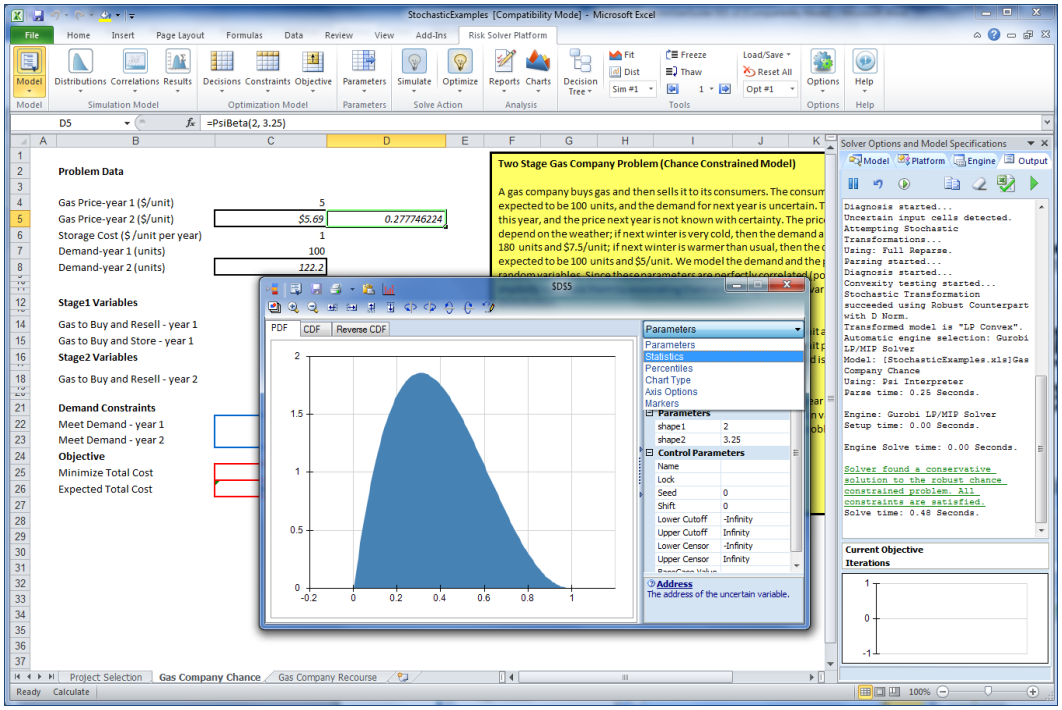
Version 11.5 **For Use With Excel 2003-2010**

FRONTLINE solvers

User Guide



Acknowledgements

Thanks to Dan Fylstra and the Frontline Systems development team for a 20-year cumulative effort to build the best possible optimization and simulation software for Microsoft Excel. Thanks to Frontline's customers who have built many thousands of successful applications, and have given us many suggestions for improvements.

Risk Solver Platform has benefited from reviews, critiques, and suggestions from several risk analysis experts:

- Sam Savage (Stanford Univ. and AnalyCorp Inc.) for Probability Management concepts including SIPs, SLURPs, DISTs, and Certified Distributions.
- Sam Sugiyama (EC Risk USA & Europe LLC) for evaluation of advanced distributions, correlations, and alternate parameters for continuous distributions.
- Savvakis C. Savvides for global bounds, censor bounds, base case values, the Normal Skewed distribution and new risk measures.

How to Order

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

Contents

Start Here: V11.5 Essentials

Getting the Most from This User Guide

Installing the Software

Run the SolverSetup program to install the software – whether you are using Risk Solver Platform or any of its subsets, such as Premium Solver Platform. The chapter "Installation and Add-Ins" covers installation step-by-step, and explains how to activate and deactivate the Risk Solver Platform Excel add-in.

Upgrading from Earlier Versions

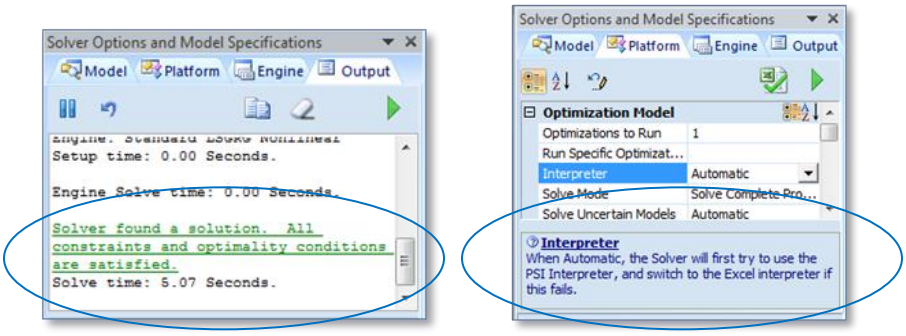
If you have our **V11.x, 10.x or V9.x software installed**, V11 will be installed into the same folder (recommended). If you have our **V8.x or earlier software installed**, V11 will be installed into a separate folder; we recommend uninstalling the earlier version. For more information and other options, see "Installation and Add-Ins."

Obtaining a License

Use Help – License Code on the Ribbon. V11.5 does not require a new license if you have a current license for V11.x. If you have a 10.x or earlier license, V11.x does have a new license manager, which makes obtaining a license much easier (you can activate a license over the Internet). However, license codes for versions prior to V11.x in your Solver.lic license file will be *ignored* in V11.x. See the chapter "Using Help, Licensing and Product Subsets" for details.

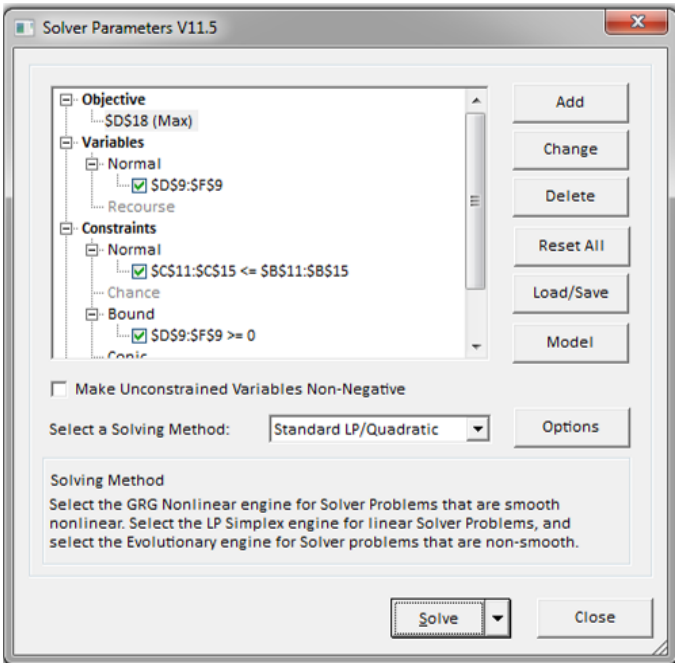
Getting Help Quickly

Click underlined topics. In V11.5 you can get quick online Help for each Solver result message and error message in the Output pane, each Platform option and Solver Engine option, and each element of your simulation model and optimization model. See the chapter "Using Help, Licensing and Product Subsets" for details.



Using the Solver Parameters Dialog

Click **Premium Solver** on the **Add-Ins tab** to display the classic Solver Parameters dialog. You can **go back and forth freely** between the Solver Parameters dialog and the new, modeless Task Pane and Ribbon. Within a few minutes of use, most users find the Task Pane and Ribbon faster and easier to use.



Finding the Examples

Use Help – Examples on the Ribbon to open a workbook with a list of examples covering optimization, simulation, simulation optimization, stochastic optimization, and decision trees. See the chapter "Using Help, Licensing and Product Subsets" for details. Some of these examples are used and described in the **Examples** chapters.

Using Existing Models

Open your existing workbook, developed in any previous version of Risk Solver Platform, Premium Solver Platform, Risk Solver Pro, Premium Solver Pro, or the standard Excel Solver. Your model should appear in the Task Pane; just click the Optimize or Simulate button. **Read Automatic Mode and Solution Time** in "Risk Solver Platform Overview" to understand how this mode can impact solution time.

Using Existing VBA Macros

Standard Excel Solver macros such as SolverOK and SolverSolve should work as-is, provided that the option **Load V11 VBA Macros** is True in the Task Pane Platform tab General group, or checked in the Options dialog General tab.

Macros using the Object-Oriented API such as Problem.Solver.Optimize should work as-is, provided that you use Tools References in the VBA Editor to set or change) the reference to **Risk Solver Platform 11 Type Library**.

Using Large-Scale Solver Engines

Run the EngineSetup program to install any or all eight large-scale Solver Engines V11.5. The new Solver Engine(s) will appear in the dropdown list at the top of the Task Pane Engine tab. Use this dropdown to select an Engine for solving, or to set its options. Use **Help – License Code** to add a Solver Engine license.

Getting Started with Tutorials

To quickly gain a good grasp of Risk Solver Platform's capabilities, work through the **Examples** chapters on Conventional Optimization, Simulation and Risk Analysis, Stochastic Optimization, and other topics.

Getting and Interpreting Results

Learn how to interpret Risk Solver Platform's result messages, error messages, reports and charts, and how to run multiple parameterized optimizations and simulations in the **Getting Results** chapters.

Automating Your Model with VBA

Use the code examples in the **Automating Your Model with VBA** chapters to get started with creation of your own custom application that uses optimization or simulation "behind the scenes."

Mastering Optimization and Simulation Concepts

Go from beginner to expert, and learn how to fully exploit the software by reading the **Mastering Concepts** chapters, and the Frontline Solvers Reference Guide.

Software License and Limited Warranty

Standalone License: Upon Frontline's receipt of payment from Licensee of the applicable Fee for a single-Use license ("Standalone License"), Licensee may Use the Software for a Permanent Term on one computer (the "PC"), and Frontline 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. If the PC fails in a manner such that Use is no longer possible, Frontline will provide Licensee with a new license code, enabling Use on a repaired or replaced PC, 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 in writing by Frontline in its sole discretion.

Flexible Use License: Upon Frontline's receipt of payment from Licensee of the applicable Fee for a multi-Use license ("Flexible Use License"), Licensee may Use the Software for a Permanent Term on a group of several computers as provided in this section, and Frontline will provide Licensee with a license code enabling such Use. The Software may be stored on one or more computers, servers or storage devices interconnected by any networking technology that supports the TCP/IP protocol (a "Network"), copied into the memory of, and Used on, any of the computers on the Network, provided that only one Use occurs at any given time, for each Flexible Use License purchased by Licensee. Frontline will provide to Licensee (under separate license) and Licensee must install and run License Server software ("LSS") on one of the computers on the Network (the "LS"); other computers will temporarily obtain the right to Use the Software from the LS. If the LS fails in a manner such that the LSS cannot be run, Frontline will provide Licensee with a new license code, enabling Use on a repaired or replaced LS, at no charge. A Flexible Use License 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 LSS will no longer be run on the first LS, and (iii) Licensee pays a license transfer fee, unless such fee is waived by Frontline in its sole discretion.

"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 more than one model at the same time, whether on one computer or multiple computers, requires more than one Standalone or Flexible Use License. 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.

Other License Restrictions: The Software includes license control features that may write encoded information about the license type and term to the PC or LS hard disk; Licensee agrees that it will not attempt to alter or circumvent such license control features. This License 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 acknowledges that the Software and its structure, organization, and source code constitute valuable Intellectual Property of Frontline and/or its suppliers and Licensee agrees that it shall not, nor shall it permit, assist or encourage any third party to: (a) copy, modify adapt, alter, translate or create derivative works from the Software; (b) merge the Software into any other software or use the Software to develop any application or program having the same primary function as the Software; (c) sublicense, distribute, sell, use for service bureau use, lease, rent, loan, or otherwise transfer the Software; (d) "share" use of the Software with anyone else; (e) make the Software available over the Internet, a company or institutional intranet, or any similar networking technology, except as explicitly provided in the case of a Flexible Use License; (f) reverse compile, reverse engineer, decompile, disassemble, or otherwise attempt to derive the source code for the Software; or (g) otherwise exercise any rights in or to the Software, except as permitted in this Section.

U.S. Government: The Software is provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Contractor/manufacturer is Frontline Systems, Inc., P.O. Box 4288, Incline Village, NV 89450.

2. ANNUAL SUPPORT.

Limited warranty: If Licensee purchases an "Annual Support Contract" from Frontline, then Frontline warrants, during the term of such Annual Support Contract ("Support Term"), that the Software covered by the Annual Support Contract will perform substantially as described in the User Guide published by Frontline in connection with the Software, as such may be amended from time to time, when it is properly used as described in the User Guide, provided, however, that Frontline does not warrant that the Software will be error-free in all circumstances. During the Support Term, Frontline shall make reasonable commercial efforts to correct, or devise workarounds for, any Software errors (failures to perform as so described) reported by Licensee, and to timely provide such corrections or workarounds to Licensee.

Disclaimer of Other Warranties: IF THE SOFTWARE IS COVERED BY AN ANNUAL SUPPORT CONTRACT, THE LIMITED WARRANTY IN THIS SECTION 2 SHALL CONSTITUTE FRONTLINE'S ENTIRE LIABILITY IN CONTRACT, TORT AND OTHERWISE, AND LICENSEE'S EXCLUSIVE REMEDY UNDER THIS LIMITED WARRANTY. IF THE SOFTWARE IS NOT COVERED BY A VALID ANNUAL SUPPORT CONTRACT, OR IF LICENSEE PERMITS THE ANNUAL SUPPORT CONTRACT ASSOCIATED WITH THE SOFTWARE TO EXPIRE, THE DISCLAIMERS SET FORTH IN SECTION 3 SHALL APPLY.

3. WARRANTY DISCLAIMER.

EXCEPT AS PROVIDED IN SECTION 2 ABOVE, THE SOFTWARE IS PROVIDED "AS IS" AND "WHERE IS" WITHOUT WARRANTY OF ANY KIND; FRONTLINE 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 OR ANY WARRANTIES ARISING FROM COURSE OF DEALING OR COURSE OF PERFORMANCE AND THE SAME ARE HEREBY EXPRESSLY DISCLAIMED TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW. WITHOUT LIMITING THE FOREGOING, FRONTLINE DOES NOT REPRESENT, WARRANTY OR GUARANTEE THAT THE SOFTWARE WILL BE ERROR-FREE, UNINTERRUPTED, SECURE, OR MEET LICENSEES' EXPECTATIONS. FRONTLINE DOES NOT MAKE ANY WARRANTY REGARDING THE SOFTWARE'S RESULTS OF USE OR THAT FRONTLINE WILL CORRECT ALL ERRORS. THE LIMITED WARRANTY SET FORTH IN SECTION 2 IS EXCLUSIVE AND FRONTLINE MAKES NO OTHER EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS WITH RESPECT TO THE SOFTWARE, ANNUAL SUPPORT AND/OR OTHER SERVICES PROVIDED IN CONNECTION WITH THIS LICENSE, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NONINFRINGEMENT.

4. LIMITATION OF LIABILITY.

IN NO EVENT SHALL FRONTLINE OR ITS SUPPLIERS HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION ANY LOST DATA, LOST PROFITS OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES), HOWEVER CAUSED AND UNDER 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. NOTWITHSTANDING ANYTHING HEREIN TO THE CONTRARY, IN NO EVENT SHALL FRONTLINE'S TOTAL CUMULATIVE LIABILITY IN CONNECTION WITH THIS LICENSE, THE SOFTWARE, AND ANY SUPPORT CONTRACTS PROVIDED BY FRONTLINE TO LICENSEE HEREUNDER, WHETHER IN CONTRACT OR TORT OR OTHERWISE EXCEED THE PRICE OF ONE STANDALONE LICENSE. LICENSEE ACKNOWLEDGES THAT THIS ARRANGEMENT REFLECTS THE ALLOCATION OF RISK SET FORTH IN THIS LICENSE AND THAT FRONTLINE WOULD NOT ENTER INTO THIS LICENSE WITHOUT THESE LIMITATIONS ON ITS LIABILITY. LICENSEE

ACKNOWLEDGES THAT THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

REGARDLESS OF WHETHER LICENSEE PURCHASES AN ANNUAL SUPPORT CONTRACT FROM FRONTLINE, LICENSEE UNDERSTANDS AND AGREES THAT ANY RESULTS OBTAINED THROUGH LICENSEE'S USE OF THE SOFTWARE ARE ENTIRELY DEPENDENT ON LICENSEE'S DESIGN AND IMPLEMENTATION OF ITS OWN OPTIMIZATION OR SIMULATION MODEL, FOR WHICH LICENSEE IS ENTIRELY RESPONSIBLE, EVEN IF LICENSEE RECEIVED ADVICE, REVIEW, OR ASSISTANCE ON MODELING FROM FRONTLINE.

5. TERM AND TERMINATION.

Term: The License shall become effective when Licensee first downloads, accepts delivery, installs or uses the Software, and shall continue: (i) in the case of an Evaluation License, for a limited term (such as 15 days) determined from time to time by Frontline in its sole discretion ("Limited Term"), (ii) in the case of Standalone License or Flexible Use License, for an unlimited term unless terminated for breach pursuant to this Section ("Permanent Term").

Termination: Frontline may terminate this License if Licensee breaches any material provision of this License and does not cure such breach (provided that such breach is capable of cure) within 30 days after Frontline provides Licensee with written notice thereof.

6. GENERAL PROVISIONS.

Proprietary Rights: The Software is licensed, not sold. The Software and all existing and future worldwide copyrights, trademarks, service marks, trade secrets, patents, patent applications, moral rights, contract rights, and other proprietary and intellectual property rights therein ("Intellectual Property"), are the exclusive property of Frontline and/or its licensors. All rights in and to the Software and Frontline's other Intellectual Property not expressly granted to Licensee in this License are reserved by Frontline. For the Large-Scale LP/QP Solver only: Source code is available, as part of an open source project, for portions of the Software; please contact Frontline for information if you want to obtain this source code.

Amendments: This License constitutes the complete and exclusive agreement between the parties relating to the subject matter hereof. It supersedes all other proposals, understandings and all other agreements, oral and written, between the parties relating to this subject matter, including any purchase order of Licensee, any of its preprinted terms, or any terms and conditions attached to such purchase order.

Compliance with Laws: Licensee will not export or re-export the Software without all required United States and foreign government licenses.

Assignment: This License 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 License.

Governing Law: Any controversy, claim or dispute arising out of or relating to this License, shall be governed by the laws of the State of Nevada, other than such laws, rules, regulations and case law that would result in the application of the laws of a jurisdiction other than the State of Nevada.

Risk Solver Platform Overview

Risk Solver Platform and Subset Products

This Guide shows you how to use **Risk Solver Platform** – Frontline Systems "super-product" that combines all the capabilities of Premium Solver Platform for optimization, Risk Solver Pro for simulation, and several powerful methods for optimization of *uncertain* models. Risk Solver Platform **includes every feature** described in this User Guide, and in the Frontline Solvers Reference Guide.

Premium Solver Platform and Premium Solver Pro for optimization, and Risk Solver Pro for simulation and simulation-optimization are available as **subsets of Risk Solver Platform**. All these products use one set of program files, and share a common user interface; the product you 'experience' is determined by your license code.

This Guide uses the following icons – which appear as tabs on the Excel 2007 and 2010 Ribbon – to indicate that certain features are available in each subset product (if no icons appear, the feature being discussed is in all product subsets):

Risk Solver Platform and its subsets Premium Solver Platform and Premium Solver Pro are **fully compatible upgrades for the Solver** bundled with Microsoft Excel, which was developed by Frontline Systems for Microsoft. Your Excel Solver models and macros will work without changes, and the classical **Solver Parameters dialog** can be used whenever needed in Risk Solver Platform.

Premium Solver Platform

Premium Solver Platform is Frontline's most powerful product for *conventional* optimization. It includes the PSI Interpreter, five built-in Solvers (LP/Quadratic, SOCP Barrier, LSGRG Nonlinear, Interval Global, and Evolutionary), and it supports plug-in large-scale Solver Engines. It has no Monte Carlo simulation, simulation optimization, stochastic programming or robust optimization.

Premium Solver Pro

Premium Solver Pro is Frontline's basic upgrade for the standard Excel Solver, for *conventional* optimization. It includes a Simplex LP Solver, GRG Nonlinear Solver, Evolutionary Solver, Branch & Bound method for integer and alldifferent constraints, and the Multi-start method for global optimization.

Risk Solver Pro

Risk Solver Pro is Frontline's full-function product for Monte Carlo simulation. It includes all of the simulation functionality of Risk Solver Platform including the Psi Interpreter for simulations.

Importantly, Risk Solver Pro can be purchased in combination with our conventional optimization products, Premium Solver Pro and Premium Solver Platform to provide full conventional optimization, simulation, and simulation optimization capabilities using one integrated interface.

Problem size limits in combination with Premium Solver Pro are 500 decision variables and 250 constraints. Problem size limits in combination with Premium Solver Platform are 1000 decision variables and 1000 constraints.

Optimization and Resource Allocation

Risk Solver Platform includes five bundled Solver Engines to find solutions for the full spectrum of optimization problems, up to certain size limits:

The **nonlinear LSGRG Solver** handles smooth nonlinear (NLP) problems of up to 1,000 decision variables and 1,000 constraints, plus bounds on the variables, and is augmented with Multistart or "clustering" methods for global optimization. For V11, Risk Solver Platform and Premium Solver Platform were upgraded to include the LSGRG engine while Premium Solver Pro includes the latest version of our GRG solver engine for problems of up to 500 decision variables and 250 constraints. For some smooth nonlinear problems, the methods used by these engines will converge in probability to the globally optimal solution. For other problems, they often yield very good solutions in an acceptable amount of time.

The **Evolutionary Solver** is a hybrid of genetic and evolutionary algorithms and classical optimization methods. It handles non-smooth (NSP) problems of up to 1,000 decision variables and 1,000 constraints, plus bounds on the variables. It was greatly enhanced with parallel algorithms in Version 10.5, further enhanced to double problem size limits in Version 11, and in V11.5 was enhanced with new state-of-of-the-art methods to help you solve hard non-smooth models with better answers in less time.

The **Simplex LP Solver**, used in Premium Solver Pro, handles LP problems with up to 2,000 variables and 2,000 constraints. It is an enhanced version of the Simplex method in the standard Excel Solver that uses both primal and dual Simplex, preprocessing and probing, and basic cut generation methods to speed the solution of LP and L/MIP problems.

The **LP/Quadratic Solver** handles both linear programming (LP) and quadratic programming (QP) problems with up to 8,000 decision variables and 8,000 constraints – four times the size of Premium Solver Pro, and *40 times* the size of Excel Solver. Advanced methods for LP/MIP problems yield solutions as much as hundreds of times faster than the Excel Solver, and 5 to 20 times faster than Premium Solver Pro.

The **Interval Global Solver** uses state-of-the-art interval methods to find the globally optimal solution to a nonlinear optimization problem, all real solutions to a system of nonlinear equations, or an "inner solution" to a system of nonlinear inequalities. It handles smooth nonlinear (NLP) problems of up to 500 decision variables and 250 constraints, plus bounds on the variables.

The **SOCP Barrier Solver** uses an interior point method to solve linear (LP), quadratic (QP), quadratically constrained (QCP), and second order cone programming (SOCP) problems – the *natural generalization* of linear and quadratic programming – with up to 2,000 variables and 8,000 constraints. In Risk Solver Platform, the SOCP Barrier Solver can be used to solve linear programming models *with uncertainty* via robust optimization with the L2 norm.

Simulation and Risk Analysis

Risk Solver Platform and Risk Solver Pro include full-featured Monte Carlo simulation capability. It implements a new approach to Monte Carlo simulation that lets you play 'what if' with uncertain values as easily as you do with ordinary numbers. Each time you change a number on the spreadsheet, a simulation with thousands of trials is performed, and a full range of simulation results and statistics may be displayed *on the spreadsheet*. It uses Frontline's *Polymorphic Spreadsheet Interpreter* technology to achieve breakthrough simulation speeds – up to **100 times faster** than normal Excel-based Monte Carlo simulation – thus making *Interactive Simulation* practical.

Risk Solver Platform and Risk Solver Pro support over 50 analytic and custom distributions (continuous and discrete), over 30 statistics and risk measures, rank order correlation, distribution fitting, multiple random number generators and sampling methods with variance reduction, and powerful capabilities for multiple parameterized simulations.

In addition, their **graphics** help you assess uncertainty, including charts of probability distributions, output frequency charts, sensitivity ("Tornado") charts, scatter plots, and Overlay, Trend, and Box-Whisker charts, in two or three dimensions. Customize chart colors, titles, legends, grid lines, and markers, resize and rotate charts, and print charts or copy them into PowerPoint slides.

Optimal Solutions for Models with Uncertainty

Risk Solver Platform supports several powerful methods for finding robust solutions to optimization problems with uncertainty:

- High-speed **simulation optimization** methods for more general (non-linear, non-smooth or non-convex) problems, where uncertainties may depend on the first-stage decisions. Normal and chance constraints may be used, and a wide range of statistical aggregates can be used to summarize uncertainty in the objective and constraints. A simulation is performed on each major iteration of the optimization. This method is very general, but computationally very expensive, and usually *not scalable* to large size problems. Risk Solver Pro requires either Premium Solver Pro or Premium Solver Platform to be installed to do simulation optimization.

- **Robust optimization** (RO) methods for linear programming problems with uncertainties affecting the objective and constraints. Chance constraints specify a probability of satisfaction, which is converted to a 'budget of uncertainty.' Monte Carlo simulation is used to obtain bound and shape information for the uncertainties. This information is used to *automatically* create a robust counterpart problem, which is then solved. This method is *scalable* to large size models. **Robust optimization** methods and **stochastic programming** (SP)

methods for *two-stage* stochastic linear programming problems with recourse ("wait and see") decisions. Scenarios are automatically created via built-in Monte Carlo simulation, or they may be drawn from user-defined cell ranges of sample values on the spreadsheet. With the benefit of second stage or *recourse* decisions, solutions are typically 'well-hedged' but not overly conservative. The RO and SP methods are relatively *scalable* to large size.

Sensitivity Analysis and Model Parameters

Risk Solver Platform and Risk Solver Pro include facilities for sensitivity analysis of your Excel model, that can be used before even starting an optimization or simulation model. It is especially easy to identify the model parameters with the most impact on your computed results – you can simply select *any formula cell*, and choose Parameters – Identify from the Ribbon to quickly find the input cells with the greatest impact on this formula, ranked and shown in a Tornado chart.

You can choose some of these input cells to serve as Sensitivity Parameters, and then produce reports and charts that show the impact on computed results of varying these parameters over a range you specify. You can also turn these parameters into (or define other cells as) Simulation Parameters or Optimization Parameters, and produce reports and charts of simulation and optimization results as your parameters are automatically varied.

Multiple Optimizations and Simulations

Risk Solver Platform and Risk Solver Pro have powerful capabilities to perform multiple optimizations or simulations, automatically varying the values of parameters you specify, and collecting both optimization and simulation results, which may be displayed on the spreadsheet, or summarized in a variety of built-in reports and charts. Many operations that would require a programming language in other systems, or VBA in the Excel Solver, can be easily performed without programming.

V11 enriches the possible reports and charts for multiple parameterized optimizations and multiple parameterized simulations. You can run an optimization or simulation for each combination of two parameter values, and create a tabular report or 3-D chart of the results.

Decision Trees on the Spreadsheet

Risk Solver Platform also includes a facility to create decision trees on your Excel spreadsheet. Using the Ribbon, you can easily create decision nodes and branches, event nodes and branches, and terminal nodes. The tree is drawn in graphic form on the spreadsheet; standard Excel worksheet formulas compute 'rollback' values at each node, and the best-choice value at the root node, based on either expected value or utility function (certainty equivalent) criteria. With a Ribbon choice, you can graphically highlight the optimal path through the tree.

Since all computations for decision trees are performed via standard Excel worksheet formulas, you can use decision trees in your simulation and optimization models. You must make discrete choices for decisions, and define discrete alternatives for events, but you can, for example, define distributions for

event outcomes and/or costs incurred for decisions, and view the composite distribution of outcomes at the root node. Our examples workbook (available from **Help > Examples** from the ribbon menu) includes links to several example models illustrating ways to use decision trees.

Large-Scale Solver Engines

Our Platform products, Risk Solver Platform, Premium Solver Platform and Solver SDK Platform all support multiple optional, plug-in Solver engines, in addition to their "bundled" Solver engines. Such Solver engines are licensed as separate products; they provide additional power and capacity to solve problems much larger and/or more difficult than the problems handled by the bundled Solver engines. Unlike most other optimization software, a license for one of Frontline's Solver engines enables you to use that Solver in Excel, Matlab, Java, C/C++, C#, Visual Basic, VB.NET, and other languages.

These optional plug-in Solver engines for our Platform products are seamlessly integrated – to use one, you simply select the Solver engine by name in the dropdown list that appears in the Task Pane Engine tab. They produce reports as Excel worksheets, like the bundled Solver engines; their options can be set in the Task Pane, or in Solver Options dialogs; and they can be controlled by VBA code in your custom applications. The Solver engines are also seamlessly integrated into the Solver Platform SDK. Trial licenses for these Solver engines are available, allowing you to evaluate how well they perform on a challenging Solver model that you've developed.

The large-scale Solver Engines are more valuable than ever in our Platform products, which can transform linear programming models *with uncertainty* into larger, conventional linear programming or second order cone programming models, using stochastic programming and robust optimization methods.

Automatic Mode and Solution Time

Risk Solver Platform V11.5 has many performance enhancements, ranging from faster model diagnosis and convexity testing to an updated Evolutionary engine and several updated plug-in solver engines. However, you should be aware of Automatic mode and its potential impact on solution time.

Automatic mode refers to the Automatic choice available for several Platform options, which formerly required some thought and manual setting by the user. Automatic is the *default* choice for these options in newly created models. The options (found on the Task Pane Platform tab) include:

Optimization Model Interpreter	Simulation Model Interpreter
Solve Uncertain Models	Supply Engine With
Nonsmooth Model Transformation	Automatically Select Engine (on the Task Pane Engine tab)
Stochastic Transformation	

Each Automatic choice means that the Platform can analyze your model and automatically choose the best setting for these options. This normally results in better solutions and faster times; however, the process of *analyzing* your model and *trying* certain alternatives can take *extra* time, especially for large models.

To improve solution time when you re-run your model, you can inspect the solution log in the Task Pane Output tab to see what choices were made

automatically, **manually set the Platform options** to these same choices, and save your workbook. This will save time when you next run the model.

Premium Solver Pro includes Automatic Mode for analyzing the model and selecting the best available engine. However, Premium Solver Pro does not include the ability to automatically transform models. This feature is only available on our Platform level products and can help get better answers in less time.

What's New in V11.5

Version 11.5 introduces a wide range of enhancements in our Platform and Solver Engine products, designed to help you build models more quickly, solve them more quickly, and obtain better solutions.

New Distribution Wizard

For most users, the main challenge when building a simulation model is how to choose the right probability distribution and the right parameters for each uncertain variable. The new Distribution Wizard in V11.5 can help! Try it by choosing **Distributions – Distribution Wizard** from the Ribbon, and simply following the prompts.

New Constraint Wizard

The main challenge when building an optimization model is specifying constraints. The new Constraint Wizard in V11.5 makes this easier – for normal constraints, 'chance constraints,' and restrictions such as 'integer' on decision variables. Try it by choosing **Constraints – Constraint Wizard** from the Ribbon, and following the prompts.

Improved Guided Mode Dialogs

Guided Mode can give you great insight into why your model is difficult or easy to solve – but when you're in a hurry, it's easy to overlook its messages. We've re-worded nearly all of the Guided Mode dialogs, so you can determine at a glance what the analysis means for your model. Try it – leave **Guided Mode on** for a while.

More Example Models, Easier to Find

We've expanded the set of example models installed with Risk Solver Platform, and made it much easier to find the examples most relevant to you. Try it – choose **Help – Examples** from the Ribbon, and select examples by their summary descriptions, special features, and industry or functional area.

Faster Solutions for Large Quadratic and Nonlinear Models

We've speeded up the PSI Interpreter, when it's analyzing your model type, and when it's being used by a Solver Engine to compute rate-of-change and curvature information to guide the Solver to an optimal solution. For large quadratic and nonlinear models, the speedup can be quite dramatic!

Next Generation Methods in Evolutionary Solver

To help you solve arbitrary Excel models with IF, CHOOSE, LOOKUP and similar functions, we've greatly enhanced the algorithmic methods in our popular Evolutionary Solver, so it finds high-quality solutions faster on many models, and *much* faster on some models. You can control which set of algorithms are used by going selecting the Evolutionary engine on the **Engine**

tab of the **Task Pane** and setting the **Legacy Mode** option to *True* (to use the older algorithms) or *False* (to use the newer set of algorithms) and see which works better for your particular model.

Visualization of Solution Values for Recourse Decision Variables

Risk Solver Platform is unique in its ability to solve stochastic optimization models with recourse decision variables – models that accurately describe many real-world plans and decisions, but that cannot even be defined in competitive products, let alone solved. When such a model is solved, each recourse decision variable has many solutions, corresponding to different realizations of uncertainty. In V11.5, recourse decision variables have built-in charts and statistics, much like uncertain variables – illuminating the properties of the optimal solution for users.

New Decision Tree Capabilities in Risk Solver Pro

Risk Solver Pro now includes decision tree capabilities formerly only available in Risk Solver Platform. Decision trees can be used alone to model risky, high-stakes decisions, and you can easily apply sensitivity analysis and simulation to decision tree models, to gain much greater insight into the decisions and outcomes you are modeling. Click on **Help – Examples** and then click on the *"Decision Tree Examples"* button to see for yourself how useful decision trees can be.

Large Scale Solver Engines Feature Best-Ever Performance

Several of our plug-in large-scale Solver Engines have major enhancements in V11.5 – notably the KNITRO Solver, Large-Scale SQP Solver (on non-smooth models), Gurobi Solver, and XPRESS Solver. If you have a large model or one that's difficult to solve, be sure to download, install, and try the V11.5 plug-in Solver Engines. By checking "Automatically Select Engine" at the top of the Engine tab in the Task Pane and then solving your model you can see quickly in the Output tab which engine was identified as best for your model.

Faster and More Flexible Distribution Fitting

We've made distribution fitting more flexible, with a new PsiFit() function that fits distributions dynamically when your model changes. In addition, we've speeded up distribution fitting dramatically.

We've made lots of other small changes in the software to make it easier and faster to use. We'd love to hear your feedback on V11.5 – contact us at info@solver.com.

Installation and Add-Ins

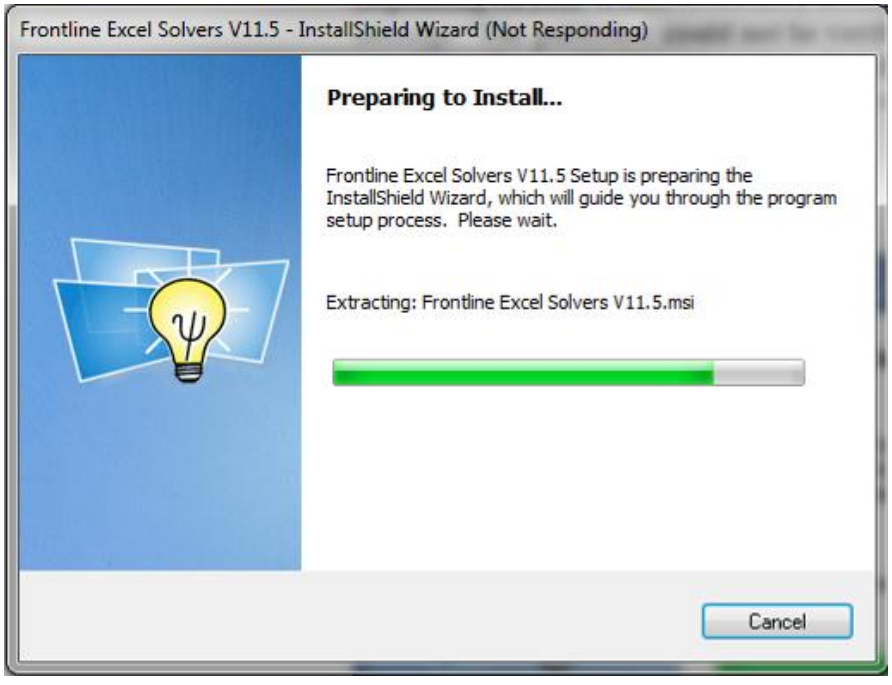
What You Need

In order to install Risk Solver Platform V11.5 software, you must have first installed Microsoft Excel 2010, Excel 2007, or Excel 2003 on Windows 7, Windows Vista, Windows XP or Windows Server 2008. It's not essential to have the standard Excel Solver installed.

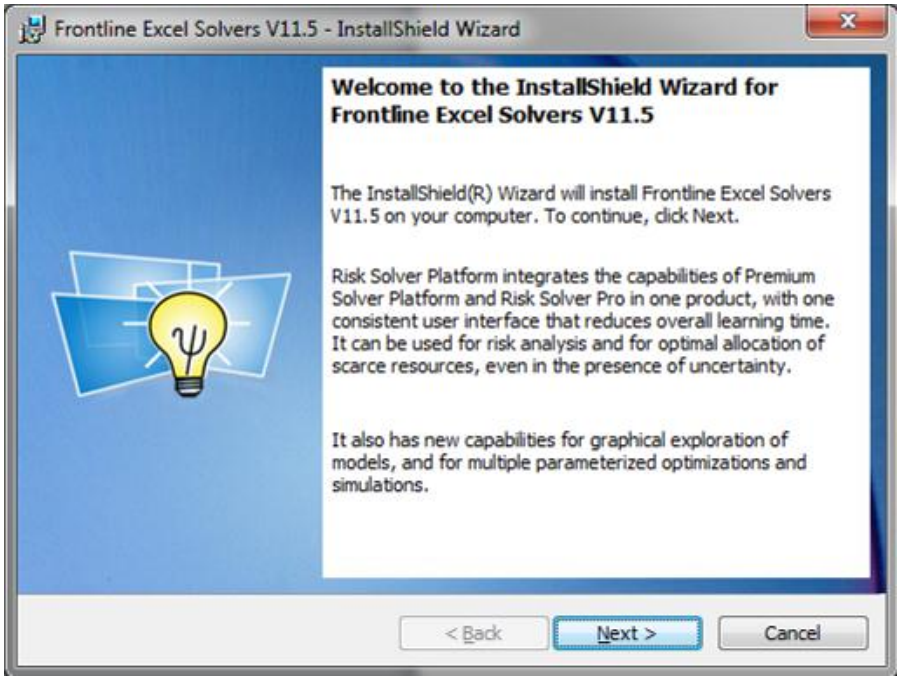
Installing the Software

To install Risk Solver Platform to work with any 32-bit version of Microsoft Excel, simply run the program **SolverSetup.exe**, which contains all of the Solver program, Help, User Guide, and example files in compressed form. To install Risk Solver Platform to work with 64-bit Excel 2010, run **SolverSetup64.exe**.

Depending on your Windows security settings, you might be prompted with a message **"The publisher could not be verified. Are you sure you want to run this software?"** You may safely click **Run** in response to this message. You'll first see a dialog like the one below, while the files are decompressed:

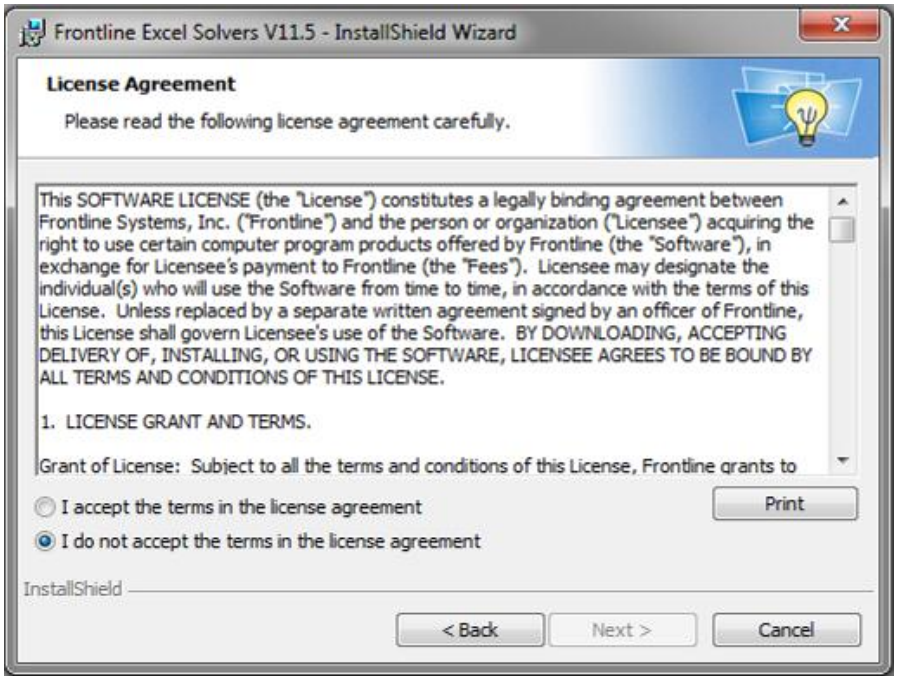


Next, you'll briefly see the standard Windows Installer dialog. Then a dialog box like the one shown below should appear:



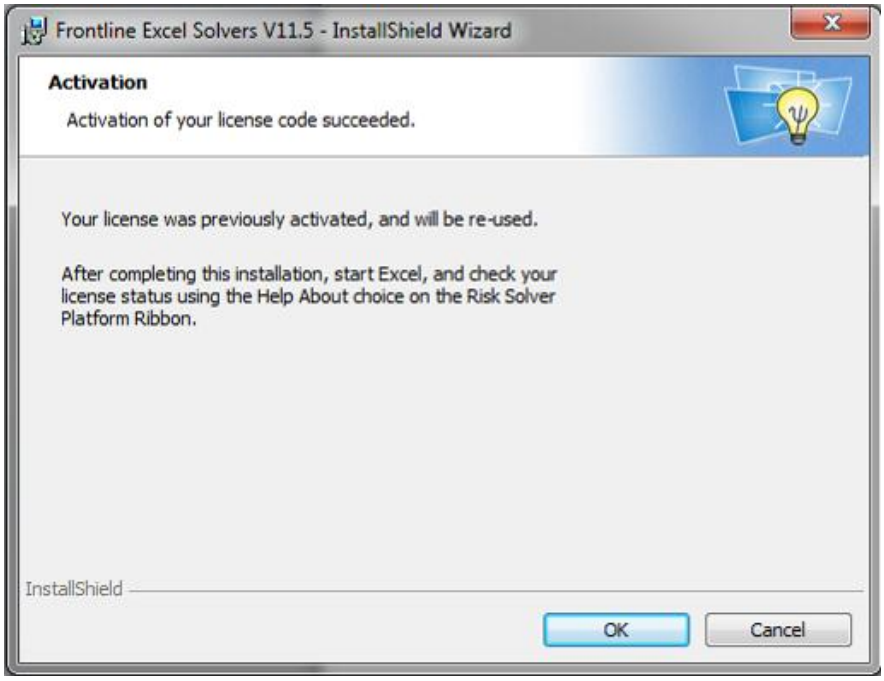
Click **Next** to proceed. You will then be prompted for an installation password , which Frontline Systems will provide via email to your registered email address. Enter it into the Installation Password field in dialog box. In addition, you have the option to enter a license activation code in that related field. Note, the Setup program looks for a license file that may already exist on your system and checks your license status. If you enter an activation code (you must have Internet access for this to succeed), the Setup program will display a dialog reporting whether your license was successfully activated. But you don't *have* to do this – just click **Next**.

Next, the Setup program will ask if you accept Frontline's software license agreement. You must click "I accept" and **Next** in order to be able to proceed.

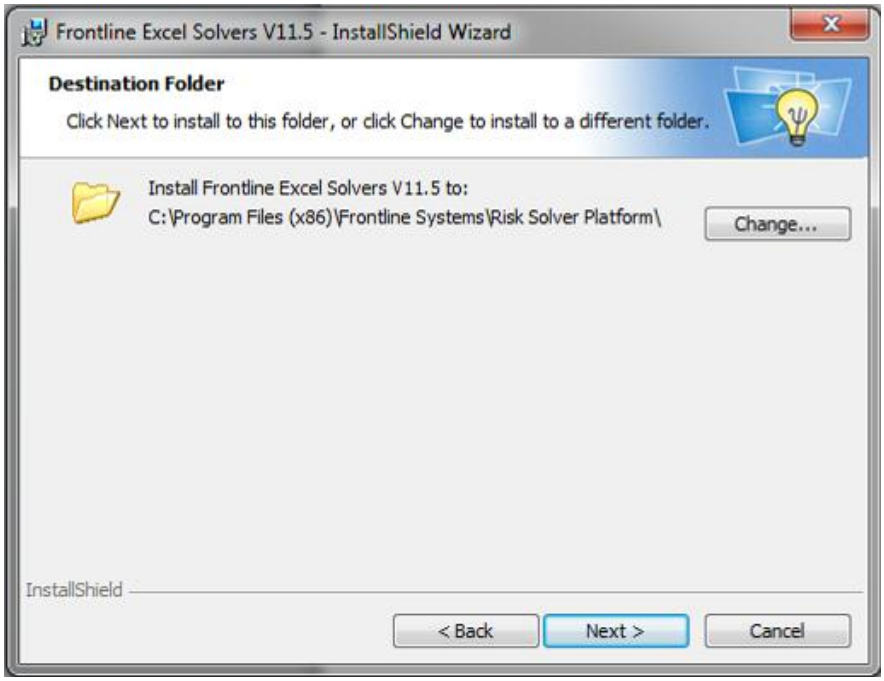


After you accept the license agreement you'll see a dialog confirming your activation code succeeded. If the dialog looks like the next screen shot below

you just need to click **OK** to proceed. Otherwise, follow the instruction shown in the dialog.

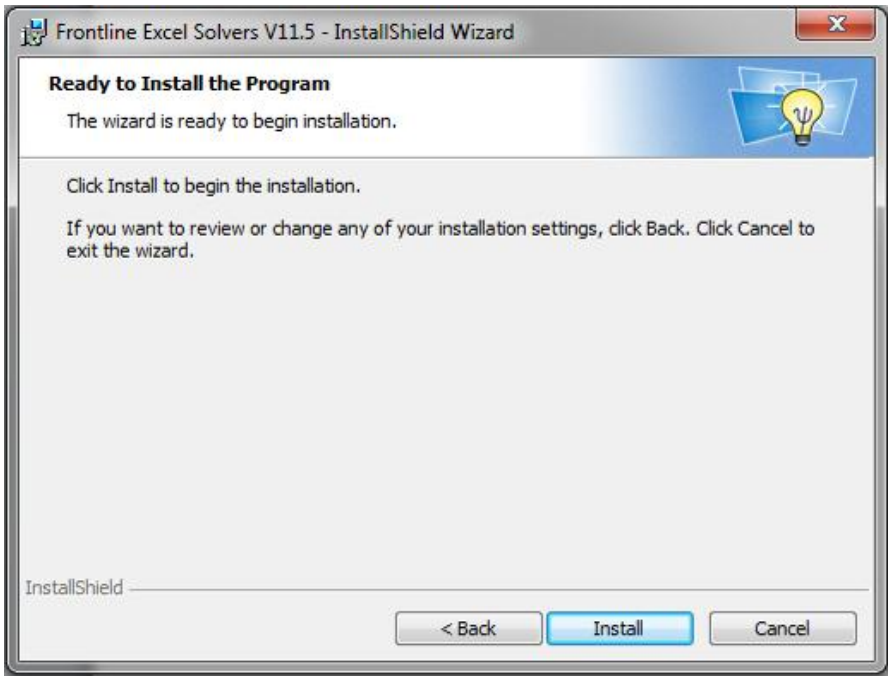


After you click OK above, the Setup program then displays a dialog box like the one shown below, where you can select or confirm the folder to which files will be copied (normally C:\Program Files\Frontline Systems\Risk Solver Platform, or if you're installing Risk Solver Platform for 32-bit Excel on 64-bit Windows, C:\Program Files (x86)\Frontline Systems\Risk Solver Platform). Click **Next** to proceed.

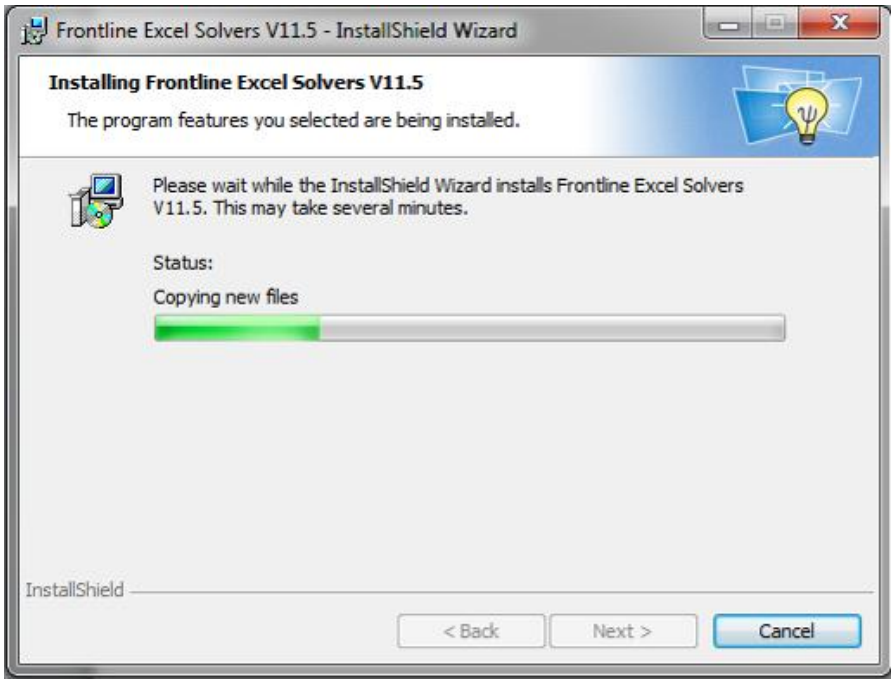


If you have an existing license, or you've just activated a license for full Risk Solver Platform, the Setup program will give you the option to run the software as a *subset* product –Premium Solver Platform, Premium Solver Pro, or Risk Solver Pro – instead of the full Risk Solver Platform. The main reason to do this is to confirm that the functionality you need is available in the subset product.

