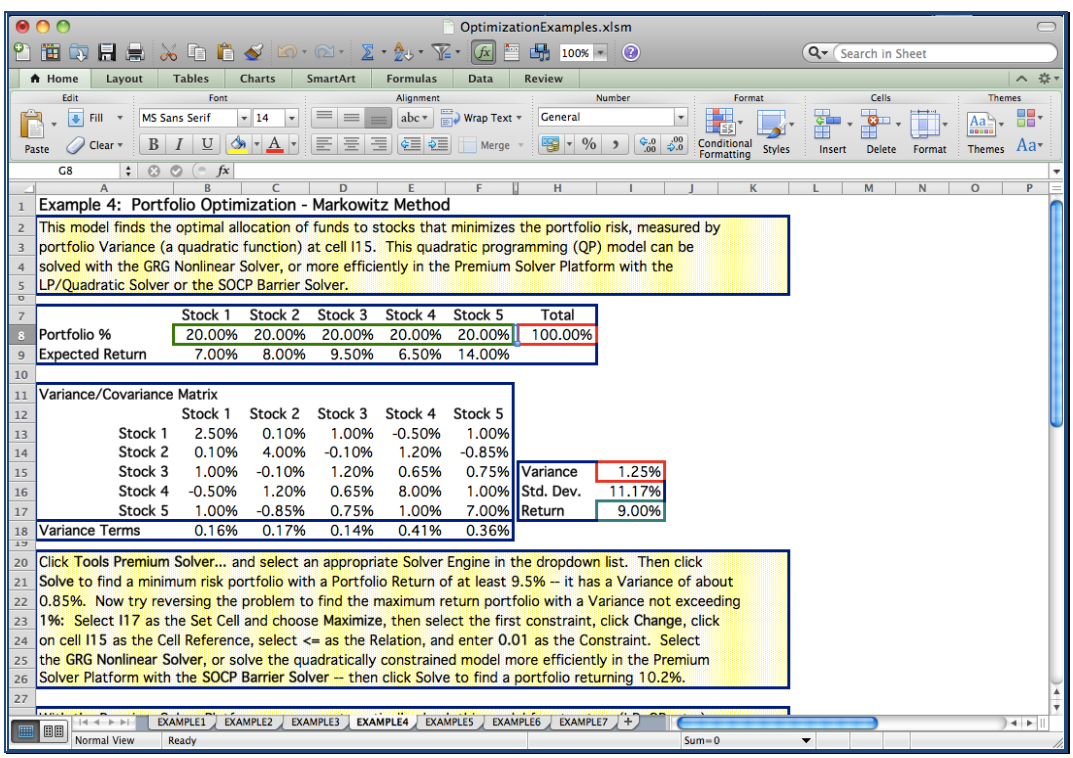
Version 10.5 **For Use With Excel 2011**

Premium Solver Platform for Mac

User Guide



FRONTLINE solvers

How to Order

Contact Frontline Systems, Inc., P.O. Box 4288, Incline Village, NV 89450.
Tel (775) 831-0300 • Fax (775) 831-0314 • Toll-Free (888) 831-0333
Email info@solver.com • Web http://www.solver.com

SOFTWARE LICENSE AND LIMITED WARRANTY

This is an agreement between Frontline Systems, Inc. ("Frontline") and the person or organization acquiring a license ("Licensee") to use the computer program products described in this User Guide (the "Software"), in exchange for Licensee's payment to Frontline. Licensee may designate the individual(s) who will use the Software from time to time, in accordance with the terms of this agreement. Unless replaced by a separate written agreement signed by an officer of Frontline, this agreement, including the Software License, Limited Warranty, and U.S. Government Restricted Rights sections below, shall govern Licensee's use of the Software; by accepting delivery of the Software or allowing Use of the Software, Licensee accepts all terms and conditions of this agreement, and agrees that this agreement supersedes the terms and conditions of any purchase order issued in connection with the license purchase.

"Use" of the Software means the use of any of its functions to define, analyze, solve (optimize, simulate, etc.) and/or obtain results for a single user-defined model. Use with multiple models at the same time, whether on one computer or multiple computers, requires either a Flexible Use License or multiple Standalone Licenses. Use occurs only during the time that the computer's processor is executing the Software; it does not include time when the Software is loaded into memory without being executed. The minimum time period for Use on any one computer shall be ten (10) minutes, but may be longer depending on the Software function used and the size and complexity of the model.

STANDALONE LICENSE

If Licensee pays for a Standalone License, Frontline grants to Licensee the right to Use the Software on one computer (the "PC") at a time, and will provide Licensee with a license code enabling such Use. The Software may be stored on one or more computers, servers or storage devices, but it may be Used only on the PC. Use of the Software may depend upon unique components of the PC, such as its hard disk ID or MAC address; in the event these components fail, Frontline will provide Licensee with a new license code, enabling Use with replacement components, at no charge. A Standalone License may be transferred to a different PC while the first PC remains in operation only if (i) Licensee requests a new license code from Frontline, (ii) Licensee certifies in writing that the Software will no longer be Used on the first PC, and (iii) Licensee pays a license transfer fee, unless such fee is waived by Frontline.

FLEXIBLE USE LICENSE

If Licensee pays for a Flexible Use License, Frontline grants to Licensee the right to Use the Software as described in this paragraph, and will provide Licensee with License Server software and a license code enabling such Use. For purposes of this agreement, a "Network" is a group of computers interconnected by any networking technology that supports the TCP/IP protocol or the IPX/SPX protocol. The Software may be (i) stored on one or more computers, servers or storage devices on the Network, (ii) accessed by and copied into the memory of other computers on the Network, and (iii) Used on any of the computers on the Network, provided that only one Use occurs at any one time. Licensee must install and run the License Server software on one of the computers on the Network (the "LS"); other computers will temporarily obtain the right to Use the Software from the License Server. Operation of the License Server may depend upon unique components of the LS, such as its hard disk ID or MAC address; in the event these components fail, Frontline will provide Licensee with a new license code, enabling operation of the License Server with replacement components, at no charge. The License Server software may be transferred to a different LS while the first LS remains in operation only if (i) Licensee requests a new license code from Frontline, (ii) Licensee certifies in writing that the License Server will no longer be run on the first LS, and (iii) Licensee pays a license transfer fee, unless such fee is waived by Frontline.

ADDITIONAL TERMS

This agreement does not grant to Licensee the right to make copies of the Software or otherwise enable use of the Software in any manner other than as described above, by any persons or on any computers except as described above, or by any entity other than Licensee. Licensee agrees that it will not rent or lease the Software, nor "share" use of the Software with anyone else, nor make the Software available over the Internet, a company or institutional intranet, or any similar networking technology, except as explicitly provided above in the case of a Flexible Use License. Licensee agrees that it will not attempt to alter or circumvent license control features of the Software or the License Server, nor reverse compile or reverse engineer the Software or the License Server. This agreement may be assigned to any entity that succeeds by operation of law to Licensee or that purchases all or substantially all of Licensee's assets (the "Successor"), provided that Frontline is notified of the transfer, and that Successor agrees to all terms and conditions of this agreement.

COPYRIGHT WARNING

LIMITED WARRANTY

Frontline Systems, Inc. ("Frontline") warrants that the CD-ROM, diskette or other media on which the Software is distributed and the accompanying User Guide (collectively, the "Goods"), but not the digital or printed content recorded thereon, is free from defects in materials and workmanship under normal use and service for a period of ninety (90) days after purchase, and any implied warranties on the Goods are also limited to ninety (90) days. SOME STATES DO NOT ALLOW LIMITATIONS ON THE DURATION OF AN IMPLIED WARRANTY, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU. Frontline's entire liability and your exclusive remedy under this warranty shall be, at Frontline's option, either (i) return of the purchase price or (ii) replacement of the Goods that do not meet Frontline's limited warranty. You may return any defective Goods under warranty to Frontline or to your authorized dealer, either of which will serve as a service and repair facility.

If you purchase an Annual Support Contract from Frontline, then Frontline warrants, during the contract term, that the Software will perform substantially as described in the User Guide, when it is properly used as described in the User Guide. Frontline's entire liability and your exclusive remedy under this warranty shall be to make reasonable commercial efforts to correct any "bugs" (failures to perform as so described) reported by you, and to timely provide such corrections in the Software to you. If you do not purchase an Annual Support Contract from Frontline, or if you allow your Annual Support Contract to expire, then THE SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND.

Whether or not you purchase an Annual Support Contract from Frontline, you understand and agree that any results obtained through your use of the Software are entirely dependent on your design and implementation of an optimization or simulation model, for which you are entirely responsible, even if you seek advice on modeling from Frontline. You understand and agree that THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE AS USED WITH YOUR OPTIMIZATION OR SIMULATION MODEL IS ASSUMED BY YOU.

EXCEPT AS PROVIDED ABOVE, FRONTLINE DISCLAIMS, AND WITHOUT EXCEPTION ITS SUPPLIERS DISCLAIM ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE. THIS WARRANTY GIVES YOU SPECIFIC RIGHTS, AND YOU MAY HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

IN NO EVENT SHALL FRONTLINE OR ITS SUPPLIERS HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE SOFTWARE OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU. In states that allow the limitation but not the exclusion of such liability, Frontline's and its Suppliers' liability to you for damages of any kind is limited to the price of one copy of the Goods and one Standalone License to use the Software.

U.S. GOVERNMENT RESTRICTED RIGHTS

The Software and Media are provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the Government is subject to restrictions as set forth in subdivision (b)(3)(ii) of The Rights in Technical Data and Computer Software clause at 252.227-7013. Contractor/manufacturer is Frontline Systems, Inc., P.O. Box 4288, Incline Village, NV 89450.

THANK YOU FOR YOUR INTEREST IN FRONTLINE SYSTEMS' PRODUCTS.

Contents

Introduction

Using Premium Solver Platform for Mac

Thank you for using Premium Solver Platform Version 10.5, Frontline Systems' newest and most powerful Solver product for Microsoft Excel for Mac 2011. Premium Solver Platform is a fully compatible upgrade for the Excel Solver for Mac, which was developed by Frontline Systems for Microsoft. This Guide covers all the features of Premium Solver Platform and the four "Solver engines" bundled with it: the standard LP/Quadratic Solver, standard SOCP Barrier Solver, standard GRG Nonlinear Solver, and standard Evolutionary Solver.

Note: Premium Solver Platform for Mac requires Excel 2011 to work. It will not work with Excel for Mac 2008, for example, as Excel for Mac 2008 doesn't support either VBA or Apple's Scripting Bridge technology which are required for Premium Solver Platform for Mac to work with Excel.

Before reading this User Guide, you may find it helpful to read the Solver-related topics in the online Help supplied with Microsoft Excel. These topics document the standard Solver's features and take you through the basic steps of using the Solver. We recommend that you try out the standard Solver on at least one problem of your own, or on one or more of the examples in the SOLVSAMP.XLS workbook which comes with Microsoft Excel.

This Guide goes well beyond the basics covered in the Microsoft Excel Help system. Premium Solver Platform can solve far larger versions of the problems handled by the standard Excel Solver, and new kinds of problems using conic and global optimization, non-smooth functions, and new types of constraints. And Premium Solver Platform can analyze and interpret your model in ways not possible with Microsoft Excel alone. This Guide will help you set up and solve much larger Solver problems, design your models for the fastest solutions, and understand the results of analyzing and solving your model – solutions, messages, and reports.

What's New in Version 10.5

Premium Solver Platform for Mac V10.5 is a major new release, designed to analyze and solve *much larger* models, *faster* than ever before. Version 10.5 is designed to work specifically with Microsoft Excel for Mac 2011 which re-introduced VBA in Excel for Mac and also includes Apple's Scripting Bridge technology, both required by Premium Solver Platform for Mac.

Solving Large Scale, Multi-Worksheet Models

Models Defined Across Multiple Worksheets

The Premium Solver Platform V10.5 supports Solver models spread across multiple worksheets in a workbook. It is not necessary to keep all of your decision variables and constraint left hand sides on the active worksheet. Yet you can still define a different Solver model (if desired) on each worksheet – and each of these models can include variables and constraints on any sheet in the workbook! You can still use the Load Model and Save Model buttons to create as many sets of model specifications as you like.

Worksheets of 16K Columns and 1 Million Rows

When used with Excel for Mac 2011, Premium Solver Platform for Mac V10.5 supports worksheets with up to 16,384 columns and 1,048,576 rows – far beyond the limits of 256 columns and 65,536 rows in previous versions of Excel. This makes it much easier to lay out your models on a worksheet, without having to split up large tables of information. Many other limits, such as the maximum length of labels and formulas, are also greatly increased in Excel for Mac 2011.

Speeding Up Analysis, Solving and Reporting

Many of the built-in Solvers and field-installable Solver Engines in Version 10.5 feature improvements in the speed of solution. But Premium Solver Platform for Mac V10.5 concentrates on speeding up "end-to-end solution time," which includes setup time, report preparation and report generation time.

Reporting Multiple Solutions from Optimization

For global optimization, non-smooth optimization, and mixed-integer programming problems, the solution process used by most Solver engines finds several candidate solutions – for example, locally optimal solutions in searching for a globally optimal solution, or feasible integer solutions with good objective values ("incumbents") for an integer programming problem. In Premium Solver Platform for Mac V10.5, the full range of built-in and plug-in Solver engines support the new Solutions Report, which lists objective and decision variable values for each of these candidate solutions (the *best* solution is 'plugged in' to the decision variable cells in your model, as usual).

Full Model Compatibility Across Platforms

Compatibility with the Standard Excel Solver

Premium Solver Platform for Mac is designed to be fully upward compatible with the Standard Excel Solver for Mac. Your models and any existing VBA code will work as-is.

Compatibility with our Windows Products

Models build in Premium Solver Platform for Mac will also work without any changes in our full line of optimization products for Windows. If you have previously

built a model in a Windows version or need to share a model you've built on a Mac version you may do either with no changes to the model required.

A Brief Tour of New Features

Premium Solver Platform for Mac provides a wide range of new features for Solver users, including the ability to solve entirely new kinds of problems vs. the standard Excel Solver for Mac. This section provides a tour of these new features, with brief explanations of what they mean for your ability to create models and find optimal solutions.

Model Analysis: The Polymorphic Spreadsheet Interpreter

The Excel Solver relies on Microsoft Excel itself to read and analyze ("parse") the formulas you enter in spreadsheet cells, and calculate values for ("interpret") these formulas whenever the Solver changes values of input cells.

Premium Solver Platform for Mac includes a new Polymorphic Spreadsheet Interpreter (*PSI* technology) for Excel formulas, developed by Frontline Systems. The term "polymorphic" has much the same meaning as it does in programming languages such as C++ and Java, but for Microsoft Excel formulas. The Interpreter does much more than simply recalculate values for Excel formulas – it can interpret the formulas in many other ways that are advantageous for the Solver.

The Interpreter can handle nearly any Microsoft Excel formula syntax, including array formulas, and almost all of the Excel built-in functions, including the financial, statistical, and engineering functions in the Analysis Toolkit

Automatic Model Diagnosis

The Polymorphic Spreadsheet Interpreter can *diagnose* your model by evaluating your Excel formulas with an 'overloaded' type for each cell, operator and function. It determines which input cells are decision variables, and which are constant in the model; then it evaluates each arithmetic operation or function in your model to determine how the formula depends on each decision variable: Whether it is independent (i.e. constant), linear, quadratic, smooth nonlinear, or non-smooth as a function of that variable.

This enables the Interpreter to diagnose your model as a linear programming, quadratic programming, conic programming, smooth nonlinear, or non-smooth optimization model. Further, you can tell the Interpreter that your goal or intent was to create (say) a *linear* model, and the Interpreter will pinpoint the specific cells containing formulas that create a *nonlinear* relationship in your model. Similarly, if you intended to create a smooth nonlinear model, the Interpreter will pinpoint cells containing non-smooth operations or functions of each decision variable.

The Interpreter can also help diagnose problems of poor scaling in your model: It can evaluate your formulas while keeping track of the magnitude of each intermediate result, and pinpoint the formulas that are likely to yield a loss of accuracy due to poor scaling, that cannot be handled via automatic rescaling in the Solver engines.

Automatic Tests for Convexity

Once your model goes beyond linear programming to include quadratic or nonlinear functions, it may remain 'easy' or it may become very difficult to solve. If your model is **convex**, it can be solved quickly and reliably to a globally optimal solution, even if it grows very large. But if your model is **non-convex**, you'll find that Solvers (of all types) can find only a locally optimal solution, and may even have trouble finding a feasible solution – and the time taken to find a solution may be so long that it limits the size of model you can solve. But most users have found it difficult or impossible to *determine whether* their nonlinear model is convex.

Premium Solver Platform for Mac V10.5 includes a ***unique, automatic test for convexity*** of your model and its objective and constraints in Excel, based on pioneering work by Frontline Systems developers. The convexity test is not always conclusive, because a conclusive test would take time exponential in the number of variables. But in many cases, with Premium Solver Platform for Mac you can determine whether your model is convex, and identify specific functions that make your model non-convex, by pressing a button.

Automatic Transformation of Non-Smooth Models

As described in the chapter "Solver Models and Optimization," using even one non-smooth function (such as IF, MIN, MAX, ABS, AND, OR, or NOT, with arguments that depend on the decision variables) in a model that is otherwise linear (using functions such as SUM and SUMPRODUCT) changes the model from a *linear programming* (LP) problem to a *non-smooth optimization* (NSP) problem.

Thanks to the Evolutionary Solver in Premium Solver Platform for Mac, you can still solve such models. But the consequences of such non-smooth functions for the Solver are considerable: Where an LP can be solved very quickly and reliably up to very large size, and the solution is basically guaranteed to be optimal, an NSP takes far more time to solve, requires inherently less reliable methods, and there are no guarantees as to whether the solution is truly optimal.

In **Version 10.5**, Premium Solver Platform for Mac can *automatically transform* your model, replacing IF, MIN, MAX, ABS, AND, OR, and NOT functions and <= and >= operators with additional variables and linear constraints that achieve the same effect, for optimization purposes, as these functions. If all non-smooth functions in your model can be transformed, the result will be a linear mixed-integer (LP/MIP) model that can be solved by a variety of Solver engines, from the standard LP/Quadratic Solver to the Gurobi Solver – giving you a better chance of finding an optimal solution with certainty, in a reasonable amount of time.

Automatic Differentiation

Most Solvers make heavy use of derivatives or gradients of the problem functions (the objective and constraints) with respect to the decision variables. Linear programming algorithms require that derivatives be evaluated once, to obtain the LP coefficient matrix. Nonlinear optimization algorithms typically require that derivatives be evaluated many times, once at each major iteration or trial point. Hence, derivative evaluation is key to both the speed and accuracy of such optimization algorithms.

The Excel Solver estimates derivative values by the method of *finite differencing*: It uses Microsoft Excel itself to recalculate values for the objective and constraints at the "current point" (i.e. values of the decision variables), and at nearby points with small changes ("perturbations") in each decision variable. This process, while often

adequate, is relatively slow and inaccurate – it takes many recalculations and may lose significant digits as it performs many division operations.

Premium Solver Platform's Polymorphic Spreadsheet Interpreter can compute derivatives directly, by evaluating your Excel formulas with an overloaded 'gradient' type for each cell, operator and function, using the methods of *automatic differentiation*. Analytic formulas for the derivatives are applied to each arithmetic operator and elementary function, and the chain rule is applied to compute derivatives for composite functions. Hence, derivative values can be obtained many times faster, and without any loss of accuracy beyond the actual function values themselves.

The Interpreter goes further to support the new SOCP Barrier Solver, MOSEK Solver, and KNITRO Solver in **Version 10.5**: It computes the *Hessian* (matrix of second order derivatives) of each problem function using the methods of automatic differentiation. The method of finite differencing is far too slow and inaccurate for this purpose – the Polymorphic Spreadsheet Interpreter makes such new methods and Solver engines practical in Microsoft Excel.

Multistart Methods for Global Optimization

As explained in the chapter "Solver Models and Optimization," the nonlinear GRG Solver – like virtually all "classical" nonlinear optimizers – will find only a *locally optimal* solution to a **non-convex** problem. Imagine a graph of the objective function with "hills" and "valleys:" The GRG Solver will typically find the peak of a hill near the starting point you specified (if maximizing), but it may not find an even higher peak on another hill that is far from your starting point. In some problems this is sufficient, but in other cases you may want to find a *globally optimal* solution.

With multistart methods in Premium Solver Platform for Mac, the nonlinear GRG Solver can be automatically run many times from judiciously chosen starting points, and the best solution found (the "highest peak" if maximizing) will be returned as the optimal solution. An algorithm called "multi-level single linkage" randomly samples starting points, collects them into "clusters" that are likely to lead to the same locally optimal solution, and runs the GRG Solver from a representative point in each cluster. This process continues until a Bayesian statistical test estimates that all locally optimal solutions have likely been found. Then the best of these solutions is returned as the 'probable globally optimal' solution.

The Evolutionary Solver

The Evolutionary Solver provides an alternative to multistart methods to seek a globally optimal solution to a **non-convex** problem, even if all the problem functions are smooth. Rather than search in the neighborhood of a single starting point, it maintains a population of candidate solutions "scattered around the landscape," and (based in part on random choices) it will attempt to improve each one.

The Evolutionary Solver is based on the principles of "genetic algorithms" and "evolutionary algorithms." In tests on a wide variety of Excel models, it outperforms competitive products whose main (or only) feature is a genetic algorithm, by finding better solutions in significantly less time. And the Evolutionary Solver doesn't require that you learn new terminology or choose from a variety of complicated "solving methods." You just select the Evolutionary Solver engine and click Solve.

A Solver based on genetic or evolutionary algorithms is not a panacea, however. Unlike the Simplex and GRG Solvers which are *deterministic* optimization methods, the Evolutionary Solver is a *nondeterministic* method: Because it is based partly on

random choices of trial solutions, by default it will often find a different "best solution" each time you run it, even if you haven't changed the model at all. And unlike the Simplex and GRG Solvers, the Evolutionary Solver has no way of knowing for certain that a given solution is optimal – even "locally optimal." Similarly, the Evolutionary Solver has no way of knowing for certain whether it should stop, or continue searching for a better solution. With Premium Solver Platform for Mac, however, you are not limited to a genetic algorithm – you can apply the most appropriate Solver engine to each problem you encounter.

Hybrid Evolutionary Solver Methods

The Evolutionary Solver in Premium Solver Platform for Mac is actually a hybrid of genetic and evolutionary algorithms and classical optimization methods, including gradient-free 'direct search' methods, classical gradient-based quasi-Newton methods, and even the Simplex method for linear subsets of the constraints. The classical methods sometimes yield rapid "local improvement" of a trial solution, and they also help to "solve for" sets of constraints. The Evolutionary Solver also includes new "filtered local search" methods that greatly improve performance on smooth global optimization problems; and new "integer heuristic" methods from the local search literature that improve performance on problems with integer variables.

Working with the Polymorphic Spreadsheet Interpreter, the Evolutionary Solver can automatically apply genetic algorithm methods to the *non-smooth* parts of a problem, and apply classical methods to the *smooth nonlinear* and *linear* parts of the problem. The Evolutionary Solver is often able to solve problems with hundreds of constraints, which are typically beyond the capabilities of genetic and evolutionary algorithms working alone.

The SOCP Barrier Solver

Premium Solver Platform for Mac **Version 10.5** includes a new, built-in Solver engine, the Standard SOCP Barrier Solver. This Solver finds optimal solutions for second-order cone programming (SOCP) problems, which are a superset of linear programming (LP), quadratic programming (QP), and quadratically constrained programming (QCP) problems, with up to 2,000 decision variables. It supports the new *second order cone* (SOC) constraints in Premium Solver Platform for Mac (see below). To use the SOCP Barrier Solver, you simply select it from the dropdown list of Solver engines, and click Solve – no changes to your model are necessary.

The term "Barrier" comes from the optimization algorithms used by this Solver engine. Where the LP/Quadratic Solver uses the Simplex method, augmented for quadratic objectives, the SOCP Barrier Solver uses a Barrier method, also called an Interior Point method. Where the Simplex method's trial solutions are always at 'corners' on the surface of the feasible region, the Barrier method's trial solutions are always in the *interior* of the feasible region, until the final steps where the optimal solution is reached. Where the number of Simplex iterations typically grows with the number of constraints, the number of Barrier iterations is *independent* of the number of constraints, and is usually between 10 and 50 (but the time taken per iteration grows with problem size).

Second Order Cone Constraints

Premium Solver Platform for Mac **Version 10.5** supports a new type of constraint, called a second-order cone (SOC) constraint. An SOC constraint is created like any other constraint, by clicking the Add button to display the Add Constraint dialog,

selecting a range of decision variable cells for the Cell Reference or left hand side, and selecting **soc** or **src** (rotated second order cone) from the Relation dropdown list. This specifies that the vector formed by the *n* decision variables must lie in the second order cone (also called the Lorentz cone or "ice cream" cone) of dimension *n*.

A linear programming problem plus one or more SOC constraints defines a **second order cone programming (SOCP)** problem. All "ordinary" non-negative decision variables also belong to a cone, called the *non-negative orthant* – hence a linear programming (LP) problem is a special case of a conic programming problem, where the only cone constraint is non-negativity. Second order cone programming is the *natural generalization of linear programming*: It includes all quadratic programming (QP) and quadratically constrained programming (QCP) problems, and many other problems in quantitative finance and engineering design. SOCP problems are always **convex**, and they can be solved quickly and reliably to very large size.

Since the Polymorphic Spreadsheet Interpreter in Premium Solver Platform for Mac supports second order cone constraints, SOCP problems can be solved by either *special-purpose*, high-performance Solvers (the SOCP Barrier Solver and MOSEK Solver), or by *general-purpose* nonlinear Solvers such as the standard GRG Solver, the Large-Scale GRG and SQP Solvers, and the KNITRO Solver engine.

Alldifferent Constraints

At times, you'll encounter a problem where you want to specify that a set of integer variables (typically representing an ordering of choices) *must all be different* at the solution. An example is the Traveling Salesman Problem (TSP), where a salesman must choose the order of cities to visit so as to minimize travel time, and each city must be visited exactly once. This condition is difficult to model using conventional constraints and integer variables.

In Premium Solver Platform for Mac, you can specify directly that a set of variables must be "alldifferent." Such variables will then have integer values from 1 to N (the number of variables), all of them different at the solution. All of the bundled Solver engines support this new type of constraint: The Branch & Bound method used by the Simplex LP (or LP/Quadratic), SOCP Barrier, nonlinear GRG, and the Evolutionary Solver implements these constraints using mutation and crossover operators for permutations. Field-installable Solver engines also support the alldifferent constraint, implementing it in different ways. This allows you to model your problem in a high-level way, and try a variety of Solver engines to see which one yields the best performance on your problem.

New Types of Reports

The standard Excel Solver offers six types of reports – the Answer Report, Sensitivity Report, Limits Report, Linearity Report, Feasibility Report, and Population Report. In addition to these reports, Premium solver Platform for Mac also includes Block Selection and Comments, Solutions Report, Scaling Report, Structure/Convexity Report, and Transformation Report. You can also select automatic outlining for the first six reports, which will organize the variables and constraints into outlined groups corresponding to the blocks you entered in the Solver Parameters dialog. You can identify each block of variables and constraints with descriptive comments. This can make it much easier to find the information you need in the reports, for models with hundreds or thousands of variables and constraints.

Feasibility, and Population Reports

When you receive the message that "Solver could not find a feasible solution," this often means that you've made a mistake entering some constraint, such as using a >= relation when you meant <=. But it can be difficult to pinpoint the source of the error, especially if you have hundreds or thousands of constraints to examine. With Premium Solver Platform For Mac, you can produce a Feasibility Report and let the Solver do the work. It will automatically re-solve the problem with subsets of the original constraints, until it isolates a subset of constraints (called an "Irreducibly Infeasible System" or IIS) which is infeasible, but which becomes feasible if any one of the constraints is removed. By examining just the constraints in the Feasibility Report, you can usually pinpoint the problem with your model very quickly.

When the Evolutionary Solver stops with a "best solution," you have the option of producing a standard Answer Report and/or a new Population Report. Where the Answer Report gives you detailed information about the single "best solution" returned by the Solver, the Population Report gives you summary information about the entire population of candidate solutions at the end of the solution process. The Population Report can give you insight into the performance of the Evolutionary Solver as well as the characteristics of your model, and help you decide whether additional runs of the Evolutionary Solver are likely to yield even better solutions.

Solutions, Scaling, Structure, and Transformation Reports

The **Solutions Report** gives you objective function and decision variable values for a number of alternative solutions found during the optimization process. For mixed-integer problems, the report shows each 'incumbent' or feasible integer solution found by the Branch & Bound method. For global optimization problems solved with the GRG, LSGRG, LSSQP, and KNITRO Solver engines, the report shows each locally optimal solution found by the Multistart method. For the Evolutionary and OptQuest Solvers, the report shows members of the final population of solutions.

Using Premium Solver Platform for Mac's new **Solver Model** dialog, which controls the Polymorphic Spreadsheet Interpreter, you can diagnose and transform your model before you solve it, produce the Structure Report to pinpoint problems in your model, and produce the Transformation Report to show how certain problematic functions were automatically replaced with "better" functions.

For the **Structure Report**, you simply select the type of model you meant to create – linear, quadratic, smooth nonlinear, or non-smooth – and ask the Solver to report any exceptions to this desired model type. The Structure Report is both more useful and more reliable than the Linearity Report, because it evaluates your model symbolically rather than numerically (so it cannot be "fooled" by poorly scaled models), and it can pinpoint not just overall constraints and variables, but individual cell formulas where the dependence of constraints on variables is nonlinear (if your assumed model is an LP) or non-smooth (if your assumed model is an NLP). These cell formulas are reported as "exceptions" in the Structure Report, with hyperlinks to the actual cells containing the formulas in question. In Premium Solver Platform for Mac, the Structure Report also tells you whether your objective and each constraint are **convex** or **non-convex** functions, when you ask the Interpreter to check the model for convexity.

The **Transformation Report** in Premium Solver Platform for Mac can be produced when you ask the Interpreter to automatically transform your model to replace non-smooth functions such as IF, MIN, MAX, ABS, AND, OR, or NOT with additional variables and linear constraints that have the same effect as the replaced functions. This report lists the new variables and constraints that are added to your model by the

transformation process. See the chapter "Analyzing and Solving Models" for more details.

The **Scaling Report** can be available when the Polymorphic Spreadsheet Interpreter evaluates all Excel formulas in your model while keeping track of the magnitudes or scales of intermediate results, and reports cases that may lead to a loss of accuracy.

User Interface Improvements

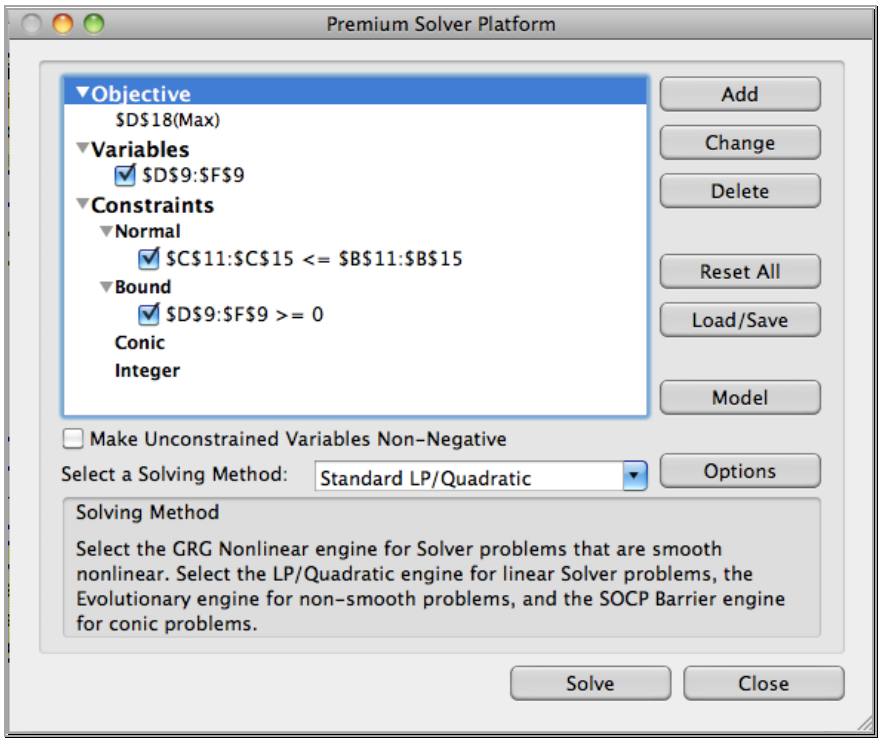
User interface for Premium Solver Platform for Mac is designed to give you more information, help you move more quickly through the Solver dialogs, and accomplish what you want in fewer steps.

Have you ever wondered whether your model was truly linear or smooth nonlinear, or (if it wasn't) exactly which formulas caused the model to be nonlinear or non-smooth? Or – having learned about how the large-scale Solver engines exploit sparsity in a model – have you wondered just how sparse *your* model actually is, or whether it could be solved faster – for example, by the Evolutionary Solver – by exploiting linear constraints or linearly occurring variables wherever possible? The Solver Model dialog in Premium Solver Platform for Mac gives you answers to all these questions.

Have you ever wondered about the size of the problem you've defined, and whether it's getting close to the size limits supported by a given Solver engine? In Premium Solver Platform For Mac, you can examine the number of variables, constraints, variable bounds and integer variables in your problem and the corresponding size limits at any time, by displaying the Problem tab in the Solver Options dialog available for each external Solver engine.

Have you ever had trouble remembering the purpose of a block of constraints in your model? In Premium Solver Platform for Mac, you can add descriptive comments to each block, which will appear in each of the Solver's reports.

Have you ever wanted to see more of the constraints at one time in the Constraints List box in the Solver Parameters dialog? In Premium Solver Platform for Mac, simply use your mouse to resize this dialog like any other window – as shown in the example on the next page.



If you solve an integer problem and find that there is no feasible solution, the Solver Results dialog provides an option to immediately solve the "relaxation," temporarily ignoring the integer constraints. And if there is still no feasible solution, you'll have the option to create a Feasibility Report, to find out why.

Have you ever found it difficult to enter all of your variables (Changing Cells) via a multiple selection in the single edit box provided by the standard Solver? Premium Solver Platform for Mac allows you to enter and modify an unlimited number of variable selections.

Finally, Premium Solver Platform for Mac provides access to algorithmic methods and tolerances used in each of the bundled Solver engines. By simply clicking on check boxes and radio buttons, or entering values in edit boxes in the Solver Options dialogs, you can control key tolerances in the Simplex method, stopping conditions for the nonlinear GRG Solver, the number of sub-problems and integer solutions to be explored by the Branch & Bound method, the population size, mutation rate, and other options for the Evolutionary Solver, and interior point methods and tolerances used by the SOCP Barrier Solver.

Speed Improvements

Premium Solver Platform for Mac offers a range of speed and accuracy improvements over the standard Excel Solver. Premium Solver Platform realizes a whole new level of speed and accuracy, compared to Excel Solver thanks to model analysis and automatic differentiation performed by the new Polymorphic Spreadsheet Interpreter.

Premium Solver Platform for Mac

In Premium Solver Platform for Mac, *fast problem setup* is still available for LP and QP models in restricted format, but the Polymorphic Spreadsheet Interpreter can speed up problem setup for virtually all models, regardless of the Excel formulas and functions they use.

The Polymorphic Spreadsheet Interpreter also greatly speeds up the solution process. Although solution times vary from model to model, on a comparative test of actual user LP models, Premium Solver Platform for Mac was *up to 20 times* faster than the Excel Solver on average. And with the Interpreter's automatic differentiation facilities, on a comparative test of actual user NLP models, Premium Solver Platform was *seven to 15 times faster* than the Excel Solver on average!

On LP/MIP models, speedups are even more dramatic, thanks to powerful branching and cut generation methods in Premium Solver Platform's LP/Quadratic Solver. Although solution times vary greatly, hundreds of times faster than the standard Excel Solver would not be unusual.

Even more dramatic is the effect of the SOCP Barrier Solver in Premium Solver Platform for Mac **Version 10.5** on problems with quadratic constraints that formerly required the GRG Nonlinear Solver, and on problems with second order cone constraints that could only be expressed with nonlinear analytic formulas previously: The SOCP Barrier Solver solves these nonlinear problems *nearly as fast as linear problems* of equivalent size!

Programmability Improvements

Premium Solver Platform for Mac is fully programmable from Excel's Visual Basic Application Edition. This means that you can build an application using the GRG Nonlinear Solver, Simplex LP (or LP/Quadratic) Solver, Evolutionary Solver, SOCP Barrier Solver, or a field-installable Solver engine, hide Premium Solver Platform user interface, and present your own customized user interface for your end users. A complete summary of the VBA functions supported by both the standard Excel Solver and Premium Solver Platform For Mac is included in this Guide.

Premium Solver Platform for Mac also provide programmatic access to new features such as the Variables list, the Solver Model dialog, the seven new types of reports, report outlining, and new options in the Solver Options dialogs for all Solver engines. Since the programmatic interface is upward compatible with the standard Excel Solver, you can use your existing VBA code, and extend it as much as you wish to utilize the new Premium Solver features.

How to Use This Guide

"Installation" takes you through the simple steps required to install Premium Solver Platform for Mac to work with your copy of Microsoft Excel. In Premium Solver Platform for Mac, installation and licensing is easier and more flexible than ever: You can install Premium Solver Platform for Mac without first installing the standard Excel Solver, share a Flexible Use license over a network, and add new license codes in Excel while the Solver is running, without re-running the Setup program.

"Solver Models and Optimization" – reviews the basic framework of the optimization problems that can be handled by the Solver, from linear programming problems to the non-smooth optimization problems handled by the Evolutionary Solver, and the use of new cone constraints and alldifferent constraints. It describes how a model is

made up of variables, constraints and an objective, and it covers the major types of optimization problems that you can solve – including **convex** and **non-convex** problems – and the tradeoffs involved.

"Building Solver Models" – provides an introduction to the art of building optimization models in Microsoft Excel, translating from algebraic notation to spreadsheet formulas and Solver Parameters dialog choices. It covers multiple selections for decision variables, use of the new Variables button, the possible forms of constraint left- and right-hand sides, use of the soc and src dropdowns for cone constraints, and use of the dif dropdown for alldifferent constraints. It also provides hints on how to build more readable, better-documented models, such as layout and formatting and use of defined names. It also describes models with variables and constraints spread across multiple worksheets.

"Analyzing and Solving Models" – describes how to use the new Solver Model dialog to analyze your model for linear, quadratic, smooth nonlinear, and non-smooth constraints and decision variables, automatically transform your model to replace non-smooth functions with additional variables and linear constraints, automatically find nonlinear or non-smooth formulas that are "exceptions" to your desired model type, and control the operation of the Polymorphic Spreadsheet Interpreter when your model is solved.

"Building Large-Scale Models" – provides a number of valuable hints when building large-scale spreadsheet models and using external data sources. It describes modeling techniques such as ratio constraints, fixed-charge constraints, either-or constraints, constraints for IF functions, piecewise linear constraints, and more.

"Diagnosing Solver Results" – 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. This chapter covers in some detail the strengths and limitations of the GRG Solver for smooth nonlinear problems, the multistart methods,, the Evolutionary Solver for non-smooth problems, and Simplex versus interior point methods for linear, quadratic, and conic optimization problems. It also provides hints on "what to do next" when you have a solution.

"Solver Options" – documents in depth the advanced options and tolerances used by each of the bundled Solver engines – including the new SOCP Barrier Solver – which can be set using new multi-tabbed Solver Options dialogs. The effect of each option and situations where you would likely choose it are described.

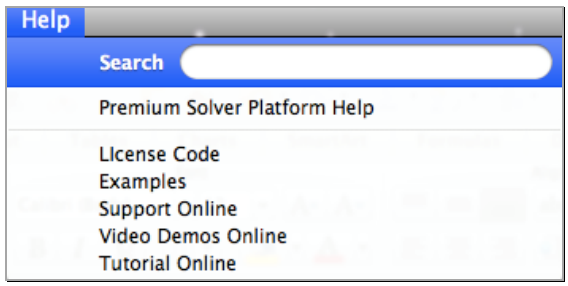
"Solver Reports" – describes the contents of reports that may be chosen from the Solver Results dialog. It shows you how to interpret the numbers in the reports, and how to use the Sensitivity Report to predict changes in the optimal solution in response to certain kinds of changes in your input data. This chapter also provides a detailed look at the new Population Report, Feasibility Report and the new **Solutions Report**, Scaling Report, Structure Report, and Transformation Report in Premium Solver Platform for Mac.

"Using VBA Functions" – describes how you can control the Solver using the VBA functions, upward compatible from the standard Excel Solver.

Using Online Help

The enhanced Help system included with Premium Solver Platform for Mac gives you online access to much of the information in this Guide. To access this

information, from within Premium Solver Platform just click on the Help menu item and you will see a drop down menu with available help resources.



If you're just getting started with Premium Solver Platform for Mac, **we highly recommend that you click the "Examples" link**, which will open the OptimizationExamples.xlsm workbook that is installed with the other Solver files. This workbook contains seven worksheets with examples illustrating many new Premium Solver Platform for Mac features, including conic optimization and automatic transformation of a model with IF functions into an equivalent model with integer variables and linear constraints.

Clicking the **Support Online menu item** will open a Web browser to our main Solutions page on **www.solver.com**, where you can download a series of additional example workbooks in Finance, Investment, Production, Distribution, Purchasing, and Scheduling. Clicking the **Tutorial Online** link will open a Web browser to the start of our highly regarded online tutorial about optimization.

You'll find a wealth of other useful information about the standard Solver and Premium Solver Platform For Mac at Frontline Systems' Web site, **www.solver.com**. Since Solver.com is frequently updated, you'll want to check it periodically for the latest news about the Solver.

If you have questions, check the Index in this Guide or in the online Help system. You can also submit questions via the Contact Us page on Solver.com, or send email to **info@solver.com**.

Solver-Related Seminars and Books

Although this Guide 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, smooth nonlinear and non-smooth optimization, 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 that 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!

You may also want to attend a public seminar on spreadsheet optimization and other advanced techniques in Excel. Because of the popularity of the Excel Solver, seminars taught by highly regarded instructors are offered in a variety of U.S. cities and other parts of the world. For the latest information on these seminars, visit **www.solver.com** or contact us at **info@solver.com**.

The Art of Modeling with Spreadsheets: Management Science, Spreadsheet Engineering, and Modeling Craft by Stephen G. Powell and Kenneth R. Baker, published by John Wiley & Sons, ISBN 978-0-470-53067-2. This new textbook is a valuable aid for anyone using Premium Solver Platform, especially for users building large-scale models. Unlike any other current textbook, this book teaches you "best practices" in modeling and spreadsheet engineering, as well as techniques of linear and nonlinear optimization, Monte Carlo simulation, and data analysis using Excel. Risk Solver Platform for Education for Windows is included free as a download with this book..

Spreadsheet Modeling and Decision Analysis: A Practical Introduction to Management Science, 6th Edition by Cliff T. Ragsdale, published by South-Western College Publishing, ISBN 0-538-74631-9. This book, a favorite in new MBA courses on management science and decision analysis, features an in-depth tutorial treatment of optimization, simulation, decision analysis, queuing models, and forecasting. This textbook also includes Risk Solver Platform for Education which is included as a free download with this book.

Academic References for Premium Solver Platform

The following academic journal articles, written by the developers of the Excel Solver, Premium Solver and Premium Solver Platform, describe many of the algorithms and technical methods used in these products. The first article describes the design of the original Excel Solver. You can download PDF versions of the first three articles at **http://www.solver.com/academic.htm**:

D. Fylstra, L. Lasdon, J. Watson and A. Waren. Design and Use of the Microsoft Excel Solver. *INFORMS Interfaces* 28:5 (Sept-Oct 1998), pp. 29-55.

I. Nenov and D. Fylstra. Interval Methods for Accelerated Global Search in the Microsoft Excel Solver. *Reliable Computing* 9 (2003): pp. 143–159.

D. Fylstra, "Introducing Convex and Conic Optimization for the Quantitative Finance Professional," *Wilmott Magazine* (March 2005), pp. 18-22.

For a technical description of the nonlinear GRG solver included with the standard Microsoft Excel Solver and Premium Solver, please consult the following:

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. *INFORMS Journal on Computing* 4:1 (1992), pp. 2-15.

Installation and Licensing

What You Need

In order to install Premium Solver Platform V10.5, you must have first installed Excel for Mac 2011.It is not necessary to have the standard Excel Solver installed.

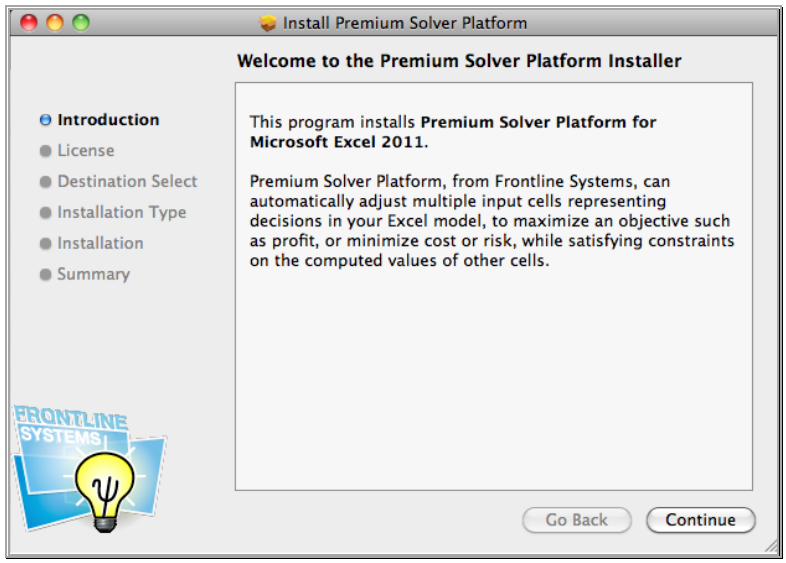
Premium Solver Platform will run on the same hardware and system software configuration that you've used to run Microsoft Excel. If you try to solve very large models, however, performance may depend on the amount of main memory (RAM) in your system. Large models with many integer constraints can take substantially more time to solve, and require more memory than models without such constraints. Use of the Polymorphic Spreadsheet Interpreter in Premium Solver Platform may also require considerable memory. Steps you can take to improve performance are outlined in the chapter "Building Large-Scale Models."

Installing the Software

Installing Premium Solver Platform or Premium Solver is a straightforward process. The Premium Solver Platform installer package, which contains all of the Solver files in compressed form, will guide you through the steps involved.

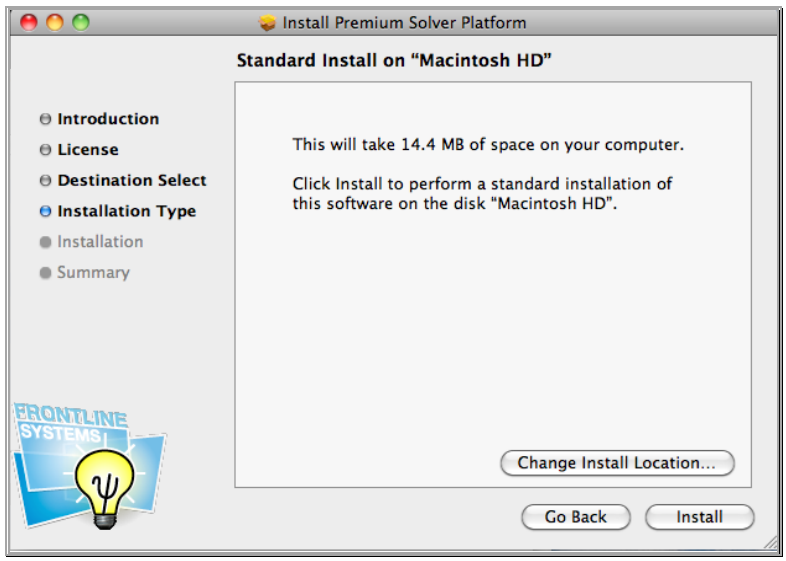
To begin the installation, insert the Frontline Systems CD or other media into your CD or disk drive. Double click on the Premium Solver Platform package on the CD to start the installation program.

A dialog box like the one shown on the next page should appear:

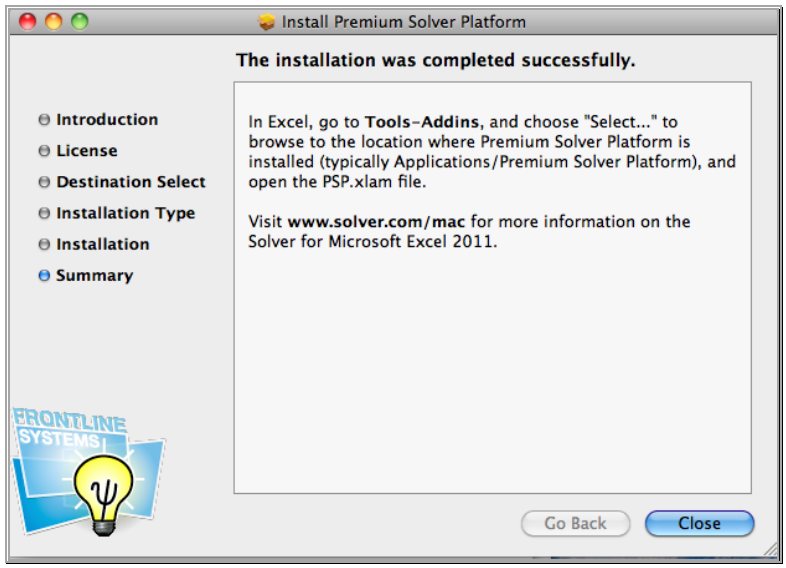


Press ENTER or click Continue to proceed.

