighter) bounds on the integer variables. A subproblem whose solution satisfies all of the integer constraints is a candidate for the solution of the overall problem; the candidate with the best objective value so far is saved as the incumbent. The algorithm uses the best objective of the remaining nodes to be fathomed to update the best bound. Each time the algorithm finds a new incumbent, it computes the relative difference between its objective and the current best bound, yielding an upper bound on the improvement in the objective that might be obtained by continuing the solution process:

$$\frac{\text{Objective}(\text{Incumbent}) - \text{Objective}(\text{BestBound})}{\text{Objective}(\text{BestBound})}$$

If this value is less than or equal to the Tolerance edit box value (Figure 4), the algorithm stops. Some users have failed to notice that the default tolerance amount is not zero but 0.05 and have therefore concluded that the Excel Solver was not finding the correct integer solution. We chose this default value, at Microsoft's request, to limit the time taken by nontrivial integer problems. It often happens that the branch-and-bound algorithm finds a reasonably good solution fairly quickly and then spends a great deal of time finding (or verifying that it has found) the true integer optimal solution.

In the standard Excel Solver, the branch-

and-bound algorithm uses a breadth-first search that branches on the unfathomed node with the best objective. Frontline Systems' Premium Solver products use much more elaborate strategies (Table 1). These include a depth-first search that continues until it finds an incumbent, followed by a breadth-first search; more sophisticated rules for choosing the next node to be fathomed; rules for reordering the integer variables chosen for branching; use of the dual-simplex method for the subproblems; and preprocessing-and-probing (P&P) strategies for binary integer variables. These improvements often dramatically reduce solution time on integer problems (Table 1).

It is possible to solve nonlinear integer problems with the Excel Solver, but users should be aware of the intrinsic limitations of this process. On a linear problem, the simplex method can conclusively determine whether each subproblem is feasible and, if so, return the globally optimal solution to that subproblem. On nonlinear integer problems, the GRG algorithm (or any gradient-based method) may fail to find a feasible solution for a subproblem even though one exists, or it may return a local optimum that is not global. This also means that the best bound used by the branch-and-bound algorithm will be based on local optima found by GRG2 and this may not be the global optimum. Because of this, the branch-and-bound algorithm is not guaranteed to find the true integer optimum for nonlinear problems, although it will often succeed in finding a "good" integer solution.

Modeling Practice

It is important for users to understand the role of the Tolerance edit box value. In

a classroom environment, instructors may wish to have students set this value to zero to ensure that the Solver will continue branching until it finds the optimal integer solution.

Users attempting to solve nonlinear integer problems should also take careful note of the limitations cited above for the branch-and-bound algorithm when used with GRG2.

Even small, academic-size integer problems may require a great deal of solution time with the standard Excel Solver. Here again, we recommend an upgrade to Excel 97, which will improve solution times for both linear and nonlinear subproblems. An even better alternative is Frontline Systems' Premium Solver for Excel 97, which offers algorithmic improvements to reduce both the number of subproblems and the time spent on each one. An academic version of the Premium Solver is available and has proven quite popular with business school instructors.

Saving the Solution and Producing Solver Reports

When one of the Excel Solver's optimizers returns a solution, the Solver places the solution values into the decision variable cells, recalculates the spreadsheet, and displays the Solver Results dialog box (Figure 8). From this dialog box, the user can choose to keep the optimal solution, or discard it and restore the initial values of the variables. In addition, the user can select one or more reports, which the Solver will then produce in the form of additional worksheets inserted into the current workbook.