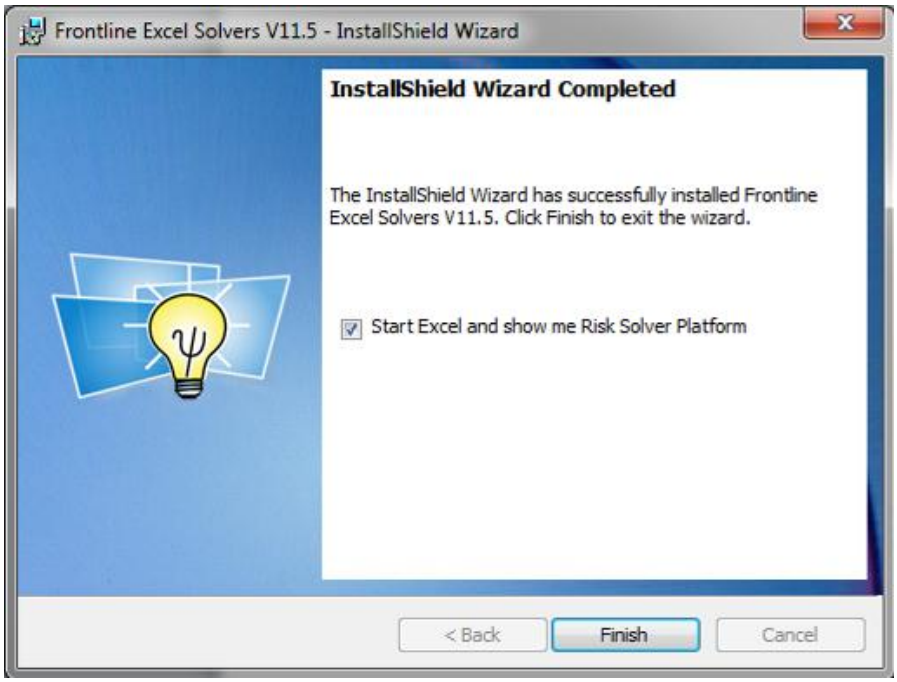
Click **Next** to proceed. You'll see a dialog confirming that the preliminary steps are complete, and the installation is ready to begin:



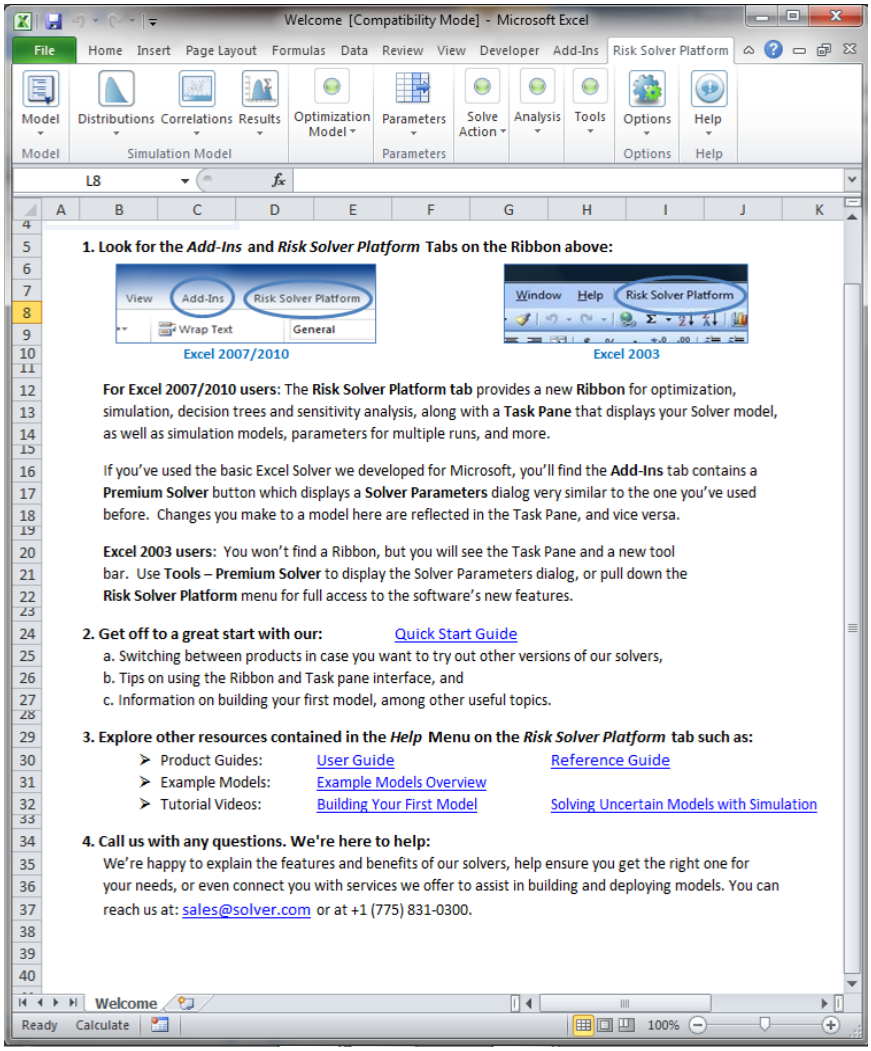
After you click **Install**, the Risk Solver Platform files will be installed, and the program file RSPAddin.xll will be registered as a COM add-in (which may take some time). A progress dialog appears, as shown below:



When the installation is complete, you'll see a dialog box like the one below. Click **Finish** to exit the installation wizard.



Risk Solver Platform, or a sub-set product if you chose to install a different version of our solvers, is now installed. Simply click "Finish" and Microsoft Excel will launch with a Welcome workbook containing information to help you get started quickly.



Uninstalling the Software

To uninstall Risk Solver Platform, just run the **SolverSetup** program as outlined above. You'll be asked to confirm that you want to remove the software.

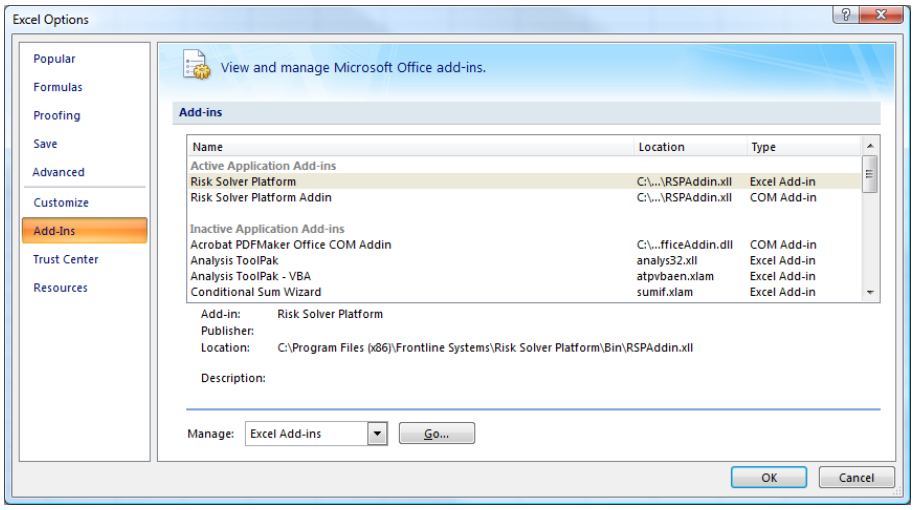
You can also uninstall by choosing **Control Panel** from the **Start** menu, and double-clicking the **Programs and Features** or **Add/Remove Programs** applet. In the list box below "Currently installed programs," scroll down if necessary until you reach lines beginning with "Frontline," select Risk Solver Platform V11.5, and click the Uninstall/Change or Add/Remove… button. Click OK in the confirming dialog box to uninstall the software.

Activating and Deactivating the Software

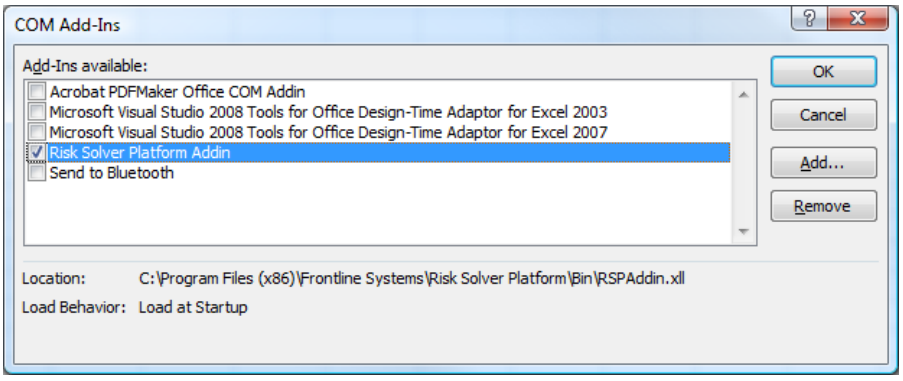
Risk Solver Platform's main program file RSPAddin.xll is a COM add-in, an XLL add-in, and a COM server. The add-in file **Solver.xla** is optional – it is needed only if you wish to use the "traditional" VBA functions to control Risk Solver Platform, instead of its new VBA Object-Oriented API.

Excel 2010 and 2007

In Excel 2010 and 2007, you can manage all types of add-ins from one dialog, reached by clicking the **File** tab in Excel 2010, or the upper left corner **button** in Excel 2007, choosing **Excel Options**, then choosing **Add-Ins** in the pane on the left, as shown below.



You can manage add-ins by selecting the type of add-in from the dropdown list at the bottom of this dialog. For example, if you select COM Add-ins from the dropdown list and clock the Go button, the dialog shown below appears.



If you uncheck the box next to "Risk Solver Platform Addin" and click OK, you will deactivate the Risk Solver Platform COM add-in, which will remove the Risk Solver Platform tab from the Ribbon, and also remove the PSI functions for optimization from the Excel 2007 Function Wizard.

Excel 2003

In earlier versions of Excel, COM add-ins and other add-ins are managed in separate dialogs, and the COM Add-In dialog is available only if you display a toolbar which is hidden by default. To display this toolbar:

1. On the **View** menu, point to **Toolbars**, and then click **Customize**.

2. Click the **Commands** tab.
3. Under **Categories**, click **Tools**.
4. Under **Commands**, click **COM Add-Ins** and drag your selection to the toolbar.

Once you have done this, you can click **COM Add-Ins** on the toolbar to see a list of the available add-ins in the COM Add-Ins dialog box, as shown above.

If you uncheck the box next to "Risk Solver Platform Addin" and click OK, you will deactivate the Risk Solver Platform COM add-in, which will remove Risk Solver Platform from the main menu bar, and also remove the PSI functions for optimization from the Insert Function dialog.

Using Help, Licensing and Product Subsets

Introduction

This chapter describes the ways Risk Solver Platform V11.x differs from its predecessors in terms of overall operation, including registration, licensing, use of product subsets, and use of the Startup Screen, online Help and examples.

Working with Licenses in V11.x

A **license** is a grant of rights, from Frontline Systems to you, to use our software in specified ways. Information about a license – for example, its temporary vs. permanent status and its expiration date – is encoded in a **license code**. The same binary files are used for Risk Solver Platform, Premium Solver Platform, Premium Solver Pro, and Risk Solver Pro. The product features you see depends on the license code you have.

Using the License File Solver.lic

Risk Solver Platform V11.x stores license codes in a text file named **Solver.lic**. Old license codes for V10.x and earlier have no negative effect in V11.x; they can appear in the Solver.lic file and will be ignored.

If you already have a Solver.lic file, SolverSetup adds license codes to this file. If not, SolverSetup creates this file in a default location:

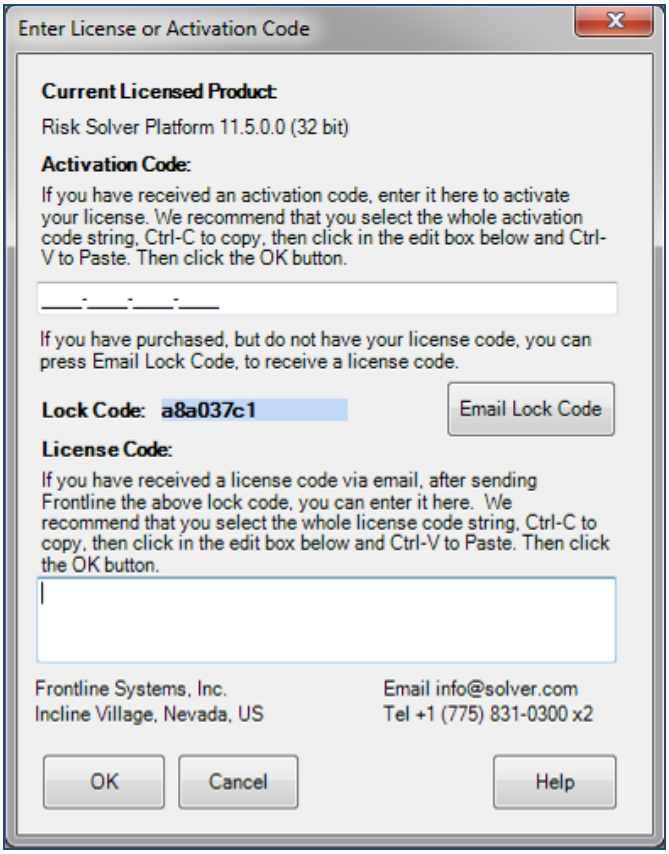
Vista: C:\ProgramData\Frontline Systems

XP: C:\Documents and Settings\All Users\Application Data\Frontline Systems

In V11 and V10, the SolverSetup program creates an environment variable **FRONTLIC** whose value is the path to the Solver.lic file. The old license manager in V9.0 and earlier versions used an environment variable LSERVRC.

License Codes and Internet Activation

You can enter a new license code at any time while you're using Risk Solver Platform. To do this, choose **Help – License Code** from the Risk Solver Platform Ribbon. A dialog like the one below will appear.



You have two options to obtain and activate a license, using this dialog:

1. If you contact Frontline Systems at (775) 831-0300 or info@solver.com, and give us the Lock Code shown in the **middle** of the dialog (click the **Email Lock Code** button to do this quickly), we can generate a license code for your PC, and email this to you. You can then select and copy (Ctrl+C) the license code, and paste it (Ctrl+V) into the **lower** edit box in this dialog.

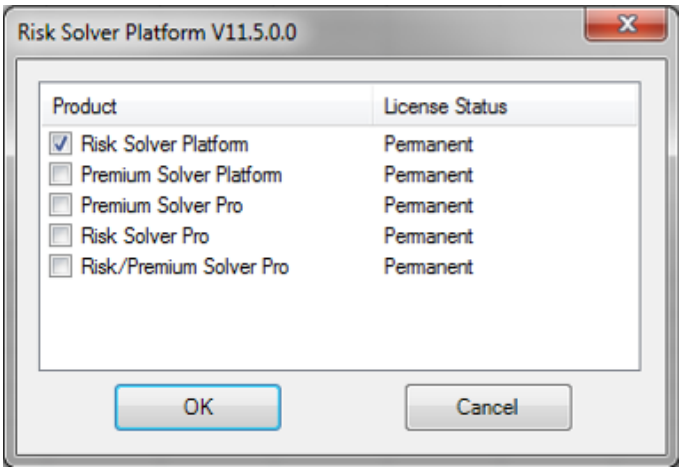
2. Even easier, and available 24x7 if you have Internet access on this PC: If you have a license Activation Code from Frontline Systems, you can copy and paste it into the **upper** edit box in this dialog. When you click OK, Risk Solver Platform contacts Frontline's license server over the Internet, sends the Lock Code and receives your license code automatically. You'll see a message confirming the license activation, or reporting any errors.

If you have questions, please contact Frontline Systems at (775) 831-0300 or **info@solver.com**.

Running Subset Products in V11.5

New users often wish to download and evaluate our products for Excel. To accommodate this, we make available downloads of the SolverSetup program with a 15-day trial license, which gives users access to all of the functionality and capacity of Risk Solver Platform. But some users will ultimately choose to purchase a license for a product that is a subset of Risk Solver Platform, such as Premium Solver Platform, Premium Solver Pro, or Risk Solver Pro.

To help users confirm that a subset product will have the capabilities and performance that they want, V11 has the ability to quickly switch Risk Solver Platform to operate as a subset product, without the need to install a new license code. To do this, choose **Help – Change Product** on the Risk Solver Platform Ribbon. A dialog like the one below will appear.



In this dialog, you can select the subset product you want, and click OK. The change to a new product takes effect immediately: You'll see the subset product name instead of Risk Solver Platform as a tab on the Ribbon, and a subset of the Ribbon options

Premium Solver Platform

Premium Solver Platform is Frontline's most powerful product for *conventional* optimization. It includes the PSI Interpreter, five built-in Solvers (LP/Quadratic, SOCP Barrier, LSGRG Nonlinear, Interval Global, and Evolutionary), and it supports plug-in large-scale Solver Engines. It has no Monte Carlo simulation, simulation optimization, stochastic programming or robust optimization.

Premium Solver Pro

Premium Solver Pro is Frontline's basic upgrade for the standard Excel Solver, for *conventional* optimization. It includes a Simplex LP Solver, GRG Nonlinear Solver, Evolutionary Solver, Branch & Bound method for integer and alldifferent constraints, and the Multistart method for global optimization.

Risk Solver Pro

Risk Solver Pro is Frontline's full-function product for Monte Carlo simulation. It includes all of the simulation functionality of Risk Solver Platform including the Psi Interpreter for simulations, and can be combined with Premium Solver Pro or Premium Solver Platform to do simulation optimizations as well.

Combining Risk Solver Pro and Premium Solver Pro

In the change product screen shot above you will notice the "Risk/Premium Solver Pro" option. This enables both products to be active at once. With this option selected, in one integrated interface, you'll get the Monte Carlo Simulation and Decision Trees capabilities of Risk Solver Pro, the Conventional Optimization power of Premium Solver Pro plus industry leading Simulation Optimization capabilities enabled by combining the two. More features and power for less than the price of competing alternatives.

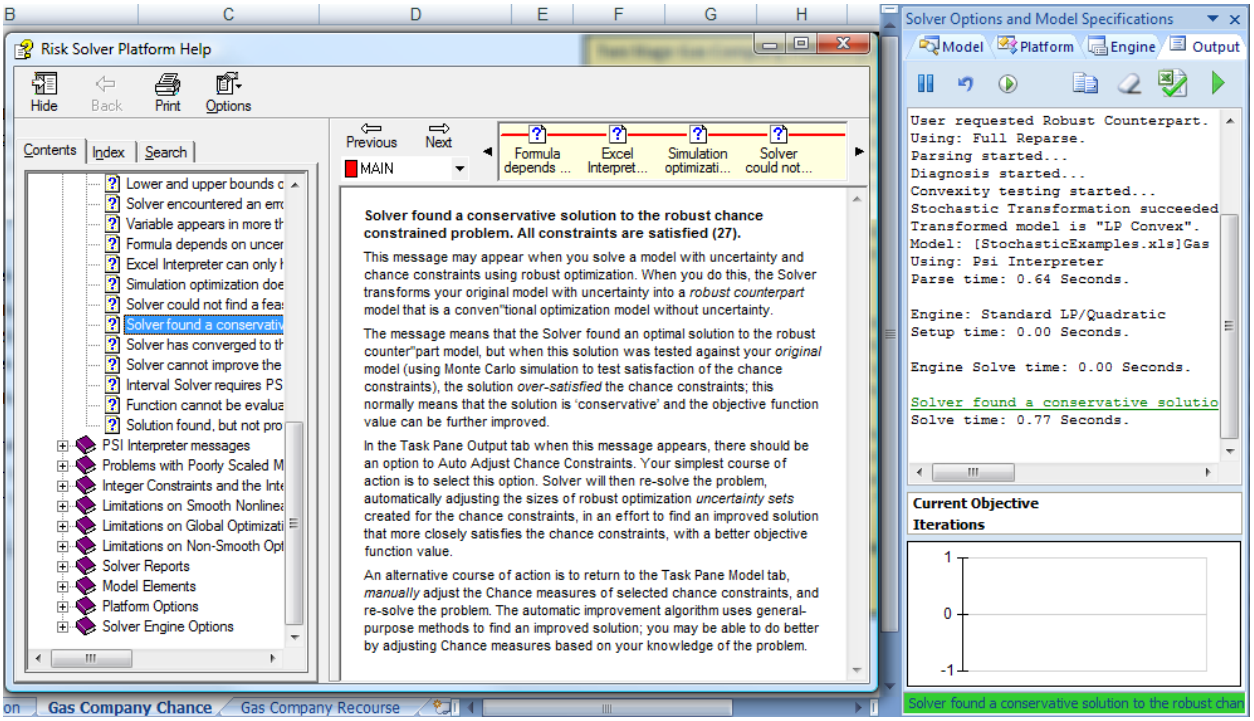
Using the Welcome Screen

This screen appears *automatically* **only** when you click the Risk Solver Platform tab on the Ribbon in Excel 2010 or 2007, or use the Risk Solver Platform menu in Excel 2003 – and then **only** if you are using a trial license. You can display the Welcome Screen manually by choosing **Help – Welcome Screen** from the Ribbon. You can control whether the screen appears automatically by selecting or clearing the check box in the lower left corner, "Show this dialog at first use."

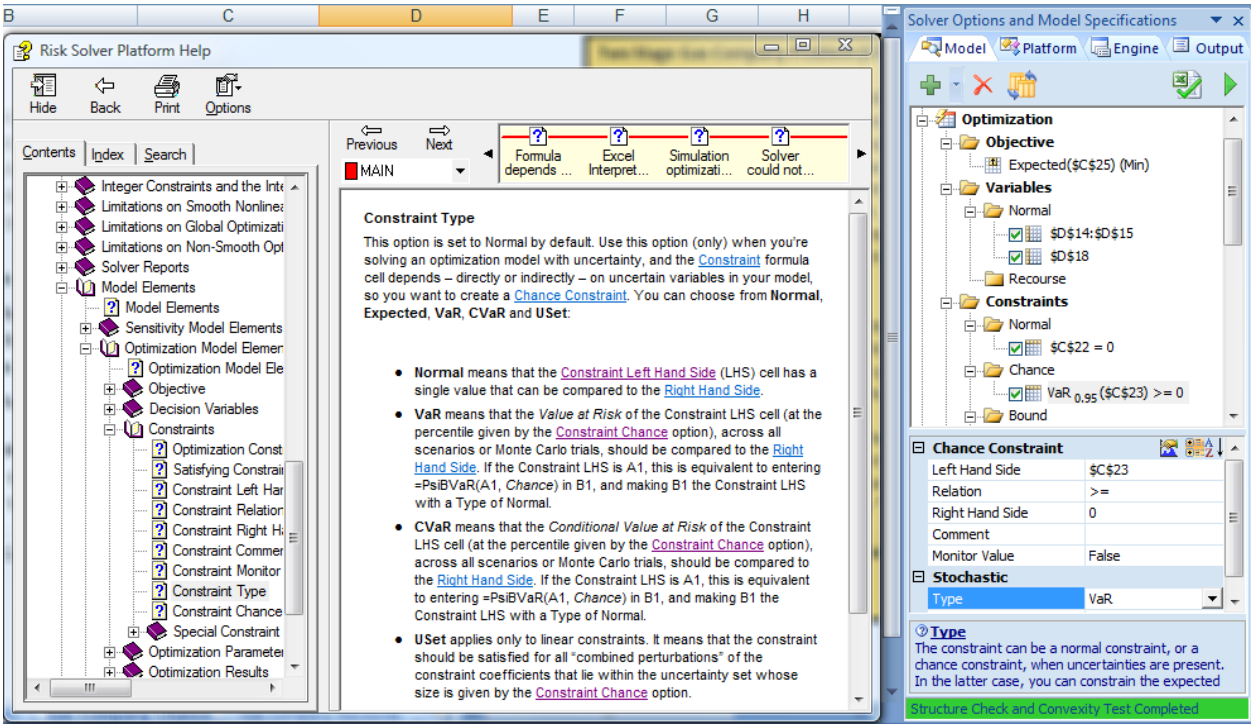
Using Online Help

Risk Solver Platform V11.5 has a significant amount of context-sensitive Help. You can quickly access Help on any Solver Result message that appears in the Task Pane Output tab, any option that appears on the Engine tab or Platform tab, or any element of your model that appears on the Model tab.

Solver Result messages appear as underlined links – you can simply click them. For example, here's the Help that appears when you click the Solver Results message for the Gas Company Chance example in StochasticExamples.xls:



To access Help on any Platform option, Engine option, or Model element, click the underlined link in the brief explanation that appears at the bottom of the Task Pane. For example, here's the Help that appears when you click the Type field link for a chance constraint in your model:



Examples: Conventional Optimization

Introduction

This chapter introduces conventional optimization (with no uncertainty) in Risk Solver Platform, with a series of examples. Except as noted in specific sections, all of these examples can be used in any of the above product subsets. These examples can be found through the "Optimization" examples text link on the Examples overview worksheet accessed from Help > Examples on the Risk Solver Platform Ribbon.

- "A First Optimization Model" takes you step-by-step through the process of creating an optimization model from scratch. It illustrates use of the Ribbon and Task Pane, and its relationship to the Excel Solver Parameters dialog.
- "Introducing the Standard Examples," "Linear Programming Examples" and "Nonlinear Optimization Examples" takes you through the workbook StandardExamples.xls, with seven examples that illustrate a number of Risk Solver Platform features.

A First Optimization Model

This section is meant for you if you're familiar with optimization, but you've never used the Excel Solver or Premium Solver Pro. It shows how you can translate from the algebraic statement of an optimization problem to a spreadsheet Solver model. If *optimization* is new to you, consult the chapter "Mastering Conventional Optimization Concepts."

Note: This first step-by-step example is a 'quick and dirty' approach 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. The models in the **StandardExamples.xls** workbook illustrate some of these features. 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.

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. We'll use A1:A3 in the example below.

2. Pick a cell to represent the objective function, and enter a formula that calculates the objective function value in this cell. We'll use A4 in the example below.

3. Pick other cells and use them to enter the formulas that calculate the left hand sides of the constraints. We'll use B1:B5 in the example below.

4. The constraint right hand sides can be entered as numbers in other cells, or entered directly in the Solver's Add Constraint dialog. We'll use C1:C5 for this purpose in the example below.

Within this overall framework, you have a great deal of flexibility in choosing cells to use, and calculating the objective and constraints. For example, the objective function will ultimately depend on the decision variable cells, but you don't have to calculate the entire function in one formula cell. You can use any number of formula cells to compute intermediate results, and use these to calculate the objective function or the constraints.

You can write a linear expression easily with Excel's SUMPRODUCT function, or you can use + and * operators, as shown below. You can use Excel's *array* formulas, and Excel functions that return vector and matrix results. And you can use Excel's rich facilities to access data in external text files, Web pages, and relational and multidimensional databases to 'populate' your model.

The Model in Algebraic and Spreadsheet Form

Consider the following problem. 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. This is a simple linear programming problem, also used to illustrate other ideas later in "Linear Programming Examples."

The Algebraic Form

The 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 $x_1, x_2, x_3 >= 0$. Note that terms like $0\ x_3$ are included purely to show the structure of the model – they can be either omitted or included when entering formulas in Excel.

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 can be entered in cell A4, much like the algebraic form above:

A4: =75*A1+50*A2+35*A3

We can go on to enter a formula in (say) cell B1 for the first constraint left hand side (Chassis) as =1*A1+1*A2+0*A3, or the simpler form:

B1: =A1+A2

Similarly, we can 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):

B2: =A1
B3: =2*A1+2*A2+A3
B4: =A1+A2
B5: =2*A1+A2+A3

Finally, we'll enter the constraint right hand side values in cells C1:C5:

C1: 400
C2: 200
C3: 800
C4: 400
C5: 600

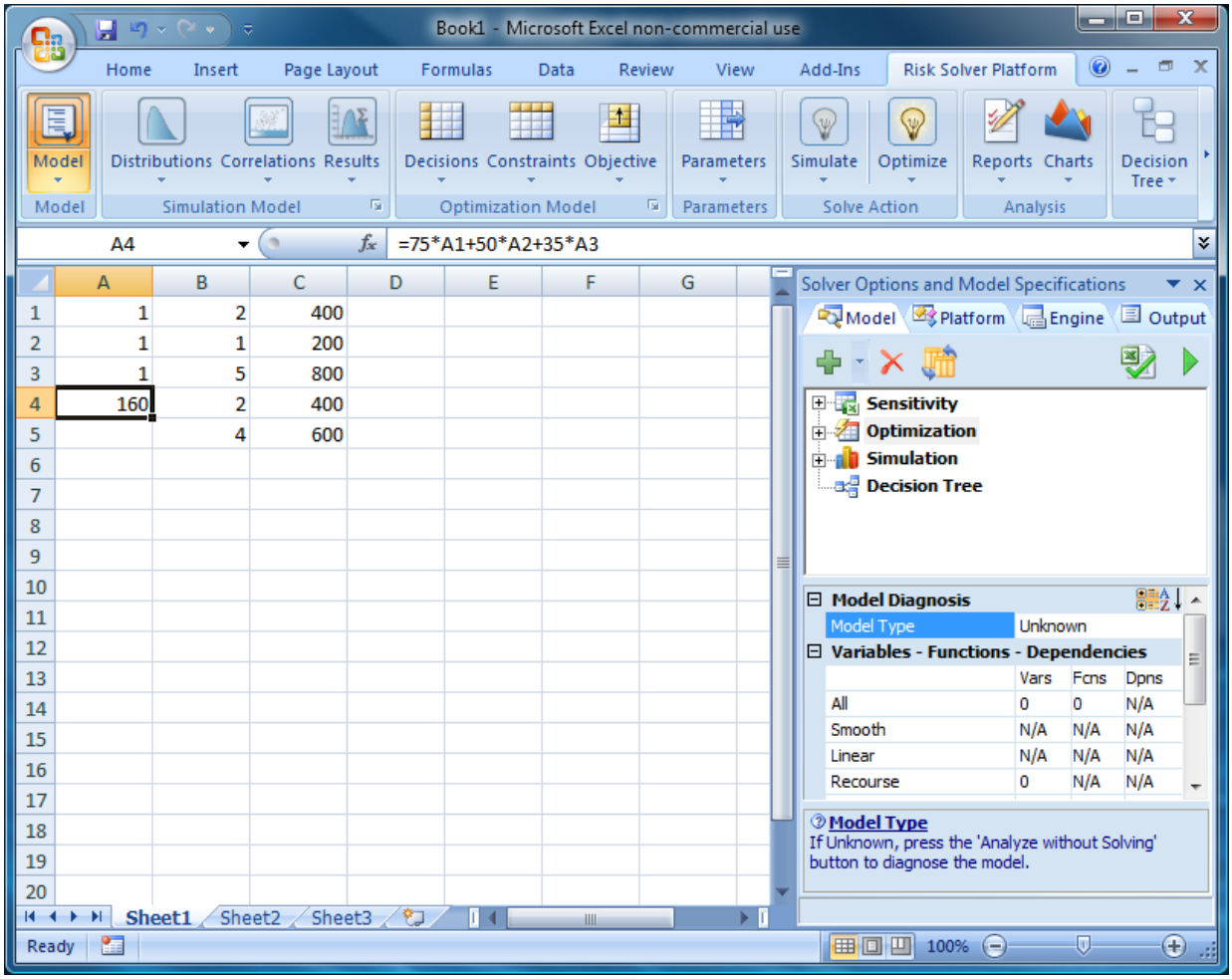
We now have a simple spreadsheet model that we can use to play 'what if.' For any values we enter for the decision variables in cells A1, A2 and A3, the objective (Total Profit) and the corresponding values of the constraint left hand sides (the numbers of parts used) will be calculated.

We want to turn this 'what if' spreadsheet model into an optimization model, where the Solver will automatically find *optimal* values for the cells A1:A3, so that the objective function at A4 is maximized, and the constraints are satisfied.

Defining and Solving the Optimization Model

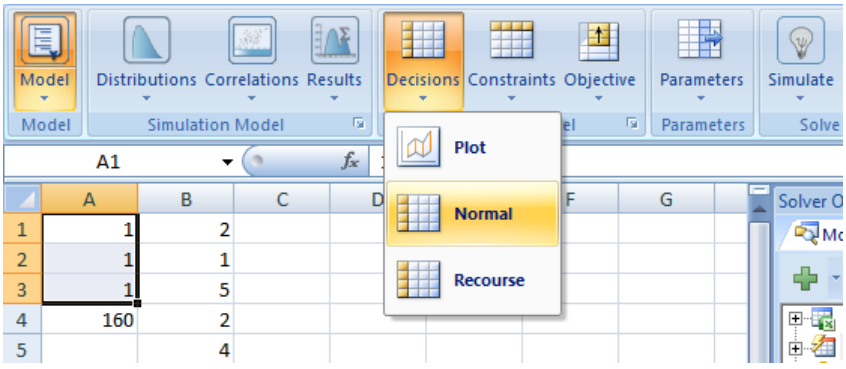
To begin, click the Risk Solver Platform tab on the Ribbon. If you have a trial license, click the button "Start Using Risk Solver Platform" in the startup dialog. Your worksheet with an empty **Task Pane** ("Solver Options and Model Specifications") **Model tab** should appear, as shown on the next page.

To define the optimization problem, we'll use the Ribbon 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. We can perform these steps in any order.



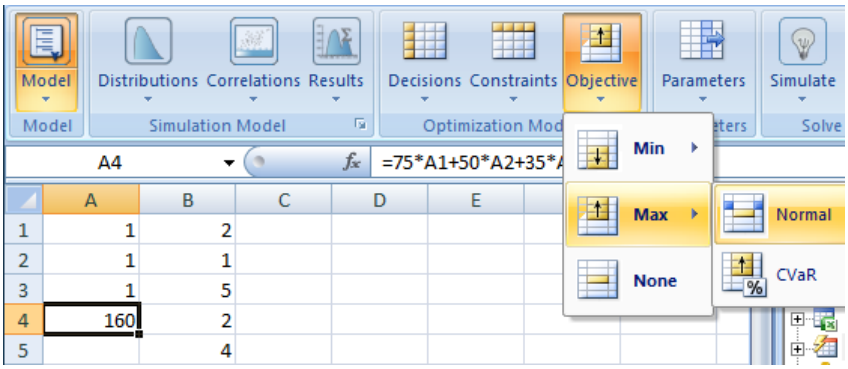
Defining Decision Variables

To define the **decision variables**, use your mouse to select the range A1:A3 on the worksheet. Then click the **Decisions** button to open the dropdown gallery (shown below), and click **Normal** to define A1:A3 as normal decision variables.



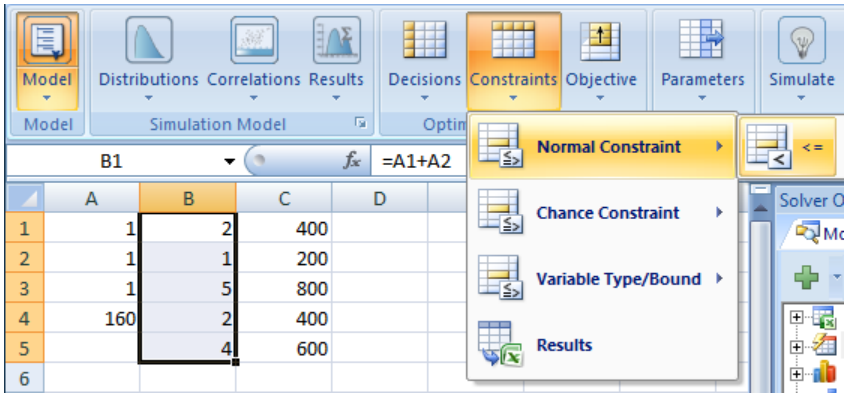
Defining the Objective

To define the **objective**, use your mouse to select cell A4 on the worksheet. Then click the **Objective** button, and click **Max** and **Normal** to define A4 as a normal objective to be maximized – as shown on the next page.

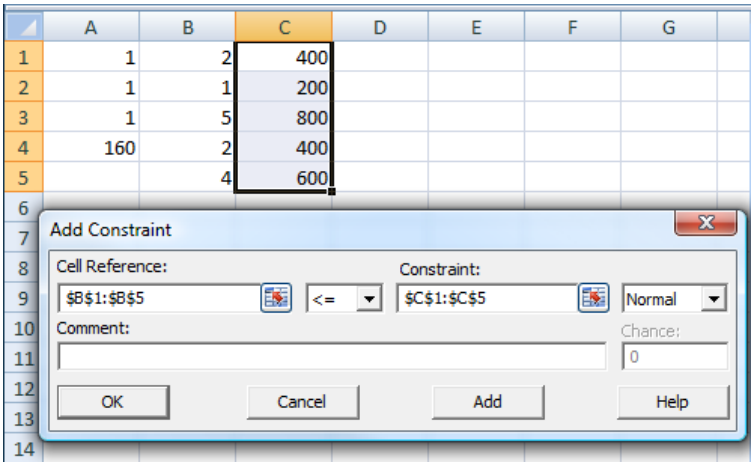


Defining Constraints

To define the **constraints**, use your mouse to select the range B1:B5 on the worksheet. Then click the **Constraints** button, and click **Normal Constraint** and <= to define B1:B5 as the left hand sides of five <= constraints.

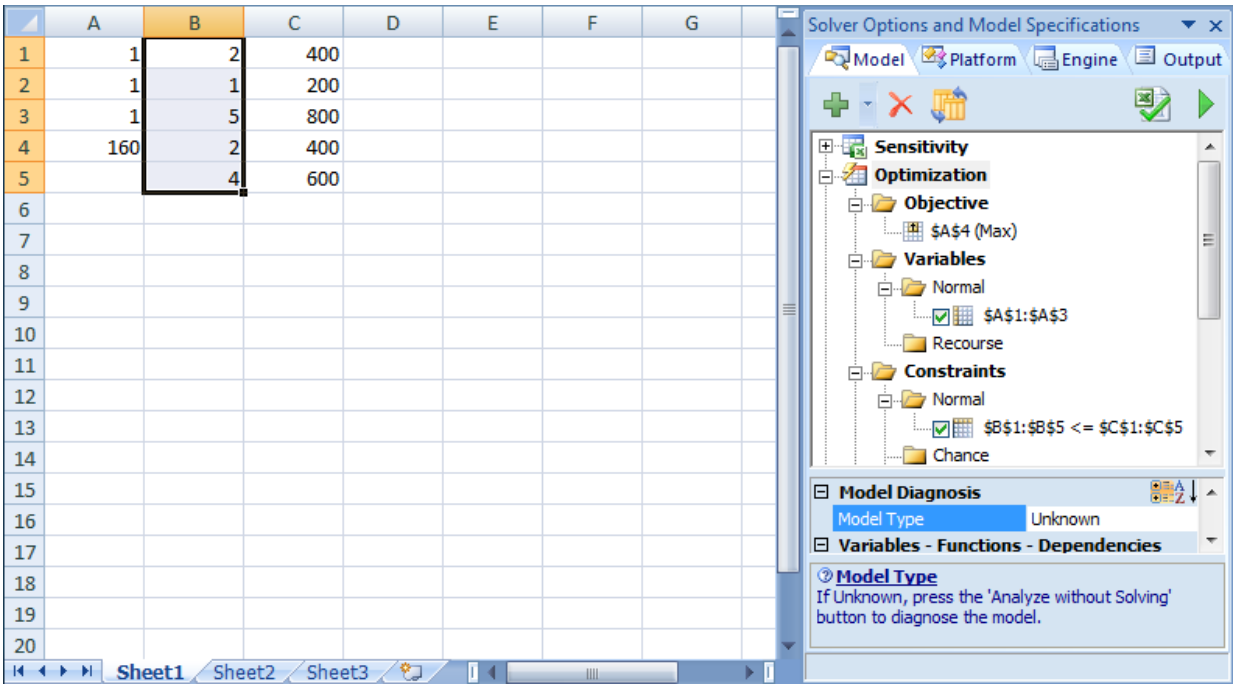


Risk Solver Platform will display its Add Constraint dialog, where you can enter the right hand side(s) as a constant number, a cell range, or a defined name. Click in the **Constraint** edit box, or click the **range selector icon** to its right, and use your mouse to select the cell range C1:C5 for the right hand sides:

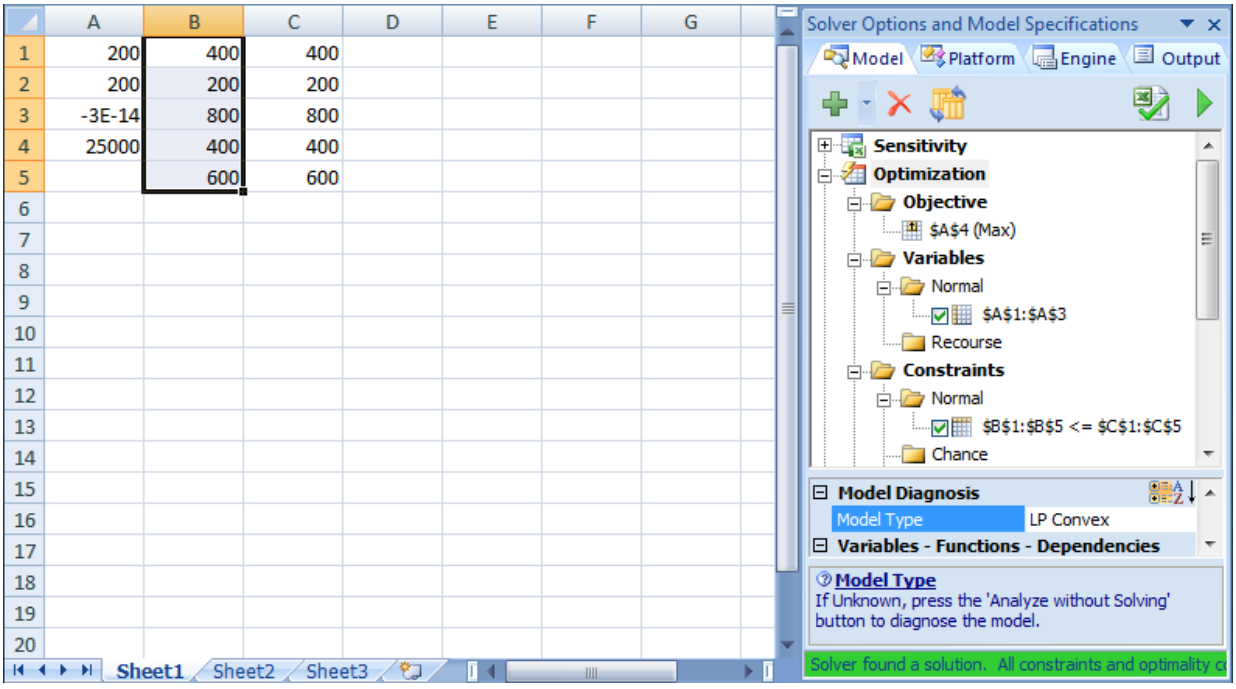


Then click OK. This will define five constraints: B1 <= C1, B2 <= C2, B3 <= C3, B4 <= C4, and B5 <= C5.

The Task Pane Model tab now shows all the elements of the optimization model you've just defined in outline form.



You can solve the model immediately by clicking the **Optimize** button on the Ribbon, or by clicking the **green arrow** at the top right of the Task Pane.



The optimal values of the decision variables at A1:A3, the objective at A4, and the constraints at B1:B5 appear on the worksheet, and a **Solver Result message** ("Solver found a solution. All constraints and optimality conditions are satisfied") appears in green at the bottom of the Task Pane.

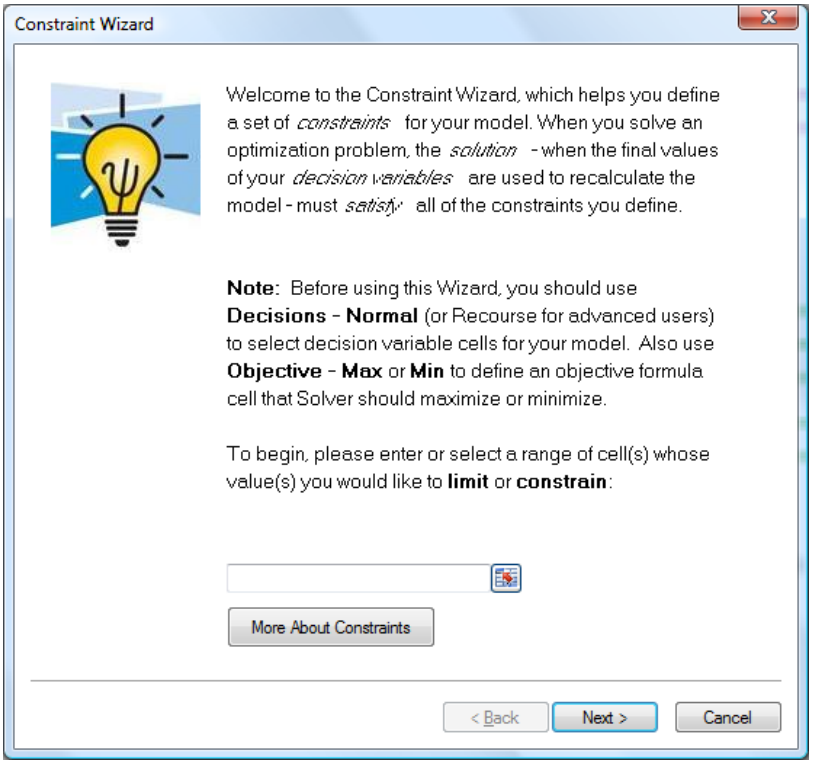
You can view a log of solution messages on the Task Pane Output tab, and you can produce optimization reports by selecting **Reports – Optimization** from the Ribbon. For more information about these features, keep reading!

Using the Constraint Wizard

The Constraint Wizard is a tool to help you define constraints, which for most users is the main (or only) challenging task in defining an optimization model. You can use it to define general constraints on formula cells, bounds or integer constraints on decision variable cells, "group" constraints (such as alldifferent and conic groups), and chance constraints on cells that are subject to uncertainty.

The Constraint Wizard is designed for new users – since it takes you step by step through the process, it's not the fastest way to define a constraint. For constraints that you've defined before, you probably don't need the Wizard. But the first time you use a specific type of constraint, whether it's an integer, alldifferent, or chance constraint, the Wizard can help you make sure you're doing the right thing, and it will also explain how to define such a constraint faster in the future.

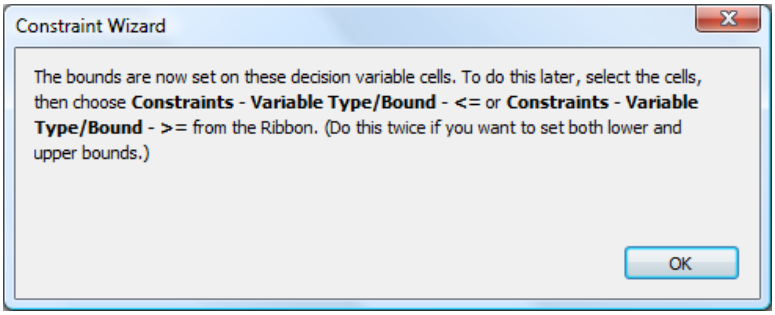
To use the Constraint Wizard, simply select **Constraints – Constraint Wizard** from the Ribbon. In the initial dialog box, select a cell or range of cells whose values you'd like to limit or constrain:



The cell range should include either decision variable cells, or formula cells whose values *depend on* decision variable cells. The Wizard will analyze these cells and their dependencies, and take you step by step when you click **Next**.

Each time you use the Constraint Wizard, and (in the last dialog) click the **Finish** button, you'll define one block of constraints, such as A1 >= 0 or A1:A100 >= 0. You'll see a confirming message such as the one below, which

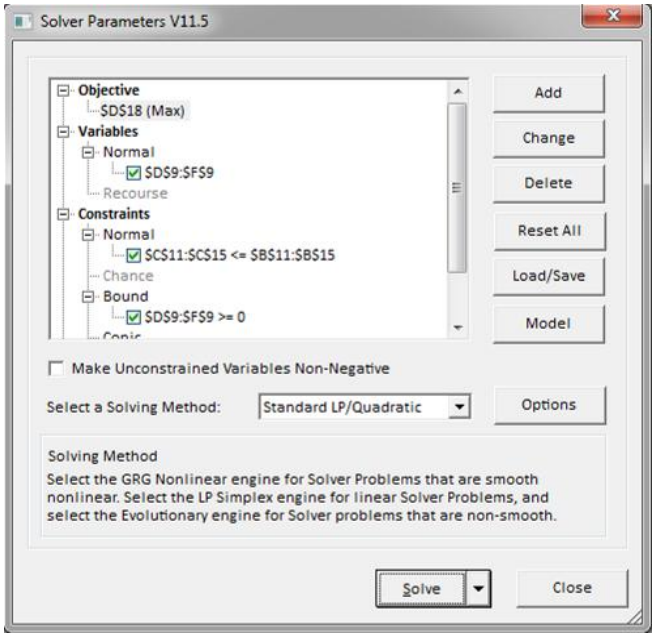
also explains how you can quickly define this type of constraint in the future, by pulling down choices directly from the Ribbon:



You can click the **Cancel** button in any Constraint Wizard dialog, which will leave your model unchanged. So feel free to experiment with the Constraint Wizard!

Using the Classical Solver Parameters Dialog

You can also view – and create or edit – this model through the classical Solver Parameters dialog (as used in the standard Excel Solver). To do so simply click **Premium Solver** on the **Add-Ins** tab on the Ribbon to display the Solver Parameters dialog:



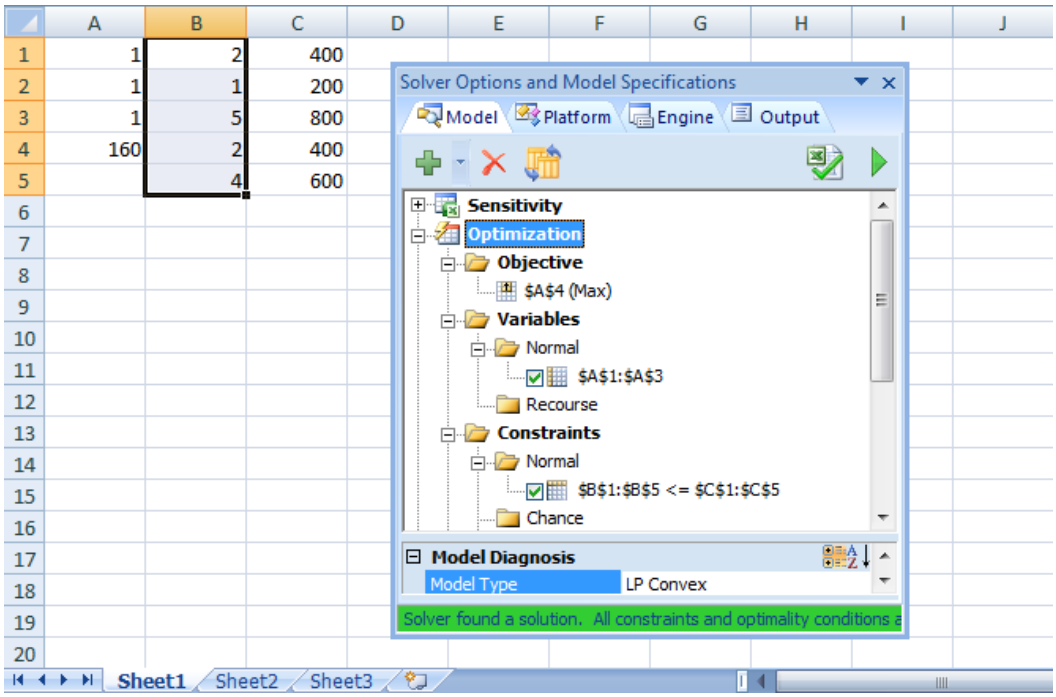
Comparing the Task Pane and Solver Parameters Dialog

As you can see, the Task Pane Model tab and the Solver Parameters dialog contain the same information. But where the Solver Parameters dialog is *modal* (moving the mouse outside the dialog displays a wait cursor – you must close the dialog to do anything else), the Task Pane is *modeless*: You can move the mouse outside the pane, edit formulas on the worksheet, or use other commands.

The Task Pane is initially docked to the right side of the Excel window, but you can select its title bar with your mouse, **drag** it to another position, and **resize** it, as shown on the next page. To "re-dock" the Task Pane, select its title bar with

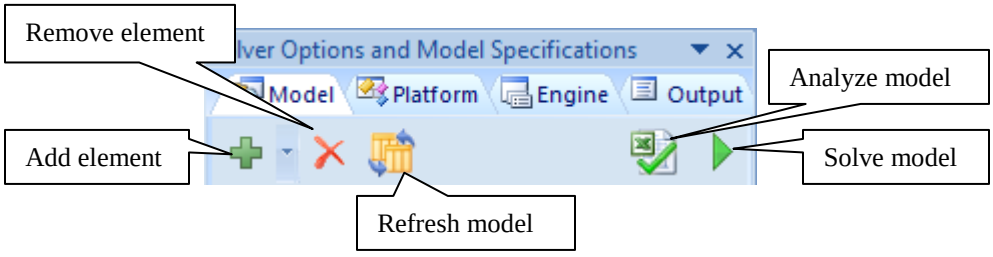
the mouse, drag to a position just beyond the right edge of the Excel window, then release the mouse.

Use the **Model tab** to view your model in outline form, and optionally edit model elements in-place. Use the **Platform tab** to view or change Platform options, such as the number of optimizations or simulations to run, or default bounds on decision variables or uncertain variables. Use the **Engine tab** to select a Solver Engine and view or change its options. Use the **Output tab** to view a log of solution messages, or a chart of the objective values.

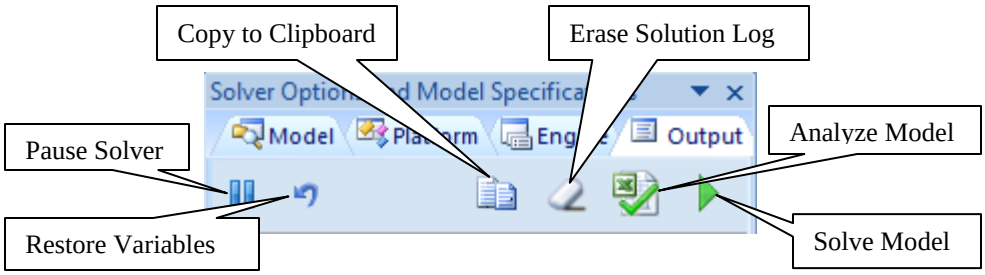


Using Buttons on the Task Pane

Use the **buttons** at the top of the **Model tab** to add or remove model elements (you can also use the Ribbon options to do this, as shown earlier), refresh the model outline when you've made unusual changes to the worksheet, analyze the structure of your model, or solve (run) the optimization or simulation model:



Use the **buttons** at the top of the **Output tab** to pause or stop the Solver, restore the original values of the decision variables, copy the solution message log to the Windows Clipboard (so you can paste the text into another application), erase the solution log, or analyze or solve the model.