After agreeing with the license agreement, the program will show a dialog like the one below, where you can select or confirm the folder to which files will be copied. We recommend that you simply click Install.



When the installation is complete, you'll see a dialog box like the one on the next page.



Be sure to read any special messages in the final dialog box(es) – they may give you important hints about new developments since this User Guide was updated. Then press ENTER or click Close. Premium Solver Platform V10.5 is now installed. Run Microsoft Excel, choose **Tools Addins, press Select and choose Psp.xlam from the folder where the Solver was installed (Typically Applications\Premium Solver Platform). Them, go to Tools-Premium Solver...** to display the new Solver Parameters dialog. Initially, the Solver engine dropdown list will contain the names of the four built-in Version 10.5 Solver engines. If you install additional Solver engines V10.5, they will also appear in this dropdown list.

Uninstalling the Software

To uninstall or remove any of Premium Solver Platform For Mac, go to Tools-Addins in Excel, and unselect Psp.xlam. Then simply drag the Applications\Premium Solver Platform folder into your trash can.

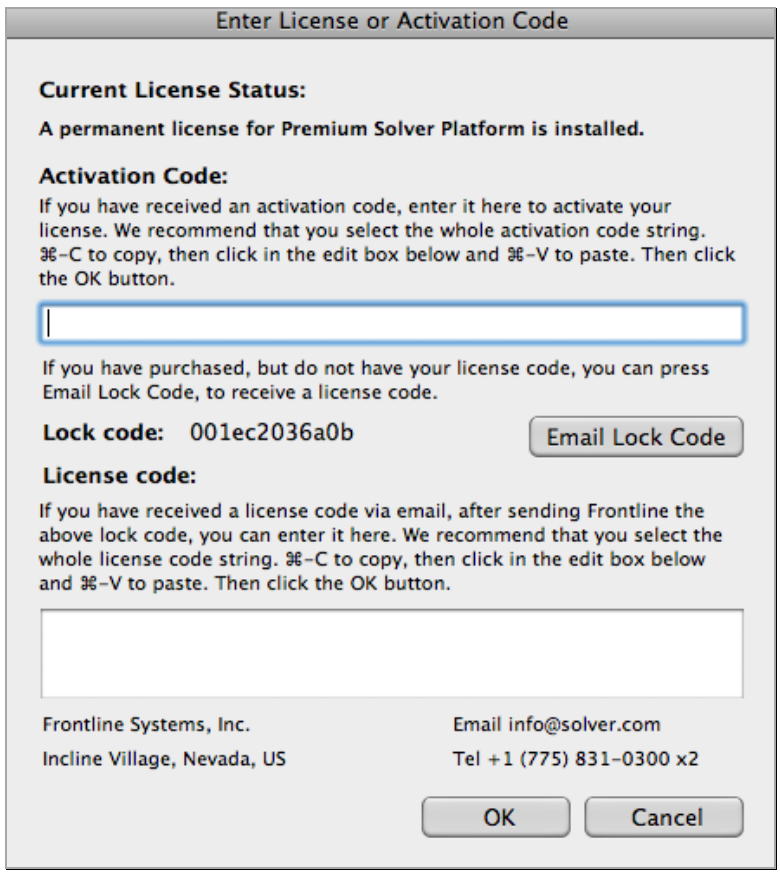
Licensing the Software

A **license** is a grant of rights, from Frontline Systems to you, to use our software in specified ways. Information about a license is encoded in a **license code**. Several types of licenses are available for both Premium Solver Platform for Mac and the field-installable Solver engines.

Frontline offers two basic types of licenses for **development** of your model and application: Standalone Licenses and Flexible Use (often called "Concurrent") Licenses. A Standalone License enables use on a single PC; a Flexible Use License enables shared use among several PCs on a network. The "Software License and Limited Warranty" section in the front of this User Guide describes the details of these licenses. Other types of licenses are available for **runtime** use: See the discussion on **www.solver.com**, then contact Frontline Systems for advice on license types best suited for your situation.

You can evaluate Premium Solver Platform for Mac free of charge for a limited period – typically 15 days – using a special trial license code that is copied to your Mac by the setup program. To obtain license codes that enable use of Premium Solver Platform for longer periods of time or on a permanent basis, please contact Frontline Systems at (775) 831-0300 or **info@solver.com**.

If your trial or time-limited license code is expired, or if some other licensing problem was detected, you'll see a dialog box like the one below when you first select Tools Premium Solver… You can display this dialog at any time by clicking the Help –License Code menu item when the Solver Parameters dialog is visible.



Click the **Cancel** button to return or continue to the Solver Parameters dialog; you will be able to create or edit a Solver model using the Add, Change and Delete buttons, but if your trial license is expired, you'll receive a licensing error message if you click the Solve button or the Check Model button in the Solver Model dialog.

If you have received an activation code or license code from Frontline Systems, you can enter it in the dialog, and press OK. The license file on your machine will be updated, and take effect next time you go to Tools-Premium Solver.

If you have received no code, you can email the displayed lock code to info@solver.com, to receive either an activation code or license code.

If you are unable to add the license code using the Enter License Code dialog box, you can use NotePad or WordPad to edit the file Solver.lic (it is a plain ASCII text

file), and append the license code to the end of the file. The Solver.lic file is located in the Package Contents, of the Premium Solver, under MacOS.

If you're just getting started with Premium Solver Platform, the initial Help menu is a great place to start. We highly recommend that you click the **Examples menu item** , which will open the OptimiztionExamples.xlsm workbook that is installed with the other Solver files. You can also click the **Support Online** menu or the **Tutorial Online** menu, which will open a Web browser to additional examples or our tutorial on **www.solver.com**.

Installing Solver Engines

To install additional Solver engines, you'll follow steps similar to those outlined above for the Premium Solver Platform for Mac, but you'll be running the Engines package install program.

You can evaluate any or all of the field-installable Solver engines free of charge for a limited period – typically 15 days – using a special trial license code that is copied to your PC by the EngineSetup program. (This 15-day period runs independently from any evaluation period for Premium Solver Platform for Mac.) To obtain license codes that enable use of a specific Solver engine for longer periods of time or on a permanent basis, please contact Frontline Systems at (775) 831-0300 or **info@solver.com**.

After you press ENTER or click Close in the final Engine install dialog box, your Solver engine is installed and ready to use. Simply run Microsoft Excel and choose Tools Premium Solver... to display the Solver Parameters dialog. Then open the Solver engine dropdown list – the name of your new Solver engine should appear in the list. Click the Options button to display a Solver Options dialog for your new Solver engine.

You can enter a new license code for a Solver engine just as described above for Premium Solver Platform for Mac: Click the **Help License Code** menu item.

Solver Models and Optimization

Introduction

This chapter explains the principles behind spreadsheet Solvers, including the types of problems you can solve, types of constraints (regular, integer, conic, alldifferent) you can specify, the nature of linear, quadratic and nonlinear functions, convex and non-convex functions, smooth and non-smooth functions, and the algorithms and methods used by Premium Solver Platform for Mac and field-installable Solver engines.

If you are just starting out with the Solver, you may find it helpful to read the first section below, "Elements of Solver Models," and then proceed to the next chapter, "Building Solver Models," for a hands-on example. If you have been using the Solver for a while, and you'd like a more in-depth review of the mathematical relationships found in Solver models, and the optimization methods and algorithms used by the Solver, read the later, more advanced sections of this chapter. We recommend that even experienced users read the new sections on **convex and conic optimization** in this chapter.

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 – that satisfies the *constraints* and that maximizes or minimizes the *objective* or Set Cell value (if there is one). Let's 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 that 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.

Decision Variables and Parameters

Some of the input values may be fixed numbers, which you cannot change in the course of finding a solution – for example, prevailing interest rates or supplier's prices. We'll call these values *parameters* of the model. Often you will have several

"cases," "scenarios," or variations of the same problem to solve, and the parameter values will change in each problem variation. Such parameter values may be conveniently captured using the Excel Scenario Manager. But the parameter values will be fixed numbers for any given run of the Solver.

Other input values may be quantities that are variable, or under your control in the course of finding a solution. We'll refer to these as the variables, *decision variables*, or Changing Cells. The Solver will find optimal values for these variables or cells. Often, some of 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 By Changing Variable 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. For example, this could be a calculated value for projected profits (to be maximized), or costs, risk, or error values (to be minimized).

You may have a Solver model that 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 that satisfies the constraints. Typically this will be only one of (infinitely) many such solutions, located close to the starting values of the decision variables.

The Excel Solver also permits you to enter a specific value that you want the objective function or Set Cell to achieve. This feature was included for compatibility with the Goal Seek... command in Excel, which allows you to seek a specific value for a cell by adjusting the value of one other cell 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 your problem requires only a single Set Cell value and a single variable or Changing Cell with no constraints, you can just use the Goal Seek... command. If you have nothing to maximize or minimize, we recommend that you leave the Set Cell blank and enter all of your constraints 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 TRUE or FALSE which you might enter in a cell. In this example, if A1 were -0.0000001, the logical formula would evaluate to FALSE, 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 Excel Solver, constraints are specified by giving a cell reference such as A1 or A1:A5 (the "left hand side"), a relation (<=, = or >=), and an expression for the "right hand side." Although Excel allows you to enter any numeric expression on the right hand side, for reasons that will be explained in the chapter "Building Large-Scale Models," we strongly encourage you to use only *constants*, or references to cells that contain *constant values* on the right hand side. (A constant value to the Solver is any value that does not depend on any of the decision variables.)

A constraint such as A1:A5 <= 10 is shorthand for A1 <= 10, A2 <= 10, A3 <= 10, A4 <= 10, A5 <=10. A constraint such as A1:A5 <= B1:B5 is shorthand for A1 <= B1, A2 <= B2, A3 <= B3, A4 <= B4, A5 <= B5.

Another type of constraint is of the form A1:A5 = integer, where A1:A5 are decision variables. This specifies that the solution values for A1 through A5 must be integers or whole numbers, such as -1, 0 or 2, *to within a small tolerance.* This form of constraint, and related forms such as A1:A5 = binary and A1:A5 = alldifferent, are explored in the next section.

A new type of constraint supported by Premium Solver Platform for Mac is of the form A1:A5 = conic, where A1:A5 are decision variables. This is called a *second order cone* constraint and is further described in the next section.

Solutions: Feasible, "Good" and Optimal

A solution (set of values for the decision variables) for which all of the constraints in the Solver model are satisfied is called a *feasible solution.* In some problems, a feasible solution is already known; in others, finding a feasible solution may be the hardest part of the problem.

An *optimal solution* is a feasible solution where the objective function reaches its maximum (or minimum) value – for example, the most profit or the least cost. A *globally optimal solution* is one where there are no other feasible solutions with better objective function values. A *locally optimal solution* is one where there are no other feasible solutions "in the vicinity" with better objective function values – you can picture this as a point at the top of a "peak" or at the bottom of a "valley" which may be formed by the objective function and/or the constraints.

The Solver is designed to find feasible and optimal solutions. In the best case, it will find the globally optimal solution – but this is not always possible. In other cases, it will find a locally optimal solution, and in still others, it will stop after a certain amount of time with the best solution it has found so far. But like many users, you may decide that it's most important to find a *good solution* – one that is better than the solution, or set of choices, you are using now.

The kind of solution the Solver can find depends on the nature of the mathematical relationships between the variables and the objective function and constraints (and the solution algorithm used). As explained below, if your model is ***smooth convex***, you can expect to find a globally optimal solution; if it is smooth but ***non-convex***, you will usually be able to find a locally optimal solution; if it is ***non-smooth***, you may have to settle for a "good" solution that may or may not be optimal.

Below, we summarize the capabilities of the five Solver engines bundled with Premium Solver Platform for Mac: the LP/Quadratic Solver, SOCP Barrier Solver, nonlinear GRG Solver, and Evolutionary Solver. Later sections of this chapter provide an overview of the optimization methods and algorithms employed by each of these Solver engines.

LP/Quadratic Solver

The LP/Quadratic Solver finds optimal solutions to problems where the objective and constraints are all linear functions of the variables. (The term *linear function* is explained below, but you can imagine its graph as a straight line.) Since all linear functions are ***convex***, the Solver normally can find the *globally optimal solution*, if one exists. Because a linear function (a straight line) can always be increased or decreased without limit, the optimal solution is always determined by the constraints; there is no natural "peak" or "valley" for the objective function itself.

This Solver handles problems where the constraints are all linear, and the objective may be linear or quadratic (explained further below). If the quadratic objective function is ***convex*** (if minimizing, or ***concave*** if maximizing) the Solver will normally find a globally optimal solution. If the objective is ***non-convex*** (further explained below), the Solver will find only a locally optimal solution.

SOCP Barrier Solver

The SOCP Barrier Solver in Premium Solver Platform for Mac finds optimal solutions to problems where the objective and constraints are all linear or convex quadratic functions of the variables. (This is in contrast to the LP/Quadratic Solver, which permits only the objective function to be quadratic.) It also finds optimal solutions to problems with a linear objective, linear constraints, and *second order cone* (SOC) constraints; this is called a second order cone programming (SOCP) problem, as explained further below. Since all linear functions and SOC constraints are ***convex***, the SOCP Barrier Solver normally finds a *globally optimal solution*, if one exists.

Nonlinear GRG Solver

The nonlinear GRG Solver finds optimal solutions to problems where the objective and constraints are all smooth (***convex*** or ***non-convex***) functions of the variables. (The term *smooth function* is explained below, but you can imagine a graph – whether straight or curved – that contains no "breaks.") For non-convex problems, the Solver normally can find a *locally optimal solution*, if one exists – but this may or may not be the globally optimal solution. A nonlinear objective function can have a natural "peak" or "valley," but in most problems the optimal solution is partly or wholly determined by the constraints. The nonlinear GRG Solver can be used on problems with all-linear functions, but it is much less effective and efficient than the LP/Quadratic Solver or the SOCP Barrier Solver on such problems.

If you use multistart methods for global optimization with the nonlinear GRG Solver, you will have a better chance (but not a guarantee) of finding the globally optimal solution. The idea behind multistart methods is to automatically start the Solver from a variety of starting points, to find the best of the locally optimal solutions – ideally the globally optimal solution. These methods are more fully described (and contrasted with other methods for global search) below under "Global Optimization" and in the chapter "Solver Options."

Evolutionary Solver

The Evolutionary Solver usually finds *good solutions* to problems where the objective and constraints include non-smooth functions of the variables – in other words, where there are no restrictions on the formulas that are used to compute the objective and constraints. For this class of problems, the Solver will return the best feasible solution (if any) that it can find in the time allowed.

The Evolutionary Solver can be used on problems with all-smooth functions that may have multiple locally optimal solutions, in order to seek a globally optimal solution, or simply a better solution than the one found by the nonlinear GRG Solver alone; however, the combination of multistart methods and the GRG Solver are likely to do as well or better than the Evolutionary Solver on such problems. It can be used on problems with smooth convex functions, but it is usually less effective and efficient than the nonlinear GRG Solver on such problems. Similarly, it can be used on problems with all-linear functions, but there is little point in doing so when the LP/Quadratic, or SOCP Barrier Solver is available.

More About Constraints

This section explains in greater depth the role of certain types of constraints, including bounds on the decision variables, equality and inequality constraints, second order cone constraints, and different forms of integer constraints.

Bounds on the Variables

Constraints of the form A1 >= -5 or A1 <= 10 (for example), where A1 is a decision variable, are called *bounds on the variables* and are treated specially by the Solver. These constraints affect only one variable, whereas general constraints have an indirect effect on several variables that have been used in a formula such as A1+A2. Each of the Solver engines takes advantage of this fact to handle bounds on the variables more efficiently than general constraints.

The most common type of bound on a variable is a lower bound of zero (A1 >= 0), which makes the variable non-negative. Many variables represent physical quantities of some sort, which cannot be negative. As a convenience, the Solver Parameters dialog offers a check box "Make Unconstrained Variables Non-Negative," which automatically places a lower bound of zero on every variable which has not been given an explicit lower bound via a constraint in the Constraints list box.

Regardless of the Solver engine chosen, bounds on the variables always help speed up the solution process, because they limit the range of values that the Solver must explore. In many problems, you will be aware of realistic lower and upper bounds on the variables, but they won't be of any help to the Solver unless you include them in the Constraints list box! Bounds on the variables are especially important to the performance of the Evolutionary Solver, and multistart methods for global optimization, as discussed below under "Global Optimization" and in the chapter "Solver Options." They are also very important if you want the Solver to automatically transform your model, replacing non-smooth functions (such as IF) with additional variables and linear constraints, as explained in the chapter "Analyzing and Solving Models."

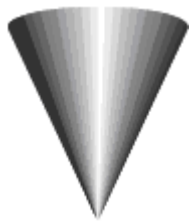
Equations and Inequalities

Constraints such as A1 = 0 are called *equality constraints* or *equations*; constraints such as A1 <= 0 are called *inequality constraints* or simply *inequalities*. An equality is much more restrictive than an inequality. For example, if A1 contains the formula =C1+C2, where C1 and C2 are decision variables, then A1 <= 0 restricts the possible solutions to a *half plane*, whereas A1 = 0 restricts the solutions to a *line* where all possible values of C1and C2 must sum to 0 (C1 = -C2 within a small tolerance, as explained above). Since there is only a tiny chance that two randomly chosen values for C1 and C2 will satisfy C1+C2 = 0, solution methods that rely on random choices, such as genetic algorithms, may have a hard time finding any feasible solutions to problems with equality constraints. To satisfy equality constraints, the Solver generally must exploit properties of the constraint formula – such as *linearity* or *smoothness*, discussed below – to solve for one variable in terms of another.

A linear equality constraint (like C1+C2 = 0 above) maintains the convexity of the overall problem, but a nonlinear equality constraint is ***non-convex***, and makes the overall problem non-convex. Interior point methods may have difficulty solving problems with nonlinear equality constraints, since they restrict the ability of the Solver to follow the "central path" inside the feasible region.

Second Order Cone Constraints

Premium Solver Platform for Mac supports constraints of the form A1:A5 = conic. This is called a *second order cone* (SOC) constraint; it specifies that the vector formed by the decision variables A1:A5 must lie within the second-order cone (also called the Lorentz cone, or "ice cream cone") of dimension 5 – a ***convex*** set that looks like the figure below in three dimensions.



Algebraically, a second-order cone constraint specifies that, given a value for one variable, the *L2*-norm of the vector formed by the remaining variables must not exceed this value: In linear algebra notation, $a_1 \geq \| a_2{:}a_5 \|_2$. In Excel, this could be written as A1 >= SQRT(SUMSQ(A2:A5)). You can also use a variant called a "rotated second order cone" constraint, as explained in the chapter "Building Solver Models." A problem with a linear objective and linear or SOC constraints is called a *second order cone programming* (SOCP) problem; it is always a ***convex*** optimization problem.

Decision variables that are constrained to be non-negative also belong to a cone, called the *non-negative orthant*. A problem with all linear functions – a linear programming problem – is a special case of an SOCP problem, where the only cone constraint is non-negativity.

A convex quadratic objective or constraint can be transformed into an equivalent second order cone constraint. Hence, a problem with a quadratic objective – a quadratic programming or QP problem – or a problem with quadratic constraints – called a QCP problem – is also a special case of an SOCP problem. The SOCP Barrier Solver and the MOSEK Solver will automatically transform quadratics into SOC form internally; you can simply define your quadratic objective and/or constraints using ordinary Excel formulas and <= or >= relations, and use these Solver engines to obtain fast, reliable, globally optimal solutions to your problem.

Integer, Binary and Alldifferent Constraints

As explained in the last section, integer constraints are of the form A1:A5 = integer, where A1:A5 are decision variables. This specifies that the solution values for A1 through A5 must be integers or whole numbers, such as -1, 0 or 2, to within a small tolerance. A common special case that can be entered directly in the Constraint List Box is A1 = binary, which is equivalent to specifying A1 = integer, A1 >= 0 and A1 <= 1. This implies that A1 must be *either 0 or 1* at the solution; hence A1 can be used to represent a "yes/no" decision. Integer constraints have many important applications, but the presence of even one such constraint in a Solver model makes the problem an integer programming problem (discussed below), which may be much more difficult to solve than a similar problem without the integer constraint.

Premium Solver Platform for Mac supports a new type of integer constraint, called the "alldifferent" constraint. Such a constraint is of the form (for example) A1:A5 = alldifferent, where A1:A5 is a group of two or more decision variables, and it specifies that these variables must be integers in the range 1 to N (N = 5 in this example), with each variable different from all the others at the solution. Hence, A1:A5 will contain a *permutation* of integers, such as 1,2,3,4,5 or 1,3,5,2,4. The

alldifferent constraint can be used to model problems involving ordering of choices, such as the Traveling Salesman Problem.

Functions of the Variables

Since there are large differences in the time it takes to find a solution and the *kinds* of solutions – globally optimal, locally optimal, or simply "good" – that you can expect for different types of problems, it pays to understand the differences between linear, quadratic, smooth nonlinear, and non-smooth functions, and especially **convex** and **non-convex** functions. To begin, let's clarify what it means to say that the spreadsheet cells you select for the objective and constraints are "functions of the decision variables."

The objective function in a Solver problem is a cell calculating a value that depends on the decision variable cells; the job of the Solver is to find some combination of values for the decision variables that maximizes or minimizes this cell's value. 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 that 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 have constant values on the right hand sides of constraints, then the same observation applies to the left hand sides of constraints: Parts of the constraint formulas (those which refer to non-decision variable cells) are unchanging in value, and only the parts that are dependent on the decision variables "count" during the optimization.

When you consider whether your objective and constraints are linear, quadratic, smooth nonlinear, or non-smooth, or **convex** or **non-convex** functions of the variables, always bear in mind that only the parts of formulas that are *dependent on the decision variables* "count." Below, we explain that linear functions are most desirable, and non-smooth and non-convex functions are least desirable in a Solver model (if you want the fastest and most reliable solutions). A formula such as =IF(C1>=10,D1,2*D1) is non-smooth if C1 depends on the decision variables; but if C1 *doesn't* depend on the variables, then only D1 or 2*D1 – not both – can be selected during the solution process. Hence if D1 is a linear function of the variables, then the IF expression is also a linear function of the variables.

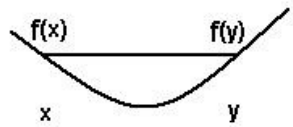
You may also find that a function that is "bad" (non-smooth or non-convex) over its full domain (any possible values for the decision variables) may be "good" (smooth and/or convex) over the domain of interest to you, determined by other constraints including bounds on the variables. For example, if C1 depends on the variables, then =IF(C1>=10,D1,2*D1) is non-smooth over its full domain, but smooth – in fact linear – if C1 is constrained to be 10 or more. =SIN(C1) is non-convex over its full domain, but is convex from –pi to 0, or from pi to 2*pi.

Convex Functions

The key property of functions of the variables that makes a problem "easy" or "hard" to solve is *convexity*. If *all* constraints in a problem are convex functions of the variables, and if the objective is convex if minimizing, or concave if maximizing, then you can be confident of finding a globally optimal solution (or determining that there is no feasible solution), even if the problem is very large – thousands to hundreds of thousands of variables and constraints.

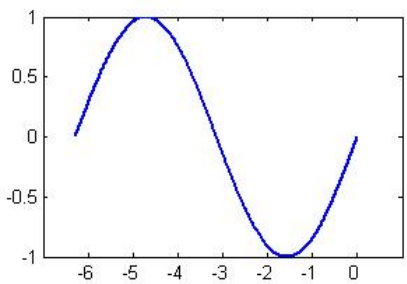
In contrast, if *any* of the constraints are non-convex, or if the objective is either non-convex, concave if minimizing, or convex if maximizing, then the problem is far more difficult: You cannot be certain of finding a feasible solution even if one exists; you must either "settle for" a locally optimal solution, or else be prepared for very long solution times and rather severe limits on the size of problems you can solve to global optimality (a few hundred to perhaps one thousand variables and constraints), even on the fastest computers. So it pays to understand convexity!

Geometrically, a function is *convex* if, at any two points x and y, the line drawn from x to y (called the *chord* from x to y) lies *on or above* the function – as shown in the diagram below, for a function of one variable. A function is *concave* if the chord from x to y lies *on or below* the function. This property extends to any number of 'dimensions' or variables, where $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$.



Algebraically, a function *f* is *convex* if, for any points x and y, and any t between 0 and 1, $f(tx + (1-t)y) <= tf(x) + (1-t)f(y)$. A function *f* is *concave* if *–f* is convex, i.e. if $f(tx + (1-t)y) >= tf(x) + (1-t)f(y)$. A *linear* function – described below – is both convex and concave: The chord from x to y lies on the line, and $f(tx + (1-t)y) = tf(x) + (1-t)f(y)$. As we'll see, a problem with all linear functions is the simplest example of a convex optimization problem that can be solved efficiently and reliably to very large size.

A non-convex function "curves up and down." A familiar example is the sine function (SIN(C1) in Excel), which is pictured on the next page.



The feasible region of an optimization problem is formed by the intersections of the constraints. The intersection of several convex constraints is always a convex region, but even one non-convex function can make the whole region non-convex – and hence make the optimization problem far more difficult to solve.

Linear Functions

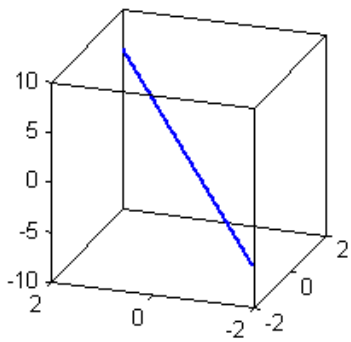
In many common cases, the objective and/or constraints are *linear* functions of the variables. This means that the function can be written as a sum of terms, where each term consists of one decision variable multiplied by a (positive or negative) constant. Algebraically, we can write:

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n$$

where the a_is, which are called the *coefficients*, stand for constant values and the x_is stand for the decision variables. A common example is =SUM(C1:C5), where C1:C5 are decision variables and the a_is are all 1. Note that a linear function does not have to be written in exactly the form shown above on the spreadsheet. For example, if

cells C1 and C2 are decision variables, B1 = C1+C2, and B2 = A1*B1 where A1 is constant in the problem, then B2 is a linear function (=A1*C1+ A1*C2).

Geometrically, a linear function is always a straight line, in *n*-dimensional space where *n* is the number of decision variables. Below is a perspective plot of $2x_1 + 1x_2$. As noted above, a linear function is always convex.



Remember that the a_is need only be *constant in the optimization problem*, i.e. not dependent on any of the decision variables. For example, suppose that the 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 the 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 function above as:

=SUMPRODUCT(A1:A2,C1:C2)

This is simple and clear, and is also useful for *fast problem setup* as described in the chapter "Building Large-Scale Models." As explained below in the section "Derivatives, Gradients, Jacobians and Hessians," each coefficient a_i in the linear expression $a_1x_1 + a_2x_2 + \ldots + a_nx_n$ is the first partial derivative of the expression with respect to variable x_i. These partial derivatives are always *constant* in a linear function – and all higher-order derivatives are zero.

A *nonlinear* function (explained further below), 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 – and its partial derivatives are *not* constant. 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.

Testing for a Linear Model

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? With Premium Solver Platform for Mac, you can easily find out by pressing the Check Model button in the Solver Model dialog, as explained in the chapter "Analyzing and Solving Models." Moreover, you can easily obtain a report showing exactly which cells contain formulas that are nonlinear.

Quadratic Functions

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

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

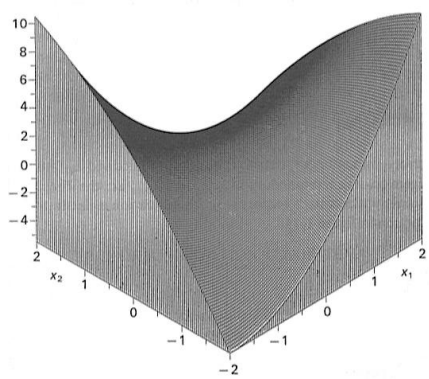
A quadratic function is a sum of terms, where each term is a (positive or negative) constant (again called a *coefficient*) multiplied by a single variable or the product of two variables. In linear algebra notation, we can write $x^TQx + cx$ where x is a vector of *n* decision variables, Q is an *n* x *n* matrix of coefficients, and c is an *n* vector of linear coefficients.

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.

As explained below in the section "Derivatives, Gradients, Jacobians and Hessians," the coefficients that multiply single variables in a quadratic function are the first partial derivatives of the function with respect to those variables; the coefficients that multiply the *products* of two variables are the *second* partial derivatives of the function, with respect to those two variables. In a quadratic function, these first and second order derivatives are always *constant*, and higher order derivatives are zero. The matrix Q of second partial derivatives is called the *Hessian* of the function.

Convex, Concave and Non-Convex Quadratics

A quadratic function of at least two variables may be convex, concave, or non-convex. The matrix Q in the general form x^TQx has a closely related algebraic property of *definiteness*. If the Q matrix is *positive definite*, the function is convex; if the Q matrix is *negative definite*, the function is concave. You can picture the graph of these functions as having a "round bowl" shape with a single bottom (or top). If the Q matrix is *semi-definite*, the function has a bowl shape with a "trough" where many points may have the same objective value, but it is still convex or concave. If the Q matrix is *indefinite*, the function is **non-convex**: It has a "saddle" shape, but its true minimum or maximum is not found in the "interior" of the function but on its boundaries with the constraints, where there may be many locally optimal points. Below is a plot of an example non-convex quadratic $x_1^2 + 2x_1x_2 - ½ (x_2^2 - 1)$:



A problem with **convex** quadratic functions is easily solved to global optimality up to very large size, but a problem with **non-convex** quadratic functions is a difficult global optimization problem that, in general, will require solution time that grows *exponentially* with the number of variables. The way that the Solver handles such functions is explained further below under "Quadratic Programming."

Nonlinear and Smooth Functions

A *nonlinear* function is any function of the variables that is not linear, i.e. which cannot be written in the algebraic form:

$a_1x_1 + a_2x_2 + \ldots + a_nx_n$

Examples, as before, are =1/C1, =LOG(C1), and =C1^2, where C1 is a decision variable. All of these are called *continuous* functions, because their graphs are curved but contain no "breaks." =IF(C1>10,D1,2*D1) is also a nonlinear function, but it is "worse" (from the Solver's viewpoint) because it is *discontinuous*: Its graph contains a "break" at C1=10 where the function value jumps from D1 to 2*D1. At this break, the rate of change (i.e. the derivative) of the function is undefined. As explained below in the section "Derivatives, Gradients, Jacobians and Hessians," most Solver algorithms rely on derivatives to seek improved solutions, so they may have trouble with a Solver model containing functions such as =IF(C1>10,D1,2*D1).

If the graph of the function's *derivative* also contains no breaks, then the original function is called a *smooth* function. If it does contain breaks, then the original function is *non-smooth*. Every discontinuous function is also non-smooth. An example of a continuous function that is non-smooth is =ABS(C1) – its graph is an unbroken "V" shape, but the graph of its derivative contains a break, jumping from –1 to +1 at C1=0. Many nonlinear Solver algorithms rely on second order derivatives of at least the objective function to make faster progress, and to test whether the optimal solution has been found; they may have trouble with functions such as =ABS(C1).

As explained below in the section "Derivatives, Gradients, Jacobians and Hessians," general nonlinear functions have first, second, and sometimes higher order derivatives that *change* depending on the point (i.e. values of the decision variables) at which the function is evaluated.

Convex, Concave and Non-Convex Smooth Functions

A general nonlinear function of even one variable may be convex, concave or non-convex. A function can be convex but non-smooth: =ABS(C1) with its V shape is an example. A function can also be smooth but non-convex: = SIN(C1) is an example. But the "best" nonlinear functions, from the Solver's point of view, are both *smooth* and *convex* (concave for the objective if you are maximizing).

If a smooth function's second derivative is always nonnegative, it is a *convex* function; if its second derivative is always nonpositive, it is a *concave* function. This property extends to any number of 'dimensions' or variables, where the second derivative becomes the Hessian and "nonnegative" becomes "positive semidefinite."

Discontinuous and Non-Smooth Functions

Microsoft Excel provides a very rich formula language, including many functions that are discontinuous or non-smooth. As noted above, discontinuous functions cause considerable difficulty, and non-smooth functions cause some difficulty for most nonlinear Solvers. Some models can only be expressed with the aid of these functions; in other cases, you have a degree of choice in how you model the real-world problem, and which functions you use. Even when you have a "full arsenal" of Solver engines available, as you do with Premium Solver Platform For Mac, you'll get better results if you try to use the most "Solver-friendly" functions in your model.

By far the most common discontinuous function in Excel is the IF function where the conditional test depends on the decision variables, as in the example =IF(C1>10,D1,2*D1). Here is a short list of common *discontinuous* Excel functions:

IF, CHOOSE
LOOKUP, HLOOKUP, VLOOKUP
COUNT
INT, ROUND
CEILING, FLOOR

Here is a short list of common *non-smooth* Excel functions:

ABS
MIN, MAX

Formulas involving relations such as <=, = and >= (on the worksheet, not in the Constraints list box) and logical functions such as AND, OR and NOT are discontinuous at their points of transition from FALSE to TRUE values. Functions such as SUMIF and the database functions are discontinuous if the criterion or conditional argument depends on the decision variables.

If you aren't sure about a particular function, try graphing it (by hand or in Microsoft Excel) over the expected range of the variables; this will usually reveal whether the function is discontinuous or non-smooth. If you have Premium Solver Platform, just create a model using the function, and use the Solver Model dialog to automatically diagnose the model type.

Premium Solver Platform for Mac Version 10.5 can **automatically transform** a model that uses IF, AND, OR, NOT, ABS, MIN and MAX, and relations <, <=, >= and > to an equivalent model where these functions and relations are replaced by additional binary integer and continuous variables and additional constraints, that have the same effect – for the purpose of optimization – as the replaced functions. This powerful facility may be able to transform your non-smooth model into a smooth or even linear model with integer variables. A real-life example is shown in the EXAMPLE5 worksheet of the OptimizationExamples.xlsm workbook, installed with the Solver files, which you can easily open from the Solver Parameters dialog by clicking Help, then clicking Examples. For more information, see the chapter "Analyzing and Solving Models."

Derivatives, Gradients, Jacobians, and Hessians

To find feasible and optimal solutions, most optimization algorithms rely heavily on derivatives of the problem functions (the objective and constraints) with respect to the decision variables. First derivatives indicate the direction in which the function is increasing or decreasing, while second derivatives provide curvature information.

The partial derivatives of a function $f(x_1,x_2,...,x_n)$ with respect to each variable are denoted $\partial f/\partial x_1$, $\partial f/\partial x_2$, …, $\partial f/\partial x_n$. They give the rate of change of the function in each dimension. For a linear function $a_1x_1 + a_2x_2 + ... + a_nx_n$, the partial derivatives are the coefficients: $\partial f/\partial x_1 = a_1$, $\partial f/\partial x_2 = a_2$, and so on.

To recap the comments about derivatives made in the sections above:

- Linear functions have *constant* first derivatives – the coefficients a_i – and all higher order derivatives (second, third, etc.) are zero.
- Quadratic functions have *constant* first and second derivatives, and all higher order (third, etc.) derivatives are zero.

- Smooth nonlinear functions have first and second derivatives that are *defined*, but not constant – they change with the point at which the function is evaluated.
- Non-smooth functions have second derivatives that are *undefined* at some points; discontinuous functions have first derivatives that are undefined at some points.

The *gradient* of a function $f(x_1,x_2,...,x_n)$ is the vector of its partial derivatives:

$$[\ \partial f/\partial x_1,\ \partial f/\partial x_2,\ \ldots,\ \partial f/\partial x_n\ ]$$

This vector points in the direction (in n-dimensional space) along which the function increases most rapidly. Since a Solver model consists of an objective and constraints, all of which are functions of the variables $x_1,x_2,...,x_n$, it is often useful to collect these gradients into a matrix, where each row is the gradient vector for one function:

$$\begin{vmatrix} \partial f_1/\partial x_1,\ \partial f_1/\partial x_2,\ \ldots,\ \partial f_1/\partial x_n \\ \partial f_2/\partial x_1,\ \partial f_2/\partial x_2,\ \ldots,\ \partial f_2/\partial x_n \\ \ldots \\ \partial f_m/\partial x_1,\ \partial f_m/\partial x_2,\ \ldots,\ \partial f_m/\partial x_n \end{vmatrix}$$

This matrix is called the *Jacobian* matrix. In a linear programming problem, this is the LP coefficient matrix, and all of its elements (the a_is) are constant.

The second partial derivatives of a function $f(x_1,x_2,...,x_n)$ with respect to each pair of variables x_i and x_j are denoted $\partial^2 f / \partial x_i \partial x_j$. There are n^2 second partial derivatives, and they can be collected into an n x n matrix:

$$\begin{vmatrix} \partial^2 f/\partial x_1\partial x_1,\ \partial^2 f/\partial x_1\partial x_2,\ \ldots,\ \partial^2 f/\partial x_1\partial x_n \\ \partial^2 f/\partial x_2\partial x_1,\ \partial^2 f/\partial x_2\partial x_2,\ \ldots,\ \partial^2 f/\partial x_2\partial x_n \\ \ldots \\ \partial^2 f/\partial x_n\partial x_1,\ \partial^2 f/\partial x_n\partial x_2,\ \ldots,\ \partial^2 f/\partial x_n\partial x_n \end{vmatrix}$$

This matrix is called the *Hessian* matrix. It provides second order (curvature) information for a single problem function, such as the objective. The Hessian of a linear function would have all zero elements; the Hessian of a quadratic function has all *constant* elements; and the Hessian of a general nonlinear function may change depending on the point (values of the decision variables) where it is evaluated.

When reading the next section, "Optimization Problems and Solution Methods," bear in mind that the different classes of Solver problems, and the computing time required to solve these problems, is directly related to the nature of the derivatives (constant, changing, or undefined) of their problem functions, as outlined above.

For example, because the first derivatives of linear functions are *constant*, they need be computed only once – and second derivatives (which are zero) need not be computed at all. For quadratic functions, the first and second derivatives can be computed only once, whereas for general nonlinear functions, these derivatives may have to be computed many times.

A major difference between Premium Solver Platform for Mac and Excel Solver is the method used to compute derivatives. As described in the chapter "Analyzing and Solving Models," the Polymorphic Spreadsheet Interpreter in Premium Solver Platform for Mac can supply fast, accurate derivatives to Solver engines via a process called *automatic differentiation*.

What if your optimization problem requires the use of non-smooth or discontinuous functions? With Premium Solver Platform for Mac, you have several choices. First, for common non-smooth functions such as ABS, MAX and MIN, and even for some IF functions, the nonlinear GRG, Large-Scale GRG and Large-Scale SQP Solvers often yield acceptable results, though you may need to use multistart methods to improve the chances of finding the optimal solution. Second, you can use the

Evolutionary Solver (which does not require any derivative values) to find a "good" solution, though you'll have to give up guarantees of finding an optimal solution, and it's likely to take considerably more computing time to find a solution. Third, you can use the automatic transformation feature to replace many of these functions with additional variables and linear constraints; if all discontinuous or non-smooth functions in the model are automatically replaced, the problem should be solvable with the nonlinear Solvers, or even with the linear Solvers in some cases. Fourth, you can manually reformulate your model with binary integer variables and associated constraints. You can then use the nonlinear GRG Solver, or even the LP/Quadratic Solver, in combination with the Branch & Bound method, to find the *true optimal solution* to your problem. These ideas are explored further in the chapter "Building Large-Scale Models."

Optimization Problems and Solution Methods

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. (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.) As noted earlier, a linear programming problem is always **convex**.

If the problem includes integer constraints, it is called an *integer linear* programming problem. A linear programming problem with some "regular" (continuous) decision variables, and some variables that are constrained to integer values, is called a *mixed-integer programming* (MIP) problem. Integer constraints are **non-convex**, and they make the problem far more difficult to solve; see below for details.

A *quadratic programming* (QP) problem is a generalization of a linear programming problem. Its objective is a **convex** *quadratic* function of the decision variables, and all of its constraints must be *linear* functions of the variables. A problem with linear and convex quadratic *constraints*, and a linear or convex quadratic objective, is called a *quadratically constrained* (QCP) problem.

A model in which the objective function and all of the constraints (other than integer constraints) are smooth nonlinear functions of the decision variables is called a *nonlinear programming* (NLP) or *nonlinear optimization* problem. If the problem includes integer constraints, it is called an *integer nonlinear* programming problem. A model in which the objective or any of the constraints are non-smooth functions of the variables is called a *non-smooth optimization* (NSP) problem.

Linear Programming

Linear programming (LP) problems are intrinsically easier to solve than nonlinear (NLP) problems. First, they are convex, where a general nonlinear problem is often non-convex. Second, since all constraints are linear, the globally optimal solution always lies at an "extreme point" or "corner point" where two or more constraints intersect. (In some problems there may be multiple solutions with the same objective value, all lying on a line between two corner points.) 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 a globally optimal solution.

Problem Size and Numerical Stability

Because of their structural simplicity, the main limitations on the size of LP problems that 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 means that the Jacobian matrix of partial derivatives of the problem functions, described earlier, will have many elements that are *zero*. Such sparsity can be exploited to save memory and gain speed in solving the problem.

The Simplex Method

LP problems are most often solved via the Simplex method. The standard Microsoft Excel Solver uses a straightforward implementation of the Simplex method to solve LP problems. Premium Solver Platform for Mac uses a far more sophisticated implementation of the Simplex method which exploits sparsity in the LP model and uses techniques such as presolving, matrix factorization using the LU decomposition , a fast and stable LU update, and dynamic Markowitz refactorization.

The Large-Scale LP/QP Solver engine for Premium Solver Platform for Mac uses an even more powerful implementation of the Simplex method, with performance rivaling the best LP solvers available. It has been tested on LP problems with over two million variables and constraints.

The Gurobi Solver engine is Frontline's fastest and most powerful Solver for linear programming and especially *mixed-integer* linear programming problems. Its ultra-sophisticated primal and dual Simplex and Barrier methods, combined with state-of-the-art Branch and Cut methods for integer problems, yield solutions in record time.

Quadratic Programming

Quadratic programming problems are more complex than LP problems, but simpler than general NLP problems. They have only one feasible region with "flat faces" on its surface (due to their linear constraints), but the optimal solution may be found anywhere within the region or on its surface. Since a QP problem is a special case of an NLP problem, it *can* be solved with the standard nonlinear GRG Solver, but this may take considerably more time than solving an LP of the same size. Premium Solver Platform for Mac's LP/Quadratic Solver solves QP problems very efficiently, using a variant of the Simplex method to determine the feasible region, and special methods based on the properties of quadratic functions to find the optimal solution.

Most quadratic programming algorithms are specialized to handle only positive definite (or negative definite) quadratics. The LP/Quadratic Solver, however, can also handle semi-definite quadratics; it will find one of the equivalent (globally) optimal solutions – which one depends on the starting values of the decision variables. When applied to an indefinite quadratic objective function, the LP/Quadratic Solver provides only the guarantees of a general nonlinear Solver: It will converge to a locally optimal solution (either a saddle point in the interior, or a locally optimal solution on the constraint surface).

The Large-Scale LP/QP Solver, Large-Scale GRG Solver, MOSEK Solver, KNITRO Solver, and Gurobi Solver engines can all be used to efficiently solve large QP problems.

Quadratically Constrained Programming

A problem with linear and convex quadratic *constraints*, and a linear or convex quadratic objective, is called a *quadratically constrained* (QCP) problem. Such a problem is more general than a QP or LP problem, but less general than a convex nonlinear problem. The Simplex-based methods used in Premium Solver Platform's LP/Quadratic Solver, the Large-Scale LP/QP Solver, and the Gurobi Solver Engine handle only quadratic objectives, not quadratic constraints. But QCP problems – since they are **convex** – can be solved efficiently to global optimality with Barrier methods, also called Interior Point methods.

Premium Solver Platform for Mac's new SOCP Barrier Solver uses a Barrier method to solve LP, QP, and QCP problems. The MOSEK Solver Engine uses an even more powerful Barrier method to solve very large scale LP, QP, and QCP problems, as well as smooth convex nonlinear problems. Both of these Solvers form a logarithmic "barrier function" of the constraints, combine this with the objective, and take a step towards a better point on each major iteration. Unlike the Simplex method, which moves from one corner point to another on the *boundary* of the feasible region, a Barrier method follows a path – called the *central path* – that lies strictly *within* the feasible region.

A Barrier method relies heavily on second derivative information, specifically the Hessian of the Lagrangian (combination of the constraints and objective) to determine its search direction on each major iteration. The ability of the Polymorphic Spreadsheet Interpreter in Premium Solver Platform for Mac to efficiently compute this second derivative information is key to the performance of this method.

Second Order Cone Programming

Second order cone programming (SOCP) problems are a further generalization of LP, QP, and QCP problems. An SOCP has a linear objective and one or more linear or *second order cone* (SOC) constraints. As explained earlier, a second order cone constraint such as "A1:A5 = conic" specifies that the vector formed by the decision variables A1:A5 must lie within the second-order cone (also called the Lorentz cone) of dimension 5. Algebraically, the constraint specifies that $a_1 \geq \| a_2{:}a_5 \|_2$. SOCPs are always **convex**; Premium Solver Platform's new SOCP Barrier Solver and the MOSEK Solver Engine are both designed to solve SOCP problems, efficiently to global optimality.

Any convex quadratic constraint can be converted into an SOC constraint, with several steps of linear algebra. A convex quadratic objective $x^TQx + cx$ can be handled by introducing a new variable *t*, making the objective minimize *t*, adding a constraint $x^TQx + cx <= t$, and converting this constraint to SOC form. The SOCP Barrier Solver and the MOSEK Solver Engine both make these transformations automatically; in effect they solve all LP, QP, QCP and SOCP problems in the same way. Second order cone programming can be viewed as the *natural generalization of linear programming*, and is bound to become more popular in the future.

You can also solve an SOCP with the GRG Nonlinear Solver or the Large-Scale GRG, or KNITRO Solver engines. Although these Solvers do not recognize SOC constraints directly, Premium Solver Platform for Mac will compute values and derivatives for SOC constraints, based on their algebraic form shown above. Hence, these general nonlinear Solvers handle SOC constraints like other general nonlinear constraints. Using these Solvers, you can find optimal solutions for problems containing a mix of linear, general nonlinear, and SOC constraints – bearing in mind that such problems may be non-convex.

Nonlinear Optimization

As outlined above, nonlinear programming (NLP) problems are intrinsically more difficult to solve than LP, QP, QCP or SOCP problems. They may be **convex** or **non-convex**, and since their second derivatives are not constant, an NLP Solver must compute or approximate the Hessians of the problem functions many times during the course of the optimization. Since a non-convex NLP may have multiple feasible regions and multiple locally optimal points within such regions, there is no simple or fast way to determine with certainty that the problem is infeasible, that the objective function is unbounded, or that an optimal solution is the "global optimum" across all feasible regions. But some NLP problems *are* convex, and many problems include linear or convex quadratic constraints in addition to general nonlinear constraints. Frontline's field-installable nonlinear Solver engines are each designed to take advantage of NLP problem structure in different ways, to improve performance.

If you use the GRG Nonlinear Solver – the only choice for NLPs in the standard Excel Solver– bear in mind that it applies the *same* method to *all* problems, even those that are really LPs or QPs. If you don't select another Solver engine from the dropdown list box in the Solver Parameters dialog (or, in the standard Microsoft Excel Solver, if you don't check the Assume Linear Model box in the Solver Options dialog), this Solver will be used – and it may have difficulty with LP or QP problems that could have been solved easily with one of the other Solvers. Premium Solver Platform for Mac can automatically determine the type of problem, and select only the "good" or "best" Solver engine(s) for that problem.

The GRG Method

The standard Excel Solver, Premium Solver and Premium Solver Platform include a standard nonlinear GRG Solver, which uses the Generalized Reduced Gradient method as implemented in Lasdon and Waren's GRG2 code. The GRG method can be viewed as a nonlinear extension of the Simplex method, which selects a basis, determines a search direction, and performs a line search on each major iteration – solving systems of nonlinear equations at each step to maintain feasibility. 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.

As with the Simplex method, the GRG method in the standard Excel 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. The Large-Scale GRG Solver engine for Premium Solver Platform for Mac uses sparse storage methods and better numerical methods for nonlinear models, such as matrix condition testing and degeneracy handling, to solve much larger NLP problems.

Interior Point and SLQP Methods

The KNITRO Solver engine uses a Barrier or Interior Point method, specialized for **non-convex** problems, to solve general nonlinear optimization problems. As with the SOCP Barrier and MOSEK Solvers, this method forms a logarithmic "barrier function" of the constraints, combines this with the objective, and takes a step towards a better point on each major iteration. (The actual process of taking a step and the path followed are more complex, because KNITRO assumes that the problem may be non-convex.) The KNITRO Solver uses the Polymorphic Spreadsheet Interpreter in Premium Solver Platform for Mac to efficiently compute second

derivative information, but it also has options to work with only first derivative information.