Assuming that the user (or a Visual Basic program controlling the Solver) decides

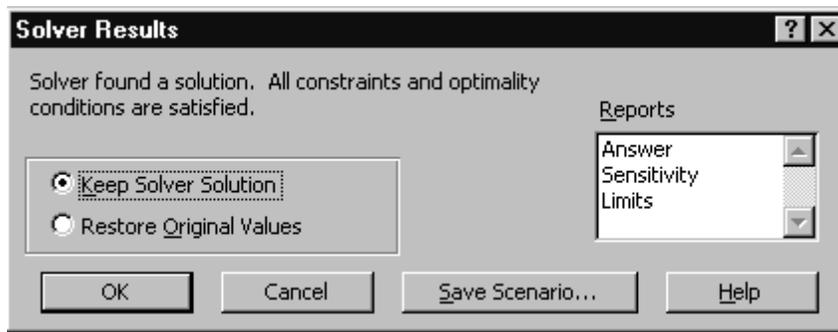


Figure 8: The Solver Results dialog box is displayed whenever the Solver stops. It allows the user to keep the solution or restore the original values of the variable cells and produce one or more of the Solver's reports.

to keep the solution, the Solver updates all of the model's results appropriately, including the objective, the constraints, and other auxiliary calculations that depend on the decision variables. One can use any of these model values to draw charts and graphs, update external databases, and the like using standard Excel facilities. A Visual Basic program may also inspect the values and may further manipulate them or store them for later use. For example, it is an easy classroom exercise to generate and graph the efficient frontier in a portfolio optimization problem in finance.

The standard Excel Solver can produce three types of reports: the Answer Report (Figure 9), the Sensitivity Report (Figure 10), and the Limits Report (Figure 11). The Premium Solver products (Table 1) can also produce a Linearity Report and a Feasibility Report. The Linearity Report highlights the constraints involved when an attempt to solve with the simplex method fails the linearity test described earlier. The Feasibility Report highlights an "irreducible inconsistent system" of constraints [Chinneck 1997] when an attempt to solve a linear problem yields no feasible

solution.

The Answer Report provides the initial and final values of the variables and the objective and optimal values for each constraint's left-hand side as well as slack values for nonbinding constraints.

The Sensitivity Report provides final solution values and dual values for variables and constraints in both linear and nonlinear models. For linear models, the dual values are labeled "reduced costs" and "shadow prices"; their values and ranges of validity are included in the report. For nonlinear models, the dual values are valid only for small changes about the optimal point, and they are labeled "reduced gradients" and "Lagrange multipliers."

The Solver creates the Limits Report by rerunning the optimizer with each decision variable selected in turn as the objective, both maximizing and minimizing, while holding all other variables fixed at their optimal values. The report shows the resulting lower limit and upper limit for each variable and the corresponding value of the original objective function. OR/MS professionals are sometimes puzzled by the inclusion of this report, but Microsoft

	A	B	C	D	E	F	G
1	Microsoft Excel 8.0 Answer Report						
2	Worksheet: [SOLVSAMP.XLS]Product Mix						
3	Report Created: 5/25/97 4:18:36 PM						
4							
5							
6	Target Cell (Max)						
7	Cell	Name	Original Value	Final Value			
8	\$D\$18	Total Profits:	\$16,000	\$25,000			
9							
10							
11	Adjustable Cells						
12	Cell	Name	Original Value	Final Value			
13	\$D\$9	Number to Build-> TV set	100	200			
14	\$E\$9	Number to Build-> Stereo	100	200			
15	\$F\$9	Number to Build-> Speaker	100	0			
16							
17							
18	Constraints						
19	Cell	Name	Cell Value	Formula	Status	Slack	
20	\$C\$11	Chassis No. Used	400	\$C\$11<=\$B\$11	Not Binding	50	
21	\$C\$12	Picture Tube No. Used	200	\$C\$12<=\$B\$12	Not Binding	50	
22	\$C\$13	Speaker Cone No. Used	800	\$C\$13<=\$B\$13	Binding	0	
23	\$C\$14	Power Supply No. Used	400	\$C\$14<=\$B\$14	Not Binding	50	
24	\$C\$15	Electronics No. Used	600	\$C\$15<=\$B\$15	Binding	0	
25	\$D\$9	Number to Build-> TV set	200	\$D\$9>=0	Not Binding	200	
26	\$E\$9	Number to Build-> Stereo	200	\$E\$9>=0	Not Binding	200	
27	\$F\$9	Number to Build-> Speaker	0	\$F\$9>=0	Binding	0	

Figure 9: The Answer Report summarizes the initial and final values of the variables, constraints, and objective and indicates whether the constraints are binding (satisfied with equality) or have slack.

specified it for competitive reasons, since the former Lotus-developed solver in 1-2-3 featured a similar report.