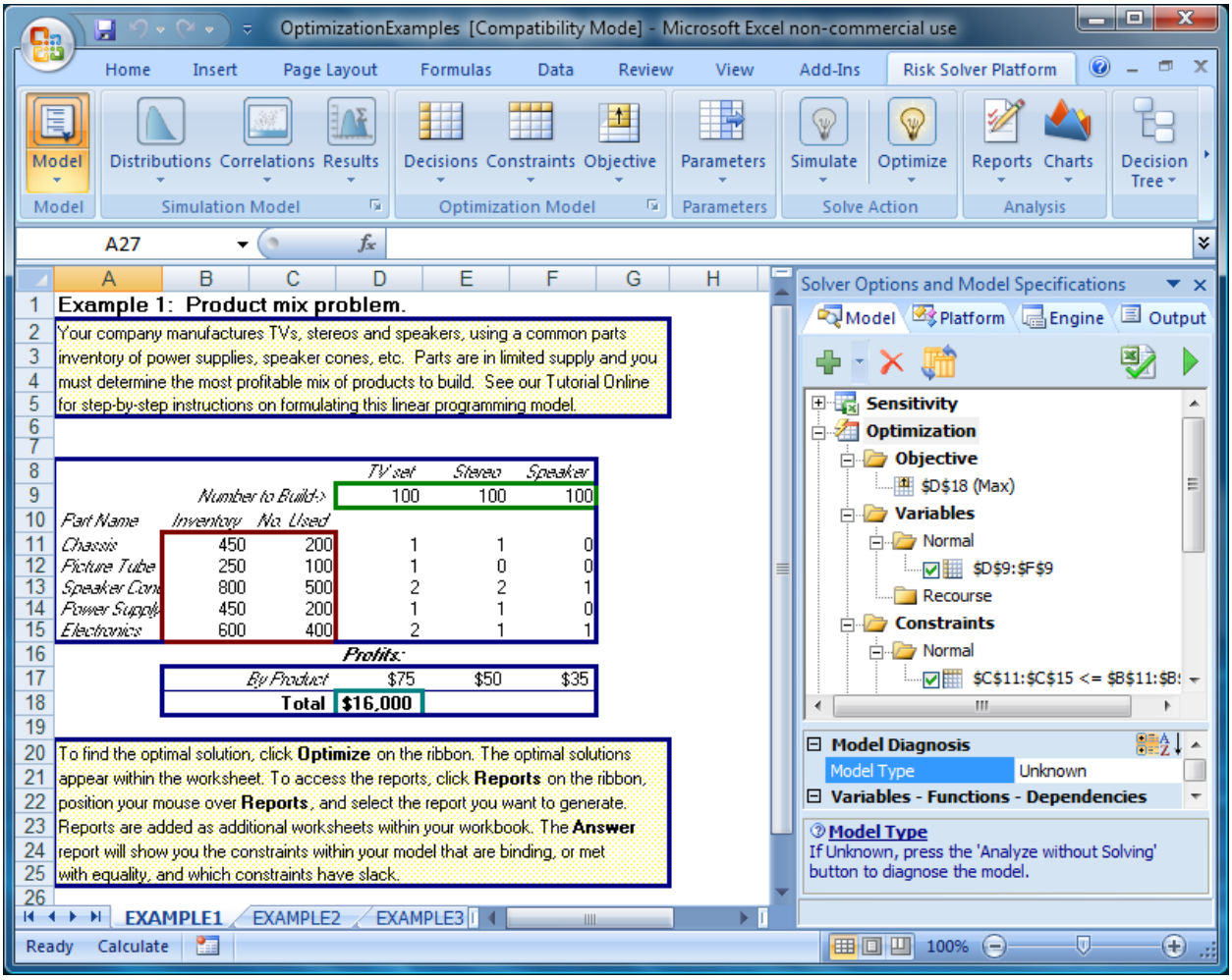
Introducing the Standard Example Models

The approach outlined above in "From Algebra to Spreadsheets" is a '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.

Opening the Examples

To see a better approach to defining this model, click **Help – Examples** on the Ribbon, which opens a workbook Frontline Solver Examples Overview. From there you can click on the Optimization examples text link below the buttons. Which will open the StandardExamples.xls file, showing the first worksheet EXAMPLE1.

An optimization model – like the one we created 'from scratch' in the previous section – is already defined on this worksheet, and appears in the Task Pane Model tab, as shown below. We could *immediately* solve this model.



How did this optimization model appear pre-defined in the workbook? We simply used the Ribbon (or the Solver Parameters dialog) to select the decision variables, objective and constraints, and the Solver engine (and any options or settings it requires). Then we saved the workbook, in the usual way, with Excel's Save command. Models created in the standard Excel Solver, or an earlier version of Premium Solver Pro (formerly called Premium Solver) or Premium Solver Platform, can also be saved this way, and opened and immediately solved in Risk Solver Platform.

More Readable and Expandable Models

Worksheet EXAMPLE1 models the same problem we described in the previous section "From Algebra to Spreadsheets," but the model is organized and laid out for easier readability and expandability.

The decision variables (A1:A3 in the previous section) are in cells D9:F9, and nearby labels show their meaning. The left hand sides of the constraints (B1:B5 in the previous section) are in cells C11:C15, and the right hand sides are in cells B11:B15 – since these constant values can be in any position on the worksheet.

Instead of using constants such as 75, 50 and 35 directly in the formula for the objective – as we did in the previous section – we've put these numbers in cells D17:F17. The formula for the objective is =SUMPRODUCT(D17:F17,D9:F9)

in cell D18. This makes it easier to change product selling prices, or to obtain this data from an external data source, such as a database or accounting data file.

Similarly, we've placed the *coefficients* of the constraints in cells D11:F15. These are exactly the constant values we saw in the algebraic statement of the problem in the previous section:

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)

In EXAMPLE1, the formula at cell C11 – the left hand side of the first constraint – is =SUMPRODUCT(D11:F11,$D$9:$F$9). The formulas at C12:C15 were created by copying C11 downward.

To enhance readability, borders and labels have been used to draw attention to the decision variables, the constraints, and the objective function. If you haven't used these Excel features before, select one or more cells, then choose the Format tab in Excel 2007, or the Format Cells… menu in Excel 2003-2000, to see how to do this.

EXAMPLE1 is also much easier to maintain and expand than a model constructed with 'hardwired' formulas, as in the previous section. You can add products by inserting new columns, and add more parts by inserting rows, and copying any one of the existing left hand side formulas to the new rows; Excel will automatically adjust the formula cell references.

In the next section, we'll solve EXAMPLE1, and create and examine the Answer Report and Sensitivity Report for this model.

Models, Worksheets and Workbooks

There are several worksheets (EXAMPLE1, EXAMPLE2, etc.) in the StandardExamples.xls workbook. As we proceed through this chapter, you'll see that we can simply click a worksheet tab to display a new and different model in the Task Pane on that worksheet. Each of these models was created in the same way, and all of them are saved when you save the Excel workbook.

In StandardExamples.xls, each model is entirely contained on one worksheet. But this is not required – one model can include cells from *many* different worksheets in the same workbook, and it can refer to constant information in other workbooks. By default, models have the same names as the worksheets on which they are created, but you can give *your own names* to models, and create as many of them as you like. This is described in the Frontline Solvers Reference Guide.

Linear Programming Examples

This section takes you on a quick tour of Risk Solver Platform features for linear programming (LP) problems. We highly recommend that you 'follow along' by opening and actually solving the examples in this section.

The first three examples are variants of the Product Mix model introduced earlier in this chapter. 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. Be sure to read "Introducing the Standard Example Models" above.

Using the Output Tab and Creating Reports

Click the tab to display the EXAMPLE1 worksheet. To solve this linear programming problem, click the **Optimize** button on the Ribbon, or click the **green arrow** at the top right of the Task Pane.

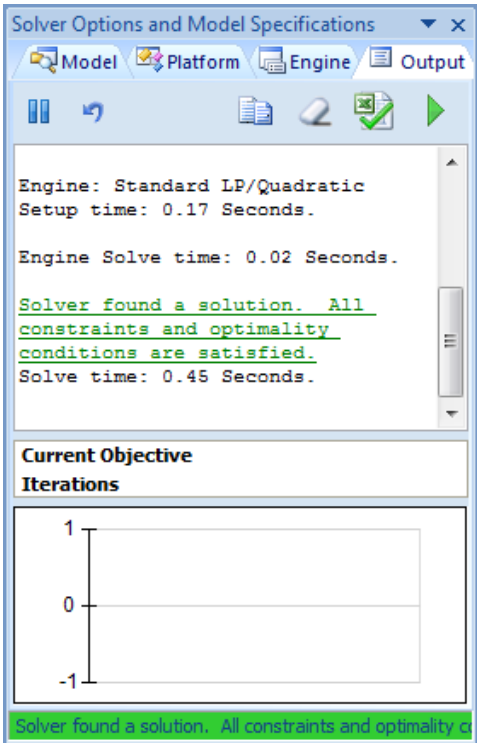
Almost immediately, the solution values appear in the decision variable cells:

			TV set	Stereo	Speaker
		Number to Build->	200	200	0
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
		Profits:			
		By Product	$75	$50	$35
		Total	$25,000		

To maximize profit, we should make 200 TV sets, 200 stereos, and 0 speakers.

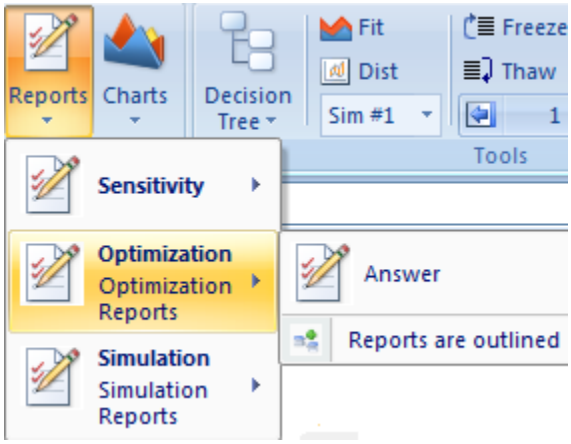
The Solver Results message "Solver found a solution. All constraints and optimality conditions are satisfied" appears in green at the bottom of the Task Pane. **Don't ignore this message**, because it provides critical information about what happened during the solution process, and what you *can* and *cannot* assume about the solution on the worksheet!

To learn more about the solution process, click the **Output** tab at the top of the Task Pane. The solution log in the upper part of this pane reports what happened during the solution process. If you are puzzled by the Solver Result message, simply **click the underlined message** in the log to open online Help to a full explanation of that message.



If solving takes more than a few seconds, the Output tab automatically appears, and current **best objective** and other information, plus a **running chart** of the objective value, appears in the bottom part of the Task Pane.

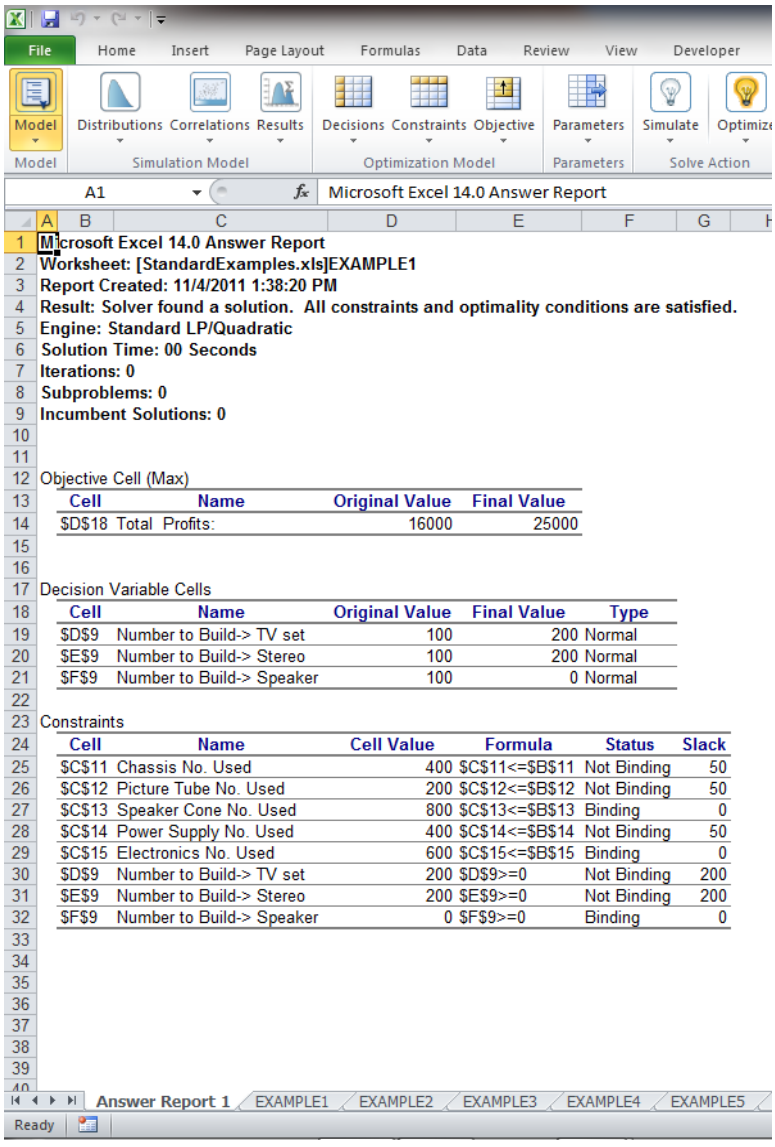
After solving, you can produce reports (such as the Answer Report and Sensitivity Report in the standard Excel Solver) by selecting **Reports – Optimization** from the Ribbon. Unlike the standard Solver, you don't have to choose reports *immediately* in the Solver Results dialog – you can select them later (until you perform another operation that creates new reports).



The Answer and Sensitivity Reports appear as new worksheets, inserted into the workbook just to the left of the EXAMPLE worksheet, as shown on the next page. The Sensitivity Report, which provides *dual values* (shadow prices and reduced costs) and associated *ranges*, is described in Frontline Solvers Reference Guide.

The Answer Report shows the message from the Task Pane Output tab, the Solver engine used, and statistics about the solution process. It shows the beginning and ending values of the objective function and the decision variables. For the constraints, it shows the final cell value and formula, a status (whether the constraint was "binding" – satisfied with equality – at the solution, and a "slack value" for non-binding constraints.

You can ask for **outlined** reports by clicking the "Reports are not outlined" choice on the Ribbon to change it to "Reports are outlined," as shown above. In this case, each block of decision variables and constraints appears in its own outline group. Click the + symbol at the left edge of the report worksheet to show or hide each group in the outline. Any comments that you enter when defining decision variables or constraints appear for each group in the report. These features help you quickly find the information you want in a report created for a large model.



Microsoft Excel 14.0 Answer Report
Worksheet: [StandardExamples.xls]EXAMPLE1
Report Created: 11/4/2011 1:38:20 PM
Result: Solver found a solution. All constraints and optimality conditions are satisfied.
Engine: Standard LP/Quadratic
Solution Time: 00 Seconds
Iterations: 0
Subproblems: 0
Incumbent Solutions: 0

Objective Cell (Max)

Cell	Name	Original Value	Final Value
$D$18	Total Profits:	16000	25000

Decision Variable Cells

Cell	Name	Original Value	Final Value	Type
$D$9	Number to Build-> TV set	100	200	Normal
$E$9	Number to Build-> Stereo	100	200	Normal
$F$9	Number to Build-> Speaker	100	0	Normal

Constraints

Cell	Name	Cell Value	Formula	Status	Slack
$C$11	Chassis No. Used	400	$C$11<=$B$11	Not Binding	50
$C$12	Picture Tube No. Used	200	$C$12<=$B$12	Not Binding	50
$C$13	Speaker Cone No. Used	800	$C$13<=$B$13	Binding	0
$C$14	Power Supply No. Used	400	$C$14<=$B$14	Not Binding	50
$C$15	Electronics No. Used	600	$C$15<=$B$15	Binding	0
$D$9	Number to Build-> TV set	200	$D$9>=0	Not Binding	200
$E$9	Number to Build-> Stereo	200	$E$9>=0	Not Binding	200
$F$9	Number to Build-> Speaker	0	$F$9>=0	Binding	0

A Model with No Feasible Solution

What does it mean when the Solver can find no feasible solution – no combination of values for the decision variables that satisfies all the constraints? It may be that the underlying business problem has no feasible solution – if so, you'd like to focus on the 'hardest-to-satisfy' constraints, to see if you can do anything about them. Or you might have just made a mistake setting up the model – for example choosing <= when you meant to choose >=.

But in a large model, with thousands of constraints, how do you find your mistake, or focus on the 'hardest-to-satisfy' constraints? Risk Solver Platform can help, by producing a Feasibility Report.

Click the tab to display the EXAMPLE2 worksheet. This is the same Product Mix model as in EXAMPLE1, but the supply of Chassis parts at B11 has been changed from 450 to -1. Click the **Optimize** button. The message "Solver could not find a feasible solution" appears in red at the bottom of the Task Pane.

Click **Reports – Optimization – Feasibility** on the Ribbon. A Feasibility Report worksheet is inserted into the workbook, immediately to the left of EXAMPLE2 – with the contents shown on the next page.

Microsoft Excel 14.0 Feasibility Report
Worksheet: [StandardExamples.xls]EXAMPLE2
Report Created: 11/4/2011 1:42:13 PM
Engine: Standard LP/Quadratic

Constraints that Make the Problem Infeasible

Cell	Name	Cell Value	Formula	Status	Slack
$C$11	Chassis No. Used	-1	$C$11<=$B$11	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	Violated	-1

Notice that the report highlights the constraint on Chassis parts used, and the non-negativity constraints on products that are built from Chassis parts. To satisfy the constraint C11 <= B11 where B11 is -1, we'd have to build a negative number of stereos (or TV sets). In a large model with thousands of constraints with no feasible solution, this report can focus your attention on the "trouble spots" – the constraints that make the problem infeasible. The set of constraints shown is called an IIS or *Irreducibly Infeasible Subset* of all the constraints. For more information, please see the Frontline Solvers Reference Guide.

An 'Accidentally' Nonlinear Model

What if you intended to build a linear programming model, but you made a mistake and introduced a nonlinear function into the model? Even if you understand the requirements of linear functions (explained in the chapter "Mastering Conventional Optimization Concepts"), in a large model with thousands of variables and constraints, where each constraint is computed by a chain of formulas, an accidentally introduced nonlinearity may be difficult to find. Sometimes, a first-generation LP model is modified to make a previously fixed parameter into a decision variable – and this leads to an accidental multiplication of two variables.

Click the tab to display the EXAMPLE3 worksheet. This is the same Product Mix model as in EXAMPLE1, but the constraint left hand side formula for

Chassis parts at C11 has been changed to read =SUMPRODUCT(D11:F11,D9:F9)^0.9. (The power function clearly makes the constraint nonlinear.) Click the **Optimize** button. The message "The linearity conditions required by this Solver Engine are not satisfied" appears in red at the bottom of the Task Pane. (Note that we're solving this model with the Standard Simplex or LP/Quadratic Solver.)

Click **Reports – Optimization – Linearity** on the Ribbon. A Linearity Report worksheet is inserted into the workbook, immediately to the left of EXAMPLE3 – with the contents shown on the next page.

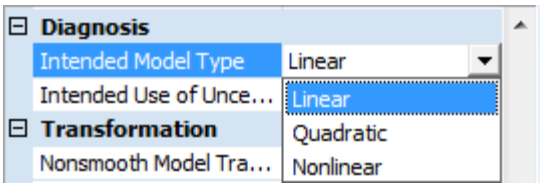
The report highlights the constraint at C11 as a nonlinear function, and variables D9 and E9 that participate in calculation of this function. (F9 does not appear, because its coefficient at F11 is zero.) In a large model, this report can help you quickly find the source of an unintended nonlinearity.

	A	B	C	D	E	F
1	**Microsoft Excel 14.0 Linearity Report**					
2	**Worksheet: [StandardExamples.xls]EXAMPLE3**					
3	**Report Created: 11/4/2011 1:43:38 PM**					
4	**Engine: Standard LP/Quadratic**					
5						
6	Objective Cell (Max)					
7		**Cell**	**Name**	**Original Value**	**Final Value**	**Linear Function**
8		$D$18	Total Profits:	$16,000	$16,000	Yes
9						
10						
11	Decision Variable Cells					
12		**Cell**	**Name**	**Original Value**	**Final Value**	**Occurs Linearly**
13		$D$9	Number to Build-> TV set	100	100	**No**
14		$E$9	Number to Build-> Stereo	100	100	**No**
15		$F$9	Number to Build-> Speaker	100	100	Yes
16						
17	Constraints					
18		**Cell**	**Name**	**Cell Value**	**Formula**	**Linear Function**
19		$C$11	Chassis No. Used	118	$C$11<=$B$11	**No**
20		$C$12	Picture Tube No. Used	100	$C$12<=$B$12	Yes
21		$C$13	Speaker Cone No. Used	500	$C$13<=$B$13	Yes
22		$C$14	Power Supply No. Used	200	$C$14<=$B$14	Yes
23		$C$15	Electronics No. Used	400	$C$15<=$B$15	Yes

The Linearity Report, which is available in all of the Risk Solver Platform subsets except Risk Solver, is very useful, but it does have two drawbacks: (i) It highlights only a *constraint* that behaves nonlinearly – but if this constraint is calculated through a chain of formulas, it may not highlight the exact *cell formula* where the nonlinearity first occurs. (ii) It is based on a *numerical* test for linear or nonlinear behavior – a test that can be 'fooled' by a poorly scaled model. (A 'poorly scaled model' is one that computes values of the objective, constraints, or intermediate results that differ by many orders of magnitude, which can lead to inaccuracy in computer arithmetic.)

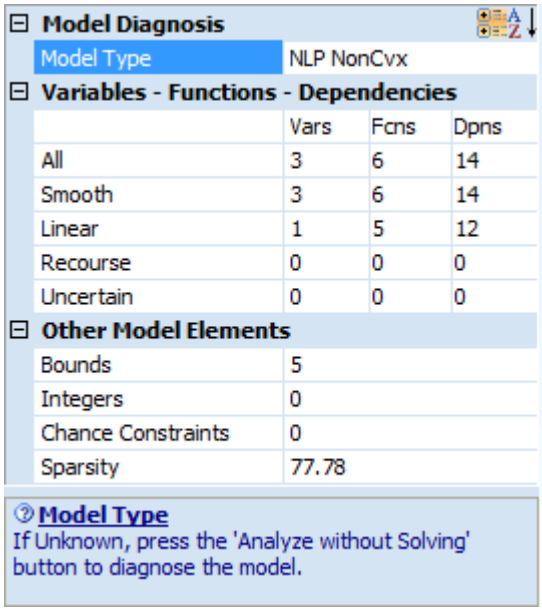
Using the PSI Interpreter to Check for Linearity

If you have Risk Solver Platform or Premium Solver Platform, you can use the Polymorphic Spreadsheet Interpreter to perform a *symbolic* test for linearity that cannot be 'fooled' by scaling problems. Click the tab to display the EXAMPLE3 worksheet. On the Task Pane Platform tab, in the Diagnosis group of options, and ensure that the **Intended Model Type** is set to **Linear**:

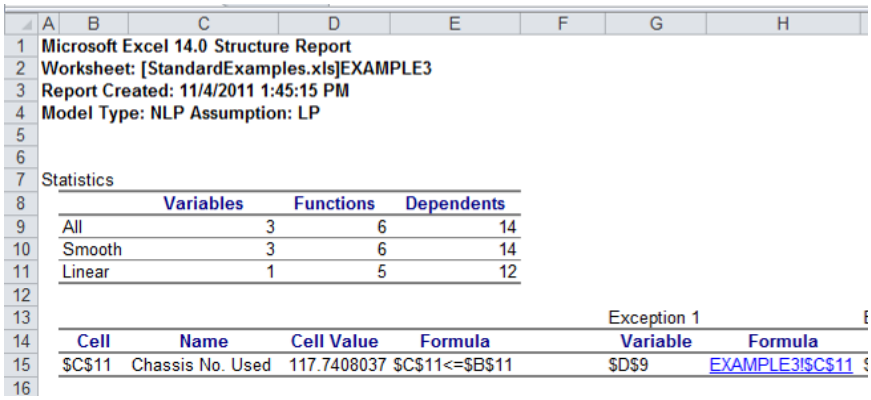


Now click the **Analyze** button in the Task Pane, or select the dropdown choice **Analyze Without Solving** below the **Optimize** button on the Ribbon. The Model Diagnosis portion of the Task Pane will pop up, showing the Model Type and related statistics, as shown on the next page.

Notice that the Model Type is "NLP NonCvx". If you're not sure about the meaning of any of the elements shown here, **click the underlined link** at the bottom of the Task Pane to display Help on that model element.



Now click **Reports – Optimization – Structure** on the Ribbon. A Structure Report is inserted into the workbook, just to the left of EXAMPLE3. This report highlights the formula cells in your worksheet that violate the requirements of your **Intended Model Type** (in this case Linear). In other words, this report shows the formulas that make your model nonlinear.



For EXAMPLE3, this report again highlights the constraint at C11 and the two variables D9 and E9 (E9 in columns I and J, not shown to save space). But

when a constraint or objective is calculated through a chain of formulas, this report normally finds the formula cell that first introduces a nonlinear relationship (such as a power operator, a function such as LOG, or multiplication of two cells that both depend on decision variables). EXAMPLE3!$C$11 in this report can be clicked to jump directly to that cell, so you can quickly inspect its formula.

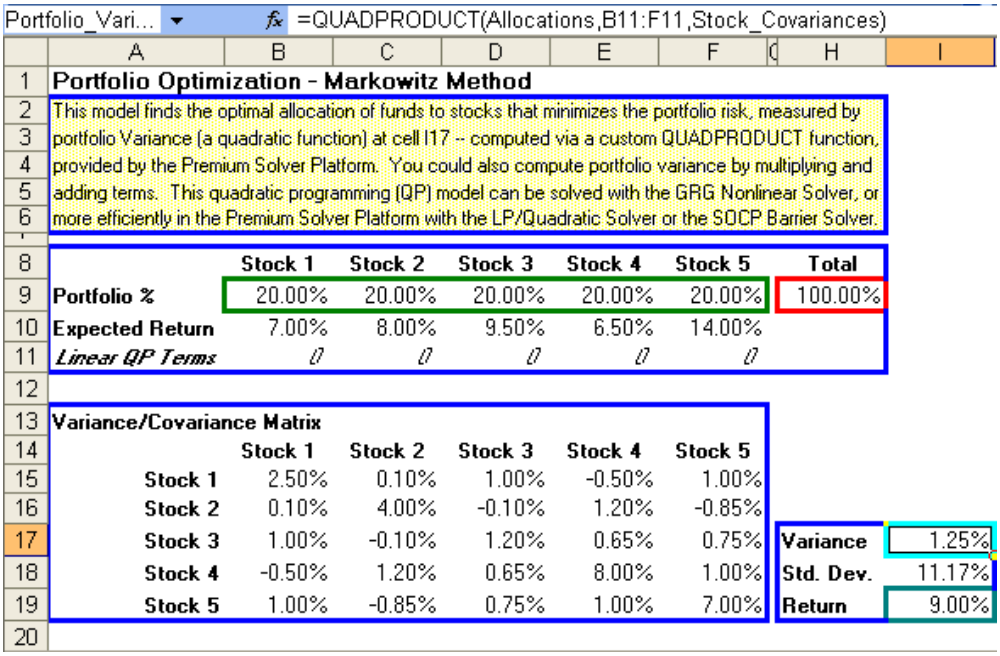
Nonlinear Optimization Examples

This section takes you on a quick tour of Risk Solver Platform features for nonlinear programming (NLP) and non-smooth optimization (NSP) problems, using the SOCP Barrier Solver, LSGRG Nonlinear Solver, and Evolutionary Solver. We highly recommend that you 'follow along' by opening and actually solving the examples in this section.

Portfolio Optimization: Quadratic Programming

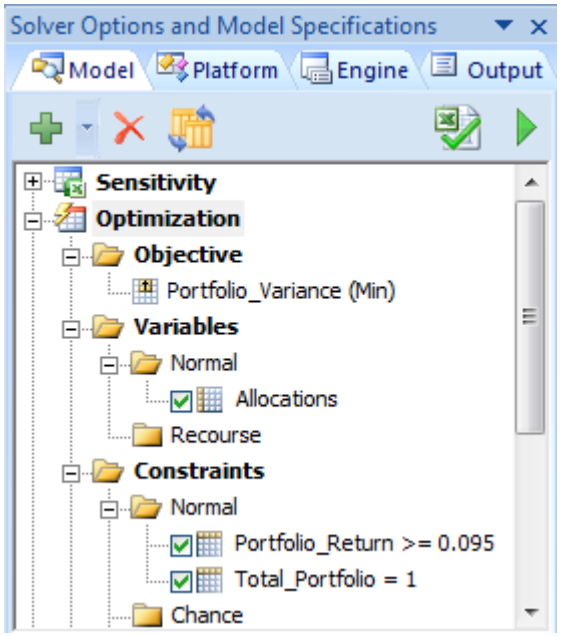
The simplest kind of nonlinear optimization model is a quadratic programming (QP) model, with a quadratic objective and all linear constraints. Such a problem can be solved with a quadratic extension to the linear programming Simplex method – as in the LP/Quadratic Solver in Risk Solver Platform and Premium Solver Platform – or with the LSGRG Nonlinear Solver included in our Platform products and the GRG solver included in Premium Solver Pro.

A classic example of this kind of model, shown in EXAMPLE4 in **StandardExamples.xls**, is a Markowitz-style portfolio optimization problem. In this model, we want to choose the mix of stocks to form an 'efficient portfolio' with the lowest possible risk (measured by portfolio variance) for a specified target rate of return. Click the EXAMPLE4 tab to display the model.



This model uses Excel **defined names** for certain cell ranges: The objective cell I17 has the name Portfolio_Variance, which appears to the left of the formula bar when I17 is selected. Cell I19 has the name Portfolio_Return. The decision

variables – the percentage of funds to invest in each of five stocks – are cells B9:F9, and this cell range has the name Allocations. Cell H9 has the name Total_Portfolio. These defined names appear in the Task Pane Model tab, as shown on the next page.



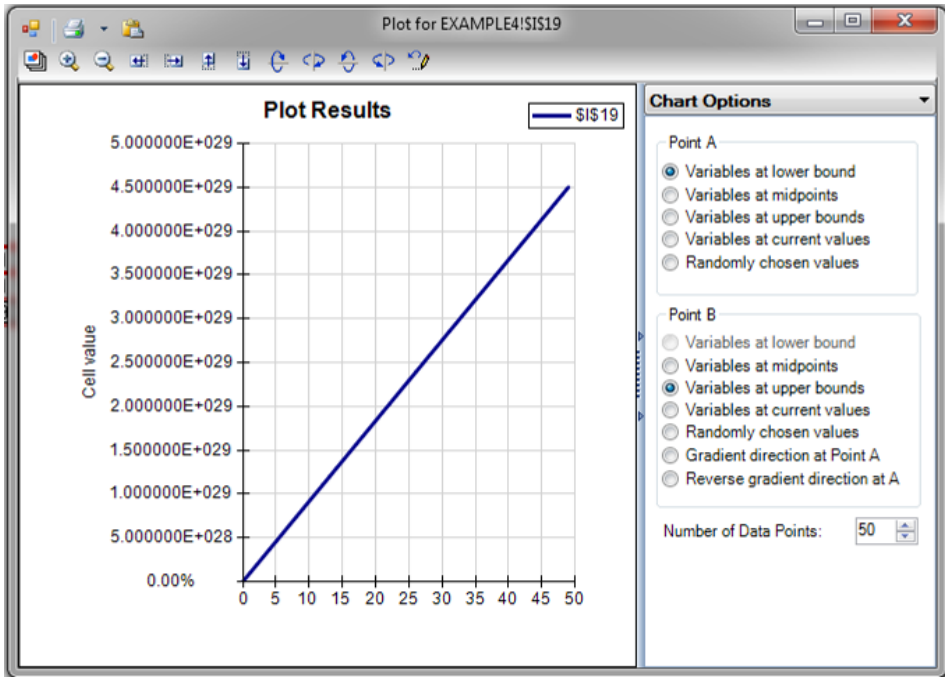
We can see immediately that this model will find values for the stock allocations that minimize portfolio variance, subject to constraints that (i) all funds are used (the sum of the stock allocations is 100% or 1.0) and (ii) the portfolio return is at least 9.5%.

Cell I17 computes the portfolio variance using a special function QUADPRODUCT, which is defined by Risk Solver Platform. This function can be used to compute any 'quadratic form' such as $x^TQx + cx$. In EXAMPLE4, the elements of the matrix Q are the covariances of returns of pairs of stocks, and the elements of the vector c (at B11:F11) are all zero. You could instead compute portfolio variance using a series of Excel formulas using multiplication and addition, but it's more convenient to use QUADPRODUCT.

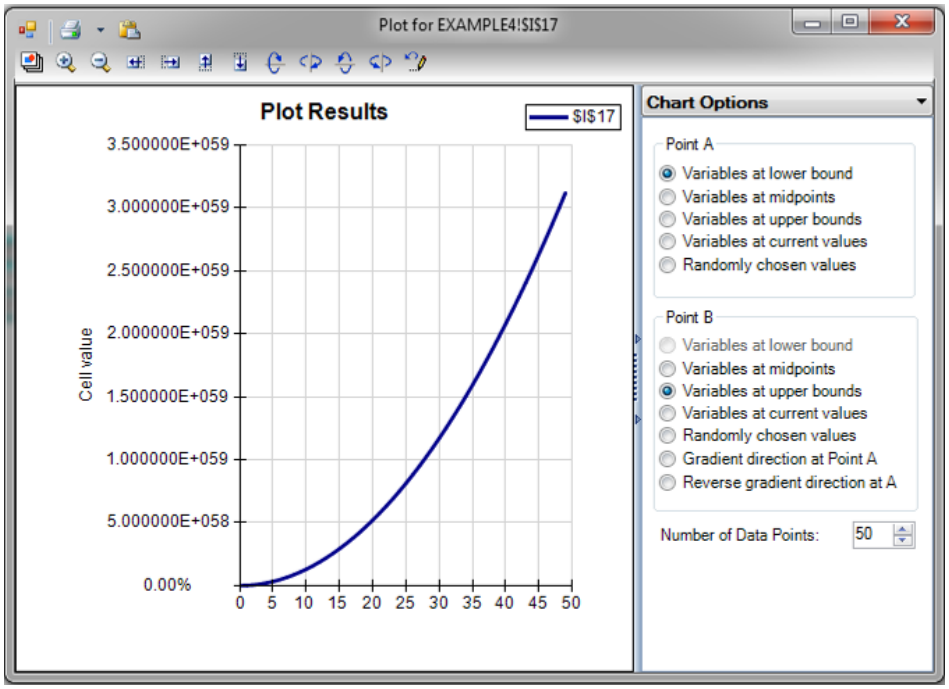
Charts of the Objective and Constraints

Risk Solver Platform can help you visualize the shape of your problem functions (the objective and constraints). If you haven't been sure of the difference between a linear, quadratic, smooth nonlinear, and non-smooth function, you can use Risk Solver Platform to plot it.

As an example, select cell I19 (the Portfolio Return) on the worksheet, and choose **Decisions – Plot** from the Ribbon. This will plot I19 (whose formula is =SUMPRODUCT(Allocations,Stock_Returns)) as a function of the decision variables for Stock 1 through Stock 5, as shown on the next page.



The SUMPRODUCT function is a linear function, so unsurprisingly, Portfolio Return plots as a straight. In contrast, select cell I17 (Portfolio Variance, the QUADPRODUCT function), and choose **Decisions – Plot** from the Ribbon. This function plots as a parabola – as expected for a quadratic function – showing only the positive values, since the decision variables are non-negative.



Both of these plots are actually projections of a function in five dimensions (Stock 1 through Stock 5) onto two dimensions, along a vector from Point A to Point B, shown in the side panel. You have many choices for Points A and B.

To find an optimal allocation of funds to stocks, just click the **Optimize** button. In an instant, the message "Solver found a solution. All constraints and optimality conditions are satisfied" appears in green at the bottom of the Task Pane, the Portfolio Variance is 0.85% and the Portfolio return is 9.5%, and the decision variables contain:

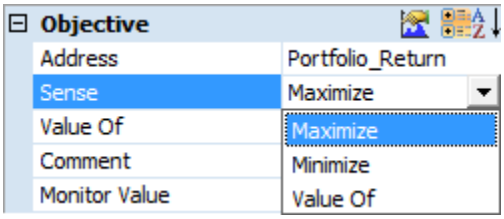
	Stock 1	Stock 2	Stock 3	Stock 4	Stock 5
Portfolio %	1.41%	24.01%	65.80%	0.00%	8.79%
Expected Return	7.00%	8.00%	9.50%	6.50%	14.00%

Solving with the GRG Nonlinear Solver yields the same values (to two decimal places) for the decision variables. If you have Risk Solver Platform or Premium Solver Platform, you can also use the Standard SOCP Barrier Solver – just select it from the dropdown list at the top of the Task Pane Engine tab and click the **Optimize** button again. This yields nearly the same values for the decision variables, Portfolio Return at I19, and Portfolio Variance at I17.

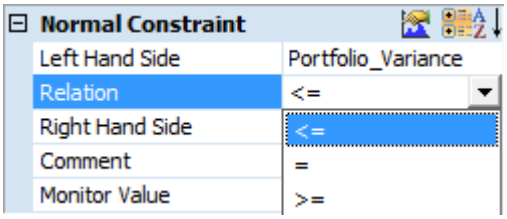
	Stock 1	Stock 2	Stock 3	Stock 4	Stock 5
Portfolio %	1.39%	23.96%	65.74%	0.08%	8.82%
Expected Return	7.00%	8.00%	9.50%	6.50%	14.00%

We can also "turn this problem around" and seek to maximize Portfolio Return, subject to a constraint that Portfolio Variance is (say) no more than 1%. To do this, we can edit the optimization model directly in the Task Pane:

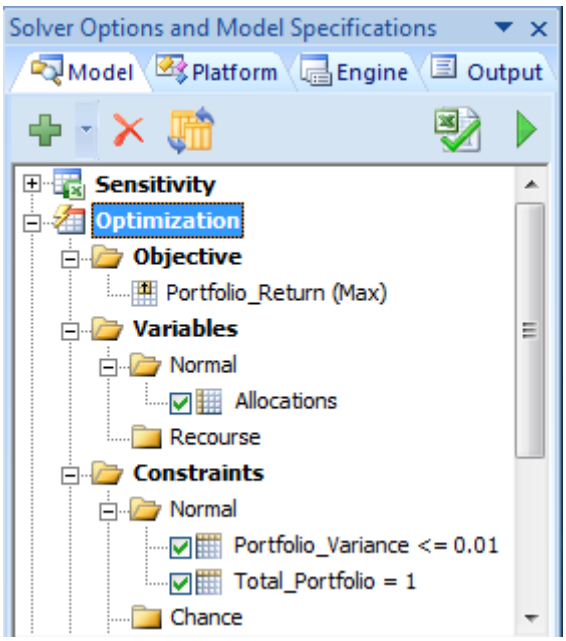
1. Select "Portfolio_Variance (Min)" under Objective in the outline.
2. In the Objective Properties area below the outline, click the **Address** property and type or select cell I19 (or type "Portfolio_Return"). Change the **Sense** dropdown choice from Minimize to Maximize.



3. Select "Portfolio_Return >= 0.095" under Constraints in the outline.
4. In the Normal Constraint Properties area below the outline, click the **Address** property and type or select cell I17 (or type "Portfolio_Variance"), change the Relation dropdown choice to <=, and type 0.01 as the Right Hand Side) value.



The Task Pane Model tab should now appear as shown on the next page. This problem now has a *linear* objective and a *quadratic* constraint; it is called a *quadratically constrained* or QCP problem. It can no longer be solved by the methods used in the LP/Quadratic Solver (a quadratic extension of the Simplex method), but it can be solved by the GRG Nonlinear Solver, or (more efficiently) by the SOCP Barrier Solver.



Solving with the SOCP Barrier Solver yields a Portfolio Return of 10.2%, a Portfolio Variance of 0.1% as expected, and the following values for the decision variables:

	Stock 1	Stock 2	Stock 3	Stock 4	Stock 5
Portfolio %	0.00%	19.33%	58.67%	0.00%	22.00%
Expected Return	7.00%	8.00%	9.50%	6.50%	14.00%

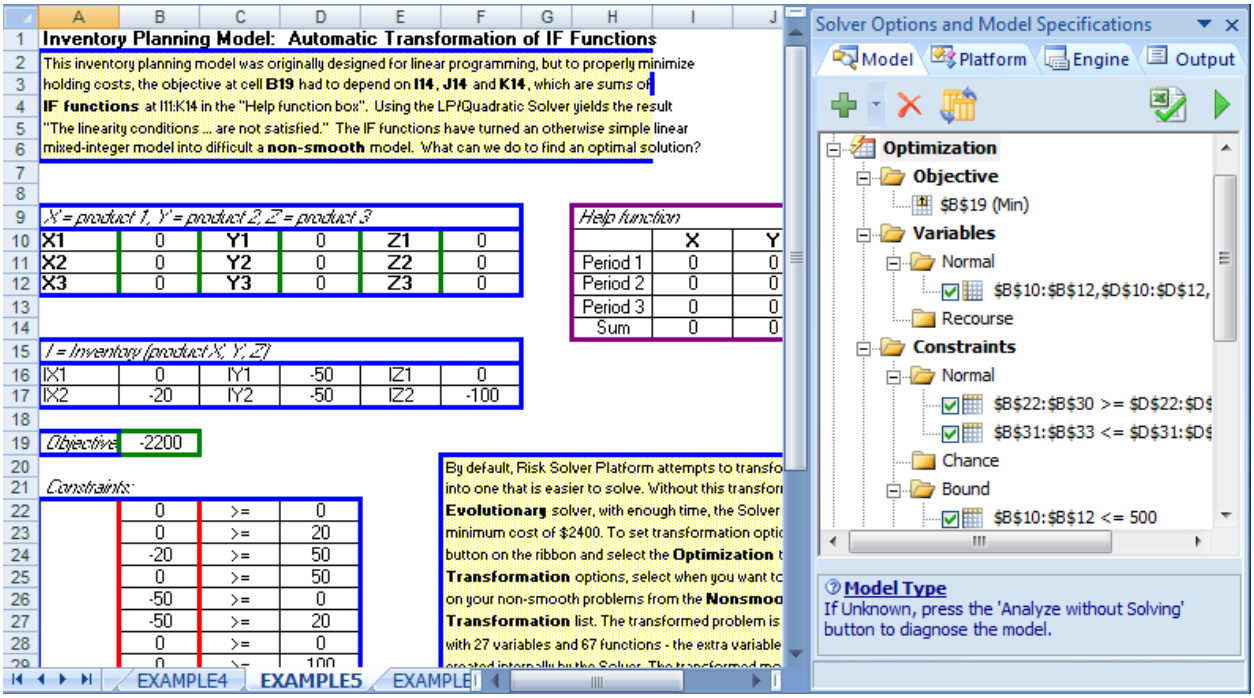
In "Multiple Parameterized Optimizations" in the chapter "Getting Results: Optimization," we return to EXAMPLE4 and show how to automatically vary the Portfolio Return threshold, solve multiple instances of this portfolio optimization model, and create a chart of the efficient frontier.

The original EXAMPLE4 QP model can be 'scaled up' to portfolios of many thousands of stocks, and solved quickly to optimality by the Large-Scale LP/QP Solver, the XPRESS Solver and the MOSEK Solver. The modified QCP model can also be scaled up to portfolios of thousands of stocks, and solved quickly to optimality by the MOSEK Solver.

A Model with IF Functions

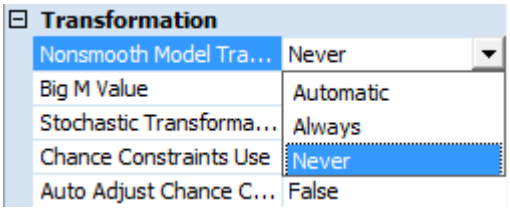
EXAMPLE5 in the **StandardExamples.xls** workbook was adapted from an actual user model, and illustrates a common situation seen by Frontline Systems in technical support. This model was *meant* to be a linear mixed-integer (LP/MIP) model, but the user calculated the objective function based on several IF functions. IF functions are certainly not linear functions – in fact they aren't even smooth nonlinear functions!

One of the remarkable things about Risk Solver Platform is that, in Automatic mode, it will transform this model to a linear mixed-integer model, and solve it to optimality in seconds, without any user intervention. But in this Guide, we'll first see how Risk Solver Platform behaves *without* Automatic mode. Click the EXAMPLE5 tab to display the model.



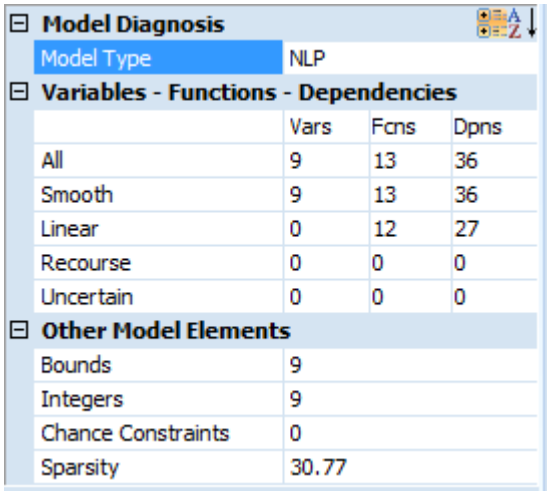
The model has 9 decision variables, all constrained to be integer. The general constraints are linear functions of the decision variables. But the objective depends on I14, J14 and K14, each of which is a sum of three IF functions.

To turn off Automatic mode, in the Task Pane Platform tab **Transformation** group of options, set the **Nonsmooth Model Transformation** option to **Never**.



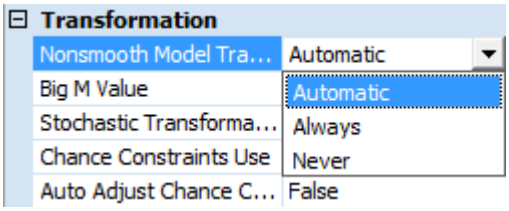
If we now click the **Optimize** button with the LP/Quadratic Solver selected, the message "The linearity conditions required by this Solver engine are not satisfied" appears in red at the bottom of the Task Pane.

If we click the **Analyze** button in the Task Pane, or select the dropdown choice **Analyze Without Solving** below the **Optimize** button on the Ribbon, Risk Solver Platform reports that the model is nonlinear (NLP), as shown on the next page. (Advanced technical point: Since Risk Solver Platform can compute directional derivatives for IF functions, they are treated as *nonlinear* rather than *non-smooth*, unless the Platform tab Advanced group option Require Smooth Functions is set to True.)

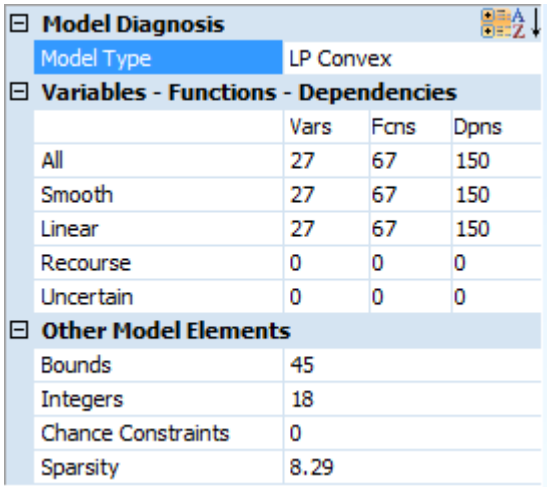


Model Diagnosis			
Model Type	NLP		
Variables - Functions - Dependencies			
	Vars	Fcns	Dpns
All	9	13	36
Smooth	9	13	36
Linear	0	12	27
Recourse	0	0	0
Uncertain	0	0	0
Other Model Elements			
Bounds	9		
Integers	9		
Chance Constraints	0		
Sparsity	30.77		

This type of model commonly arises, but is difficult to solve with the standard Excel Solver, or with Premium Solver Pro. But Risk Solver Platform and Premium Solver Platform can *transform* the model to *eliminate* the IF functions, replacing them with additional binary and continuous variables and linear constraints that have the *same effect* on the model as the IF functions. The additional variables and constraints don't appear on the worksheet, but they are handled internally by the PSI Interpreter and the selected Solver engine. To do this, we simply set the **Nonsmooth Model Transformation** option to Automatic:



If we again **Analyze** the model with this option setting, we find that the *transformed* model has 27 variables (18 of them integer) and 67 constraints, as shown on the next page. But it's now a linear programming (LP) model – so it can be solved by the LP/Quadratic Solver. By solving this transformed model, we can find a solution to the original model on the EXAMPLE4 worksheet.



Model Diagnosis			
Model Type	LP Convex		
Variables - Functions - Dependencies			
	Vars	Fcns	Dpns
All	27	67	150
Smooth	27	67	150
Linear	27	67	150
Recourse	0	0	0
Uncertain	0	0	0
Other Model Elements			
Bounds	45		
Integers	18		
Chance Constraints	0		
Sparsity	8.29		

When we now click the **Optimize** button, the message "Solver found a solution. All constraints and optimality conditions are satisfied" appears in green at the bottom of the Task Pane, and the objective function value at cell B19 is 2400. The message indicates that the Solver has found a **proven optimal** solution for the original problem with IF functions.

The PSI Interpreter can automatically transform models containing IF, AND, OR, NOT, ABS, MIN and MAX, and relations <, <=, >= and > into equivalent models where these non-smooth functions are eliminated. When this transformation is sufficient to make the overall model linear – as it was in this case – you can use the LP/Quadratic Solver or a large-scale LP/MIP Solver engine to solve the problem.

A Model with Cone Constraints

In the EXAMPLE6 and EXAMPLE7 worksheets, we wish to solve a model to optimize the location of an airline hub. The hub will serve six cities, each of which is located at some latitude and longitude (in these models, we use simple X, Y coordinates). We want to choose a location (X, Y coordinate) for the airline hub that will minimize the maximum distance from the hub to any of the six cities. The distance from the hub (X, Y) to a city (Xc, Yc) is SQRT((Xc - X)^2 + (Yc - Y)^2). The model is pictured below. In EXAMPLE6, we model and solve the problem using the GRG Nonlinear Solver. In EXAMPLE7, which requires Risk Solver Platform or Premium Solver Platform, we model the problem using **cone constraints**, and solve it using the SOCP Barrier Solver.

Modeling the Problem with Nonlinear Functions

	A	B	C	D	E	F	G
1	**Optimal Location of Airline Hub - Nonlinear Optimization Model**						
2	You are planning the location of an airline hub that will serve six cities, located at						
3	different points (represented by X, Y coordinates in this simple model). The optimal						
4	location for the hub will minimize the total distance from the hub to each of the cities.						
5	Using the Pythagorean Theorem, the distance from the hub (X, Y) to a city (Xc, Yc)						
6	is SQRT((Xc - X)^2 + (Yc - Y)^2) -- a nonlinear function. In this EXAMPLE6, we						
7	use the GRG Nonlinear Solver to solve the problem as an NLP. In EXAMPLE7, we use						
8	the Premium Solver Platform's Barrier Solver to solve the same problem as an SOCP.						
9							
10							
11					X	Y	
12				**Location of Airline Hub**	1.2500	4.0000	**Distance**
13			**Coordinates**	*Reno*	1	4	0.2500
14				*Sacramento*	0.5	3	1.2500
15				*Salt Lake City*	2	4	0.7500
16				*Phoenix*	2	2	2.1360
17				*Boise*	2	5	1.2500
18				*Seattle*	0.5	6	2.1360
19						**Objective**	**2.1360**
20							
21							
22	To find the optimal solution, select **Tools Solver**..., then click the **Solve** button.						

In EXAMPLE6, our decision variables are the (X, Y) coordinates of the airline hub, at cells D12:E12, and a third decision variable at F19. Cell F19 also serves as the objective function, to be minimized. We calculate the distance from the hub to each of the six cities in cells F13:F18. We define a block of constraints

F13:F18 <= F19 – this means that the distance from the hub to each city must be less than or equal to the decision variable F19, which the Solver will minimize. The effect of these constraints is to cause the Solver to find values for D12:E12 – the (X, Y) coordinates of the hub – that minimize the maximum distance to any of the cities.

Click the EXAMPLE6 tab to display the model, and click the Analyze button. We see that the model is diagnosed as smooth nonlinear (NLP), with 3 variables and 7 functions. Notice that the Standard GRG Nonlinear Solver is selected.

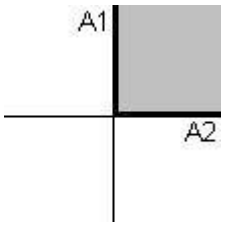
Click the **Optimize** button. The message "Solver found a solution" appears in green at the bottom of the Task Pane. The optimal location for the airline hub is at X = 1.25, Y = 4.0, and the maximum distance from this location to any of the six cities is 2.1360 units.