The KNITRO Solver engine also includes a new, high performance Sequential Linear-Quadratic (SLQP) method, which is an "active set" method similar to the SQP method. On highly constrained problems, notably those with equality constraints, this method typically outperforms the Interior Point method. On loosely constrained or unconstrained problems, the Interior Point method can greatly outperform SQP and GRG methods, solving problems much larger than either of these methods. Benchmark studies in the academic literature have demonstrated exceptionally good performance for the KNITRO Solver, on a wide range of test problems.

The GRG, SQP and Interior Point methods are all subject to the intrinsic limitations cited above for nonlinear optimization problems: For smooth **convex** nonlinear problems, they will (subject to the limitations of finite precision computer arithmetic) find the globally optimal solution; but for **non-convex** problems, they can only guarantee a locally optimal solution. To have a reasonable chance – let alone a guarantee – that you'll find the globally optimal solution to a non-convex problem, you must use special methods for global optimization.

Global Optimization

Premium Solver Platform for Mac includes powerful tools to help you find the globally optimal solution for a smooth nonlinear **non-convex** problem. These tools include *multistart methods*, which can be used with the nonlinear GRG Solver, the Large-Scale GRG Solver , and the KNITRO Solver; and the Evolutionary Solver, for global solutions of smooth and non-smooth problems.

The Multistart Method

The basic idea of the multistart method is to automatically run a nonlinear Solver from different starting points, reaching different locally optimal solutions, then select the best of these as the proposed globally optimal solution. Both *clustering* and *topographic search* multistart methods are included in Premium Solver Platform for Mac.

The multistart method operates by generating candidate starting points for the nonlinear Solver (with randomly selected values between the bounds you specify for the variables). These points are then grouped into "clusters" – through a method called *multi-level single linkage* – that are likely to lead to the same locally optimal solution, if used as starting points for the Solver. The nonlinear Solver is then run repeatedly, once from (a representative starting point in) each cluster. The process continues with successively smaller clusters that are increasingly likely to capture each possible locally optimal solution. A Bayesian test is used to determine whether the process should continue or stop.

For many smooth nonlinear problems, the multistart method has a limited guarantee that it will "converge in probability" to a globally optimal solution. This means that as the number of runs of the nonlinear Solver increases, the probability that the globally optimal solution has been found also increases towards 100%. (To attain convergence for constrained problems, an exact penalty function is used in the process of "clustering" the starting points.) For most nonlinear problems, this method will at least yield very good solutions. As discussed below, the multistart method, like the Evolutionary Solver, is a *nondeterministic* method, which by default may yield different solutions on different runs. (To obtain the *same* solution on each run, you can set a Random Seed option for either of these solution algorithms, as discussed in the chapter "Solver Options.")

As discussed below, in recent versions of Premium Solver Platform for Mac, the Evolutionary Solver has been enhanced with "filtered local search" methods that offer many of the benefits of multistart methods – making the Evolutionary Solver even more effective for global optimization problems.

The multistart method can be used on smooth nonlinear problems that also contain integer variables and/or "alldifferent" constraints. But this can take a great deal of solution time, since the multistart method is used for each subproblem generated by the Branch & Bound method for integer problems, and it can also impact the Solver's ability to find feasible integer solutions, as described in the chapter "Diagnosing Solver Results." If you have many integer variables, or alldifferent constraints, try the Evolutionary Solver as an alternative to the multistart method.

Non-Smooth Optimization

The most difficult type of optimization problem to solve is a non-smooth problem (NSP). Such a problem may not only have multiple feasible regions and multiple locally optimal points within each region – because some of the functions are non-smooth or even discontinuous, derivative or gradient information generally cannot be used to determine the direction in which the function is increasing (or decreasing). In other words, the situation at one possible solution gives very little information about where to look for a better solution.

In all but the simplest problems, it is impractical to exhaustively enumerate all of the possible solutions and pick the best one, even on a fast computer. Hence, most methods rely on some sort of controlled random search, or sampling of possible solutions – combined with deterministic (non-random) methods for exploring the search space. .

A drawback of these methods is that a solution is "better" only in comparison to other, presently known solutions; the Evolutionary Solver normally *has no way to test whether a solution is optimal*. This also means that these methods must use heuristic rules to decide *when to stop*, or else stop after a length of time, or number of iterations or candidate solutions, that you specify.

Genetic and Evolutionary Algorithms

A non-smooth optimization problem can be attacked – though not often solved to optimality – using a genetic or evolutionary algorithm. (In a genetic algorithm the problem is encoded in a series of bit strings that are manipulated by the algorithm; in an "evolutionary algorithm," the decision variables and problem functions are used directly. Most commercial Solver products are based on evolutionary algorithms.)

An evolutionary algorithm for optimization is different from "classical" optimization methods in several ways. First, it relies in part on random sampling. This makes it a *nondeterministic* method, which may yield different solutions on different runs. (To obtain the *same* solution on each run, you can set a Random Seed option for the Evolutionary Solver, as discussed in the chapter "Solver Options.")

Second, where most classical optimization methods maintain a single best solution found so far, an evolutionary algorithm maintains a *population* of candidate solutions. Only one (or a few, with equivalent objectives) of these is "best," but the other members of the population are "sample points" in other regions of the search space, where a better solution may later be found. The use of a population of solutions helps the evolutionary algorithm avoid becoming "trapped" at a local optimum, when an even better optimum may be found outside the vicinity of the current solution.

Third – inspired by the role of mutation of an organism's DNA in natural evolution – an evolutionary algorithm periodically makes random changes or *mutations* in one or more members of the current population, yielding a new candidate solution (which may be better or worse than existing population members). There are many possible ways to perform a "mutation," and the Evolutionary Solver actually employs five different mutation strategies. The result of a mutation may be an infeasible solution, and the Evolutionary Solver attempts to "repair" such a solution to make it feasible; this is sometimes, but not always, successful.

Fourth – inspired by the role of sexual reproduction in the evolution of living things – an evolutionary algorithm attempts to combine elements of existing solutions in order to create a new solution, with some of the features of each "parent." The elements (e.g. decision variable values) of existing solutions are combined in a *crossover* operation, inspired by the crossover of DNA strands that occurs in reproduction of biological organisms. As with mutation, there are many possible ways to perform a "crossover" operation – some much better than others – and the Evolutionary Solver actually employs multiple variations of four different crossover strategies.

Fifth – inspired by the role of natural selection in evolution – an evolutionary algorithm performs a *selection* process in which the "most fit" members of the population survive, and the "least fit" members are eliminated. In a constrained optimization problem, the notion of "fitness" depends partly on whether a solution is feasible (i.e. whether it satisfies all of the constraints), and partly on its objective function value. The selection process is the step that guides the evolutionary algorithm towards ever-better solutions.

Hybrid Evolutionary and Other Algorithms

You might imagine that better results could be obtained by combining the strategies used by an evolutionary algorithm with the "classical" optimization methods used by the nonlinear GRG and linear Simplex Solvers. In Premium Solver Platform for Mac, Frontline Systems has done just that.

The Evolutionary Solver operates as described above, but it also employs classical methods in two situations: First, when the evolutionary algorithm generates a new best point, a local search is conducted to try to improve that point. This step can use a "random local search" method, a gradient-free, deterministic direct search method, a gradient-based quasi-Newton method, or a "linearized local gradient" method. Second, when the evolutionary algorithm generates an infeasible point, the Solver can use "repair methods", a quasi-Newton method, or even a specialized Simplex method (for subsets of the constraints that are linear) to transform the infeasible point into a feasible one.

In Premium Solver Platform for Mac, the Evolutionary Solver takes maximum advantage of the diagnostic information available from the Polymorphic Spreadsheet Interpreter: It automatically applies genetic algorithm methods to non-smooth variable occurrences (where classical methods cannot be used) and classical methods to smooth and linear variable occurrences. In the local search phase, it can either fix non-smooth variables, or allow them to vary. And it can automatically select the most appropriate local search method, based on linearity and smoothness of the problem functions.

The Evolutionary Solver uses a "distance filter" and a "merit filter" to determine whether to carry out a local search when the genetic algorithm methods find an improved starting point. The "distance filter" plays a role similar to "clustering" in the multistart methods described earlier; both filters contribute to the excellent performance of the Evolutionary Solver on global optimization problems.

The "Achilles' heel" of most evolutionary algorithms is their handling of constraints – they are typically unable to handle more than a few inequalities, or any equality constraints at all. In contrast, the hybrid Evolutionary Solver in Premium Solver Platform for Mac has been able to find good solutions to non-smooth problems with many – even hundreds – of constraints.

Integer Programming

When a Solver model includes integer constraints (for example A1:A10 = integer, A1:A10 = binary, or A1:A10 = alldifferent), it is called an *integer programming* problem. Integer constraints effectively make a model **non-convex**, and finding the optimal solution to an integer programming problem is equivalent to solving a global optimization problem. Such problems may require *far* more computing time than the same problem without the integer constraints.

The standard Microsoft Excel Solver uses a basic Branch & Bound method, in conjunction with the linear Simplex or nonlinear GRG Solver, to find optimal solutions to problems involving general integer or binary integer variables. Premium Solver Platform for Mac uses a much more sophisticated Branch & Bound method that is extended to handle alldifferent constraints, and that often greatly speeds up the solution process for problems with integer variables. Premium Solver Platform for Mac's LP/Quadratic Solver uses improved pseudocost-based branch and variable selection, reduced cost fixing, primal heuristics, cut generation, Dual Simplex and preprocessing and probing methods to greatly speed up the solution of *integer linear* programming problems.

The Evolutionary Solver handles integer constraints, in the same form as the other Solver engines (including alldifferent constraints), but it does not make use of the Branch & Bound method; instead, it generates many trial points and uses "constraint repair" methods to satisfy the integer constraints. (The constraint repair methods include classical methods, genetic algorithm methods, and integer heuristics from the local search literature.) The Evolutionary Solver can often find good solutions to problems with integer constraints, but where the Branch & Bound algorithm can *guarantee* that a solution is optimal or is within a given percentage of the optimal solution, the Evolutionary Solver cannot offer such guarantees.

The Branch & Bound Method

The Branch & Bound method begins by finding the optimal solution to the "relaxation" of the integer problem, ignoring the integer constraints. If it happens that in this solution, the decision variables with integer constraints 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.

Cut Generation

Premium Solver Platform for Mac's LP/Quadratic Solver, the Large-Scale LP/QP Solver, MOSEK Solver, and the Gurobi Solver Engine all make use of "cut generation" methods to improve performance on integer linear programming problems. Cut generation derives from so-called "cutting plane" methods that were among the earliest methods applied to integer programming problems, but they combine the advantages of these methods with the Branch & Bound method to yield a highly effective approach, often referred to as a "Branch & Cut" algorithm.

A *cut* is an automatically generated linear constraint for the problem, in addition to the constraints that you specify. This constraint is constructed so that it "cuts off" some portion of the feasible region of an LP subproblem, without eliminating any possible integer solutions. Many cuts may be added to a given LP subproblem, and there are many different methods for generating cuts. For example, *Gomory cuts* are generated by examining the reduced costs at an LP solution, while *knapsack cuts*, also known as *lifted cover inequalities*, are generated from constraints involving subsets of the 0-1 integer variables. Cuts add to the work that the LP solver must perform on each subproblem (and hence they do not always improve solution time), but on many problems, cut generation enables the overall Branch & Cut algorithm to more quickly discover integer solutions, and eliminate branches that cannot lead to better solutions than the best one already known.

The Alldifferent Constraint

In Premium Solver Platform for Mac, a constraint such as A1:A5 = alldifferent specifies that the variables A1:A5 must be integers in the range 1 to 5, with each variable different from all the others at the solution. Hence, A1:A5 will contain a *permutation* of the integers from 1 to 5, such as 1,2,3,4,5 or 1,3,5,2,4.

To solve problems involving alldifferent constraints, Premium Solver Platform For Mac employ an extended Branch & Bound method that handles these constraints as a native type. Whenever variables in an "alldifferent group" have non-integral solution values, or integral values that are not all different, the Branch & Bound method chooses one such variable and "branches," creating two new subproblems where the value of that variable is more tightly constrained.

The nonlinear GRG Solver bundled with Premium Solver Platform for Mac, and the Large-Scale GRG Solver, KNITRO Solver, and MOSEK Solver engines use this extended Branch & Bound method to solve problems with integer and alldifferent constraints.

The Large-Scale LP/QP Solver, and Gurobi Solver use their own Branch & Cut methods. They transform alldifferent constraints into equivalent sets of binary integer variables and additional linear constraints, then apply their preprocessing, probing and cut generation methods to these variables and constraints.

The Evolutionary Solver uses methods from the genetic algorithm literature to handle alldifferent constraints as permutations, including several mutation operators that preserve the "alldifferent property," and several crossover operators that generate a "child" permutation from "parents" that are also permutations.

Since Solver engines use quite different methods to handle the alldifferent constraint, you'll want to try a variety of Solver engines to see which one performs best on your

model. This is especially true if your model uses smooth nonlinear or – even better – linear functions aside from the alldifferent constraint.

Building Solver Models

Introduction

This chapter takes you step by step through the process of setting up and solving a simple linear programming model in Excel, using Premium Solver Platform for Mac. It then describes in depth the features of the Solver Parameters dialog that you can use to define the essential elements of your Solver model: decision variables, constraints, and the objective function.

Our step-by-step example is a "quick and dirty" setup that can be used to solve the example problem, but is not well documented or easy to maintain. Microsoft Excel has many features that can help you organize and display the structure of your model, 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. Hence, the last section in this chapter provides hints for designing a model that is understandable, maintainable and scalable.

From Algebra to Spreadsheets

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

Setting Up a Model

To set up an optimization model as a Microsoft Excel 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 that 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. If the model is easily expressed in vector-matrix algebraic notation, you may want to use defined names for the vectors and built-in functions such as MMULT to compute the constraint left hand sides.

A Sample Linear Programming Model

Consider the following LP problem, a variation on the "Product Mix" worksheet in the OptimizationExamples.xlsm workbook included with Microsoft Excel. Our factory is building three products: TV sets, stereos and speakers. Each product is assembled from parts in inventory, and there are five types of parts: chassis, picture tubes, speaker cones, power supplies and electronics units. Our goal is to produce the mix of products that will maximize profits, given the inventory of parts on hand.

The Algebraic Form

This problem can be described in algebraic form as follows. The decision variables are the number of products of each type to build: x_1 for TV sets, x_2 for stereos and x_3 for speakers. There is a fixed profit per unit for each product, so the objective function (the quantity we want to maximize) is:

Maximize $75\ x_1 + 50\ x_2 + 35\ x_3$ (Profit)

Building each product requires a certain number of parts of each type. For example, TV sets and stereos each require one chassis, but speakers don't use one. The number of parts used depends on the mix of products built (the left hand side of each constraint), and we have a limited number of parts of each type on hand (the corresponding constraint right hand side):

Subject to
$1\ x_1 + 1\ x_2 + 0\ x_3 <= 400$ (Chassis)
$1\ x_1 + 0\ x_2 + 0\ x_3 <= 200$ (Picture tubes)
$2\ x_1 + 2\ x_2 + 1\ x_3 <= 800$ (Speaker cones)
$1\ x_1 + 1\ x_2 + 0\ x_3 <= 400$ (Power supplies)
$2\ x_1 + 1\ x_2 + 1\ x_3 <= 600$ (Electronics)

Since the number of products built must be nonnegative, we also have the constraints $x_1, x_2, x_3 >= 0$.

The Spreadsheet Formulas

The fastest (though not necessarily the best) way to lay out this problem on the spreadsheet is to pick (for example) cell A1 for x_1, cell A2 for x_2 and cell A3 for x_3. Then the objective might be entered in cell A4 as the formula:

=75*A1+50*A2+35*A3

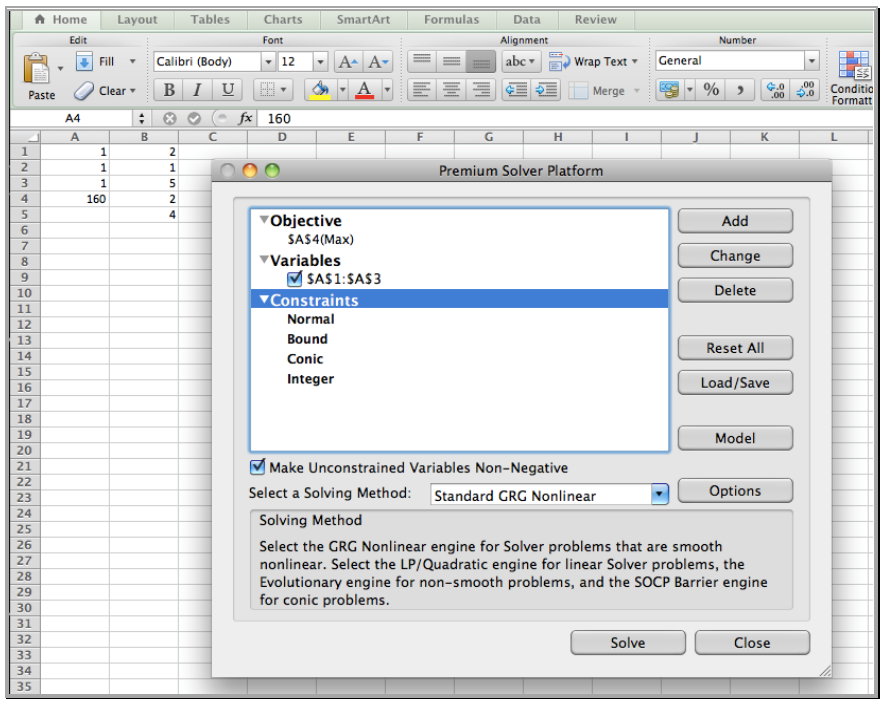
We'd go on to enter a formula in (say) cell B1 for the first constraint left hand side (Chassis), such as =1*A1+1*A2+0*A3, or perhaps the equivalent =A1+A2. Similarly, we'd use cell B2 for the formula =A1 (Picture tubes), B3 for the formula =2*A1+2*A2+A3 (Speaker cones), B4 for the formula =A1+A2 (Power supplies), and B5 for the formula =2*A1+A2+A3 (Electronics).

We now have a simple spreadsheet model, with which we can practice "what if." For any values we enter for the decision variables in cells A1, A2 and A3, the objective (Total Profit) and the corresponding values of the constraint left hand sides (the numbers of parts used) will be calculated.

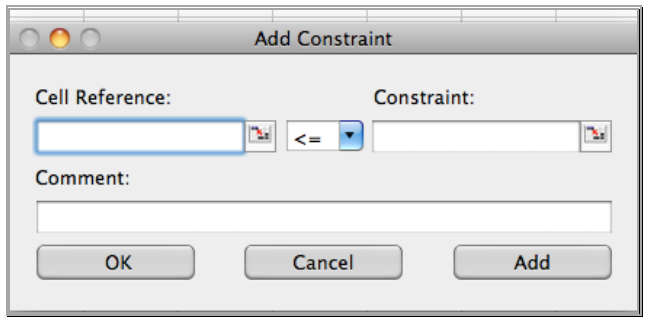
The Solver Dialogs

To prepare the model for optimization, we will use the Solver Parameters dialog to point out to the Solver (i) the cells that we've reserved for the decision variables, (ii) the cell that calculates the value of the objective function, and (iii) the cells that calculate the constraint left hand sides. We'll also enter values for the constraint right hand sides, and non-negativity constraints on the variables.

The simplest way to proceed is to choose Premium Solver... command from the Tools menu. The Solver Parameters dialog will appear. Select Objective, and choose Add, and type in A4. The default choice of Max is correct for this problem. Hit OK. To select the decision variables, Select Variables, and choose Add A1:A3 and click OK. At this point your spreadsheet and Solver Parameters dialog should look like this:

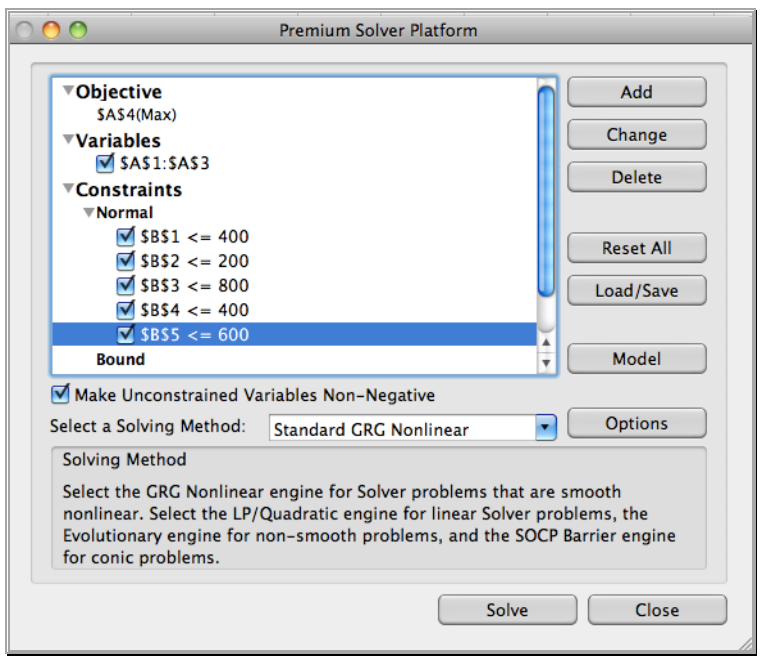


The next step is to enter the constraints, including the non-negativity constraint on the decision variables. To enter the non-negativity constraint, select Constraints and click on the Add button to bring up the Add Constraints dialog. The input focus is on the Cell Reference edit box, so you can either type A1:A3 or use the mouse to select cells A1 to A3 on the spreadsheet. Next, click on the Constraint (right hand side) edit box and enter 0 there. Finally, click on the down arrow next to Relation to display a list of relation symbols, and select >= from the list. Your Add Constraint dialog box should look like the one on the next page.



To accept the non-negativity constraint and continue with entry of another constraint, click the Add button. The Add Constraint dialog box will reappear with the edit fields blank. With the input focus on Cell Reference, type B1, the first constraint left hand side. Then click on the Constraint edit box, and enter 400 there. The default relation <= is correct for this constraint, so you are now ready to click the Add button to accept this constraint.

Continue with entry of the remaining four constraints in a similar manner. When you have entered the Cell Reference (B5) and Constraint value (600) for the last constraint, click the OK button instead of the Add button. The Solver Parameters dialog will reappear, and the constraints you have entered should appear in the Constraints list box, as shown below:

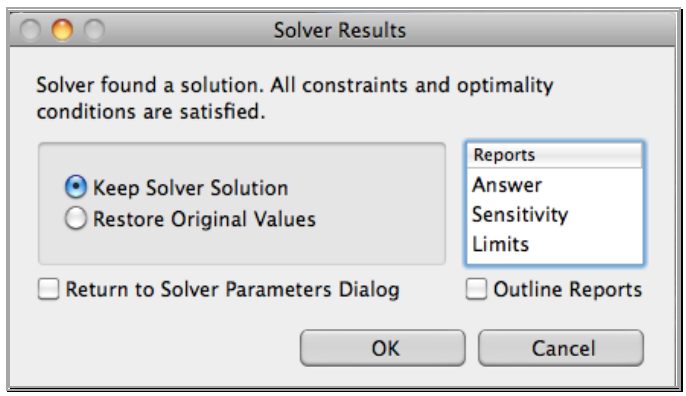


Selecting the Solver Engine

There is a dropdown list of Solver engines available in the main Solver Parameters dialog. The default choice is the standard GRG Solver (which, while slower, is fully capable of solving linear as well as well as nonlinear problems). To choose the LP/Quadratic Solver, click on the down arrow symbol to display the list of Solver engines, and click on the Solver engine of your choice.

Solving the Problem

To find the optimal solution for this LP model, click on the Solve button. After an instant or two, the solution values (A1 = 200, A2 = 200, A3 = 0) should appear in the cells for the decision variables, and the Solver Results dialog should appear, as shown below.



Here you have several choices: You can select one or more reports to be produced from the Reports list box (and check the Outline Reports box, if you'd like the reports outlined);; and you can either discard the solution values (restoring the original cell values) or save the solution values in the decision variable cells. The reports are described in more detail in the chapter "Solver Reports" later in this Guide. For now, click on OK to save the optimal solution in the decision variable cells. You'll then return to worksheet Ready mode, unless you checked the box "Return to Solver Parameters Dialog," in which case the Solver Parameters dialog would reappear, ready to solve another problem.

Congratulations – You've set up and solved a simple but complete Solver problem! The next sections will go into much greater depth on the choices available to you in the Solver Parameters dialog.

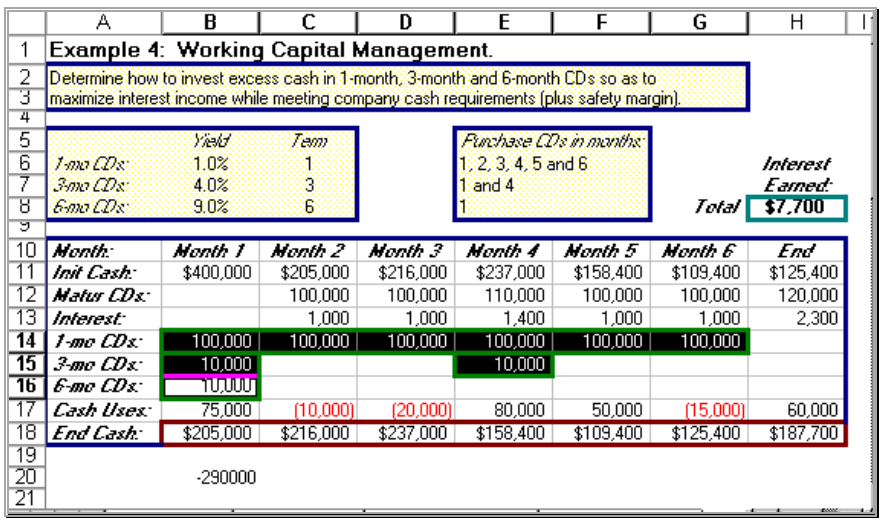
Decision Variables and Constraints

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 used the simplest forms: the decision variables were all adjacent cells in one column, and the constraint right hand sides were constants. In this section, we'll cover more general forms of specifying both variables and constraints.

Variables and Multiple Selections

In the standard Excel Solver, the Solver Parameters dialog provides just one Changing Cells edit box to specify all of the decision variables in a model. This edit box accepts only cell selections, which may be typed in as cell coordinates (or as defined names equivalent to cell coordinates), or entered by clicking with the mouse on the desired cells in the spreadsheet. However, you can use this box to enter the most general form of cell selection permitted by Microsoft Excel, called a *multiple selectio*n. A multiple selection consists of one of more individual selections,

separated by commas (when English is chosen in the Regional Settings – you may be using different settings). Each individual selection may be a single cell, a column or row of cells, or a rectangular set of (contiguous or adjacent) cells. An example of a multiple selection from the "Maximizing Income" sheet in the OptimizationExamples.xlsm workbook included with Microsoft Excel is shown below:



	A	B	C	D	E	F	G	H
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		*Yield*	*Term*		*Purchase CDs in months:*			
6	*1-mo CDs:*	1.0%	1		1, 2, 3, 4, 5 and 6			
7	*3-mo CDs:*	4.0%	3		1 and 4		*Interest Earned:*	
8	*6-mo CDs:*	9.0%	6		1		*Total*	$7,700
9								
10	*Month:*	*Month 1*	*Month 2*	*Month 3*	*Month 4*	*Month 5*	*Month 6*	*End*
11	*Init Cash:*	$400,000	$205,000	$216,000	$237,000	$158,400	$109,400	$125,400
12	*Matur CDs:*		100,000	100,000	110,000	100,000	100,000	120,000
13	*Interest:*		1,000	1,000	1,400	1,000	1,000	2,300
14	*1-mo CDs:*	100,000	100,000	100,000	100,000	100,000	100,000	
15	*3-mo CDs:*	10,000			10,000			
16	*6-mo CDs:*	10,000						
17	*Cash Uses:*	75,000	(10,000)	(20,000)	80,000	50,000	(15,000)	60,000
18	*End Cash:*	$205,000	$216,000	$237,000	$158,400	$109,400	$125,400	$187,700
19								
20		-290000						
21								

The decision variables in this problem are the amounts to invest in 1-month CDs, 3-month CDs and 6-month CDs. We have opportunities to invest in 1-month CDs every month, but 3-month CDs are available only in Month 1 and Month 4, and 6-month CDs are available only in Month 1. To enter all of these cells as decision variables, we need a multiple selection: It must consist of at least three individual selections, separated by commas. Note that although all of the cells to be selected "touch" each other, they cannot be selected as one rectangular area. We could select these cells in several different ways: For example, as (B14:G14,B15:B16,E15) or as (B14:B16,C14:G14,E15). If you display the Solver Parameters dialog for this example, you will see that the Changing Cells edit box uses another selection, (B14:G14,B15,E15,B16). All of these selections are equivalent as far as the Solver is concerned. In general, the areas of a multiple selection must be rectangular, but they need not "touch" each other as they did in the example above. You should avoid entering overlapping areas in a multiple selection: For example, Excel will allow you to enter the above selection as (B14:B16,C14:G14,E14:E15), but the duplication of variable cells will slow down the Solver during problem setup and reporting, and may yield results different from the ones you expected.

There are several ways to enter a multiple selection in a formula or a dialog box: (i) You can simply type the cell coordinates, entering each rectangular area in the form *FromCell* : *ToCell*, separating the areas by commas (or other language-specific separators); (ii) you can select each area by clicking with the mouse, typing a comma between each mouse selection; or (iii) you can make the entire multiple selection with the mouse by pressing the CTRL key as you make the first selection, and holding it down until you have selected all of the rectangular areas.

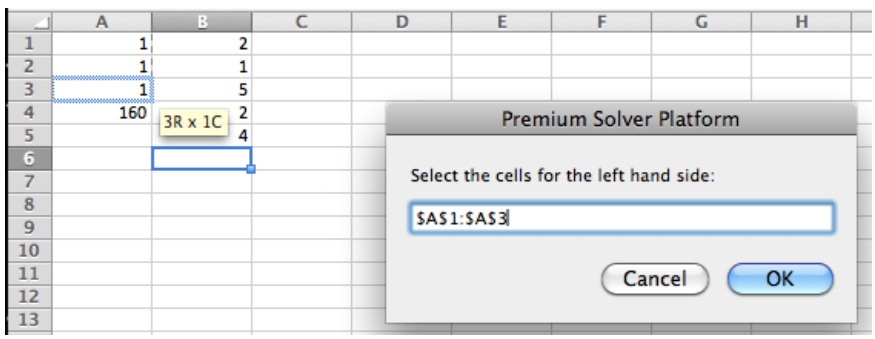
If you enter the individual selections by clicking with the mouse, you'll notice that the cell reference is entered in "absolute" coordinates, such as $B$14:$G$14. Further, you'll find that regardless of whether you include the dollar signs when you

type in the cell coordinates, the cell reference is treated as absolute, and it appears with the dollar signs the next time you display the dialog. "Relative" cell references have significance when you copy a formula from one cell to another, but in the Solver dialogs all cell references are absolute.

Using the Range Selector

Premium Solver Platform for Mac has a convenience feature for selecting cells with the mouse, called the Range Selector. With the Range Selector, you can temporarily "hide" the dialog box where the cell selection will be entered, so that you can more easily see and move about on the worksheet itself. This is most useful for the Set Cell and Changing Cells edit boxes in the Solver Parameters dialog, since that dialog can cover large portions of the viewable worksheet area; but the Range Selector is available in all of the cell selection edit boxes in the standard Solver and Premium Solver Platform for Mac.

You activate the Range Selector by clicking the small rectangular button at the right edge of the cell selection edit box, as shown, for example, in the Solver Parameters dialog just before "Selecting the Solver Engine." This causes the dialog to be hidden and a cell cursor ("thick cross") to appear. The text form of the currently selected range is shown just below and to the right of the cell cursor, as illustrated below:



You can now select whatever cell range you want by clicking and dragging with the mouse. When you release the mouse button, the original dialog will reappear, with the cell selection in the appropriate edit box.

Constraint Left and Right Hand Sides

In specifying the constraints for the sample LP model earlier in this chapter, we entered the constraint left hand sides as single cell references, and the right hand sides as constants 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 *individual* selection, such as a column, row, or rectangular area of cells. Multiple selections are not permitted here. In the example shown earlier, we could have placed the constants 400, 200, 800, 400, 600 in cells C1:C5, then entered all five constraint left hand sides at once as B1:B5, and the five right hand sides as C1:C5.

Overlapping and Conflicting Constraints

You should be careful about entering equivalent or overlapping cell references in the left hand sides of different rows of constraints in the Constraints list box. The only situation where this makes sense is when one constraint uses the >= relation to specify a lower bound, and the other uses the <= relation to specify an upper bound. (Of course, the lower bound must be less than the upper bound, or else there will be no feasible solution to the problem.) If you place multiple lower or upper bounds on the same cells, Premium Solver Platform for Mac will use the "tighter" bounds. For example, if you enter constraints such as A1:A5 <= 10 and A3:C3 <= 5, you've specified both A3 <= 10 and A3 <= 5, so the Solver will use A3 <= 5.

A pair of constraints such as A1:A5 <= 10 and A1:A5 >= 10 has the same effect as A1:A5 = 10, but is considerably less efficient and may cause problems for some of the Solver's advanced solution strategies. Hence, you should always use the form with the = relation in the constraint.

If you specify both a <= or >= constraint and a binary integer or alldifferent constraint for the same group of variable cells, Premium Solver Platform for Mac will display an error message when you try to solve the problem, unless the bounds you specify agree with the binary integer or alldifferent constraint. For example, A1:A5 >= 3 and either A1:A5 = binary or A1:A5 = alldifferent will cause the Solver to display an error message. (The values of variables in an "alldifferent group" must vary from 1 to N, where N is the number of variables; of course, you can always use formulas in other cells to shift this range of values to another range.) As a convenience, you *can* specify a <= or >= constraint for a binary integer variable that further restricts it to *be* either 0 or 1, without getting an error message. "Fixing" variables in this way can be useful when experimenting with an integer programming model.

Constraint Right Hand Sides

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. An (individual) selection such as C1:C5.
4. An arbitrary formula such as C1+1 or C2/D2.
5. "integer", "binary" or "alldifferent"
6. "conic" (Premium Solver Platform only)

Option 5 is for integer constraints only and is discussed below under "Using Integer Constraints." Option 6 is discussed below under "Using Conic Constraints." 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. As we noted in the example shown earlier, you can enter the right hand side values 400, 200, 800, 400 and 600 into cells C1 to C5, and enter 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 Guide. 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. All Solver engines work internally with constant bounds on the constraint functions.

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, both the standard Solver and Premium Solver Platform for Mac provide a Make Unconstrained Variables Non-Negative check box in the Solver Parameters dialog. When this box is checked, all variables that 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 use the Make Unconstrained Variables Non-Negative box to give all other variables zero lower bounds.

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 handled efficiently by all of the Solver engines.

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 Solver model.

On the other hand, a constraint right hand side that is a formula – even a simple one like 2+2 – will incrementally increase the solution time for the model. The Solver treats *any* such formula as a RHS potentially dependent on the variables, and it 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: Because the formula has already been analyzed by Microsoft Excel when it was entered in the cell, the Solver can determine whether it is dependent on the variables.

Using Integer and Alldifferent Constraints

Integer constraints can only be applied to cells that 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 Changing Cells edit box, or the Variable Cell list box. Integer constraints specify that the selected variable cells must have solution values that are integers or whole numbers, such as -1, 0 or 2, to within a small tolerance. Variable cells that have *binary* integer constraints must be either 0 or 1 at the optimal solution. Variable cells subject to an alldifferent constraint must have values from 1 to N, where N is the number of cells specified on the constraint left hand side, and each cell must have a value different from all the others.

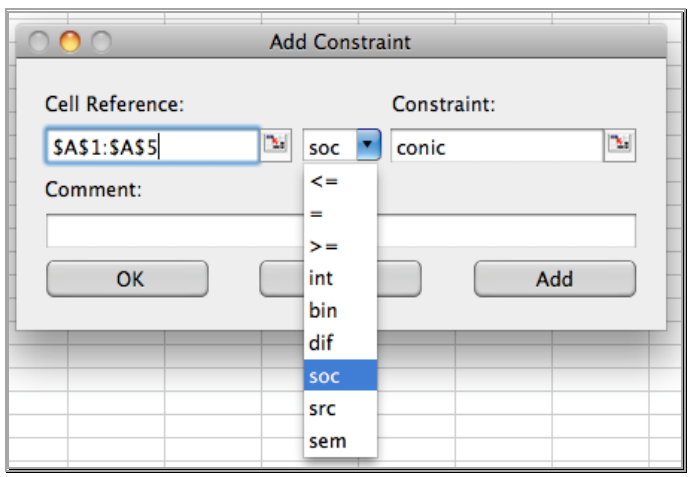
You specify an integer, binary or alldifferent constraint by selecting the "int", "bin" or "dif" choice from the Relation dropdown list in the Add/Change Constraint dialog.

The Solver displays such constraints in the Constraint list box in the form "A1:A5 = integer," "A1:A5 = binary" or "A1:A5 = alldifferent".

Be sure that you select "int", "bin" or "dif" from the Relation dropdown list. If you select = from the dropdown list and *type* the word "integer," "binary" or "alldifferent" 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".

Using Conic Constraints

Conic constraints are a new feature of Premium Solver Platform for Mac, discussed in the previous chapter "Solver Models and Optimization." They can only be applied to cells that 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 Changing Cells edit box, or the Variable Cell list box. To add a conic constraint, you select either the "soc" (second order cone) or "src" (rotated second order cone) choice from the relation dropdown list, as shown below.

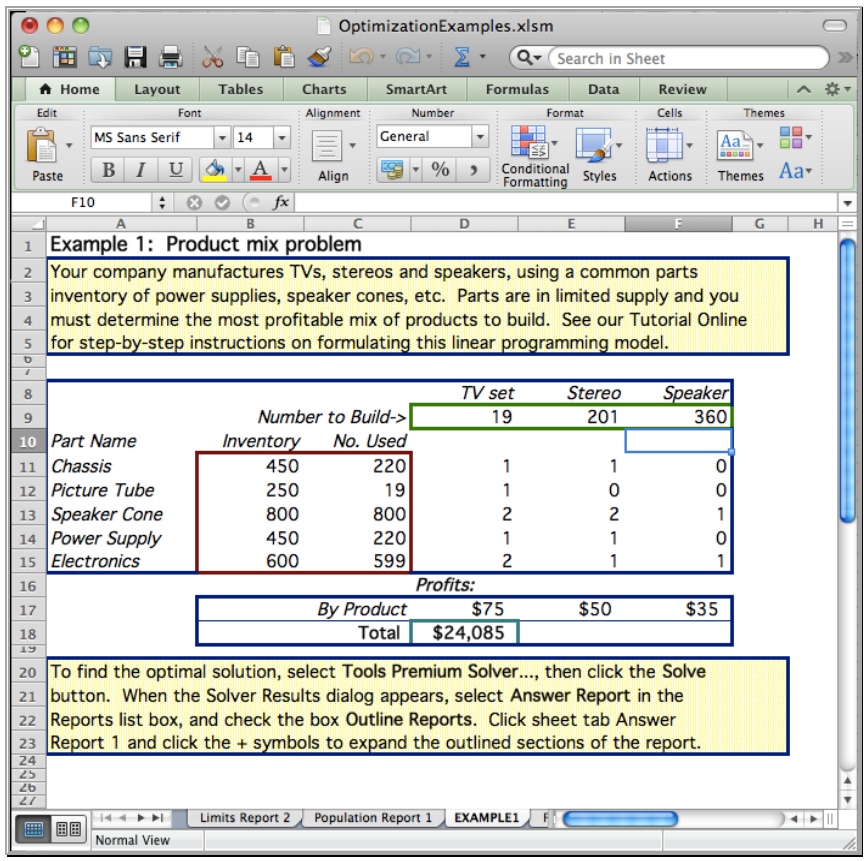


This constraint specifies that the vector formed by *n* decision variables must belong to the second order cone of dimension *n*. An "soc" constraint is equivalent to the formula A1 >= SQRT(SUMSQ(A2:A5)) – in linear algebra, $a_1 \geq \| a_2{:}a_5 \|_2$ – or if A1 is non-negative, A1^2 >= SUMSQ(A2:A5). An "src" constraint is equivalent to the formula 2*A1*A2 >= SUMSQ(A3:A5) – in linear algebra $2a_1a_2 \geq \| a_3{:}a_5 \|_2^2$.

More Readable Models

This section focuses on features of the Solver and Microsoft Excel you can use to build more readable, maintainable, scalable models. The approach outlined above in "From Algebra to Spreadsheets" is the "quick and dirty" way to translate from a model in algebraic form to an equivalent spreadsheet model, ready for optimization. However, that approach will soon prove to be short-sighted when you wish to change the data (for example unit profits or parts on hand), expand the model to include more products or parts, or show the model to someone unfamiliar with the problem or uncomfortable with algebraic notation.

For a better approach to laying out this model, consider the EXAMPLE1 worksheet in the OptimizationExamples.xlsm workbook, shown below:



You are encouraged to open this worksheet in Microsoft Excel and examine its formulas, row, column and cell formatting, and use of labels. If you are not familiar with Excel's "Format Cells" tabbed dialog box, this is a good opportunity to see how it works. Just select one or more cells in EXAMPLE1, choose Format Cells and the appropriate tab to see how the fonts, patterns and borders have been set.

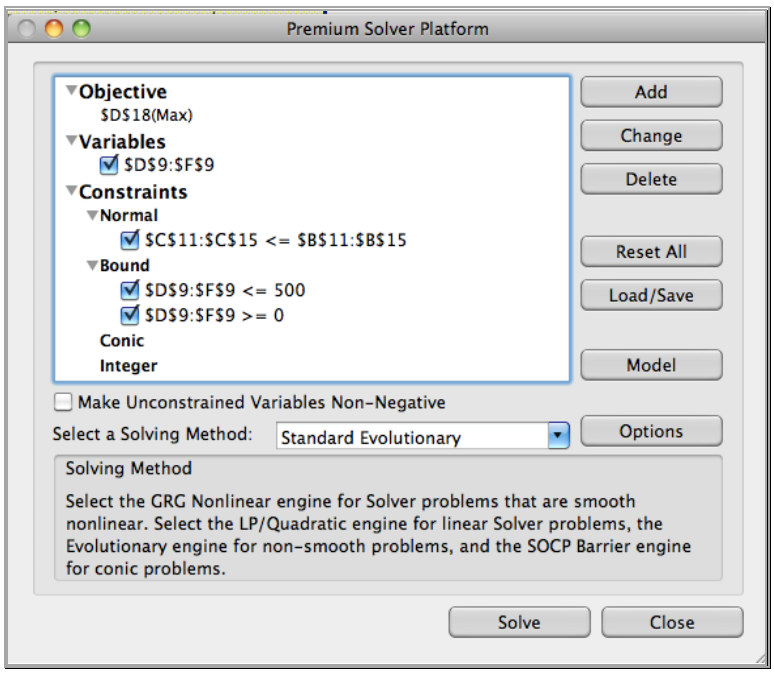
Layout and Formatting

EXAMPLE1 shows one way (not the only way!) to set up an LP model in a more readable and maintainable fashion. To enhance readability, borders and labels have been used to draw attention to the decision variables at D9 to F9, the constraint left hand side formulas at C11 to C15, and the right hand sides at B11 to B15. Your client or management won't miss the objective function calculation at B17 to F18.

EXAMPLE1 is also much easier to maintain and expand than a model constructed with "hardwired" formulas as outlined in the previous chapter. The parts required for each product and the unit profit per product built (i.e. the coefficients of this model) are laid out in cells on the spreadsheet. To add products, you can simply insert new columns in the range of columns D through F; the constraint formulas will "expand" automatically. To add more parts, you can insert new rows between rows 11 and 15, then copy any one of the existing formulas in column C into the new rows.

The SUMPRODUCT function is used in EXAMPLE1 to calculate the value of the objective function and the constraint left hand sides.

If you choose Premium Solver... from the Tools menu, the Solver Parameters dialog for EXAMPLE1 will appear, as shown below:



This dialog illustrates a simple case of the definition of blocks of constraints at one time. There are five constraints of the form C11 <= B11, C12 <= B12, etc., but they can be entered all at once in the Constraints list box. If you haven't previously tried this, click the Delete button to remove the first line in the Constraints list box, then re-enter it with the following steps:

1. Click on Add... to bring up the Add Constraints dialog.
2. With the Cell Reference field ready for input, type C11:C15.
3. Click on the Constraint field (the default <= relation is OK) and type B11:B15.
4. Click on OK to add this block of information.

Using Defined Names

On the worksheet and in the Solver Parameters dialog in EXAMPLE1, the decision variables and the constraint left-hand and right-hand sides were referred to by their cell coordinates. For example, the number of TV sets is at D9, and the number of products built as a whole is D9:F9. You can make your spreadsheets more readable and more flexible by using defined names instead of such cell coordinates.

A defined name is created by selecting the Insert Name Define... menu command and entering the name and the range of cells it should refer to. For example, you could define the name Products to refer to D9:F9, and then type "Products" into the Changing Cells field, or the Cell Reference field of the Add Constraint dialog.

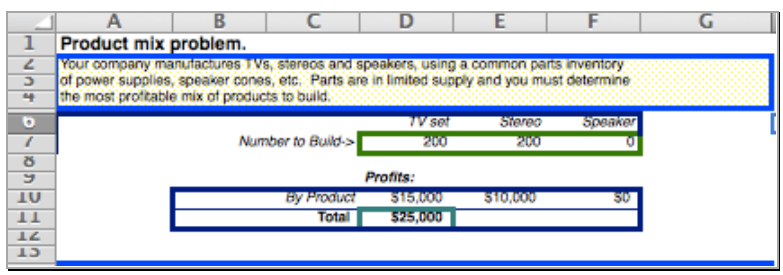
The Insert Name Create... menu command in Excel provides a shortcut way to define a group of names at once. For example, in EXAMPLE1 you could select the range A11 to B15, choose Insert Name Create... and click OK to create the names Chassis for B11, Picture_Tube for B12, Speaker_Cone for B13, Power_Supply for B14 and Electronics for B15.

But you may find it more useful to define names for *blocks* of cells – for example, "Inventory" for cells B11:B15, and "Parts_Used" for cells C11:C15. As you add defined names, the Solver recognizes them, and uses them in preference to cell coordinates in the Set Cell, By Changing Cells, and Constraints boxes. After defining "Total_Profit" for cell D18 and "Products", "Inventory" and "Parts_Used" in EXAMPLE1, you can select Tools Premium Solver… and display a much more readable Product Mix optimization model.

If you take the time to organize and lay out your model in "block form," use defined names for both individual cells and groups of cells, and make effective use of cell borders, colors and other formatting, you'll find that it's easier to maintain your model, and to communicate your results to coworkers, clients and management.

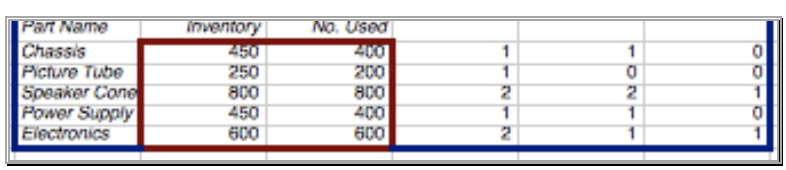
Models Defined Across Multiple Worksheets

Premium Solver Platform for Mac allows you to define decision variables and constraint left hand sides on any worksheet of a workbook. For example, we can split the EXAMPLE1 model illustrated in the previous section into two parts:



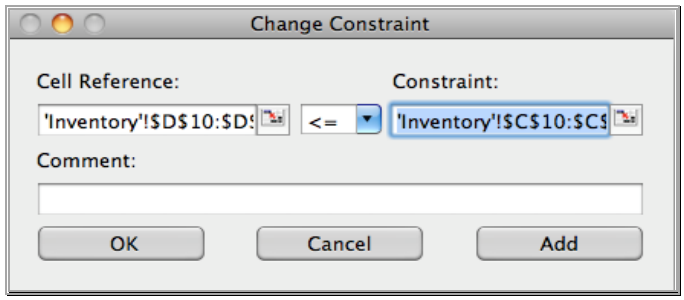
	A	B	C	D	E	F	G
1	**Product mix problem.**						
2	Your company manufactures TVs, stereos and speakers, using a common parts inventory						
3	of power supplies, speaker cones, etc. Parts are in limited supply and you must determine						
4	the most profitable mix of products to build.						
6				*TV set*	*Stereo*	*Speaker*	
7			*Number to Build->*	200	200	0	
8							
9				***Profits:***			
10			*By Product*	$15,000	$10,000	$0	
11			**Total**	**$25,000**			

Worksheet Example1 contains the decision variables, and the coefficients and SUMPRODUCT formula for the objective. Worksheet Inventory contains the constraint left hand sides (formulas), coefficients and right hand sides.