Report Pitfalls

There are two pitfalls that users sometimes encounter with these reports. The more common problem arises from the fact that the report spreadsheets are constructed so that each cell “inherits” its formatting from the corresponding cell in the user’s model. This feature, which Microsoft specified, has the advantage that the

report values are automatically formatted with dollars and cents, percent symbols, scientific notation, or whatever custom formatting was used in the model. The pitfall arises when users format their models to display variable and constraint values rounded to integers (say), which causes the corresponding dual values to be formatted as integers also. Not realizing this, some users think that the dual values are wrong. However, the Solver stores the dual values to full precision on the report

	A	B	C	D	E	F	G	H
1	Microsoft Excel 8.0 Sensitivity Report							
2	Worksheet: [SOLVSAMP.XLS]Product Mix							
3	Report Created: 5/25/97 4:18:36 PM							
4								
5								
6	Adjustable Cells							
7			Final	Reduced	Objective	Allowable	Allowable	
8	Cell	Name	Value	Cost	Coefficient	Increase	Decrease	
9	\$D\$9	Number to Build-> TV set	200	0	75	25	5	
10	\$E\$9	Number to Build-> Stereo	200	0	50	25	12.5	
11	\$F\$9	Number to Build-> Speaker	0	-3	35	2.5	1E+30	
12								
13	Constraints							
14			Final	Shadow	Constraint	Allowable	Allowable	
15	Cell	Name	Value	Price	R.H. Side	Increase	Decrease	
16	\$C\$11	Chassis No. Used	400	0	450	1E+30	50	
17	\$C\$12	Picture Tube No. Used	200	0	250	1E+30	50	
18	\$C\$13	Speaker Cone No. Used	800	13	800	100	100	
19	\$C\$14	Power Supply No. Used	400	0	450	1E+30	50	
20	\$C\$15	Electronics No. Used	600	25	600	50	200	

Figure 10: The Sensitivity Report shows, for linear problems, reduced costs for the variables and shadow prices for the constraints, as well as the ranges of validity of these dual values.

	A	B	C	D	E	F	G	H	I	J
1	Microsoft Excel 8.0 Limits Report									
2	Worksheet: [SOLVSAMP.XLS]Product Mix									
3	Report Created: 5/25/97 4:18:36 PM									
4										
5										
6			Target							
7	Cell	Name	Value							
8	\$D\$18	Total Profits:	\$25,000							
9										
10										
11			Adjustable	Lower Target		Upper Target				
12	Cell	Name	Value	Limit	Result	Limit	Result			
13	\$D\$9	Number to Build-> TV set	200	0	10000	200	25000			
14	\$E\$9	Number to Build-> Stereo	200	0	15000	200	25000			
15	\$F\$9	Number to Build-> Speaker	0	0	25000	0	25000			
16										

Figure 11: The Limits Report shows the objective value obtained by maximizing and minimizing each variable in turn while holding the other variables' values constant.

spreadsheet; one can inspect each value by selecting it with the mouse, and one can easily reformat the values to whatever precision one desires.

The second pitfall relates only to the Sensitivity Report. The Excel Solver recognizes constraints that are simple bounds on the variables and passes them in this form to both the simplex and GRG2 optimizers, where they are handled more efficiently than if they were included as general constraints. If one of these constraints is binding at the solution this actually means that the corresponding decision variable has been driven to its bound. The dual value for this binding constraint will appear as a reduced cost for the decision variable, rather than as a shadow price for the constraint; it will be nonzero if the variable was nonbasic at the solution. (In fact, constraints that are simple bounds on the variables are never listed in the Constraints section of the Sensitivity Report.)