Modeling the Problem with Cone Constraints

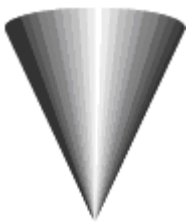
In EXAMPLE4, we solved the simplest kind of nonlinear optimization problem, a portfolio optimization QP model with a quadratic objective and all linear constraints, and a variation that was a QCP model, with a quadratic constraint . We saw that we could solve the problem with the GRG Nonlinear Solver, but we could also solve both variations of the problem with the **SOCP Barrier Solver** – and we learned that we could more easily 'scale up' the SOCP problem to large size, with thousands of stocks or more.

We can do the same thing with the airline hub problem. The model is a little more complex, with more variables and constraints – but this allows the Solver to exploit the structure of the model, and solve it more quickly and reliably. To do this, we will explicitly use cone constraints.

We have actually used a simple kind of cone constraint before – in EXAMPLE1, our first linear programming problem, at the beginning of this chapter. In that model, we used an objective and constraints that were linear functions of the decision variables, and *non-negativity* constraints on the variables. These constraints specified that the variables must lie within a simple kind of cone, called the *non-negative orthant*. This first order cone places a bound on the *L1*-norm of the vector of decision variables:



In our reformulation of the airline hub problem in EXAMPLE7, we'll specify that groups of variables must lie within a *second order cone* (also called a Lorentz cone, or "ice cream cone") – a ***convex*** set that looks like this:



This cone places a bound on the *L2*-norm of the vector of decision variables. If A1:A3 are variables that lie within this cone, then A1 >= SQRT(SUMSQ(A2:A3)) must hold. A problem with a linear objective and linear or second order cone (SOC) constraints is called a *second order cone programming* (SOCP) problem; it is always a ***convex*** optimization problem.

Second order cone programming is the *natural generalization* of linear programming. It offers the same advantages of convexity and scalability to large problems offered by linear programming – but for a broader class of models. For history buffs, Premium Solver Platform V6.0 was the first commercial software product to offer broad support for second order cone programming.

The EXAMPLE7 worksheet is shown below. In this model, we compute the simple *differences* Xc – X and Yc – Y between the airline hub coordinates and the coordinates of each city. We define new variables F21:F26, G21:G26, and H21:H26. We use constraints to make G21:G26 equal to the Xc – X differences, and H21:H26 equal to the Yc – Y differences.

Then we specify that each set of three variables (for example F21, G21, H21) must belong to a *second order cone* – hence F21 >= SQRT(SUMSQ(G21:H21)) – and likewise for F22, G22, H22, etc. Finally, we use one more variable F27 as the objective to be minimized, and we add a block of constraints F21:F26 <= F27 – much as we did in EXAMPLE6. The effect of these constraints is to cause the Solver to find values for the (X, Y) coordinates of the hub – now at D14:D15 – that minimize the maximum distance to any of the cities.

	A	B	C	D	E	F	G	H	I
1	**Optimal Location of Airline Hub - Conic Optimization Model**								
2	As in EXAMPLE6, you are planning the location of an airline hub that will serve six cities, located at different points								
3	(represented by X, Y coordinates in this simple model). The optimal location for the hub will minimize the total distance from								
4	the hub to each of the cities. In this model, we simply compute the difference between the X coordinates of the hub and								
5	each city at cells D21:D26, and the difference between the Y coordinates at cells E21:E26. We introduce new variables at								
6	F21:H26, and a new variable at cell F27. The variables F21:F26 are each constrained to be less than F27, which will be								
7	minimized. Constraints also make variables G21:G26 equal to the X differences, and variables H21:H26 equal to the Y								
8	differences. Finally, we specify that each set of three variables (F21:H21, F22:H22, etc.) must belong to the second order								
9	cone of dimension 3. By minimizing variable F27, we find the X, Y coordinates for the hub that minimizes the sum of the								
10	distances from the hub to the different cities. This SOCP model can be solved faster and more reliably than the NLP model.								
11									
12									
13					X	Y			
14			**Location of Airline Hub**		1.2501	4.0000			
15			**Coordinates**	*Reno*	1	4			
16				*Sacramento*	0.5	3			
17				*Salt Lake City*	2	4			
18				*Phoenix*	2	2			
19				*Boise*	2	5			
20				*Seattle*	0.5	6		**Cone Constraints**	
21			**Distance**	*Reno*	-0.2501	0.0000	1.1005	-0.2501	0.0000
22				*Sacramento*	-0.7501	-1.0000	1.5781	-0.7501	-1.0000
23				*Salt Lake City*	0.7499	0.0000	1.1863	0.7499	0.0000
24				*Phoenix*	0.7499	-2.0000	2.1360	0.7499	-2.0000
25				*Boise*	0.7499	1.0000	1.5367	0.7499	1.0000
26				*Seattle*	-0.7501	2.0000	2.1360	-0.7501	2.0000
27							**2.1360**	**Objective**	
28									
29									
30	To find the optimal solution, select **Tools Solver**..., then click the **Solve** button.								

Click the EXAMPLE7 tab to display the model, and click the **Analyze** button. We see that this formulation of the model is diagnosed as second order cone

programming problem (SOCP), with 21 variables and 19 functions. Notice that the SOCP Barrier Solver is selected to solve the problem.

Click the **Optimize** button. After a moment, the same optimal solution as in EXAMPLE6 is found. In this simple example, both the Standard GRG Nonlinear Solver and the SOCP Barrier Solver solve the problem in sub-second time. But if we were to **scale up** this model to large size – say with 1,000 cities – the SOCP Barrier Solver would likely show a speed advantage.

Examples: Simulation and Risk Analysis

Introduction

This chapter introduces simulation and risk analysis in Risk Solver Platform, with a series of examples. Each of these examples with the exception of the last one can be used in Risk Solver.

- "A First Simulation Example" takes you step-by-step through the process of building and analyzing a risk analysis model using Monte Carlo simulation.
- "Airline Revenue Management Example" takes you through three versions of an airline yield management problem, to illustrate a single simulation, multiple parameterized simulations, and simulation optimization.

A First Simulation Example

Building a simulation model in Risk Solver Platform is straightforward: You simply build a conventional spreadsheet model, designed for 'what-if' analysis. Next, you identify the ***inputs*** to your model that are uncertain, and use PSI Distribution functions to describe the uncertainty. Then, you identify the ***outputs*** of special interest (such as Net Profit), and use PSI Statistics functions to examine or summarize how they behave in light of the uncertainty.

To open this example, click **Help – Examples** on the Ribbon, which opens a workbook with a link to Simulation Examples. In that workbook click the link to open example BusinessForecast.xls. We'll build the simulation model step-by-step in this section, but you can open the completed model as BusinessForecastPsi.xls if you like.

Uncertain Variables

In any problem, there are factors or inputs that you can control – for example, the price you set for a product, and factors or inputs that you *cannot* control – for example, customer demand, interest rates, etc. Risk Solver Platform uses **uncertain variables** (*random variables* in mathematics) to represent inputs that are uncertain and beyond your control. (It uses decision variables to represent factors or inputs that you *can* control.)

Uncertain Functions

You will also have outputs or results of interest – such as Net Profit – that you can compute, using formulas that depend on the factors influencing the problem – possibly both decision variables and uncertain variables. We'll use the term **uncertain functions** for quantities whose calculation depends on uncertain variables (in mathematics these are called *functions of random variables*).

A Business Planning Example

We'll illustrate the process of building a simulation model step by step, using a simple business planning example. Imagine you are the marketing manager for a firm that is planning to introduce a new product. You need to estimate the first year profit from this product, which will depend on:

- Sales in units
- Price per unit sold
- Unit manufacturing cost
- Fixed costs and overhead

Profit will be calculated as **Profit = Sales * (Price - Unit cost) - Fixed costs**.

Fixed costs are known to be **$120,000**. But the other factors all involve some uncertainty. Sales in units can cover quite a range, and the selling price per unit will depend on competitor actions. Unit manufacturing costs will also vary depending on vendor prices and production experience.

Uncertain Variables: Sales and Price

Based on your market research, you believe that there are equal chances that the market demand will be Slow, OK, or Hot for this product:

- In the Slow market demand scenario, you expect to sell 5**0,000** units at an average price of **$11.00** per unit.
- In the OK market scenario, you expect to sell **75,000** units, but you'll likely realize a lower average selling price of **$10.00** per unit.
- In the Hot market scenario, you expect to sell **100,000** units, but this will bring in competitors who will drive down the average selling price to **$8.00** per unit.

Since the scenarios are equally likely, your *average* volume is **75,000** units, and your *average* price per unit is **$9.67**. But think: How likely is this *average* case? (Will it ever actually occur?)

Uncertain Variables: Unit Cost

Your firm's production manager advises you that unit costs may be anywhere from **$5.50** to **$7.50**, with a most likely cost of **$6.50**. The most likely cost is also the *average* cost.

Uncertain Function: Net Profit

Net Profit is calculated as **Profit = Sales * (Price - Unit cost) -Fixed costs**. Sales, Price and Unit costs are all uncertain variables, so Net Profit is an uncertain function.

A What-If Spreadsheet Model

At this point, you can summarize the problem in the Excel model pictured below, which calculates Net Profit based on *average* sales, price, and unit cost. This model is in the your Examples folder, named **BusinessForecast.xls**.

B11 =B4*(B5-B6)-B7

	A	B	C	D	E	F
1						
2	**Financial Forecast**					
3				**Sales Scenarios**	**Volume**	**Price**
4	Sales Volume	75,000				
5	Selling Price	$9.67		Hot Market	100,000	$8.00
6	Unit Cost	$6.50		OK Market	75,000	$10.00
7	Fixed Costs	$120,000		Slow Market	50,000	$11.00
8						
9				**Cost Scenarios**		
10						
11	Net Profit	$117,750		Minimum Cost	$5.50	
12	True Average			Most Likely Cost	$6.50	
13				Maximum Cost	$7.50	
14						

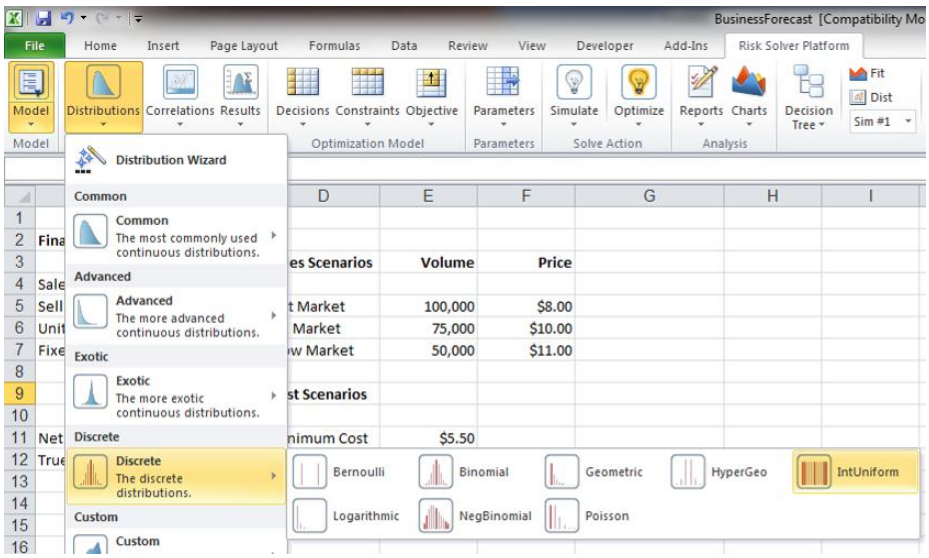
The Net Profit figure of $117,750 (=B4*(B5-B6)-B7) calculated by this model, based on *average* values for the uncertain factors, is quite misleading, as we'll see in a moment. The ***true average*** Net Profit is closer to $93,000! In the spirit of Prof. Sam Savage's new book *The Flaw of Averages*, we'll refer to the model above as the Flawed Average model.

Defining a Simulation Model

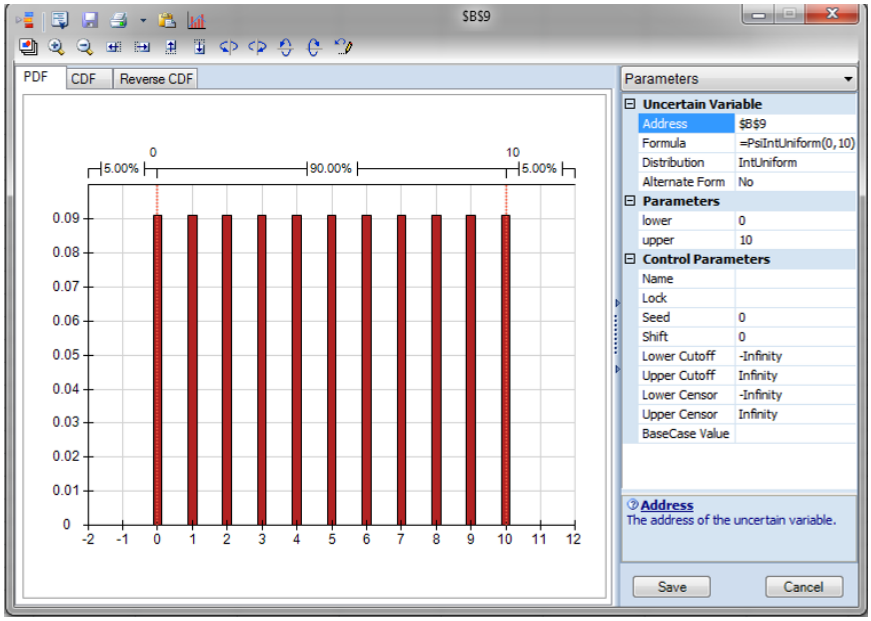
To "stress test" this model, we need to replace the fixed Sales, Price and Unit cost amounts with *variable* amounts that reflect their uncertainty.

Since there are equal chances that the market will be will be Slow, OK, or Hot, we want to create an uncertain variable that selects among these three possibilities, by drawing a random number – say 1, 2 or 3 – with equal probability. We can do this easily in Risk Solver Platform using an **integer uniform** probability distribution. We'll then base our Sales Volume and Selling Price on this uncertain variable.

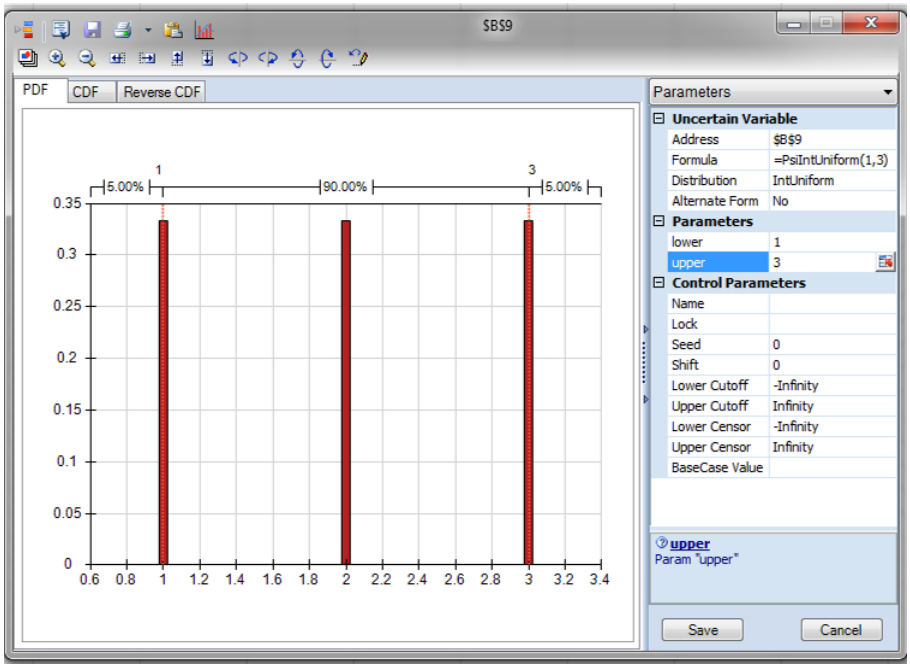
We'll choose a currently empty cell, B9, to hold this distribution by clicking on it, we then select **Distributions - Discrete** on the Ribbon. This displays a gallery of discrete probability distributions, as shown on the next page (some of the gallery choices, including IntUniform, are cut off from view on the right). A sample drawn from a *discrete* distribution is always one of a set of discrete values, such as integer numbers.



We click to choose **IntUniform** from the gallery (again off the right edge in the picture above). Risk Solver Platform displays the Uncertain Variable dialog for a integer uniform distribution, initially with parameters lower 0 and upper 10:



We click in the Value column and change the parameters to read **lower** 1 and **upper** 3. This means that on each trial, we'll draw a number 1, 2 or 3 from this distribution. As we do, the chart of probability mass (density) is updated:



When we click the **Save** icon in the dialog toolbar, a formula **=PsiIntUniform(1,3)** is written to B9. B9 is now an *uncertain variable*. If we press F9 to recalculate the spreadsheet, a different value – either 1, 2 or 3 – appears each time in cell B9. When we run a Monte Carlo simulation of (say) 1,000 trials, 1,000 different values of 1, 2 or 3 will be sampled for this cell's value.

Now, we need to select one of the three sales scenarios in formulas for Sales Volume and Selling Price. With cell B4 selected, we enter the formula:

=CHOOSE(B9,E5,E6,E7) for Sales Volume

This will cause B4 to return 100,000, 75,000, or 50,000, depending on the value in B9. Next, with cell B5 selected, we enter the formula:

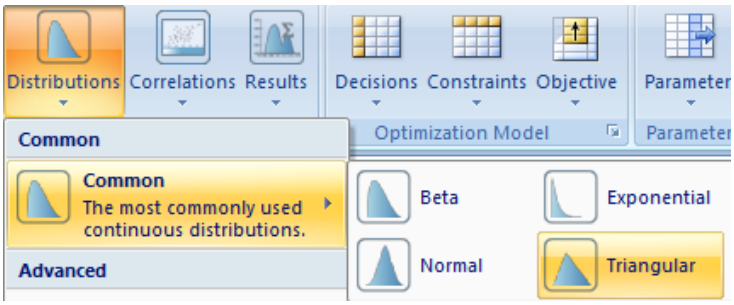
=CHOOSE(B9,F5,F6,F7) for Selling Price

This will cause B5 to return $8, $10 or $11, depending on the value in B9.

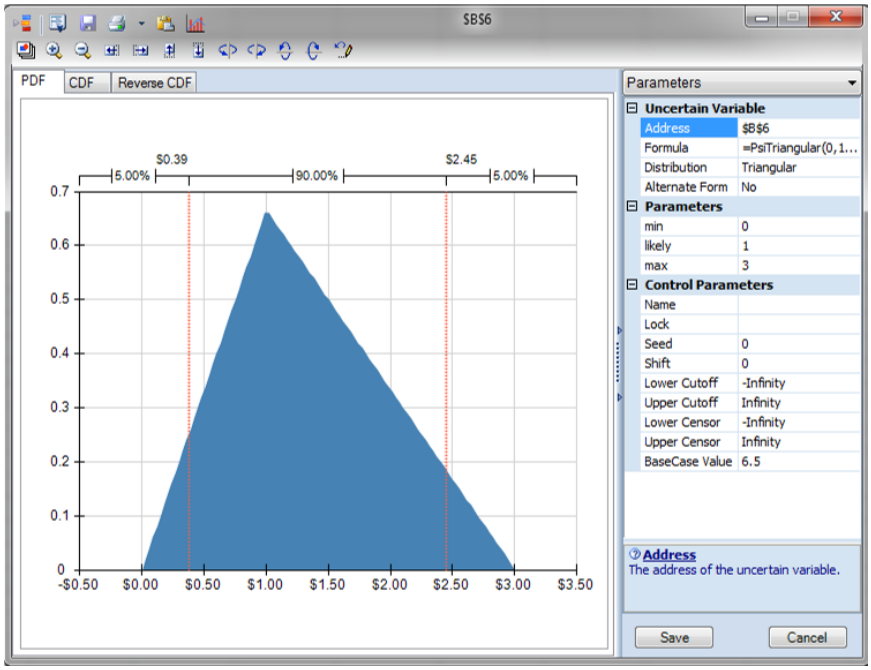
Notice that the values returned by B4 and B5 are related, or ***correlated***: Higher sales volume is accompanied by lower selling prices, and vice versa. If we had scenarios with 100,000 units sold at $11 each, our model would be unrealistic. Risk Solver Platform has more versatile ways to specify correlation between uncertain variables, but this approach is easy to understand in this example.

Next, we'll deal with Unit Cost. We have not just three, but *many* possible values for this variable: It can be anywhere from $5.50 to $7.50, with a most likely cost of $6.50. A crude but effective way to model this is to use a *triangular* distribution. Risk Solver Platform provides a function called **PsiTriangular()** for this distribution.

With cell B6 selected, we select **Distributions – Common – Triangular** from the Ribbon's dropdown gallery, as shown on the next page.



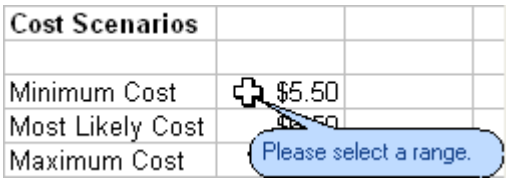
(Unlike a discrete distribution, a sample drawn from a *continuous* distribution can be any numeric value, such as 5.8 or 6.01, in a range.) Risk Solver Platform displays the Uncertain Variable dialog with a chart of the triangular distribution, initially from 0 to 3, peaking at 1.



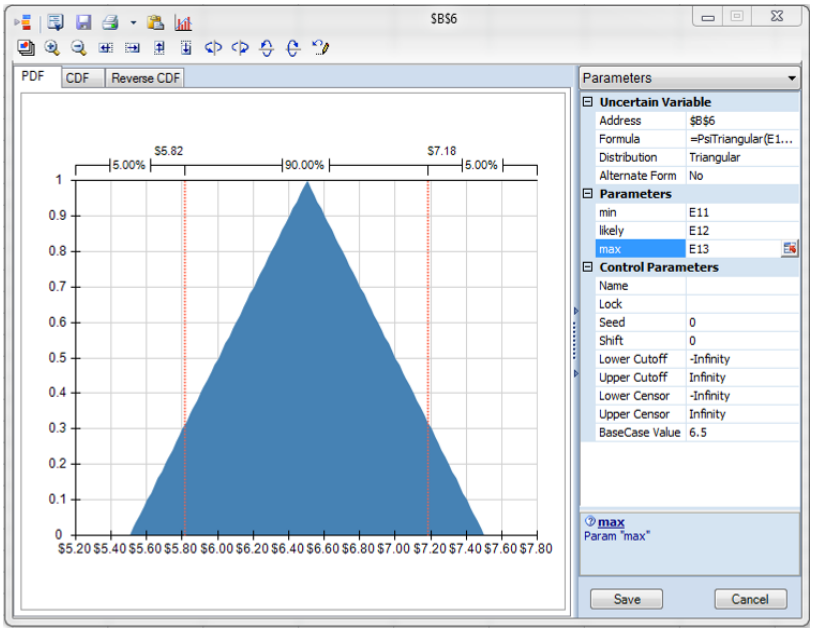
To see more of the features of this dialog, try clicking on the Parameters drop down at the top of the right panel.

The minimum, most likely, and maximum values for Unit Cost are in cells E11, E12 and E13. If we enter these *cell references* – not just the numbers 5.50, 6.50 and 7.50 – into our PsiTriangular() function call, we'll have a more flexible model that we can use in *Interactive Simulation*, as described below. To start, we simply click in the Value field for the **min (a)** parameter.

Now we can either type in a number or cell reference, or we can click the  button at the right edge of the field: When we do this, a small "balloon" follows the mouse pointer as we point and click on cell E11, as shown below.



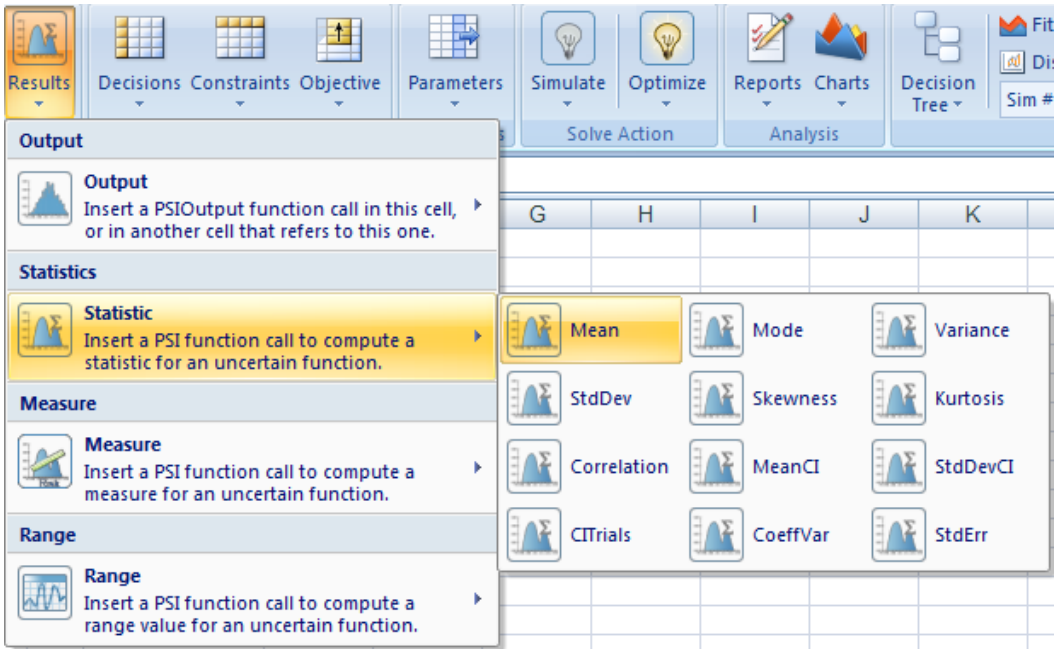
Now E11 appears as the **min (a)** parameter. Since at this moment, the **min (a)** value of 5.50 is greater than the **max (b)** value of 3, the parameter turns red – this is a hint that the parameters are inconsistent. The red disappears once we enter references to cell E12 for **likely (c)** and E13 for **max (b)**. When we click the **Save** icon, **=PsiTriangular(E11,E12,E13)** is inserted into cell B6.



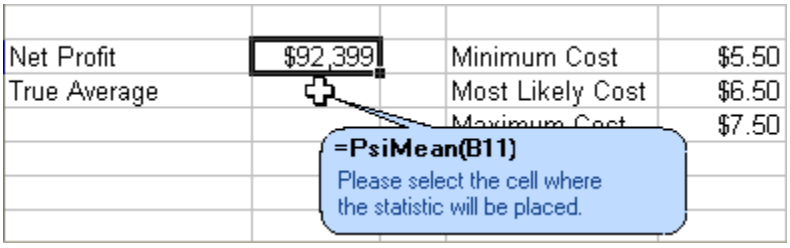
We've now defined the **uncertain variables** in our model. Anything calculated from these uncertain variables is an **uncertain function**, but usually we're interested only in specific results such as Net Profit at cell B11. When we "turn on" Interactive Simulation, B11 will effectively hold an **array of values**, each one calculated from different values sampled for B4, B5, B6 and B7.

What would we like to know about the array of values for Net Profit at cell B11? The simplest summary result is the average (or mean) Net Profit. Note that this will be the ***true average*** of Net Profit across 1,000 or more scenarios or trials – not a single calculation from *average* values of the *inputs*.

With cell B11 selected, we select **Results – Statistics** from the Ribbon. A dropdown gallery shows us the available statistics functions, which we can 'drag and drop" into a worksheet cell.



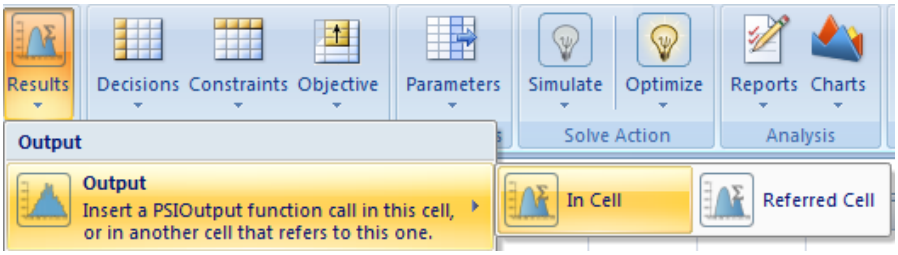
When we click the **Mean** button, a small "balloon" like the one below appears and follows the mouse pointer as we move to a worksheet cell, in this case B12. When we click to "drop," the formula **=PsiMean(B11)** appears in the cell.



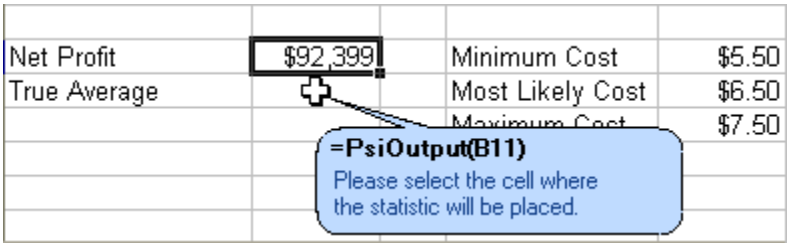
Selecting Uncertain Functions

When we define a summary statistic, such as =PsiMean(B11), we've implicitly designated cell B11 as an **uncertain function**. Risk Solver Platform will keep track of the full range of trial values for B11 during a simulation, and will display frequency and sensitivity charts, statistics and percentiles for it on demand. As noted above, in principle anything calculated from the uncertain variables is an "uncertain function" – but to save time and memory, Risk Solver Platform keeps track of trial values only for the formula cells that we ***designate*** as uncertain functions.

What if we want to designate cell B11 as an uncertain function without calculating any summary statistic for it on the worksheet? With cell B11 highlighted, we can select **Results – Output – In Cell**, as shown on the next page:



This will modify cell B11's formula =B4*(B5-B6)-B7 to read **=B4*(B5-B6)-B7 + PsiOutput()**. If we don't want to modify this formula, we can instead choose **Results – Output – Referred Cell**. Just as we saw with the Statistics gallery choice, a small "balloon" like the one below appears and follows the mouse pointer as we move to a worksheet cell.



If we click in cell B12 as shown above, this cell will contain **=PsiOutput(B11)**. Any **one** of these three actions – defining a summary statistic such as PsiMean() on B11, add PsiOutput() to the formula in cell B11, or placing =PsiOutput(B11) in another cell – is sufficient to designate cell B11 as an uncertain function.

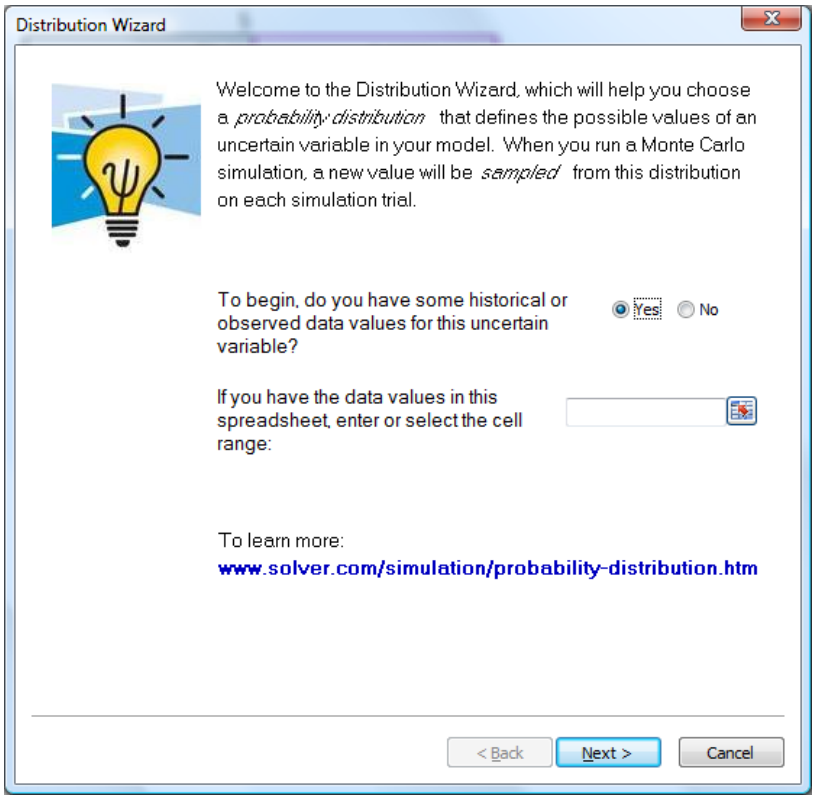
Using the Distribution Wizard

The Distribution Wizard is a tool to help you define uncertain variables and their probability distributions, which for most users is the main challenging task in defining a simulation model.

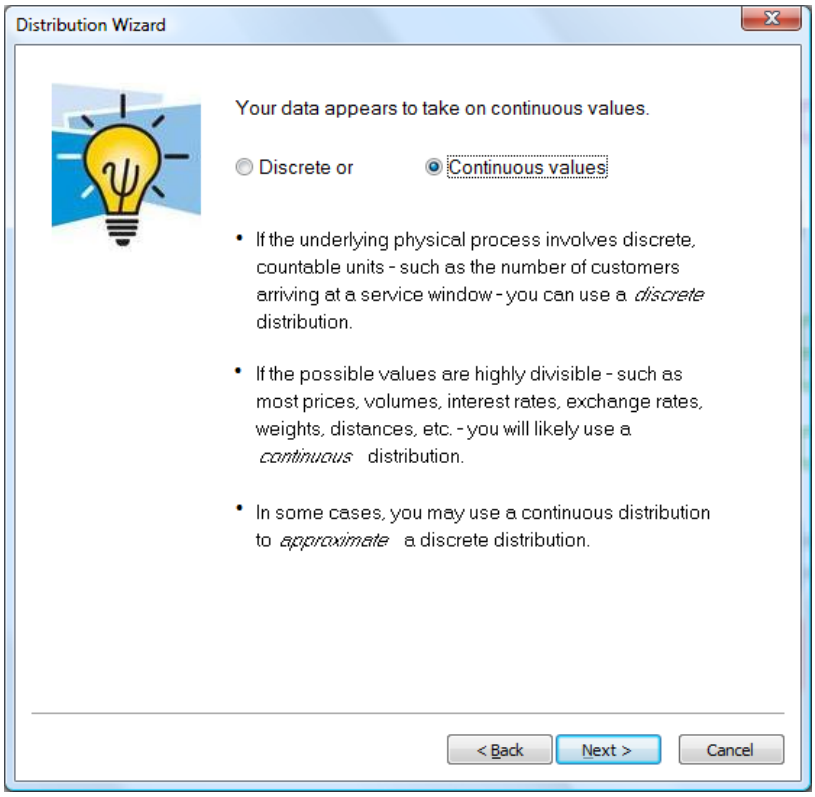
The Distribution Wizard is designed for new users – since it takes you step by step through the process, it's not the fastest way to define a new distribution. But using it can greatly increase your confidence that you're choosing the right kind of distribution, and the Wizard will lead you to the dialog options and charts you'll use to quickly create new distributions in the future.

To use the Distribution Wizard, simply select **Distributions – Distribution Wizard** from the Ribbon. In the initial dialog box, choose **Yes** if you have some historical or observed data (a series of numbers) for the input quantity you're trying to model with an uncertain variable, or **No** if you don't have such data.

If you choose **Yes**, an edit box and cell range selector will appear, which you can use to select a cell range in your spreadsheet where you have the historical or observed data. If you do, the Wizard will analyze this data and proceed. If you have some data, but it is not in your spreadsheet at present, you can still click **Next** – the Wizard will give you some guidance on how to use the data, and then proceed without the data for now.



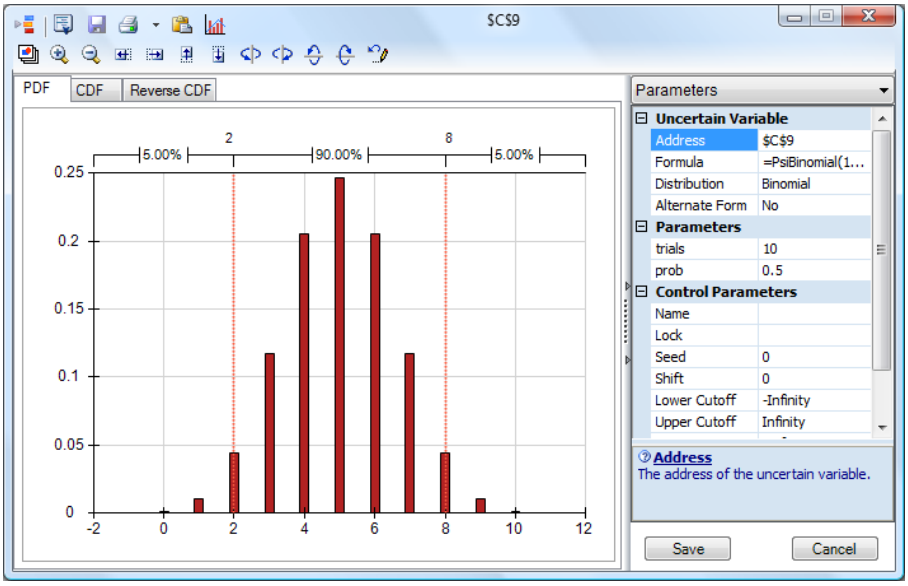
When you proceed to the next step, the Wizard will ask whether this uncertain variable will have *discrete* or *continuous* values (it provides an explanation of these terms, and if you supply historical data, it will analyze the data, make a preliminary determination, and ask you to confirm):



The Wizard will next ask you about bounds on the possible values of this uncertain variable. You'll have to think about the behavior of the physical, financial or other quantity that you're trying to model – but the Wizard will effectively narrow down the possible choices from 50+ probability distributions to just a handful. If you have historical data, the Wizard can optionally *fit* probability distributions and their parameters to your data, and let you choose from a ranked list of the best fits.

You can click the **Cancel** button in any Wizard dialog, which will leave your model unchanged. So feel free to experiment with the Distribution Wizard!

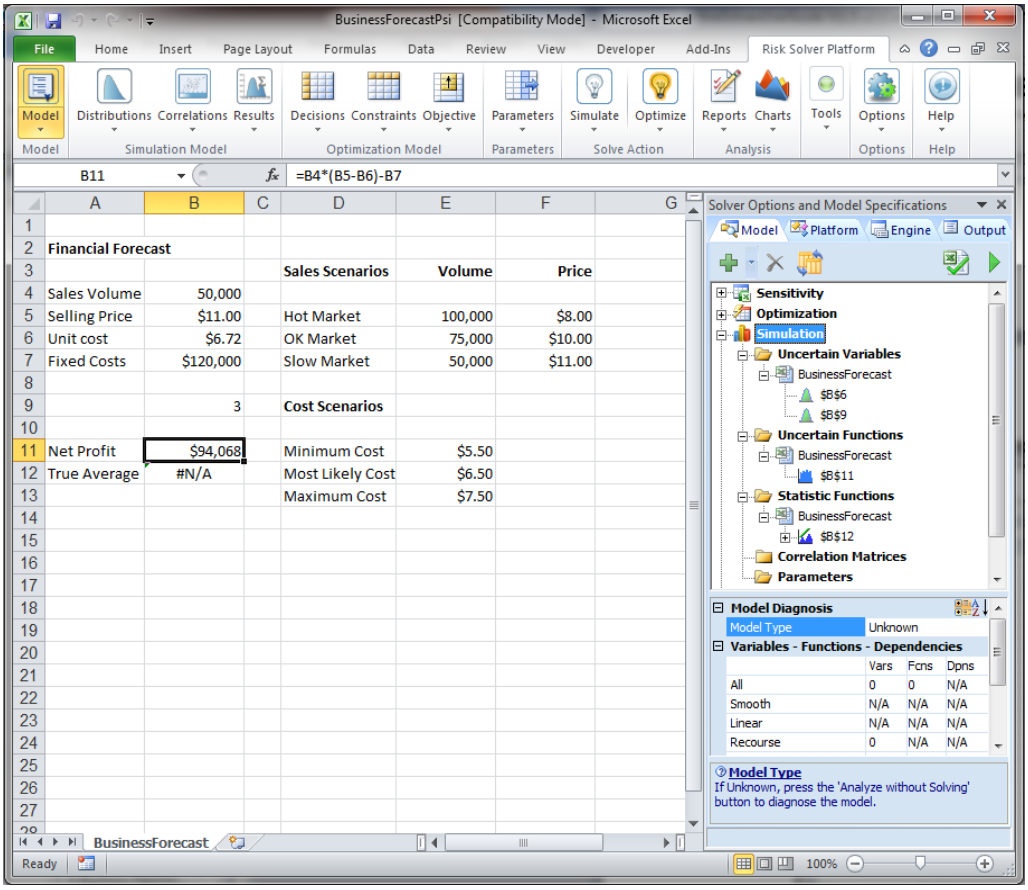
When it has all the information it needs, the Distribution Wizard will summarize the choices you've made. When you click **Finish**, it will open the Uncertain Variable dialog with your choices filled in, as shown in the example below:



When you click the **Save** button, the corresponding distribution formula (for example =PsiBinomial(10,0.5)) will be saved in the active cell on your spreadsheet. You can then edit the cell formula directly, as you would with any formula in Excel, or double-click the cell to display the Uncertain Variable dialog again.

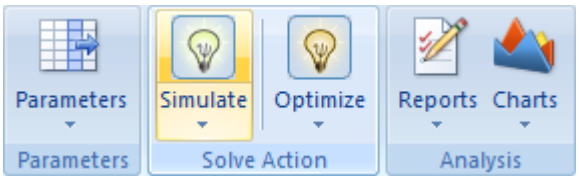
Using Interactive Simulation

Our model now looks like the one pictured below. Notice that the Task Pane **Model tab** shows our simulation model in outline form, and cell B12, where we placed the **PsiMean()** function, displays as #N/A – because Risk Solver Platform's Interactive Simulation is not yet turned on.



With an old-fashioned simulation software package, you'd press a button to start a simulation, then perhaps get a cup of coffee. Because simulations ran slowly, software packages were designed for "batch" operation: You'd spend time getting everything set up just right, run a simulation and wait (sometimes quite a while), then spend time analyzing the results. But with Risk Solver Platform, simulations run so fast that *fully Interactive Simulation* is practical.

To turn on **Interactive Simulation**, simply click the **light bulb** on the Ribbon. It will "light up," as shown below. In the blink of an eye, your first Monte Carlo simulation is complete!



Now 1,000 Monte Carlo simulation trials (the default number) will be executed *each time we change the spreadsheet*, and cell B12 will display the ***true average*** for Net Profit across these trials – as shown below.

	A	B	C	D	E	F
1						
2	**Financial Forecast**					
3				**Sales Scenarios**	**Volume**	**Price**
4	Sales Volume	100,000				
5	Selling Price	$8.00		Hot Market	100,000	$8.00
6	Unit Cost	$6.95		OK Market	75,000	$10.00
7	Fixed Costs	$120,000		Slow Market	50,000	$11.00
8						
9		1		**Cost Scenarios**		
10						
11	Net Profit	-$15,431		Minimum Cost	$5.50	
12	True Average	$93,075		Most Likely Cost	$6.50	
13				Maximum Cost	$7.50	
14						

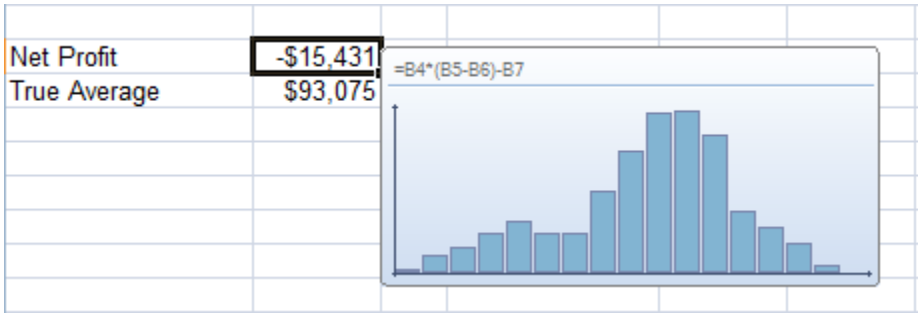
The result of "shaking the ladder" is striking: Our ***true average*** Net Profit for these 1,000 trials is only $93,075 – quite a bit less than the "Flawed Average" Model figure of $117,750! And we also see that we can **lose money** – the last of the 1,000 trials, which appears on the worksheet, shows a loss of $15,431. (If you run this model yourself with no fixed random seed, you might have to press F9 a few times to see a final Net Profit similar to this one.)

Try pressing F9 (the Excel recalculate key) on this model: Each time you do, another 1,000 Monte Carlo trials are run, and a slightly different true average Net Profit figure will be displayed – but normally much less than $117,750.

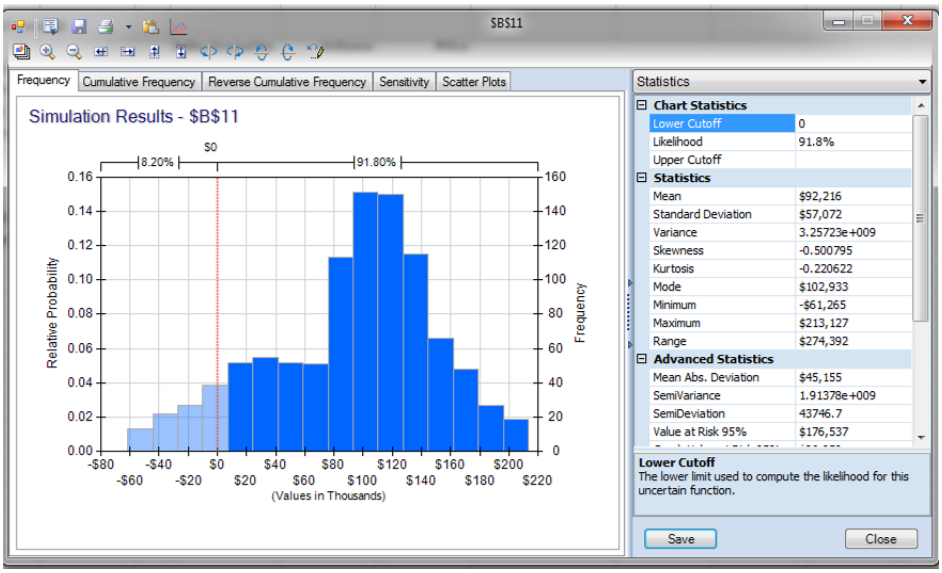
Viewing the Full Range of Profit Outcomes

We've just seen that the mean or 'True Average' Net Profit, over 1,000 different simulated outcomes, is less than we expected from our naïve "Flawed Average" model. We've also seen that in some outcomes, our Net Profit is actually a loss. A quick look at the dropdown galleries for **Results – Statistic**, **Measure** and **Range** on the Ribbon suggests that we can easily compute and view many other statistics about Net Profit. But we'd really like to see the *full range of outcomes* in this model. This is *very* easy to do in Risk Solver Platform.

With Interactive Simulation turned on, simply **move the mouse pointer to B11** and wait about 1 second. A ***miniature***, ***live*** frequency distribution chart of the simulation trial values for cell B11 appears automatically:



To see and do more, just **double-click on B11**, the cell calculating Net Profit, to display a frequency chart of these outcomes, as shown below.

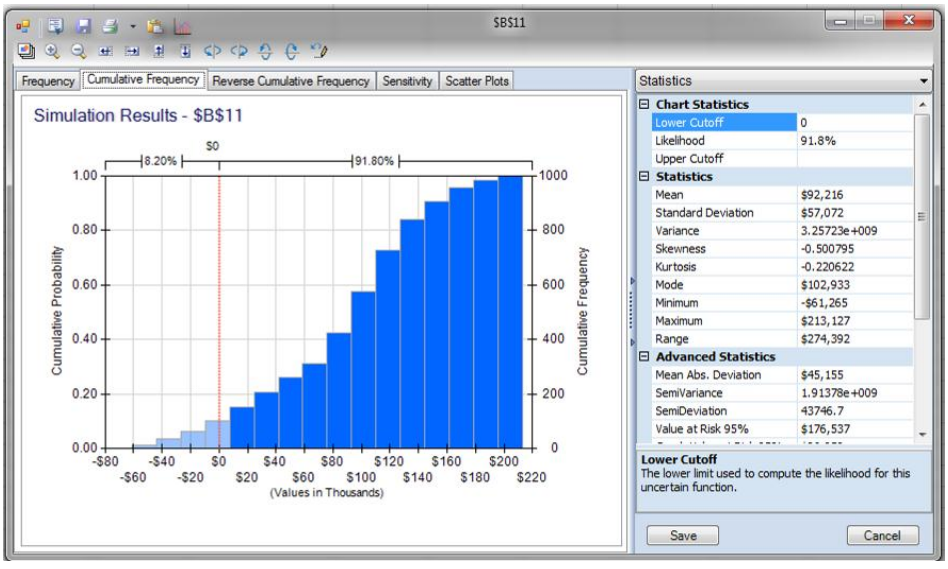


We see immediately that in some outcomes, we can lose of lot of money – over $60,000! We also see that we make a profit in *most* outcomes – but how many exactly? By setting a **Lower-Cutoff** of "0" in the Chart Statistics section of the right panel, or by **right-clicking** over the chart, we can display a vertical bar showing that cutoff.

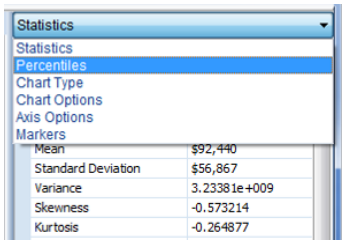
After typing in a 0 you will see the probabilities automatically shown above the chart. In this case there is an 8.80% chance of a loss and a 91.20% chance of a profit.

Looking at the right panel you can see the Statistics view, which includes summary statistics for the full range of Net Profit outcomes. We can see that the worst case outcome of this simulation was -$67,066, and the best case outcome was +$210,712. Value at Risk 95% shows that we have a 95% chance of making $172,321 or less, and Conditional Value at Risk 95% shows that the average of *all* the possible outcomes greater than or equal to the VaR at 95%.

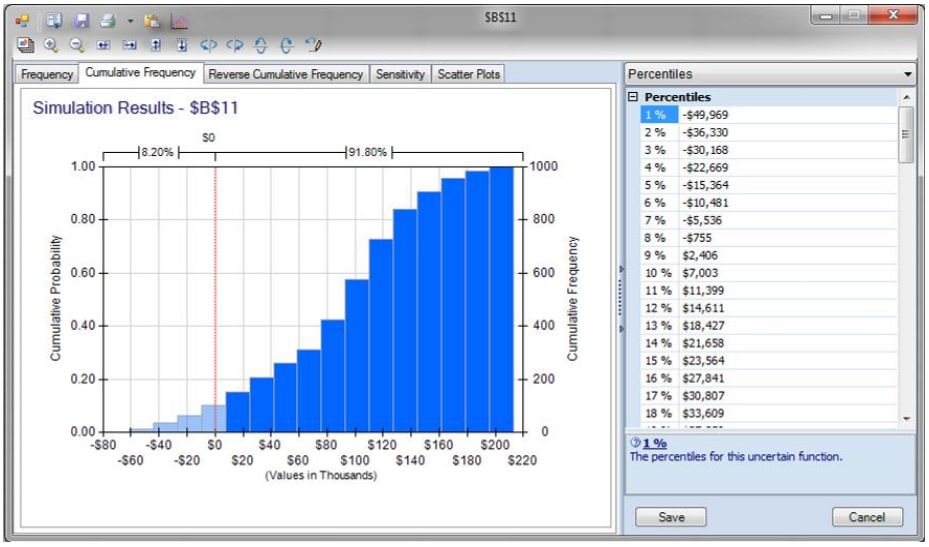
Another view of the full range of outcomes is shown on the Cumulative Frequency tab.



By clicking on the Statistics drop down at the top of the right panel and choosing Percentile.