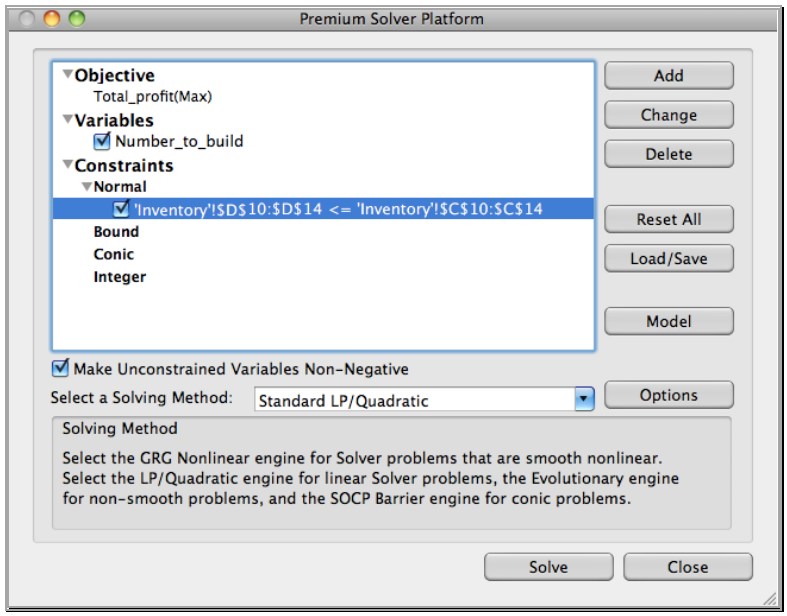


Part Name	*Inventory*	*No. Used*			
Chassis	450	400	1	1	0
Picture Tube	250	200	1	0	0
Speaker Cone	800	800	2	2	1
Power Supply	450	400	1	1	0
Electronics	600	600	2	1	1

Note that the constraint formula refers to the variables as Example1!$D9:$F9. Starting from worksheet Example1, we can display the Solver Parameters dialog, click Add or Change, then simply point and click to select the constraint left and right hand sides on worksheet Inventory, as shown below.



The resulting Solver Parameters dialog looks like this:



Analyzing and Solving Models

Introduction

This chapter explains how to use the Solver Model dialog to analyze and transform your model, and control the solution process. The Solver Model dialog is the user interface to features of the Polymorphic Spreadsheet Interpreter in Premium Solver Platform for Mac, described briefly in the "Introduction."

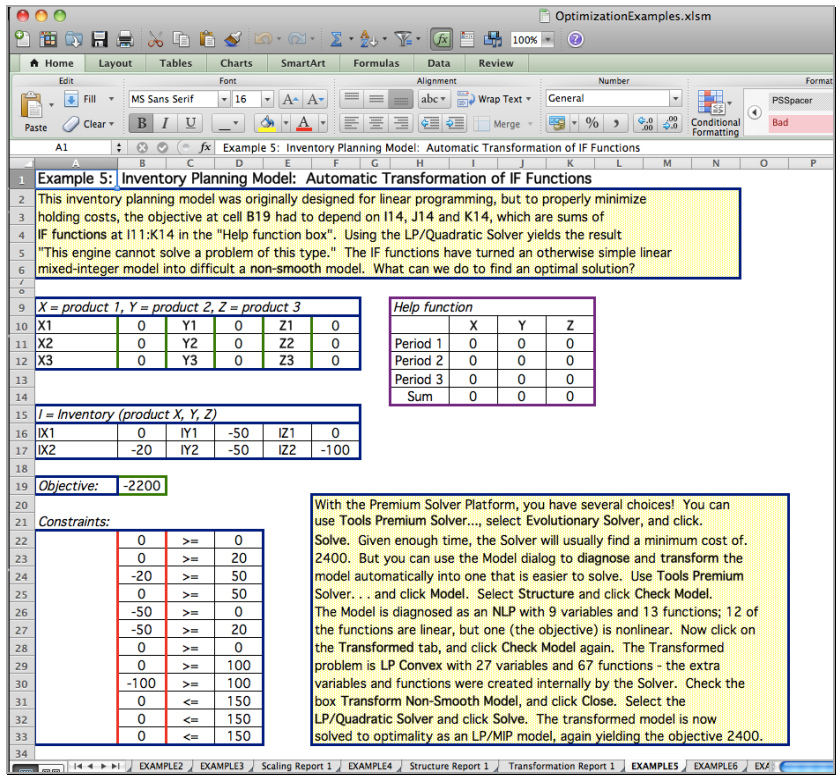
The first sections of this chapter explain how to use the Solver Model dialog to diagnose your model's type (LP, QP, NLP, etc.), sparsity, and convexity; identify "problem formulas" that make your model non-linear, non-smooth or non-convex; automatically *transform* your model to replace uses of certain non-smooth functions with smooth and even linear counterparts; and set options for use of the Interpreter when you solve your model. The last section describes in greater depth how the Interpreter works, in comparison to the Microsoft Excel formula recalculator, and how certain Solver engines take advantage of the Interpreter's greater capabilities.

Using the Solver Model Dialog

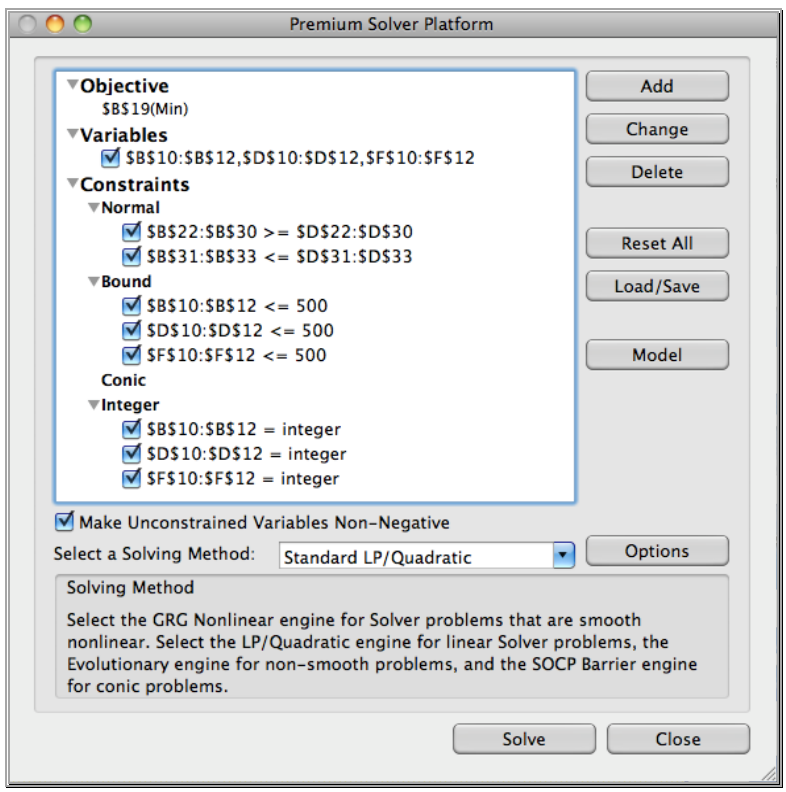
You use the Solver Model dialog to analyze and optionally transform your model, but not to solve it. Through this dialog, you can run the Polymorphic Spreadsheet Interpreter on your model, without running any Solver engines. You can also set options to determine whether and how the Interpreter will be used when you *do* solve your model, by clicking the Solve button in the Solver Parameters dialog.

We can illustrate this process with the EXAMPLE5 worksheet, a simple Inventory Planning model included in the OptimizationExamples.xlsm workbook, installed with the Solver files, shown on the next page. You can easily open this workbook from the Solver Parameters dialog by clicking Help, then clicking Examples.

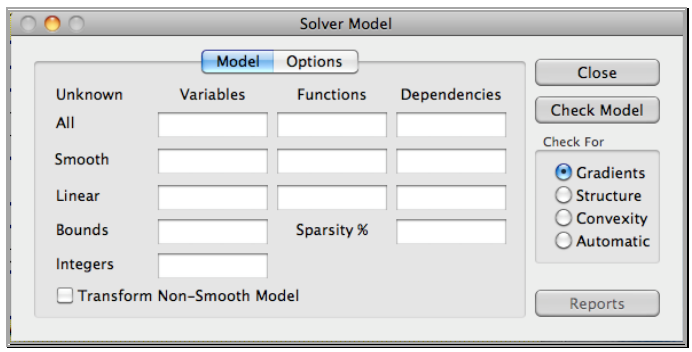
EXAMPLE5 was originally designed to be a linear programming model with a few integer variables – but to properly minimize holding costs, the objective at cell B19 had to depend on I14, J14 and K14, which are sums of IF functions at I11:K14 in the "Help function box." If you attempt to solve this model with the LP/Quadratic Solver, you'll receive the message "The selected engine can not solve a problem of this type. Please select another engine." What can we do to improve this situation?



Pictured below is the Solver Parameters dialog for this model.

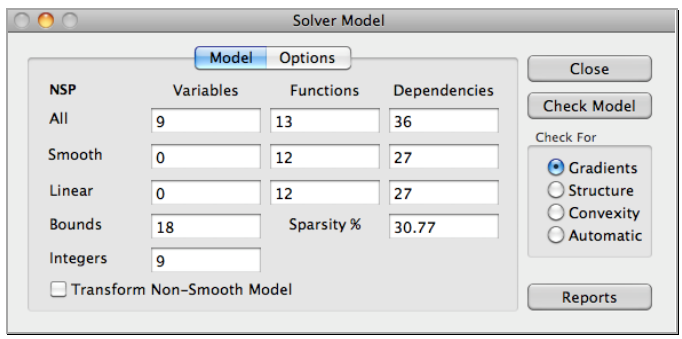


Clicking the **Model** button in this dialog will display the Solver Model dialog as shown on the next page. Clicking the **Solve** button will run the currently selected Solver engine on the current model, using the current settings in the Solver engine's Options dialog, and the current settings for the Polymorphic Spreadsheet Interpreter in the Solver Model dialog.



Analyzing Model Structure

We can get information by selecting the Check For Structure option, and clicking the Check Model button – yielding the dialog shown below. (See "Using the Check Model Button" below for a complete discussion of the analysis performed for the Check For Gradients, Structure, and Convexity options.)



The upper left corner of the Solver Model dialog displays **NSP**, the Interpreter's diagnosis of the type of model on the EXAMPLE5 worksheet. Recall that this model was originally intended to be an LP (linear programming) model. But the IF functions at I11:K14 are not linear functions. In the section "Analyzing Model Exceptions" below, we'll use the Interpreter to pinpoint these IF functions – which could be hard to find in a large Solver model. But first we'll review the displayed statistics, options and buttons in this dialog.

Using Model Statistics

The columns of the Solver Model dialog contain counts of the **Variables**, **Functions**, and **Dependencies** in your model. The rows labeled **All**, **Smooth**, and **Linear** will

display – respectively – (i) the *total* number of variables, functions, and nonzeroes (dependencies) in the model; (ii) the number of *smooth* variables, functions, and dependencies, and (iii) the number of *linear* variables, functions, and dependencies in the model. The Bounds box displays the total number of bounds on variables, and the Integers box shows the total number of integer, binary, and alldifferent variables in your model. The Sparsity % box helps measure the total size and sparsity of your model, as further discussed below.

Types of Functions and Variables

For an explanation of linear, and smooth functions, please consult the section "Functions of the Variables" in the chapter "Solver Models and Optimization." *All functions* includes both non-smooth and smooth functions; *smooth functions* includes all smooth nonlinear, and linear functions. However, *linear functions* include only functions of that type. A decision variable is a "linear variable" if, everywhere that it occurs in formulas of your model, the expression where it occurs (taken alone) would be a linear function. A variable is counted as a "smooth variable" if, everywhere that it occurs, the expression (taken alone) would be a smooth nonlinear, or linear function. For example, if your model had only an objective defined by the formula, =3*A1 + A2^2 + INT(A3), variable A1 would be counted as a linear variable, since the expression 3*A1 taken alone is a linear function; both A1 and A2 would be counted as smooth variables; and variable A3 would appear only in the count of all variables.

Types of Dependencies and NonZeroes

A "nonzero" is counted each time that a given function is found to depend on a given variable. For example, if cell B1 is the objective function, it contains =A1+A2-B2, cell B2 contains =A2*A3, and cells A1:A3 are all variables, this function would contribute 3 nonzeroes to the total count. (Note that a variable, such as A2 in this example, is counted only once per function, even if it is referenced more than once in the function's cell formulas.) Since B1 *depends on* cells A1:A3, the corresponding *partial derivatives* (elements of the Jacobian matrix, as discussed in the chapter "Solver Models and Optimization) will be *nonzero*.

The Sparsity % box contains the results of calculating [All NonZeroes] / ([All Variables] * [All Functions]), expressed as a percentage. Model sparsity was mentioned briefly in the chapters "Introduction" and "Solver Models and Optimization," but here we can describe it more precisely: A *dense* model will have a high Sparsity % and a *sparse* model will have a low Sparsity % figure. The EXAMPLE5 model has a sparsity of 30.77% -- an intermediate figure.

As mentioned in the earlier chapters, large optimization models tend to be sparse in nature. Often, a linear programming (LP) model of over 10,000 variables will have a Sparsity % figure of as little as 2% or 3%. Frontline's Large-Scale LP/QP, Large-Scale GRG, Gurobi, and KNITRO Solvers are designed to exploit sparsity in a model to save memory and solution time, and improve accuracy.

Using the Check Model Button

The three radio buttons in the "Check For" option group determine how much analysis the Interpreter will carry out for your model when you click the **Check Model** button, and when you click the **Solve** button in the main Solver Parameters dialog. Clicking this button with the Transform Non-Smooth checkbox unchecked analyzes the original model; when checked, the interpeter analyzes the transformed model.

- Selecting **Gradients** causes the Interpreter to scan all of the formulas in your model, to determine whether it can compute values and gradients for all of these formulas. If you've used Excel features that the Interpreter does not support – such as circular references, some references to other workbooks, and some user-defined functions – an error message dialog will appear.
- Selecting **Structure** causes the Interpreter to perform the same analysis as Gradients (which may yield an error message), then analyze the structure (dependencies) in your model, fill in the model statistics described above, and classify your model as an LP, QP, QCP, SOCP, NLP, or NSP. (See the chapter "Solver Models and Optimization" for an explanation of these abbreviations.)
- Selecting **Convexity** causes the Interpreter to perform the same analysis as Structure, then seek to determine whether each function in your model is convex or non-convex over the feasible region, as described below. The overall result is displayed in the upper left corner of the Solver Model dialog.

Analyzing Model Convexity

An innovation in Premium Solver Platform for Mac, not available in other modeling systems, is automatic testing of problem functions for convexity. As mentioned briefly in the Introduction, this test may yield conclusive results (that the problem is either **convex** or **non-convex**) or inconclusive results (meaning that the test was not able to prove convexity, nor was it able to prove non-convexity.) A convexity test that always yielded conclusive results would take time that grew exponentially with the number of variables, and hence would be impractical for even modest-size models. The methods used in Premium Solver Platform for Mac are designed to yield useful results in many, but not all cases, while taking a "reasonable" amount of time. The methods used to analyze model convexity are further described later in this chapter, in the section "More on the Polymorphic Spreadsheet Interpreter."

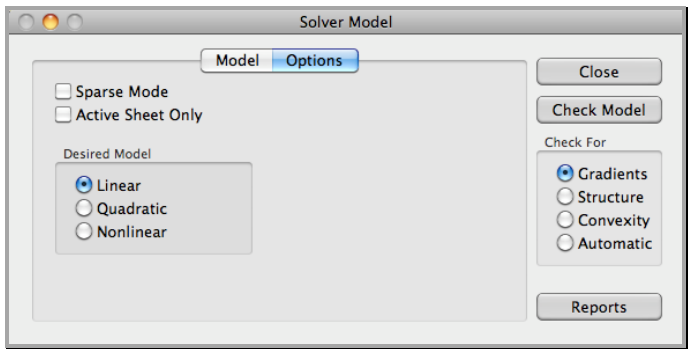
As explained in the Introduction, an optimization model is convex only if *all* of its functions are convex (if the objective is being *maximized* rather than minimized, then this function must be *concave* rather than convex). The overall convexity result for the model is displayed in the upper left corner of the Solver Model dialog following the result of structure diagnosis (LP, QP, QCP, etc.) as "Convex," "NonCvx," or blank if the convexity test is inconclusive. You can also obtain a report of the convexity test results for each problem function, as explained below.

Diagnosis Tab: Analyzing Model Exceptions

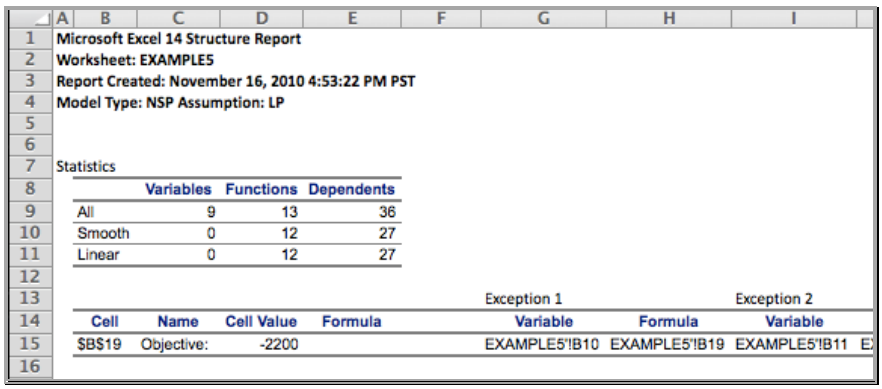
In EXAMPLE5, we intended to create a linear programming (LP) model, but when we clicked the Check Model button, the Interpreter reported that the model is nonlinear (NLP). In a model that we intended to be smooth nonlinear, which we hoped to solve using (say) the Large-Scale GRG or KNITRO Solver, the Interpreter might report that the model is non-smooth (NSP). How do we find and correct the problem in a large Solver model? The Solver Model dialog can tell you which cell formulas contributed to a diagnosis of a model type different from what you intended.

The Structure Report

To find out why EXAMPLE5 was diagnosed as NSP rather than LP, click the Options tab in the Solver Model dialog, which will display the options shown on the next page.



The **Desired Model** option group is used to help you find formulas that are "exceptions" to the type of model that you intended to create. Simply click the desired type of model – Linear, Quadratic, or Nonlinear and click the Check Model button again. This will re-run the analysis. Now, choose the Reports button, and select the Structure report, and hit OK. If you do this for EXAMPLE5, a report like the one shown on the next page will be inserted into your workbook.



Microsoft Excel 14 Structure Report
Worksheet: EXAMPLE5
Report Created: November 16, 2010 4:53:22 PM PST
Model Type: NSP Assumption: LP

Statistics

	Variables	Functions	Dependents
All	9	13	36
Smooth	0	12	27
Linear	0	12	27

				Exception 1		Exception 2
Cell	Name	Cell Value	Formula	Variable	Formula	Variable
$B$19	Objective:	-2200		EXAMPLE5'!B10	EXAMPLE5'!B19	EXAMPLE5'!B11

This report shows that the objective function at cell B19 is an exception to your assumption of a linear model – it is a nonlinear function. Further, it depends nonlinearly on variables B10 and B11 (and possibly others). This dependence was first found at cell B19 on worksheet EXAMPLE5. If your objective formula had referred to a chain of other cell formulas, the report would show you the specific cell where a nonlinear operation or function was first found. EXAMPLE5!$B$19 in the report is a hyperlink – you can click on it to jump to the specific cell in question. In a large model, these links will help you quickly identify the "problem" formulas.

To save space, a maximum of three exceptions of each type is shown in the report for each problem function. After modifying any formulas that were shown as exceptions in the report, you should create a new report to identify any further exceptions to your assumed model type – until the model type changes to the type you expected.

If you've selected **Check For Convexity** instead of Structure when you click the Check Model button (with "Show Exceptions to Desired Model" checked), the Structure Report will additionally list all functions that are diagnosed as non-convex or whose convexity could not be determined. For example, if you select Desired

Model Quadratic and Check For Convexity, the report will list any problem functions that are (i) not quadratic or linear, *and* any that are (ii) quadratic but are not convex.

Transforming a Non-Smooth Model

As described in the chapter "Solver Models and Optimization," the presence of the non-smooth function IF in EXAMPLE5 makes the model much harder to solve. With the Evolutionary Solver in Premium Solver Platform for Mac, you can still solve such models. But where an LP can be solved very quickly and reliably up to very large size, and the solution is basically guaranteed to be optimal, a non-smooth model may take far more time to solve, and there are no guarantees as to whether the solution is truly optimal.

In EXAMPLE5, the IF functions are not *essential* to correctly model the real-world problem: you could use binary integer variables and linear constraints to achieve the same effect (a "fixed-charge constraint"). Techniques for doing this are described under "Improving the Formulation of Your Model" in the chapter "Building Large-Scale Models." Modelers with training in operations research or management science often use these techniques when first formulating their models.

However, if you don't have time to learn these techniques, you may resort to the familiar functions IF, MIN, MAX, ABS, AND, OR, and NOT to model your real-world problem. To improve your results, Premium Solver Platform for Mac can *automatically* apply techniques analogous to those described above, to transform the model from your formulation to a different formulation that is easier to optimize.

To ensure that the transformed problem is well-scaled, it is **important to enter upper and lower bounds for all decision variables** in the Constraints list box. These bounds are used to compute well-scaled values for so-called "Big M" constants that are used in the constraints added during the transformation.

Effects of Model Transformation

If your model includes non-smooth functions such as IF, MIN, MAX, ABS, AND, OR, and NOT, but is otherwise linear, the result of the Platform's automatic transformation will be a linear mixed-integer (LP/MIP) model, with more variables and constraints. This transformed model may be solved by a variety of Solver engines, from the built-in LP/Quadratic and SOCP Barrier Solvers to the Large-Scale LP/QP and Gurobi Solvers. The result may be a much faster solution that is guaranteed to be optimal.

The automatic transformation process is not "magic:" You will still pay a price in solution time for the use of non-smooth functions, because the transformed model will be larger (more variables and constraints) and will include integer variables. As described in the chapter "Solver Models and Optimization" of Premium Solver Platform for Mac User Guide, the presence of integer variables in a model makes it much harder to solve.

However, the automatic transformation from a problem with *non-smooth functions* to a problem with *integer variables* means that the 'arsenal' of Solver engines available to optimize the problem is much larger. Many years of effort by Solver developers to improve the technology of solving LP/MIP models can now be applied to *your* model if it includes IF, MIN, MAX, ABS, AND, OR, and NOT functions.

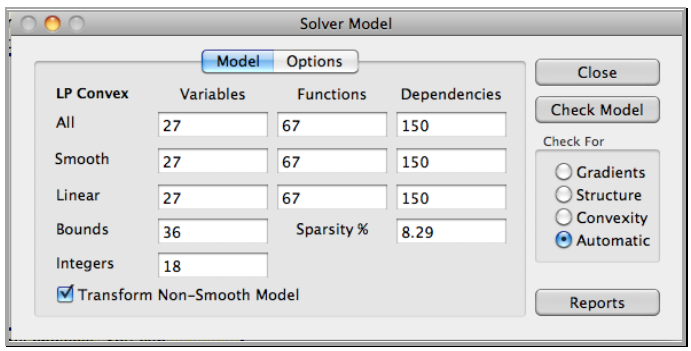
The automatic transformation process uses general-purpose methods to replace IF, MIN, MAX, ABS, AND, OR, and NOT functions, and relational operators such as <,

<=, >= and > with new binary integer and continuous variables, and new constraints. There are still good uses for the techniques described under "Improving the Formulation of Your Model," because these techniques can yield a "tighter" formulation that solves in less time than the automatically transformed version of your model.

If the arguments you supply to IF, MIN, MAX, ABS, AND, OR, and NOT functions are linear functions of the variables, then the new constraints added to the problem will be linear functions; if the arguments you supply are not linear, the transformation process will still work, but the resulting model won't be linear. If you also use other non-smooth functions (for example, CHOOSE, LOOKUP, or the Excel database functions, with arguments that depend on the variables) in your model, the result will still be a non-smooth model, and you will still need the Evolutionary Solver to find a solution.

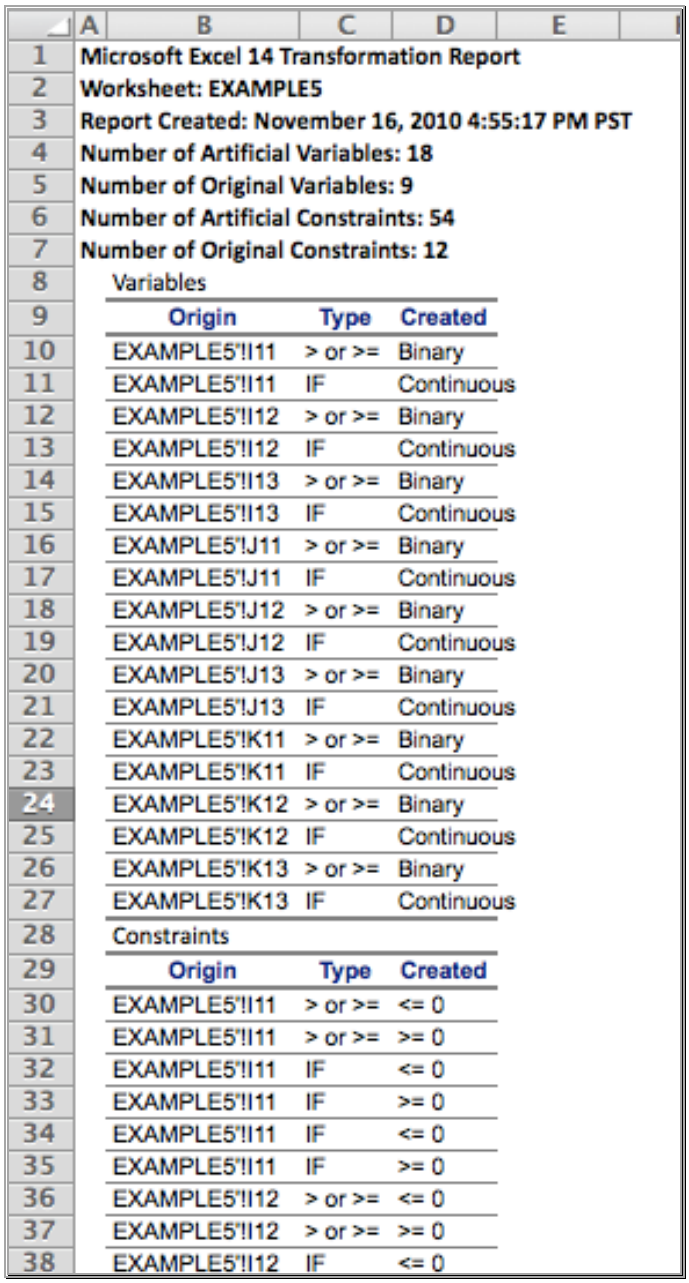
Using Automatic Model Transformation

Just click the Transform Non-Smooth Model check box, and click the Check Model button. If you do this for the EXAMPLE5 model, the Solver Model dialog below will be displayed.



The transformed model has 18 additional variables, 9 of which are new integer variables, and 54 additional constraints – but it is now a linear integer (LP/MIP) model. It's also more sparse than before (8.29% versus 30.77%) – the LP/Quadratic Solver and other sparsity-exploiting Solvers will be able to take advantage of this.

If you're interested in the details of the additional variables and constraints, click the Reports button, and choose the Transformation report. This will add a Transformation Report like the one shown (in part) on the next page to your workbook.



Microsoft Excel 14 Transformation Report
Worksheet: EXAMPLE5
Report Created: November 16, 2010 4:55:17 PM PST
Number of Artificial Variables: 18
Number of Original Variables: 9
Number of Artificial Constraints: 54
Number of Original Constraints: 12

Variables

Origin	Type	Created
EXAMPLE5'!I11	> or >=	Binary
EXAMPLE5'!I11	IF	Continuous
EXAMPLE5'!I12	> or >=	Binary
EXAMPLE5'!I12	IF	Continuous
EXAMPLE5'!I13	> or >=	Binary
EXAMPLE5'!I13	IF	Continuous
EXAMPLE5'!J11	> or >=	Binary
EXAMPLE5'!J11	IF	Continuous
EXAMPLE5'!J12	> or >=	Binary
EXAMPLE5'!J12	IF	Continuous
EXAMPLE5'!J13	> or >=	Binary
EXAMPLE5'!J13	IF	Continuous
EXAMPLE5'!K11	> or >=	Binary
EXAMPLE5'!K11	IF	Continuous
EXAMPLE5'!K12	> or >=	Binary
EXAMPLE5'!K12	IF	Continuous
EXAMPLE5'!K13	> or >=	Binary
EXAMPLE5'!K13	IF	Continuous

Constraints

Origin	Type	Created
EXAMPLE5'!I11	> or >=	<= 0
EXAMPLE5'!I11	> or >=	>= 0
EXAMPLE5'!I11	IF	<= 0
EXAMPLE5'!I11	IF	>= 0
EXAMPLE5'!I11	IF	<= 0
EXAMPLE5'!I11	IF	>= 0
EXAMPLE5'!I12	> or >=	<= 0
EXAMPLE5'!I12	> or >=	>= 0
EXAMPLE5'!I12	IF	<= 0

To solve the transformed model, simply check the box "Transform Non-Smooth Model". Then click the Close button to return to the main Solver Parameters dialog, and click the Solve button. The currently selected Solver engine will be used to solve the transformed problem. (To solve the original problem, return to the Solver Model dialog, and uncheck the "Transform Non-Smooth Model" box.) If you do this for the EXAMPLE5 model, the Solver Results dialog will appear with the message "Solver found a solution. All constraints and optimality conditions are satisfied." Since this is now an LP/MIP model and the Integer Tolerance is set to zero, the solution is globally optimal.

Model Analysis When Solving

The Polymorphic Spreadsheet Interpreter offers many advantages when solving a problem: Besides computing values for your spreadsheet formulas, it can compute accurate *gradients* – which are needed by most Solver engines – at high speed, and it can tell the Solver engine which functions in your model are linear, quadratic, smooth nonlinear, or non-smooth – several Solver engines use this information to realize greater speed or solution accuracy.

Using the Check For Options

The Check For options group determines how much model analysis is done when you click **Check Model** in the Solver Model dialog, or click **Solve** in the main Solver Parameters dialog. The options Gradients, Structure, and Convexity take progressively more time "up front" and yield progressively more information about the model for you on Check Model, or for the Solver engine on Solve. The Automatic setting – often your best choice – allows the Solver engine to choose the option (Gradients, Structure, and Convexity) to be used when solving.

As noted above, the SOCP Barrier Solver, and MOSEK Solver Engine will operate only if the Check For option is set to Structure, Convexity, or Automatic. All other Solver engines can operate with any option setting, but they solve most efficiently with specific settings.

Please see the last section of this chapter for background information on the Polymorphic Spreadsheet Interpreter, and the meaning of "finite differencing," "automatic differentiation," and "dependents analysis" in the following paragraphs.

Check For = Gradients

Choosing Gradients specifies that the PSI Interpreter should "parse" cell formulas on each Solve step, prior to running the selected Solver engine. When this is done, and the Solver engine requests function values and derivatives, they will be computed by the Interpreter; fast, accurate derivatives will be obtained via automatic differentiation. However, no structure or dependencies analysis will be available to the Solver engine.

Check For = Structure

Choosing Structure specifies that the PSI Interpreter should "parse" cell formulas and perform a structure analysis on each Solve step, prior to running the selected Solver engine. When this is done, and the Solver engine requests function values and derivatives, they will be computed by the Interpreter; fast, accurate derivatives will be obtained via automatic differentiation. Further, structure or dependencies analysis information will be available, if the Solver engine requests it.

Check For = Convexity

Choosing Convexity specifies that the PSI Interpreter should "parse" cell formulas, perform a structure analysis, and perform a convexity analysis on each Solve step, prior to running the selected Solver engine. No Solver engine currently requires this option, but it is built into Premium Solver Platform for Mac for future use.

Check For = Automatic

Choosing Automatic – the default – specifies that either the Gradients or the Structure option will be chosen automatically, based on the type of model and the currently selected Solver engine's ability to use the information.

As you may notice when using the Check Model button to diagnose your model, these steps can take some time for larger models – and they can also require significant amounts of memory. Structure analysis takes significantly more time and memory than Gradients analysis. The resources spent on this analysis are often repaid many times over when the Solver engine runs, but this depends on the Solver engine, and also, to some degree, on the model.

For example, the LP/Quadratic Solver and the Large-Scale LP/QP Solver both use Gradients when Solve With = Automatic, but they don't use Structure, since the model is expected to be an LP, and a Structure analysis would simply show that all variables, functions and dependents were linear. But the KNITRO Solver uses Structure when Solve With = Automatic, since they are designed to take advantage of linear dependents in a model that is nonlinear overall.

Options Tab: Using Advanced Options

Sparse Mode

When the Sparse Mode box is checked, the PSI Interpreter operates internally in "Sparse mode," when it is unchecked (the default), the Interpreter operates in "Dense mode." This option affects *only the PSI Interpreter*, not the Solver engines – the latter are typically designed either for dense problems, like the GRG Nonlinear Solver, or for large, sparse problems. Sparse mode also enables a Solver engine to request that **non-smooth variable occurrences should be ignored** when computing derivatives via automatic differentiation.

The Polymorphic Spreadsheet Interpreter can use significant amounts of memory (RAM), especially when diagnosing a model and computing derivatives via automatic differentiation. Memory usage grows with model size, and is greatest when the Solver is running (and using significant amounts of memory itself) and requesting derivatives. If your model is large enough, the memory required may exceed available RAM and cause Windows to beginning swapping to disk – with a severe impact on solution time.

When it operates in "Sparse mode," the PSI Interpreter uses sparse data structures, including "packed" gradient vectors and Hessian matrices, and index lists for the occurrences of variables in problem functions. Extra time is required to perform a Structure analysis (which is required to take advantage of sparsity) and to create the sparse data structures.

For a large, sparse model, Sparse mode can save a significant amount of memory – and if this prevents swapping to disk, it will also save significant time. But if the model is very dense, Sparse mode can actually take more time and memory than Dense mode. Hence, you should check the Sparse box only for models where the Sparsity % figure in the Solver Model dialog is quite low.

If you have a *very* large, sparse linear or quadratic model, you should experiment to see whether the Sparse box yields the best performance. Bear in mind that checking the Sparse box will cause a Structure analysis to be performed before running the Solver engine. It's possible that the Structure analysis will require more time up-front than Sparse mode saves during the solution process.

For the **KNITRO Solver**, Sparse mode has a huge impact on performance. This Solver operates most effectively when it can obtain second derivatives (Hessians) from the Interpreter using automatic differentiation. But this process can consume large amounts of time and memory when the Interpreter is in Dense mode. If a large nonlinear model is sparse – as is usually the case – or if it includes many linear occurrences of variables (which contribute nothing to the Hessian of the function in which they occur), second derivative information can be computed far more efficiently in Sparse mode.

A Solver engine can request that, if Check For = Automatic, the Interpreter will run in Sparse mode *regardless* of the setting of the Sparse check box. The KNITRO Solver makes this request, since the Sparse box is unchecked by default, and Sparse mode is so critical to its performance. You can still force the Interpreter to run in its own Dense mode by setting Check For = Structure (or another choice different from Automatic) and leaving the Sparse box unchecked. But we recommend that you run most Solver engines with the default settings (Check For = Automatic), which will yield the best performance in the majority of cases.

For the **Evolutionary Solver**, Sparse mode can also have an important impact. Typical models for the Evolutionary Solver – which is limited to 500 decision variables – are not large enough to require this option for memory-saving purposes. But when the PSI Interpreter operates in Sparse mode, the Evolutionary Solver can – and will – ask the Interpreter to *ignore non-smooth variables* in automatic differentiation. The effect of this is to "fix" the non-smooth variables, making their partial derivatives zero, and to allow the Solver to proceed with automatic differentiation of any non-smooth function.

So, if the Evolutionary Solver stops with the message "Solver encountered an error computing derivatives," you should check Sparse Mode box in the Solver Model dialog, and click Solve again.

Active Sheet Only

When the Active Sheet Only box is checked, the PSI Interpreter will evaluate cells only on the active (frontmost) worksheet in the active workbook. Cells on other worksheets in the active workbook, or on sheets in other workbooks, that are referenced in formulas making up the Solver model will be treated as *constant* in the problem.

When the Active Only box is unchecked (the default), the Interpreter will evaluate all cells, on all worksheets, referenced in formulas involved in the Solver model. If the model references cells on sheets in other workbooks, these workbooks will be opened if available; otherwise the external cell's "last known value" (as stored in the active workbook) is used and treated as constant in the model.

This box should be checked only if you have a large model that is spread across multiple worksheets, and you *want* all cells on worksheets other than the active sheet to be treated as constant in the problem. Note that, if these cells actually contain formulas that depend on the decision variables, this fact will be *ignored* and you will be solving a problem where these cells are effectively held constant at their last known values. If you have a large number of such cells on other worksheets, and especially if they contain formulas that do not depend on the decision variables, checking this box will save time in the Interpreter.

More on the Polymorphic Spreadsheet Interpreter

The Polymorphic Spreadsheet Interpreter in Premium Solver Platform for Mac fundamentally changes the way the Solver operates, and it affects – often dramatically – the performance of both the built-in and field-installable Solver engines. This optional section will give you more insight into how the Interpreter works, in comparison to the Microsoft Excel formula recalculator, and how certain Solver engines take advantage of the Interpreter's considerably greater capabilities. To appreciate this section, you may need to review "Functions of the Variables" in the chapter "Solver Models and Optimization."

The Microsoft Excel Recalculator

Microsoft Excel includes an "Interpreter" of its own for Excel formulas, that is usually referred to as the formula recalculator. The recalculator is used to compute up-to-date values for formulas in your model whenever you enter or edit information in spreadsheet cells (when Excel is in "Automatic Calculation mode") or when you press the F9 (Calc Now) key. As the standard "Interpreter" from Microsoft, it computes values for every kind of formula syntax or function that is legal in Microsoft Excel. It is controlled by options on the Calculation tab in the Tools Options dialog in Excel.

While it is invoked automatically when you work interactively with your spreadsheet, the Microsoft Excel recalculator can also be invoked programmatically, by VBA code or by an add-in such as the Solver. Indeed, the Solver traditionally worked by writing new values into cells for decision variables, asking Excel to recalculate the model, then reading the computed values of cells for the objective and constraints.

Although it is fast and accurate, the Microsoft Excel recalculator has a specific and limited purpose: To calculate function *values* in formula cells, given new values for other cells. It does not perform other tasks such as computing function *derivatives*, or analyzing formulas for linear or nonlinear dependents.

Finite Differencing

Since the Microsoft Excel recalculator computes only function values, but most Solver engines require both function values and function derivatives, the Excel Solver has traditionally used the Excel recalculator to compute *approximations* of partial derivatives, using the method of *finite differencing*. This method is based on the definition of the partial derivative of a function f with respect to a variable x_j:

$$\partial f / \partial x_j = \lim_{\Delta \to 0} \frac{f(x + e_j\Delta) - f(x)}{\Delta}$$

where x represents the vector of decision variables $[\, x_1\, x_2\, \ldots\, x_n\,]$ and e_j is a unit vector (with 1 in the jth position and 0 elsewhere). While the definition applies only in the limit when Δ goes to zero, an *approximation* of the partial derivative can be computed by choosing a very small value such as 10^{-8} for Δ. So the Solver uses the following steps:

1. Set the cells for the decision variables to $x = [\, x_1\, x_2\, \ldots\, x_n\,]$.
2. Ask Excel to recalculate the model, thereby computing $f(x)$.
3. Set the cell for the jth variable to the "perturbed" value $x_j + \Delta$.
4. Ask Excel to recalculate the model, thereby computing $f(x + e_j\Delta)$.

5. Compute the difference of $f(x + e_j\Delta)$ and $f(x)$, divided by Δ.

These steps compute a partial derivative with respect to *one* variable. To compute the function *gradient* – the partial derivatives with respect to *all* of the variables – steps 3 through 5 above must be performed n times if there are n decision variables.

Most Solver algorithms require the gradient of the objective *and* the gradients of all the constraints. That is, they require the *Jacobian* matrix of partial derivatives, where each matrix row is the gradient of one function (see "Derivatives, Gradients, Jacobians, and Hessians" in the chapter "Solver Models and Optimization"):

$$\begin{vmatrix} \partial f_1/\partial x_1, & \partial f_1/\partial x_2, & \ldots, & \partial f_1/\partial x_n \\ \partial f_2/\partial x_1, & \partial f_2/\partial x_2, & \ldots, & \partial f_2/\partial x_n \\ \ldots & & & \\ \partial f_m/\partial x_1, & \partial f_m/\partial x_2, & \ldots, & \partial f_m/\partial x_n \end{vmatrix}$$

This is not quite as expensive in computing time as it looks, because when the Solver asks Excel to recalculate the model at steps 2 and 4 above, Excel will calculate values for all of the problem functions at once. So the Solver can obtain approximate values for all partial derivatives by performing steps 1 – 2 once, and steps 3 – 5 n times (once for each variable). In other words, the Solver obtains values for all of the partial derivatives in one column of the Jacobian matrix each time it asks Excel to recalculate the model at step 4.

For more than a decade, the Excel Solver and Premium Solver have used the finite differencing method to successfully solve optimization problems. But the method does have several drawbacks:

- It is relatively slow, since the model must be recalculated $n + 1$ times (and when solving a nonlinear problem, this must be done at each Trial Solution).
- It is relatively inaccurate, since the subtraction and division typically result in a loss of significance – in the worst case *half* of the significant digits are lost.
- If the Solver algorithm needs the gradient of only *one* function at each Trial Solution (perhaps because the constraints are all linear, with constant gradients), this takes as much time as it would to compute gradients of *all* the functions.
- Each time it recalculates, Excel will compute values for *all formula cells in the spreadsheet* that depend on the perturbed decision variable cells – even cells that do not participate in the objective and constraints.
- Computing second order partial derivatives (the Hessian matrix, as described in the chapter "Solver Models and Optimization") is not practical – this would require n^2 worksheet recalculations (a million for a 1,000-variable problem!) at each Trial Solution, and would yield derivative values of very low accuracy.

The slowness of finite differencing directly impacts solution time – especially for nonlinear problems, where finite differencing is performed many times. Since the Solver uses derivative values to determine the direction in which to search, the loss of accuracy in derivatives can lead to less-than-ideal search directions. While most Solver algorithms can "correct course" as they proceed, by computing a new search direction at each Trial Solution, less accurate derivatives will often mean that more major iterations will be needed to make the "course corrections," and they may lead to less accurate final solutions.

Because many Solver models in Excel are really just part of a larger spreadsheet model that has many formulas calculating values of interest for other purposes, but not participating in the optimization problem, often the greatest drawback of using

the Excel recalculator is the fact that it always computes values for every formula cell that depends on the perturbed decision variable cells.

To achieve greater speed, accuracy, and control of the computation of derivative values, and to make it possible to evaluate the Solver model in other ways – for example, to determine linear and nonlinear dependents– Frontline Systems developed its own Interpreter for Microsoft Excel.

The Polymorphic Spreadsheet Interpreter

The Polymorphic Spreadsheet Interpreter in Premium Solver Platform for Mac reads cell formulas, in the form that you write them such as =A1*SUM(B1:B5)/EXP(-C1), and translates them into a compact intermediate code that can be processed efficiently each time that function values or derivatives are needed. It also builds a symbol table of names and cell references, used to look up current cell values and identify occurrences of decision variables.

The Interpreter acts in response to requests from Solver engines for function values and derivatives – or in response to your requests for a Gradients, Structure, or Convexity analysis, when you click the Check Model button. It scans the intermediate code for one or more problem functions (objective and constraints), and computes numeric values, using the current values of the decision variables (set by the Solver engine) and constants, arithmetic operators, and the like in the intermediate code.

The Microsoft Excel recalculator and the PSI Interpreter are both designed to be very efficient. But where Excel reads and translates every cell formula that you create in a spreadsheet, the Interpreter translates only the cell formulas that are involved in calculating your objective and constraints (as you've defined them in the Solver Parameters dialog). And where Excel always computes values for every formula cell in the spreadsheet that depends on the changed decision variable cells, the Interpreter computes values for only the functions that the Solver engine actually needs. Because of this, on a spreadsheet where there are many cell formulas that aren't directly involved in the optimization model, the Interpreter is usually faster than the Excel recalculator when computing function values. But the Interpreter's greatest benefit by far lies in computing function derivatives.

Automatic Differentiation

When the PSI Interpreter computes partial derivatives for your objective and constraints, it uses a very different approach than the finite differencing method outlined earlier, called *automatic differentiation* in the technical literature. In essence, the PSI Interpreter *computes derivatives at the same time that it computes values* for functions, using algebraic relationships such as:

- Sums: $\partial[f(x) + g(x)]/\partial x = \partial f(x)/\partial x + \partial g(x)/\partial x$
- Products: $\partial[f(x) * g(x)]/\partial x = \partial f(x)/\partial x * g(x) + \partial g(x)/\partial x * f(x)$
- Exponents: $\partial x^n/\partial x = n * x^{n-1}$
- Trig functions: $\partial sin(x)/\partial x = cos(x)$, $\partial cos(x)/\partial x = -sin(x)$, etc.

The PSI Interpreter implements both "forward mode" and "reverse mode" automatic differentiation (further described in the technical literature), for both first partial derivatives (the Jacobian matrix) and second partial derivatives (the Hessian matrix). These partial derivatives are *computed to the same accuracy as the function values* themselves – hence, Solver engines can sometimes find the optimal solution with fewer Trial Solutions than required when finite differencing is used. And because of

the way derivatives are computed, the time required is dramatically less than the time required for finite differencing – especially for "reverse mode" automatic differentiation (which is used for all expressions except array formulas).

Thanks primarily to automatic differentiation, on a sample of small and medium-size actual user models, total solution times (which include much more than the time spent computing derivatives) for Premium Solver Platform for Mac were on average *twice as fast* for linear problems and *seven times faster* for nonlinear problems. Since the speed advantage of automatic differentiation *grows* with the number of variables in the problem, larger models may experience even greater speed gains (provided that they are run on PCs with sufficient RAM for the Interpreter).

Model Diagnosis and Structure Analysis

The PSI Interpreter is also responsible for diagnosing your model as linear (LP), quadratic (QP), quadratically constrained (QCP), second order cone (SOCP), smooth nonlinear (NLP) or non-smooth (NSP) and providing statistics on linear, quadratic and smooth variables, functions and nonzeroes that you see when you click the Check Model button in the Solver Model dialog. And it provides model type, sparsity and "Structure analysis" information to Solver engines during the solution process, when you select the Check For = Structure option.

The way the PSI Interpreter does this is very similar to the way it computes derivatives via automatic differentiation. The Interpreter computes symbolic "values" for the dependence of functions on decision variables for every cell formula in your model, using algebraic relationships such as:

- Sums: *The sum of two linear functions is a linear function*
- Products: *The product of a constant (independent) function and a linear function is a linear function; the product of two linear functions is a quadratic function*

The PSI Interpreter is also responsible for the Scaling Report, described later in this Guide in the chapter "Solver Reports." It computes symbolic "values" for every cell formula in your model, based on the magnitudes of the values of decision variables, and algebraic relationships that capture the effect of addition, multiplication, and similar operations on the magnitudes of function results. This is another unique capability of Premium Solver Platform for Mac.

Convexity Analysis

Finally, the PSI Interpreter is responsible for the new, automatic test for convex models and functions in Premium Solver Platform for Mac. As mentioned earlier, this test may yield conclusive or inconclusive results; a convexity test that always yielded conclusive results would take time that grew exponentially with the number of variables, and hence would be impractical for even modest-size models. The methods used in Premium Solver Platform for Mac are designed to yield useful results in many, but not all cases, while taking a "reasonable" amount of time. In the worst case, the test involves computing the interval Hessians of all of the problem functions and performing interval vector-matrix operations on each of these Hessians.

A **linear function** is always convex (and concave), and its Hessian matrix is always zero. Hence the convexity test takes no extra time for linear functions beyond the analysis done for the Check For Structure option.

A **quadratic function** has a constant Hessian matrix, which means that the interval Hessian is the same as the real Hessian, and the convexity test will yield a conclusive result, based on the positive (or negative) definiteness of the Hessian matrix. (A

positive definite or semidefinite Hessian means that the quadratic function is convex; a negative definite or semidefinite Hessian means that the function is concave.)

For **general smooth nonlinear functions**, the convexity test first computes an "outer approximation" of the feasible region, which can be pictured as a box (bounds on the decision variables) that encloses the actual feasible region determined by the intersections of the constraints. (The Interpreter starts with the variable bounds that you specify, then uses constraint propagation methods to "shrink" this box.)

The convexity test then quickly computes a result based on the sign of the interval Hessian over this box. For some – but not all – functions, this is sufficient to determine the convexity of the function. If this test is not sufficient, the full interval Hessian is computed, and several numerical methods are applied to test whether this interval matrix is positive (or negative) definite.

The convexity test is yet another capability of Premium Solver Platform for Mac that is not available in other, far more expensive modeling systems and Solvers.

Excel Built-in Functions

Microsoft Excel has over 320 built-in functions, including the financial, statistical, and engineering functions that are part of the Excel Analysis ToolPak. The Interpreter supports almost all of these functions. Functions that are recognized but not supported include:

CALL	HYPERLINK
CELL	OFFSET
GETPIVOTDATA	REGISTER.ID
INDIRECT	SQLREQUEST
INFO	CUBExxx (Excel for Mac 2011)

The most commonly used of these functions – OFFSET and perhaps INDIRECT – are not supported because they can reference arbitrary cell "addresses," and the Interpreter is designed to process only the formula cells that participate in calculation of your objective and constraints, not the full spreadsheet "grid."