We encourage modelers to take advantage of the fact that the reports are spreadsheets. Not only can they view them but they can easily modify them, use them to draw charts and graphs, transfer them to other programs, or inspect them using Visual Basic programs. Since the reports show a text label as well as a cell reference for each variable and constraint, users can easily design their spreadsheet models so that meaningful labels appear on the reports. The algorithm for constructing these labels is very simple: starting from the variable or constraint cell on the model worksheet, the Solver looks left and up for the first text label in the same row and the first text label in the same column. It then concatenates these two labels to

form the label that appears for that cell in the report.

Users should avoid the pitfalls cited above. Because the default formatting for cells is general, report values will appear to full precision unless the user defines custom formatting for the variable or constraint cells. If one wants such formatting, one must simply bear in mind its effect on the reports. To see the dual values for simple variable bounds in the Constraints section of the Sensitivity Report, one can modify the constraint right-hand side to be (say) the formula $0 + 5$ rather than the constant 5. In this case, the Solver will not recognize the constraint as a simple variable bound. In Frontline Systems' Premium Solver products, we changed this report so that dual values always appear in the Constraints section of the report, even for constraints that are recognized as simple variable bounds, making this work-around unnecessary.

Use of the Solver in Industry

We have heard many opinions about use of the Excel Solver from OR/MS professionals. Many view spreadsheet solvers as suitable only for quite small problems or only for educational rather than industrial use. Some wonder how such tools can be successfully employed by individuals with little if any formal training in OR/MS methods. Some, seeing little usage of the Excel Solver among their colleagues, think that the Solver is widely distributed but not very widely used.

We do not have enough systematic data to project the actual number of users of the Excel Solver among the 30-million-plus copies of Microsoft Office and Excel distributed to date. But based on our contacts

with users and the data we do have, we believe that OR/MS professionals are seeing only the proverbial tip of the iceberg and that use of the Excel Solver is far more widespread than their comments would suggest.

Problem Size

Having worked with commercial users for more than five years, we are very confident that spreadsheet solvers are capable of solving the majority of industrial LP models, as well as many integer and nonlinear models. We base this belief on our own experience and on information about problem size gained in discussions with other vendors of (non-spreadsheet-based) optimization software. In fact, we believe that the median-size industrial LP model is smaller than many OR/MS professionals might expect—possibly as small as 2,000 rows and columns. Spreadsheet optimizers can readily handle problems well above this size.

Model Developers

OR/MS professionals usually create optimization models in situations where the modeling task is challenging enough and the economic value of the problem is large enough to justify expert consulting help. These problems are often much larger than our median size estimate. But this is a tiny part of the spectrum of optimization applications that we see. Many spreadsheet models are straightforward, successful adaptations of classic forms, such as transportation, blending, multiperiod inventory, and portfolio-optimization problems. These models are created by functional managers who base them on the examples supplied with Excel or found in various books (indeed, such users often seek out

the textbooks that we feature on the Frontline Systems' Web site). In other cases, these spreadsheet optimization models are created by outside consultants with industry expertise, rather than OR/MS expertise per se.

OR/MS Training

Every day we see successful Solver applications created by spreadsheet users with little or no formal OR/MS training. Users of Frontline Systems' Premium Solver products are typically solving LP models in the range of several hundred to a few thousand (some as large as 10,000) decision variables and constraints and integer and nonlinear problems of somewhat smaller size. Although this group is self-selected for applications more ambitious than those built with the standard Excel Solver, we estimate that 90 to 95 percent of these users have no affiliation with the OR/MS community. They are clearly "dispersed practitioners" [Geoffrion 1991].

Yet this is just another layer of the iceberg. A much larger number of Excel Solver users visit Frontline Systems' World Wide Web site (www.frontsys.com), which receives more than 10,000 "hits" per day. We have some survey data on these users that indicate that a surprising number of Solver applications are below 200 variables in size but are of sufficient value that their developers are planning to distribute copies of these applications within their organizations or commercially. This survey data and our experience in technical support lead us to believe that this class of applications is at least five times and perhaps 10 times larger than the class of applications above 200 variables.