The Percentiles tab shows the same information as the Cumulative Frequency tab, but in numeric form. Below, we've shown the 50th through 70th percentiles:



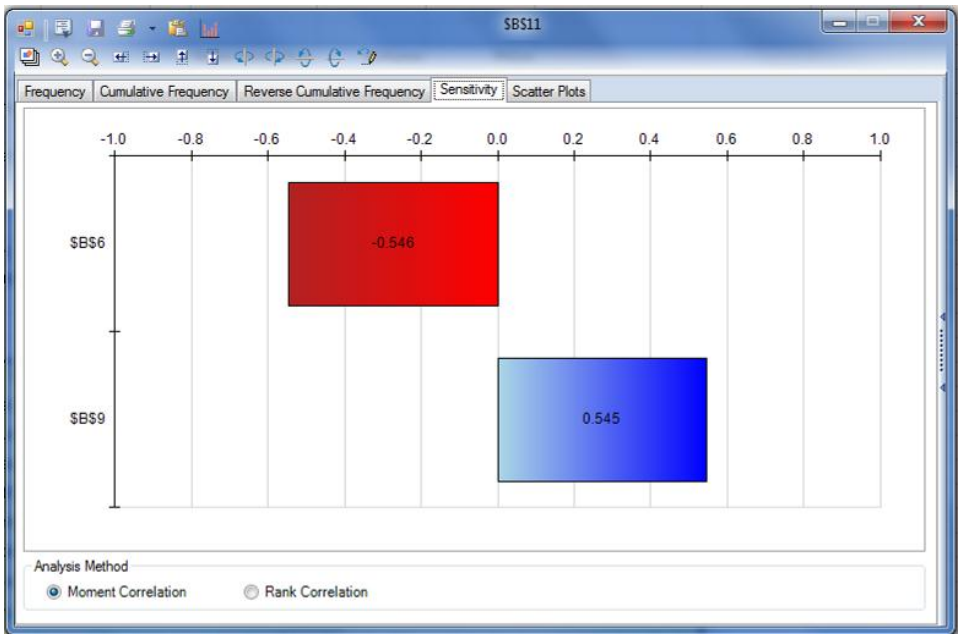
Note that you can close and open the right hand panel simply by clicking on the two arrows on the vertical bar between the chart and the right hand panel.

Analyzing Factors Influencing Net Profit

Other tabs on the Uncertain Function dialog can help us understand how the uncertain variables in our model influence our Net Profit.

Sensitivity Tab

The **Sensitivity tab** displays a "Tornado chart" that shows you how much Net Profit changes with a change in the uncertain variables – the Triangular distribution for Unit Cost at cell B6, and the integer uniform distribution at cell B9. In this model there are only two uncertain variables, but in a large model with many such variables, it's usually not obvious which ones have the greatest impact on outcomes such as Net Profit. A Tornado chart highlights the key variables: B6 has a negative correlation with Net Profit, whereas B9 has a positive correlation.



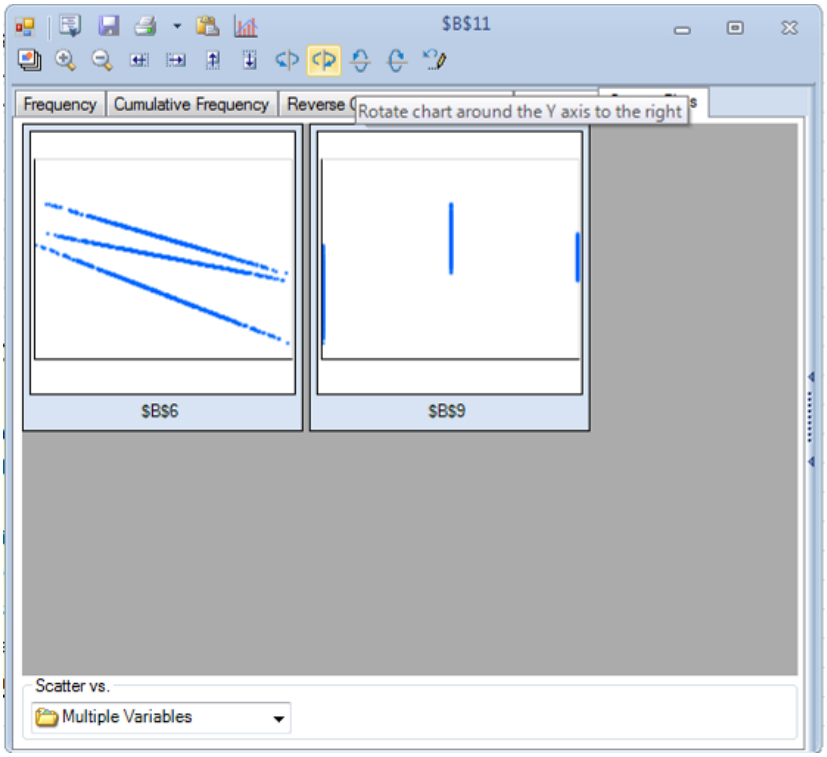
The fact that Net Profit goes down as our Unit Cost at B6 goes up is expected. But why does Net Profit go up as cell B9 goes up? Note that B9 = 1 corresponds to the Hot market scenario, 2 corresponds to the OK market, and 3 corresponds to the Slow market scenario. The fact that B9 is positively correlated with Net Profit is telling us that we make *higher* profits when the market is *slow*, not when it's hot. Our typical Selling Price is lower when the market is hot, and the increased Sales Volume doesn't make up for this.

Scatter Plots Tab

The **Scatter Plots tab**, shown on the next page, gives us a different view of the relationship of Net Profit to our uncertain variables: Unit Cost at cell B6, and the integer uniform distribution at cell B9.

Notice that the scatter plot against Unit Cost has three downward-sloping lines: Each line corresponds to a different market scenario (Hot, OK or Slow), and shows that as Unit Cost rises, our Net Profit drops – dropping fastest in the Hot Market scenario, because our profit margin is so narrow.

The scatter plot against the integer uniform distribution shows three high-low ranges for Net Profit, again corresponding to a different market scenario – Hot, OK, or Slow. We see again that the range of Net Profit is lowest in the Hot Market scenario.



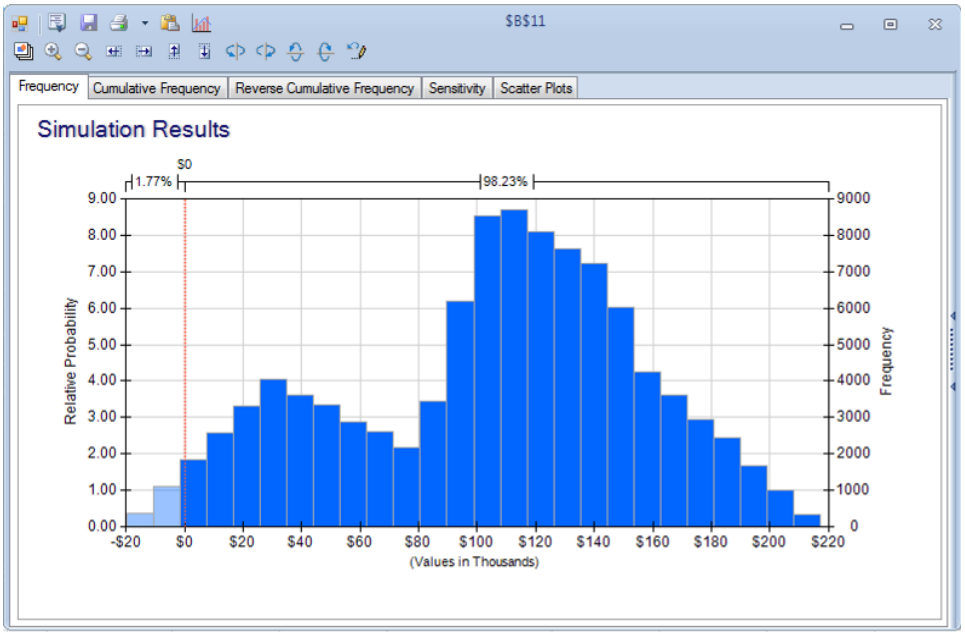
Interactive Simulation with Charts and Graphs

Interactive Simulation makes Risk Solver Platform fundamentally different from other Monte Carlo simulation tools for Excel. The kinds of charts we've just seen can be produced by other tools, but only *at the end* of a "simulation run." In contrast, Risk Solver Platform makes these charts ***live as you play what-if*** with your model.

After seeing this model, your production manager might think of a way to reduce the maximum Unit Cost to $7.00 instead of $7.50. What would be the impact of this change on Net Profit, over the *full range outcomes*? With Risk Solver Platform, this is as easy as any other 'what-if' question in Excel:

Click the Frequency tab to re-display the frequency chart of outcomes for cell B11. Then simply change the number in cell E13 from 7.50 to 7. ***Immediately***, a thousand Monte Carlo trials are performed, and the chart is updated. The effect is striking: We have a 97.8% chance of making a profit, and – checking the Percentiles tab – we see that instead of a 1% chance of losing about $48,000, we have a 1% chance of losing just $6,210!

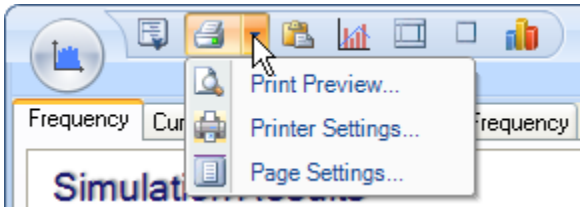
But Risk Solver Platform is even faster than this: Click the Options button on the Ribbon, and change the number of Monte Carlo trials from 1,000 to 100,000 (or more). In a second or less on most modern PCs, ***one hundred thousand*** Monte Carlo trials are performed, and the chart is updated!



The Flawed Average model presented a limited and misleading picture of this business situation. In contrast, the Risk Solver model has illuminated the situation considerably. We can see what can go *right*, and what can go *wrong*. We can make an informed decision about whether the reward is worth the risk. And – most important – we can interactively explore ways to *improve* the reward and *reduce* the risk. This is risk analysis at work.

Charts and Graphs for Presentations

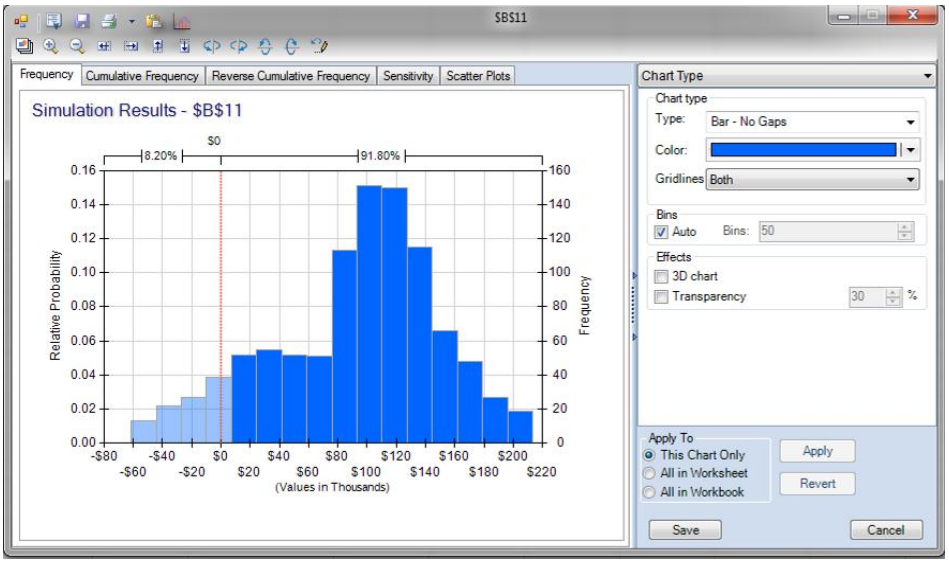
Often, you may be called up to present your results to others. With Risk Solver, one great way to do this is in *Excel itself, live!* But at times you may need to print a chart, or copy it into a Word document or PowerPoint presentation. This is *very* easy to do in Risk Solver Platform, with the toolbar buttons in the title bar of the Uncertain Variable or Uncertain Function dialog:



Click the **Clipboard** icon to copy the currently displayed chart to the Windows Clipboard. You can then choose Edit Paste in Word, Excel, PowerPoint and many other applications to paste the chart image into your document. (Choosing Edit Paste in Excel inserts a static, non-updating chart image in the worksheet.)

Click the **Print** icon to immediately print the currently displayed chart on your default printer, or click the down arrow next to this icon to display the menu choices shown above: Print Preview, Printer Settings and Page Settings. You can choose a printer and set printer options, set page margins, and preview your output using these menu choices.

Risk Solver Platform also makes it easy to control the format of your charts. Below, we've re-opened the right hand panel and clicked on the drop down menu to choose **Chart Type**.



You can control the chart type, color, dimensionality and transparency, bin density, titles and legends, axis labels and number formats, horizontal axis scaling, and more. As you change chart options in the right pane, the chart is *immediately* updated so you can see the results (unlike some other simulation products for Excel). When you're satisfied with the chart format, you can save and apply it to just this chart, to all charts on this worksheet, or to all charts in the workbook when you click the Apply button at the bottom of the right pane.

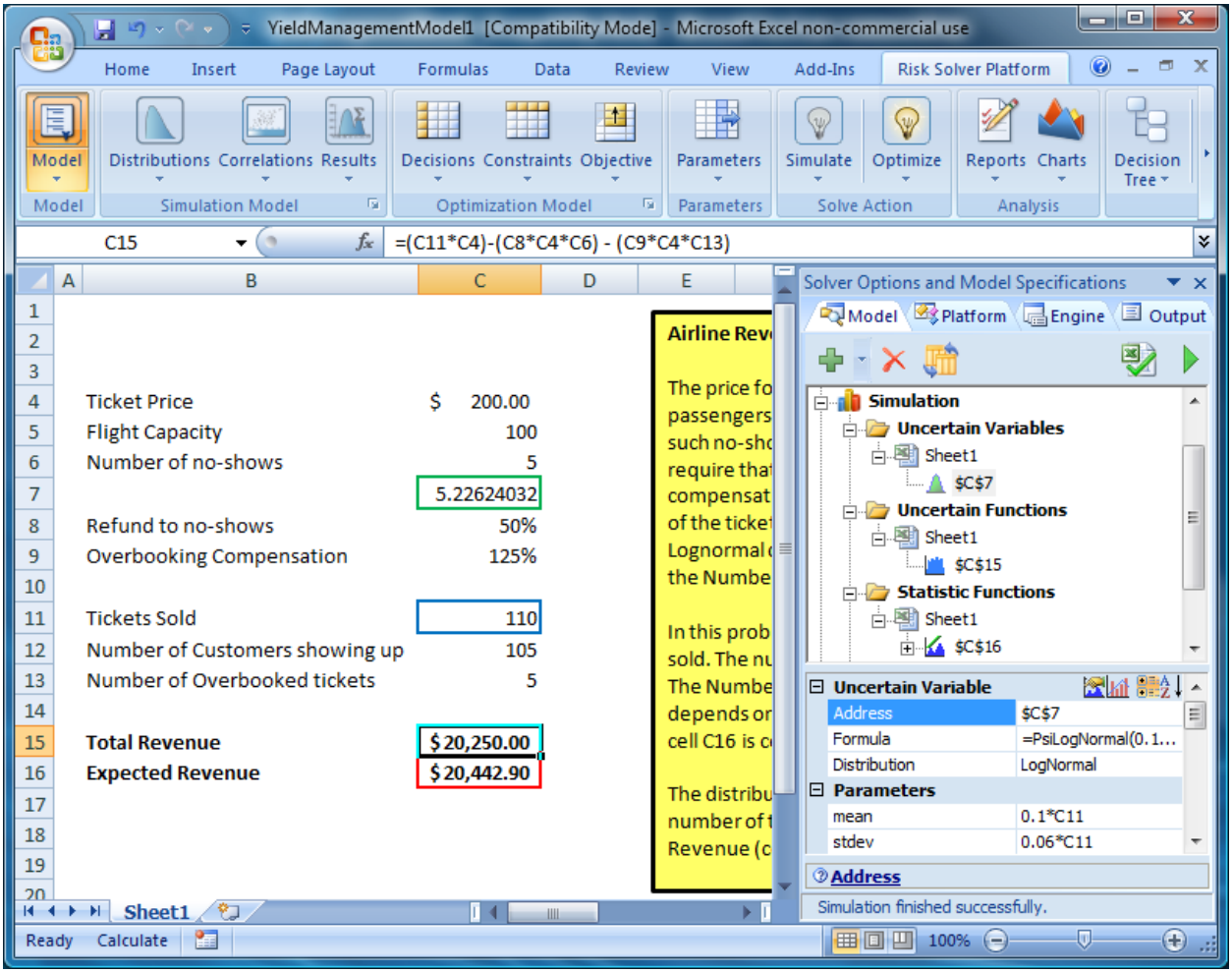
An Airline Revenue Management Model

In this section, we'll explore a simple airline revenue management model, also known as a yield management model. We'll start with a simple simulation model, like the one in the previous section. But in this section, we'll see we can answer further questions by running *multiple parameterized simulations* in Risk Solver, and by solving a *simulation optimization* model in Risk Solver Pro Risk Solver Platform.

To open this example, click **Help – Examples** on the Ribbon, which opens a workbook where you can click on Simulation examples with the list of example workbooks. Click the links to open examples YieldManagement1.xls (a simple simulation model), YieldManagement2.xls (a model with multiple parameterized simulations), and YieldManagement3.xls (a simulation optimization model).

A Single Simulation

The model YieldManagement1.xls is shown on the next page.



The model depicts a hypothetical airline flight from San Francisco to Seattle. The flight has 100 seats, and tickets are $200 per seat. Some passengers who purchase tickets are "no-shows" whose seats will be empty; in this example we assume that such passengers receive a refund of 50% of their purchase price.

To utilize their 'perishable inventory' of seats, the airline would like to sell more than 100 tickets for each flight. But we assume that Federal regulations require that any ticketed passenger who is unable to board the flight due to overbooking is entitled to compensation of 125% of the ticket price.

The airline would like to know how much revenue it will generate from each flight, less refunds for no-shows and compensation for 'bumped' passengers. As shown above, this net revenue amount is calculated at cell C15, for any specific number of tickets sold (110 above) and number of no-shows (5 above).

The uncertain quantity in this model is the number of no-shows; hence we should model this with an *uncertain variable*. We quickly realize that the **number of no-shows** will depend on the **number of tickets sold**. After some research, we decide that we can use a LogNormal distribution for the number of no-shows: Cell C7 contains the formula **=PsiLogNormal(0.1*C11,0.06*C11)**. Cell C8 contains =ROUND(C7,0) to ensure that the number of no-shows is an integer value – and this is used to compute Net Revenue at cell C15.

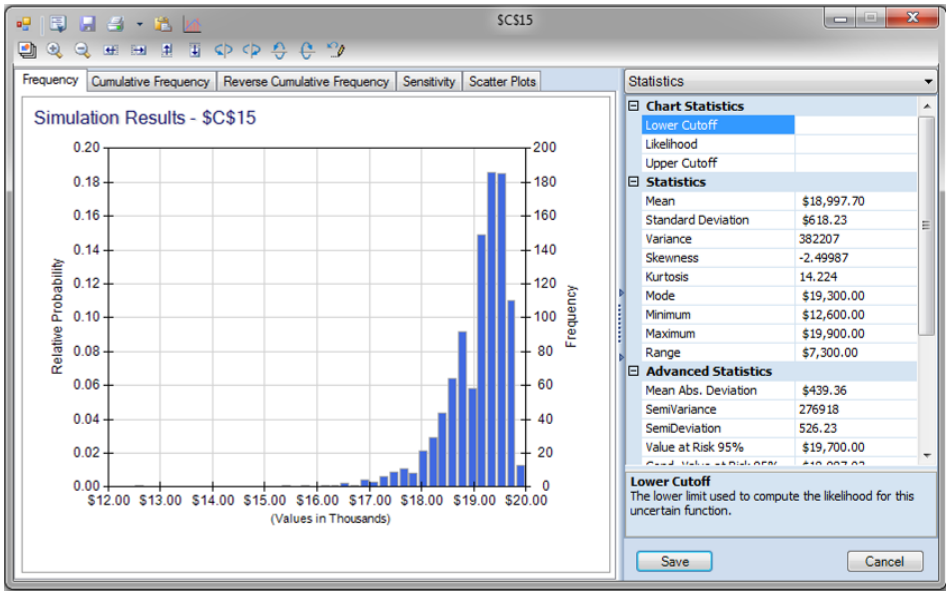
Note that the bottom area of the Task Pane is showing the properties of cell C7: for example, the *mean* of the distribution is 10% of the number of tickets sold.

See the chapter "Mastering Simulation and Risk Analysis Concepts" for advice on choosing distributions for your uncertain variables.

As shown in the Task Pane, cell C15 is an uncertain function; it is defined as such because cell C16 contains =PsiMean(C15) to compute the expected (or average) revenue across all Monte Carlo trials.

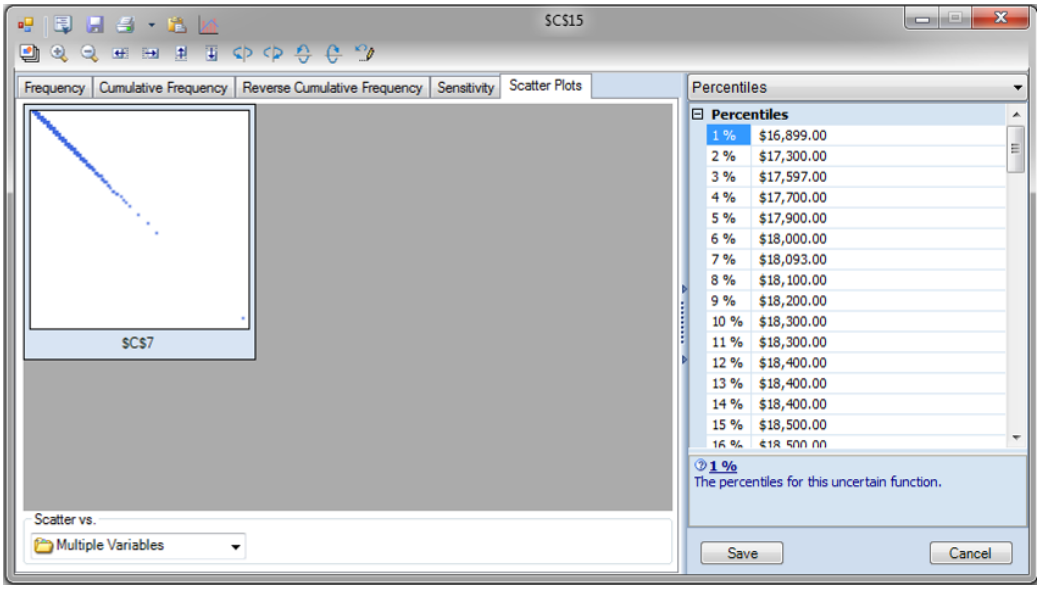
Performing the Simulation

To perform a single simulation (with 1,000 Monte Carlo trials), we select **Simulate – Run Once** from the Ribbon. If we then double-click cell C15, we can see the distribution of outcomes in the Uncertain Function dialog:



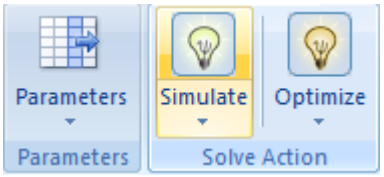
The right hand panel shows the statistics, or from cell C16, we see that the mean or expected net revenue is about $19,000. From the **Cumulative Frequency tab** over the chart or by clicking on the drop down menu in the right hand panel (currently showing "Statistics") and selecting **Percentiles**, we can see that the 10% percentile is $18,300 – if we sell 110 tickets, we'll earn as much revenue as a full flight (100 seats * $200) 90% of the time. You can see why airlines find that overbooking makes sense as a policy!

The **Scatter Plots tab** shows you quick scatter plots of this uncertain function against one or more uncertain variables, or other uncertain functions. In this example – shown on the next page – it shows a scatter plot of C15 (net revenue) against C7 (number of no-shows). Since we're selling 110 tickets, we see that revenue grows at first as the number of no-shows increases, since we avoid overbooking compensation and sell each empty seat for more than the refund issued to the no-show passenger. But when the number of no-shows passes a certain level, net revenue declines. Perhaps we should sell more than 110?



Using Interactive Simulation

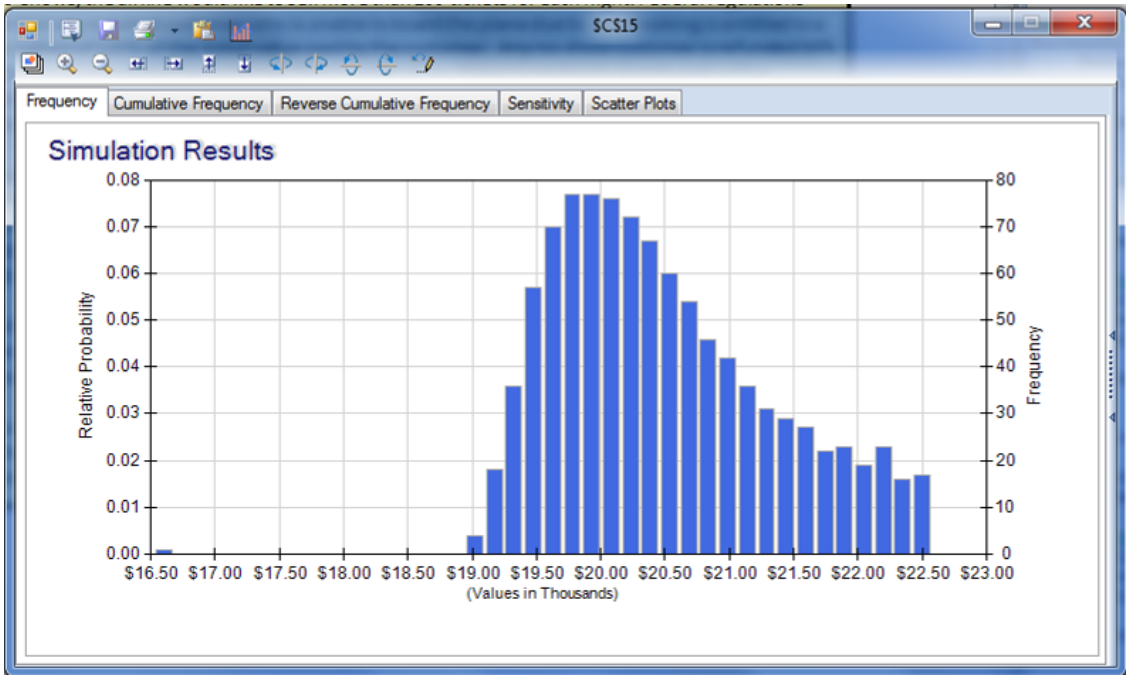
We can explore the question of how many tickets to sell with Interactive Simulation: We simply click the Simulate button to "turn on the lights:"



Now we can simply type a new number of Tickets Sold, such as **125**, into cell C11. Instantly, Risk Solver Platform performs a new simulation of 1,000 Monte Carlo trials, and updates the Uncertain Function dialog and the worksheet with the results – as shown on the next page.

The new distribution of outcomes is quite different – we have more high-revenue data points – but the mean or expected revenue is not so different -- $20,502. Selling 125 tickets does seem to be better than selling 110 tickets, but we're not yet sure that this is the *best* number of tickets to sell. (We'll answer that question in the next two sections.)

In the meantime, however, Interactive Simulation allows us to "play what-if" with this model, changing the parameter that we *can* control (the number of tickets sold), while properly modeling the behavior of the parameter we *cannot* control (the number of no-shows). As Prof. Sam Savage says, "Interactive Simulation does for uncertainty what the spreadsheet did for numbers."



Multiple Parameterized Simulations

After playing "what-if" with Interactive Simulation, we can move to ask Risk Solver Platform to automate this process: Vary a parameter (the number of tickets sold) over a range, performing a simulation for each different number of tickets sold, and summarize the results.

You can see this by choosing **Help – Examples** on the Ribbon, and clicking on the simulation text link and then selecting **YieldManagement2.xls** in the list of example workbooks. But you can create YieldManagement2.xls from YieldManagement1.xls in just three steps:

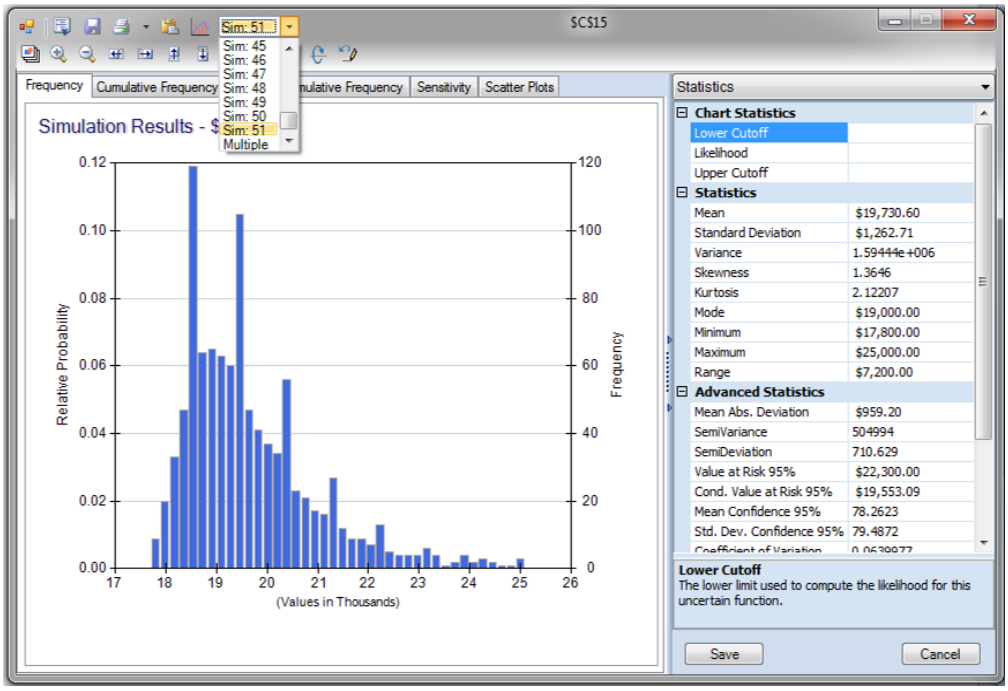
1. Select cell C11, choose **Parameters – Simulation** from the Ribbon, and enter a lower limit of 100 and upper limit of 150 in the dialog. This creates a formula =PsiSimParam(100,150) in cell C11.
2. In the Task Pane Platform tab Simulation group, set the **Number of Simulations** to 51. Cell C11 will have a value 100 on the first simulation, 101 on the second simulation, and so on, through 150 on the 51st simulation.
3. Select cell C15, choose **Results – Statistic – Mean**, and select a *range* of cells (C25:BA25 is used in YieldManagement2.xls). Risk Solver Platform will place =PsiMean(C15,1) in the first cell, =PsiMean(C15,2) in the second cell, and so on for as many cells as you specify.

YieldManagement2.xls does this slightly differently, with the same result.

Now, when you choose **Simulate – Run Once**, Risk Solver Platform will perform 51 simulations, each with 1,000 Monte Carlo trials, and give you access to *all* of the results. On the worksheet, you'll see the expected revenue for 100, 101, …, 150 tickets sold. And you can use the Uncertain Function dialog to display the full distribution of outcomes for any of the 51 simulations.

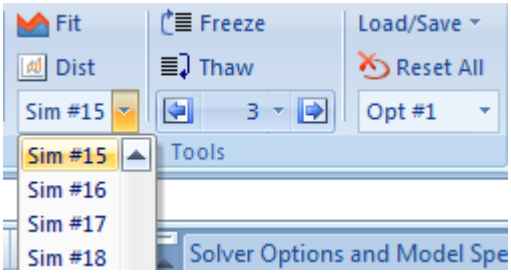
Examining Results Across Simulations

To do this, double-click cell C15 to display the Uncertain Function dialog:



The dialog now includes a dropdown list that allows you to select a simulation to display. You can examine the Frequency, Cumulative Frequency, Reverse Cumulative Frequency, Sensitivity, or Scatter Plots tabs for any simulation.

The worksheet will initially display values for the last Monte Carlo trial of the last simulation performed (except for the cells containing PSI Statistics function calls such as PsiMean()). But you can display any trial from any simulation, by changing the selections in the Sim # and Trial # controls on the Ribbon:

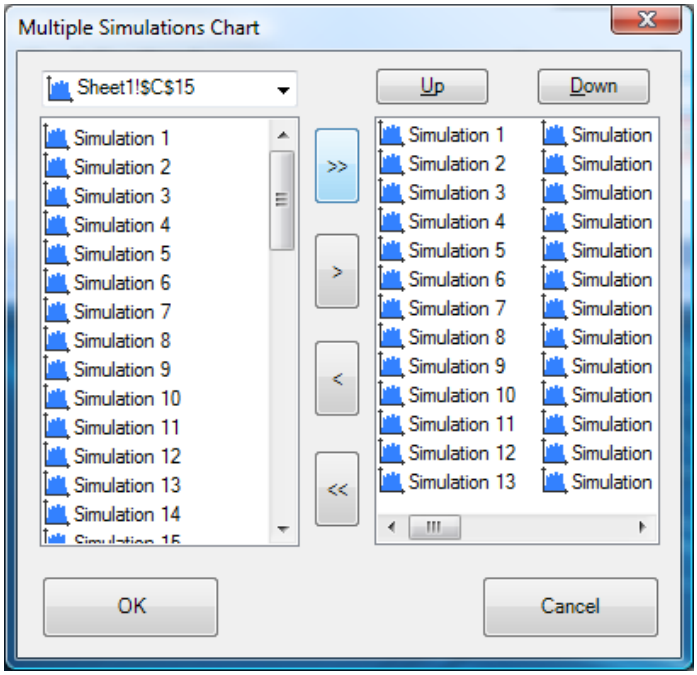


By examining this data, you can get a better idea of how many tickets to sell. But Risk Solver Platform can help your further.

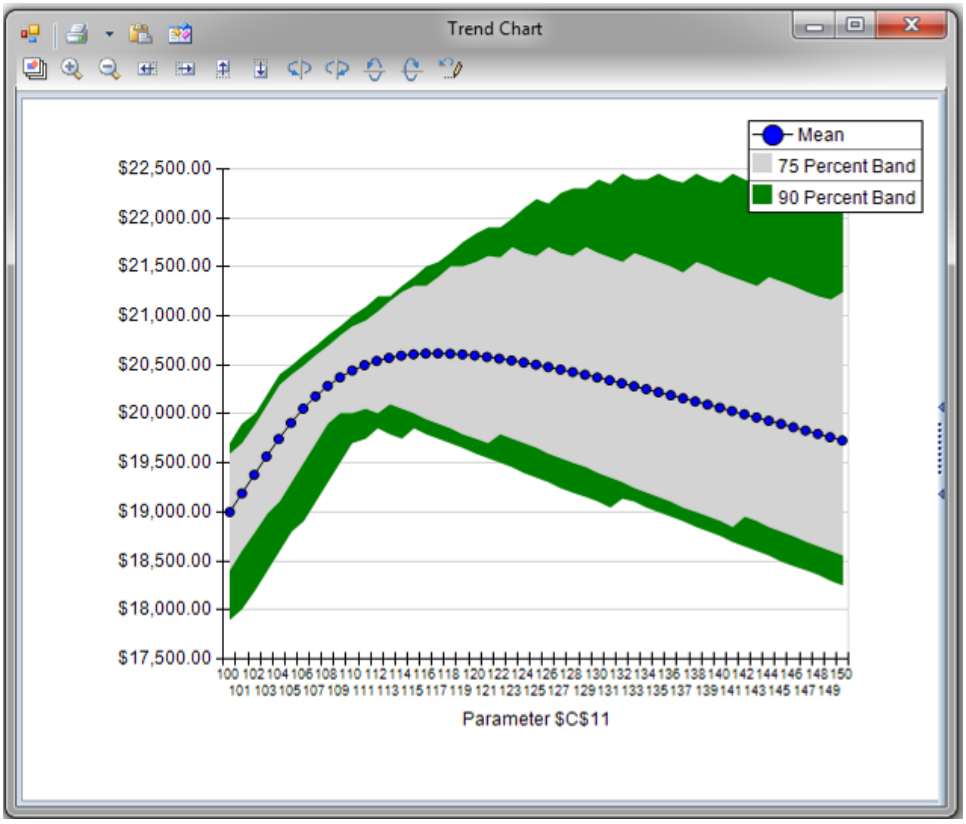
Reports and Charts of Multiple Simulations

Risk Solver Platform has rich facilities for reports and charts across multiple simulations. We'll cover just two chart examples; see the Frontline Solvers Reference Guide for further information.

First, select **Charts – Multiple Simulations – Trend** from the Ribbon. Risk Solver Platform displays a dialog where you can select one or more simulations to include in the chart. Click the >> button to select all 51 simulations:



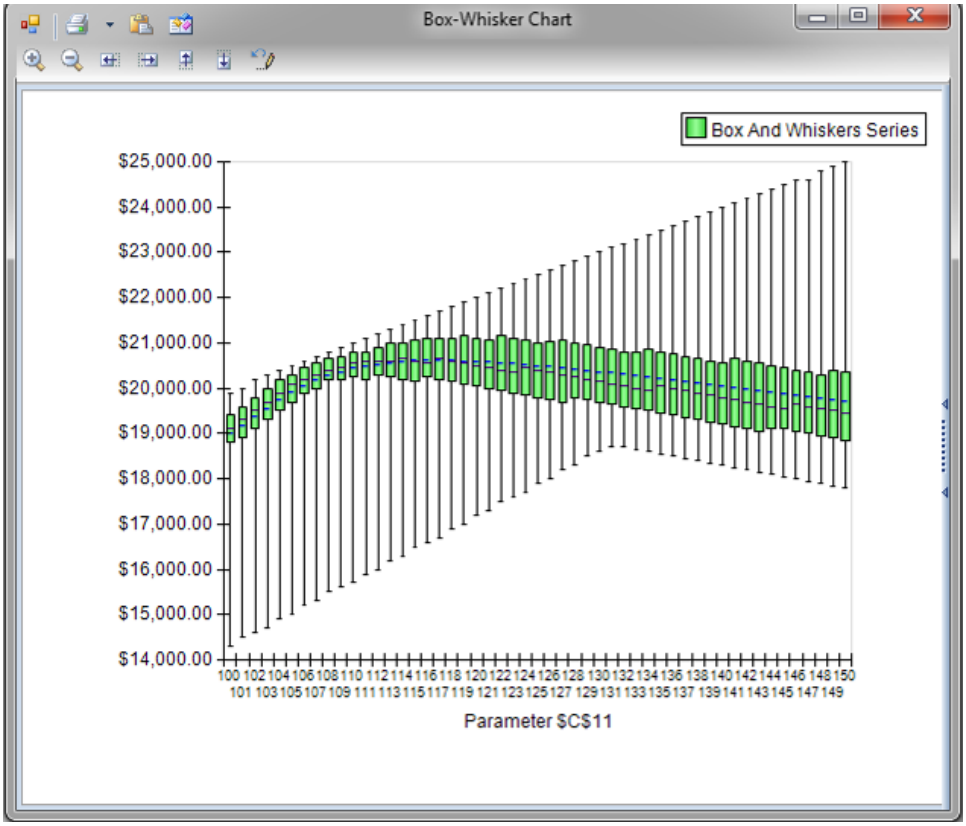
Risk Solver Platform then draws the Trend Chart:



The number of tickets sold is on the horizontal axis. Risk Solver Platform shows the mean, 25th and 75th percentile, and 10th and 90th percentile values of net revenue at C15 (this was the only choice in the dropdown list at the top left

of the Multiple Simulations dialog). We can see that the mean or expected net revenue peaks at about 116 or 177 tickets sold.

For another view of this information, select **Charts – Multiple Simulations – Box-Whisker** from the Ribbon, and again select all 51 simulations to be included in the chart. Risk Solver Platform draws the following chart:



This shows a Box-Whisker diagram for each of the 51 simulations, with the mean, median, 25th and 75th percentiles, and the minimum and maximum value for net revenue on each one. You can click the right edge of the Trend and Box-Whisker charts to open a right panel with options for customizing these charts. You can use the icons on the title toolbar to print the chart, or copy it to the Clipboard, where it can be pasted into another application such as PowerPoint.

Simulation Optimization

Running multiple parameterized simulations, and examining the results on the worksheet and in chart form, has given us a good deal of intuition about the behavior of this simulation model. But Risk Solver Platform and Risk solver Pro (in combination with Premium Solver Pro or Premium Solver Platform) can *directly* answer the question "How many tickets should we sell to realize the *maximum expected* net revenue?"

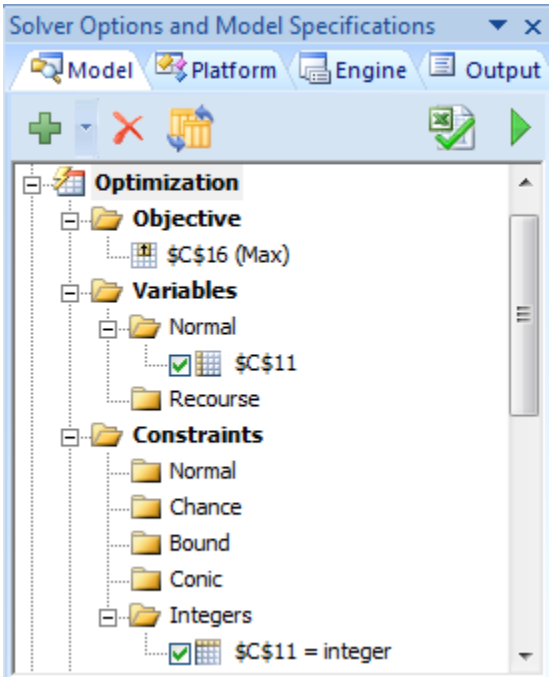
You can see this by choosing **Help – Examples** on the Ribbon, and on the overview page clicking first on the **Simulations** link and then clicking **YieldManagement3.xls** in the list of example workbooks. But you can create YieldManagement3.xls from YieldManagement1.xls in just three steps:

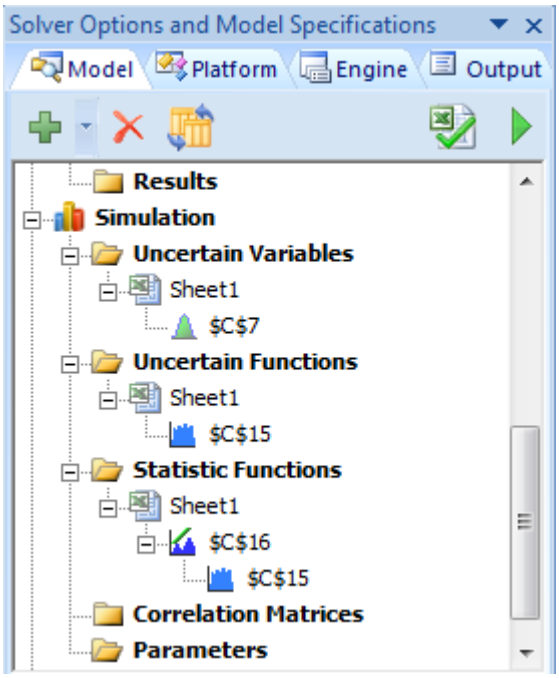
1. Select cell C11, and choose **Decisions - Normal** from the Ribbon. This makes the number of tickets sold a decision variable for optimization.

2. With C11 still selected, choose **Constraints – Variable Type/Bound – Integer** from the Ribbon. This specifies that C11 must have an integer value at the optimal solution.

3. Select cell C16, and choose **Objective – Max – Normal** from the Ribbon. The optimizer will maximize C16: = PsiMean(C15), the expected value of net revenue.

The model in Task Pane will now look like the one pictured below (we scrolled down to show both the optimization and simulation model elements):

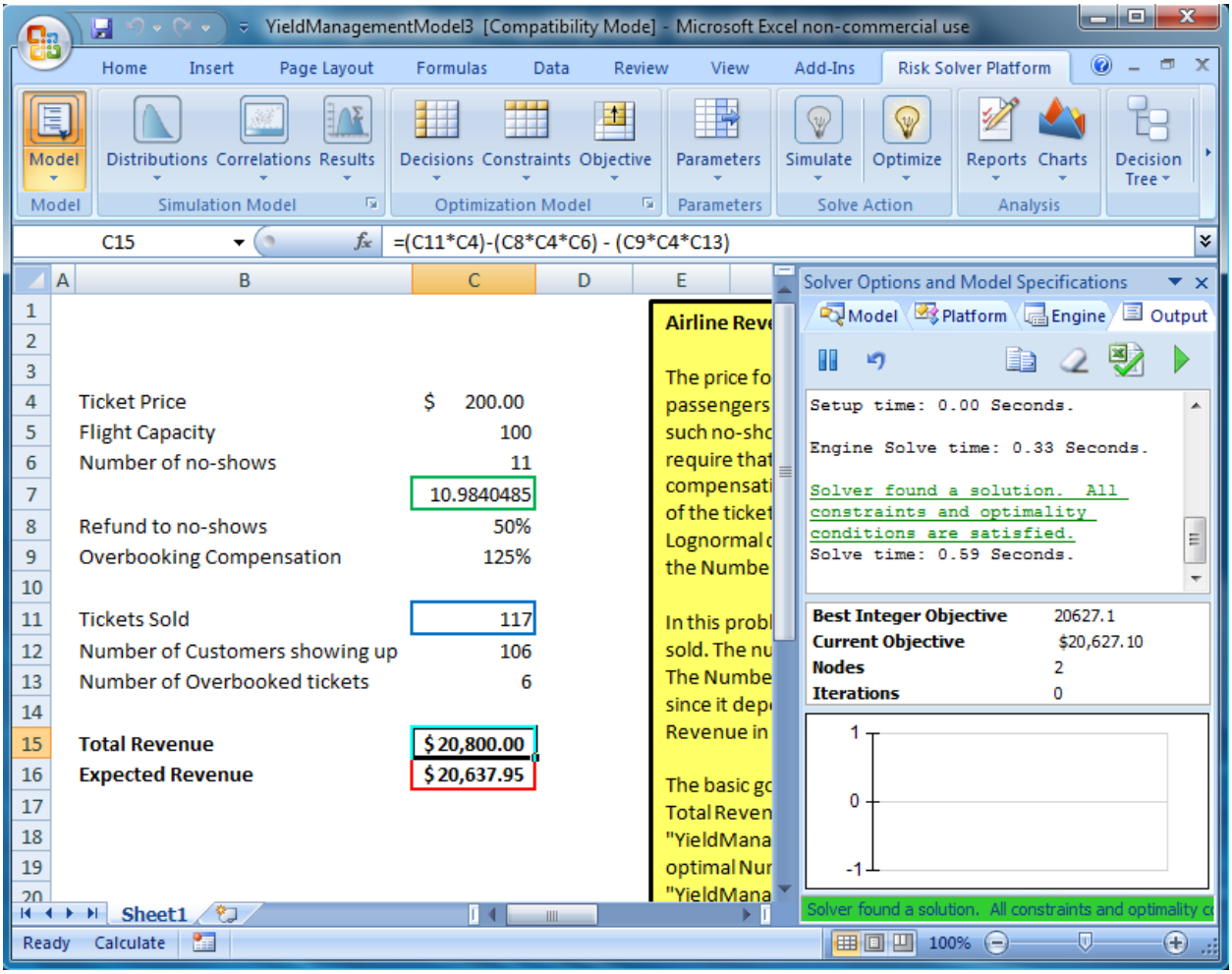




In YieldManagement3.xls, we've made just these selections, and we've also chosen the LSGRG Nonlinear Solver from the dropdown list on the Engine tab, to solve the problem. (Other choices are possible, but this one is simplest for this example problem.)

Now we can simply click the **Optimize** button on the Ribbon. In a fraction of a second, the solution of 116 or 117 tickets sold appears in C11 (the exact number depends on the simulation Random Seed), and the expected value of net revenue for this number of tickets appears at C16 (about $20,619). "Solver found a solution. All constraints and optimality conditions are satisfied" appears in green at the bottom of the Task Pane, as shown on the next page.

Note that this required less than a page to explain how you can get answers to questions like "How many tickets should we sell to realize the *maximum expected* net revenue?"


YieldManagementModel3 [Compatibility Mode] - Microsoft Excel non-commercial use
Home
Insert
Page Layout
Formulas
Data
Review
View
Add-Ins
Risk Solver Platform
Model
Distributions
Correlations
Results
Decisions
Constraints
Objective
Parameters
Simulate
Optimize
Reports
Charts
Decision Tree
Simulation Model
Optimization Model
Solve Action
Analysis
C15
=(C11*C4)-(C8*C4*C6) - (C9*C4*C13)
Ticket Price $ 200.00
Flight Capacity 100
Number of no-shows 11
10.9840485
Refund to no-shows 50%
Overbooking Compensation 125%
Tickets Sold 117
Number of Customers showing up 106
Number of Overbooked tickets 6
Total Revenue $20,800.00
Expected Revenue $20,637.95
Solver Options and Model Specifications
Model
Platform
Engine
Output
Setup time: 0.00 Seconds.
Engine Solve time: 0.33 Seconds.
Solver found a solution. All constraints and optimality conditions are satisfied.
Solve time: 0.59 Seconds.
Best Integer Objective 20627.1
Current Objective $20,627.10
Nodes 2
Iterations 0
Sheet1
Ready
Calculate
100%

Examples: Stochastic Optimization

Introduction

This chapter introduces stochastic optimization in Risk Solver Platform, with a series of examples.

To open the examples, click **Help – Examples** on the Ribbon, which opens a workbook where you can click on the text link Stochastic Examples to open the workbook StochasticExamples.xls.

We use the term stochastic optimization to mean optimization of models that include *uncertainty*, using any solution method. Risk Solver Platform offers an exceptional level of power to find robust optimal solutions to models with uncertainty, using three different solution methods:

- Simulation optimization
- Stochastic programming
- Robust optimization

The first method, **simulation optimization**, is also available in Risk Solver Pro, and is the only method in other software products such as Crystal Ball Professional and @RISK Industrial. It handles very general models, but it is not scalable to large models (with thousands of variables and constraints), and it doesn't support the important modeling concept of recourse decisions.

Stochastic programming and **robust optimization** can be applied only to linear and quadratic programming models with uncertainty, but they are scalable to large models. Full Risk Solver Platform is required to use these methods.

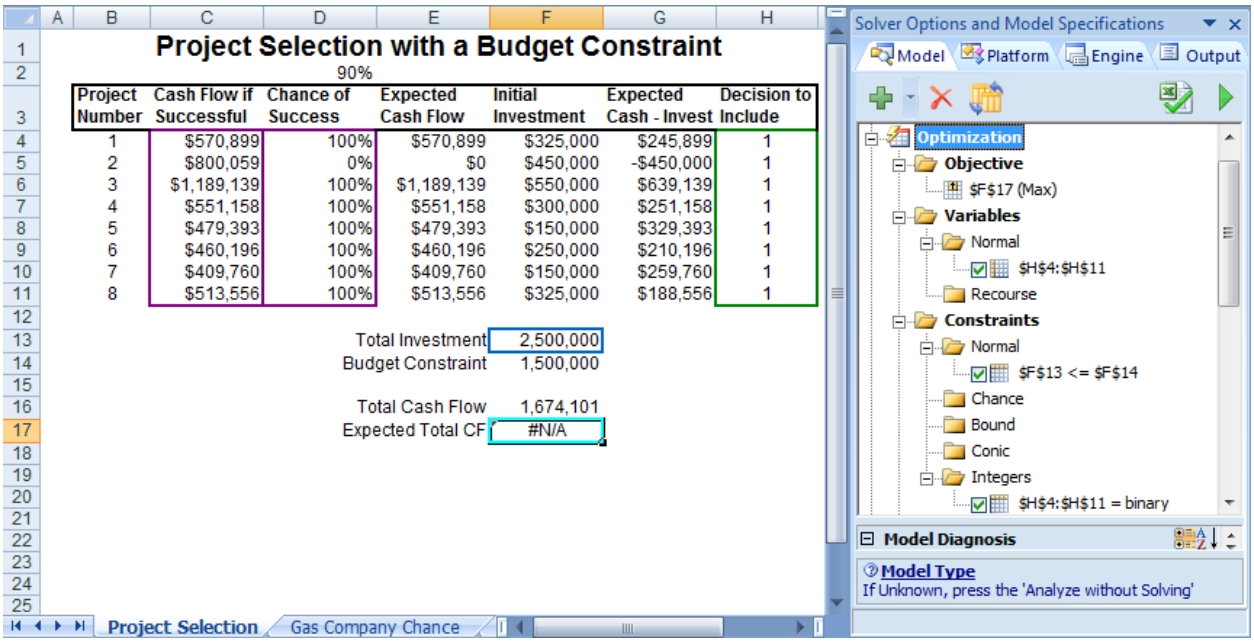
A Project Selection Model

Our first example may surprise you: It is a capital budgeting problem, where the projects being considered for funding have uncertain future cash flows. Models similar to **Project Selection** have been used with other software products for many years to illustrate how **simulation optimization** can be used to optimize models with uncertainty. We'll show how to do this in Risk Solver Platform or Risk Solver Pro, using simulation optimization and the Evolutionary Solver – with much better performance. But when we use Risk Solver Platform to *analyze the structure* of this model, we'll see that this model *doesn't require simulation optimization at all*. We can solve this model in a fraction of a second – using the LP/Quadratic Solver!