Since the Interpreter computes values for your problem functions without using the Microsoft Excel recalculator, how confident can you be that the values computed this way will match the values that would have been computed by Excel? While discrepancies are always possible, one reason for confidence is that Frontline Systems actually developed, under contract to Microsoft, implementations of most of the Excel built-in functions for use in the Internet Explorer "spreadsheet component" that is included with Microsoft Office 2000, XP, 2003 and 2007 (file msowcf.dll in the Microsoft Office program directory). Frontline's implementation of these functions was tested against the same functions in Microsoft Excel, in an extensive quality assurance process during the development of Office 2000. Because of this, Frontline Systems was uniquely qualified to develop a full-scale Interpreter for Microsoft Excel and its extensive library of built-in functions.

When a Solver engine displays a final solution with the Solver Results dialog box, or an intermediate solution with the Show Trial Solution dialog box, the current values of the decision variables are placed in cells on the spreadsheet, and at this point the *Microsoft Excel recalculator* is used to compute values for the objective and constraints – even when the Interpreter has been used internally during the solution process. So you can be 100% confident that the values you see won't change when you save and later reopen your workbook!

Building Large-Scale Models

Introduction

It's a maxim that a successful Solver model will grow in size over time. When the initial results from an optimization model demonstrate ways to achieve significant cost savings, improved schedules, higher quality or increased profits, management is naturally interested in applying these methods to bigger problems. This might involve extending the model to include more plants, warehouses, assembly lines, or personnel; to bring in other divisions or geographic regions; or to cover more time periods, more detailed process steps, or more specific parts or products. The result is an increase in the number of decision variables, constraints, and cells in your model.

When your model grows in size, it becomes more challenging to design and maintain, and also more challenging to solve. Good modeling practices – touched upon in the chapter "Building Solver Models" – become *far* more important, so your model remains comprehensible to other Excel users, auditable for errors, and easy to modify. Issues such as your model type (LP, QP, QCP, SOCP, NLP or NSP), sparsity, and scaling also become *far* more important, since they strongly influence the time it takes to solve your model, and the reliability of the solutions you obtain.

This chapter can only briefly survey good modeling practices – entire books have been devoted to this subject (we will recommend some). It focuses on steps you can take to obtain faster and more reliable solutions for large models Premium Solver Platform for Mac, including:

- Steps towards better performance that are easy to apply in most situations
- Steps you can take – with more design and modeling effort – to improve the *formulation* of your model, by replacing non-smooth or nonlinear constraints with linear (or integer linear) constraints
- Steps you can take to enable Premium Solver Platform for Mac to analyze your model more efficiently

Designing Large Solver Models

A large Solver model in Microsoft Excel is both a large spreadsheet workbook and a large optimization model. If you plan to build such a model, you'll be well advised to learn about good spreadsheet modeling practices, and about good optimization modeling techniques.

We highly recommend the textbook *The Art of Modeling with Spreadsheets: Management Science, Spreadsheet Engineering, and Modeling Craft* by Stephen G. Powell and Kenneth R. Baker, published by John Wiley & Sons, listed at the end of the chapter "Introduction." Unlike other management science textbooks, this book teaches you "best practices" in modeling and spreadsheet engineering, as well as techniques of linear and nonlinear optimization using Excel.

Other books on good spreadsheet design are hard to find, but through resources like the Amazon.com Marketplace, you may be able to locate a copy of John Nevison's book *Microsoft Excel Spreadsheet Design* (Prentice-Hall, 1990), or his earlier works *1-2-3 Spreadsheet Design* (1989) or *The Elements of Spreadsheet Style* (1987), both of which are still useful in designing modern spreadsheets. A relatively new (2003) book, *Excel Best Practices for Business* by Loren Abdulezer, includes chapters on spreadsheet construction techniques, "makeovers" of spreadsheets developed by others, and spreadsheet auditing.

Training courses in Microsoft Excel often cover at least some elements of good spreadsheet design. They are offered in many venues, from universities and community colleges to public seminars, in-house corporate training, and classes sponsored by computer dealers. Check the course outline or syllabus to see if it features spreadsheet design and good modeling practices, and other topics most relevant to you.

A readily available book on optimization modeling techniques is H. Paul William's *Model Building in Mathematical Programming, 4th Edition* (John Wiley, 1999), listed at the end of the chapter "Introduction." Focusing on modeling for linear and integer programming problems, it includes a treatment of large-scale model structure and decomposition methods that is hard to find elsewhere.

Spreadsheet Modeling Hints

Below is a brief set of suggestions for planning, designing and constructing large Solver models:

Start with a Plan. Plunging in and entering numbers and formulas immediately will quickly lead to problems when constructing a large spreadsheet. Write down your objectives and sketch out a design before you begin working on the real spreadsheet.

Build a Prototype. Plan in advance to build a prototype, *throw it away*, and then build the real spreadsheet model. What you learn from building and solving the prototype will probably save you time in the long run.

Create a Table of Contents. In the upper left corner of your first worksheet, include comments that point readers to the major areas or sections of the spreadsheet.

Separate Data and Formulas. Avoid using constants in formulas, unless they are intrinsic to the mathematical definition of the function you are using. Instead, place constants in cells, and refer to those cells in formulas. Create separate areas on the spreadsheet for input data and for calculations, and identify these with distinct colors, borders or shading.

Document Assumptions, Parameters and Methods. As John Nevison suggested, seek to "surface and label every assumption" in your model. Use labels or cell comments to document key formulas and complex calculations.

Use defined names. Use Excel's Insert Name Define and Insert Name Create commands to assign meaningful names to individual cells and cell ranges. This will help make your formulas clearer and more flexible.

Use and Separate Two-Dimensional Tables. Many elements of your model will lend themselves to a row-column table representation. Create separate table areas, with distinct colors, borders and shading. Collect non-table data (such as individual parameters) into a separate area.

Use Excel Tools to View and Audit Your Spreadsheet. Use the View Zoom command to get a high-level view of your spreadsheet's structure. Use the View tab in the Tools Options dialog to display formulas instead of values, and scan them for consistency. Learn to use the Auditing Toolbar (Tools Auditing…) to trace precedents and dependents of your formulas.

Use a Spreadsheet Auditing Tool. Several auditing tools are available, including *SpACE* from the UK Customs and Excise Audit unit, *OAK* from Operis Ltd. in the UK, and the *Spreadsheet Detective* from Southern Cross Software in Australia.

Optimization Modeling Hints

Identify Your Model's Index Sets. Your decision variables, constraints, and many intermediate calculations will fall into groups that are *indexed* by elements such as products (A, B, …), regions (North, South, …), time periods (January, February, …) and similar factors. Identify and write down these index sets and their members. Then organize the columns and rows of your table areas using these index sets. Use the top row and left column of each table area for index set member names as labels.

Identify Your Decision Variables. Once you've identified the quantities that will be decision variables, and how they are indexed (for example, units made by product A, B,… or shipments by region North, South,…), it's usually easier to determine the constraints and their indexing.

Determine the Data You'll Need. In building large optimization models, you will frequently spend a good part of your time figuring out what data you need, how you will get it (and keep it up to date), and how you'll have to summarize or transform it for the purposes of the model. This may involve getting help from your IT department or from other groups that create or maintain the data.

Define Balance Constraints. It is easy to overlook "balance" or "continuity" constraints that arise from the physical or logical structure of your model. For example, in a multi-period inventory model, the ending inventory at time t must equal the beginning inventory at time $t+1$. At each node of a network model (such as a warehouse), the beginning item quantity plus incoming deliveries minus outgoing shipments must equal the ending item quantity ("what goes in must come out").

Learn to Use Binary Integer Variables. Many relationships that you might find difficult to model at all, and many where you might otherwise use IF, CHOOSE or other non-smooth or discontinuous functions, can be effectively modeled with binary integer variables. The section below "Improving the Formulation of Your Model" describes many situations where you can use such variables to organize your model.

Using Multiple Worksheets and Data Sources

Large Solver models and their data are often organized into multiple worksheets of a single workbook. Some large models reference data found in other workbooks. Given the large number of data elements, the sources from which you are getting the data, and the procedures you use to keep the data up to date, multiple worksheets are often necessary or at least useful for organizing your data.

Premium Solver Platform for Mac allows you to define decision variables and constraint left hand sides on any worksheet of a workbook. For this and many other reasons, you are well advised to upgrade to Premium Solver Platform for Mac if your model grows in size. With either product, the formulas in your objective and constraint cells can refer to cells on other worksheets, and those cells on other worksheets can contain formulas that depend, directly or indirectly, on decision variable cells. For more information, see "Models Defined Across Multiple Worksheets" in the chapter "Building Solver Models."

Several commentators on good spreadsheet modeling practice feel that models defined on a single worksheet are easier to understand and maintain. In Excel for Mac 2011, a single worksheet can have up to 16,384 columns and 1,048,576 rows. So you may want to keep the core of your Solver model – the formulas (i) that are used to compute your objective and constraints and (ii) that depend on the decision variables – on a *single worksheet*. If you find that you can better structure your model by placing decision variables and constraints on different worksheets, it's highly recommended that you adopt a consistent scheme for choosing blocks of variable and constraint (and other formula) cells, and referencing these cells across worksheets.

Some of the data you need may be available in relational databases, OLAP databases or data warehouses. Microsoft Excel provides rich facilities, such as external data ranges and PivotTables, to bring such data into an Excel worksheet. The raw data, even if partially summarized from database records or transactional data, often needs to be further transformed and summarized on your worksheet(s). This is usually easy to do with Excel formulas. But for clarity in your model, we recommend that you use separate worksheet areas, with distinct colors, borders or shading, for formulas that simply massage the data and do not participate in the solution process (i.e. do not depend on the variables). The Solver can determine which formulas depend on the variables, but *you* or your colleagues may find it difficult to do so if the formulas are intermixed.

Quick Steps Towards Better Performance

The rest of this chapter focuses on steps you can take to obtain *faster and more reliable solutions* for large models from Premium Solver Platform. This section describes steps that are easy to apply in most situations.

For users of Premium Solver Platform for Mac, we highly recommend that you *try solving your model with our field-installable Solver Engines* – especially the KNITRO Solver, MOSEK Solver Engine, and Gurobi Solver Engine. While the difference in cost may be greater, the same rationale applies: If you can solve your model more quickly or more reliably by upgrading the software, this is almost always cheaper (and yields results sooner) than spending many hours or days of valuable professional time.

Ensure that You Have Enough Memory

If the Solver seems unusually slow, check whether the hard disk activity LED (present on most PCs) is flickering during the solution process. If it is, memory demands may be causing Windows to swap data between main memory and disk, which greatly slows down the Solver. If you're investing money and, especially, hours of your time to develop an optimization model, consider that RAM is very cheap, and relatively easy to install. We recommend *at least* 512MB RAM if you are working with large Solver models – 1 GB or more is certainly desirable.

Analyze Your Model for Scaling Problems

Poorly scaled calculations are a frequent cause of long solution times and unreliable solution results, for both linear and nonlinear problems. For a further discussion, see "Problems with Poorly Scaled Models" in the chapter "Diagnosing Solver Results." In Premium Solver Platform for Mac, use the Scaling Report to automatically diagnose scaling problems, as described in the chapter "Solver Reports."

Add Constraints to Your Model

Frequently, you can improve solution time by adding constraints to your model which may not be *essential* in defining the problem, but which do *further constrain* the search space that the Solver must explore. It's true that the Solver must do more work to handle the additional constraints, but this extra work usually has an excellent payoff if the constraints are "binding" (i.e. satisfied with equality) at some point during the solution process.

The greatest payoff often comes from additional constraints that are simple bounds on the decision variables. This is because (i) it's usually easier for you to determine realistic lower and upper bounds on the variables than to formulate new general constraints, (ii) it's easy to enter bounds on the variables in the Constraints list box, and (iii) each of the Solver engines is able to handle bounds on the variables more efficiently than general constraints.

Users often omit upper bounds on their decision variables, and sometimes omit lower bounds as well. A first step towards improving performance is to enter the tightest bounds on the variables that you can, without eliminating possible good solutions.

Since bounds on the variables are especially important for the performance of the Evolutionary Solver and for multistart methods for global optimization used with the nonlinear Solver engines, the Options dialogs for these Solver engines include a check box "Require Bounds on Variables," which is *checked* by default. When this box is checked, the Solver will stop with an error message if some variables do not have lower or upper bounds at the time you click Solve.

Improving the Formulation of Your Model

The type of problem you are trying to solve, and the solution method or Solver engine that must be used, has a major impact on solution time:

- Linear programming problems can be solved most quickly.
- Quadratic programming problems take somewhat more time.
- Nonlinear optimization problems take considerably more time.
- Non-smooth problems take by far the greatest amount of time.

This section discusses techniques you can use to replace nonlinear functions, and even non-smooth functions, with equivalent (or nearly equivalent) linear or quadratic functions, or with linear functions and binary integer variables. As explained in the chapter "Solver Models and Optimization," a problem with integer variables can take much longer to solve than a problem without such variables. However, an integer linear problem formulated using the techniques described in this section may still take significantly *less* time to solve than the equivalent nonlinear or non-smooth problem. Moreover, if your problem is integer linear, you can find a *guaranteed* optimal solution, or a solution that is guaranteed to be within at least x% of optimal, whereas with a nonlinear or non-smooth problem you will have no such guarantees.

As a rough guide, non-smooth models with more than 1,000 variables may be difficult or impossible to solve in a reasonable amount of time – but equivalent models formulated with linear functions and binary integer variables can often be solved efficiently with the LP/Quadratic Solver. And with the Large-Scale LP/QP Solver Engine or the Gurobi Solver Engine, you can often solve linear integer problems of 10,000, 100,000 or more variables in a reasonable amount of time.

A caveat: If you currently have a model with many nonlinear or non-smooth functions, and you decide to implement some of these techniques to speed up solution of your model, bear in mind that you can use the LP/Quadratic Solver, Large-Scale LP/QP Solver, or Gurobi solver engine only for models where *all of the problem functions* are linear (except for the objective function, which may be quadratic). If you create a model with a *mix* of nonlinear or non-smooth functions and linear functions using binary integer variables, it may still take a long time to solve.

Techniques Using Linear and Quadratic Functions

Below are three common situations where you might at first expect that a nonlinear function is required to express the desired relationship – but with a simple transformation or approximation, you can use a linear or quadratic function instead.

Ratio Constraints

You may want to express a relationship that seems to require dividing one or more variables by other variables. Suppose that you have a portfolio of 1-month, 3-month and 6-month CDs, with the amounts of each CD in cells C1, D1 and E1, and you wish to limit the average maturity to 3 months. You might write a constraint such as:

(1*C1 + 3*D1 + 6*E1) / (C1 + D1 + E1) <= 3

This constraint left hand side is a nonlinear function of the variables, so you would have to use the GRG Solver to find a solution. However, the same constraint can be rewritten (multiplying both sides by the denominator, then collecting terms) as:

(1*C1 + 3*D1 + 6*E1) <= 3*(C1 + D1 + E1), i.e. -2*C1 + 3*E1 <= 0

This constraint is a linear function of the variables, so you would be able to use the much faster Simplex or LP/Quadratic Solver to find a solution. (This transformation above relies on the fact that C1 + D1 + E1 >= 0.)

Mini-Max and Maxi-Min

You may want to minimize the maximum of a group of cells such as C1:C5 (or maximize the minimum of a group of cells). It is tempting to use an objective function such as MAX(C1:C5)– but as explained in the chapter "Solver Models and Optimization," MAX (and MIN) are non-smooth functions, so you'd need to use at least the GRG Solver, and perhaps the Evolutionary Solver to find a solution. Instead, you can introduce another variable D1, make D1 the objective to be minimized, and add the constraint:

C1:C5 <= D1

The effect of this constraint is to make D1 equal to the maximum of C1:C5 at the optimal solution. And if the rest of your model is linear, you can use the much faster Simplex or LP/Quadratic Solver to find a solution.

Quadratic Approximations

If you cannot represent the entire problem using linear functions of the variables, try to formulate it as a quadratic (QP) or quadratically constrained (QCP) problem, with a quadratic objective and/or constraints. You may be able to use a local, quadratic approximation to a smooth nonlinear function *f* near a point *a*:

$$f(x) \cong f(a) + f'(a)(x - a) + \tfrac{1}{2} f''(a)(x - a)^2$$

where $f'(a)$ denotes the first derivative, and $f''(a)$ denotes the second derivative of the function *f* at the point *a*. Several Solver engines offer excellent performance on QP problems, and the SOCP Barrier Solver and the MOSEK Solver Engine offer good to excellent performance on QCP problems.

Even if you cannot eliminate nonlinear functions from your problem altogether, you can improve solution time by making an effort to ensure that as many variables as possible occur linearly in the objective and all of the constraints. You can select the "Recognize Linear Variables" check box in the GRG Solver Options dialog to save time during the solution process. And the Large-Scale GRG engine also recognizes both *linearly occurring variables* and *linear constraints* automatically, for still faster solutions.

You can use the Solver Model dialog in Premium Solver Platform for Mac to easily determine the number of linear variables, functions, and occurrences of variables in functions, as described in the chapter "Analyzing and Solving Models."

Techniques Using Linear Functions and Binary Integer Variables

Below are three common situations where you might at first expect that a non-smooth function such as IF is required to express the desired relationship – but you can instead use a binary integer variable and one or two linear functions to define an equivalent relationship. The techniques described here are similar to those used when Premium Solver Platform for Mac *automatically transforms* your model (see the chapter "Analyzing and Solving Models"), but you can apply these techniques yourself to handle situations where the automatic transformation is not available.

Fixed-Charge Constraints

You may have a quantity *x* in your model that must "jump" from zero to some (fixed or variable) non-zero value, under certain conditions. For example, a machine on a production line may have a fixed setup time or cost if it is used at all, plus a time or cost per unit produced. You can avoid creating a non-smooth function for *x* by introducing a binary integer variable *y* (which is 1 if x is used and 0 if it isn't), and adding a constraint $x <= My$, where *M* is a constant that is larger than any possible value for *x*.

For example, suppose you have a machine that has a setup time of 10 minutes, but once set up will process a widget every 30 seconds. Let cell C1 hold the number of widgets you are producing on this machine, and use cell E1 for a binary integer variable *y* that is 1 if you produce *any* widgets on this machine. Then the total production time can be computed as =0.5*C1+10*E1. Assuming that C1 can be at most 10,000, let M1 = 10000 and add a constraint:

C1 <= M1*E1 (or C1 – M1*E1 <= 0)

If variable C1 is nonnegative (C1 >= 0) and variable E1 is binary integer (E1 = binary), then C1 is forced to be 0 whenever E1 is 0, or equivalently E1 is forced to be

1 whenever C1 is greater than 0. Since the production time calculation and the constraint are both linear functions, you can solve the problem with the Simplex (or LP/Quadratic) Solver and the Branch & Bound method. This is called a *fixed-charge* constraint.

Either-Or Constraints

Constraints in an optimization problem are implicitly connected by the logical operator AND – all of them must be satisfied. Sometimes, however, your model may call for either one constraint (say $f(x) <= F$) or another constraint (say $g(x) <= G$) to be satisfied. You might consider using the OR function in Excel, but as noted in the chapter "Solver Models and Optimization," this function is non-smooth. Instead, you can introduce a binary integer variable y and a constant M, where M is greater than any possible value for $f(x)$ or $g(x)$, and add the constraints $f(x) – F <= My$ and $g(x) – G <= M(1-y)$. Now, when y=0, $g(x)$ is unrestricted and $f(x) <= F$; but when y=1, $f(x)$ is unrestricted and $g(x) <= G$.

For example, imagine you want to allocate your purchases among several suppliers in different geographic regions, each of whom has imposed certain conditions on their price bids. Suppose that one supplier's bid requires that you either purchase at least 400 units from their Chicago warehouse or else purchase at least 600 units from their Phoenix warehouse, in order to obtain their most favorable pricing. Let cell C1 hold the number of units you would purchase from Chicago, and cell D1 hold the number of units you would purchase from Phoenix. Assume that cell M1 contains 10,000 which is more than the maximum number of units you intend to purchase. You can model the supplier's either-or requirement with a binary integer variable in cell E1 and the following constraints:

400 – C1 <= M1*E1
600 – D1 <= M1*(1-E1)

Notice that we have reversed the sense of the constraint left hand sides to reflect the "at least" (>=) requirement. If E1=0, then C1 (units purchased from Chicago) must be at least 400, and the second constraint has no effect. If E1=1, then D1 (units purchased from Phoenix) must be at least 600, and the first constraint has no effect.

IF Functions

In the chapter "Solver Models and Optimization," we used =IF(C1>10,D1,2*D1), where C1 depends on the decision variables, as an example of a non-smooth function: Its value "jumps" from D1 to 2*D1 at C1=10. If you use this IF function directly in your model, you'll either have to try the Transformation tab in the Solver Model dialog, or else solve the model with the Evolutionary Solver. Instead, you can avoid the IF function and solve the problem with the nonlinear GRG Solver – or even the linear Simplex Solver – by introducing a binary integer variable (say E1) that is 1 if the conditional argument of the IF is TRUE, and 0 otherwise. Add the constraints:

C1 – 10 <= M1*E1
10 – C1 <= M1*(1–E1)

When E1 is 0, the first constraint forces C1 <= 10, and the second constraint has no effect. When E1 is 1, the first constraint has no effect, and the second constraint forces C1 >= 10. (If C1=10 exactly, this formulation allows either E1=0 or E1=1, whichever one yields the better objective.) The value of the IF function can then be calculated as D1*E1 + 2*D1*(1-E1), which simplifies to D1*(2-E1) in this example. If D1 is constant in the problem, this is a linear function; if D1 depends linearly on the variables, it is a quadratic; otherwise, it is a smooth nonlinear function. In all cases, the non-smooth behavior has been eliminated.

Depending on how you use the result of the IF function in the rest of your model, you may be able to take this strategy further. Suppose, for example, that if $f(x) >= F$ then you want to impose the constraint $g(x) <= G$; if $f(x) < F$ then you don't need this constraint. You can then use a binary variable y (cell E1 in the example above), and impose constraints like the pair above plus an additional constraint on $g(x)$:

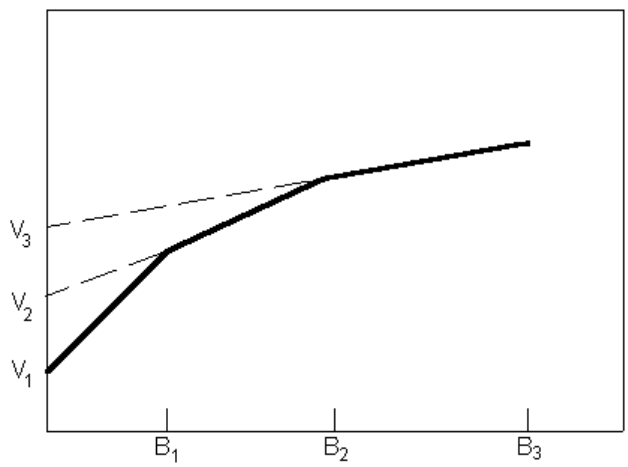
$f(x) - F <= My$
$F - f(x) <= M(1-y)$
$g(x) - G <= M(1-y)$

If y is 0, $f(x) <= F$ is enforced, and the second and third constraints have no effect. If y is 1, $f(x) >= F$ and $g(x) <= G$ are enforced, and the first constraint has no effect. If $f(x)$ and $g(x)$ are linear functions of the variables, the constraints involving y remain linear, and the problem can be solved with Branch & Bound and the Simplex Solver.

Using Piecewise-Linear Functions

Many problems involve "stepped" price schedules or quantity discounts, where you might at first expect that a non-smooth function such as CHOOSE or LOOKUP is required to express the relationship. You might be surprised to learn that you can instead use linear functions and binary integer variables to express the relationship.

For example, you might be purchasing parts from a vendor who offers discounts at various quantity levels. The graph below represents such a discount schedule, with three prices and three "breakpoints." You have a decision variable x representing the quantity to order.



The three prices (slopes of the line segments) are c_1, c_2 and c_3. V_1 represents a fixed initial cost; V_2 and V_3 are also constant in the problem and can be computed from:

$V_2 = V_1 + c_1*B_1 - c_2*B_1$
$V_3 = V_2 + c_2*B_2 - c_3*B_2$

In the model, the variable x is replaced by three variables x_1, x_2 and x_3, representing the quantity ordered or shipped at each possible price. Also included are three 0-1 or binary integer variables y_1, y_2 and y_3. Since you want to minimize costs, the objective and constraints are:

Minimize $V_1*y_1 + V_2*y_2 + V_3*y_3 + c_1*x_1 + c_2*x_2 + c_3*x_3$
Subject to $x_1 \le B_1*y_1$, $x_2 \le B_2*y_2$, $x_3 \le B_3*y_3$

If the cost curve is concave as shown above, this is sufficient; but if the function is non-concave (it may vary up and down), additional "fill constraints" are needed:

$y_1 + y_2 + y_3 \leq 1$
$x_1 \leq B_1 * y_2$
$x_2 \leq B_2 * y_3$

This approach is called a “piecewise-linear” function. It can be used in place of a CHOOSE or LOOKUP function, and it results in a linear integer model instead of a difficult-to-solve non-smooth model. Piecewise-linear functions can also be used to approximate a smooth nonlinear function, by using line segments with slopes matching the gradient of the nonlinear function at various intermediate points.

Diagnosing Solver Results

If You Aren't Getting the Solution You Expect

This chapter will help you understand the results reported by the Solver and 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, quadratic, smooth nonlinear or non-smooth optimization problem – as discussed in previous chapters. If you then build your model in a *well-structured, readable and efficient form* – as outlined at the beginning of the chapter "Building Large-Scale Models" – 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) that 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 Result Message shown in the Solver Results dialog. Users sometimes contact Frontline Systems about "wrong solutions", but they don't know which Solver Result Message they received – this is crucial to diagnosing the problem. Read carefully the discussion of your Solver Result Message in the following sections.**
- **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.**
- **In Premium Solver Platform for Mac, many Solver Result Messages from the Polymorphic Spreadsheet Interpreter refer to a specific problem at a specific cell address in your worksheet. You may have to modify the formula in this cell to use the Interpreter.**
- **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.**
- **If you receive the message "Solver could not find a feasible solution," select and examine the Feasibility Report to determine which subset of the constraints is making the problem infeasible.**

- **If you receive the message "The selected Solver engine can not solve a problem of this type" follow the steps in "Diagnosis Tab: Analyzing Model Exceptions" in the chapter "Analyzing and Solving Models" to pinpoint the exact cell formulas that aren't linear.**
- **Read the section in this chapter on "Problems with Poorly Scaled Models." In Premium Solver Platform for Mac, if you see the Scaling Report listed in the Reports list box of the Solver Results dialog, select this report and examine its contents for strong clues about poor scaling in your model.**
- **Later sections of this chapter discuss characteristics and limitations of the GRG Solver for smooth nonlinear problems and the Evolutionary Solver for non-smooth problems. Read the section(s) most relevant for the type of problem you are solving.**

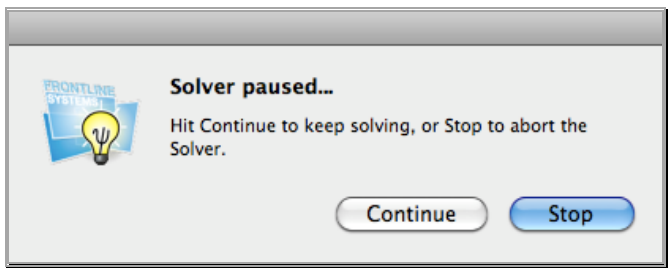
During the Solution Process

When you click the Solve button in the Solver Parameters dialog, the solution process is started. When it completes, the Solver Results dialog appears, displaying a Solver Result Message, and giving you the option to save or discard the results and generate reports.

You can interrupt the solution process at any time by pressing the ESC key. This will display the Solver paused… dialog, pictured below. The Solver paused... dialog also appears if you have checked the box Show Iteration Results in the Solver Options dialog.

Choosing to Continue, Stop or Restart

At the time the Solver paused… dialog appears, your worksheet will contain the current values of the decision variables, objection function and constraints. In this dialog you can choose to *continue*, or *stop* the solution process.

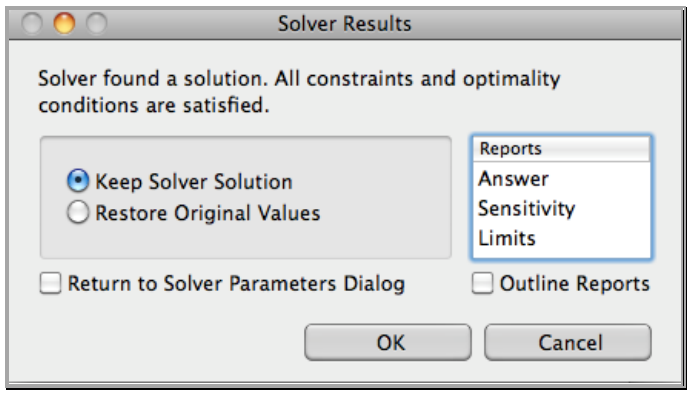


If you choose to *continue* the solution process, the Solver continues to run. If you choose to *stop* the solution process, the Solver "cleans up" and then displays the Solver Results dialog.

When the Solver Finishes

When the solution process completes, the Solver Results dialog appears, displaying a Solver Result Message, as shown below. At this point, your worksheet contains the

best solution found – values for the decision variable cells, and calculated values for the objection function and constraints.



In this dialog (or by calling the SolverFinish function), you can select one or more reports, and choose one of the options "Keep Solver Solution" or "Restore Original Values,". When you click OK, the reports will be produced. Clicking Cancel instead will discard the solution and cancel generation of the reports. The reports are further described in the chapter "Solver Reports."

After the reports (if any) are produced, the Solver will return to worksheet Ready mode unless you've checked the box "Return to Solver Parameters Dialog." When you check the "Return to Solver Parameters Dialog" box, it remains checked (until you change it) for the duration of your Excel session. To return to worksheet Ready mode, you can click the Close button in the Solver Parameters dialog, uncheck this box, or click Cancel in the Solver Results dialog.

Standard Solver Result Messages

The LP/Quadratic Solver, SOCP Barrier Solver, nonlinear GRG Solver, and Evolutionary Solver, and the Branch & Bound and multistart methods bundled with Premium Solver Platform for Mac return the integer result codes and display the Solver Result Messages described in this section. Some of these messages have a slightly different interpretation depending on which Solver engine you are using; see the explanations of each message, particularly for return code 0, "Solver found a solution." Please note that the Branch & Bound and multistart methods usually return result codes 14 through 17, documented later in this section.

Field-installable Solver engines are designed to return the same codes and display the same messages as the built-in Solver engines whenever possible, but they can also return custom result codes (starting with 1000) and display custom messages, as described in their individual documentation.

-1. A licensing problem was detected, or your trial license has expired.

This message appears if a Premium Solver product cannot find its licensing information, if the licensing information is invalid, or if you have a time-limited evaluation license that has expired. *Click the Help button* for further information about the licensing problem. Please call Frontline Systems at (775) 831-0300, or send email to us at info@solver.com for further assistance.

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 or quadratic, smooth nonlinear, global optimization, or integer programming problem, as outlined below. Solvers for non-smooth problems rarely if ever display this message, because they have no way of testing the solution for true optimality.

If you are solving a *linear* programming problem or a *convex quadratic* programming problem with the LP/Quadratic Solver, the Solver has found the *globally* optimal solution: There is *no* other solution satisfying the constraints that has a better value for the objective (Set 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 *linear* (LP)*, convex quadratic* (QP) or *quadratically constrained* (QCP)*, or second order cone programming* (SOCP) problem with the SOCP Barrier Solver, the Solver has found the *globally* optimal solution: There is *no* other solution satisfying the constraints that has a better value for the objective (Set Cell). It's 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 *smooth nonlinear* optimization 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 that yields a better value for the objective (Set 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 and still satisfy the constraints.

If you are solving a *mixed-integer* programming problem (any problem with integer constraints) using a Premium Solver product, this message means that the Branch & Bound method has found a solution satisfying the constraints (including the integer constraints) with the "best possible" objective value (but see the next paragraph). If the problem is linear or quadratic, the true integer optimal solution has been found. If the problem is smooth nonlinear, the Branch & Bound process has found the best of the locally optimal solutions found for sub-problems by the nonlinear Solver.

In the standard Microsoft Excel Solver, this message *also* appears for mixed-integer problems where the Solver stopped because the solution was within the range of the true integer optimal solution allowed by the Tolerance value in the Solver Options dialog (5% by default). In Premium Solver Platform for Mac, when the Branch & Bound process stops due to a nonzero Tolerance without "proving optimality," the message "Solver found an integer solution within tolerance. All constraints are satisfied" (result code 14) is displayed to distinguish this condition (see below).

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

This means that the Solver has found a series of "best" solutions that satisfy the constraints, and that have very similar objective function values; however, no single solution strictly satisfies the Solver's test for optimality. The exact meaning depends on whether you are solving a smooth nonlinear problem with the GRG Solver or a non-smooth problem with the Evolutionary Solver.

When the GRG Solver is being used, this message means that the objective function value is changing very slowly as the Solver progresses from point to point. More precisely, the Solver stops if the absolute value of the relative (i.e. percentage) change in the objective function, in the last few iterations, is less than the Conver-

gence tolerance 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.

When the Evolutionary Solver is being used, this message means that the "fitness" of members of the current population of candidate solutions is changing very slowly. More precisely, the Evolutionary Solver stops if 99% or more of the members of the population have "fitness" values whose relative (i.e. percentage) difference is less than the Convergence tolerance in the Solver Options dialog. The "fitness" values incorporate both the objective function and a penalty for infeasibility, but since the Solver has found some feasible solutions, this test is heavily weighted towards the objective function values. If you believe that the Solver is stopping prematurely when this test is satisfied, you can make the Convergence tolerance smaller, but you may also want to increase the Mutation Rate and/or the Population Size, in order to increase the diversity of the population of trial solutions. For more information, see the discussion of "Evolutionary Solver Stopping Conditions" below.

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

This means that the Solver has found solutions that satisfy the constraints, but it has been unable to further improve the objective, even though the tests for optimality ("Solver found a solution") and convergence ("Solver converged to the current solution") have not yet been satisfied. The exact meaning depends on whether you are solving a smooth nonlinear problem with the GRG Solver or a non-smooth problem with the Evolutionary Solver.

When the GRG Solver is being used, this message occurs very 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.

When the Evolutionary Solver is being used, this message is much more common. It means that the Solver has been unable to find a new, better member of the population whose "fitness" represents a relative (percentage) improvement over the current best member's fitness of more than the Tolerance value on the Limit Options dialog tab, in the amount of time specified by the Max Time without Improvement option in the same dialog. Since the Evolutionary Solver has no way of testing for optimality, it will normally stop with either "Solver converged to the current solution" or "Solver cannot improve the current solution" if you let it run for long enough. If you believe that this message is appearing prematurely, you can either make the Tolerance value smaller (or even zero), or increase the amount of time allowed by the Max Time without Improvement option. For more information, see the discussion of "Evolutionary 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 . You may increase the value in the Iterations box. But you should also consider whether re-scaling your model or adding constraints might reduce the total number of iterations required.

If you are solving a *mixed-integer* programming problem (any problem with integer constraints), this message is relatively unlikely to appear. The Evolutionary Solver uses the Max Subproblems and Max Feasible Solutions options on the Limit Options dialog tab, and the Branch & Bound method (employed by the other Solver engines on problems with integer constraints) uses the Max Subproblems and Max Integer Solutions options on the Integer Options dialog tab, to control the overall solution process. The count of iterations against which the Iteration limit is compared is reset on each new subproblem, so this limit usually is not reached.

4. The Set 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 Set Cell, while still satisfying the constraints. Remember that, if you've selected Minimize, the Set Cell may take on negative values without limit unless this is prevented by the constraints or bounds on the variables. Check the Assume Non-Negative box in the Solver Options dialog to impose >= 0 bounds on all variables.

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 that 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 variable cells, the constraint left hand sides and the objective should provide a strong clue about what happened.

The Evolutionary Solver *never* displays this message, because it has no way of systematically increasing (or decreasing) the objective function, which may be non-smooth. If you have forgotten a constraint, the Evolutionary Solver *may* find solutions with very large (or small) values for the objective – thereby making you aware of the omission – but this is not guaranteed.

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 that allows all of the constraints to be satisfied *simultaneously*. If you are using the LP/Quadratic Solver or the SOCP Barrier Solver, and the model is well scaled, the Solver has determined for *certain* that there is no feasible solution.

If you are using the nonlinear GRG Solver, the GRG method (which always starts from the initial values of the variables) was unable to find a feasible solution; but there could be 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.

If you are using the Evolutionary Solver, the evolutionary algorithm was unable to find a feasible solution; it might succeed in finding one if you run it with different initial values for the variables and/or increase the Precision value in the Solver Options dialog (which reduces the infeasibility penalty, thereby allowing the evolutionary algorithm to explore more "nearly feasible" points).

In any case, you should first look for conflicting constraints, i.e. conditions that *cannot* be satisfied simultaneously. Most often this is due to choosing the wrong

relation (e.g. <= instead of >=) on an otherwise appropriate constraint. The easiest way to do this is to select the Feasibility Report, shown in the Reports list box when this message appears, and click OK. (This report can take time for the LP/Quadratic Solver or SOCP Barrier Solver and *more* time for the GRG Solver; it is not available for the Evolutionary Solver.) For an example of using the Feasibility Report to diagnose an infeasible solution, see "The Feasibility Report" in the chapter "Solver Reports."

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.

7. The selected engine cannot solve a problem of this type.

In the standard Excel Solver, this message is worded "The conditions for Assume Linear Model are not satisfied," and it can appear only when the Assume Linear Model box in the Solver Options dialog is checked.

In Premium Solver Platform for Mac, this message appears, for example, if you've selected the LP/Quadratic Solver and the Solver's tests determine that the constraints are not linear functions of the variables or the objective is not a linear or convex quadratic function of the variables. To understand exactly what is meant by a linear or quadratic function, read the section "Functions of the Variables" in the chapter "Solver Models and Optimization."

Field-installable Solver engines can also display this message. If you've selected the Large-Scale LP/QP Solver, this message appears if the constraints are not linear functions of the variables or the objective is not a linear or convex quadratic function of the variables. If you've selected the MOSEK Solver Engine (Standard Edition), this message appears if the constraints or the objective are not linear or convex quadratic functions of the variables.

If you receive this message, examine the formulas for the objective and constraints for nonlinear or non-smooth functions or operators applied to the decision variables. Simply follow the steps in "Analyzing Model Exceptions" in the chapter "Analyzing and Solving Models" to pinpoint the exact cell formulas that aren't linear.

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

This message – or the more specific message **Too many adjustable cells**, **Too many constraints**, or **Too many integer adjustable cells** – appears when the Solver determines that your model is too large for the Solver engine that is selected (in the Solver engine dropdown list) at the time you click Solve. You'll have to select – or possibly install – another Solver engine appropriate for your problem, or else reduce the number of variables, constraints, or integer variables in order to proceed.

You can check the size (the number of variables, constraints, bounds, and integers) of the problem you have defined, and compare it to the size limits of the Solver engine you are using, by displaying the Problem tab in the Solver Options dialog for that Solver engine. The problem size is also displayed in the Solver Model dialog.

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 (Changing Cells), and discovers an error value such as #VALUE!, #NUM!, #DIV/0! or #NAME? in the cell calculating the objective (Set Cell) or one of the constraints. Inspecting the worksheet for error values like these will usually indicate the source of the problem. If you've 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, it's better to use only constants and cell references on the right hand sides of constraints.

If you see #VALUE!, #N/A or #NAME?, look for names or cell references to rows or columns that you have deleted. If you see #NUM! or #DIV/0!, look 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 $A$1 >= 0, try a constraint such as $A$1 >= 0.0001 instead.

In Premium Solver Platform for Mac, when the Polymorphic Spreadsheet Interpreter is used (Solve With = PSI Interpreter), a more specific message usually appears instead of "Solver encountered an error value in a (nonspecific) target or constraint cell." At a minimum, the message will say "Excel error value returned at cell *address*," where *address* (e.g. Sheet1!$A$1) tells you exactly where the error was encountered. Other messages may tell you more about the error. The general form of the message is:

***Error condition* at cell *address*. Edit your formulas, or use Excel Interpreter in the Solver Model dialog.** *Error condition* is one of the following:

Floating point overflow	Invalid token
Runtime stack overflow	Decision variable with formula
Runtime stack empty	Decision variable defined more than once
String overflow	Missing Diagnostic/Memory evaluation
Division by zero	Unknown function
Unfeasible argument	Unsupported Excel function
Type mismatch	Excel error value returned
Invalid operation	Non-smooth special function

See also result code 21, "Solver encountered an error computing derivatives," and result code 12, with messages that can appear when the Interpreter first analyzes the formulas in your model (when you click the Check Model or Solve button).

"Floating point overflow" indicates that the computed value is too large to represent with computer arithmetic; "String overflow" indicates that a string is too long to be stored in a cell. "Division by zero" would yield #DIV/0! on the worksheet, and "Unfeasible argument" means that an argument is outside the domain of a function, such as =SQRT(A1) where A1 is negative.

"Unknown function" appears for functions whose names are not recognized by the Interpreter, such as user-written functions in VBA. "Unsupported Excel function" appears for the few functions that the Interpreter recognizes but does not support (see the list in the section "More on the Polymorphic Spreadsheet Interpreter" in the chapter "Analyzing and Solving Models"). "Non-smooth special function" appears if your model uses functions ABS, IF, MAX, MIN or SIGN, and the Require Smooth box is checked in the Solver Model dialog (see "Analyzing and Solving Models").

The Evolutionary Solver and the field-installable OptQuest Solver rarely, if ever, display this message – since they maintain a *population* of candidate solutions and can generate more candidates without relying on derivatives, they can simply discard trial solutions that result in error values in the objective or the constraints. If you have a model that frequently yields error values for trial solutions generated by the Solver, and you are unable to correct or avoid these error values by altering your formulas or by imposing additional constraints, you can still use the Evolutionary Solver to find (or make progress towards) a "good" solution.

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. You may increase the value in the Max Time. 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. However, since Microsoft Windows supports a "virtual memory" much larger than your available RAM by swapping data to your hard disk, before you see this message you are likely to notice that solution times have greatly slowed down, and the hard disk activity light in your PC is flickering during the solution process, or even when "Analyzing Solver Model," "Diagnosing Problem Function" or "Setting Up Problem" appears on the Excel status bar.

The Polymorphic Spreadsheet Interpreter in Premium Solver Platform for Mac can use a considerable amount of memory, when you solve a problem by clicking the Solve button, and when you click the Check Model button in the Solver Model dialog. You can progressively reduce the memory used by the Interpreter by taking the following actions in order, using the Solver Model dialog:

1. Check the Sparse box in the Advanced options group.
2. Set the Check For option to Gradients.
3. Set the Solve With option to Excel Interpreter.

When Solve With = Excel Interpreter, the PSI Interpreter is not used and does not use any memory; any further problems are due to memory demands of the Solver engines, Microsoft Excel and Windows. You can save some memory by closing any Windows applications other than Excel, closing programs that run in the System Tray, and closing any Excel workbooks not needed to solve the problem.

12. *Error condition* at cell *address*. Edit your formulas, or use Excel Interpreter in the Solver Model dialog.

This message appears when the Polymorphic Spreadsheet Interpreter first analyzes the formulas in your model after you click the Solve button or the Check Model button in the Solver Model dialog. *Address* is the worksheet address of the cell (in Sheet1!$A$1 form) where the error was encountered, and *Error condition* is one of the following:

OLE error	Missing (
Invalid token	Missing)

Unexpected end of formula	Wrong number of parameters
Invalid array	Type mismatch
Invalid number	Code segment overflow
Invalid fraction	Expression too long
Invalid exponent	Symbol table full
Too many digits	Circular reference
Real constant out of range	External name
Integer constant out of range	Multi-area not supported
Invalid expression	Non-smooth function
Undefined identifier	Unknown function
Range failure	Loss of significance

Many of these messages will never appear as long as you entered your formulas in the normal way through Microsoft Excel, because Excel "validates" your formulas and displays its own error messages as soon as you complete formula entry. Some of the messages you may encounter are described in the following paragraphs.

Undefined identifier appears if you've used a name or identifier (instead of a cell reference such as A1) in a formula, and that name was not defined using the Insert Name Define… or Insert Name Create… menu commands in Excel. If you've used "labels in formulas" and checked the box "Accept labels in formulas" on the Calculation tab of the Tools Options… dialog in Excel, this message will appear. The Interpreter does not support this use of labels in formulas – you'll have to define these labels with the Insert Name Define… or Insert Name Create… commands, or else set Solve With = Excel Interpreter to avoid using the PSI Interpreter.

Circular reference appears if Excel has already warned you about a circular reference in your formulas, and it can also appear if you've used array formulas in a "potentially circular" way. (For example, if cells A1:A2 contain {=1+B1:B4} and cells B3:B4 contain {=1+A1:A4}, Excel doesn't consider this a circular reference, but the PSI Interpreter does.) If you must use circular references in your model, you'll have to set Solve With = Excel Interpreter to avoid using the PSI Interpreter.

External name appears if your formulas use references to cells in other *workbooks* (not just other worksheets), and the Interpreter is unable to open those workbooks. You should ensure that the external workbooks are in the same folder as the Solver workbook, or for better performance, move or copy the worksheets you need into the workbook containing the Solver model.

Multi-area not supported or **Missing)** appears if your formulas or defined names use multiple selections such as (A1:A5,C1:H1). While the Interpreter does accept *argument lists* consisting of single selections, such as =SUM(A1:A5,C1:H1), it does not accept multiple selections for defined names, or for single arguments such as =SUMSQ((A1:A5,C1:H1), (B1:B5,C2:H2)). If you must use such multiple selections, you'll have to set Solve With = Excel Interpreter.

Note: Result code 12 was formerly associated with the message "Another Excel instance is using SOLVER32.DLL. Try again later," which does not occur in the modern versions of Excel and Windows supported by Premium Solver Platform.

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

This message means that the internal "model" (information about the variable cells, Set Cell, constraints, Solver options, etc.) is not in a valid form. An "empty" or incomplete Solver model, perhaps one with no objective in the Set Cell edit box and

no constraints other than bounds on the variables in the Constraints list box, can cause this message to appear. You might also receive this message if you are using the *wrong version* of either Solver.xla or Solver32.dll, or if you've modified the values of certain hidden defined names used by the Solver, either interactively or in a VBA program. ***To guard against this possibility, you should avoid using any defined names beginning with "solver" in your own application.***

14. Solver found an integer solution within tolerance. All constraints are satisfied.

If you are solving a mixed-integer programming problem (any problem with integer constraints) using one of Premium Solver Platform for Mac, with a *non-zero value* for the integer Tolerance setting on the Integer tab of the Solver Options dialog, the Branch & Bound method has found a solution satisfying the constraints (including the integer constraints) where the relative difference of this solution's objective value from the *true* optimal objective value does not exceed the integer Tolerance setting. (For more information, see "Options for Mixed-Integer Problems" in the chapter "Solver Options.") This may actually *be* the true integer optimal solution; however, the Branch & Bound method did not take the extra time to search all possible remaining sub-problems to "prove optimality" for this solution. If all sub-problems *were* explored (which can happen even with a non-zero Tolerance in some cases), Premium Solver Platform for Mac will produce the message "Solver found a solution. All constraints are satisfied" (result code 0, shown earlier in this section).

15. Stop chosen when the maximum number of feasible [integer] solutions was reached.

If you are using the Evolutionary Solver, this message appears when (i) the Solver has found the maximum number of feasible solutions (values for the variables that satisfy all constraints) allowed by the Max Feasible Sols box on the Limits tab of the Solver Options. You may increase the value in the Max Feasible Sols box.

If you are using one of the other Solver engines on a problem with integer constraints, this message appears when (i) the Solver has found the maximum number of integer solutions (values for the variables that satisfy all constraints, including the integer constraints) allowed by the Max Integer Sols box on the Integer tab of the Solver Options dialog. You may increase the value in the Max Integer Sols box. But you should also consider whether the problem is formulated correctly, and whether you can add constraints to "tighten" the formulation. If you are using the LP/Quadratic Solver in Premium Solver Platform for Mac, try activating more Cuts and Heuristics on the Integer tab of the Solver Options dialog.

16. Stop chosen when the max number of feasible [integer] sub-problems was reached.

If you are using the Evolutionary Solver, this message appears when (i) the Solver has explored the maximum number of sub-problems allowed in the Max Sub-problems box on the Limits tab of the Solver Options dialog. You may increase the value in the Max Sub-problems box.

If you are using one of the other Solver engines on a problem with integer constraints, this message appears when (i) the Solver has explored the maximum number of integer sub-problems (each one is a "regular" Solver problem with additional bounds on the variables) allowed in the Max Sub-problems box on the Integer tab of the Solver Options dialog. You may increase the value in the Max Sub-problems box. But you should also consider whether the problem is formulated correctly, and whether you can add constraints to "tighten" the formulation. If you are using the

LP/Quadratic Solver in Premium Solver Platform for Mac, try activating more Cuts and Heuristics on the Integer tab of the Solver Options dialog.