Still deeper in the iceberg are the smaller-size spreadsheet solver applications that are developed for use within only one department or office and not for redistribution. These users may well find that the standard Excel Solver, Microsoft technical support, and the variety of trade books about Excel meet all of their needs. We believe that this group is the largest of all, but we are unable to estimate its size. In any case, we are reasonably certain that OR/MS professionals collectively are involved in, at most, a fraction of one percent of the Excel Solver applications actually in use.

Economic Value

Small optimization models may yield high economic value. In one case, a Fortune 50 company (which prefers to remain anonymous) used the standard Excel Solver to build a purchasing logistics model used in negotiating contracts for over a billion dollars worth of a single commodity. This model, whose size was a function of the number of supplier locations and company plants, fit within the 200-variable limit of the standard Solver. Savings from use of this model amounted to nearly \$3 million in the first round of purchasing negotiations and the company estimates future savings of \$7 million per year. A major difference from the OR/MS successes of the 1970s was the time and effort required to formulate, test, and gain acceptance of this model. One individual, with no formal OR/MS training, completed the entire project in three person-months, with about one month spent on the actual optimization model. The resulting spreadsheet is operated directly by the senior vice president of purchasing. The

return on investment in such application projects is extremely high.

Use of the Solver in Education

Spreadsheets have become the preferred tool for teaching quantitative methods to undergraduate and graduate business students. Their use is strongly endorsed in a recent report of the operating subcommittee of the INFORMS Business School Education Task Force [Jordan et al. 1997]. In July 1994, the presidents of ORSA and TIMS, Dick Larson and Gary Lilien, chartered the INFORMS Business School Education Task Force in response to the decline of OR/MS content in business education that began in the early '90s. The task force's survey of business school OR/MS faculty (306 responses) revealed that many faculty members planned to increase their use of spreadsheets (Table 2) to strengthen the role of OR/MS in their MBA programs.

The subcommittee also conducted structured telephone interviews with program administrators at 21 of the leading MBA programs in the US. One of the questions they asked was "What particular sets of quantitative skills are in greatest demand from employers of your graduates?" The interpretation of responses was "Demand for particular 'hard' OR/MS skills is very low. Where technique is needed, it involves statistics more than OR/MS. There is demand for general skill in model formulation and interpretation and in quantitative reasoning." A related question was "What level of competence is appropriate for MBAs?" and it had the summarized response "MBAs need to be able to use spreadsheets and statistical software at the level of the 'educated consumer.' "

Questions and responses	%
Which of the following “fixes” have the highest potential to strengthen the role of MS/OR in your particular school of business?	
More use of cases and real-world examples	60
More emphasis on modeling skills and numeracy and less on algorithms	55
Better math background for students	49
Use of spreadsheets instead of special purpose OR/MS software	39
What changes are you planning to make in your MS/OR course in the near future?	
More emphasis on modeling and less on the teaching of algorithms	55
Increasing the role of the computer in the course	43
More use of spreadsheets in the course	37
More case analyses	34

Table 2: Two questions and the most often selected responses from the INFORMS Business School Education Task Force’s 1997 survey of business school OR/MS faculty (306 respondents).

The authors of the report conclude that OR/MS courses in business schools should focus on common, realistic business situations, acknowledge important nonmathematical issues, use spreadsheets, and emphasize model formulation and assessment more than model structuring. Recommendations include the following: embed analytical material strongly in a business context; use spreadsheets as a delivery vehicle for OR/MS algorithms; and stress the development of general modeling skills.