In **Project Selection**, eight different capital projects are proposed for funding. Each one has a known initial investment. Each project has a 90% chance of success, and if a project succeeds, it will have an uncertain (but positive) future

cash flow. Funding all eight projects would require a total initial investment of $2.5 million, but our capital budget is only $1.5 million. Hence, we must choose a subset of the projects to fund that will maximize our expected total future cash flow, while ensuring that our total initial investment doesn't exceed our $1.5 million budget.

Click the tab for the Projection Selection model, shown below.



To model the uncertainty in this problem, we use PSI functions that define *uncertain variables* with certain probability distributions. We can create these functions via the Distributions dropdown gallery and the Uncertain Variable dialog, as shown in the last chapter, or we can just type in the formulas.

To model the "Cash Flow if Successful" in column C, we use the **PsiTriangular()** function, specifying a minimum, most likely, and maximum cash flow. For example, the formula in C4 is =PsiTriangular(400000, 500000, 900000).

To model the "Chance of Success" in column D, we use **=PsiBinomial(1,D2) where D2=.9**. On each trial, this distribution returns 1 with probability 90% and 0 with probability 10%. In column E we multiply the "Cash Flow if Successful" and the "Chance of Success" to obtain the "Expected Cash Flow."

Column G computes the *net* cash flow – the "Expected Cash Flow" in column E minus the "Initial Investment" in column F. Column H holds our decision variables, which are constrained to be binary integer (0 or 1 at the solution).

Cell F13 computes the total initial investment as the SUMPRODUCT of the project initial investments (column F) and the 0-1 variables in column H. In the model, F13 is constrained to be less than the $1.5 million budget in F14.

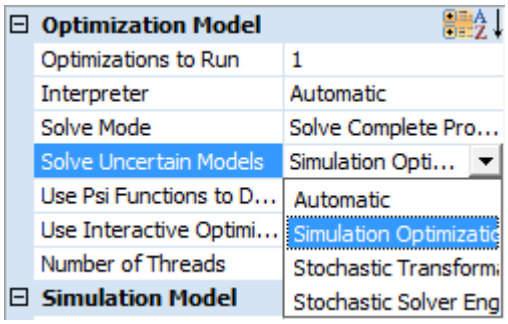
Cell F16 computes the total cash flow as the SUMPRODUCT of the project net cash flows and the 0-1 variables in column H. Cell F17 – unique to a model with uncertainty – contains =PsiMean(F16): It computes the mean (*expected*) value of total net cash flow. This is the objective function we want to maximize.

We're asking Risk Solver Platform to find the best combination of projects – by finding 0 or 1 values for the selection variables at H4:H11 – to maximize cell

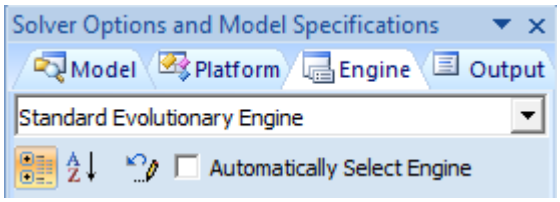
F17, the expected value of total net cash flow, subject to the constraint that the total initial investment doesn't exceed our $1.5 million budget.

Solving with Simulation Optimization

We'll first solve this problem using **simulation optimization**. To do this, go to the Task Pane Platform tab, and change the **Solve Uncertain Models** option to **Simulation Optimization**. (We'll see later what happens when we allow Risk Solver Platform to choose the solution method automatically.)



On the Engine tab, select the **Evolutionary Solver** from the dropdown list.

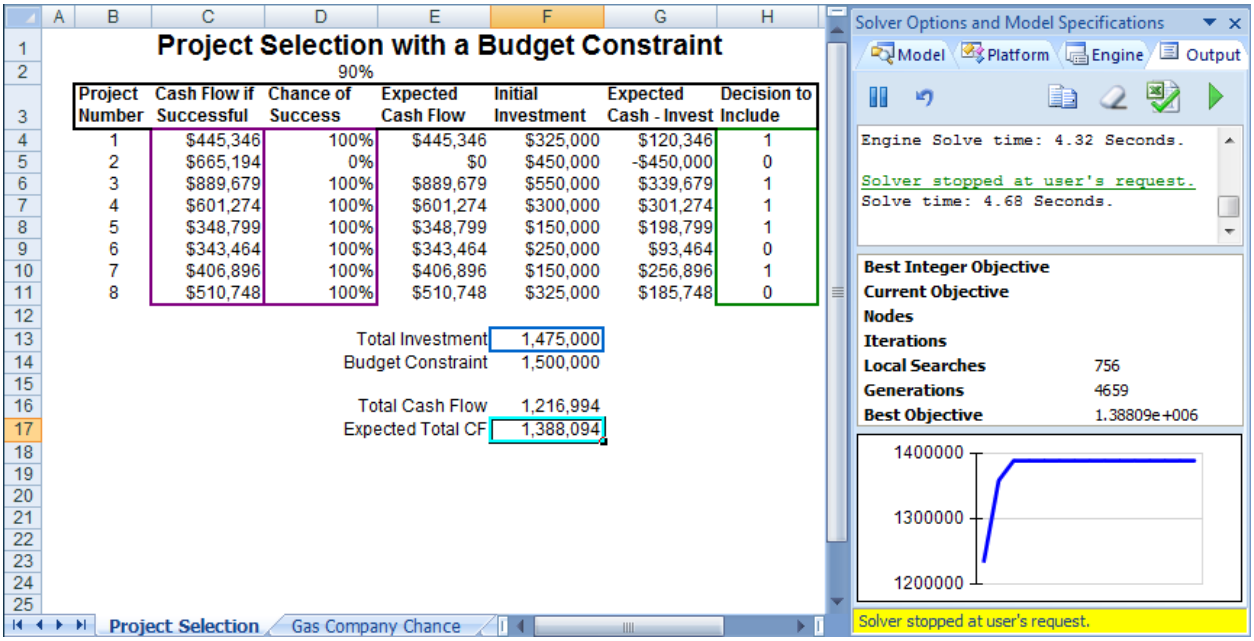


Why do we choose these settings? In the case of other software products, it's because simulation optimization is the only technology they have – and "if your only tool is a hammer, every problem looks like a nail." Simulation optimization is also a very general approach that can handle nonlinear and non-smooth functions, and it's comparatively easy to understand.

The idea behind simulation optimization is straightforward: **For each** set of values for the decision variables considered by the optimizer, we perform **one simulation**, a compute of user-specified *summary measures* – such as =PsiMean(F16) in the Project Selection model – for the constraints and/or objective that depend on uncertainty. The optimizer uses these summary measures to decide what set of values it should try *next* for the decision variables – and the process is repeated.

The great strength of simulation optimization is its **generality** – but this is also its weakness: It requires a new simulation **at each step** of the optimization, and because the method assumes no structure in the model, in general the **number of step**s can grow **exponentially** with the number of variables and constraints.

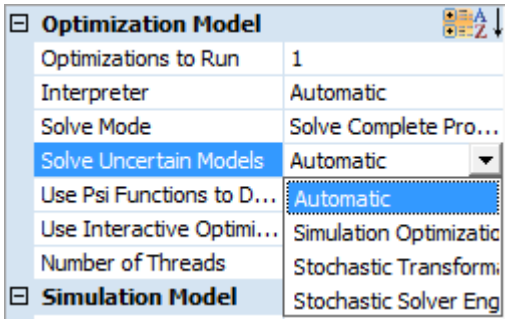
Now click the **Optimize** button on the Ribbon, or the green arrow in the Task Pane. After a second or two (much faster than other software products), you can press ESC to stop the Evolutionary Solver and display the best solution found so far – as shown on the next page.



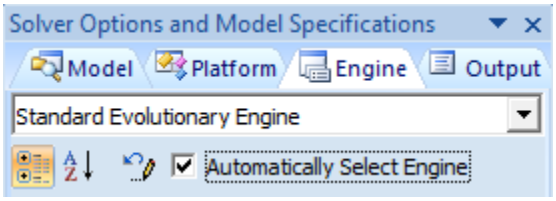
We've decided to fund projects #1, 3, 4, 5, and 7 for an expected total net cash flow of $1,388,094. That's a great result. But – *how much work* did we do to solve this model, and did we *need* to do that much work? It might not matter for a small model like this one – but for a 'scaled-up' model that might involve hundreds or thousands of projects, the solution might have taken far more time.

Solving Automatically

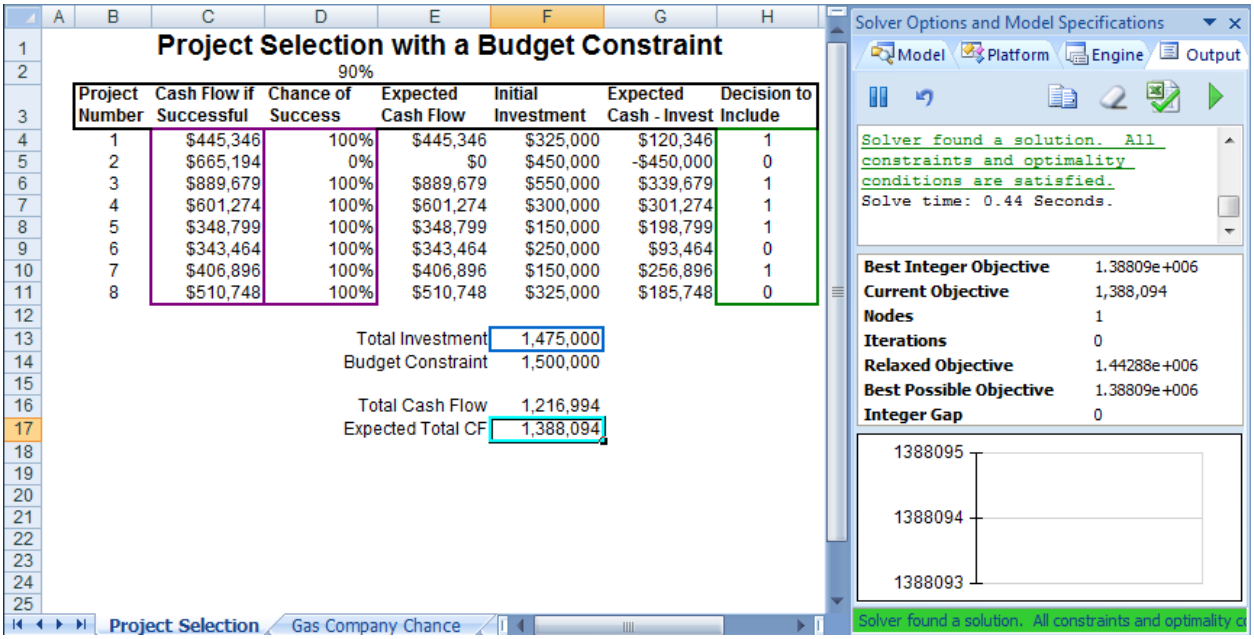
Let's allow Risk Solver Platform to choose the solution method and the Solver Engine automatically. To do this, go to the Task Pane Platform tab, and change the **Solve Uncertain Models** option to **Automatic**.



On the Engine tab, check the box to automatically select a Solver Engine.



Now click the **Optimize** button on the Ribbon, or the green arrow in the Task Pane. The results are shown on the next page.



In a *fraction of a second*, a solution appears with the *same* objective value of $1,388,094 and the same projects selected, with the message "Solver found a solution. All constraints and optimality conditions are satisfied." This means that Risk Solver Platform found a *proven* globally optimal solution – whereas with simulation optimization, we never know whether the solution we found is optimal. How did this happen?

If you examine the solution log, you'll see these messages:

```
---- Start Solve ----
Uncertain input cells detected.
Parsing started...
Diagnosis started...
Model diagnosed as "Stochastic LP".
Parsing started...
Diagnosis started...
Automatic engine selection: LP/Quadratic Solver
Stochastic Transformation succeeded using Deterministic
Equivalent.
Transformed model is "LP Convex".
Model: [StochasticExamples.xls]Project Selection
Using: Psi Interpreter
Parse time: 0.27 Seconds.
Engine: LP/Quadratic Solver
Setup time: 0.00 Seconds.
Engine Solve time: 0.00 Seconds.
Solver found a solution.  All constraints and optimality
conditions are satisfied.
Solve time: 0.44 Seconds.
```

Risk Solver Platform analyzed the model and diagnosed it as a Stochastic LP (linear programming) problem, with 8 variables, 2 functions (objective and budget constraint), and 16 uncertainties (the PsiTriangular() and PsiBinomial() functions) that affect the objective. As you'll see in the chapter "Mastering Simulation and Risk Analysis Topics," Risk Solver Platform can solve a Stochastic LP using two *other* methods besides simulation optimization – stochastic programming and robust optimization. For this model, it chose a transformation to Stochastic Programming Deterministic Equivalent form.

But this model is so simple that Risk Solver Platform determined that *no* transformations were needed to solve the problem. Despite the presence of uncertainty, this model *doesn't require simulation optimization at all*. There was *no need* to run hundreds of Monte Carlo simulations (or many more, with some alternative software products). It was sufficient to run *one* Monte Carlo simulation at the beginning, to compute the *constant* linear coefficients of the objective function. Project Selection is simply an LP/MIP (linear mixed-integer programming) model!

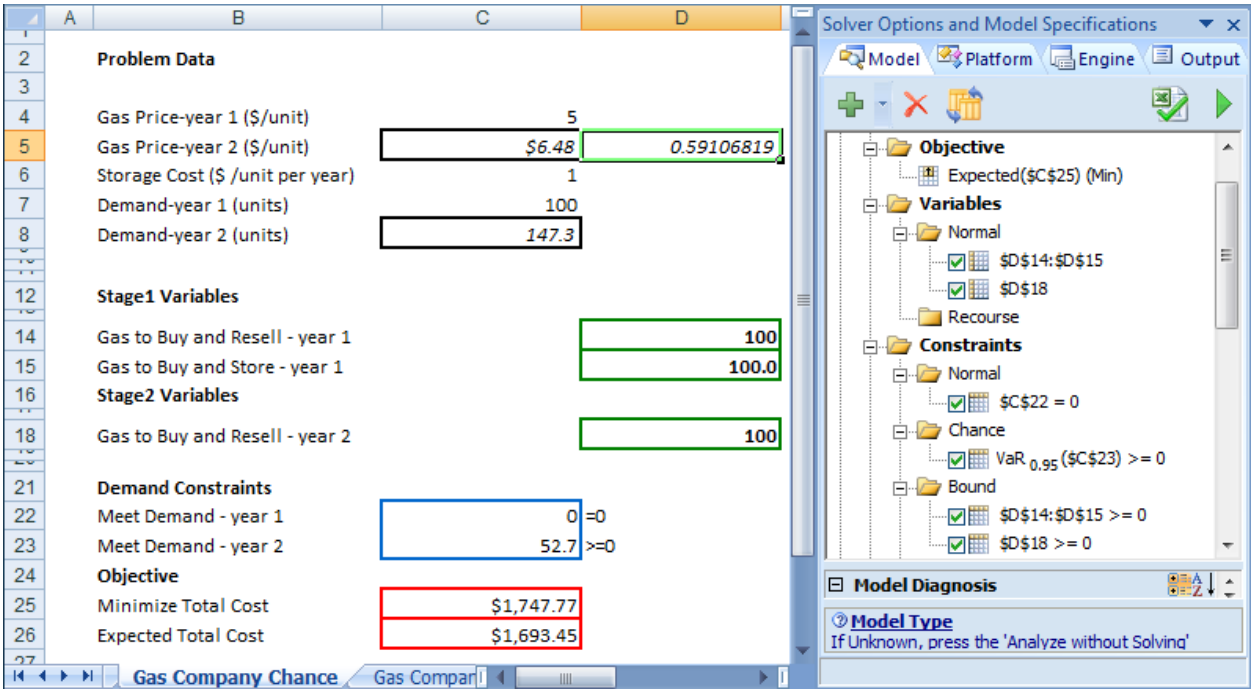
In this small model, we found the same solution within seconds using both methods. But in a 'scaled-up' model in a large company, with hundreds or even thousands of projects, Risk Solver Platform's ability to *analyze the structure* of the model could make all the difference – between a solution of unknown quality, found after a long wait using the *wrong approach* – and a proven optimal solution, found within seconds using the *right approach*.

A Model with Chance Constraints

The worksheets Gas Company Chance and Gas Company Recourse in StochasticExamples.xls are related – they both describe the same problem, but with different assumptions about *when* a certain decision must be made. With Gas Company Chance, we'll show how a problem with uncertainty can be formulated with a *chance constraint*, and solved using robust optimization methods. With Gas Company Recourse, we'll show how to use a "wait and see" or **recourse decision** – a capability *not available at all* with simulation optimization – to obtain a *better* optimal solution.

We hope these examples will motivate you to read the chapter "Mastering Simulation and Risk Analysis Topics," which will give you a *unified view* of the crucial characteristics of models with uncertainty, and the options of solving them with simulation optimization, stochastic programming, and robust optimization methods.

Click the **Gas Company Chance** tab to display the model, which is pictured on the next page.



In this model, a gas company purchases natural gas from suppliers and resells it to consumers in its service area. Consumer demand is known to be 100 units this year (cell C7), but is uncertain for next year (cell C8); the company must meet this demand with high probability, or it will encounter public relations and regulatory problems. The price of gas is known to be $5.00/unit this year (cell C4), but is uncertain for next year (cell C5): If the weather is very cold, demand may reach 180 units, and the price may reach $7.50/unit, but if the weather is warm, demand may remain at 100 units, and the price may remain at $5.00/unit. The company's goal is to minimize its total cost of purchased and stored gas (cell C25). It decisions are:

- How much gas to purchase and resell to consumers *this year* (cell D14)
- How much gas to purchase *this year* and store (cell D15) , at a cost of $1 per unit, for use *next year*
- How much to purchase and resell to consumers *next year* (cell D18)

In Gas Company Chance, we assume that the gas company must commit to all of its gas purchases this year, *before* next year's demand is known: All three decision variables D14, D15 and D18 are normal or 'here and now' decisions. (In the next section, we'll consider Gas Company Recourse, where we assume that next year's gas purchases can be determined on a 'wait and see' basis.)

Gas Company Chance has a constraint that the company must have enough gas to meet next year's *uncertain* demand "with high probability." We'll model this in Risk Solver Platform with a ***chance constraint***. Our objective function – total cost, to be minimized – depends in part on next year's uncertain gas price. We will therefore seek to minimize the ***expected value*** **of total cost**.

The **objective** appears as "Expected($C$25) (Min)" – in Gas Company Chance, we used the Ribbon to define the objective interactively, rather than a PsiMean() function on the worksheet, as we did earlier in Project Selection. The decision variables are cells D14, D15 and D18, as explained earlier; each of these is constrained to be non-negative, as shown in the Bound section of the outline.

At C22 we calculate =D14-C7 (this year's gas purchases minus this year's demand), and at C23 we calculate =D15+D18-C8 (gas purchased and stored this year, plus gas purchased next year, minus next year's uncertain demand).

The **normal constraint** $C$22 = 0 specifies that we must meet this year's consumer demand exactly (forcing D14 to 100). The **chance constraint $\mathbf{VaR_{0.95}}$($C$23) >= 0** requires that the 95th percentile of C23, over *all* realizations of the uncertainty, must be non-negative – in other words, we must meet demand with 95% probability.

We *could* express this chance constraint by placing =PsiPercentile(D15+D18-C8,0.95) in cell C23, and using the Ribbon to define a *normal* constraint C23 >= 0. But while PsiPercentile and other PSI Statistics functions are acceptable for simulation optimization, the transformation for *robust optimization* requires that we express the chance property directly in the constraint itself, not in a formula.

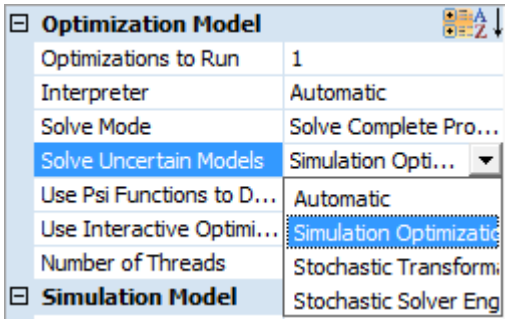
Solving with Robust Optimization

We could solve this model using *simulation optimization*, as we did for Project Selection in the previous section. But here, we want to show how Risk Solver Platform can automatically *transform* this stochastic linear programming problem – with its expected value objective and chance constraint – into a larger conventional optimization model, using the methods of ***robust optimization***.

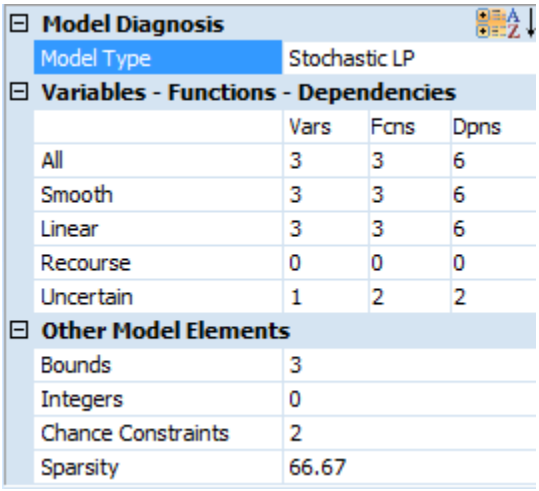
The idea behind robust optimization is to *transform* an optimization problem with known *structure* – such as a stochastic linear programming problem – into a larger, *conventional* 'robust counterpart' problem that accounts for the impact of *bounded* uncertainty on the constraints and objective. Risk Solver Platform performs **one simulation** at the outset, to assess the impact of uncertainty and construct the robust counterpart problem. Solving the robust counterpart – a **single LP or SOCP problem** – gives us an approximate solution to the original stochastic problem.

The strength of robust optimization is its **scalability** to very large problems – since only one simulation and one optimization of an LP or SOCP problem is required. But robust optimization cannot solve more general nonlinear, non-convex problems.

As saved, the model is ready to be solved using robust optimization. But to see the difference between the original model – a stochastic linear programming model – and the robust counterpart model that is the result of the transformation, we'll first set the **Solve Uncertain Models** option to **Simulation Optimization**:

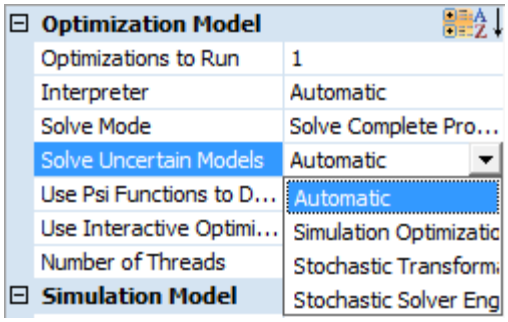


and then click the **Analyze** button to analyze the structure of the model. The Model Diagnosis section of the Task Pane Model tab "pops up" to show the results, as shown on the next page.

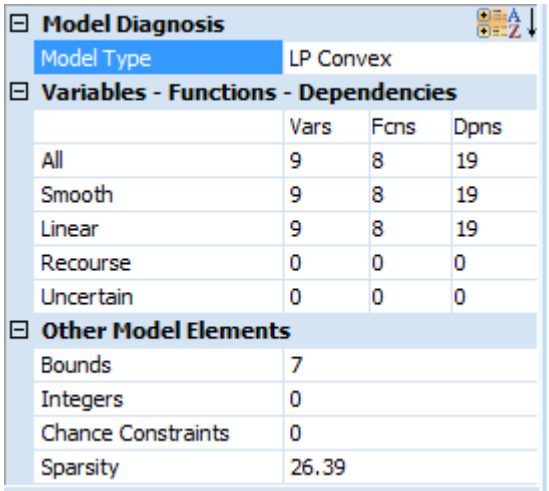


Model Diagnosis			
Model Type	Stochastic LP		
Variables - Functions - Dependencies			
	Vars	Fcns	Dpns
All	3	3	6
Smooth	3	3	6
Linear	3	3	6
Recourse	0	0	0
Uncertain	1	2	2
Other Model Elements			
Bounds	3		
Integers	0		
Chance Constraints	2		
Sparsity	66.67		

The model is diagnosed as a Stochastic LP, with 3 variables, 3 functions, and 1 uncertainty (the Beta distribution at cell D5, which models the weather). Now, we'll set the **Solve Uncertain Models** option to **Automatic**:



and click the **Analyze** button again – this will analyze the structure of the *transformed* robust counterpart model.

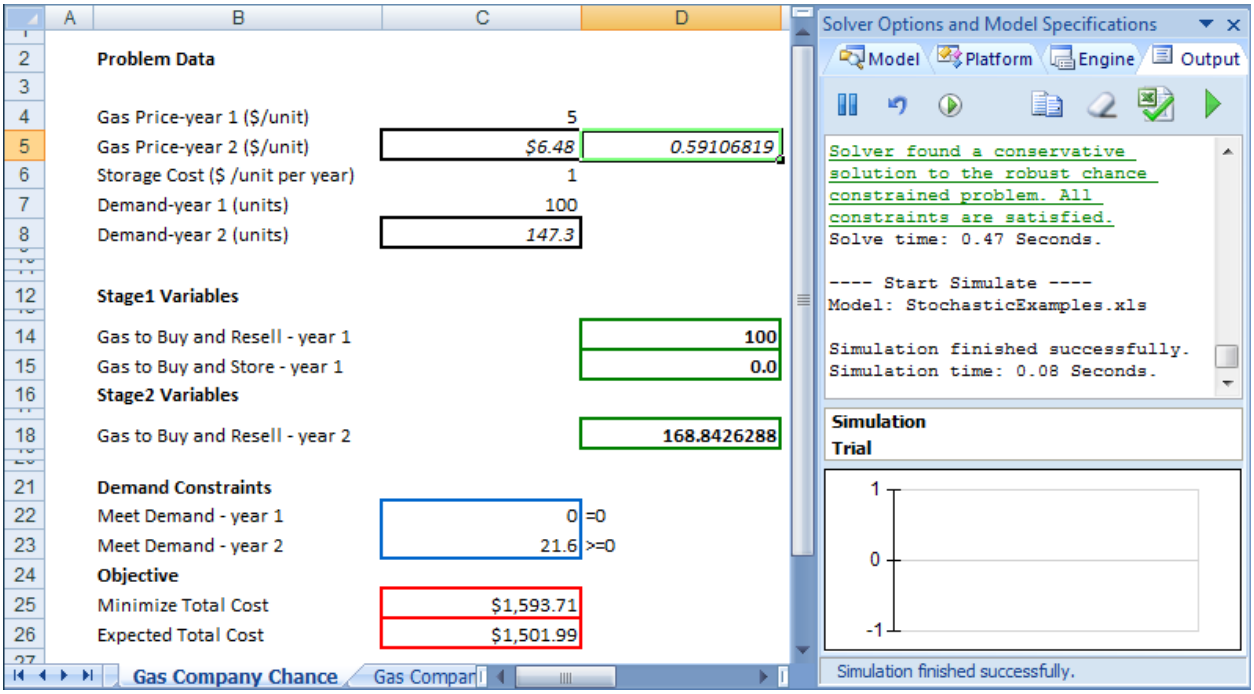


Model Diagnosis			
Model Type	LP Convex		
Variables - Functions - Dependencies			
	Vars	Fcns	Dpns
All	9	8	19
Smooth	9	8	19
Linear	9	8	19
Recourse	0	0	0
Uncertain	0	0	0
Other Model Elements			
Bounds	7		
Integers	0		
Chance Constraints	0		
Sparsity	26.39		

The *transformed* model has 9 variables and 8 functions (instead of 3 and 3), but it has *0 uncertainties* – it's a conventional linear programming model. Solving this robust counterpart model will give us an *approximate* (and somewhat conservative) solution to the original stochastic LP problem.

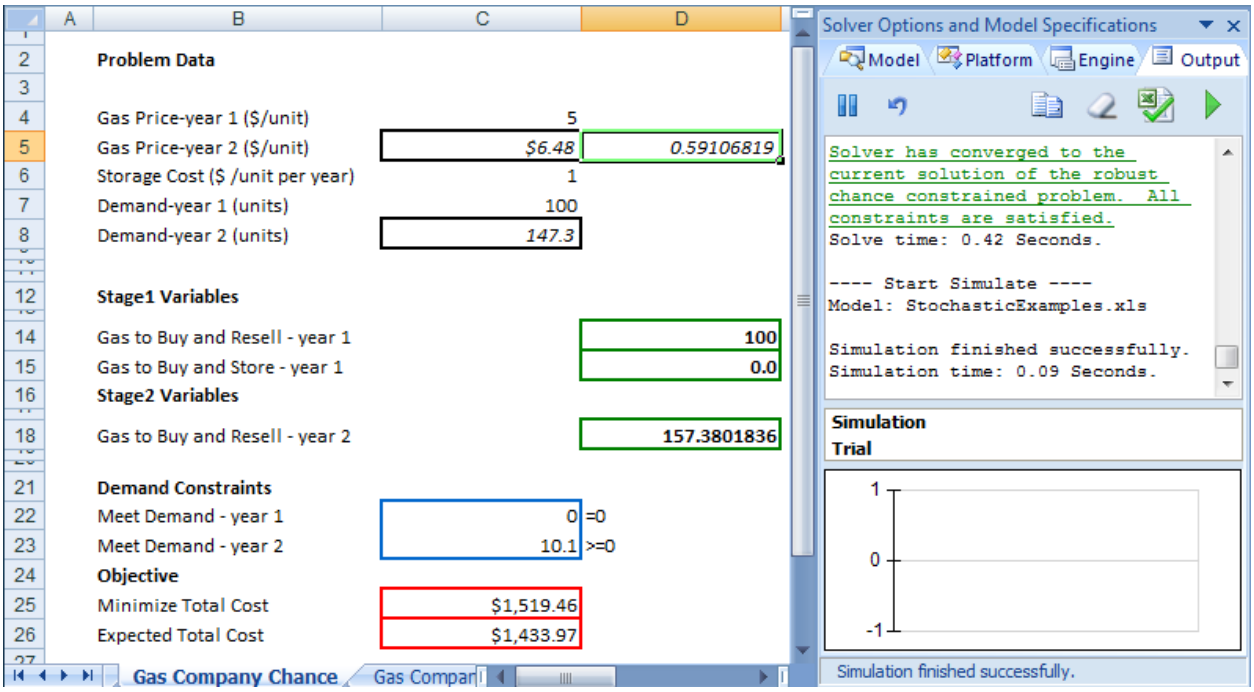
Technical note, explained in the chapter "Mastering Simulation and Risk Analysis Topics:" The robust counterpart model is an LP because the Platform tab Chance Constraints Use option is set to D Norm. If it had been set to L2 Norm, the robust counterpart model would have been an SOCP.

Now click the **Optimize** button, or the green arrow on the Task Pane. In a fraction of a second, the solution appears, with the message "Solver found a conservative solution to the robust chance constrained problem. All constraints are satisfied." You can **click this message to display Help** with a more complete explanation of what this means.



We run one simulation to see the expected value of the objective, which is about $1,502, lower than the initial value of $1,693. But our chance constraint is not only satisfied but *over*-satisfied – we meet consumer demand about 99% of the time, when we asked for only 95%. This solution is more conservative than necessary.

But Risk Solver Platform can *automatically improve* this solution, by adjusting the size of the uncertainty set for the chance constraint. To do this, click the special button to "Auto Adjust Chance Constraints" that appears at the top of the Task Pane. After a moment, a new solution appears with the message "Solver has converged to the current solution of the robust chance constrained problem. All constraints are satisfied," as shown on the next page.



We run one simulation to see the expected value of the objective, which is about $1,434. As noted earlier, this very simple model could have been solved via simulation optimization; but with robust optimization, we can *scale up* a model like this one to large size, and solve it efficiently using the LP/Quadratic Solver, or any of several large-scale Solver Engines.

A Model with Recourse Decisions

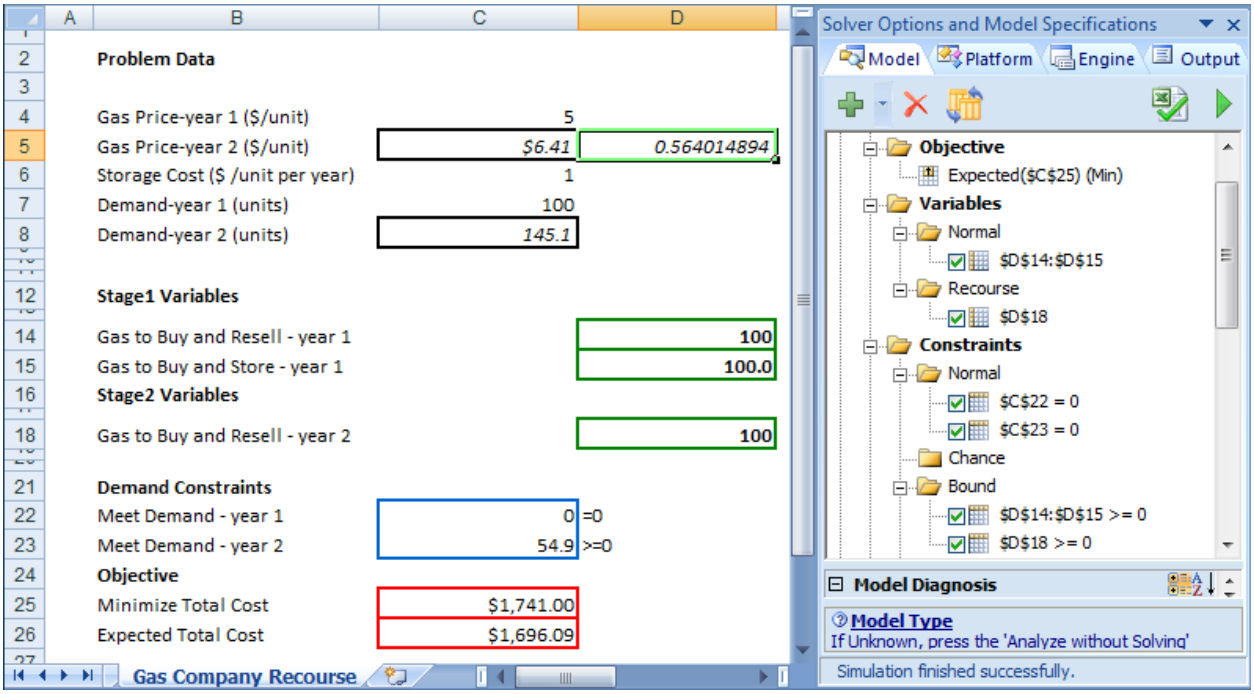
We now have a good solution to the Gas Company problem, found with either simulation optimization or robust optimization. But *we can do better than this* – we can reduce our expected total cost to $1,276, a significant improvement.

We can do this provided that *the business situation permits us* to make the decision of how much gas to purchase next year on a 'wait and see' basis, *after* the uncertainty of the weather is resolved – a so-called **recourse decision**. In this example, our gas supplier must accept our purchase order next year, and not require us to commit to a specific amount of gas this year for delivery next year.

One of the most important messages we can convey is this: **If the business situation permits you to make some decisions on a 'wait and see' basis, it's crucial to *include* this in your optimization model.** If you *don't* do this, you may find an optimal solution to *the wrong model* for the actual business problem.

We emphasize this because so many people have built models that *don't* include 'wait and see' decisions, because (i) they haven't learned this concept and (ii) their software hasn't allowed them to create models with recourse decisions. Simulation optimization is very popular, but as described in the technical literature and implemented in software, ***it has no concept of recourse decisions***.

In the next example, Gas Company Recourse, we'll assume that our decision of how much gas to purchase next year *can* be made on a 'wait and see' basis, *after* the uncertainty of the weather has been resolved. Click the **Gas Company Recourse** tab to display the model, which is pictured on the next page.



This model has the same objective (C25), the same decision variables (D14, D15 and D18), and the same constraints (C22 and C23) as in EXAMPLE2. It differs in just two ways: (i) Cell D18 – the amount of gas to purchase next year – appears as a **recourse decision** variable. (ii) Cell C23 is **no longer a chance constraint** – it is a normal constraint, and in fact it's = rather than >= – we expect to satisfy next year's consumer demand *exactly*. The recourse decision variable allows us to do this – in each possible future scenario, once the weather uncertainty is resolved and consumer demand is known, we will purchase just the amount of gas we need to meet demand.

Solving with Robust Optimization

We cannot solve Gas Company Recourse with simulation optimization, since this method has no concept of recourse decisions. But we *can* use robust optimization methods to transform and solve Gas Company Recourse, much as we did with Gas Company Chance. We can proceed in much the same way. If we analyze the structure of the original model (by clicking the Analyze button with the **Solve Uncertain Models** option set to **Simulation Optimization**), we find that it is a stochastic LP:

Model Diagnosis			
Model Type	Stochastic LP		
Variables - Functions - Dependencies			
	Vars	Fcns	Dpns
All	3	3	6
Smooth	3	3	6
Linear	3	3	6
Recourse	1	2	2
Uncertain	1	2	2
Other Model Elements			
Bounds	3		
Integers	0		
Chance Constraints	1		
Sparsity	66.67		

Again the model has 3 variables, 3 functions, and 1 uncertainty – but now 1 of the 3 variables is a **recourse decision** variable. Note that the recourse variable plays a role in 2 functions – the objective, and the constraint that requires us to meet next year's consumer demand. Our recourse decision variable will have ***many possible values*** **at the optimal solution** – *one for each realization* of the uncertain demand. With default settings in Risk Solver Platform (1,000 Monte Carlo trials in a simulation), the Solver will find 1,000 different values for D18.

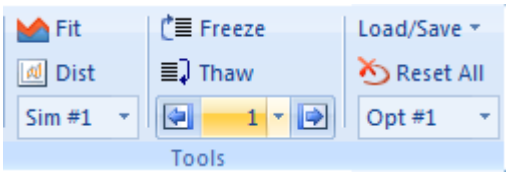
If we analyze the structure of the robust counterpart model (by clicking the Analyze button with the **Solve Uncertain Models** option set to **Automatic** or **Stochastic Transformation**, and the **Stochastic Transformation** option set to **Robust Counterpart**), the result is:

Model Diagnosis			
Model Type	LP Convex		
Variables - Functions - Dependencies			
	Vars	Fcns	Dpns
All	9	8	20
Smooth	9	8	20
Linear	9	8	20
Recourse	0	0	0
Uncertain	0	0	0
Other Model Elements			
Bounds	7		
Integers	0		
Chance Constraints	0		
Sparsity	27.78		

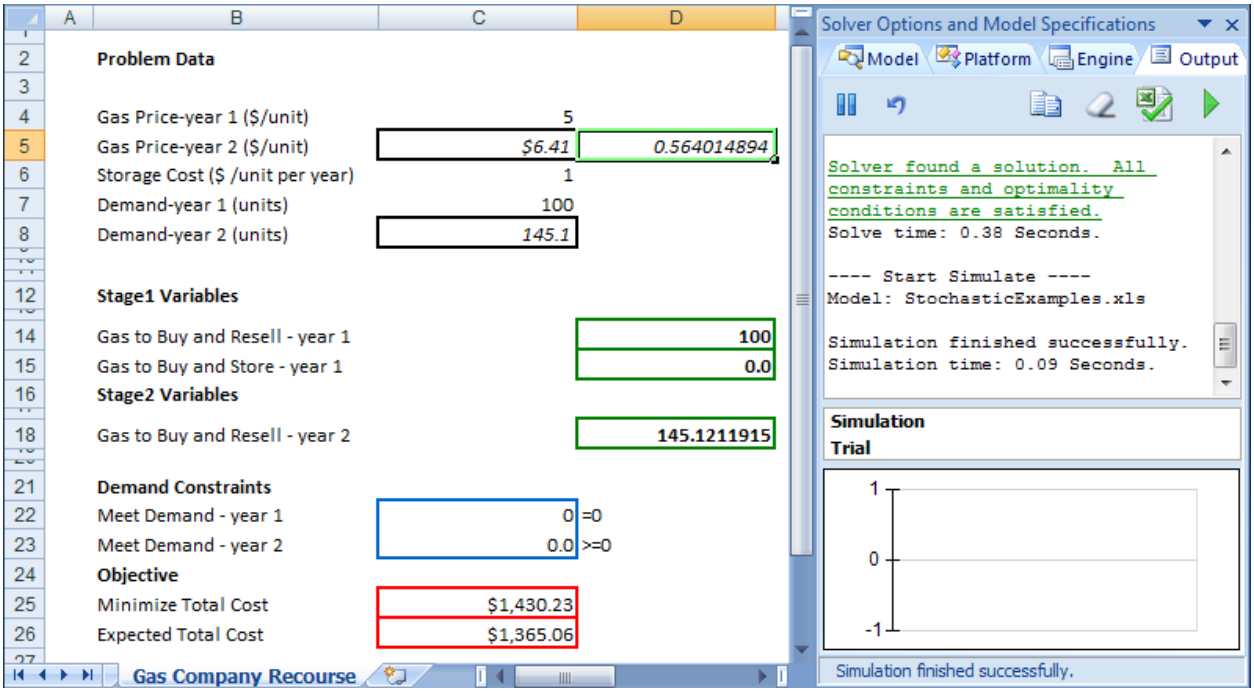
The robust counterpart has 9 variables and 8 functions, but the uncertainty and the recourse decision have been eliminated, yielding a conventional linear programming model. As with Gas Company Chance, solving this robust counterpart model will give us an *approximate* solution to the original problem.

Now click the **Optimize** button, or the green arrow on the Task Pane. In a fraction of a second, the solution appears, with the message "Solver found a solution. All constraints and optimality conditions are satisfied." As shown on the next page, the expected value of the objective is now about $1,365– a considerable improvement over the minimum cost of about $1,434 in Gas Company Chance.

As noted above, our recourse decision variable will have ***many possible values at the optimal solution*** – *one for each realization* of the uncertain demand. We can see these different realizations (there are 1,000 different values) by clicking the left and right arrows on the Ribbon, which display different Monte Carlo trials from the last simulation:



If you examine the trials, you'll see the pattern: The optimal solution is to not store any gas in Year 1 for use in Year 2, but instead to buy just enough gas to meet the uncertain demand in Year 2. (With higher gas costs or lower storage costs, the optimal solution might change.).



This solution offers a lower *expected* cost (the average cost over *all future* scenarios) because of the *flexibility* introduced by use of a recourse decision. And we were able to do this because we created the *right kind of model* for the business situation.

The 'moral of this story' is that Risk Solver Platform gives you the power to solve such problems, using three different technologies – simulation optimization, stochastic programming, and robust optimization. But it's *up to you* to take advantage of the concept of recourse decisions – when they can be used – and create the *right kind of model* for your company's business situation.

Examples: Parameters and Sensitivity Analysis

Introduction

This chapter introduces the use of **parameters**, and functions for **sensitivity analysis** in Risk Solver Platform, with examples. In Version 11, the role of parameters – for optimization, simulation, and sensitivity analysis – was enhanced, and sensitivity analysis functions were added.

Sensitivity analysis is designed to help you explore a conventional Excel spreadsheet model, often before you define an optimization or simulation model. The goal of sensitivity analysis is to identify the key input parameters that have the *greatest impact* on results of interest in your model (such as Net Profit), and to see the effect of varying these key input parameters over a range, in reports and graphs. Risk Solver Platform helps you *automatically identify* sensitivity parameters, and *automatically vary* them to produce reports and graphs.

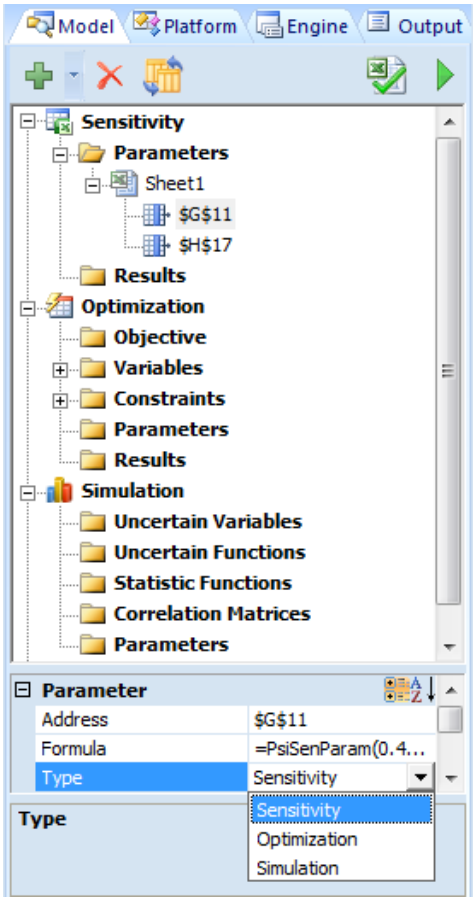
Parameters and Results

Risk Solver Platform and Risk Solver Pro support three kinds of parameters:

- A **sensitivity parameter** (PsiSenParam) is automatically varied when you perform a sensitivity analysis. You do this by selecting **Sensitivity** from the Reports or Charts galleries on the Ribbon.
- An **optimization parameter** (PsiOptParam) is automatically varied when you perform multiple optimizations. You do this by clicking the Optimize button on the Ribbon, or the green arrow in the Task Pane, with the Number of Optimizations set to a value greater than 1.
- A **simulation parameter** (PsiSimParam) is automatically varied when you perform multiple simulations. You do this by clicking the Simulate button on the Ribbon, or the green arrow in the Task Pane, with the Number of Simulations set to a value greater than 1.

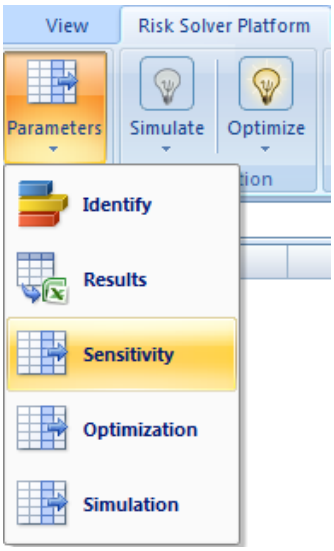
Viewing Parameters in the Task Pane

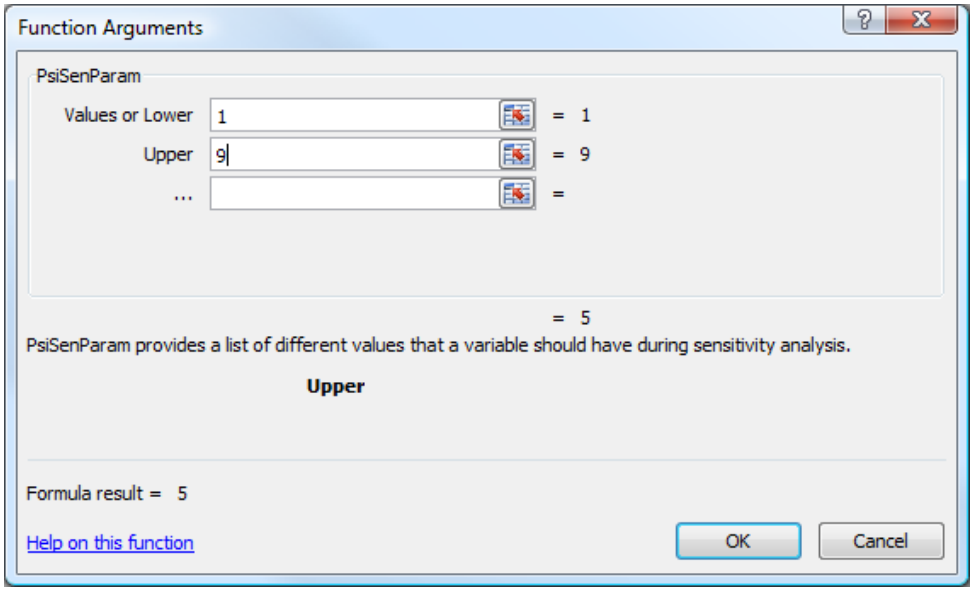
All three types of parameters appear in the Task Pane Model outline. In the example on the next page, two sensitivity parameters at G11 and H17 have been defined by the user. Parameter G11 is selected, and its properties are shown in the lower part of the Task Pane. It is possible to **change the type** of a parameter, by simply selecting from the dropdown list in the lower part of the Task Pane as shown: For example, to use G11 in a simulation model, automatically varied across multiple simulations, you would select **Simulation** from the dropdown list.



Defining a Parameter

You can manually define an input cell as a parameter by simply selecting the cell, choosing **Parameters** and **Sensitivity**, **Optimization** or **Simulation** from the Ribbon, and entering a lower and upper limit on values for the parameter, or a list of explicit values for the parameter. In the example on the next page, we are defining a sensitivity parameter.





You can also simply type a formula such as =PsiSenParam(1,9) in a cell, which has the same effect as the above steps. In both cases, the parameter appears in the Task Pane Model outline.

If you specify a lower and upper limit, the **step size** for successive values is determined when you perform a parameterized sensitivity analysis, or multiple parameterized optimizations or simulations. Risk Solver Platform will subdivide the range from the lower to the upper limit into equal-size intervals.

For example, for the above sensitivity parameter, you would select **Reports** or **Charts** – **Sensitivity** – **Parameter Analysis** and specify the number of points on the major axis. If you specified 18 points, the step size would be 0.5.

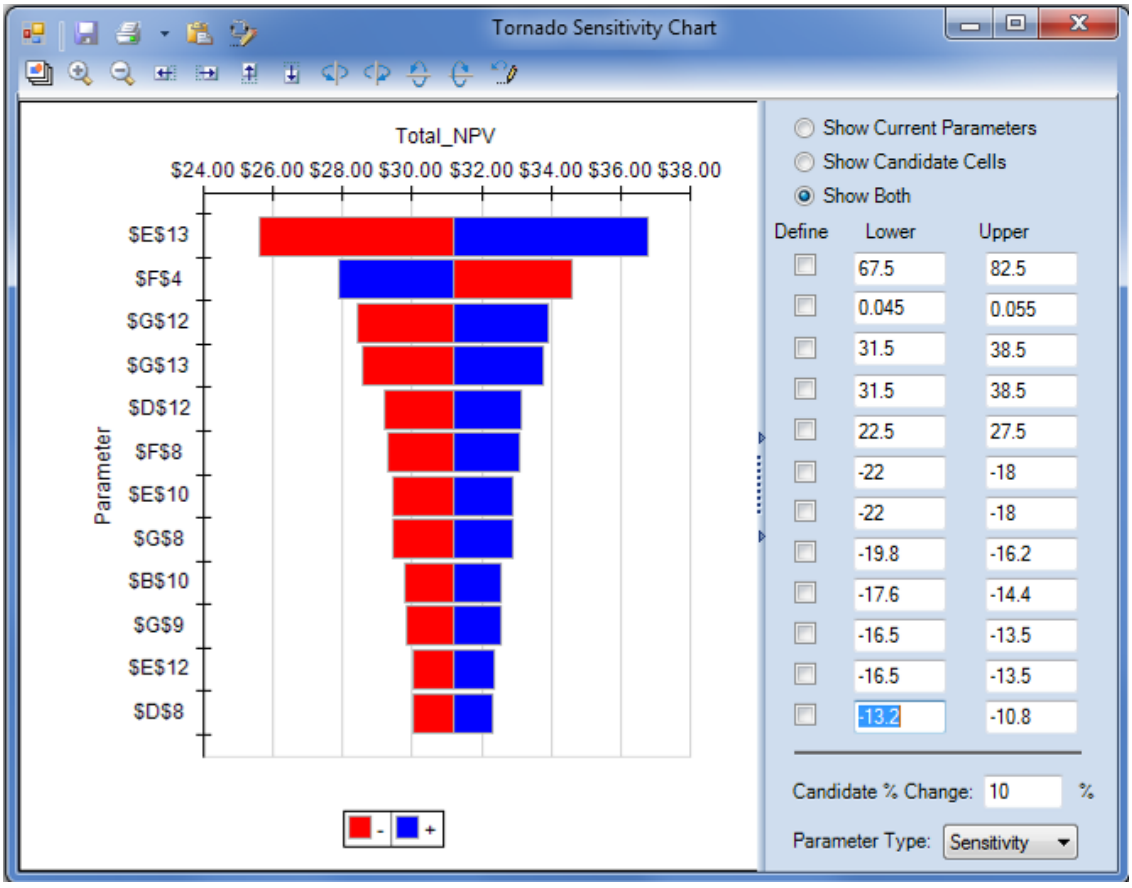
Automatic Parameter Identification

Risk Solver Platform can *automatically* find the input cells with the greatest impact on any selected formula cell. You can then choose one or more of these input cells to serve as sensitivity (or optimization or simulation) parameters.

To do this, **select any formula cell in an Excel model, and choose Parameters – Identify from the Ribbon**.

Risk Solver Platform traces through all 'precedent' formulas to find input cells (leaves of the formula tree) on which the selected formula cell depends. It then varies each input cell independently, computing its impact on the selected formula cell. The input cells are ranked in descending order by (absolute) impact, and the highest-impact cells are displayed in the Tornado chart.