17. Solver converged in probability to a global solution.

If you are using the multistart methods for global optimization, with either the nonlinear GRG Solver or a field-installable nonlinear Solver engine (by checking the Global Optimization options in the appropriate Solver Options dialog), this message appears when the multistart method's Bayesian test has determined that all of the locally optimal solutions have *probably* been found; the solution displayed on the worksheet is the best of these locally optimal solutions, and is *probably* the globally optimal solution to the problem.

The Bayesian test initially assumes that the number of locally optimal solutions to be found is equally likely to be 1, 2, 3, … etc. up to infinity, and that the relative sizes of the regions containing each locally optimal solution follow a uniform distribution. After each run of the nonlinear GRG Solver or field-installable Solver engine, an updated estimate of the most probable total number of locally optimal solutions is computed, based on the number of subproblems solved and the number of locally optimal solutions found so far. When the number of locally optimal solutions actually found so far is within one unit of the most probable total number of locally optimal solutions, the multistart method stops and displays this message.

18. All variables must have both upper and lower bounds.

If you are using the Evolutionary Solver or the multistart methods for global optimization, and you have checked the box "Require Bounds on Variables" in the Solver Options dialog (it is checked by default), this message will also appear. You should add the missing bounds and try again. Upper bounds must be entered in the Constraints list box. Lower bounds of zero can be applied to all variables by checking the "Assume Non-Negative" box in the Solver Options dialog; non-zero lower bounds must be entered in the Constraints list box. For the Evolutionary Solver or the multistart methods, such bounds are not absolutely required (you can uncheck the box "Require Bounds on Variables"), but they are a practical necessity if you want the Solver to find good solutions in a reasonable amount of time.

19. Variable bounds conflict in binary or alldifferent constraint.

This message appears if you have both a binary or alldifferent constraint on a decision variable and a <= or >= constraint on the same variable (that is inconsistent with the binary or alldifferent specification), or if the same decision variable appears in more than one alldifferent constraint. Binary integer variables always have a lower bound of 0 and an upper bound of 1; variables in an alldifferent group always have a lower bound of 1 and an upper bound of N, where N is the number of variables in the group. You should check that the binary or alldifferent constraint is correct, and ensure that alldifferent constraints apply to non-overlapping groups of variables. If a <= or >= constraint causes the conflict, remove it and try to solve again.

20. Lower and upper bounds on variables allow no feasible solution.

This message appears if you've defined lower and upper bounds on a decision variable, where the lower bound is greater than the upper bound. This (obviously) means there can be no feasible solution, but most Solver engines will detect this

condition before even starting the solution process, and display this message instead of "Solver could not find a feasible solution" to help you more quickly identify the source of the problem. If you have defined your bounds and other constraints in uniform blocks, the lower and upper bounds on a given range of cells will appear consecutively in the Constraints list box (where they are sorted), making it easy to spot the inconsistent bounds.

21. Solver encountered an error computing derivatives

This message appears when the Polymorphic Spreadsheet Interpreter in Premium Solver Platform for Mac is being used (Solve With = PSI Interpreter), and the Interpreter encounters an error when computing derivatives via automatic differentiation. (For more information, see "More on the Polymorphic Spreadsheet Interpreter" in the chapter "Analyzing and Solving Models.") The most common cause of this message is a non-smooth function in your objective or constraints, for which the derivative is undefined. But in general, automatic differentiation is somewhat more strict than finite differencing: As a simple example, =SQRT(A1) evaluated at A1=0 will yield this error message when the Solver is using automatic differentiation (since the derivative of the SQRT function is algebraically undefined at zero), but it won't yield an error when Solve With = Excel Interpreter and the Solver is using finite differencing.

If you receive this message, follow the steps in "Analyzing Model Exceptions" in the chapter "Analyzing and Solving Models" to pinpoint the exact cell formulas that are non-smooth. If you cannot modify your formulas to eliminate the non-smooth functions, you can use a Solver engine, such as the Evolutionary Solver, that doesn't require derivatives.

Problems with Poorly Scaled Models

A *poorly scaled* model is one that computes values of the objective, constraints, or intermediate results that differ by several orders of magnitude. A classic example is a financial model that computes a dollar amount in millions or billions and a return or risk measure in fractions of a percent. Because of the finite precision of computer arithmetic, when these values of very different magnitudes (or others derived from them) are added, subtracted, or compared – in the user's model or in the Solver's own calculations – the result will be accurate to only a few significant digits. After many such steps, the Solver may detect or suffer from "numerical instability."

The effects of poor scaling in a large, complex optimization model can be among the most difficult problems to identify and resolve. It can cause Solver engines to return messages such as "Solver could not find a feasible solution," "Solver could not improve the current solution," or even "The linearity conditions required by this Solver engine are not satisfied," with results that are suboptimal or otherwise very different from your expectations. The effects may not be apparent to you, given the initial values of the variables, but when the Solver explores Trial Solutions with very large or small values for the variables, the effects will be greatly magnified.

Dealing with Poor Scaling

Most Solver engines include a Use Automatic Scaling box in their Solver Options dialogs. When this box is checked, the Solver rescales the values of the objective

and constraint functions internally in order to minimize the effects of poor scaling. But this can only help with the Solver's own calculations – it cannot help with poorly scaled results that arise *in the middle of your Excel formulas*.

The best way to avoid scaling problems is to carefully choose the "units" implicitly used in your model so that all computed results are within a few orders of magnitude of each other. For example, if you express dollar amounts in units of (say) millions, the actual numbers computed on your worksheet may range from perhaps 1 to 1,000.

In Premium Solver Platform for Mac, if you're experiencing results that may be due to poor scaling, you can check your model for scaling problems that arise *in the middle of your Excel formulas* by selecting the Scaling Report when it appears in the Solver Results dialog, and examining the results of this report, as described in the chapter "Solver Reports."

Historical Note on Scaling and Linearity Tests

Poor scaling is an ever-present issue for the Solver, and for almost any kind of mathematical software. Successive versions of the Solver have used increasingly sophisticated methods to deal with poor scaling, culminating in Premium Solver Platform for Mac's tools for analyzing your model for scaling problems.

The Use Automatic Scaling option has been available in the standard Microsoft Excel Solver since Excel 5.0, but in Excel 5.0 and 7.0, this option was effective only for nonlinear problems solved with the GRG Solver. Because of this, the Solver's linearity test (used when the "Assume Linear Model" box was checked) could be "fooled" by an all-linear, but poorly scaled model – yielding the error message "The conditions for Assume Linear Model are not satisfied."

In Excel 97, 2000, XP, 2003, and Excel for Mac 2011 and Premium Solver Platform for Mac, the Use Automatic Scaling option is effective for all types of models, and the Solver also uses a more robust test for linearity. Since no automatic scaling method will work in *all* situations, it is still ***good practice to ensure that the model on your worksheet is well scaled*** – even if you do take advantage of the Use Automatic Scaling option.

The Tolerance Option and Integer Constraints

Users who solve problems with integer constraints using the standard Excel Solver occasionally report that "Solver claims it found an optimal solution, but I manually found an even better solution." What happens in such cases is that the Solver stops with the message "Solver found a solution" because it found a solution *within the range* of the true integer optimal solution *allowed by the Tolerance* option in the Solver Options dialog. In similar cases, Premium Solver Platform for Mac display a message "Solver found an integer solution within tolerance," to avoid confusion.

When you solve a mixed-integer programming problem (any problem with integer constraints) using the Simplex, LP/Quadratic, SOCP Barrier, or GRG Solver, all of which employ the Branch & Bound method, the solution process is governed by the integer Tolerance option on the Solver Options or Integer Options dialog tab. 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 that is better than the one found by the Solver.

The reason that the default setting of the integer 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 integer Tolerance default setting is a compromise value that often saves a great deal of time, and still ensures that a solution found by the Solver is within 5% of the true optimal solution.

To ensure that the Solver finds the true integer optimal solution – possibly at the expense of far more solution time – set the integer Tolerance option to zero. In Premium Solver Platform for Mac, look for the Tolerance edit box on the Integer tab of the Solver Options dialog.

Limitations on Smooth Nonlinear Optimization

As discussed in the chapter "Solver Models and Optimization," nonlinear problems are intrinsically more difficult to solve than linear problems, and there are fewer guarantees about what the Solver can do. If your smooth nonlinear problem is **convex**, the Solver will normally find the *globally optimal* solution (subject to issues of poor scaling and the finite precision of computer arithmetic). But if your problem is **non-convex**, the Solver will normally find only a *locally optimal* solution, close to the starting values of the decision variables, when you click Solve.

As discussed in the chapter "Analyzing and Solving Models," in Premium Solver Platform for Mac you can easily check whether your model is **convex** or **non-convex**, by clicking the Model button, selecting Check For Convexity, and clicking the Check Model button. The result of the convexity test may be *conclusive* (the Solver has proven that the model is convex or non-convex) or *inconclusive* (the Solver was unable to prove either condition). If the test is inconclusive, you are best advised to assume that your model is non-convex, unless you can prove through your own mathematical analysis that it is convex.

When dealing with a **non-convex** 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." In Premium Solver Platform for Mac, you can check the Global Optimization options in the Solver Options dialog for the nonlinear GRG Solver, and use the **multistart** method to *automatically* run the Solver from multiple starting points.

Note that, when the GRG Nonlinear Solver is selected in the dropdown list in the Solver Parameters dialog, the 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 or Barrier method. The GRG method will *usually* find the optimal solution to a linear problem, but occasionally you will receive a Solver Result 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.

GRG Solver Stopping Conditions

It is helpful to understand what the nonlinear GRG Solver can and cannot do, and what each of the possible Solver Result Messages means for this Solver engine. At best, the GRG Solver alone – like virtually all "classical" nonlinear optimization algorithms – can find a *locally optimal* solution to a reasonably *well-scaled*, non-convex 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.

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 that yields a better value for the objective function. Figuratively, this means that the Solver has found a "peak" (if maximizing) or "valley" (if minimizing) – but if the model is non-convex, 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).

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 has 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 edit 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, try using the Evolutionary Solver. To go further with the GRG Solver, you may need specialized consulting assistance.

GRG Solver with Multistart Methods

The multistart methods for global optimization included in Premium Solver Platform for Mac can overcome some of the limitations of the GRG Solver alone, but they are not a panacea. The multistart methods will automatically run the GRG Solver (or a field-installable nonlinear Solver engine) from a number of starting points and will display the best of several locally optimal solutions found, as the probable globally optimal solution. Because the starting points are selected at random and then "clustered" together, they will provide a reasonable degree of "coverage" of the space enclosed by the bounds on the variables. The *tighter the variable bounds* you specify and the longer the Solver runs, the better the coverage.

However, the performance of the multistart methods is generally limited by the performance of the GRG Solver on the subproblems. If the GRG Solver stops prematurely due to slow convergence, or fails to find a feasible point on a given run, the multistart method can improve upon this only by finding another starting point from which the GRG Solver can find a feasible solution, or a better locally optimal solution, by following a different path into the same region.

If the GRG Solver reaches the same locally optimal solution on many different runs initiated by the multistart method, this will tend to decrease the Bayesian estimate of the number of locally optimal solutions in the problem, causing the multistart method to stop relatively quickly. In many cases this indicates that the globally optimal solution has been found – but you should always inspect and think about the solution, and consider whether you should run the GRG Solver manually from starting points selected based on your knowledge of the problem.

GRG Solver and Integer Constraints

Like the multistart methods, the performance of the Branch & Bound method on nonlinear problems with integer constraints is limited by the performance of the GRG Solver on the subproblems. If the GRG Solver stops prematurely due to slow convergence, or fails to find a feasible point on a given run, this may prevent the Branch & Bound method from finding the true integer optimal solution. In most cases, the combination of the Branch & Bound method and the GRG Solver will at least yield a relatively good integer solution. However, if you are unable to find a sufficiently good solution with this combination of methods, consider using one of the field-installable nonlinear Solver engines for Premium Solver Platform for Mac.

Limitations on Global Optimization

With Premium Solver Platform for Mac, you have several choices available for solving global optimization problems: You can use the nonlinear GRG Solver (or a field-installable nonlinear Solver engine) with multistart methods; you can use the Evolutionary Solver to seek global solutions to smooth nonlinear problems, though they are designed primarily for non-smooth problems. Overall, Premium Solver Platform for Mac is very likely the world's best platform for global optimization.

Limitations on Non-Smooth Optimization

As discussed in the chapter "Solver Models and Optimization," non-smooth problems – where the objective and/or constraints are computed with discontinuous or non-smooth Excel functions – are the most difficult types of optimization problems to

solve. There are few, if any, guarantees about what the Solver (or *any* optimization method) can do with these problems.

The most common discontinuous function in Excel is the IF function where the conditional test is dependent on the decision variables. Other common discontinuous functions are CHOOSE, the LOOKUP functions, and COUNT. Common non-smooth functions in Excel are ABS, MIN and MAX, INT and ROUND, and CEILING and FLOOR. Functions such as SUMIF and the database functions are discontinuous if the criterion or conditional argument depends on the decision variables.

If your optimization problem contains discontinuous or non-smooth functions, your simplest course of action is to use the Evolutionary Solver to find a "good" solution. You should read the section "Evolutionary Solver Stopping Conditions" below and the discussion earlier in this chapter of specific Solver Result Messages, to ensure that you understand what the various messages say about your model. You can try using the nonlinear GRG Solver, or even the linear Simplex Solver, on problems of this type, but you should be aware of the effects of non-smooth functions on these Solver engines, which are summarized below.

You *can* use discontinuous functions such as IF and CHOOSE in calculations on the worksheet that 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 make the overall Solver model non-smooth. Users sometimes fail to realize that certain functions, such as ABS and ROUND, are non-smooth. For more information on this subject, read the section "Non-Smooth Functions" in the chapter "Solver Models and Optimization."

Effect on the GRG and Simplex Solvers

A smooth nonlinear solver, such as the GRG Solver, relies on derivative or gradient information to guide it towards a feasible and optimal solution. Since it is unable to compute the gradient of a function at points where the function is discontinuous, or to compute curvature information at points where the function is non-smooth, it cannot guarantee that any solution it finds to such a problem is truly optimal. In practice, the GRG Solver can sometimes deal with discontinuous or non-smooth functions that are "incidental" to the problem, but as a general statement, this Solver engine requires smooth nonlinear functions for the objective and constraints.

If you are using Premium Solver Platform for Mac with default settings, the Interpreter will compute derivatives of the problem functions using *automatic differentiation*. (For further information, see "More on the Polymorphic Spreadsheet Interpreter" in the chapter "Analyzing and Solving Models.") If you try to solve a problem with non-smooth or discontinuous functions (other than the 'special functions' ABS, IF, MAX, MIN or SIGN) using the GRG Solver, you'll likely receive the message "Solver encountered an error computing derivatives." If you check the Require Smooth box in the Solver Model dialog, you'll also receive this message for models that use the 'special functions.' You can set the Solve With option to Excel Interpreter and solve your model – but only with the caveats noted above.

If you try to solve a problem with non-smooth or discontinuous functions with the linear Simplex Solver (using Solve With = Excel Interpreter in Premium Solver Platform), 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 *are* linear over the range considered by the linearity test – but there are no guarantees at all that the solution found is optimal!)

Evolutionary Solver Stopping Conditions

It is helpful to understand what the Evolutionary Solver can and cannot do, and what each of the possible Solver Result Messages means for this Solver engine. At best, the Evolutionary Solver – like other genetic or evolutionary algorithms – will be able to find a *good* solution to a reasonably *well-scaled* model. Because the Evolutionary Solver does not rely on derivative or gradient information, it cannot determine whether a given solution is optimal – so it never really *knows* when to stop. Instead, the Evolutionary Solver stops and returns a solution either when certain heuristic rules (discussed below) indicate that further progress is unlikely, or else when it exceeds a limit on computing time or effort that you've set.

"Good" Versus Optimal Solutions

The Evolutionary Solver makes almost no assumptions about the mathematical properties (such as continuity, smoothness or convexity) of the objective and the constraints. Because of this, it *actually has no concept of an "optimal solution,"* or any way to test whether a solution is optimal. The Evolutionary Solver knows only that a solution is "better" in comparison to other solutions found earlier. It may sometimes find the true optimal solution, on models with a limited number of variables and constraints; on such models, the heuristic stopping rules discussed below may cause the Solver to stop at an appropriate time and report this solution. But the Evolutionary Solver will not be able to *tell you* that this solution is optimal.

When you use the Evolutionary Solver, you may find – like other users of genetic and evolutionary algorithms – that you spend a lot of time running and re-running the Solver, trying to find better solutions. This is an inescapable consequence of using a Solver engine that makes few or no assumptions about the nature of the problem functions. You can never be sure whether you've found the best solution, or what the payoff might be of running the evolutionary algorithm for a longer time. When the Evolutionary Solver stops, you may very well find that, if you keep the resulting solution and restart the Evolutionary Solver, it will find an even better solution. You may also find that starting the GRG Solver from the point where the Evolutionary Solver stops will yield a better (sometimes *much* better) solution.

When Solver has Converged to the Current Solution

This message means that the "fitness" of members of the current population of trial solutions is changing very slowly. More precisely, the Evolutionary Solver stops if 99% or more of the members of the population have "fitness" values whose relative (i.e. percentage) difference is less than the Convergence tolerance in the Solver Options dialog. This condition may mean that the Solver has found a globally optimal solution – if so, new members of the population (that replace other, less fit members) will tend to "crowd around" this solution. However, it may also mean that the population has lost diversity – a common problem in genetic and evolutionary algorithms – and hence the evolutionary algorithm is unable to generate new and better solutions through mutation or crossover of current population members. In this latter case, it may help to interrupt the Solver with the ESC key and click the Restart button (which replaces the worst half of the population with newly sampled points), or to run the Evolutionary Solver again with a larger Population Size and/or an increased Mutation Rate, which increases the chances of a diverse population.

When Solver Cannot Improve the Current Solution

This message means that the Solver has been unable to find a new, better member of the population whose "fitness" represents a relative (percentage) improvement over

the current best member's fitness of more than the Tolerance value on the Limits tab of the Solver Options dialog, in the amount of time specified by the Max Time without Improvement option in the same dialog. Under this heuristic stopping rule, the Evolutionary Solver will continue searching for better solutions as long as it is making the degree of progress that you have indicated via the Tolerance value; if it is unable to make that much progress in the time you've specified, the Solver will stop and report the best solution found.

Evaluating a Solution Found by the Evolutionary Solver

Once you have a solution from the Evolutionary Solver, what can you do with it? Here are some ideas:

1. Keep the resulting solution, restart the Evolutionary Solver from that solution, and see if it is able to find an even better solution in a reasonable length of time.
2. Tighten the Convergence and Tolerance values, increase the Max Subproblems and Max Feasible Sols values, and restart the Evolutionary Solver. This will take more time, but will allow the Solver to explore more possibilities.
3. Increase the Population Size and/or the Mutation Rate, and restart the Evolutionary Solver. This will also take more time, but will tend to increase the diversity of the population and the portion of the search space that is explored.
4. Keep the resulting solution, switch to the GRG Solver and start it from that solution, and see if it finds the same or a better solution. If the GRG Solver displays the message "Solver found a solution," you may have found at least a *locally optimal* point (but remember that this test depends on smoothness of the problem functions).
5. Select and examine the Population Report. If the Best Values are similar from run to run of the Evolutionary Solver, and if the Standard Deviations are small, this may be reason for confidence that your solution is close to the global optimum. Since optimization tends to drive the variable values to extremes, if the solution is feasible and the Best Values are close to the Maximum or Minimum Values listed in the Population Report, this may indicate that you have found an optimal solution.

As you work with the Evolutionary Solver, you will appreciate its ability to find "good" solutions to previously intractable optimization problems, but you will also come to appreciate its limitations. The Evolutionary Solver allows you to spend less time analyzing the mathematical properties of your model, and still obtain "good" solutions – but as we suggested in the Introduction, it is not a panacea.

If your problem is large, or if the payoff from a true optimal solution is significant, you may want to invest more effort to formulate a model that satisfies the requirements of a smooth nonlinear optimization problem, or even an integer linear problem. The chapter "Building Large-Scale Models" describes many techniques you can use to replace non-smooth functions with smooth nonlinear or integer linear expressions. With enough work, you may be able to obtain a significantly better solution with the other Solver engines, and to know with some certainty whether or not you have found the optimal solution.

Solver Options

This chapter describes the options available in the Solver Options dialog for the standard Microsoft Excel Solver, and in the Solver Options dialogs for each of the bundled Solver engines in Premium Solver Platform for Mac. It also briefly describes how these options may be examined or set programmatically.

In Premium Solver Platform for Mac, options may be examined or set interactively via the Solver Options dialogs shown in this chapter, or programmatically using the **VBA functions** described in the later chapter "Using VBA Functions."

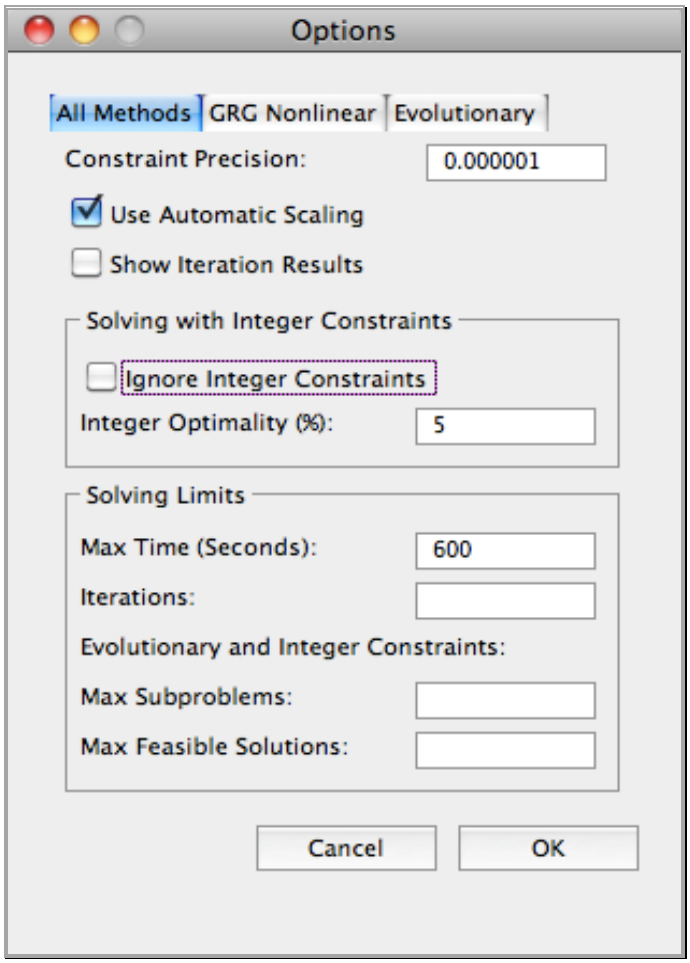
Bear in mind that the options that control numerical tolerances and solution strategies are pre-set to the choices that are most appropriate for the majority of problems; you should change these settings only when necessary, after carefully reading this chapter. The options you will use most often are 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.

A special option in the Premium Solver Platform for Mac, is the option "Ask permission to Save Workbook before Solving". Before the Premium Solver Platform can solve the model on the worksheet, or check the model, it needs to save the workbook in xlsm format. When this checkbox is not checked, the Solver will silently do this, which means your workbook will be save **and you can not undo any previous changes** to this workbook.

When this option is checked, the Solver will always ask permission to save the workbook, before doing so.

The Standard Microsoft Excel Solver

There is just one Solver Options dialog displayed by the standard Microsoft Excel Solver, containing options for the linear Simplex Solver, the nonlinear GRG Solver, and the Evolutionary Solver engines. All of the options in this are also present in the options dialogs for Premium Solver Platform for Mac.



We will first discuss the options common to all Solver engines. Next, we'll describe the additional options specific to the LP/Quadratic Solver, the SOCP Barrier Solver, the nonlinear GRG Solver (including the multistart methods), and the Evolutionary Solver in the Premium Solver Platform. Finally, we'll discuss loading, saving, and merging Solver models.

Common Solver Options

Precision

VBA / SDK: Parameter Name "Precision", 0 < value < 1

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" in the chapter "Solver Models and Optimization" 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 constraint such as A1 >= 0.

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 Microsoft Excel 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 very large (say in millions or billions of dollars), consider adjusting your formulas and input data to work in *units of millions,* or checking the Use Automatic Scaling box instead of altering the Precision setting. Generally, this setting should be kept in the range from 1.0E-6 (0.000001) to 1.0E-4 (0.0001).

Precision and Integer Constraints

Another use of Precision is determining whether an integer constraint, such as A1:A5 = integer, A1:A5 = binary or A1:A5 = alldifferent, is satisfied. If the difference between the decision variable's value and the closest integer value is less than the Precision setting, the variable value is treated as an integer.

Use Automatic Scaling

VBA / SDK: Parameter Name "Scaling", value 1/True or 0/False

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 that computes values of the objective, constraints, or intermediate results that differ by several orders of magnitude. Poorly scaled models may cause difficulty for both linear and nonlinear solution algorithms, due to the effects of finite precision computer arithmetic. For more information, see "Problems with Poorly Scaled Models" in the chapter "Diagnosing Solver Results," and "The Scaling Report" in the chapter "Solver Reports."

Note: In older versions of Microsoft Excel (prior to Excel 97), *this option is effective only for nonlinear optimization problems* solved with the GRG Solver.

If your model is nonlinear and you 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.

Show Iteration Results

VBA: Parameter Name "StepThru", value 1/True or 0/False
SDK: Define an Evaluator for Eval_Type_Iteration

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

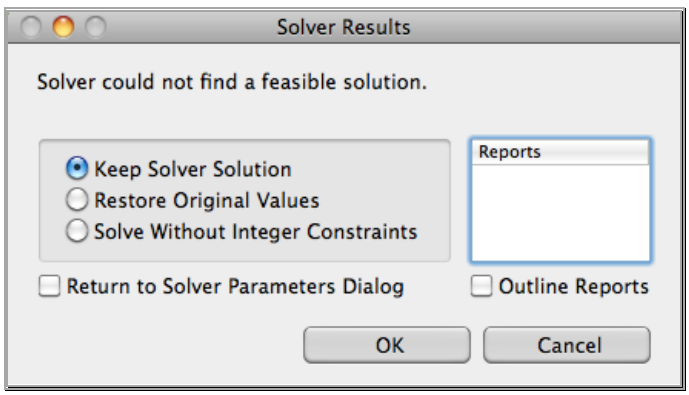
This is the same dialog that appears when you press ESC at any time during the solution process, but when the Show Iteration Results box is checked it appears automatically on every iteration. When this dialog appears, the best values so far for the decision variables appear on the worksheet, which is recalculated to show the values of the objective function and the constraints. You may click the Continue button to go on with the solution process, the Stop button to stop immediately. For more information on this dialog, see the section "During the Solution Process" in the chapter "Diagnosing Solver Results."

Ignore Integer Constraints

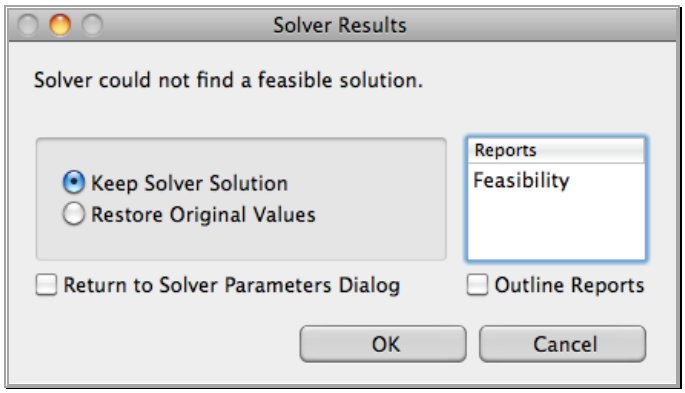
VBA / SDK: Parameter Name "SolveWithout", value 1/True or 0/False

When you click the Solve button (in the Solver Parameters dialog) while this box is checked, the Solver *ignores* integer constraints (including alldifferent constraints) and solves the "relaxation" of the problem. It is often useful to solve the relaxation, and it's much more convenient to check this box than to delete the integer constraints and add them back again later.

This option remains in effect until you uncheck the Solve Without Integer Constraints box. When you solve an integer programming problem (without this option) and the Solver finds no feasible integer solution, you are offered the option of solving the relaxation on a "one-time-only" basis in the Solver Results dialog, as shown below.



When chosen this way, the option is effective for only one solution attempt, when you click OK. It is possible that the relaxation of the problem – ignoring the integer constraints – is still infeasible; in this case, you will next see the Solver Results dialog shown on the following page, which will allow you to produce a Feasibility Report for the relaxation of the problem. Using the Feasibility Report, you can more easily locate and correct the conflicting constraints that make the problem infeasible.



Integer Tolerance

VBA / SDK: Parameter Name "IntTolerance", 0 <= value <= 1

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 computing 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. In Solver Platform, the Solver Result Message will be "Solver found an integer solution within tolerance." 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.

Make Unconstrained Variables Non-Negative

VBA: Parameter Name "AssumeNonneg", value 1/True or 0/False
SDK: Use Variable object NonNegative method or SolverVarNonNegative function

When this box is checked, any decision variables that are not given explicit lower bounds via >=, binary, or alldifferent 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 that *do* have explicit >= constraints, even if those constraints allow the variables to assume negative values.

Max Time and Iterations

VBA / SDK: Parameter Names "MaxTime", "Iterations", integer value >= 0

The value in the Max Time edit box determines the maximum time in seconds that the Solver will run before it stops, including problem setup time and time to find the optimal solution. For problems with integer constraints, this is the total time taken to solve all subproblems explored by the Branch & Bound method. The default value is 0 seconds, which means no limit is set.

The value in the Iterations edit box determines the maximum number of iterations ("pivots" for the Simplex Solver, or major iterations for the GRG Solver) that the Solver may perform on one problem. A new "Trial Solution" is generated on each iteration; the most recent Trial Solution is reported on the Excel status bar. For problems with integer constraints, the Iterations setting determines the maximum number of iterations for any *one* subproblem. The default value is 0 iterations, which means no limit is set.

Max Subproblems

VBA / SDK: Parameter Name "MaxSubProblems", integer value >= 0

The value in the Max Subproblems edit box places a limit on the number of subproblems that may be explored by the Branch & Bound algorithm before the Solver pauses and asks you whether to continue or stop the solution process. Each subproblem is a "regular" Solver problem with additional bounds on the variables.

In a problem with integer constraints, this limit should be used in preference to the Iterations limit in the Solver Options dialog; the Iterations limit should be set high enough for each of the individual subproblems solved during the Branch & Bound process.

Max Feasible (Integer) Solutions

VBA / SDK: Parameter Name "MaxIntegerSols", integer value >= 0

The value in the Max Feasible Sols edit box places a limit on the number of feasible integer solutions found by the Branch & Bound algorithm before the Solver pauses and asks you whether to continue or stop the solution process. Each feasible integer solution satisfies all of the constraints, including the integer constraints; the Solver retains the integer solution with the best objective value so far, called the "incumbent."

It is entirely possible that, in the process of exploring various subproblems with different bounds on the variables, the Branch & Bound algorithm may find the same feasible integer solution (set of values for the decision variables) more than once; the Max Feasible Solutions limit applies to the total number of integer solutions found, not the number of "distinct" integer solutions.

Convergence

VBA / SDK: Parameter Name "Convergence", 0 <= value <= 1

As discussed in the chapter "Diagnosing Solver Results," the GRG Solver will stop and display the message "Solver has 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 1.0E-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 Trial Solutions until the optimality (KKT) 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 1.0E-5 or 1.0E-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.

Derivatives

VBA / SDK: Parameter Name "Derivatives", value 1-Forward or 2-Central

On each major iteration, the GRG Solver requires values for the gradients of the objective and constraints (i.e. the Jacobian matrix). In Premium Solver Platform for Mac, these derivatives may be computed via *automatic differentiation* or via *finite differencing*. For more information, see the section "More on the Polymorphic Spreadsheet Interpreter" in the chapter "Analyzing and Solving Models."

In Premium Solver Platform for Mac, when you are using the Interpreter (Solve With = PSI Interpreter), automatic differentiation is used, highly accurate derivative values are computed, and the Derivatives setting is ignored. In Premium Solver Platform for Mac when Solve With = Excel Interpreter, the method used for finite differencing is determined by the Derivatives setting.

Forward differencing (the default choice) uses the point from the previous iteration – where the problem function values are already known – in conjunction with the current point. *Central* differencing relies only on the current point, and perturbs the decision variables in opposite directions from that point. This requires up to twice as much time *on each iteration*, but it may result in a better choice of search direction when the derivatives are rapidly changing, and hence fewer total iterations. (Bear in mind that automatic differentiation is much faster than either Forward or Central differencing.)

Multistart Search

VBA / SDK: Parameter Name "MultiStart", value 1/True or 0/False

If this box is checked, the multistart methods are used to seek a globally optimal solution. If this box is unchecked, the other options described in this section are ignored. The multistart methods will generate candidate starting points for the GRG Solver (with randomly selected values between the bounds you specify for the variables), group them into "clusters" using a method called multi-level single linkage, and then run the GRG Solver from a representative point in each cluster. This process continues with successively smaller clusters that are increasingly likely to capture each possible locally optimal solution.

Population Size

VBA / SDK: Parameter Name "PopulationSize", integer value >= 0

The multistart methods generate a number of candidate starting points for the GRG Solver equal to the value that you enter in this box. This set of starting points is referred to as a "population," because it plays a role somewhat similar to the population of candidate solutions maintained by the Evolutionary Solver. The minimum population size is 10 points; if you supply a value less than 10 in this box, or leave it blank, the multistart methods use a population size of 10 times the number of decision variables in the problem, but no more than 200.

Random Seed

VBA / SDK: Parameter Name "RandomSeed", integer value >= 0

The multistart methods use a process of random sampling to generate candidate starting points for the GRG Solver. This process uses a random number generator that is normally "seeded" using the value of the system clock – so the random number sequence (and hence the generated candidate starting points) will be different each time you click Solve. At times, however, you may wish to ensure that the *same* candidate starting points are generated on several successive runs – for example, in order to test different GRG Solver options on each search for a locally optimal solution. To do this, enter an integer value into this box; this value will then be used to "seed" the random number generator each time you click Solve.

Mutation Rate

VBA / SDK: Parameter Name "MutationRate", 0 <= value <= 1

The Mutation Rate is the probability that some member of the population will be mutated to create a new trial solution (which becomes a candidate for inclusion in the population, depending on its fitness) during each "generation" or subproblem considered by the evolutionary algorithm. In the Evolutionary Solver, a subproblem consists of a possible mutation step, a crossover step, an optional local search in the vicinity of a newly discovered "best" solution, and a selection step where a relatively "unfit" member of the population is eliminated.

There are many possible ways to mutate a member of the population, and the Evolutionary Solver actually employs five different mutation strategies, including "permutation-preserving" mutation strategies for variables that are members of an "alldifferent" group. The Mutation Rate is effectively subdivided between these strategies, so increasing or decreasing the Mutation Rate affects the probability that each of the strategies will be used during a given "generation" or subproblem.

Max Time without Improvement

VBA / SDK: Parameter Name "MaxTimeNoImp", integer value > 0

This option works in conjunction with the Tolerance option to limit the time the evolutionary algorithm spends without making any significant progress. If the relative (i.e. percentage) improvement in the best solution's "fitness" is less than the Tolerance value for the number of seconds in the Max Time without Improvement edit box, the Evolutionary Solver stops and displays the Solver Results dialog. The message is "Solver cannot improve the current solution," unless the evolutionary algorithm has discovered no feasible solutions at all, in which case the message is "Solver could not find a feasible solution." If you believe that this stopping condition is being met prematurely, you can either make the Tolerance value smaller (or even zero), or increase the number of seconds allowed by the Max Time without Improvement option.

LP/Quadratic Solver Options

Primal Tolerance and Dual Tolerance

VBA / SDK: Parameter Names "PrimalTolerance", "DualTolerance", 0 < value < 1

The Primal Tolerance is the maximum amount by which the primal constraints can be violated and still be considered feasible. The Dual Tolerance is the maximum amount by which the dual constraints can be violated and still be considered feasible. The default values of 1.0E-7 for both tolerances are suitable for most problems.

Presolve

VBA / SDK: Parameter Name "Presolve", value 1/True or 0/False

When this box is checked (which is the default setting), the LP/Quadratic Solver performs a Presolve step before applying the Primal or Dual Simplex method. Presolving often reduces the size of an LP problem by detecting singleton rows and columns, removing fixed variables and redundant constraints, and tightening bounds.

Derivatives for the Quadratic Solver

When a quadratic programming (QP) problem – one with a quadratic objective and all linear constraints – is solved with the LP/Quadratic Solver, the quadratic Solver extension requires first or second partial derivatives of the objective function at various points. In Premium Solver Platform for Mac, these derivatives may be computed via *automatic differentiation* or via *finite differencing*. For more information, see the section "More on the Polymorphic Spreadsheet Interpreter" in the chapter "Analyzing and Solving Models."

When you are using the Interpreter (Solve With = PSI Interpreter in the Solver Model dialog), automatic differentiation is used, exact derivative values are computed, and the setting of the Derivatives choice is ignored. When Solve With = Excel Interpreter, the method used for finite differencing is determined by the setting of the Derivatives choice. *Forward* differencing uses the point from the previous iteration – where the problem function values are already known – in conjunction with the current point. *Central* differencing relies only on the current point, and perturbs the decision variables in opposite directions from that point. For QP problems, the Central differencing choice yields essentially exact (rather than approximate) derivative values, which can improve solution accuracy and reduce the total number of iterations; however the initial computation of derivatives may take up to twice as long as with Forward differencing. (Bear in mind that automatic differentiation is much faster than either Forward or Central differencing.)

Integer Cutoff

VBA / SDK: Parameter Name "IntCutoff", -1E30 < value < +1E30

This option provides another way to save time in the solution of mixed-integer programming problems. If you know the objective value of a feasible integer solution to your problem – possibly from a previous run of the same or a very similar problem – you can enter this objective value in the Integer Cutoff edit box. This allows the

Branch & Bound process to *start* with an "incumbent" objective value (as discussed above under Integer Tolerance) and avoid the work of solving subproblems whose objective can be no better than this value. If you enter a value here, you must be *sure* that there is an integer solution with an objective value at least this good: A value that is too large (for maximization problems) or too small (for minimization) may cause the Solver to skip solving the subproblem that would yield the optimal integer solution.

Preprocessing, Cuts, Heuristics

Preprocessing, Cuts, and Heuristics – to gain the benefit of these methods. For each of these options, you can select None, Automatic, or Aggressive. Each choice activates specific combinations of methods that Frontline has found effective on many models. As usual with LP/MIP problems, performance is very model-dependent – but overall, you can expect a *significant* speed improvement on your LP/MIP problems.

SOCP Barrier Solver Options

Gap Tolerance

VBA / SDK: Parameter Name "GapTolerance", 0 < value < 1

The SOCP Barrier Solver uses a primal-dual method that computes new objective values for the primal problem and the dual problem at each iteration. When the gap or difference between these two objective values is less than the Gap Tolerance, the SOCP Barrier Solver will stop and declare the current solution optimal.

Step Size Factor

VBA / SDK: Parameter Name "StepSizeFactor", 0 < value < .99

This parameter is the relative size (between 0 and 1) of the step that the SOCP Barrier Solver may take towards the constraint boundary at each iteration.

Feasibility Tolerance

VBA / SDK: Parameter Name "FeasibilityTolerance", 0 < value < 1

The SOCP Barrier Solver considers a solution feasible if the constraints are satisfied to within this tolerance.

Search Direction

VBA / SDK: Parameter Name "SearchDirection", value 1-Power Class, 2-Power Class with Predictor-Corrector, 3-Dual Scaling, 4- Dual Scaling with Predictor-Corrector

The SOCP Barrier Solver offers four options for computing the search direction on each iteration.

Power Class

This option uses the *power class*, which is a subclass of the commutative class of search directions over symmetric cones with the property that the long-step barrier algorithm using this class has polynomial complexity.

Power Class with Predictor-Corrector

This option uses the *power class* as described above, plus a predictor-corrector term.

Dual Scaling

This option uses HKM (Helmberg, Kojima and Monteiro) dual scaling, a Newton direction found from the linearization of a symmetrized version of the optimality conditions.

Dual Scaling with Predictor-Corrector

This option uses HKM dual scaling, plus a predictor-corrector term.

Power Index

VBA / SDK: Parameter Name "PowerIndex", integer value >= 0

This parameter is used to select a specific search direction when the Search Direction is computed via the Power Class or Power Class with Predictor-Corrector methods.

GRG Nonlinear Solver Options

Search and Estimates

The default values for the Estimates, Derivatives and Search options can be used for most problems. If you'd like to change these options to improve performance on your model, this section will provide some general background on how they are used by the GRG Solver. For more information, consult the academic papers on the GRG method listed at the end of the Introduction.

On each major iteration, the GRG Solver requires values for the gradients of the objective and constraints (i.e. the Jacobian matrix). The Derivatives option is concerned with how these partial derivatives are computed.

The GRG (Generalized Reduced Gradient) solution algorithm 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 that are being varied have a signifi-

cant impact on the effectiveness of the search. The Estimates option is concerned with how these estimates are obtained.

Estimates

VBA / SDK: Parameter Name "Estimates", value 1-Tangent or 2-Quadratic

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.

Search

VBA / SDK: Parameter Name "SearchOption", value 1-Newton or 2-Conjugate

It would be 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. Instead, a 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.

Recognize Linear Variables

VBA / SDK: Parameter Name "RecognizeLinear", value 1/True or 0/False

This check box activates an "aggressive" strategy to speed the solution of nonlinear problems that may be useful in Premium Solver Platform for Mc when the Polymorphic Spreadsheet Interpreter is not used (Solve With = Excel Interpreter). As explained in the chapter "Solver Models and Optimization," a Solver problem is nonlinear (and must be solved with the GRG Solver engine) if the objective or any of the constraints is a nonlinear function of even one decision variable. But in many such problems, some of the variables occur linearly in the objective and all of the constraints. Hence the partial derivatives of the problem functions with respect to these variables are constant, and need not be re-computed on each iteration.

If you check the Recognize Linear Variables box, the GRG Solver will look for variables whose partial derivatives are not changing over several iterations, and then will *assume* that these variables occur linearly, hence that their partial derivatives remain constant. At the solution, the partial derivatives are recomputed and compared to the assumed constant values; if any of these values has changed, the Solver will display the message "The linearity conditions required by this Solver engine are not satisfied." If you receive this message, you should uncheck the Recognize Linear Variables box and re-solve the problem.

In Premium Solver Platform for Mac, when the Interpreter is used (which is the default), checking this box will not save any time, because partial derivatives are computed via *automatic differentiation* rather than *finite differencing*. The field-installable Large-Scale GRG Solver, Large-Scale SQP Solver, and KNITRO Solver engines are designed to take advantage of information provided by the Interpreter, and will exploit partial linearity in the problem functions much more effectively than the GRG Solver with the Recognize Linear Variables option.

Topographic Search

VBA / SDK: Parameter Name "TopoSearch", value 1/True or 0/False

If this box (and the Multistart Search box) are checked, the multistart methods will make use of a "topographic" search method. This method uses the objective value computed for the randomly sampled starting points to determine a "topography" of overall "hills" and "valleys" in the search space, in an effort to find better clusters and start the GRG Solver from an improved point (already in a "hill" or "valley") in each cluster. Determining the topography takes extra computing time, but on some problems this is more than offset by reduced time taken by the GRG Solver on each subproblem.

Evolutionary Solver Options

Convergence

VBA / SDK: Parameter Name "Convergence", 0 <= value <= 1

As discussed in the chapter "Diagnosing Solver Results," the Evolutionary Solver will stop and display the message "Solver has converged to the current solution" if nearly all members of the current population of solutions have very similar "fitness" values. Since the population may include members representing infeasible solutions, each "fitness" value is a combination of an objective function value and a penalty for infeasibility. Since the population is initialized with trial solutions that are largely chosen at random, the comparison begins after the Solver has found a certain minimum number of improved solutions that were generated by the evolutionary process. The stopping condition is satisfied if 99% of the population members all have fitness values that are within the Convergence tolerance of each other.

If you believe that the message "Solver has converged to the current solution" is appearing prematurely, you can make the Convergence tolerance smaller, but you may also want to increase the Mutation Rate and/or the Population Size, in order to increase the diversity of the population of trial solutions.

Population Size

VBA / SDK: Parameter Name "PopulationSize", integer value > 0

As described in the chapter "Solver Models and Optimization," the Evolutionary Solver maintains a population of candidate solutions, rather than a "single best solution" so far, throughout the solution process. This option sets the number of candidate solutions in the population. The minimum population size is 10 members; if you supply a value less than 10 for this option, or leave the edit box blank, the

Evolutionary Solver uses a population size of 10 times the number of decision variables in the problem, but no more than 200.

The initial population consists of candidate solutions chosen largely at random, but it always includes at least one instance of the starting values of the variables (adjusted if necessary to satisfy the bounds on the variables), and it may include more than one instance of the starting values, especially if the population is large and the initial values represent a feasible solution.

A larger population size may allow for a more complete exploration of the "search space" of possible solutions, especially if the mutation rate is high enough to create diversity in the population. However, experience with genetic and evolutionary algorithms reported in the research literature suggests that a population need not be very large to be effective – many successful applications have used a population of 70 to 100 members.

Random Seed

VBA / SDK: Parameter Name "RandomSeed", integer value > 0

The Evolutionary Solver makes extensive use of random sampling, to generate trial points for the population of candidate solutions, to choose strategies for mutation and crossover on each "generation," and for many other purposes. This process uses a random number generator that is normally "seeded" using the value of the system clock – so the random number sequence (and hence trial points and choices made by the Evolutionary Solver) will be different each time you click Solve. Because of these random choices, the Evolutionary Solver will normally find at least slightly different (and sometimes very different) solutions on each run, even if you haven't changed your model at all. At times, however, you may wish to ensure that exactly the *same* trial points are generated, and the same choices are made on several successive runs. To do this, enter a positive integer value into this box; this value will then be used to "seed" the random number generator each time you click Solve.

Require Bounds on Variables

VBA / SDK: Parameter Name "RequireBounds", value 1/True or 0/False

If the check box "Require Bounds on Variables" is selected, and some of the decision variables do not have upper or lower bounds specified in the Constraints list box (or via the Assume Non-Negative option) at the time you click Solve, the Solver will stop immediately with the message "All variables must have both upper and lower bounds" – as illustrated in the section "Multistart Search Options" earlier in this chapter. If this option is not selected, the Solver will not require upper and lower bounds on the variables, but will attempt to solve the problem without them. Note that this box is *checked by default*.

Bounds on the variables are especially important to the performance of the Evolutionary Solver. For example, the initial population of candidate solutions is created, in part, by selecting values at random from the ranges determined by each variable's lower and upper bounds. Bounds on the variables are also used in the mutation process – where a change is made to a variable value in some member of the existing population – and in several other ways in the Evolutionary Solver. If you do not specify lower and upper bounds for all of the variables in your problem, the Evolutionary Solver can still proceed, but the almost-infinite range for these variables may significantly slow down the solution process, and make it much harder to find

"good" solutions. Hence, it pays for you to determine realistic lower and upper bounds for the variables, and enter them in the Constraints list box.

Local Search

VBA / SDK: Parameter Name "LocalSearch", value 1-Randomized Local Search, 2-Deterministic Pattern Search, 3-Gradient Local Search, 4-Automatic Choice

This option determines the local search strategy employed by the Evolutionary Solver. As noted under the Mutation rate option, a "generation" or subproblem in the Evolutionary Solver consists of a possible mutation step, a crossover step, an optional local search in the vicinity of a newly discovered "best" solution, and a selection step where a relatively "unfit" member of the population is eliminated. You have a choice of strategies for the local search step. In Premium Solver Platform for Mac, you can use Automatic Choice (the default), which selects an appropriate local search strategy automatically based on characteristics of the problem functions.

Randomized Local Search

This local search strategy generates a small number of new trial points in the vicinity of the just-discovered "best" solution, using a probability distribution for each variable whose parameters are a function of the best and worst members of the current population. (If the generated points do not satisfy all of the constraints, a variety of strategies may be employed to transform them into feasible solutions.) Improved points are accepted into the population.

Deterministic Pattern Search

This local search strategy uses a "pattern search" method to seek improved points in the vicinity of the just-discovered "best" solution. The pattern search method is deterministic – it does not make use of random sampling or choices – but it also does not rely on gradient information, so it is effective for non-smooth functions. It uses a "slow progress" test to decide when to halt the local search. An improved point, if found, is accepted into the population.

Gradient Local Search

This local search strategy makes the assumption that the objective function – even if non-smooth – can be approximated locally by a quadratic model. It uses a classical quasi-Newton method to seek improved points, starting from the just-discovered "best" solution and moving in the direction of the gradient of the objective function. It uses a classical optimality test and a "slow progress" test to decide when to halt the local search. An improved point, if found, is accepted into the population.

Automatic Choice

This option allows the Solver to select the local search strategy automatically in Premium Solver Platform for Mac. In Premium Solver Platform for Mac, the Solver uses diagnostic information from the Polymorphic Spreadsheet Interpreter to select a linear Gradient Local Search strategy if the problem has a mix of non-smooth and linear variables, or a nonlinear Gradient Local Search strategy if the objective function has a mix of non-smooth and smooth nonlinear variables. It also makes limited use of the Randomized Local Search strategy to increase diversity of the points found by the local search step.