There are now strong trends in these directions, most of which began well before the INFORMS report appeared. They are most prominent in the form of new textbooks for the basic OR/MS course for undergraduate or graduate business students. Such texts include those of Hesse [1997], Ragsdale [1997], Winston and Albright [1997], a revision of the Eppen and Gould text [Eppen et al. 1998], and a book by Sam Savage [1997]. The authors use spreadsheet models as the focus around which they base all discussion and

examples. All use the Excel Solver for optimization, and several use spreadsheet add-ins for decision tree analysis and Monte Carlo simulation. All include a disk containing a complete set of spreadsheet files, bundled with the text and intended for student use, and an instructor’s disk or CD-ROM containing spreadsheets for each problem and case. Some contain a shell version of the instructor spreadsheets, in which the members and formulas are omitted. These greatly ease the instructor’s task of grading many spreadsheets, especially when he or she uses the grading macros that are provided for some problems.

In the introductions to these texts, the authors advocate a course based on learning modeling by doing examples. They include many traditional examples from the operations management area of business: production and inventory planning, distribution inventory models, and so forth. In addition, they include problems from finance (portfolio selection, options pricing, cash management), and marketing (sales-

force allocation). Problems in finance and marketing are often of more interest to MBA students than the traditional operations examples.

Outside the traditional OR/MS course, new texts are also appearing with a focus on spreadsheets. For example, the marketing textbook by Lilien and Rangaswamy [1997] includes 17 models in Excel, most using the Solver, controlled by the authors' programs written in Visual Basic for Applications.

For more on configuring a successful OR/MS course for business students see the articles by Bodily [1996] and Powell [1995] and many other articles in the "Teachers forum" section of *Interfaces*.

Conclusions and Directions for Future Work

We designed the Excel Solver to "make optimization a feature of spreadsheets." Where OR/MS professionals tend to see it as simply another tool for doing optimization, managers in industry tend to see it as an extension of spreadsheet technology that enables them to solve resource-allocation problems in a new way, in their own work groups, without outside help.

Our most important direction for future work is to extend the range of optimization problems that managers can solve without special OR/MS training or outside help. Classical linear and smooth nonlinear functions are too restrictive for many of the problems our users want to solve. The use of integer variables and special constraints to express such constructs as fixed charges and either-or conditions is unnecessarily complex for users; familiar spreadsheet functions, such as IF, CHOOSE, and LOOKUP (which may de-

pend on the variables), could be used to express these concepts directly. In the future, we would like to support the creation of optimization models using as much of the full power of the spreadsheet formula language as possible. To do this, we expect to perform more analysis and transformation of the spreadsheet formulas, obtaining the Jacobian matrix through a combination of automatic differentiation of the most common operators and functions and the selective use of finite differences for others. We are also considering approaches to global optimization and heuristics and algorithmic methods that yield good solutions that may not be provably optimal (for example, clustering methods, genetic algorithms, and simulated annealing), since our users have clearly indicated their interest in such methods.

Spreadsheets, such as Excel, have become so ubiquitous that they serve as a kind of lingua franca for quantitative models, understood by nearly every decision maker in industry, government, and education. Because of this universality, spreadsheet software has become an excellent delivery vehicle for such OR/MS techniques as optimization, as the Excel Solver clearly demonstrates. We encourage OR/MS professionals to gain experience with these tools and explore the world of spreadsheet-based problem solving that continues to grow outside the traditional boundaries of the field.

We also encourage OR/MS professionals to communicate with Microsoft and with Frontline Systems about their desires for the Excel Solver. E-mail is the preferred method: Microsoft welcomes feedback on the Solver and other Excel features sent to

xlwish@microsoft.com, while Frontline Systems welcomes feedback sent to info@frontsys.com. By making their voices heard, OR/MS professionals can influence the future direction of software such as the Excel Solver.