In the example below, we've used **Help – Optimization Examples**, clicked on **Finance Examples.xls**, and selected the **Budget2** worksheet. We've set cells B17:G17 to 1 (if these cells are all 0, then no projects are selected and the Total NPV is always 0). To see which cells have the greatest impact Total NPV, we simply select that cell (I28) and choose **Parameters – Identify** from the Ribbon. After a moment, a Tornado chart like the one below appears.



The chart shows us, at a glance, that Total NPV is most sensitive to cell E13 – which is a large positive cash flow in Year 6 for Opportunity 4. It is next most sensitive (in the opposite direction, decreasing when the parameter increases) to the interest rate in cell F4.

The right panel of the chart shows the lower value and upper value that were used for each input cell when computing the formula value. An ordinary cell containing a number is set to its current value – *n*% and its current value + *n*%, where *n*% (initially 10%) is specified in the edit box in the lower right corner of the chart. If any input cell is already *defined as a parameter*, the lower and upper limit arguments of its PsiXxxParam() call are used instead.

By choosing the radio button corresponding to "Show Current Parameters", "Show Candidate Cells", or "Show both" at the top of the right panel, you can show either one or both types of input cells in the Tornado chart. By **checking the box on the same row** as the cell, you can *define* a candidate cell as a **sensitivity parameter** (or, if you change the selection in the dropdown list in the lower right corner of the chart, an **optimization** or **simulation parameter**). These definitions take effect when you click the Save icon at the upper left corner of the chart. For this example, we'll check the boxes on the same rows as cells **E13** and **F4**; we'll also set F4's Lower value to **0.01** (1%) and the Upper Value to **0.10** (10%).

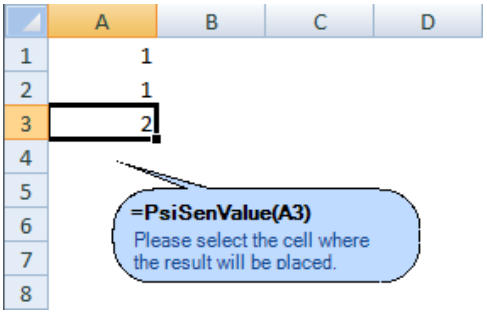
The simplest way to remove a parameter is to select the parameter cell on the worksheet and press the DEL key – which erases its cell formula – or type a number value that will overwrite the formula in the cell. The parameter will then disappear from the Task Pane.

Defining Results

When you perform a sensitivity analysis (which will automatically vary sensitivity parameters), or multiple optimizations or simulations (which will vary optimization or simulation parameters), Risk Solver Platform will keep track of one or more result cell values, corresponding to each parameter value.

- For optimizations, Risk Solver Platform tracks the final objective and decision variable values found by the Solver; you can also track a constraint value by setting its Monitor property in the Task Pane.
- For simulations, Risk Solver Platform tracks the values of all uncertain functions (simulation outputs, referenced inPsiOutput() or PSI Statistics function calls).
- For sensitivity analysis, Risk Solver Platform tracks results from the cell *currently selected* at the time you perform the analysis, and any other results you define.

To define a sensitivity result cell, select the cell (such as A3 below – always a formula cell) on the worksheet, and choose **Parameters – Results** from the Ribbon. A small "balloon" will appear, containing a function call such as =PsiSenValue(A3). You can move the mouse to "drag and drop" this formula into an empty cell. When you do this, cell A3 is defined as a sensitivity result cell, and appears in the Task Pane Model outline under Sensitivity Results.



Sensitivity Analysis Reports and Charts

The **Parameters – Identify** step just described performs a simple kind of sensitivity analysis; in some situations, this may be all that you need. But it is often useful to document the sensitivity of some computed result to systematic variations in the values of one or more parameters. Risk Solver Platform can produce both reports and charts to document these results.

How Parameters are Varied

In reports and charts, Risk Solver Platform can automatically vary your parameters' values in two ways:

1. Varying **all parameters simultaneously**, from their respective lower to upper limits, for the number of steps that you specify.
2. Varying **two parameters independently**, from their lower to upper limits, computing a result for all combinations of the two parameters.

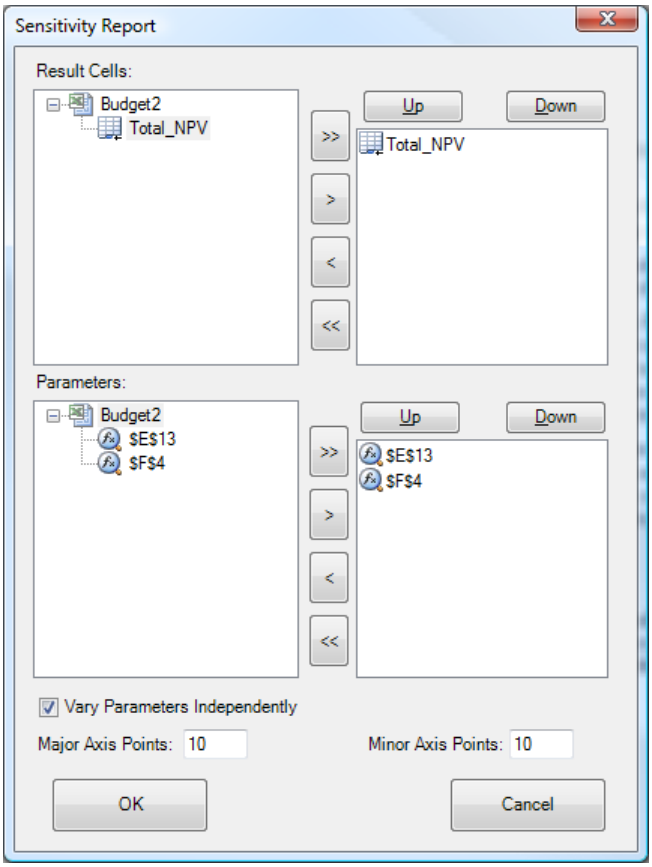
You select the second choice by checking the box "Vary Parameters Independently" at the bottom of the dialog shown on the next page. If you do

this, you must select *exactly one* result cell, and *exactly two* parameter cells in the dialog below. A report will appear as a two dimensional table, and a chart will appear in 3-D with the two parameters on the axes.

If you don't check this box – meaning that all parameters will be varied simultaneously – you can select one or more parameter cells, and one or more result cells in the dialog below. A report will contain a column for each parameter and each result, and a chart will contain a plot of each result cell.

Creating Sensitivity Reports

To create a sensitivity analysis report, select a formula cell on the worksheet (whose value should be computed in the report), and choose **Reports – Sensitivity – Parameter Analysis** from the Ribbon. A dialog like the one shown below will appear.

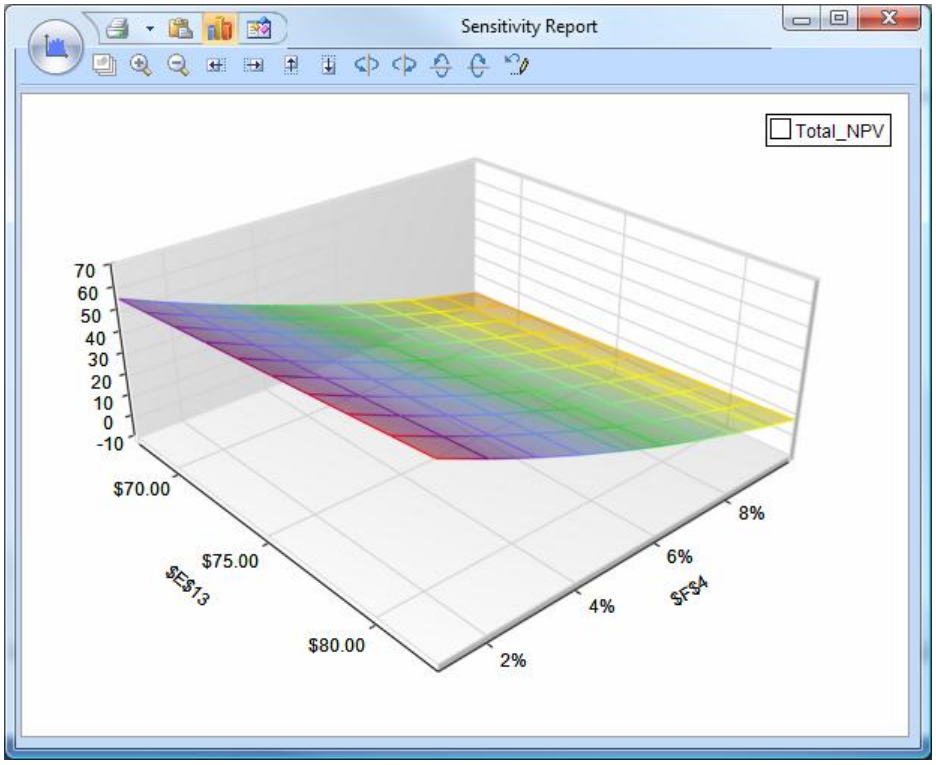


Use the arrow buttons in the top section to select one or more result cells (exactly one if you check the box "Vary Parameters Independently"). Use the arrow buttons in the bottom section to select one or more sensitivity parameter cells (exactly two if you check the box "Vary Parameters Independently"). Then click OK to produce the report, as a new worksheet in into your Excel workbook, just to the left of the active worksheet. An example is shown below.

	A	B	C	D	E	F	G	H	I	J	K
1	Total_NPV : $E$13 by $F$4										
2											
3											
4	$E$13	$F$4									
5		1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
6	$67.50	$55.56	$47.19	$39.43	$32.25	$25.59	$19.41	$13.69	$8.38	$3.46	($1.11)
7	$69.17	$57.13	$48.67	$40.83	$33.56	$26.83	$20.59	$14.80	$9.43	$4.45	($0.17)
8	$70.83	$58.70	$50.15	$42.22	$34.88	$28.07	$21.76	$15.91	$10.48	$5.45	$0.78
9	$72.50	$60.27	$51.63	$43.62	$36.20	$29.32	$22.94	$17.02	$11.53	$6.44	$1.72
10	$74.17	$61.84	$53.11	$45.01	$37.51	$30.56	$24.11	$18.13	$12.58	$7.43	$2.66
11	$75.83	$63.41	$54.59	$46.41	$38.83	$31.80	$25.29	$19.24	$13.63	$8.43	$3.60
12	$77.50	$64.98	$56.07	$47.81	$40.15	$33.05	$26.46	$20.35	$14.68	$9.42	$4.54
13	$79.17	$66.55	$57.55	$49.20	$41.47	$34.29	$27.64	$21.46	$15.73	$10.41	$5.48
14	$80.83	$68.12	$59.03	$50.60	$42.78	$35.53	$28.81	$22.57	$16.78	$11.41	$6.42
15	$82.50	$69.69	$60.51	$51.99	$44.10	$36.78	$29.99	$23.68	$17.83	$12.40	$7.36

Creating Sensitivity Charts

To create a sensitivity analysis chart, select a formula cell on the worksheet (whose value should be computed in the chart), and choose **Charts – Sensitivity – Parameter Analysis** from the Ribbon. A dialog like the one on the previous page will appear. Follow the same steps as for sensitivity analysis reports, and click OK. A chart like the one below will appear:



You can use icons on the toolbar at the top of the chart to resize and rotate the chart, print the chart, or copy it to the Windows Clipboard (where you can paste it into other applications).

Optimization and Simulation Reports and Charts

Reports and charts for multiple parameterized optimizations and simulations work in much the same way as the reports and charts for sensitivity analysis shown in the previous section. But where for sensitivity analysis, Risk Solver Platform simply performs a **worksheet recalculation**, for these reports and charts, Risk Solver Platform performs a **complete optimization or simulation** for each set of parameter values.

To produce these reports and charts, select **Reports** or **Charts – Multiple Optimizations** from the Ribbon. For multiple parameterized simulations, select **Reports** or **Charts – Multiple Simulations** from the Ribbon.

When Optimizations and Simulations are Run

As shown in the earlier Examples chapters, optimizations are run when you click the Optimize button on the Ribbon, or the green arrow in the Task Pane, and simulations are run when you click the Simulate button on the Ribbon, or (assuming there's no optimization model) when you click the green arrow in the Task Pane. When you select Reports and Charts of Multiple Optimizations or Multiple Simulations *other than* Parameter Analysis, the results **cached in memory** from the last optimization or simulation run are used to create the reports or charts.

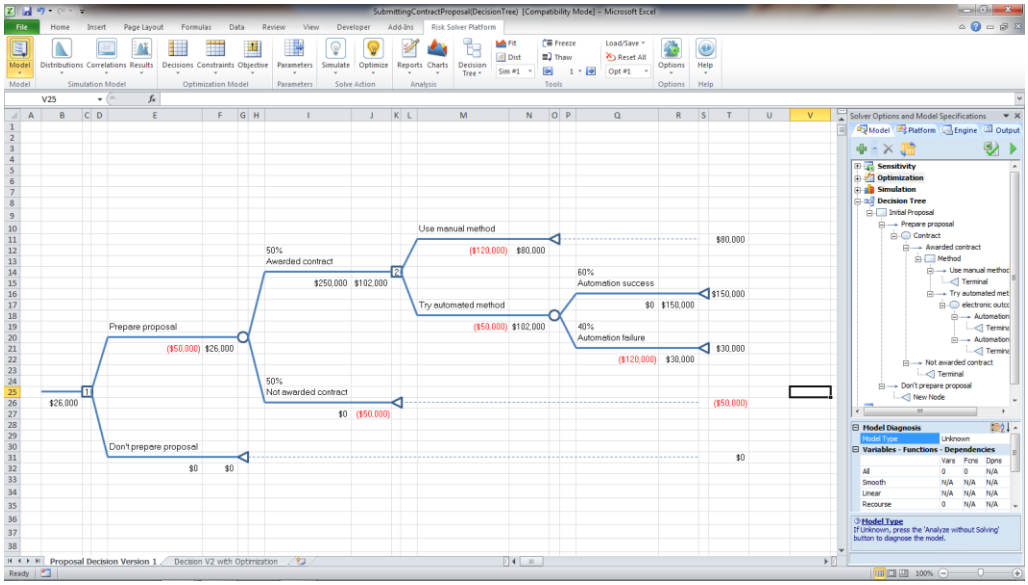
However, when you select Reports and Charts of Multiple Optimizations or Multiple Simulations with **Parameter Analysis**, which gives you greater control over how parameters are varied, Risk Solver Platform cannot use the results cached in memory. Instead, a new set of optimizations or simulations is run, varying the values of the parameters as you've specified; you can for example vary two parameters independently, yielding all combinations of these parameter values. The number of optimizations or simulations to run is independent of the settings you use for Number of Optimizations or Number of Simulations in the Task Pane Platform tab; it is the ***product*** of the number of **Major Axis Points** and **Minor Axis Points** you select in the Reports or Charts selection dialog. With the default value of 10 points each, 100 optimizations or simulations are required. This can take some time, especially for optimizations; a progress dialog is displayed during this process.

Examples: Decision Trees

Introduction

This chapter introduces **decision trees** in Risk Solver Platform. Such trees are a useful graphical and computational aid for problems that involve sequential decisions and events, where there are a small number of decision alternatives at each step, and a small number of alternative outcomes for each event.

In the example below, we've used **Help – Examples**, and clicked on the Decision Tree examples button and the Submitting a Contact Proposal example, which is based on an example by Prof. Mike Middleton.



In this example, a firm must decide (1) **whether to prepare a proposal** for a contract, and (2) if the contract is awarded, **what method to use** to satisfy the contract. The first number shown below each branch (horizontal line) is the amount the firm pays out (if negative) or receives (if positive) when it follows that branch. For example, in the path that ends at the terminal node at cell S21, the firm (1) pays $50,000 to prepare the proposal, (2) receives $250,000 up front when awarded the contract, (3) spends $50,000 to try the automatic method, and since that method fails, (4) spends $120,000 on the manual method. The net result is a positive cash flow of $30,000 shown at cell T21.

A decision tree is "solved" by Excel formulas calculating each of the numbers you see on the worksheet. **Terminal values** such as T21 sum all the partial cash flows along the path leading to that terminal node. The tree is then **rolled back** by computing expected values (or certainty equivalents) at event nodes, and by

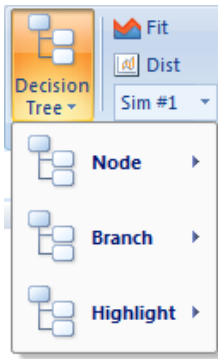
maximizing (or minimizing) the alternative values at decision nodes. The rollback values appear below and just to the left of each node, showing the value of the subtree rooted at that node. The numbers "inside the boxes" at decision nodes show **which alternative is optimal** for that decision. In the example, the 1 in the decision node at C25 indicates that it is optimal to prepare the proposal, and the 2 in the decision node at K14 indicates the firm should try the automatic method because that alternative leads to a higher expected value ($102,000) than the mechanical method ($80,000).

Since all the computations are performed with Excel worksheet formulas, a decision tree is a 'regular' Excel worksheet model, where you can ask what-if, perform sensitivity analysis, or create simulation or optimization models if desired. For example, the formula at cell F21, for the event where we are or are not awarded the contract, is =IF(ABS(1-SUM(I12,I24))<=0.00001,SUM(I12*J15,I24*J27),NA()) – an expected value calculation – and the formula at B26, for the decision whether to prepare a proposal, is simply =MAX(F21,F32)since we are maximizing EV.

The Task Pane Platform tab includes a Decision Tree group of options, where you can choose to maximize or minimize values at decision nodes, and to compute expected values (which are risk neutral) or certainty equivalents (which reflect risk aversion) at each node. If you choose certainty equivalents, you can specify the parameters of an exponential utility function.

Creating Decision Trees

You create and edit a decision tree with the Decision Tree choice on the Ribbon:

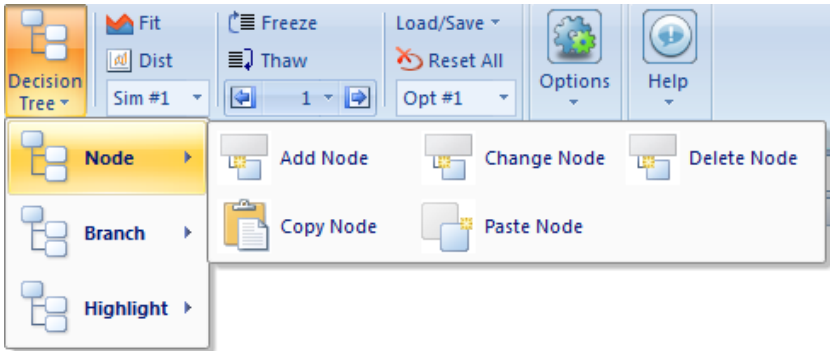


You can also create and edit a decision tree in the Task Pane Model tab, using the Add (+) and Remove (X) icons and the properties area at the bottom of the Task Pane; but you can highlight the best or worst decision strategy (see below) only from the Ribbon.

A decision tree consists of **nodes** and **branches**. Nodes may be either decision **nodes**, **event nodes**, or **terminal nodes**; branches represent alternative choices at decision nodes, and alternative outcomes at event nodes.

Creating and Editing Nodes

Selecting **Node** in the dropdown presents choices for working with nodes:

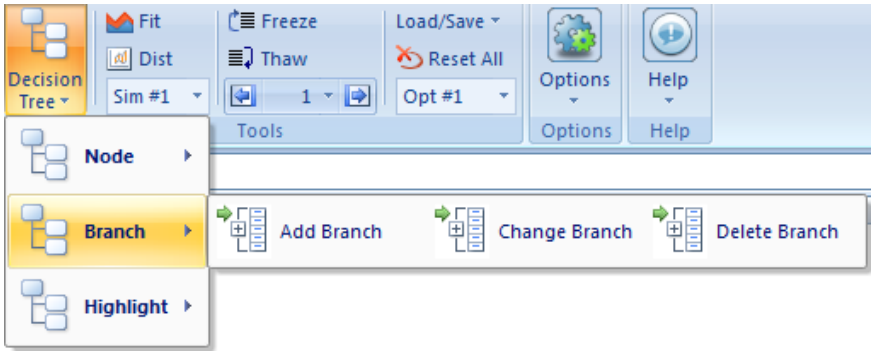


The **Add Node** and **Change Node** options display the dialog shown on the next page. To change a node, you must first *select* it on the Excel worksheet – you may select either the node graphic or one of the immediately adjacent cells.

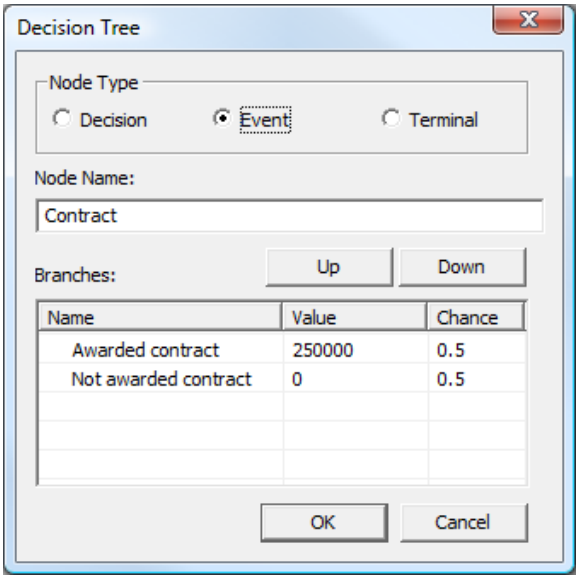
The **Copy Node** and **Paste Node** choices can be used to copy a subtree (rooted at the selected node) and paste the copy at another position in the decision tree; this is very useful when there are identical choices at later stages in the tree.

Creating and Editing Branches

Selecting **Branch** in the dropdown presents choices for working with branches:



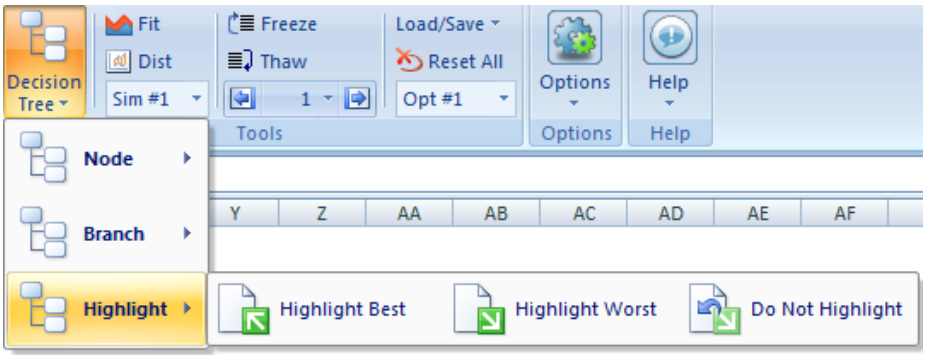
The **Add Branch** and **Change Branch** options display the Decision Tree dialog shown below. To change a branch, you must first *select* it on the Excel worksheet – you may select either the branch graphic or one of the immediately adjacent cells. Note that Add/Change Node and Add/Change Branch options display the *same* Decision Tree dialog: Branches are associated with the node to their left, and you can edit properties of the node and its branches in this dialog.



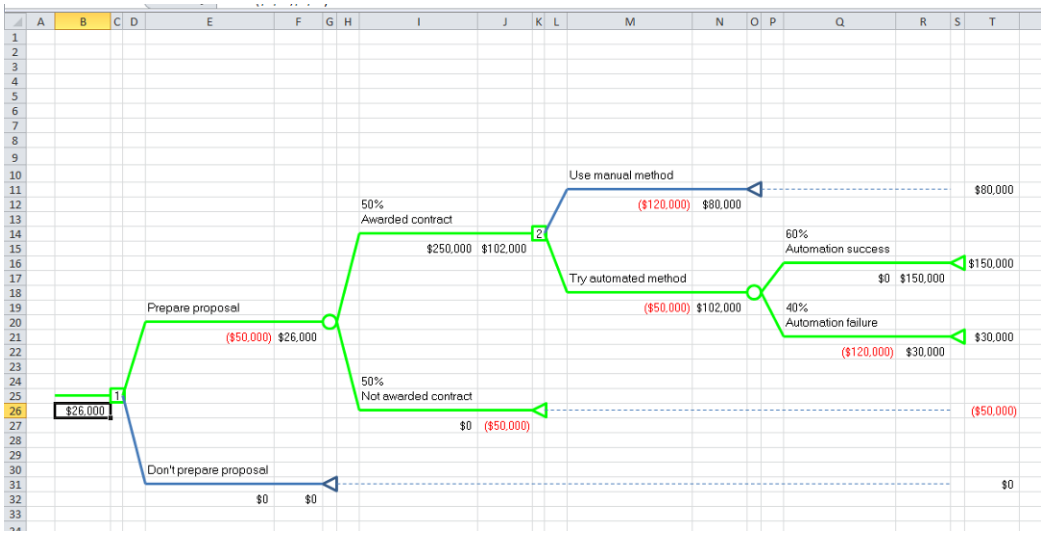
In this dialog, you can change the type of the node, the name of the node, and you can add, edit, reorder, or remove branches for this node. Each branch for a **decision node** has a **value** (the cost or payoff of taking that decision); each branch for an **event node** has a **value** and a **probability** (the cost or payoff, and the probability of occurrence, of that outcome).

Highlighting a Decision Strategy

A *decision strategy* is a complete sequence of decisions that you can make through the decision tree. You can highlight the best or worst decision strategy by selecting **Highlight** in the Decision Tree dropdown list, as shown on the next page. If you are maximizing, the best strategy maximizes EV or CE, and the worst strategy minimizes EV or CE. If you are minimizing, the opposite applies.



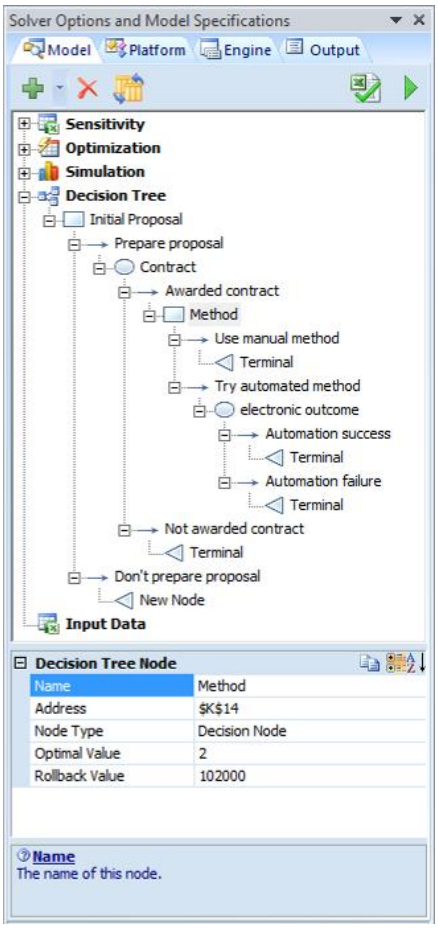
The next page shows the example decision tree shown earlier, with Excel worksheet gridlines turned off, and the best decision strategy highlighted in green. If you choose to highlight the worst decision strategy that will be shown in red.



Decision Trees in the Task Pane

Decision trees may also be viewed and edited directly in the Task Pane Model tab, and choices for tree evaluation (see "Platform tab" below) can only be made in the Task Pane. When a node or branch is selected in the Task Pane outline, properties of that node or branch may be viewed and edited in the lower portion of the Task Pane, as shown on the next page. You can also **double-click a node** in the Model outline to display the same Decision Tree dialog that appears when you add or change a node from the Ribbon.

Model Tab



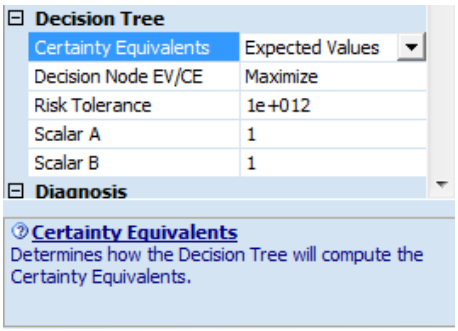
In this example, we've selected the decision node named Method, represented by the graphic symbol at cell K14 on the worksheet. The optimal choice at this node (given that we're maximizing expected value or EV) is branch #2, and the rollback value of the subtree rooted at this node is $102,000.

Note the small **Copy** icon in the upper right corner of the Decision Tree Node properties pane: You can click this icon to **copy a subtree** rooted at this node. When you've selected a Terminal node in the Model outline, you'll notice a small **Paste** icon in this same area: You can click this icon to **paste a subtree** at the position of the Terminal node.

Platform Tab

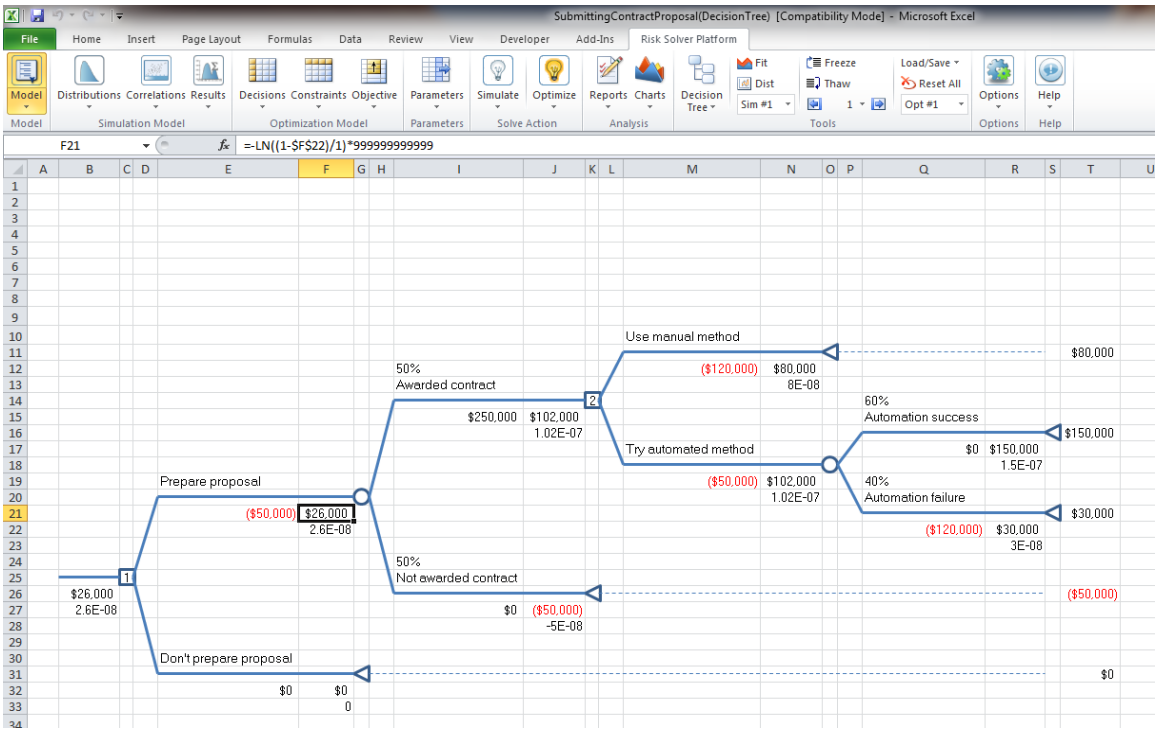
By default, rollback values for a decision tree are computed assuming that you want to Maximize Expected Value or EV. You can change this by setting options in the Decision Tree section of the Task Pane Platform tab, as shown on the next page.

You can select **Maximize** or **Minimize** in the dropdown list for the Decision Node EV/CE option. For the Certainty Equivalents option, you can choose either **Expected Value** or **Exponential Utility Function**. The remaining three options come into play only when you choose Exponential Utility Function – they set parameters of this function.



For the **Maximize** option with exponential utility, the rollback formulas are U = A–B*EXP(X/RT) and CE = -LN((A-EU)/B)*RT, where X and EU are cell references. For the **Minimize** option with exponential utility, the formulas are U = A-B*EXP(X/RT) and CE = LN((A-EU)/B)*RT. RT is the value you specify for the **Risk Tolerance** option on the Platform tab, and A and B are the values of the **Scalar A** and **Scalar B** options on the Platform tab.

When you use an Exponential Utility Function, the utility (U =) and certainty equivalent (CE =) formulas are computed in extra cells on the worksheet, as shown on the next page. For example, B27 computes the **utility** of the decision at B26; B26 contains =MAX(F21,F32) as before, to choose the decision with the maximum **certainty equivalent** value; F21 contains =-LN((1-F22)/1)*999999999999.



Getting Results: Optimization

Introduction

This chapter explains how to obtain and interpret results from optimization in Risk Solver Platform and its subset optimization products: Premium Solver Platform and Premium Solver Pro. We'll also discuss **what can go wrong**, and what to do about it, and also **how to get more** than a single optimal solution from your model.

Risk Solver Platform has many powerful optimization algorithms, and will fully exploit the power of your PC. But the *model that you create* may be relatively easy to optimize (if you use linear functions like SUM) or extremely difficult to optimize (if you use non-convex and non-smooth functions like LOOKUP). The results you get depend on the model you create.

In the following sections, we'll focus on immediate actions you can take when you get an unexpected result – but if you read the chapter "Mastering Conventional Optimization Concepts," you'll learn more about optimization models and solution methods, and better understand *why* the unexpected result appeared, and how to design your model to get the solutions you want.

What Can Go Wrong, and What to Do About It

When you click the **Optimize** button on the Ribbon, or the **green arrow** on the Task Pane to solve, you'll normally get one of these outcomes:

1. A solution that makes sense to you. This is normally accompanied by a Solver Result message in **green** at the bottom of the Task Pane. You can proceed to "When Things Go Right: Getting Further Results."
2. A Solver Result error message that you understand and can correct, in **red** at the bottom of the Task Pane. You can take corrective action.
3. A Solver Result error message that you *don't* understand, in **red** at the bottom of the Task Pane. You should **read the solution log** in the Output tab, **click the error message** and read Help about the message.
4. A *solution* that you *don't* understand, or that seems wrong. Again before doing anything else, you should **read the solution log** in the Output tab, **click the error message** to display Help, **run available reports** as described below, and **read the section** below "When the Solution Seems Wrong."
5. Solving runs for a very long time, and you don't get a solution or a Solver Result message until you press ESC or click Pause/Stop. You should **read the section** below "When Solving Takes a Long Time."

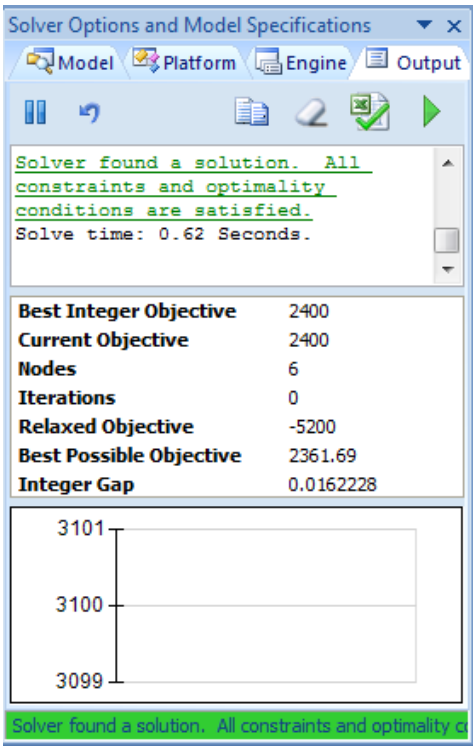
In rare cases, you might find Excel shutting down or "locking up" (so nothing happens when you press and hold the ESC key for several seconds). In this case please contact Frontline Systems Technical Support at (775) 831-0300 x4 or

support@solver.com. Some Solver Result error messages ask you to contact Technical Support. If you can send us your model, this will be very helpful.

But experience shows that 99% of all technical support cases involve "pilot error" by the user, and that 90% of all such cases could be easily resolved by reading online Help or the User Guide. So we hope you'll keep reading, and that you'll take these steps before calling technical support!

Review Messages in the Output Tab

Your first step should be to **review the messages** in the **solution log** in the Task Pane Output tab. Below is an example of the Output tab at the solution of EXAMPLE5 in **StandardExamples.xls** (described more fully in the chapter "Examples: Conventional Optimization"):



If you click the Copy  icon, the contents of the solution log will be copied to the Windows Clipboard, where you can paste it into Microsoft Word, NotePad, or an **email message to Frontline Systems Technical Support**. Below is the complete solution log from the above example:

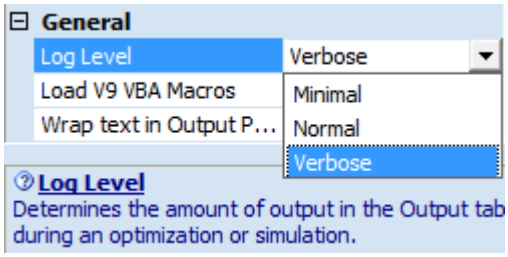
```
---- Start Solve ----
Using: Full Reparse.
Parsing started...
No uncertain input cells.
Diagnosis started...
Model diagnosed as "NSP".
Attempting Transformation.
Using: Full Reparse.
Parsing started...
Diagnosis started...
Model transformed into an LP. Transformation will be used.
User engine selection: Standard LP/Quadratic
Model: [OptimizationExamples.xls]EXAMPLE5
Using: Psi Interpreter
```

```
Parse time: 0.22 Seconds.

Engine: Standard LP/Quadratic
Setup time: 0.00 Seconds.

Engine Solve time: 0.20 Seconds.

Solver found a solution.  All constraints and optimality
conditions are satisfied.
Solve time: 0.62 Seconds.
```

This was a *successful* solution – but in cases where you have a Solver Result *error message* that you don't understand, or a *solution* that you don't understand, the solution log can be quite helpful.

More Detail in the Solution Log

You can obtain *more detailed* output in the solution log by setting the Task Pane Platform tab General group **Log Level** option to **Verbose** before you solve:



Below is a portion of the solution log (from the LP/Quadratic Solver) for the EXAMPLE5 model:

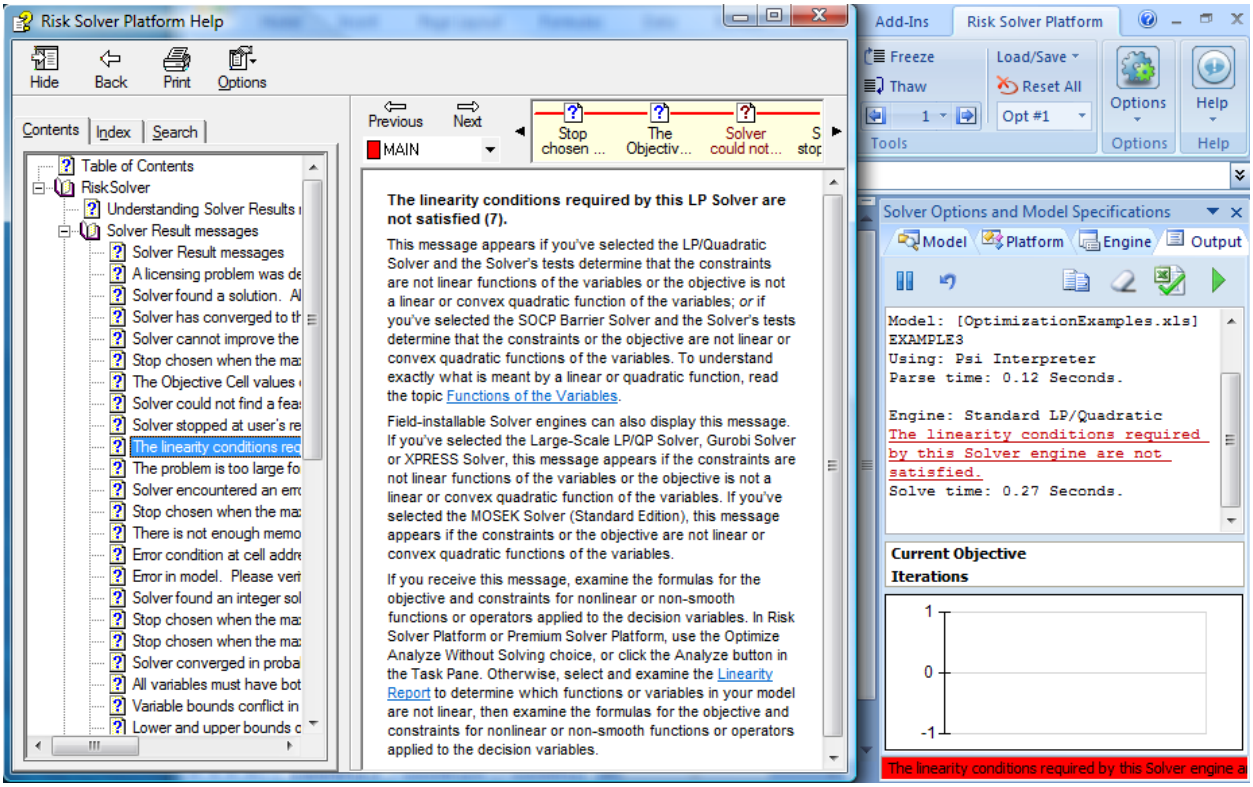
```
Integer solution of 3100 found by Feasibility Pump after 0
iterations and 0 nodes (0.01 seconds)
After 0 nodes, 1 on tree, 3100 best solution, best possible
2355.01 (0.18 seconds)
After 1 nodes, 2 on tree, 3100 best solution, best possible
2355.01 (0.18 seconds)
Integer solution of 2630 found by Unknown after 285 iterations
and 2 nodes (0.19 seconds)
After 2 nodes, 2 on tree, 2630 best solution, best possible
2361.69 (0.19 seconds)
Integer solution of 2400 found after 294 iterations and 3 nodes
(0.20 seconds)
After 3 nodes, 1 on tree, 2400 best solution, best possible
2361.69 (0.20 seconds)
Engine Solve time: 0.20 Seconds.
```

If you are having problems finding the solution you want or expect, this detailed log can sometimes be helpful.

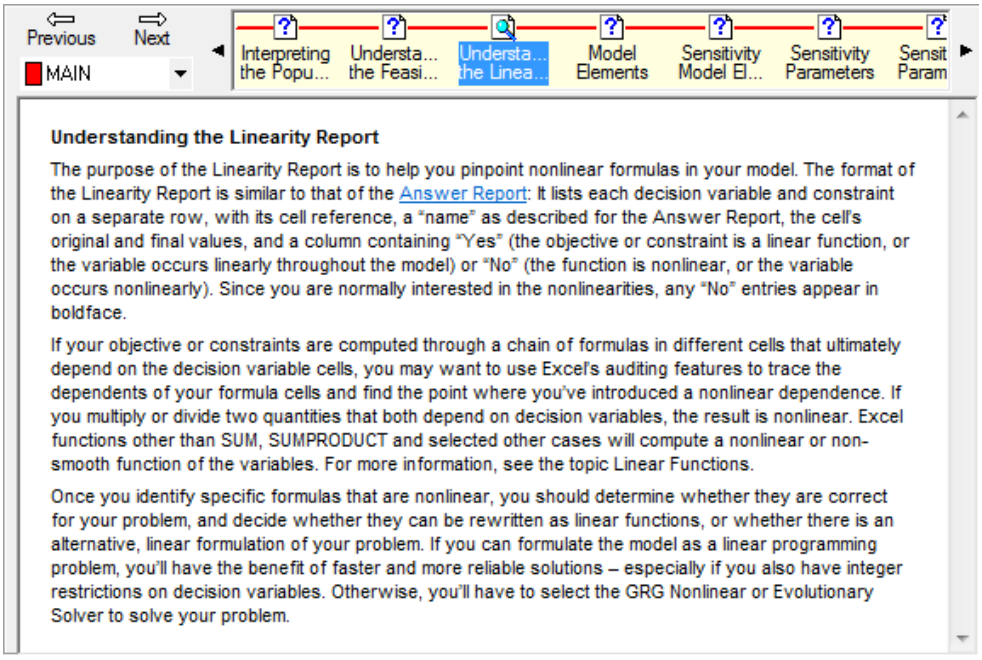
Click the Solver Result Message for Help

The Solver Result message is always **underlined** – it is a **hyperlink** to Help. If you aren't sure that you fully understand it, **click the link** to open online Help to a detailed discussion of the message.

Below is an example of Help that appears when you solve EXAMPLE3 in **StandardExamples.xls**, and click on the Solver Result error message in red, "The linearity conditions required by this Solver engine are not satisfied."



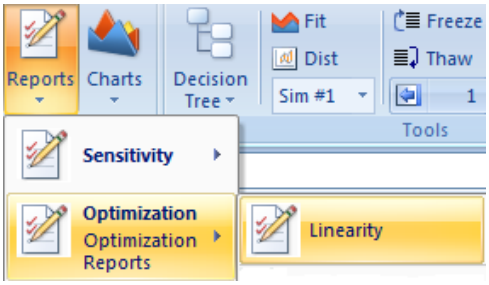
Notice the two hyperlinks in the Help text, to Functions of the Variables and the Linearity Report. If you click the latter link, the Help text below appears.



Given our experience that 90% of technical support cases could be easily resolved by **reading online Help** or the User Guide, you can save yourself, as well as Frontline Systems, time if you *do* read online Help first.

Choose Available Optimization Reports

The Help above suggests that we select the Linearity Report when we receive this Solver Result error message. As a general rule, it's good idea to **examine the available reports** and produce the reports that may help you understand what's wrong with your model. Just select **Reports – Optimization Reports** on the Ribbon – the reports in the gallery are updated each time you solve.



- The **Linearity Report** and (in Risk Solver Platform and Premium Solver Platform) the **Structure Report** can help when you encounter "The linearity conditions required by this Solver engine are not satisfied."
- The **Feasibility Report** and **Feasibility-Bound**s Report can help when you encounter "Solver could not find a feasible solution."
- The **Scaling Report** can help when you encounter either of these messages, or other unexpected messages or solution values. See the section below "Problems with Poorly Scaled Models."

See the section "An 'Accidentally' Nonlinear Model" in the chapter "Examples: Conventional Optimization" to see how the **Linearity Report**, and (in Premium Solver Platform and full Risk Solver Platform) the **Structure Report** produced by the PSI Interpreter can help us pinpoint the nonlinear formulas in this model.

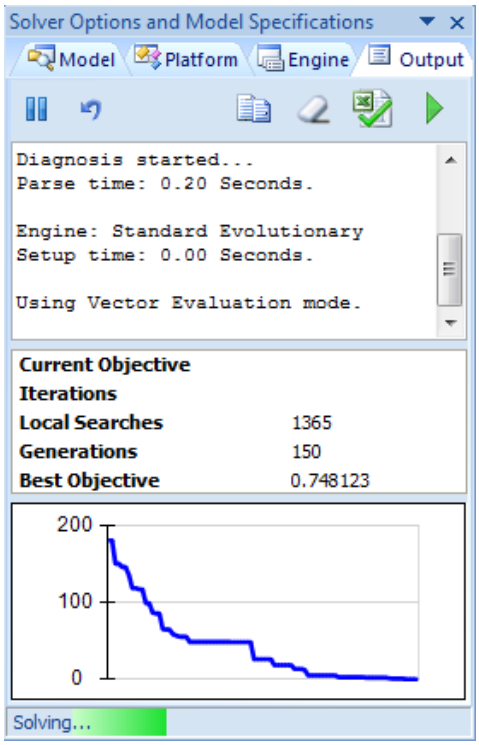
When Solving Takes a Long Time

When your model takes a long time to solve, the Task Pane Output tab can be helpful *during* the solution process – at a minimum, to reassure you that the Solver is still making progress, and has not "hung up." If it is not already visible when you first start solving, the Output tab will **appear automatically** after a few seconds of solution time, as long as the Task Pane itself is visible.

The Output tab shows the objective of the best solution found so far, and for problems with integer constraints, the Best Integer Objective ("incumbent"), the Best Possible Objective ("best bound"), and the Integer Gap or percentage difference between these two objectives.

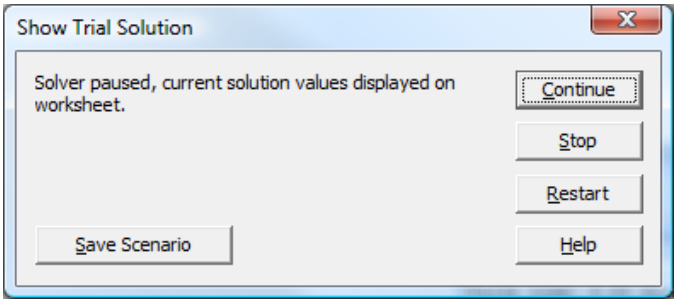
Running Chart of the Objective

The bottom part of the Output tab shows a running chart of the objective of the best solution found so far. On the next page is an example of the Output tab on a problem called **SPACE2.xls** that takes about 2 minutes to solve, with the V11 Evolutionary Solver, on a modern quad core PC:



Interrupting the Solution Process

You can interrupt the solution process at any time, by **pressing the ESC key**, or by **clicking the** ⏸ **button** in the Task Pane Output tab. If your model is very large, you might have to hold down the ESC key for a second or two. Since the input focus may not be on the Task Pane when you click with the mouse, you might have to click the ⏸ button twice. A dialog like the one below will appear:



Click **Continue** if you want to continue solving; click **Stop** to cause the Solver to stop the solution process (this may take a few seconds) and display the message "Solver stopped at user's request." Clicking **Restart** may be useful with the Evolutionary Solver – see the Frontline Solvers Reference Guide.

Reasons Why Solving Takes a Long Time

There are several reasons why solving might take a long time:

1. Most often, you've used Excel formulas or functions that make your model non-smooth or non-convex, and much more difficult to solve. You might gain a lot from consulting assistance, or by reading the chapter "Mastering Conventional Optimization Concepts."

2. If **model diagnosis** or **problem setup** is slow (see timing messages in the Task Pane Output tab), you might be using array formulas, LOOKUP functions with large ranges as arguments, or long "chains" of formulas where each formula depends on an earlier formula.

3. You might be using a Solver Engine, or Engine option settings, that aren't appropriate for your problem. You can **check the box** "Automatically Select Engine" on the Task Pane Engine tab. **Click Engine options** to display Help on each option. You can gain further insight by reading "Optimization Problems and Solution Methods" in the chapter "Mastering Conventional Optimization Concepts."

4. Your model might be well formulated, but very large, or it might require non-smooth functions, or many integer variables. A more powerful plug-in Solver Engine may help considerably, and a faster processor, multi-core processor, or more memory can often help.

Adjusting Automatic Mode Options to Gain Speed

Be sure to read "Automatic Mode and Solution Time" in the chapter "Risk Solver Platform Overview." If you're using the **Automatic** setting (the ***default*** for new models) for any of the seven options described in this section, you can save time by pre-setting these options to the same values chosen automatically.

Model Design for Maximum Optimization Speed

Paying attention to your Excel worksheet layout and design, and avoiding certain practices, will help you get the most from the PSI Interpreter. Below is a brief list of Do's and Don'ts that will help you realize the best possible speed:

- **Do** build worksheets starting from the "upper left corner." **Don't** place cells or formulas at extreme row and column addresses all over the worksheet.
- **Do** build the model on a small number of worksheets. **Don't** include references on these worksheets to other worksheets that aren't required for the simulation model.
- **Do** use numeric and logical formulas and functions. **Don't** create string results in the middle of numeric calculations (see example below).
- **Do** use built-in Excel functions (including Analysis ToolPak functions) freely. **Don't** use third-party or user-written functions unless truly needed. Such functions must not have "side-effects" other than the value returned.
- **Do** use operators like + - * / ^ and functions like SUM, AVERAGE, MAX, MIN, AND, OR, NOT, IF, CHOOSE. **Don't** use INDIRECT, OFFSET, or TEXT, SEARCH, REPLACE, FIXED, DOLLAR, or ROMAN.
- If possible, **Don't** use array formulas, LOOKUP functions with large ranges as arguments, or long "chains" of formulas where each formula depends on an earlier formula (and hence on *all* earlier formulas). Each of these things can slow down analysis of your model considerably (by 10x in some cases).

A common practice that slows down the PSI Interpreter, and is easy to avoid, is illustrated by the following:

A1: =IF(B1>100,"Yes","No") and later A10: =IF(A1="Yes",A2,50)

You don't need "Yes" and "No" when Excel provides built-in values TRUE and FALSE. You could write =IF(B1>100,TRUE,FALSE), but this is simpler:

A1: =B1>100 and later A10: =IF(A1,A2,50)

The formula =B1>100 evaluates to either TRUE or FALSE. A1 holds this value and you can test it later in another formula.

When the Solution Seems Wrong

When the solution on the worksheet seems wrong, before doing anything else, you should **read the solution log** in the Output tab, **click the error message** to display Help, and **run available reports** as described above. We emphasize this, because very often when we are contacted in technical support, the user has not taken these basic steps.

Although software bugs are always possible, consider carefully the possibility that the solution found by the Solver is **correct for the model you've defined**, and that your expectation is wrong. This may mean that what your model actually says is different from what you intended. In the majority of cases we see in technical support, the user has an error in a formula or in the expression of a constraint that leads to the unexpected solution.