Fix Nonsmooth Variables

VBA / SDK: Parameter Name "FixNonSmooth", value 1/True or 0/False

In Premium Solver Platform for Mac, this option determines how non-smooth variable occurrences in the problem will be handled during the local search step. If this box is checked, the non-smooth variables are fixed to their current values (determined by genetic algorithm methods) when a nonlinear Local Gradient or linear Local Gradient search is performed; only the smooth and linear variables are allowed to vary. If this box is unchecked, all of the variables are allowed to vary.

Since gradients are undefined for non-smooth variables at certain points, fixing these variables ensures that gradient values used in the local search process will be valid. On the other hand, gradients *are* defined for non-smooth variables at *most* points, and the search methods are often able to proceed in spite of *some* invalid gradient values, so it often makes sense to vary all of the variables during the search. Hence, this box is unchecked by default; you can experiment with its setting on your problem.

The behavior of the Fix Nonsmooth Variables option on a small group of special functions – currently ABS, IF, MAX, MIN and SIGN – is affected by the setting of the Require Smooth check box in the Solver Model dialog, as described in "Using Analyzer Advanced Options" in the chapter "Analyzing and Solving Models." When the Require Smooth box is unchecked (the default), checking the Fix Nonsmooth Variables box will *not* fix variables occurring in these special functions.

Filtered Local Search

In Premium Solver Platform for Mac, the Solver applies two tests or "filters" to determine whether to perform a local search each time a new point generated by the genetic algorithm methods is accepted into the population. The "merit filter" requires that the objective value of the new point be better than a certain threshold if it is to be used as a starting point for a local search; the threshold is based on the best objective value found so far, but is adjusted dynamically as the Solver proceeds. The "distance filter" requires that the new point's distance from any known locally optimal point (found on a previous local search) be greater than the distance traveled when that locally optimal point was found.

Thanks to its genetic algorithm methods, improved local search methods, and the distance and merit filters, the Evolutionary Solver in Premium Solver Platform for Mac performs exceedingly well on smooth global optimization problems, and on many non-smooth problems as well.

The local search methods range from relatively "cheap" to "expensive" in terms of the computing time expended in the local search step; they are listed roughly in order of the computational effort they require. On some problems, the extra computational effort will "pay off" in terms of improved solutions, but in other problems, you will be better off using the "cheap" Randomized Local Search method, thereby spending relatively more time on the "global search" carried out by the Evolutionary Solver's mutation and crossover operations.

In addition to the Local Search options, the Evolutionary Solver employs a set of methods, corresponding to the four local search methods, to transform infeasible solutions – generated through mutation and crossover – into feasible solutions in new regions of the search space. These methods, which also vary from "cheap" to "expensive," are selected dynamically (and automatically) via a set of heuristics. For problems in which a significant number of constraints are smooth nonlinear or even linear, these methods can be highly effective. Dealing with constraints is traditionally a weak point of genetic and evolutionary algorithms, but the hybrid

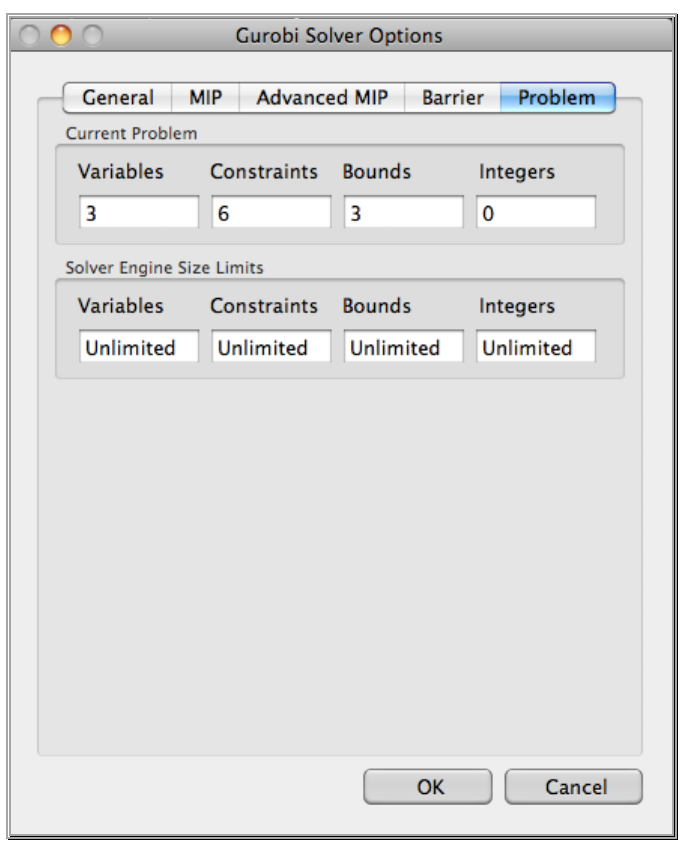
Evolutionary Solver in Premium Solver Platform for Mac is unusually strong in its ability to deal with a combination of constraints and non-smooth functions.

For the reasons described in "Using Analyzer Advanced Options" in the chapter "Analyzing and Solving Models," **if the Evolutionary Solver stops with the message "Solver encountered an error computing derivatives," you should check the Analyzer Sparse box in the Model dialog, and click Solve again.**

The Problem Tab

In Premium Solver Platform, each external Solver Options dialog includes a Problem tab. Clicking on this tab displays statistics on the size of the current problem and the corresponding Solver engine size limits, including the number of decision variables, number of constraints, number of bounds on the variables, and number of integer variables. These edit controls are "read-only" – the current problem sizes are computed automatically, and the Solver engine size limits are obtained automatically from both built-in and field-installable Solver engines.

When the Gurobi Solver is selected from the Solver engine dropdown list, the Options button is clicked, and the Problem tab is clicked, the options shown on the next page will be displayed.



Loading, Saving and Merging Solver Models

The Solver Parameters dialog also includes a Load/Save button, which allows you to save and restore the specifications (variable, constraint and objective cell selections plus option settings) of a Solver model on the worksheet. The model specifications are stored as formulas in cells, which include references to the variable, constraint and objective cells and values for the Solver options.

The "current" Solver model defined for each worksheet is automatically saved "behind the scenes" in that worksheet. So it is not necessary to use this feature to keep track of a single Solver model – the last set of specifications you defined will be saved automatically when the workbook is saved, and restored when it is re-opened. But the Load/Save button can be used to save more than one Solver model on the same worksheet, and to merge two models into one.

Using Multiple Solver Models

It is possible – and often useful – to define more than one Solver model based on the same worksheet formulas. An example of this is provided in the "Portfolio of Securities" worksheet in the SOLVSAMP.XLS workbook that is included with Microsoft Excel. This worksheet defines a portfolio optimization model, where the Solver must determine what percentage of available funds to invest in four different stocks (A, B, C and D) and Treasury bills. The worksheet formulas calculate the portfolio rate of return, and the portfolio risk as measured by the statistical variance of returns. There are two possible approaches to solving this model: (1) Find the maximum rate of return, subject to an upper limit on the portfolio's risk, or (2) Find the minimum risk (variance), subject to a lower limit on the portfolio's return.

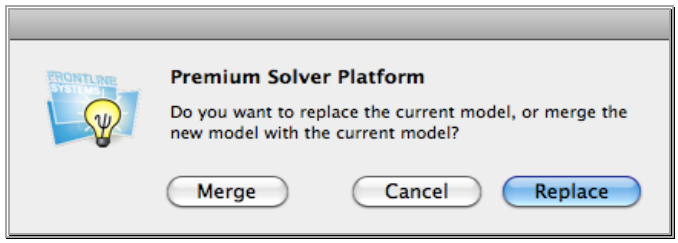
The "current" Solver problem on this worksheet is the one that maximizes return, subject to a constraint on portfolio risk. But both Solver problems ("Maximize Return" and "Minimize Risk") have been set up and their specifications saved (in Classic format) in the lower part of this worksheet, starting at row 21. If you click on the Load Model... button in the Solver options dialog, select cells D21:D29, and click OK, you'll load the specifications for the problem that minimizes risk subject to a constraint on return.

Transferring Models Between Spreadsheets

Another application of Load Model... and Save Model... is to transfer Solver model specifications from one worksheet to another.

Merging Solver Models

You can merge the model specifications being loaded with the current model specifications. Where the specifications necessarily overlap – as in the selection of the objective (and the "maximize" or "minimize" setting) and in the settings of Solver options – the newly loaded specifications take precedence. But the variable cell selections and the constraint left hand sides, relations and right hand sides being loaded are merged into the current model. You are prompted to choose between replacing the current model specifications and merging in the new specifications:



Merging model specifications can be quite useful, for it allows you to build and test smaller, simple Solver models and then combine them into a larger model. Suppose, for example, that you wanted to create a planning model for a manufacturing firm that would take into account both the mix of products being built and the routes along which they were being shipped. You might create two models on one worksheet, one based on the Product Mix example and the other based on the Shipping Routes example in SOLVSAMP.XLS, and test them individually. Then you could combine them with the Merge function, and test the production-distribution model as a whole.

Solver Reports

Introduction

This chapter will help you use the information in the Solver Reports, which can be produced when the Solver finds a solution – or when it *fails* to find a solution, and instead reports that the linearity conditions are not satisfied, or that your model is infeasible. We'll explain how to interpret the values in the Answer, Sensitivity and Limits Reports, available in the standard Excel Solver and Premium Solver Platform for Mac, and how to use the diagnostic Scaling, Linearity and Feasibility Reports and the specialized Solutions and Population Reports. To illustrate the reports, we'll use EXAMPLE1 through EXAMPLE4 in the workbook OptimizationExamples.xlsm, which you can examine by clicking Help, then the Examples button in the initial Help dialog. For the Solutions Report, we'll use other examples including a historically interesting nonlinear equation.

Structure and Transformation Reports

In addition to the eight types of reports described in this chapter, Premium Solver Platform for Mac offers two additional reports that are produced by the new Polymorphic Spreadsheet Interpreter, and requested via the **Solver Model** dialog, as explained in the chapter "Analyzing and Solving Models."

The **Structure Report**, described and illustrated in "Analyzing Model Exceptions," analyzes in depth the linear, quadratic, smooth nonlinear, and non-smooth variables and functions in your model, and helps you find and fix "exceptional" formulas if you're having difficulty build a linear or quadratic programming model.

The **Transformation Report**, shown in "Transforming Your Non-Smooth Model," documents how the Polymorphic Spreadsheet Interpreter can automatically transform your model, replacing non-smooth functions such as IF, MIN, MAX, ABS, AND, OR, and NOT with equivalent expressions using new variables and linear constraints.

Answer, Sensitivity and Limits Reports

The Answer, Sensitivity and Limits Reports are available when the Solver finds an optimal solution for your model; they give you additional information about the solution and its range of applicability. All three 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, which 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.

In Excel VBA, you can use the new object-oriented API to access the information in the Answer and Sensitivity Reports via the properties InitialValue, FinalValue, DualValue, DualUpper, and DualLower of the Variable and Function objects. These objects and properties can also be used in the Solver Platform SDK, outside of Excel. See the chapter "Using the Object-Oriented API" for further information.

Scaling Report

The Scaling Report – available only in Premium Solver Platform for Mac, since it uses the Polymorphic Spreadsheet Interpreter – helps you find and fix poorly scaled formulas in your model. It appears in the Reports list box of the Solver Results dialog when you get a result – such as "Solver could not find a feasible solution," "Solver could not improve the current solution," or "The linearity conditions required by this Solver engine are not satisfied" – that generally indicate other conditions, but *may be due to a poorly scaled model*. If you are puzzled by a result, and you see that the Scaling Report is available, we highly recommend that you select it, click OK, and then examine the report contents. This takes only a moment, and it may save you hours of time if it reveals a scaling problem. See "The Scaling Report" below for a realistic example, using the EXAMPLE4 Portfolio Optimization model.

Feasibility Report

The Feasibility Report helps you diagnose problems in your models.

With the Feasibility Report, you can pinpoint the constraints that interact to make your model infeasible, and correct them as needed. Using the object-oriented API or the Solver Platform SDK, you can access the information in the Feasibility Report via the BoundIndex, BoundStatus, ConstraintIndex and ConstraintStatus properties of the OptIIS object, which is a member of each Variable and Function object.

Solutions Report

Where the Answer Report gives you detailed information about the single "best solution" that appears on the worksheet when the Solver Results dialog is displayed, the Solutions Report gives you objective function and decision variable values for a number of alternative solutions, found during the optimization process. For mixed-integer problems, the report shows each 'incumbent' or feasible integer solution found by the Branch & Bound method. Note that for the LP/Quadratic engine and the Large Scale LP/QP engine, "preprocessing" must be turned of for this to work. For global optimization problems solved with the GRG, LSGRG, and KNITRO Solver engines, the report shows each locally optimal solution found by the Multistart method. For the Evolutionary Solver, the report shows members of the final population of solutions.

Population Report

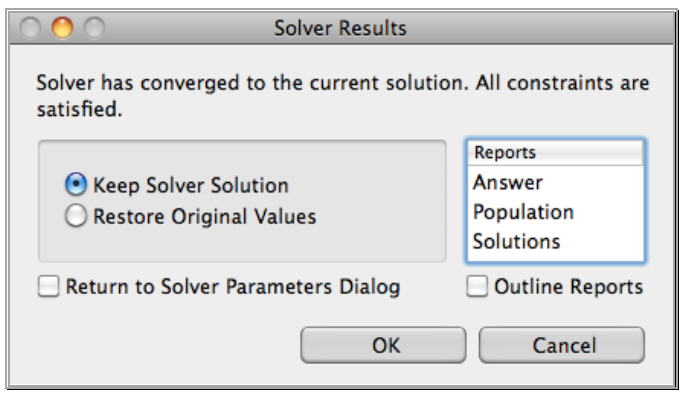
The Population Report is supported by the Evolutionary Solver; it gives you summary statistical information about the entire population of candidate solutions maintained by the Evolutionary Solver at the time the solution process was terminated. It can give you further insight into the quality of solutions found by the Evolutionary Solver.

All of the reports are *Microsoft Excel worksheets*, with grid lines and row and column headings turned off. You can turn the grid lines and headings back on, if you

wish, by choosing Tools Options... and selecting the View tab in the resulting dialog. In Premium Solver Platform for Mac, you can request *outlined* reports, which are worksheets where certain rows are grouped together in an outline structure that you can expand or collapse as you wish. 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 Premium Solver Platform for Mac's reports considerably more useful than those produced by standalone optimization software packages.

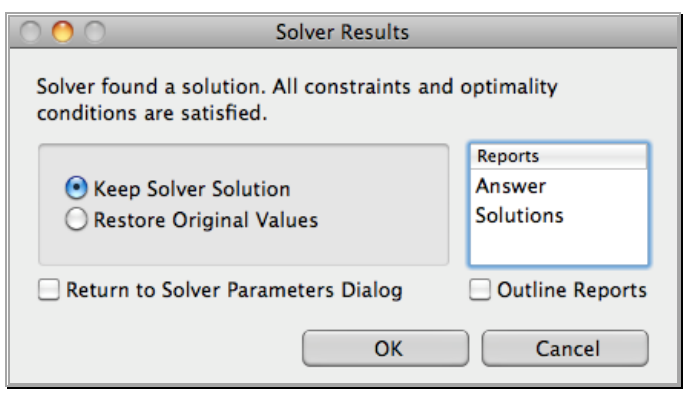
Selecting the Reports

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.



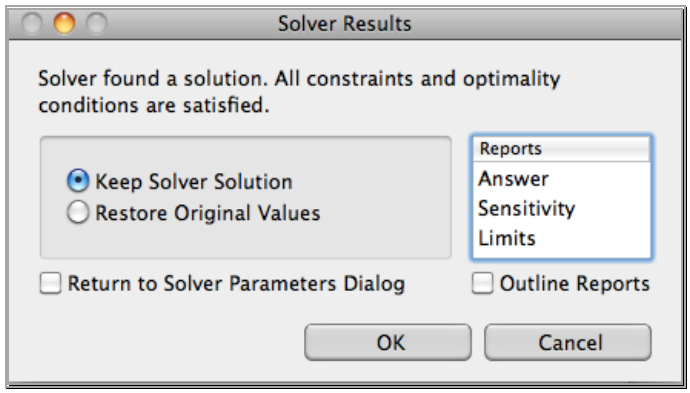
If the solution process was terminated prematurely, the Reports list box in the dialog above will be replaced by the legend "No reports available." If you checked the Bypass Solver Reports box in the Solver Options dialog, the Reports list box will appear with the choices that would otherwise have been available, but they will be grayed out and you will unable to select them.

When the LP/Quadratic Solver, SOCP Barrier Solver, or GRG Nonlinear Solver finds the solution to a mixed-integer programming problem, the Reports list box includes only the Answer Report – the Sensitivity and Limits Reports are not meaningful in this situation. If (and *only* if) the Solver finds more than one integer feasible solution or 'incumbent', the list box also includes the Solutions Report:



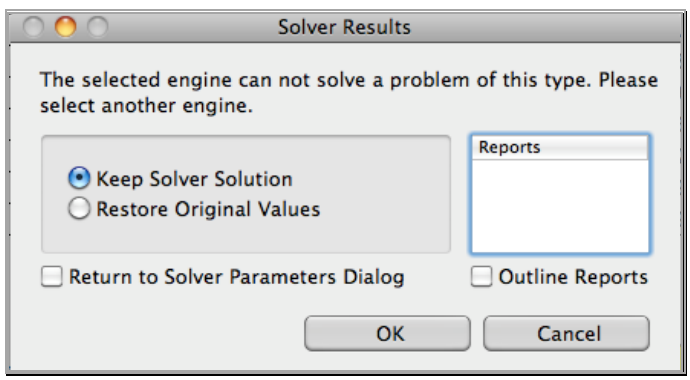
Similarly, when the GRG Nonlinear Solver finds the solution to a global optimization problem, the Reports list box includes only the Answer Report. If the GRG Solver, run with the 'Multistart Search' box checked, finds more than one locally optimal solution, the Reports list box includes the Solutions Report, as shown above.

If you are using the Evolutionary Solver, when the solution process is terminated – for any reason – you'll see a Solver Results dialog like the one below:

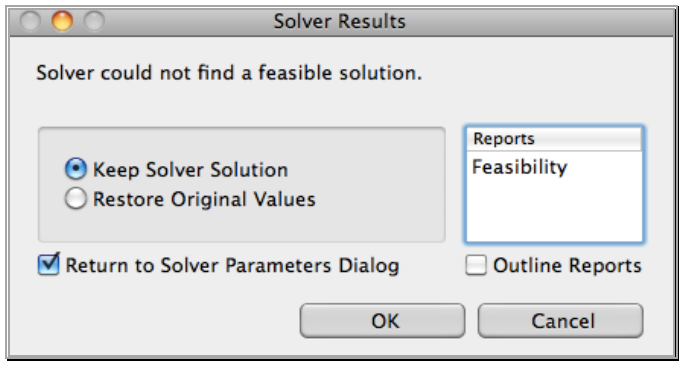


Since the Evolutionary Solver always maintains a population of candidate solutions, including a "best" solution found so far, it offers the Answer, Population and Solutions Reports in all cases – even if it has not found a feasible solution. But since the Evolutionary Solver has no strict test for optimality, linearity or even feasibility, the Linearity, Feasibility, Limits and Sensitivity Reports are not available.

If you've selected the Simplex LP or LP/Quadratic Solver engine, but your model contains nonlinear functions of the decision variables, the Solver will report the error via the Solver Results dialog shown below:



If the Solver finds that your model is infeasible, it displays a Solver Results dialog like the one shown on the next page:



In this case, you can select the "Feasibility". "Feasibility" performs a complete analysis of your model, including bounds on the variables, to find the smallest possible subset of these constraints that is still infeasible. .

Outlining and Comments in Reports

A useful way to control the information you see in reports is to check the box "Outline Reports" in the Solver Results dialog, to produce the reports you've selected in outlined format. Outlining groups the variables and constraints in the reports into "blocks," just as you entered them in the Solver Parameters dialog; you can expand or collapse the groups to see only the information you want.

Reports in outlined format can display a descriptive comment on each "block" of variables and constraints. Comments for constraint blocks are entered in the Add Constraint and Change Constraint dialogs, displayed when you click the Add and Change buttons in the Solver Parameters dialog. To add comments to blocks of variables, click the Variables button to display the Variables list box, then click the Add or Change buttons to display the Add Variable or Change Variable dialog.

Using the Solver Results Dialog

When the Reports list box is available, you can select one or more of the reports shown. Simply click on the report names to select the reports you want, or press Alt-R and then down-arrow from the keyboard. To select more than one report, hold down the APPLE 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," and optionally save the decision variable values in a named scenario by clicking on the Save Scenario... button. When you click on OK, the reports will be produced. Clicking on Cancel instead will cancel generation of the reports, and will discard the solution (restoring the original values). The reports are Microsoft Excel worksheets that are inserted in the current workbook, just before the sheet containing the Solver model.

After the reports (if any) are produced, the Solver will return to worksheet Ready mode unless you have checked the box "Return to Solver Parameters Dialog." This check box saves you the effort of selecting Tools Premium Solver... over and over, if you are solving several variations of the same problem, or solving with different option settings. When you check the "Return to Solver Parameters Dialog" box, it remains checked (until you change it) for the duration of your Excel session. To return to worksheet Ready mode, you can either click the Close button in the Solver Parameters dialog, or uncheck this box in the Solver Results dialog.

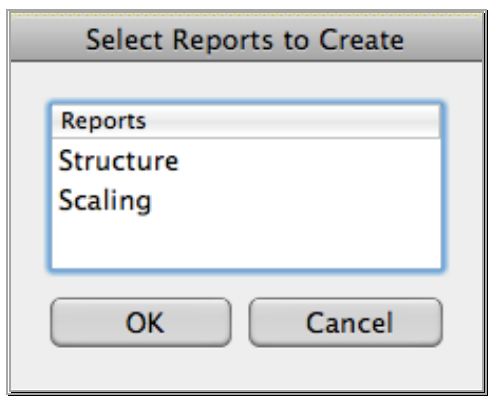
The Scaling Report

The effects of poor scaling in a large, complex optimization model can be among the most difficult problems to identify and resolve. It can cause Solver engines to return a variety of messages, with results that are suboptimal or otherwise very different from your expectations. Most Solver engines include an Automatic Scaling option to deal with scaling problems, but this can only help with the Solver's internal calculations – not with poor scaling that occurs in the middle of your Excel model.

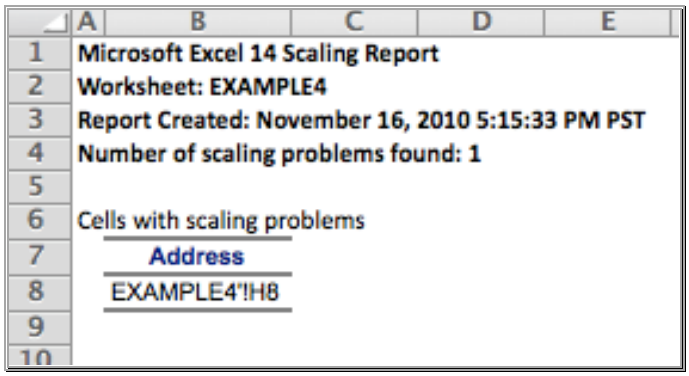
For example, if one of your formulas adds or subtracts two quantities of very different magnitudes, such as a dollar amount in millions or billions and a return or risk measure in fractions of a percent, the result will be accurate to only a few significant digits. The effect might not be apparent given the initial values of the variables, but when the Solver explores Trial Solutions with very different values for the variables, the effect will be magnified.

You can see an example of the effects of poor scaling if you open **OptimizationExamples.xlsm** (in the Solver Parameters dialog, just click the Help button, then the Examples button), and select worksheet EXAMPLE4. Suppose that in this Portfolio Optimization model, you decide to make a simple change: Instead of using percentages for the stock allocations, you'd rather see the actual dollars to be invested, in your $1 billion institutional portfolio. So you change the constraint TotalPortfolio = 1 (or 100%) to be TotalPortfolio = 1000000000. You change the cell formatting to display large numbers instead of percentages, and you select the GRG Nonlinear Solver (for the sake of this example, since it is more susceptible to scaling problems than the LP/Quadratic Solver). When you click Solve, you're surprised to find that the Solver reports it cannot find a feasible solution, as shown on the next page.

The Trial Solution on the worksheet doesn't even allocate the entire $1 billion to stocks. Since you're familiar with the Solver options, you turn on the Use Automatic Scaling box in the GRG Solver Options dialog and click Solve again, but you just get a different infeasible result. What's wrong with the Solver? (Past users of Premium Solver Platform for Mac have on occasions wrestled with problems just like this.)



Noticing that the **Scaling Report** is available in the Solver Model dialog, after performing a Check Model, you select this report and click OK. The Scaling Report is inserted into your workbook:



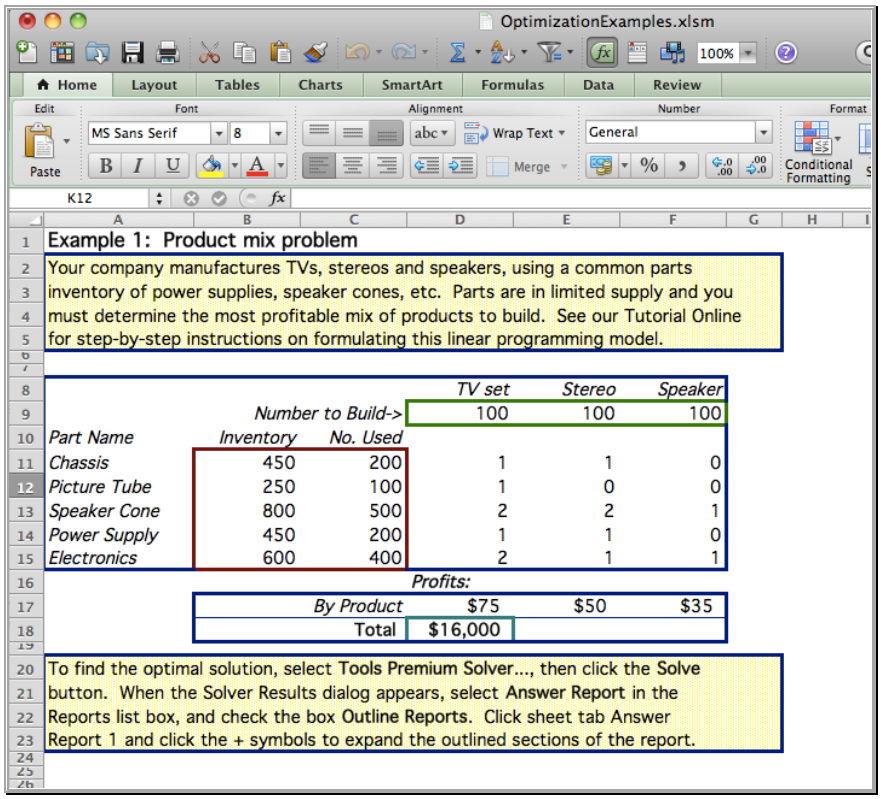
	A	B	C	D	E
1	Microsoft Excel 14 Scaling Report				
2	Worksheet: EXAMPLE4				
3	Report Created: November 16, 2010 5:15:33 PM PST				
4	Number of scaling problems found: 1				
5					
6	Cells with scaling problems				
7		Address			
8		EXAMPLE4'!I8			
9					
10					

The report indicates that there are scaling problems with the formula at the cell with defined name 'Portfolio_Variance' (EXAMPLE4!I17). In a very large model, this cell might be very hard to find by manual inspection. You can click on the underlined cell reference to jump to cell I17 on the EXAMPLE4 worksheet. You see that the Variance is a *very* large number – 8.26E+14, or 826 trillion. The Scaling Report has drawn your attention to a scaling issue in the formulas that calculate your model – outside of the Solver's own calculations.

In seeking the optimal solution, the Solver is likely to try extreme values – large and small – for the variables. This doesn't cause problems when the largest value is 100% or 1 and the smallest is 0, but it does cause problems when the largest value is 1 billion. At this point, calculation of the Portfolio Variance involves adding a *very small* value and a *very large* one (the Stock 5 variance times 1 billion *squared*) which leads to a loss of accuracy. This loss of accuracy leads directly to the Solver's problems finding a solution.

An Example Model

To illustrate the other reports provided by Premium Solver Platform for Mac, we'll start with the model on worksheet EXAMPLE1 in the workbook OptimizationExamples.xlsm.



You can easily load this model by clicking the Help button, then the Examples button in the Solver Parameters dialog

First, we'll solve this model in its original form, using the LP/Quadratic and GRG Solvers, and produce Answer, Sensitivity and Limits Reports. 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.

Next, we'll change the available inventory of Chassis at cell B11 to -1. This is shown in OptimizationExamples.xlsm on the EXAMPLE2 worksheet. When we attempt to solve, we receive the message "Solver could not find a feasible solution," and we can produce the report shown below in the section "The **Feasibility Report**."

Next, we'll deliberately introduce a *nonlinear function* into the model, by editing the formula at cell C11 to read =SUMPRODUCT(D11:F11,$D$9:$F$9)^0.9. This is shown in OptimizationExamples.xlsm on the EXAMPLE3 worksheet. When we attempt to solve this model with the Simplex LP or LP/Quadratic Solver, we'll receive the message "The linearity conditions required by this Solver engine are not satisfied," and we can produce the report shown below in "The **Linearity Report**."

Returning to the unmodified version of EXAMPLE1, we'll solve the model using the Evolutionary Solver, waiting until we receive the message "Solver cannot improve the current solution." This allows us to produce the report shown below in "The **Population Report**."

In Premium Solver Platform V10.5, the **Solutions Report** has been generalized to report multiple solutions for integer programming problems, global optimization problems, and non-smooth optimization problems, solved by any of the built-in or plug-in Solver engines. We'll illustrate this useful report with additional examples.

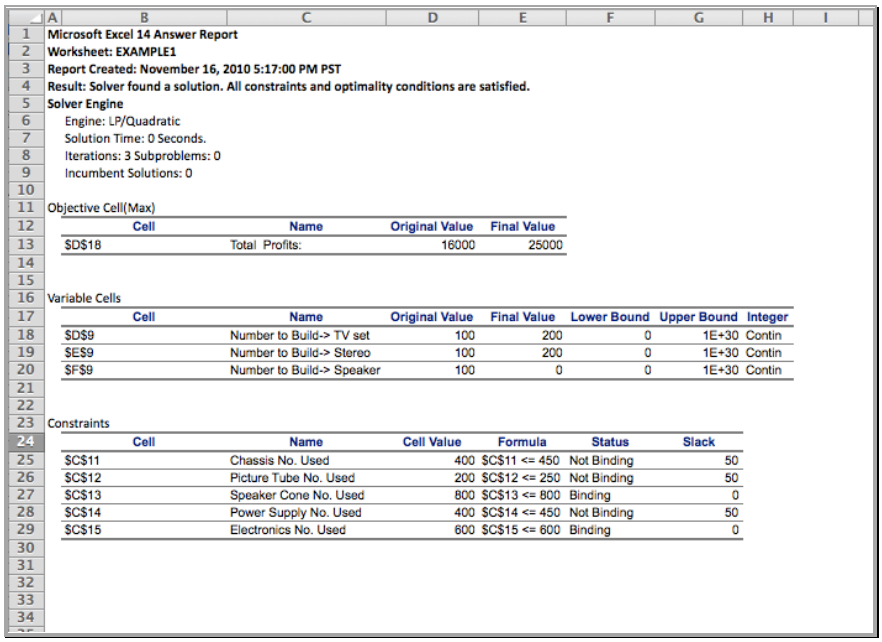
The Answer Report

The Answer Report, which is available whenever a solution has been found, provides basic information about the decision variables and constraints in the model. 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. In Version 10.5, the Answer Report includes the message that appeared in the Solver Results dialog, the name of the Solver engine used to solve the problem, and statistics such as the time, iterations and subproblems required to solve the problem. An example Answer Report for the worksheet model EXAMPLE1 (when there are no upper bounds on the decision variables) is shown on the next page.

First shown are the objective function (Set 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. (In the standard Microsoft Excel Solver, an exception to this occurs when the right hand side of a constraint is itself a function of the decision variables. In Premium Solver Platform for Mac, this special case is handled differently, and the slack value for a binding constraint will *always* be zero.)

This example shows the effect of automatic outlining of the Solver reports, which you can select via the "Outline Reports" check box in the Solver Results dialog. The outline groups correspond directly to the blocks of variables and constraints you entered in the Solver Parameters dialog – one group per row in the Constraints or Variables list box. Comments entered in the Add Constraint and Add Variable dialogs for each block appear in the Answer Report; they are visible whether the outline is expanded or collapsed.



Microsoft Excel 14 Answer Report
Worksheet: EXAMPLE1
Report Created: November 16, 2010 5:17:00 PM PST
Result: Solver found a solution. All constraints and optimality conditions are satisfied.
Solver Engine
Engine: LP/Quadratic
Solution Time: 0 Seconds.
Iterations: 3 Subproblems: 0
Incumbent Solutions: 0

Objective Cell(Max)

Cell	Name	Original Value	Final Value
$D$18	Total Profits:	16000	25000

Variable Cells

Cell	Name	Original Value	Final Value	Lower Bound	Upper Bound	Integer
$D$9	Number to Build-> TV set	100	200	0	1E+30	Contin
$E$9	Number to Build-> Stereo	100	200	0	1E+30	Contin
$F$9	Number to Build-> Speaker	100	0	0	1E+30	Contin

Constraints

Cell	Name	Cell Value	Formula	Status	Slack
$C$11	Chassis No. Used	400	$C$11 <= 450	Not Binding	50
$C$12	Picture Tube No. Used	200	$C$12 <= 250	Not Binding	50
$C$13	Speaker Cone No. Used	800	$C$13 <= 800	Binding	0
$C$14	Power Supply No. Used	400	$C$14 <= 450	Not Binding	50
$C$15	Electronics No. Used	600	$C$15 <= 600	Binding	0

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 variable (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 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 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 can make a significant difference in how the Reduced Gradient, Lagrange Multiplier, Reduced Cost and Shadow Prices are displayed. 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 with the Format menu.

An example of a Sensitivity Report generated for EXAMPLE1 when the Solver engine is the nonlinear GRG solver (and there are no upper bounds on the variables) 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 Solver, the report will also appear in this format.

Microsoft Excel 14 Sensitivity Report
Worksheet: EXAMPLE1
Report Created: November 16, 2010 5:18:39 PM PST

Objective Cell(Max)

Cell	Name	Final Value
$D$18	Total Profits:	25000

Decision Variable Cells

Cell	Name	Final Value	Reduced Gradient
$D$9	Number to Build-> TV set	200	0
$E$9	Number to Build-> Stereo	200	0
$F$9	Number to Build-> Speaker	0	-2.5

Constraints

Cell	Name	Final Value	Lagrange Multiplier
$C$11	Chassis No. Used	400	0
$C$12	Picture Tube No. Used	200	0
$C$13	Speaker Cone No. Used	800	12.49999981
$C$14	Power Supply No. Used	400	0
$C$15	Electronics No. Used	600	25

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 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 above, the dual value for producing speakers is -2.5, 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

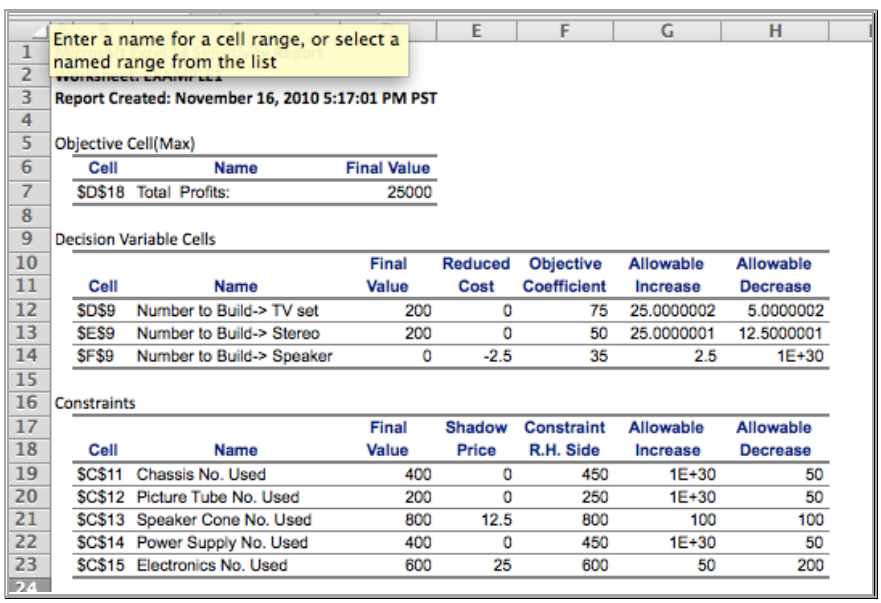
on the previous page, increasing the number of electronics units from 600 to 601 will allow the Solver to increase total profit by $25.

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 problems includes this range information.

A Sensitivity Report for EXAMPLE1 when the Solver engine is the standard Simplex or LP/Quadratic Solver (and there are no upper bounds on the decision variables) 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.



Report Created: November 16, 2010 5:17:01 PM PST

Objective Cell(Max)

Cell	Name	Final Value
$D$18	Total Profits:	25000

Decision Variable Cells

Cell	Name	Final Value	Reduced Cost	Objective Coefficient	Allowable Increase	Allowable Decrease
$D$9	Number to Build-> TV set	200	0	75	25.0000002	5.0000002
$E$9	Number to Build-> Stereo	200	0	50	25.0000001	12.5000001
$F$9	Number to Build-> Speaker	0	-2.5	35	2.5	1E+30

Constraints

Cell	Name	Final Value	Shadow Price	Constraint R.H. Side	Allowable Increase	Allowable Decrease
$C$11	Chassis No. Used	400	0	450	1E+30	50
$C$12	Picture Tube No. Used	200	0	250	1E+30	50
$C$13	Speaker Cone No. Used	800	12.5	800	100	100
$C$14	Power Supply No. Used	400	0	450	1E+30	50
$C$15	Electronics No. Used	600	25	600	50	200

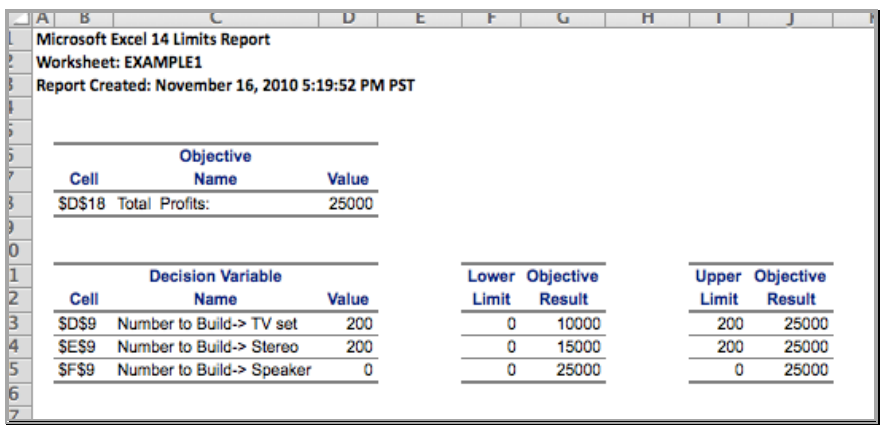
For each decision variable, the report shows its coefficient in the objective function, and the amount by which this coefficient could be increased or decreased without changing the dual value. In the example below, we'd still build 200 TV sets even if the profitability of TV sets decreased up to $5 per unit. Beyond that point, or if the unit profit of speakers increased by more than $2.50, 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 1E+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*.

The Limits Report

The Limits Report was designed by Microsoft to provide a specialized kind of "sensitivity analysis" information. It is created by re-running the Solver 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 EXAMPLE1 is shown below.



Microsoft Excel 14 Limits Report
Worksheet: EXAMPLE1
Report Created: November 16, 2010 5:19:52 PM PST

		Objective	
Cell		Name	Value
$D$18	Total	Profits:	25000

		Decision Variable		Lower	Objective	Upper	Objective
Cell		Name	Value	Limit	Result	Limit	Result
$D$9	Number to Build->	TV set	200	0	10000	200	25000
$E$9	Number to Build->	Stereo	200	0	15000	200	25000
$F$9	Number to Build->	Speaker	0	0	25000	0	25000

The Feasibility Report

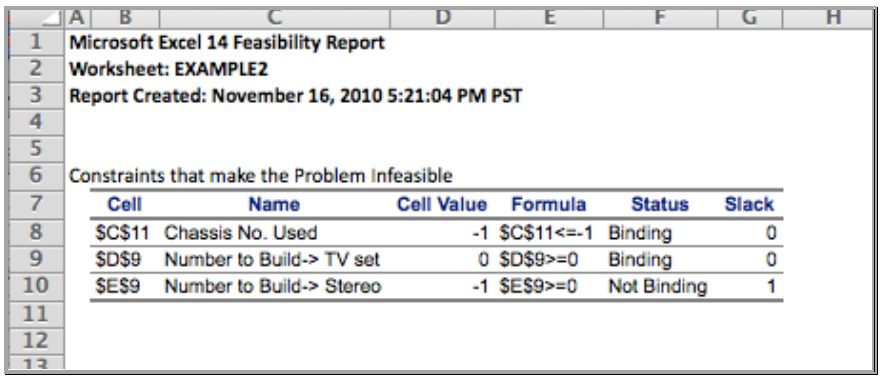
The purpose of the Feasibility Report is to help you isolate the source of infeasibilities in your model. Most often, an infeasible result simply means that you've made a mistake in formulating your model, such as specifying a <= relation when you meant to use >=. However, if your model contains hundreds or thousands of constraints, it can be quite challenging to locate an error of this type. By isolating the infeasibility to a small subset of the constraints, the Feasibility Report can show you where to look, and hence save you a good deal of time.

To produce the Feasibility Report, the Solver may test many different variations of your model, each one with different combinations of your original constraints. This process ultimately leads to a so-called "Irreducibly Infeasible System" (IIS) of constraints and variable bounds which, taken together, make the problem infeasible, but with the property that if any one of the constraints or bounds is removed from the IIS, the problem becomes feasible.

In a model with many constraints that "interact" with each other in complex ways, there may be many possible subsets of the constraints and bounds that constitute an IIS. Often, some of these subsets have many fewer constraints than others. The Solver attempts to find an IIS containing as few constraints as possible, trying first to

eliminate “formula” constraints and then to eliminate simple variable bounds – since it is usually easier to understand the effects of variable bounds on the infeasibility of the resulting IIS.

If we attempt to solve EXAMPLE2 in the **OptimizationExamples.xlsm** workbook – which is identical to EXAMPLE1 except that cell B11 (the right hand side of the constraint C11 <= B11) is set to -1 – we receive the message “Solver could not find a feasible solution.” At this point, we know only that the problem is somewhere in the set of five constraints (C11:C15 <= B11:B15) and three bounds on the variables. To pinpoint the problem, we select Feasibility from the Reports list box, producing a report like the one shown below. The Feasibility Report narrows the full set of constraints to the single constraint C11 <= B11 and bounds on variables D9 and E9.



Microsoft Excel 14 Feasibility Report
Worksheet: EXAMPLE2
Report Created: November 16, 2010 5:21:04 PM PST

Constraints that make the Problem Infeasible

Cell	Name	Cell Value	Formula	Status	Slack
$C$11	Chassis No. Used	-1	$C$11<=-1	Binding	0
$D$9	Number to Build-> TV set	0	$D$9>=0	Binding	0
$E$9	Number to Build-> Stereo	-1	$E$9>=0	Not Binding	1

If your model is very large, computing the IIS may take a good deal of time. The Solver displays an estimated “% Done” on the Excel status bar as it solves variations of your model, and you can always interrupt the process by pressing ESC (in which case no report appears). Instead of the full Feasibility Report, which analyzes both the constraints and variable bounds in your model and attempts to eliminate as many of them as possible, you can produce the “Feasibility-Bounds” version of the report, which analyzes only the constraints while keeping the variable bounds in force. This report may be sufficient to isolate the source of the infeasibility, but you must take into account the bounds on all of the variables when reading it.

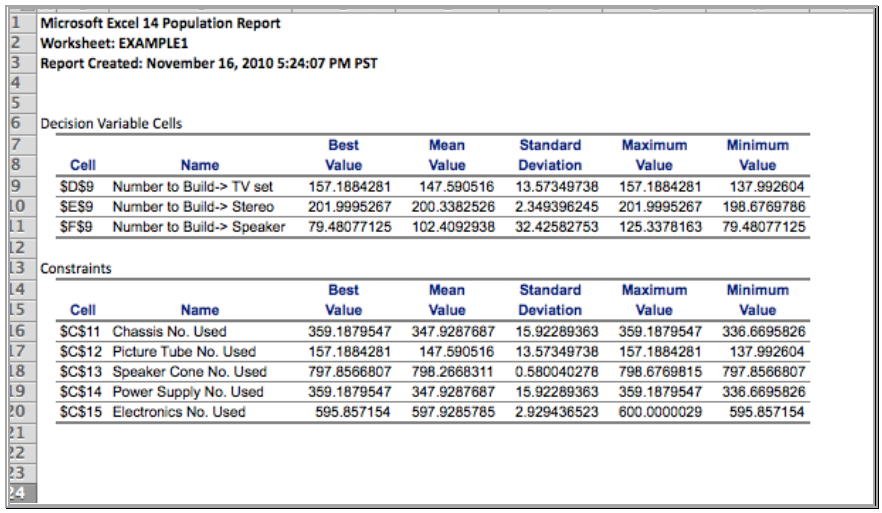
In some cases, of course, there may be no error in your model – it may correctly describe the real-world situation, and the fact that it is infeasible will probably tell you something important about the situation you are modeling. Even in such cases, the Feasibility Report can help you focus on the aspects of the real-world situation that contribute to the infeasibility, and what you can do about them.

The Population Report

The Population Report gives you summary information about the entire population of candidate solutions maintained by the Evolutionary Solver at the end of the solution process. The Population Report can give you insight into the performance of the Evolutionary Solver as well as the characteristics of your model, and help you decide whether additional runs of the Evolutionary Solver are likely to yield even better solutions.

For each variable and constraint, the Population Report shows the best value found by the Evolutionary Solver, and the mean (average) value, standard deviation, maximum value, and minimum value of that variable or constraint across the entire population of candidate solutions at the end of the solution process. These values will give you an idea of the diversity of solutions represented by the population.

If we run the Evolutionary Solver on EXAMPLE1 with a Max Subproblems limit of 5000, and upper bounds of 200 on the variables, we find a solution of D9 = E9 = 200 and F9 = 0 (the same as the linear programming optimal solution). We can then produce a Population Report like the one below.



Microsoft Excel 14 Population Report
Worksheet: EXAMPLE1
Report Created: November 16, 2010 5:24:07 PM PST

Decision Variable Cells

Cell	Name	Best Value	Mean Value	Standard Deviation	Maximum Value	Minimum Value
$D$9	Number to Build-> TV set	157.1884281	147.590516	13.57349738	157.1884281	137.992604
$E$9	Number to Build-> Stereo	201.9995267	200.3382526	2.349396245	201.9995267	198.6769786
$F$9	Number to Build-> Speaker	79.48077125	102.4092938	32.42582753	125.3378163	79.48077125

Constraints

Cell	Name	Best Value	Mean Value	Standard Deviation	Maximum Value	Minimum Value
$C$11	Chassis No. Used	359.1879547	347.9287687	15.92289363	359.1879547	336.6695826
$C$12	Picture Tube No. Used	157.1884281	147.590516	13.57349738	157.1884281	137.992604
$C$13	Speaker Cone No. Used	797.8566807	798.2668311	0.580040278	798.6769815	797.8566807
$C$14	Power Supply No. Used	359.1879547	347.9287687	15.92289363	359.1879547	336.6695826
$C$15	Electronics No. Used	595.857154	597.9285785	2.929436523	600.0000029	595.857154

You can see that the Best Values of the variables are far from the Mean Values across the whole population, but they are equal to the Maximum Values for cells D9 and E9, and to the Minimum Value of cell F9. Since the solution is feasible, and since the optimization process tends to drive variable values to extremes, this may indicate that we have found a globally optimal solution (which is true in this case). The Standard Deviations are relatively large, but this is not too surprising since points in the population have not yet converged to the point where we would receive "Solver has converged to the current solution."