References

- Bodily, S. 1996, "Teaching MBA quantitative business analysis with cases," *Interfaces*, Vol. 26, No. 6, pp. 132–149.
- Brooke, A.; Kendrick, D.; and Meeraus, A. 1992, *GAMS, A User's Guide*, Boyd and Fraser, Danvers, Massachusetts.
- Chinneck, J. W. 1997, "Finding a useful subset of constraints for analysis in an infeasible linear program," *INFORMS Journal on Computing*, Vol. 9, No. 2, pp. 164–174.
- Conway, D. and Ragsdale, C. 1987, "Modeling optimization problems in the unstructured world of spreadsheets," *Omega*, Vol. 25, No. 3, pp. 313–322.
- Enfin Software Corporation 1988, *Optimal Solutions User Manual*, San Diego, California.
- Eppen, G. D.; Gould, F.; Schmidt, C.; Moore, J.; and Weatherford, L. 1998, *Introductory Management Science: Decision Modeling with Spreadsheets*, fifth edition, Prentice Hall, Englewood Cliffs, New Jersey.
- Fourer, R.; Gay, D. M.; and Kernighan, B. W. 1993, *AMPL: A Modeling Language for Mathematical Programming*, Duxbury Press, Pacific Grove, California.
- Frontline Systems Inc. 1990, *What-If Solver User Guide*, Incline Village, Nevada.
- Frontline Systems Inc. 1994, *Solver User Guide: Premium, Quadratic, and Large-Scale LP Solvers*, Incline Village, Nevada.
- Fylstra, D. 1992, *The Student Edition of What-If Solver*, Addison-Wesley Longman, Reading, Massachusetts.
- Geoffrion, A. M. 1991, "Forces, trends, and opportunities in management science and operations research," *Operations Research*, Vol. 4, No. 3, pp. 423–445.
- Gill, P. E.; Murray, W.; and Wright, M. H. 1981, *Practical Optimization*, Academic Press, San Diego, California.
- Griewank, A. and Corliss, G. F. 1991, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, SIAM Press, Philadelphia, Pennsylvania.
- Hesse, R. 1997, *Managerial Spreadsheet Modeling and Analysis*, Richard D. Irwin, Burr Ridge, Illinois.
- Jordan, E.; Lasdon, L.; Lenard, M.; Moore, J.; Powell, S.; and Willemain, T. 1997, "OR/MS and MBA's—Mediating the mismatches," *OR/MS Today*, February, pp. 36–41.
- Lasdon, L. S.; Waren, A. D.; Jain, A.; and Ratner, M. 1978, "Design and testing of a generalized reduced gradient code for nonlinear programming," *ACM Transactions on Mathematical Software*, Vol. 4, No. 1, pp. 34–49.
- Lasdon, L. S. and Smith, S. 1992, "Solving large sparse nonlinear programs using GRG," *ORSA Journal on Computing*, Vol. 4, No. 1, pp. 2–15.
- Lilien, G. and Rangaswamy, A. 1997, *Marketing Engineering: Computer-Assisted Marketing Analysis and Planning*, Addison-Wesley Longman, Reading, Massachusetts.
- Lotus Development Corp. 1990, *1-2-3/G User Guide*, Cambridge, Massachusetts.
- Ng, E. and Char, B. W. 1979, "Gradient and Jacobian computation for numerical applications," *Proceedings of the 1979 Macsyma User's Conference*, Washington, DC, pp. 604–621.
- Person, R. 1997, *Using Microsoft Excel 97*, Que Corp./Macmillan Computer Publishing, Indianapolis, Indiana.
- Powell, S. G. 1995, "Teaching the art of modeling to MBA students," *Interfaces*, Vol. 25, No. 3, pp. 88–94.
- Ragsdale, C. T. 1997, *Spreadsheet Modeling and Decision Analysis*, second edition, South-Western Publishing, Cambridge, Massachusetts.
- Savage, S. L. 1985, *What'sBest! User Manual*, General Optimization Inc., Chicago, Illinois.
- Savage, S. L. 1997, *INSIGHT Business Analysis Tools for Excel*, Duxbury Press, Pacific Grove, California.
- Winston, W. L. and Albright, S. C. 1997, *Practical Management Science: Spreadsheet Modeling and Applications*, Duxbury Press, Pacific Grove, California.