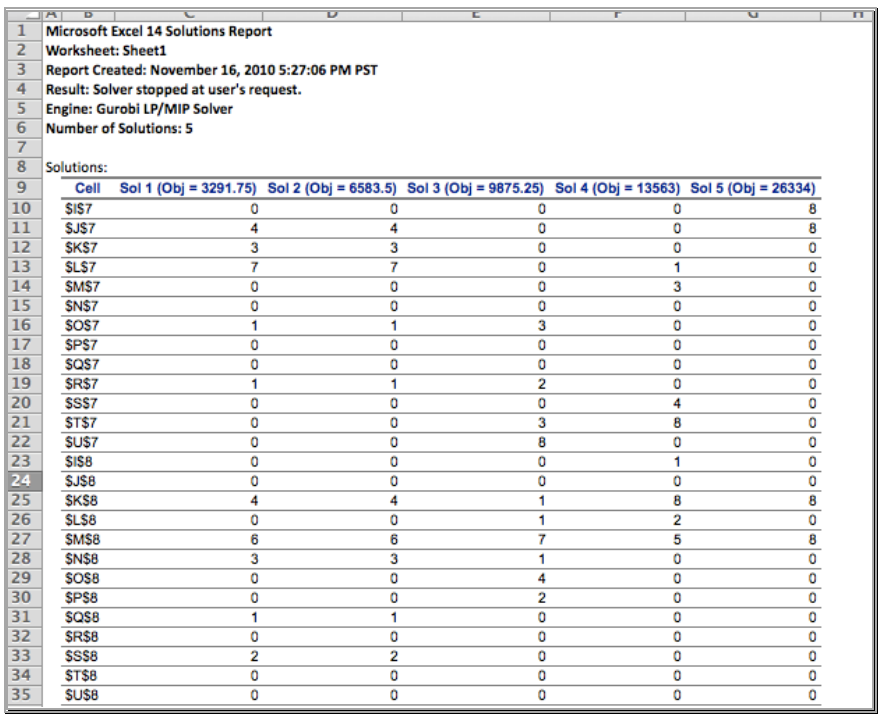
How you interpret the Population Report depends in part on your knowledge of the problem, and past experience solving it with the Evolutionary Solver or with other Solver engines. For example, if the Best Values are similar from run to run, and if the Standard Deviations are small, this may be reason for confidence that your solution is close to the global optimum. However, if the Best Values vary from run to run, small Standard Deviations might indicate a lack of diversity in the population, suggesting that you should increase the Mutation Rate and run the Solver again.

The Solutions Report

Where the Answer Report gives you detailed information about the single "best solution" that appears on the worksheet when the Solver Results dialog is displayed, the Solutions Report gives you objective function and decision variable values for a number of alternative solutions, found during the optimization process.

Integer Programming Problems

For mixed-integer problems, the report shows each 'incumbent' or feasible integer solution found by the Branch & Bound method during the solution process. Below is an example of the Solutions Report:

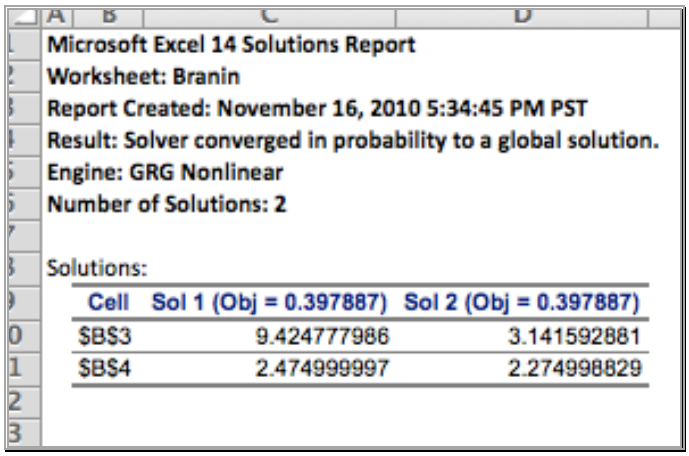


1	Microsoft Excel 14 Solutions Report					
2	Worksheet: Sheet1					
3	Report Created: November 16, 2010 5:27:06 PM PST					
4	Result: Solver stopped at user's request.					
5	Engine: Gurobi LP/MIP Solver					
6	Number of Solutions: 5					
7						
8	Solutions:					
9	Cell	Sol 1 (Obj = 3291.75)	Sol 2 (Obj = 6583.5)	Sol 3 (Obj = 9875.25)	Sol 4 (Obj = 13563)	Sol 5 (Obj = 26334)
10	$I$7	0	0	0	0	8
11	$J$7	4	4	0	0	8
12	$K$7	3	3	0	0	0
13	$L$7	7	7	0	1	0
14	$M$7	0	0	0	3	0
15	$N$7	0	0	0	0	0
16	$O$7	1	1	3	0	0
17	$P$7	0	0	0	0	0
18	$Q$7	0	0	0	0	0
19	$R$7	1	1	2	0	0
20	$S$7	0	0	0	4	0
21	$T$7	0	0	3	8	0
22	$U$7	0	0	8	0	0
23	$I$8	0	0	0	1	0
24	$J$8	0	0	0	0	0
25	$K$8	4	4	1	8	8
26	$L$8	0	0	1	2	0
27	$M$8	6	6	7	5	8
28	$N$8	3	3	1	0	0
29	$O$8	0	0	4	0	0
30	$P$8	0	0	2	0	0
31	$Q$8	1	1	0	0	0
32	$R$8	0	0	0	0	0
33	$S$8	2	2	0	0	0
34	$T$8	0	0	0	0	0
35	$U$8	0	0	0	0	0

This problem was solved by the LP/Quadratic Solver with the Integer Tolerance set to 0.0 and all Cuts & Heuristics disabled. (On this problem, with Cuts & Heuristics enabled, the Solver quickly finds the true integer optimal solution as the first incumbent; the Solutions Report is available only when multiple incumbents are found.) As shown above, three incumbents were found.

Global Optimization Problems

For global optimization problems, the report shows each locally optimal solution found by the Multistart method. On the next page is an example of the Solutions Report for a simple two-variable global optimization problem Branin.xls, solved by the GRG Nonlinear Solver with the Multistart Search option selected:



Microsoft Excel 14 Solutions Report
Worksheet: Branin
Report Created: November 16, 2010 5:34:45 PM PST
Result: Solver converged in probability to a global solution.
Engine: GRG Nonlinear
Number of Solutions: 2

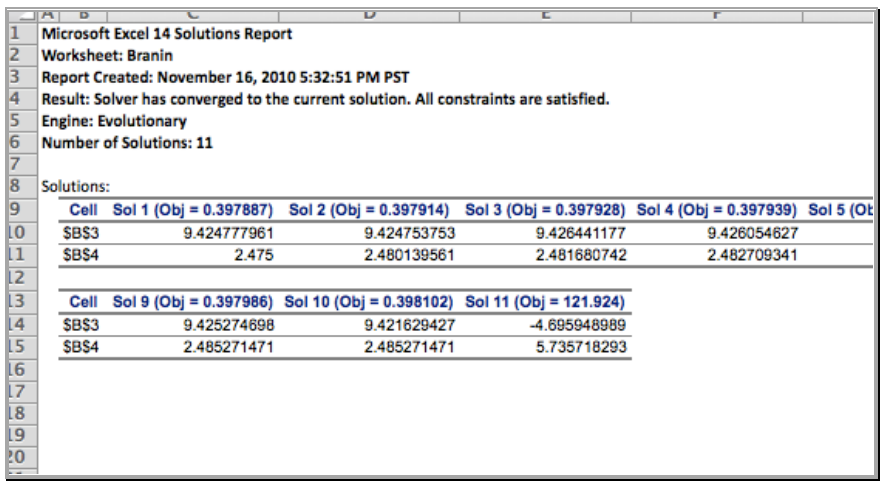
Solutions:

Cell	Sol 1 (Obj = 0.397887)	Sol 2 (Obj = 0.397887)
$B$3	9.424777986	3.141592881
$B$4	2.474999997	2.274998829

In this problem, the "Branin function" must be minimized for variables x and y, subject to bounds -5 <= x, y <= 10. There are three distinct locally optimal solutions with objective values 2.7911 (worst), 0.5989 (better) and 0.3979 (best and globally optimal). The Solver was started at the point x = -2.5, y = 10, which is close to the worst of the three locally optimal solutions. The Multistart Search process runs the Solver from representative starting points in 'clusters' of randomly selected points; on this run, it first found a solution close to the worst locally optimal point, then found a solution at the best and globally optimal point.

Non-Smooth Optimization Problems

For arbitrary non-smooth optimization problems, the report shows members of the Solver's final population of solutions. Below is an example of the Solutions Report for the global optimization problem Branin.xls, solved by the Evolutionary Solver.



Microsoft Excel 14 Solutions Report
Worksheet: Branin
Report Created: November 16, 2010 5:32:51 PM PST
Result: Solver has converged to the current solution. All constraints are satisfied.
Engine: Evolutionary
Number of Solutions: 11

Solutions:

Cell	Sol 1 (Obj = 0.397887)	Sol 2 (Obj = 0.397914)	Sol 3 (Obj = 0.397928)	Sol 4 (Obj = 0.397939)	Sol 5 (Ob
$B$3	9.424777961	9.424753753	9.426441177	9.426054627	
$B$4	2.475	2.480139561	2.481680742	2.482709341	

Cell	Sol 9 (Obj = 0.397986)	Sol 10 (Obj = 0.398102)	Sol 11 (Obj = 121.924)
$B$3	9.425274698	9.421629427	-4.695948989
$B$4	2.485271471	2.485271471	5.735718293

Again the Solver was started at the point x = -2.5, y = 10, close to the worst of the three locally optimal solutions, and it was given a limit of only 200 subproblems. Unlike the Solutions Report for gradient-based nonlinear optimizers like the GRG Nonlinear Solver, the final population of solutions is not likely to include many distinct locally optimal points. The best solutions in the Evolutionary Solver's final

population are all in the neighborhood of the globally optimal solution, which is x = 3.14159, y = 2.2750. But since the Evolutionary Solver doesn't require gradient information or tests for local optimality, it is unlikely to find the globally optimal solution with very high accuracy for a smooth nonlinear problem like Branin.xls.

Using VBA Functions

Controlling the Solver's Operation

This chapter explains how to control the Solver using VBA functions, which are upward compatible from the VBA functions supported by the standard Excel SolverUsing traditional VBA functions, you can display or completely hide the Solver dialog boxes, create or modify the choices of objective (Set Cell), variables (Changing Cells) and constraints, and produce reports. You do this by calling a set of Solver-specific functions from a macro program you write in Visual Basic Applications Edition (VBA). If you need to work with solution values or report information in your VBA code, create and solve multiple optimization problems, or 'port' your code to run as a standalone application, you may find that the object-oriented API is a better choice.

Running Predefined Solver Models

Controlling the Solver can be as simple as *adding one line* to your macro program 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 VBA module, you can simply add a reference to the Solver add-in, activate the worksheet, and add one line to call the function **SolverSolve** in VBA.

Limitations in VBA on the Mac

The Premium Solver for the Mac is a separate application, which gets started each time you solve a model, or perform a "Check Model". Because this is a separate process, this means that when you run a macro that has a line such as SolverSolve in it, the macro will not wait for the Solver to finish; it will keep executing your VBA code. Please keep this in mind as you write your macros. For more information on this, also see the SolverSolve function described below.

Referencing Functions in Visual Basic

To use the VBA functions, your Visual Basic module must include a reference to the **Solver** add-in (Solver.xla). Press Alt-F11 to open the Visual Basic Editor, choose Tools References... and make sure that the box next to **PremiumSolver** is checked.

Checking Function Return Values

The Solver functions generally return integer values, *which you should check in your VBA code*. The normal return value is 0, indicating that the function succeeded. Other possible return values are given in the descriptions of the individual functions. If the arguments you supply are invalid, an error condition can be raised, which you would have to handle via an On Error VBA statement.

One group of functions can return a variety of numeric, logical, string or array values, depending on the arguments you supply. These functions (SolverGet, SolverOkGet, etc.) may be used to "read" the settings of the current Solver model, on the active sheet or any other worksheet whose name you supply.

Standard, Model and Premium Macro Functions

The following sections describe each of the VBA function calls supported by Premium Solver Platform for Mac. These functions are a compatible superset of the function calls available in the standard Excel Solver.

The functions are listed alphabetically in three groups. The first group consists of functions available in both the standard Excel Solver and Premium Solver Platform for Mac. The second group (Premium VBA Functions) consists of functions that are available only in Premium Solver Platform for Mac. The third group (Solver Model VBA Functions) consists of functions that are available only in Premium Solver Platform for Mac. If you want to write VBA code that will work with both the standard Solver and Premium Solver Platform for Mac, you should limit yourself to functions in the first group, and consult the notes on each function call to determine which arguments are supported by the standard Solver.

Standard VBA Functions

The VBA functions in this section are available in both the standard Excel Solver and Premium Solver Platform for Mac. Some of these functions have extra arguments that are supported only in Premium Solver Platform for Mac, as noted in each function description.

SolverAdd (Form 1)

Equivalent to choosing Solver... from the Tools menu and pressing the Add button in the Solver Parameters dialog box. Adds a constraint to the current problem.

VBA Syntax

SolverAdd (CellRef:=, Relation:=, FormulaText:=, Comment:=)

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 hand 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)
6	**dif** (CellRef is an alldifferent group)
7	**soc** (CellRef belongs to a second order cone)
8	**src** (CellRef belongs to a rotated second order cone)

FormulaText is the right hand side of the constraint and will often be a single number, but it may be a formula (as text) or a reference to a range of cells.

Comment is a string corresponding to the Comment field in the Add Constraint dialog, only in Premium Solver Platform for Mac.

The standard Excel Solver supports only **Relation** values 1 to 5. If **Relation** is 4 to 8, **FormulaText** is ignored, and **CellRef** must be a subset of the 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 call.

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

SolverAdd (Form 2)

Equivalent to choosing Solver... from the Tools menu, 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. This form is supported only by Premium Solver Platform for Mac.

VBA Syntax

SolverAdd (CellRef:=, Comment:=)

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

Comment is a string corresponding to the Comment field in the Add Variable Cells dialog, only in Premium Solver Platform for Mac.

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 Solver... from the Tools menu and pressing the Change button in the Solver Parameters dialog box. Changes the right hand side of an existing constraint.

VBA Syntax

SolverChange (CellRef:=, Relation:=, FormulaText:=, Comment:=)

For an explanation of the arguments and selection of 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, then use SolverAdd to add the constraint in the form you want.

SolverChange (Form 2)

Equivalent to choosing Solver... from the Tools menu, pressing the Variables button, and then pressing the Change button in the Solver Parameters dialog box. Changes a set of decision variable cells. This form is supported only by Premium Solver Platform for Mac.

VBA Syntax

SolverChange (CellRef:=, Relation:=, Comment:=)

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

Comment is a string corresponding to the Comment field in the Change Variable Cells dialog, only in Premium Solver Platform for Mac.

Report is no longer used, but is included for compatibility with previous versions of Premium Solver Platform for Mac.

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 Solver... from the Tools menu and pressing the Delete button in the Solver Parameters dialog box. Deletes an existing constraint.

VBA Syntax

SolverDelete (CellRef:=, Relation:=, FormulaText:=)

For an explanation of the arguments and selection of constraints, see **SolverAdd**. The **FormulaText** argument is optional, but if present, it is used to confirm that the correct constraint block is being deleted.

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 Solver... from the Tools menu, pressing the Variables button, and then pressing the Delete button in the Solver Parameters dialog box. Deletes an existing set of variable cells. This form is supported only by Premium Solver Platform for Mac.

VBA Syntax

SolverDelete (CellRef:=)

CellRef is a reference to a cell or a range of cells on the active worksheet, currently defined in the Variable Cells list box as decision 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 selecting options and clicking OK in the Solver Results dialog that appears when the solution process is finished. The dialog box will *not* be displayed.

VBA Syntax

SolverFinish (KeepFinal:=, ReportArray:= , OutlineReports:=, ReportDesc:=)

The **ReportDesc** and **OutlineReports** arguments are available only in Premium Solver Platform for Mac.

KeepFinal is the number 1, 2 or 3 and specifies whether to keep or discard the final solution. If **KeepFinal** is 1 or omitted, the final solution values are kept in the variable cells. If **KeepFinal** is 2, the final solution values are discarded and the former values of the variable cells are restored.

ReportArray is an array argument specifying what reports should be produced. If the Solver found a solution, it may have any of the following values:

If ReportArray is	The Solver creates
Array(1)	An Answer Report
Array(2)	A Sensitivity Report
Array(3)	A Limits Report
Array(4)	A Solutions Report

Array(4) is used only for integer programming and global optimization problems. A combination of these values produces multiple reports. For example, if **ReportArray** = Array(1,2), the Solver will create an Answer Report and a Sensitivity Report.

If you are using the Evolutionary Solver engine, you can produce an Answer Report, a Population Report or a Solutions Report unless **SolverSolve** returns 18, 19 or 20 (which means that the Solver returned an error before a population was formed):

If ReportArray is	The Solver creates

Array(1)	An Answer Report
Array(2)	A Population Report
Array(3)	A Solutions Report

If the Solver could not find a feasible solution (**SolverSolve** returns 5), you can produce either version of the Feasibility Report, or a Scaling Report:

If ReportArray is	**The Solver creates**
Array(1)	A Feasibility Report
Array(2)	A Feasibility-Bounds Report

If you are using Premium Solver Platform for Mac and a field-installable Solver engine, it may produce some or all of the reports mentioned above and/or its own custom reports. To determine what you should use for the **ReportArray** argument, solve a problem interactively with this Solver engine, and examine the Reports list box in the Solver Results dialog. Then use the ordinal position of the report you want:

If ReportArray is	**The Solver creates**
Array(1)	The first report listed
Array(2)	The second report listed (and so on)

ReportDesc is an array of character strings that allows you to select reports by their names, rather than their ordinal positions in the Reports list. For example, you can select an Answer Report with Array ("Answer"), or both the Answer Report and the Sensitivity Report with Array ("Answer", "Sensitivity"). The possible strings are:

"Answer"	Answer Report	
"Sensitivity"	Sensitivity Report	
"Limits"	Limits Report	
"Solutions"	Solutions Report	
"Population"	Population Report	"Feasibility"
Feasibility Report (full version)		
"Feasibility-Bounds"	Feasibility Report (w/o bounds)	

The report names you can include in the array depend on the currently selected Solver engine and the integer value returned by **SolverSolve**, as described above.

OutlineReports is a logical value corresponding to the Outline Reports check box. If TRUE, any reports you select will be produced in outlined format, and comments (if any) associated with each block of variables and constraints will be included in the report; if it is FALSE, the reports will be produced in "regular" format.

SolverFinishDialog

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

VBA Syntax

SolverFinishDialog (KeepFinal:=, ReportArray:=, ReportDesc:=, OutlineReports:=)

For an explanation of the arguments of this function, 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 chapter. Values of the TypeNum:= argument from 1 to 18 are supported by the standard Excel Solver.

SolverGet is provided for compatibility with the standard Excel Solver and earlier versions of Premium Solver Platform for Mac. For programmatic control of new features and options included in Version 5.0 or later of Premium Solver Platform for Mac, see the dialog-specific "Get" functions in the sections "Solver Model VBA Functions" and "Premium VBA Functions."

VBA 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 the #N/A error value 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 Changing Cells box (in the Premium Solvers, 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 hand sides of the constraints as text
7	An array of numbers corresponding to the relations between the left and right hand sides of the constraints: 1 = <= 2 = = 3 = >= 4 = int 5 = bin 6 = dif 7 = soc 8 = src
8	An array of the right hand 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	In the standard Solver: TRUE if the Assume Linear Model check box is selected; FALSE otherwise. In the Premium Solvers: TRUE if the linear Simplex or LP/Quadratic Solver is selected; FALSE if any other Solver is selected

```
14        TRUE if the Show Iteration Result check box is selected;
          FALSE otherwise

15        TRUE if the Use Automatic Scaling check box is selected;
          FALSE 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
```

The following settings are supported by Premium Solver Platform For Mac:

```
TypeNum   Returns

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

20        TRUE if the Make Unconstrained Variables Non-Negative check
box is selected; FALSE
          otherwise

21        The Integer Cutoff value (as a decimal number)

22        TRUE if the Bypass Solver Reports check box is selected;
          FALSE otherwise

23        An array of the entries in the Variables list box as text

24        A number corresponding to the Solver engine dropdown list
          for the currently selected Solver engine:
          1  =  Nonlinear GRG Solver
          2  =  Simplex or LP/Quadratic Solver
          3  =  Evolutionary Solver

          4  =  SOCP Barrier Solver
          In Premium Solver Platform for Mac, other values may be
returned
          for field-installable Solver engines

25        The Pivot Tolerance (as a decimal number) in the
          Simplex, LP/Quadratic, and Large-Scale LP Solvers

26        The Reduced Cost Tolerance (as a decimal number) in the
          Simplex, LP/Quadratic, and Large-Scale LP Solvers

27        The Coefficient Tolerance (as a decimal number) in the
          Large-Scale LP Solver

28        The Solution Tolerance (as a decimal number) in the
          Large-Scale LP Solver

29        TRUE if the Estimates option in the GRG Solver is set to
          Tangent; FALSE if the Estimates option is set to Quadratic

30        A number corresponding to the type of Scaling in the
          Large-Scale LP Solver:
          1  =  None
          2  =  Row Only
          3  =  Row & Col
```

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 Solver... from the Tools menu, choosing the Options button from the Solver Parameters dialog box, and choosing the Load Model... button in the Solver Options dialog box. Loads Solver model specifications that you have previously saved on the worksheet.

VBA Syntax

SolverLoad (LoadArea:=, Merge:=)

LoadArea is a reference to a range of cells from which you want to load a complete model specification. It can be used on any worksheet.

Merge is a logical value corresponding to either the Merge button or the Replace button in the dialog that appears after you select the **LoadArea** reference and click OK. If it is TRUE, the variable cell selections and constraints from the **LoadArea** are merged with the currently defined variables and constraints. If FALSE, the current model specifications and options are erased (equivalent to a call to the **SolverReset** function) before the new specifications are loaded.

The first cell in **LoadArea** contains a formula for the Set Cell edit box; the second cell contains a formula for the changing cells; subsequent cells contain additional variable selections and constraints in the form of logical formulas. The final cells optionally contain an array of Solver option values.

SolverOk

Equivalent to choosing Solver... from the Tools menu and specifying options in the Solver Parameters dialog. Specifies basic Solver options. The dialog box will *not* be displayed.

VBA Syntax

SolverOk (SetCell:=, MaxMinVal:=, Valueof:=, ByChange:=, Engine:=, EngineDesc:=)

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**.

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.

ByChange indicates the changing cells (decision variables), as entered in the By Changing Variable Cells edit box. **ByChange** must be a cell reference (usually a cell range or multiple reference) on the active worksheet. In Premium Solver Platform For Mac, you can add more changing cell references using Form 2 of the **SolverAdd** function.

Engine corresponds to the engine dropdown list in the Solver Parameters dialog. See the **EngineDesc** argument for an alternative way of selecting the Solver "engine."

Engine	Solver engine specified
1	Nonlinear GRG Solver
2	Simplex or LP/Quadratic Solver
3	Evolutionary Solver
4	SOCP Barrier Solver

In Premium Solver Platform for Mac, other values for **Engine** may be specified to select field-installable Solver engines. However, these values depend on the ordinal position of the Solver engine in the dropdown list, which may change when additional Solver engines are installed.

EngineDesc, which is supported only by Solver Platform for Mac, provides an alternative way to select the Solver engine from the dropdown list in the Solver Parameters dialog. **EngineDesc** allows you to select a Solver engine by name rather than by ordinal position in the list:

EngineDesc	Solver engine specified
"Standard GRG Nonlinear"	Nonlinear GRG Solver
"Standard Simplex LP"	Simplex LP Solver
"Standard LP/Quadratic"	LP/Quadratic Solver
"Standard Evolutionary"	Evolutionary Solver
"Standard SOCP Barrier"	SOCP Barrier Solver
"KNITRO Solver"	KNITRO Solver
"Large-Scale GRG Solver"	Large-Scale GRG Solver
"Large-Scale LP Solver"	Large-Scale LP Solver
"MOSEK Solver"	MOSEK Solver Engine
"Gurobi LP/MIP Solver"	Gurobi Solver Engine

SolverOkDialog

Equivalent to choosing Solver... from the Tools menu and specifying options in the Solver Parameters dialog. The Solver Parameters dialog box *will* be displayed, and the user will be able to change the options you initially specify.

VBA Syntax

SolverOkDialog (SetCell:=, MaxMinVal:=, Valueof:=, ByChange:=, Engine:=, EngineDesc:=)

For an explanation of the arguments of this function, see SolverOk.

SolverOptions

Equivalent to choosing Solver... from the Tools menu, then choosing the Options button in the Solver Parameters dialog box. Specifies Solver algorithmic options. Arguments supported by the standard Excel Solver include **MaxTime**, **Iterations**, **Precision**, **AssumeLinear**, **StepThru**, **Estimates**, **Derivatives**, **SearchOption**, **IntTolerance**, **Scaling**, **Convergence** and **AssumeNonNeg**.

SolverOptions is provided for compatibility with the standard Excel Solver and early versions of Premium Solver Platform for Mac. For programmatic control of new features and options included in Premium Solver Platform for Mac, see the functions in the sections "Solver Model VBA Functions" and "Premium VBA Functions."

VBA Syntax

SolverOptions (MaxTime:=, Iterations:=, Precision:=, AssumeLinear:=, StepThru:=, Estimates:=, Derivatives:=, SearchOption:=, IntTolerance:=, Scaling:=, Convergence:=, AssumeNonNeg:=, IntCutoff:=, BypassReports:=, PivotTol:=, ReducedTol:=, CoeffTol:=, SolutionTol:=, Crash:=, ScalingOption:=)

The arguments correspond to the options in the Solver Options dialog box. If an argument is omitted, the Solver maintains the current setting for that option. If any of the arguments are of the wrong type, the function returns the #N/A error value. 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 return value 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 logical value corresponding to the Assume Linear Model check box. This argument is included for compatibility with the standard Microsoft Excel Solver. It is ignored by Premium Solver Platform for Mac, which use the **Engine** or **EngineDesc** argument of **SolverOk** or **SolverOkDialog** instead.

StepThru is a logical value corresponding to the Show Iteration Results check box. If TRUE, Solver pauses at each trial solution; if FALSE it does not. If you have supplied **SolverSolve** with a valid VBA function argument, your function 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 logical value corresponding to the Use Automatic Scaling check box. If TRUE, then Solver rescales the objective and constraints internally to similar orders of magnitude. If FALSE, Solver uses values directly from the worksheet. In early Excel versions for Windows, this option affects the nonlinear GRG Solver only; in Excel 97, 2000, XP, 2003 and 2007 and Premium Solver Platform for Mac, this option affects all Solver engines.

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

AssumeNonNeg is a logical value corresponding to the Make Unconstrained Variables Non-Negative check box. If TRUE, Solver supplies a lower bound of zero for all variables without explicit lower bounds in the Constraint list box. If FALSE, no action is taken.

IntCutoff is a number corresponding to the Integer Cutoff edit box. This argument applies only if integer constraints have been defined.

BypassReports is currently not used.

PivotTol is currently not used.

ReducedTol is currently not used.

CoeffTol is currently not used.

SolutionTol is currently not used.

Crash is currently not used.

ScalingOption is currently not used.

SolverReset

Equivalent to choosing Solver... from the Tools menu 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 on the Solver Options, Limit Options and Integer Options dialog tabs to their defaults. The SolverReset function may be automatically performed when you call SolverLoad.

VBA Syntax

SolverReset

SolverSave

Equivalent to choosing Solver... from the Tools menu, choosing the Options button from the Solver Parameters dialog box, and choosing the Save Model... button in the Solver Options dialog box. Saves the model specifications on the worksheet.

VBA Syntax

SolverSave (SaveArea:=)

SaveArea is a reference to a range of cells or to the topmost cell in a column of cells where you want to save the current model's specifications.

The first cell in **SaveArea** contains a formula for the Set Cell edit box; the second cell contains a formula for the changing cells; subsequent cells contain additional variable selections and constraints in the form of logical formulas. The final cells optionally contain an array of Solver option values.

Remarks

If you specify only one cell for **SaveArea**, the area is extended downwards for as many cells as are required to hold the model specifications.

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

SolverSolve

Equivalent to choosing Solver... from the Tools menu 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.

VBA Syntax

SolverSolve (UserFinish:=, ShowRef:=)

UserFinish is a logical value specifying whether to show the standard Solver Results dialog box.

If **UserFinish** is TRUE, SolverSolve returns its integer value without displaying anything. Your VBA code should decide what action to take (for example, by examining the return value or presenting its own dialog box); If **UserFinish** is FALSE or omitted, 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 currently not used.

Remarks

If a Solver problem has not been completely defined, **SolverSolve** returns the #N/A error value. Otherwise the Solver engine is started, and the problem specifications are passed to it. Note that your macro will immediately proceed to the next line of code, without waiting for the Solver to finish. Please keep this in mind as you write your macro.

Solver Model VBA Functions

The VBA functions in this section are available only in Premium Solver Platform for Mac. You can use these functions to programmatically use the Polymorphic Spreadsheet Interpreter to check your model for Gradients, Structure and Convexity, obtain model statistics, produce the Structure Report and Transformation Report, determine whether and how the Interpreter will be used when you call SolverSolve, and control the Interpreter's advanced options.

SolverModel

Equivalent to choosing Solver... from the Tools menu, choosing the Model button in the Solver Parameters dialog, setting options in the Solver Model dialog, and clicking Close. Specifies options for the Polymorphic Spreadsheet Interpreter.

VBA Syntax

SolverModel (CheckFor:=, SolveTransformed:=, DesiredModel:=, Interactive:=, , Engines:=, Sparse:=, ActiveOnly:=)

The arguments correspond to the options in the Solver Model dialog box. If an argument is omitted, the Solver maintains the current setting for that option. If any of the arguments are of the wrong type, the function returns the #N/A error value. 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 return value indicates that all options were accepted.

CheckFor is a number corresponding to the option selected in the Check For option group:

CheckFor	Option Selected
1	Gradients
2	Structure
3	Convexity
4	Automatic

SolveTransformed is a logical value corresponding to the Transform Non-Smooth Model check box in the Model dialog. If TRUE, the Solver solves the Transformed problem when the SolverSolve function is called. If FALSE, the Solver solves the Original problem when the SolverSolve function is called.

DesiredModel is a number corresponding to the option selected in the Desired Model option group:

DesiredModel	Desired Model Type
1	Linear
2	Quadratic
3	Nonlinear

Interactive is currently not used.

Engines is currently not used.

Sparse is a logical value corresponding to the Sparse check box in the Advanced options group on the Options tab. If TRUE, the Polymorphic Spreadsheet Interpreter will operate internally in its own Sparse mode. If FALSE, the Interpreter operates in Dense mode.

ActiveOnly is a logical value corresponding to the Active Only check box in the Advanced options group on the Options tab. If TRUE, the Polymorphic Spreadsheet Interpreter will analyze objective and constraint function formulas only on the active sheet. If FALSE, the Interpreter analyze all objective and constraint function formulas in the workbook.

SolverModelCheck

Equivalent to choosing Solver... from the Tools menu, choosing the Model button in the Solver Parameters dialog box, and clicking the Check Model button in the Solver Model dialog. The type of analysis performed is determined by the current setting of the SolverModel CheckFor argument.

VBA Syntax

SolverModelCheck (Transformed:=)

Transformed is a logical value that corresponds to the Transform Non-Smooth Model check box. If TRUE, the Polymorphic Spreadsheet Interpreter checks the Transformed model. If FALSE, the Interpreter checks the Original model.

SolverModelGet

Returns Solver Model option settings for the current Solver problem on the specified sheet. These settings are entered in the Solver Model dialog.

VBA Syntax

SolverModelGet (TypeNum:=, SheetName:=)

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

```
TypeNum   Returns

1         A number corresponding to the Check For option: 1 for
           Gradients, 2 for Structure, or 3 for Convexity

2         TRUE if the Solve Transformed Problem check box is selected;
           FALSE otherwise
```

3	A number corresponding to the Desired Model option: 1 for Linear, 2 for Quadratic, 3 for Conic, 4 for Nonlinear, or 4 for Nonsmooth
4	TRUE if the Interactive Optimization check box is selected; FALSE otherwise
5	Currently not used.
6	TRUE if the Sparse check box is selected; FALSE otherwise
7	TRUE if the Active Only check box is selected; FALSE otherwise

Premium VBA Functions

To control most of the new features and options in Premium Solver Platform for Mac, you'll need to use these functions – notably, the **SolverEVOptions**, **SolverLimOptions** and **SolverIntOptions** functions If you want to write VBA code that can be used with both the standard Solver and Premium Solver Platform For Mac, you should use only functions in the section "Standard VBA Functions."

SolverEVGet

Returns Evolutionary Solver option settings for the current Solver problem on the specified sheet. These settings are entered in the Solver Options dialog when the Evolutionary Solver is selected in the Solver Engine dropdown list.

VBA Syntax

SolverEVGet (TypeNum:=, SheetName:=)

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

TypeNum	Returns
1	The Max Time value (as a number in seconds)
2	The Iterations value (max number of iterations)
3	The Precision value (as a decimal number)
4	The Convergence value (as a decimal number)
5	The Population Size value (as a decimal number)
6	The Mutation Rate value (as a decimal number)
7	TRUE if the Require Bounds on Variables check box is selected; FALSE otherwise
8	TRUE if the Show Iteration Result check box is selected; FALSE otherwise
9	TRUE if the Use Automatic Scaling check box is selected; FALSE otherwise
10	TRUE if the Assume Non-Negative check box is selected; FALSE otherwise
11	TRUE if the Bypass Solver Reports check box is selected; FALSE otherwise

```
12        The Random Seed value (as a decimal number)

13        A number corresponding to the Local Search option:  1 for
          Randomized Local Search, 2 for Deterministic Pattern Search,
          3 for Gradient Local Search, or 4 for Automatic Choice

14        TRUE if the Fix Nonsmooth Variables check box is selected;
          FALSE otherwise
```

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.

SolverEVOptions

Equivalent to choosing Solver... from the Tools menu and then choosing the Options button in the Solver Parameters dialog box when the Evolutionary Solver is selected in the Solver Engine dropdown list. Specifies options for the Evolutionary Solver.

VBA Syntax

SolverEVOptions (MaxTime:=, Iterations:=, Precision:=, Convergence:=, PopulationSize:=, MutationRate:=, RandomSeed:=, RequireBounds:=, StepThru:=, Scaling:=, AssumeNonNeg:=, BypassReports:=, LocalSearch:= FixNonSmooth:=)

The arguments correspond to the options in the Solver Options dialog box. If an argument is omitted, the Solver maintains the current setting for that option. If any of the arguments are of the wrong type, the function returns the #N/A error value. 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 return value 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.

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

PopulationSize must be an integer greater than or equal to zero. It corresponds to the Population Size edit box.

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

RandomSeed must be an integer greater than zero. It corresponds to the Random Seed edit box.

RequireBounds is a logical value corresponding to the Require Bounds on Variables check box. If TRUE, the Evolutionary Solver will return immediately from a call to the **SolverSolve** function with a value of 18 if any of the variables do not have both lower and upper bounds defined. If FALSE, the Evolutionary Solver will attempt to solve the problem without bounds on all of the variables.

StepThru is a logical value corresponding to the Show Iteration Results check box. If TRUE, Solver pauses at each trial solution; if FALSE it does not. If you have supplied **SolverSolve** with a valid VBA function, your function will be called each

time Solver pauses; otherwise the standard Show Trial Solution dialog box will appear.

Scaling is a logical value corresponding to the Use Automatic Scaling check box. If TRUE, then Solver rescales the objective and constraints internally to similar orders of magnitude. If FALSE, Solver uses values directly from the worksheet.

AssumeNonNeg is a logical value corresponding to the Assume Non-Negative check box. If TRUE, Solver supplies a lower bound of zero for all variables without explicit lower bounds in the Constraint list box. If FALSE, no action is taken.

BypassReports is a logical value corresponding to the Bypass Solver Reports check box. If TRUE, the Solver will skip preparing the information needed to create Solver Reports. If FALSE, the Solver will prepare for the reports. For large models, bypassing the Solver Reports can speed up the solution considerably.

LocalSearch is a number corresponding to the option button selected in the Local Search option group:

LocalSearch	Local Search Strategy
1	Randomized Local Search
2	Deterministic Pattern Search
3	Gradient Local Search
4	Automatic Choice

FixNonSmooth is a logical value corresponding to the Fix Nonsmooth Variables check box. If TRUE, the Solver will fix the non-smooth variables to their current values during each local search, and allow only smooth and linear variables to be varied. If FALSE, the Solver will allow all of the variables to be varied.

SolverGRGGet

Returns GRG Solver option settings for the current Solver problem on the specified sheet. These settings are entered in the Solver Options dialog when the GRG Solver is selected in the Solver Engine dropdown list.

VBA Syntax

SolverGRGGet (TypeNum:=, SheetName:=)

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

TypeNum	Returns
1	The Max Time value (as a number in seconds)
2	The Iterations value (max number of iterations)
3	The Precision value (as a decimal number)
4	The Convergence value (as a decimal number)
5	TRUE if the Show Iteration Result check box is selected; FALSE otherwise
6	TRUE if the Use Automatic Scaling check box is selected; FALSE otherwise
7	TRUE if the Assume Non-Negative check box is selected; FALSE otherwise
8	TRUE if the Bypass Solver Reports check box is selected; FALSE otherwise

```
9         TRUE if the Recognize Linear Variables check box is selected;
          FALSE otherwise

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

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

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

13        The Population Size value (as a decimal number)

14        The Random Seed value (as a decimal number)

15        TRUE if the Multistart Search check box is selected; FALSE
          otherwise

16        TRUE if the Topographic Search check box is selected; FALSE
          otherwise

17        TRUE if Require Bounds on Variables check box is selected;
          FALSE otherwise
```

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.

SolverGRGOptions

Equivalent to choosing Solver... from the Tools menu and then choosing the Options button in the Solver Parameters dialog box when the GRG Nonlinear Solver is selected in the Solver Engines dropdown list. Specifies options for the GRG Solver.

VBA Syntax

SolverGRGOptions (MaxTime:=, Iterations:=, Precision:=, Convergence:=, PopulationSize:=, RandomSeed:=, StepThru:=, Scaling:=, ,AssumeNonNeg:=, BypassReports:=, RecognizeLinear:=, MultiStart:=, TopoSearch:=, RequireBounds:=, Estimates:=, Derivatives:=, SearchOption:=)

The arguments correspond to the options in the Solver Options dialog box. If an argument is omitted, the Solver maintains the current setting for that option. If any of the arguments are of the wrong type, the function returns the #N/A error value. 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 return value 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.

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

PopulationSize must be an integer greater than or equal to zero. It corresponds to the Population Size edit box.

RandomSeed must be an integer greater than zero. It corresponds to the Random Seed edit box.

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

Scaling is a logical value corresponding to the Use Automatic Scaling check box. If TRUE, then Solver rescales the objective and constraints internally to similar orders of magnitude. If FALSE, Solver uses values directly from the worksheet.

AssumeNonNeg is a logical value corresponding to the Assume Non-Negative check box. If TRUE, Solver supplies a lower bound of zero for all variables without explicit lower bounds in the Constraint list box. If FALSE, no action is taken.

BypassReports is a logical value corresponding to the Bypass Solver Reports check box. If TRUE, the Solver will skip preparing the information needed to create Solver Reports. If FALSE, the Solver will prepare for the reports. For large models, bypassing the Solver Reports can speed up the solution considerably.

RecognizeLinear is a logical value corresponding to the Recognize Linear Variables check box. If TRUE, the Solver will recognize variables whose partial derivatives are not changing during the solution process, and assume that they occur linearly in the problem. If FALSE, the Solver will not make any assumptions about such variables. See the chapter "Solver Options" for a further discussion of this option.

MultiStart is a logical value corresponding to the Multistart Search check box. If TRUE, the Solver will use Multistart Search, in conjunction with the GRG Solver, to seek a globally optimal solution. If FALSE, the GRG Solver alone will be used to search for a locally optimal solution.

TopoSearch is a logical value corresponding to the Topographic Search check box. If TRUE, and if Multistart Search is selected, the Solver will construct a topography from the randomly sampled initial points, and use it to guide the search process.

RequireBounds is a logical value corresponding to the Require Bounds on Variables check box. If TRUE, the Solver will return immediately from a call to the **SolverSolve** function with a value of 18 if any of the variables do not have both lower and upper bounds defined. If FALSE, then Multistart Search (if selected) will attempt to find a globally optimal solution without bounds on all of the variables.

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.

SolverIntGet

Returns integer (Branch & Bound) option settings for the current Solver problem on the specified sheet. These settings are entered on the Integer Options dialog tab for any of the Solver engines.

VBA Syntax

SolverIntGet (TypeNum:=, SheetName:=)

TypeNum is a number specifying the type of information you want. The following settings are specified on the Integer Options dialog tab box.

TypeNum	Returns
1	The Max Subproblems value (as a decimal number)
2	The Max Integer Sols value (as a decimal number)
3	The Integer Tolerance value (as a decimal number)
4	The Integer Cutoff value (as a decimal number)
5	TRUE if the Solve Without Integer Constraints check box is selected; FALSE otherwise
6	1 if Preprocessing is set to Automatic, 2 if it is set to None, and 3 if it is set to Aggressive.
7	1 if Cuts is set to Automatic, 2 if it is set to None, and 3 if it is set to Aggressive.
8	1 if Heuristics is set to Automatic, 2 if it is set to None, and 3 if it is set to Aggressive.

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.

SolverIntOptions

Equivalent to choosing Solver... from the Tools menu, choosing the Options button in the Solver Parameters dialog box, then choosing the Integer Options button in the Solver Options dialog. Specifies options for the integer (Branch & Bound) Solver.

VBA Syntax

SolverIntOptions (MaxSubproblems:=, MaxIntegerSols:=, IntTolerance:=, IntCutoff:=, SolveWithout:=,PreProcessing:=. Cuts:=, Heuristics:=)

The arguments correspond to the options on the Integer Options dialog tab. The first five options are common to both the Simplex LP and LP/Quadratic Solvers; the next ten options are specific to the Simplex LP Solver; and the remaining options are specific to the LP/Quadratic Solver. If an argument is omitted, the Solver maintains the current setting for that option. If any of the arguments are of the wrong type, the function returns the #N/A error value. 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 return value indicates that all options were accepted.

MaxSubproblems must be an integer greater than zero. It corresponds to the Max Subproblems edit box.

MaxIntegerSols must be an integer greater than zero. It corresponds to the Max Integer Sols (Solutions) edit box.

IntTolerance is a number between zero and one, corresponding to the Tolerance edit box.

IntCutoff is a number (any value is possible) corresponding to the Integer Cutoff edit box.

SolveWithout is a logical value corresponding to the Solve Without Integer Constraints check box. If TRUE, the Solver ignores any integer constraints and solves the "relaxation" of the mixed-integer programming problem. If FALSE, the Solver uses the integer constraints in solving the problem.

PreProcessing is a value of 1,2 or 3, corresponding to the choices “Automatic”, “None” or “Aggressive”.

Cuts is a value of 1,2 or 3, corresponding to the choices “Automatic”, “None” or “Aggressive”.

Heuristics is a value of 1,2 or 3, corresponding to the choices “Automatic”, “None” or “Aggressive”.

SolverLimGet

Returns Limit Option settings for the Evolutionary Solver problem (if any) defined on the specified sheet. These settings are entered on the Limit Options dialog tab for the Evolutionary Solver.

VBA Syntax

SolverLimGet (TypeNum:=, SheetName:=)

TypeNum is a number specifying the type of information you want. The following settings are specified on the Limit Options dialog tab.

TypeNum	Returns
1	The Max Subproblems value (as a decimal number)
2	The Max Feasible Sols value (as a decimal number)
3	The Tolerance value (as a decimal number)
4	The Max Time w/o Improvement value (as a decimal number)
5	TRUE if the Solve Without Integer Constraints check box is selected; FALSE otherwise

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.

SolverLimOptions

Equivalent to choosing Solver... from the Tools menu, choosing the Options button in the Solver Parameters dialog box when the Evolutionary Solver is selected in the Solver Engine dropdown list, then choosing the Limit Options button in the Solver Options dialog. Specifies Limit Options for the Evolutionary Solver.

VBA Syntax

SolverLimOptions (MaxSubproblems:=, MaxFeasibleSols:=, Tolerance:=, MaxTimeNoImp:=, SolveWithout:=)

The arguments correspond to the options on the Limit Options dialog tab. If an argument is omitted, the Solver maintains the current setting for that option. If any of the arguments are of the wrong type, the function returns the #N/A error value. 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 return value indicates that all options were accepted.

MaxSubproblems must be an integer greater than zero. It corresponds to the Max Subproblems edit box.

MaxFeasibleSols must be an integer greater than zero. It corresponds to the Max Feasible Sols (Solutions) edit box.

Tolerance is a number between zero and one, corresponding to the Tolerance edit box. This argument works in conjunction with the MaxTimeNoImp argument below.

MaxTimeNoImp is a number corresponding to the Max Time w/o Improvement edit box. This argument works in conjunction with the Tolerance argument above to determine when the Evolutionary Solver will stop with the message "Solver cannot improve the current solution."

SolveWithout is a logical value corresponding to the Solve Without Integer Constraints check box. If TRUE, the Evolutionary Solver ignores any integer constraints and solves the "relaxation" of the problem. If FALSE, the Solver uses the integer constraints in solving the problem.

SolverLPGet

Returns Simplex LP or LP/Quadratic Solver option settings for the current Solver problem on the specified sheet. These settings are entered in the Solver Options dialog when the Simplex LP or LP/Quadratic Solver is selected in the Solver Engine dropdown list.

VBA Syntax

SolverLPGet (TypeNum:=, SheetName:=)

TypeNum is a number specifying the type of information you want. The following settings are specified in the Simplex LP or LP/Quadratic Solver Options dialog box.

TypeNum	Returns
1	The Max Time value (as a number in seconds)
2	The Iterations value (max number of iterations)
3	The Precision value (as a decimal number)
4	Not used.
5	Not used.
6	TRUE if the Show Iteration Result check box is selected; FALSE otherwise
7	TRUE if the Use Automatic Scaling check box is selected; FALSE otherwise
8	TRUE if the Assume Non-Negative check box is selected; FALSE otherwise
9	Not used.
10	A number corresponding to the Derivatives group selection: 1 = Forward 2 = Central
11	The LP/Quadratic Primal Tolerance (as a decimal number)
12	The LP/Quadratic Dual Tolerance (as a decimal number)
13	TRUE if the Do Presolve check box is selected; FALSE otherwise.

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.

SolverLPOptions

Equivalent to choosing Solver... from the Tools menu and then choosing the Options button in the Solver Parameters dialog box when the Simplex LP or LP/Quadratic Solver is selected in the Solver Engine dropdown list. Specifies options for the Simplex LP and LP/Quadratic Solvers.

VBA Syntax

SolverLPOptions (MaxTime:=, Iterations:=, Precision:=, PivotTol:=, ReducedTol:=, StepThru:=, Scaling:=, AssumeNonNeg:=, BypassReports:=, Derivatives:=, PrimalTolerance:=, DualTolerance:=, Presolve:=)

The arguments correspond to the options in the Solver Options dialog box. The PivotTol and ReducedTol options are available only for the Simplex LP Solver; the Derivatives, PrimalTolerance, DualTolerance, and Presolve options are available only for the LP/Quadratic Solver. If an argument is omitted, the Solver maintains the current setting for that option. If any of the arguments are of the wrong type, the function returns the #N/A error value. 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 return value 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.

PivotTol is currently not used..

ReducedTol is currently not used..

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

Scaling is a logical value corresponding to the Use Automatic Scaling check box. If TRUE, then Solver rescales the objective and constraints internally to similar orders of magnitude. If FALSE, Solver uses values directly from the worksheet.

AssumeNonNeg is a logical value corresponding to the Assume Non-Negative check box. If TRUE, Solver supplies a lower bound of zero for all variables without explicit lower bounds in the Constraint list box. If FALSE, no action is taken.

BypassReports is currently not used.

Derivatives is the number 1 or 2 and corresponds to the Derivatives option group for the LP/Quadratic Solver: 1 for Forward and 2 for Central.

PrimalTolerance is a number between zero and one, but not equal to zero or one. It corresponds to the Primal Tolerance edit box for the LP/Quadratic Solver.

DualTolerance is a number between zero and one, but not equal to zero or one. It corresponds to the Dual Tolerance edit box for the LP/Quadratic Solver.

Presolve is a logical value corresponding to the Do Presolve check box. If TRUE, the Solver performs a presolve step before starting the Simplex method that detects singleton rows and columns, removes fixed variables and redundant constraints, and tightens bounds. If FALSE, no action is taken.

SolverOkGet

Returns variable, constraint and objective selections and settings for the current Solver problem on the specified sheet. These settings are entered in the Solver Parameters dialog.

VBA Syntax

SolverOkGet (TypeNum:=, SheetName:=)

TypeNum is a number specifying the type of information you want:

```
TypeNum   Returns

1         The reference in the Set Cell box, or the #N/A error value 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 Changing Cells box (in the Premium
          Solvers, 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 hand sides of the constraints as text

7         An array of numbers corresponding to the relations
          between the left and right hand sides of the constraints:
          1  =  <=
          2  =  =
          3  =  >=
          4  =  int
          5  =  bin
          6  =  dif
          7  =  soc
          8  =  src

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

9         An array of the entries in the Variables list box as text

10        A number corresponding to the Solver engine dropdown list
          for the currently selected Solver engine:
          1  =  Nonlinear GRG Solver
          2  =  Simplex or LP/Quadratic Solver
          3  =  Evolutionary Solver
          4  =  SOCP Barrier Solver
          In Premium Solver Platform for Mac, other values may be
returned
          for field-installable Solver engines
```

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.