Problems with Poorly Scaled Models

Many unexpected Solver Result messages are due to 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. 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** option on the Task Pane Engine tab. When this option is set to True, 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 can't 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.

If you're using Risk Solver Platform or Premium Solver Platform, and 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 after solving your model. If you're using Premium Solver Pro, you'll have to go through each of your formulas and play "what-if" manually to identify such problems.

The Integer Tolerance Option

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 standard Solver's Options dialog. In similar cases, Risk Solver Platform displays a message "Solver found an integer solution within tolerance," to avoid confusion.

When you solve a problem with integer constraints, the solution process in almost all Solver engines is governed by the **Integer Tolerance** option on the Task Pane Engine tab. When this option value is non-zero – say 0.05, as in the standard Excel Solver – the Solver engine 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.

To avoid this common problem, the *default* **Integer Tolerance** value in Risk Solver Platform V9.0 and beyond is **0** rather than 0.05. But this has an important consequence for solution time: The solution process for integer problems 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.

To avoid this extra time, you can either set the Integer Tolerance to a non-zero value – say 0.025 or 0.05 – or you can watch the Task Pane Output tab during the solution process, which displays the current Integer Gap – the value against which the Integer Tolerance is compared. Here's an example from the solution of the EXAMPLE5 model shown earlier, where the Integer Gap was 1.6% just before the Solver "proved optimality" and reported the solution:

Best Integer Objective	2400
Current Objective	2400
Nodes	6
Iterations	0
Relaxed Objective	-5200
Best Possible Objective	2361.69
Integer Gap	0.0162228

You can make a real-time decision that the current solution is "good enough," and press ESC or click the Pause/Stop button, then click the Stop button in the Show Trial Solution dialog, to stop the Solver with the current solution.

When Things Go Right: Getting Further Results

When you *do* receive a solution that makes sense to you, and you have a Solver Result message in **green** at the bottom of the Task Pane, there are several ways you can get further results. We'll cover three possibilities:

1. **Obtaining dual values** from a linear or smooth nonlinear optimization problem.
2. **Obtaining multiple solutions** from an integer programming problem, or a global optimization problem.
3. Performing **multiple parameterized optimizations**, and capturing all the solutions in reports and charts.

Dual Values

When you formulate and solve a linear programming problem, or a smooth nonlinear optimization problem, the solution process also yields numbers, called *dual values*, for the decision variables and constraints that are "pressed to the limit" at the optimal solution. A dual value can tell you, for example, how much you could pay to acquire more units of a scarce resource that is fully utilized in the solution; it is sometimes called a *shadow price* or *marginal value*.

- The **dual value for a decision 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 change in the objective function's value *per unit change* in the variable's value.
- 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 change in the objective function's value *per unit change* in the constraint's bound.

In nonlinear optimization problems, 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 as soon as you move away from the optimal solution. In linear programming problems, the dual values remain constant over a *range* of increases and decreases in the variables' objective coefficients and the constraints' right hand sides, respectively.

To obtain dual values in report form, simply select **Reports – Optimization – Sensitivity** from the Ribbon. An example Sensitivity Report appears below.

You can also obtain dual values via the Risk Solver Platform Object-Oriented API, with simple references such as myProb.VarDecision.DualValue(i) or myProb.FcnConstraint.DualValue(i) in your VBA code. This is described in greater depth in the chapter "Automating Optimization in VBA."

On the next page is a Sensitivity Report for EXAMPLE1, the Product Mix model which is the first example in the chapter "Examples: Conventional Optimization." At the optimal solution, we use all 800 Speaker Cones and 600 Electronics units, but not all of the other components. We don't produce any Speakers – cell F9 is driven to its lower bound of 0. What do the dual values tell us about the two binding constraints, and the one "nonbasic" decision variable?

The dual value of 12.5 for Speaker Cones tells us that we could increase Total Profits by $12.50 for every additional Speaker Cone we can acquire, up to 100 more. Similarly, the dual value of 25 for Electronics units tells us we could use up to 50 more units and increase Total Profits by $25.00 for each extra unit.

The dual value of -2.5 for F9 tells us that, if we were forced to produce some Speakers, we would reduce Total Profits by $2.50 for each Speaker we made (because we'd give up production of another product that is more profitable).

Microsoft Excel 14.0 Sensitivity Report
Worksheet: [StandardExamples.xls]EXAMPLE1
Report Created: 11/7/2011 3:15:32 PM
Engine: Standard LP/Quadratic

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	-3	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	13	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

Multiple Solutions

When you solve an integer programming problem or a global optimization problem, the final solution you see on the Excel worksheet is typically the *best* of *several* candidate solutions that were found during the solution process. You can obtain and examine these other solutions; this can be useful, especially if you have other criteria, not captured in the formulation of the optimization model, for preferring one solution over another. For each candidate solution, you can examine the final values of the decision variables and the objective.

To obtain multiple solutions in report form, select **Reports – Optimization – Solutions** from the Ribbon.

- For **integer programming** problems, the report shows each 'incumbent' or feasible integer solution found by the Branch & Bound method during the solution process.
- For **global optimization** problems, the report shows each locally optimal solution found by the Multistart method.
- For **non-smooth optimization** problems solved with the Evolutionary Solver, the report shows key members of the final population of solutions.

For all types of problems, if only one candidate solution was found, the **Solutions choice will *not* appear** in the Reports – Optimization gallery. For example, advanced Solvers for integer programming problems may find optimal solutions at the root node of the Branch & Bound tree, without considering any other incumbent solutions. For integer programming problems solved with the LP/Quadratic or Large-Scale LP/QP Solver, the Engine tab Integer group PreProcessing option must be set to None in order to create a Solutions Report.

On the next page is an example of a Solutions Report created when the GRG Nonlinear Solver with Multistart was used to solve a model called SPACE2, a global optimization problem with many locally optimal solutions. (Given more time, the Solver would find an even better solution than the ones shown.)

	A	B	C	D	E	F	G
1	**Microsoft Excel 12.0 Solutions Report**						
2	**Worksheet: [SPACE2.XLS]SPACE**						
3	**Report Created: 6/6/2009 5:03:49 PM**						
4	**Result: Solver stopped at user's request. All constraints are satisfied.**						
5	**Engine: Standard GRG Nonlinear**						
6	**Number of Solutions: 5**						
7							
8	Solutions:						
9		**Cell**	**Sol 1 (Obj = 34.7253)**	**Sol 2 (Obj = 64.1799)**	**Sol 3 (Obj = 83.9529)**	**Sol 4 (Obj = 128.433)**	**Sol 5 (Obj = 234.14)**
10		$Q$5	13.5749214	8.456336378	13.68280564	14.32402405	29.97326433
11		$R$5	-2.095903093	2.010721557	-5.08001014	4.639308354	5.341570539
12		$Q$6	1.811523778	0	2.062710716	14.24859304	29.97326433
13		$R$6	-5.257083287	-2.560558607	5.892599166	6.28	5.853042759
14		$Q$7	0	9.30459138	8.44261354	14.23285127	29.97326433
15		$R$7	2.996457642	4.512760989	-5.830352376	5.349088447	3.776442452
16		$Q$8	15.28086712	1.808298863	18.05082827	14.28245672	29.97326433
17		$R$8	5.845985996	4.872126044	1.681395998	3.137035536	0.505707235
18		$Q$9	4.057961706	10.88038115	2.254625979	14.27034649	29.97326433
19		$R$9	-1.956273311	-5.741591407	1.095416798	5.27517222	-0.113197958
20		$Q$10	0	14.32502798	13.59691407	14.28358621	0
21		$R$10	3.592993083	1.092410086	-0.274317073	5.384580377	1.907630166
22		$Q$11	0	2.038932336	0.245928784	14.27051412	29.97326433
23		$R$11	2.328517421	-1.155301348	-4.205806717	5.324449587	4.659735782
24		$Q$12	0	11.92315601	12.87823407	14.25784822	29.97326433
25		$R$12	4.516592658	4.74256651	-1.567517054	5.039754335	0.12752328
26		$Q$13	0	5.443173799	12.73820493	14.26263951	24.32734009
27		$R$13	3.725235279	0.233200929	2.651486361	4.92729056	-6.28

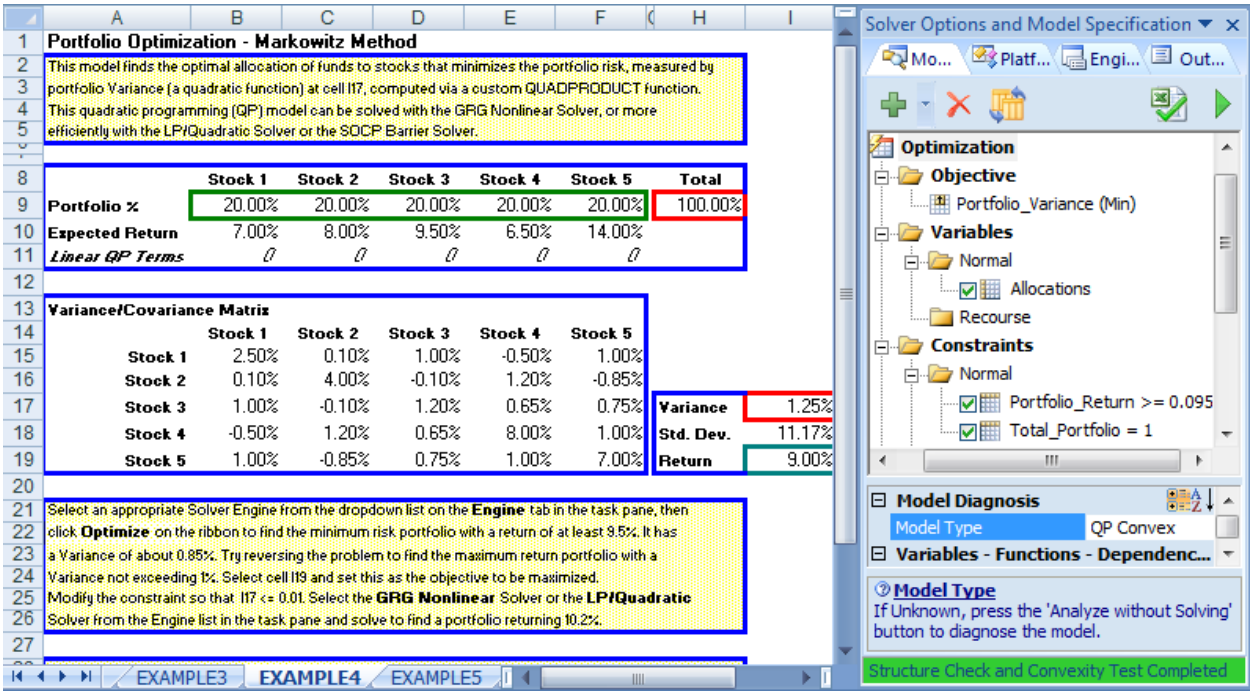
You can also obtain multiple solutions via the Risk Solver Platform Object-Oriented API, by accessing a property such as myProb.Solver.NumSolutions, setting the property myProb.Solver.SolutionIndex to choose a solution, then accessing the solution in the usual manner.

Multiple Parameterized Optimizations

Once you have a model where you're getting an optimal solution that makes sense for one set of inputs or parameters, it's often useful to vary one or more parameters across a range of values, and find the optimal solution for each individual parameter value. With Risk Solver Platform, you can easily define optimization parameters, run multiple optimizations and save the solutions to each optimization, and summarize the results in reports and charts.

The earlier chapter "Examples: Parameters and Sensitivity Analysis" gives an overview of the role of parameters for optimization, simulation, and sensitivity analysis, and the Frontline Solvers Reference Guide fully documents the features available for multiple parameterized optimization.

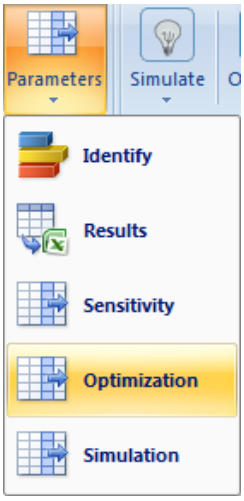
In this chapter, we'll illustrate what you can do with EXAMPLE4, the Markowitz portfolio optimization model in **StandardExamples.xls** described in the chapter "Examples: Conventional Optimization" under "Nonlinear Programming Examples." The model is pictured on the next page.



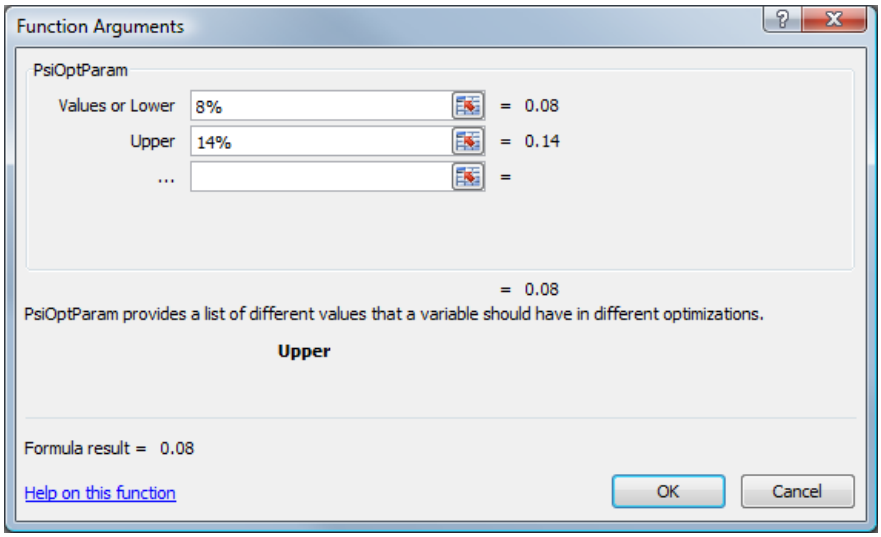
When we perform a single optimization with this model, we get an optimal allocation of funds to stocks that minimizes portfolio variance (a risk measure) while earning a portfolio rate of return of at least 9.5%, as shown in the Task Pane Model tab. This solution is a single point on the "efficient frontier;" we can find other points on this frontier by solving the problem for different levels of the required portfolio return.

This is easy to do in Risk Solver Platform: We'll define a single optimization parameter, use it in the portfolio return constraint right hand side, and then run multiple (say 10) optimizations where the parameter is automatically varied from (say) 8% to 14%. Let's do this step by step.

To define an optimization parameter, select an empty cell – say **I20**, and choose **Parameters – Optimization** from the Ribbon, as shown on the next page.

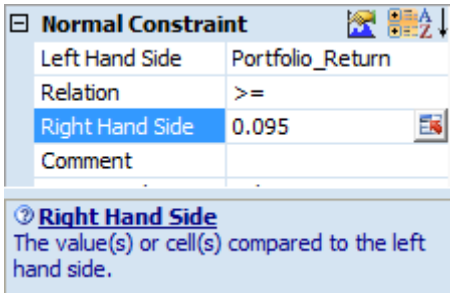


Risk Solver Platform displays a dialog where you can enter a lower and upper limit for the parameter value, or alternatively a list of values or a cell range:

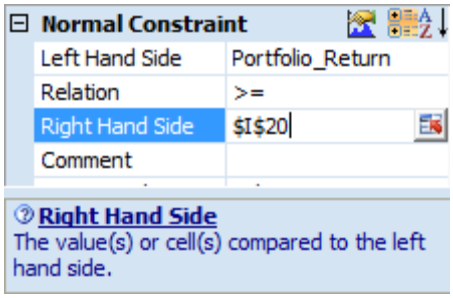


When you click OK, the formula **=PsiOptParam(0.08,0.14)** appears in cell I20.

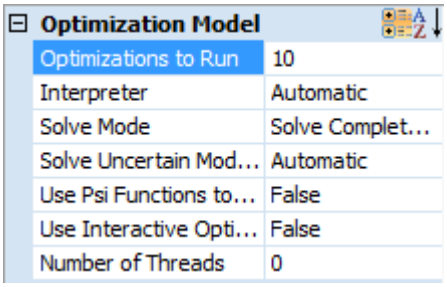
Next, we'll use this parameter in the portfolio return constraint. To do this, we select the constraint 'Portfolio Return >= 0.095' in the Task Pane Model tab, and edit its properties in the lower part of the Task Pane:



We click the cell selector icon to the right of the field containing 0.095, and then point and click to select cell I20. The properties of the constraint are updated to use cell I20 for the right hand side, as shown on the next page.

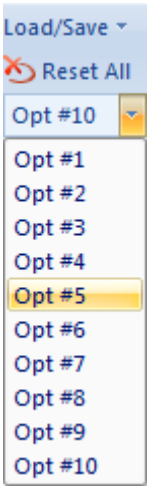


Just one more step is required: We must specify how many optimizations we want to perform. On the Task Pane Platform tab, we set the very first option **Optimizations to Run** to 10, as shown on the next page.

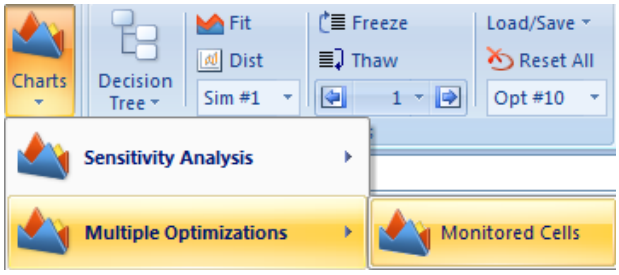


Now we simply click the **Optimize** button on the Ribbon, or the green right arrow on the Task Pane, to run all 10 optimizations. By default, Risk Solver Platform saves the optimal objective and decision variable values from all 10 optimizations; you can save other results, such as constraint values, by setting their Monitor property in the Task Pane to True.

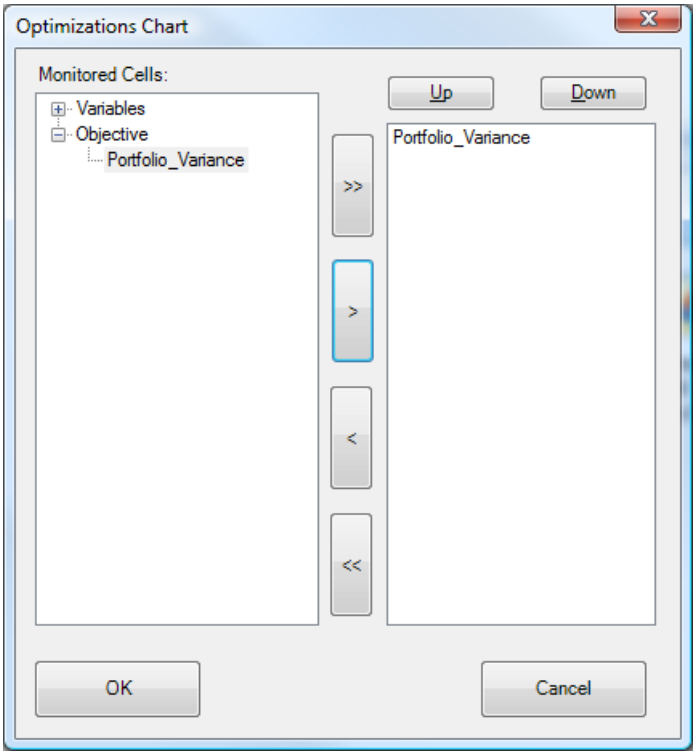
Just after solving, Risk Solver Platform displays the results of the last optimization on the worksheet. But you can display the results of any of the other 9 optimizations by selecting from the **Opt # dropdown list** on the Ribbon:



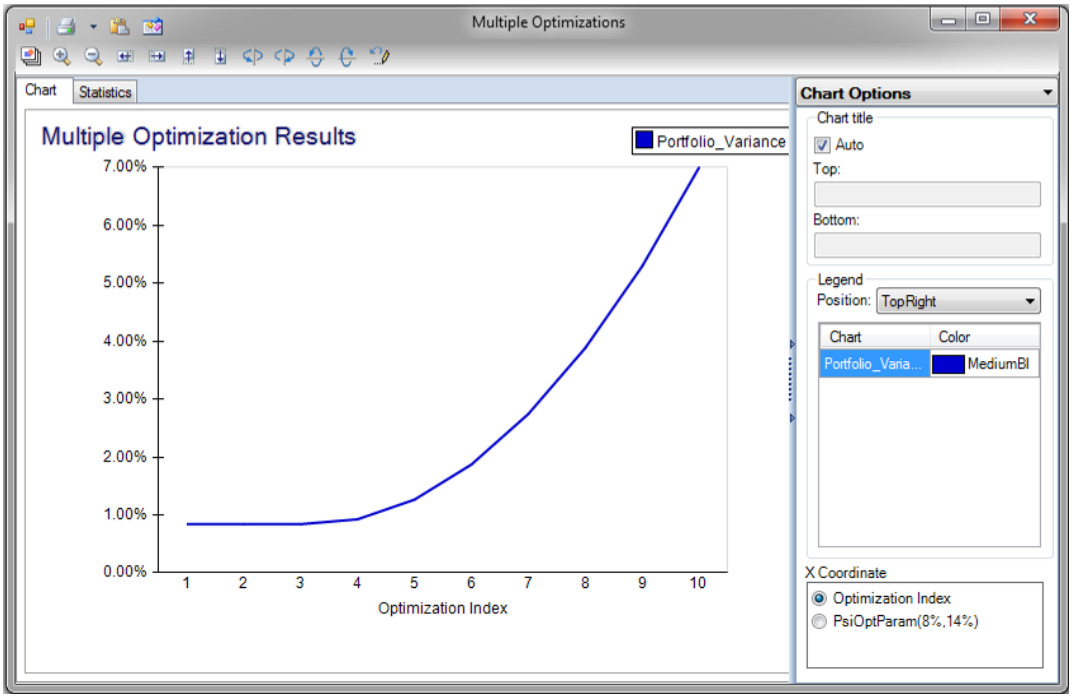
Risk Solver Platform has built-in facilities to create charts of multiple parameterized optimizations. Just select **Charts – Multiple Optimizations – Monitored Cells** from the Ribbon, as shown on the next page.



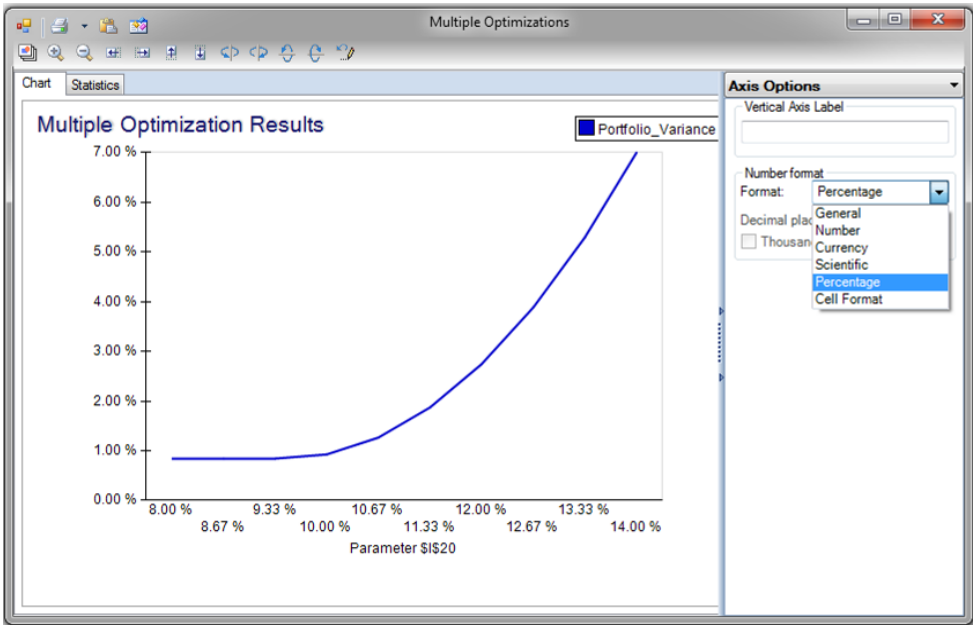
Select the optimization result you would like to chart: In this example, we want to plot the optimal objective (Portfolio Variance) across the 10 optimizations, where the Portfolio Return threshold is varied on each optimization.



When you click OK, Risk Solver Platform draws the chart, as shown below. That's a lot more useful results than a single optimal solution! Note, that you can see the specific values used for each optimization by, in the right pane of the chart, clicking on the radio button for PsiOptParam(8%,14%) in the X Coordinate section at the bottom.



Also, if you want to change how the X coordinate values are displayed simply change Chart Options to Axis Options in the drop down menu in the right pane and select the format you prefer.



Getting Results: Simulation

Introduction

This chapter explains how to obtain and interpret results from simulation in Risk Solver Platform and its subset product Risk Solver Pro. We'll discuss **what can go wrong**, and what to do about it, and also **how to get maximum insight** from the results of a simulation.

Simulation is 'simpler' than optimization in the sense that it requires only (i) sampling of input values, (ii) calculation of your Excel model with these values, and (iii) collection of the results; no 'search for a best solution' is involved. A simulation can be performed on almost any model, using any Excel functions.

But the way you build your model does affect the *speed* of simulation, or the time required to get results. And the structure of your model will become even more important in the next chapter, when we consider optimization and models that include uncertainty.

In the following sections, we'll focus on immediate actions you can take when you get an unexpected result – but if you read the chapter "Mastering Simulation and Risk Analysis Concepts," you'll learn more about the Monte Carlo simulation process, and better understand how to design your model to get the best results in the least time.

What Can Go Wrong, and What to Do About It

When you click the **Simulate** button on the Ribbon, or the **green arrow** on the Task Pane to run a simulation, you'll normally get one of these outcomes:

1. Results on the worksheet, and the message "Simulation finished successfully" in **green** at the bottom of the Task Pane. You can proceed to "When Things Go Right: Getting Further Results."
2. An error message that you understand and can correct, in **red** at the bottom of the Task Pane. You can take corrective action.
3. An error message that you *don't* understand, in **red** at the bottom of the Task Pane. You should **read the solution log** in the Output tab, **click the error message** and read Help about the message.
4. Results on the worksheet that you don't understand. Again before doing anything else, you should **read the solution log** in the Output tab, **click the error message** to display Help, and **read the section** below "When Simulation Results Seem Wrong."
5. Simulation runs for a very long time, and you don't get results or a message in the Task Pane until you press ESC or click Pause/Stop. You should **read the section** "When Simulation Takes a Long Time."

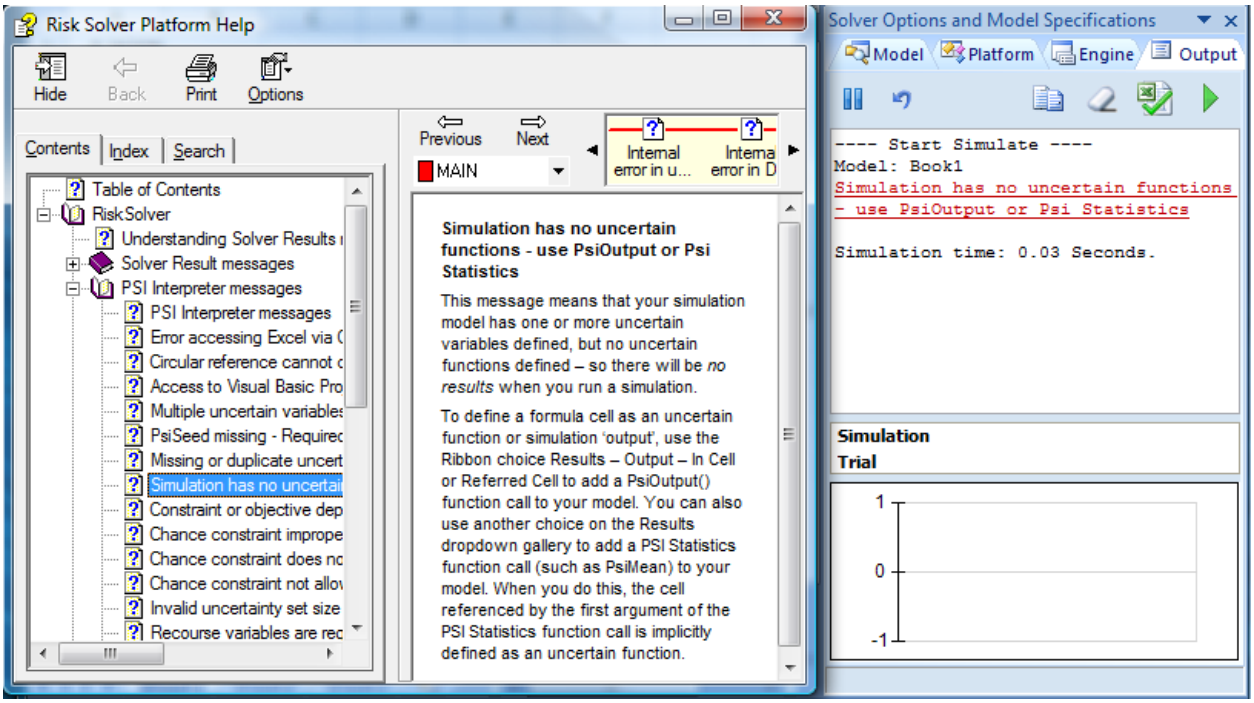
In rare cases, you might find Excel shutting down or "locking up" (so nothing happens when you press and hold the ESC key for several seconds). In this case

please contact Frontline Systems Technical Support at (775) 831-0300 x4 or support@solver.com. Some error messages ask you to contact Technical Support. If you can send us your model, this will be very helpful.

But experience shows that 99% of all technical support cases involve "pilot error" by the user, and that 90% of all such cases could be easily resolved by reading online Help or the User Guide. So we hope you'll keep reading, and that you'll take these steps before calling technical support!

Review Messages in the Output Tab

Your first step should be to **review the messages** in the **solution log** in the Task Pane Output tab. Below is an example of the most common simulation error message, and Help that appears when you click the message:



Click the Error Message for Help

Most simulation error messages are **underlined** – they are **hyperlinks** to online Help. If you aren't sure that you fully understand it, **click the link** to open Help to a detailed discussion of the message.

As the Help text explains in the example above, if your model doesn't define any simulation outputs, either via a PsiOutput() function that is added to the formula in an output cell or that refers to the output cell, or via a PSI Statistics function that refers to the output cell, there will be *no results* when you run a simulation. As corrective action, you simply need to define the formula cells for which you want results as simulation outputs.

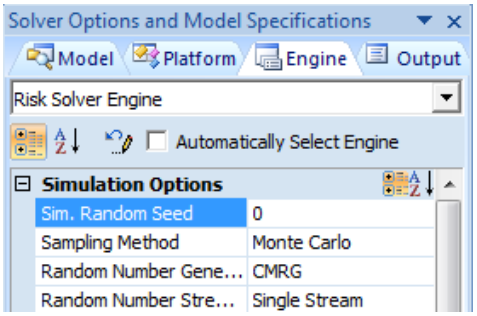
Role of the Random Number Seed

The **seed**, or initial value, of the **random number generator** used in the simulation process determines whether your results are *exactly reproducible*

when you re-run a simulation, or whether your results are *similar but not identical* because a different random sample was drawn.

By **default**, the seed for each simulation is set from the value of the system clock, which is **different** each time you run a simulation. If you set a seed value as explained below, the same seed value is used on each simulation, which means that the entire stream of random numbers drawn, and hence your simulation results, will be *exactly reproducible* on each simulation run.

To set the seed, use the Task Pane **Engine tab**. From the dropdown list at the top of the tab, select **Risk Solver Engine**, the software engine that actually performs a simulation. The options for Risk Solver Engine will be displayed:

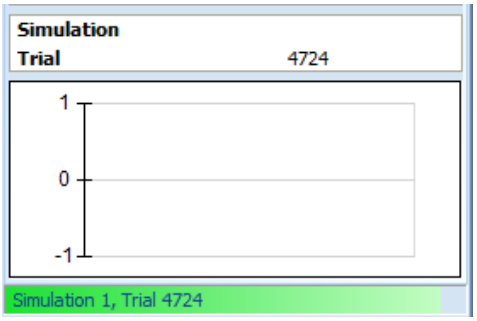


Enter a positive number for the option **Sim. Random Seed**. A value of 0 here means "use the value of the system clock as the seed on each simulation."

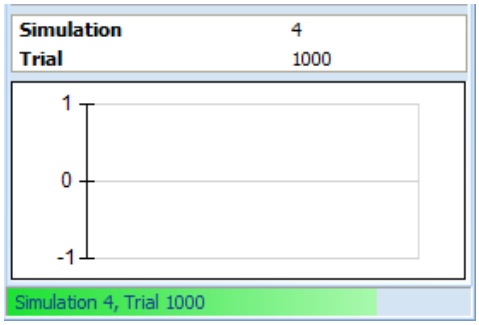
When Simulation Takes a Long Time

When a simulation takes a long time, or if you are running multiple parameterized simulations, the Task Pane Output tab shows you a progress indicator. If it is not already visible when you first start solving, the Output tab will **appear automatically** after a few seconds, as long as the Task Pane itself is visible.

If you have the Task Pane Platform tab Simulation group Interpreter option set to **Excel Interpreter**, the progress indicator will be updated during the Monte Carlo trials of a single simulation, as shown in the example below.

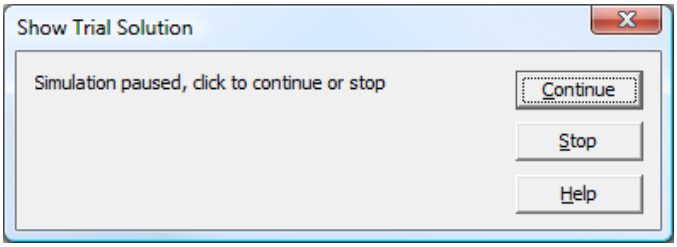


If you have the Interpreter option set to **Psi Interpreter**, the progress indicator usually ***won't*** be updated during the trials of a single simulation, because all of these trials are being executed in parallel. If you are running a multiple parameterized simulation, the progress indicator will be updated for each new simulation, as shown in the example below.



Interrupting the Simulation Process

You can interrupt the simulation process at any time, by **pressing the ESC key**, or by **clicking the button** in the Task Pane Output tab. If your model is very large, you might have to hold down the ESC key for a second or two. Since the input focus may not be on the Task Pane when you click with the mouse, you might have to click the button twice. A dialog like the one below will appear:



Click **Stop** to stop the simulation, or **Continue** if you want to continue.

Reasons Why Simulation Takes a Long Time

Risk Solver Platform is by far the fastest Monte Carlo simulation software for Excel on the market, and it takes full advantage of multi-core processors for even faster simulations. But it's still possible to create a model where a simulation takes a long time. There are several reasons for this:

1. If you have the Platform tab Simulation group Interpreter option set to **Excel Interpreter**, Excel is used to recalculate the worksheet on each Monte Carlo trial. This is usually an order of magnitude slower than the PSI Interpreter, though it requires significantly less memory.
2. If you have Platform tab Simulation group Interpreter option set to **Psi Interpreter**, and the simulation is slow, you may be using certain Excel functions in ways that slow down the PSI Interpreter. See below for Do's and Don'ts to get the most from the PSI Interpreter.
3. Your model may simply be **very large**, with many complex formulas, references to other worksheets or workbooks, etc. that take a long time to evaluate. If your workbook includes sizable elements that aren't **essential to the simulation model** and its results, these extra elements may be slowing down the simulation process.

Time Versus Memory Tradeoffs

The PSI Interpreter is designed to take maximum advantage of available memory, and trade off memory against time. When Excel interprets =A1+A2*(A3-4), it uses memory for only four numbers – the values of A1, A2, A3 and the constant 4. But when PSI interprets this formula, it uses memory for

3,001 numbers – 1,000 each for A1, A2 and A3. If your PC has plenty of memory, PSI will help you take advantage of it. But if you are tight on memory, you may find that simulations slow down, because Windows will use 'virtual memory' and swap your data between main memory and your hard disk. The best solution is to *get more memory* – DRAM chips for PCs and notebooks are an amazing value today!

Model Design for Maximum Simulation Speed

Paying attention to your Excel worksheet layout and design, and avoiding certain practices, will help you get the most from the PSI Interpreter. Below is a brief list of Do's and Don'ts that will help you realize the best possible speed:

- **Do** build worksheets starting from the "upper left corner." **Don't** place cells or formulas at extreme row and column addresses all over the worksheet.
- **Do** build the model on a small number of worksheets. **Don't** include references on these worksheets to other worksheets that aren't required for the simulation model.
- **Do** use numeric and logical formulas and functions. **Don't** create string results in the middle of numeric calculations (see example below).
- **Do** use built-in Excel functions (including Analysis ToolPak functions) freely. **Don't** use third-party or user-written functions unless truly needed. Such functions must not have "side-effects" other than the value returned.
- **Do** use operators like + - * / ^ and functions like SUM, AVERAGE, MAX, MIN, AND, OR, NOT, IF, CHOOSE. **Don't** use INDIRECT, OFFSET, or TEXT, SEARCH, REPLACE, FIXED, DOLLAR, or ROMAN.
- If possible, **Don't** use array formulas, LOOKUP functions with large ranges as arguments, or long "chains" of formulas where each formula depends on an earlier formula (and hence on *all* earlier formulas). Each of these things can slow down analysis of your model considerably (by 10x in some cases).

A common practice that slows down the PSI Interpreter, and is easy to avoid, is illustrated by the following:

A1: =IF(B1>100,"Yes","No") and later A10: =IF(A1="Yes",A2,50)

You don't need "Yes" and "No" when Excel provides built-in values TRUE and FALSE. You could write =IF(B1>100,TRUE,FALSE), but this is simpler:

A1: =B1>100 and later A10: =IF(A1,A2,50)

The formula =B1>100 evaluates to either TRUE or FALSE. A1 holds this value and you can test it later in another formula.

When Simulation Results Seem Wrong

When the simulation results seem wrong, before doing anything else, you should **read any messages** in the Output tab, and **click the message** to display Help as described above. We emphasize this, because very often when we are contacted in technical support, the user has not taken these basic steps. As *next* steps:

1. Make sure you understand how your formulas behave in an ordinary **Excel recalculation**. If you've "raced ahead" with a simulation model before doing this, you can use the **Freeze** button on the Ribbon to turn your simulation model back into a "what-if" model (by moving all PSI function

calls into cell comments), and later use the **Thaw** button to restore the PSI functions that make up your simulation model.

2. Bear in mind that an uncertain function cell normally displays the calculated value for the **last** Monte Carlo trial of the **last** simulation run. It's usually more meaningful to look at the results returned by PSI Statistics functions, such as PsiMean() or PsiPercentile(), across all the trials of the simulation.

3. It can be very helpful to **cycle through the Monte Carlo trials** of a simulation, and examine the values of uncertain function cells, as well as intermediate formula cells on which they depend. Just use the **left and right arrows** on the Ribbon to change the **trial index**. You may find that your model calculates a result you did not expect for some values of the uncertain variables.

4. If you are using multiple simulations, check that the **Sim # index** on the Ribbon, or the **Sim # dropdown** in the Uncertain Function dialog title bar, are selecting the simulation you want, and that parameters (PsiSimParam() functions) are returning the values you expect for that simulation.

5. If you haven't already done so, double-click uncertain function cells and examine the **Frequency tab** chart and **Percentiles tab** numbers to see how the values of this function were distributed across Monte Carlo trials in the simulation. This will help explain the values of PSI Statistics functions.

Although software bugs are always possible, consider carefully the possibility that the simulation results are **correct for the model you've defined**, and that your expectation is wrong. We sometimes find in technical support that what your model actually says is different from what you intended.

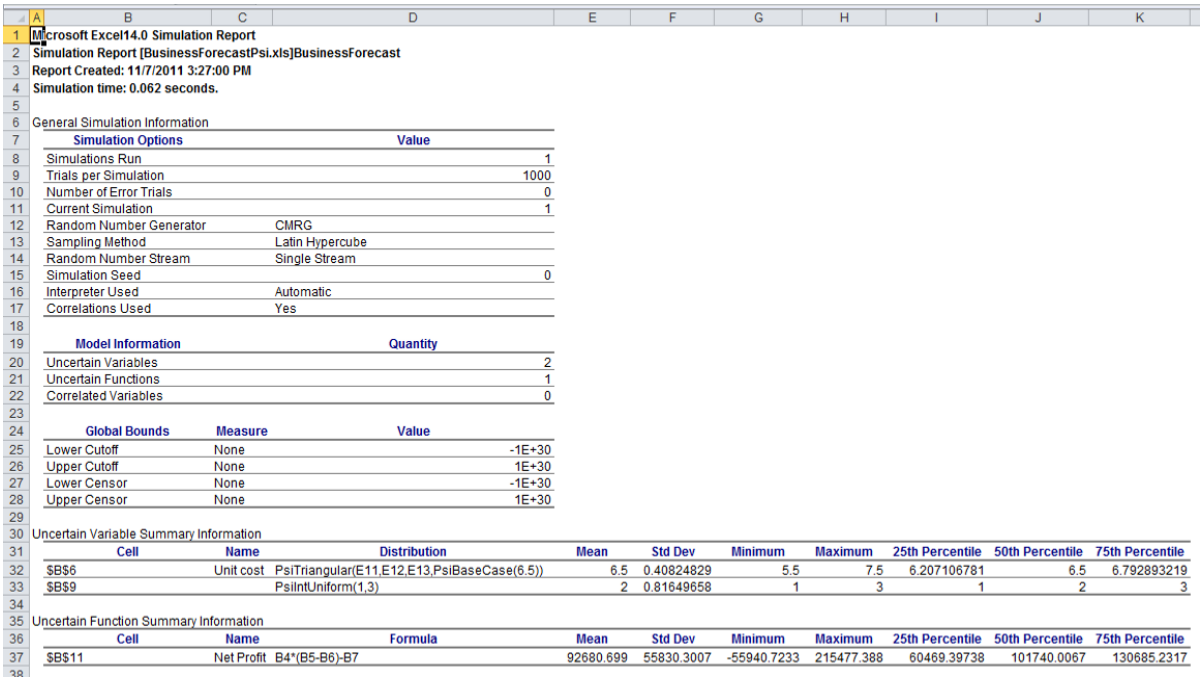
When Things Go Right: Getting Further Results

When you have simulation results on the worksheet that make sense to you, and you have the message "Simulation finished successfully" in **green** at the bottom of the Task Pane, there are several ways you can get further results. We'll cover five possibilities:

1. **Documenting your results** in a Simulation Report.
2. **Using all the features** of the Uncertain Function dialog.
3. **Fitting an analytic distribution** to uncertain function results.
4. **Charting multiple uncertain functions**, typically over time.
5. Performing **multiple parameterized simulations**, and capturing the results on the worksheet or in charts.

Using the Simulation Report

One quick step you can take, when you have simulation results that you want to recall later, is to select **Reports – Simulation – Simulation** to produce a report worksheet, inserted into your workbook, like the one for the example model **BusinessForecastPsi.xls**, shown on the next page.



Microsoft Excel14.0 Simulation Report
Simulation Report [BusinessForecastPsi.xls]BusinessForecast
Report Created: 11/7/2011 3:27:00 PM
Simulation time: 0.062 seconds.

General Simulation Information

Simulation Options	Value
Simulations Run	1
Trials per Simulation	1000
Number of Error Trials	0
Current Simulation	1
Random Number Generator	CMRG
Sampling Method	Latin Hypercube
Random Number Stream	Single Stream
Simulation Seed	0
Interpreter Used	Automatic
Correlations Used	Yes

Model Information	Quantity
Uncertain Variables	2
Uncertain Functions	1
Correlated Variables	0

Global Bounds	Measure	Value
Lower Cutoff	None	-1E+30
Upper Cutoff	None	1E+30
Lower Censor	None	-1E+30
Upper Censor	None	1E+30

Uncertain Variable Summary Information

Cell	Name	Distribution	Mean	Std Dev	Minimum	Maximum	25th Percentile	50th Percentile	75th Percentile
$B$6	Unit cost	PsiTriangular(E11,E12,E13,PsiBaseCase(6.5))	6.5	0.40824829	5.5	7.5	6.207106781	6.5	6.792893219
$B$9		PsiIntUniform(1,3)	2	0.81649658	1	3	1	2	3

Uncertain Function Summary Information

Cell	Name	Formula	Mean	Std Dev	Minimum	Maximum	25th Percentile	50th Percentile	75th Percentile
$B$11	Net Profit	B4*(B5-B6)-B7	92680.699	55830.3007	-55940.7233	215477.388	60469.39738	101740.0067	130685.2317

The Simulation Report documents the option settings used to perform the simulation, and provides summary information about the uncertain variables and uncertain functions in your simulation model. You are much more likely to use simulation charts, or snapshots of your own worksheet layout when you are presenting reports to others, but this report can be quite useful when you want to refer later to the assumptions behind a simulation analysis.

Using the Uncertain Function Dialog

The Uncertain Function dialog has many features, described in the Frontline Solvers Reference Guide, that you can use to get enhanced results from your simulation. For an introduction to these features, read "A First Simulation Example" in the chapter "Examples: Simulation and Risk Analysis," especially the sections starting with "Viewing the Full Range of Profit Outcomes." Here, we'll just summarize the results available at your fingertips in the Uncertain Function dialog:

Dialog Tabs

- The **Frequency**, **Cumulative Frequency**, and **Reverse Cumulative Frequency** tabs provide three different views of the full range of outcomes for an uncertain function. You can click to display "crosshairs" with numerical values on these charts, or add Lower and Upper bounds to see the estimated probability of a profit or loss, for example.
- The Tornado chart on the **Sensitivity** tab quickly shows you which uncertain variables have the **greatest impact** on this uncertain function, across the full range of Monte Carlo trials. The **Scatter Plots** tab often reveals further insights about the behavior of this uncertain function versus each uncertain variable, or versus other uncertain functions.

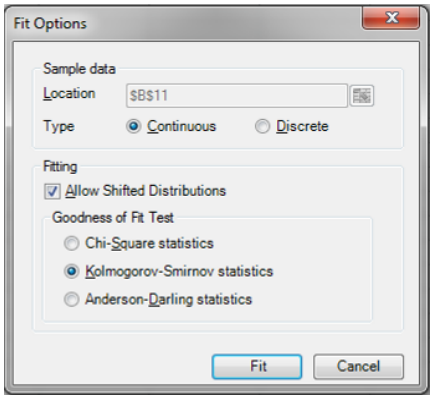
Panels and Toolbars

- Using the **right panel** of the dialog, you can access a drop down menu including **Statistics**, **Percentiles**, **Chart Type**, **Chart Options**, **Axis Options**, and **Markers**. From the **Options dialog** (accessible from the Options button on the Ribbon), on the **Charts** and **Markers** tabs, you can set default chart and marker settings for all of your charts.
- Using the **title toolbar icons**, you can save your settings, **print** the charts or numbers from any of the dialog tabs, or **copy** the charts or numbers to the Windows Clipboard, where they can be pasted into other applications. You can also **fit a distribution** to the simulation results, as described below.
- Click the rightmost icon on the title toolbar to display the **3-D toolbar** that lets you – with a single click – shift between 2-D and 3-D, **shrink** or **magnify** the chart, or **move** or **rotate** the chart to a different perspective.

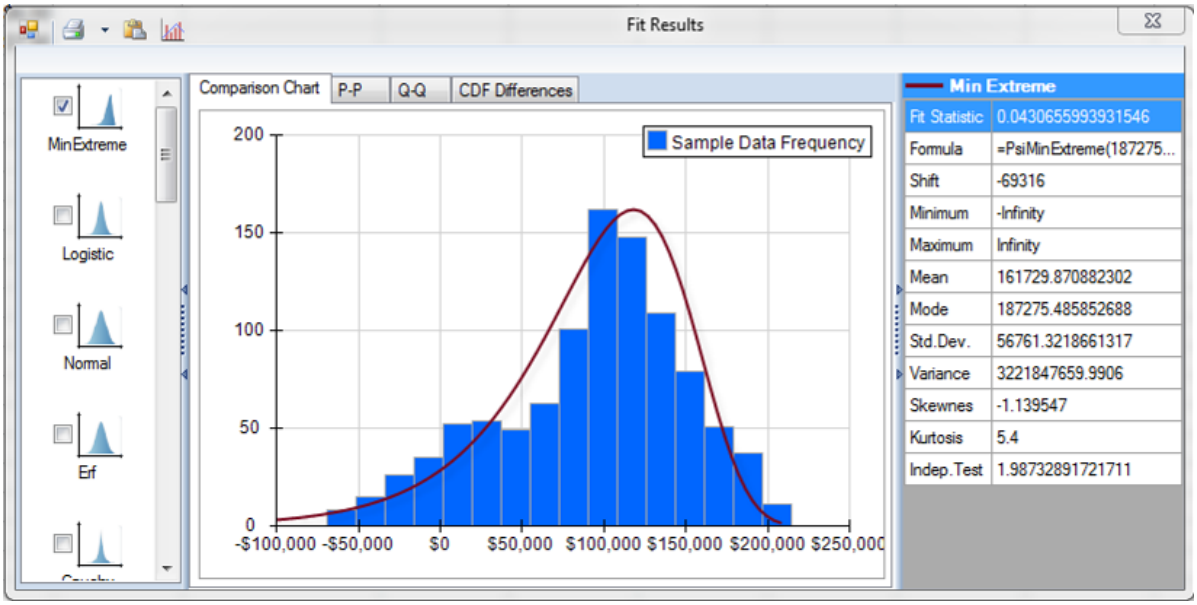
Fitting a Distribution to Simulation Results

One way to analyze your simulation results is to see whether the distribution of outcomes is similar to a well-known analytic distribution. Risk Solver Platform and Risk solver Pro support a wide range of analytic distributions, and can fit a distribution and its parameters to sample data. To fit your sample simulation results, click the  icon on the Uncertain Function dialog title toolbar.

When we click the title toolbar icon to fit a distribution to simulation results for Net Profit (cell B11) in the example **BusinessForecastPsi.xls**, a **Fit Options** dialog appears like the one shown below.

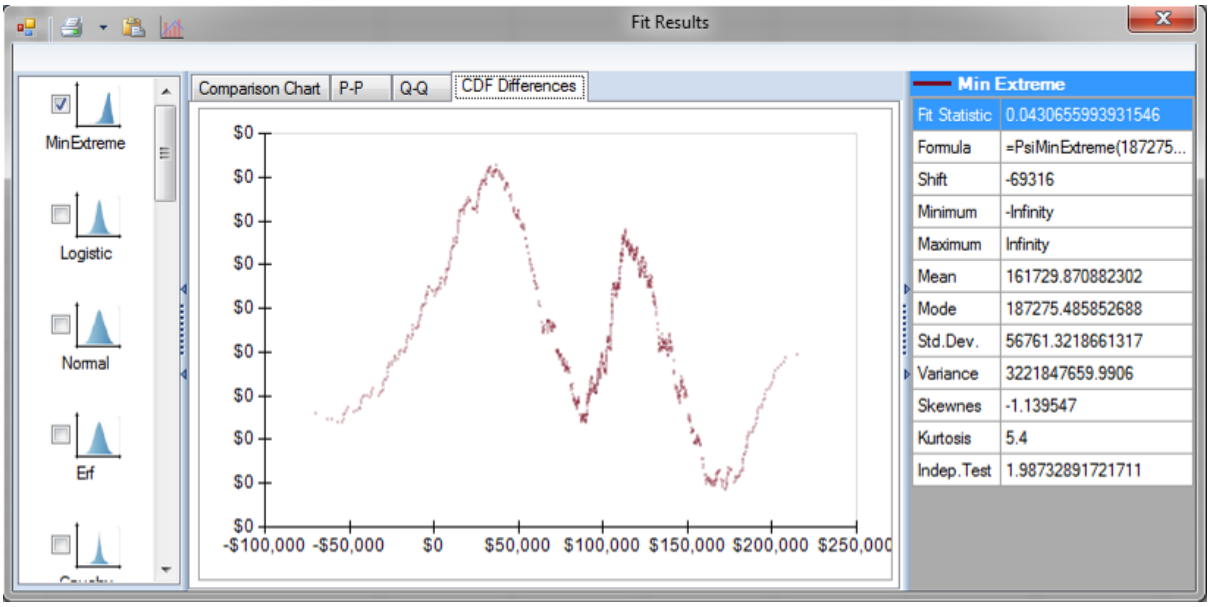


If we accept the default settings and click the **Fit** button in this dialog, Risk Solver Platform displays a dialog like the one on the next page.



In a few seconds, Risk Solver Platform fits a wide range of distributions to the sample data, ranks them in order of the selected goodness-of-fit criterion (Kolmogorov-Smirnoff by default), and displays the best-fitting distribution. You can superimpose other distributions on the frequency chart by clicking on the check box to the left of each distribution.

Other tabs of this dialog give you a visual picture of the fit, in the form of P-P and Q-Q charts, and a chart of CDF Differences, shown below.



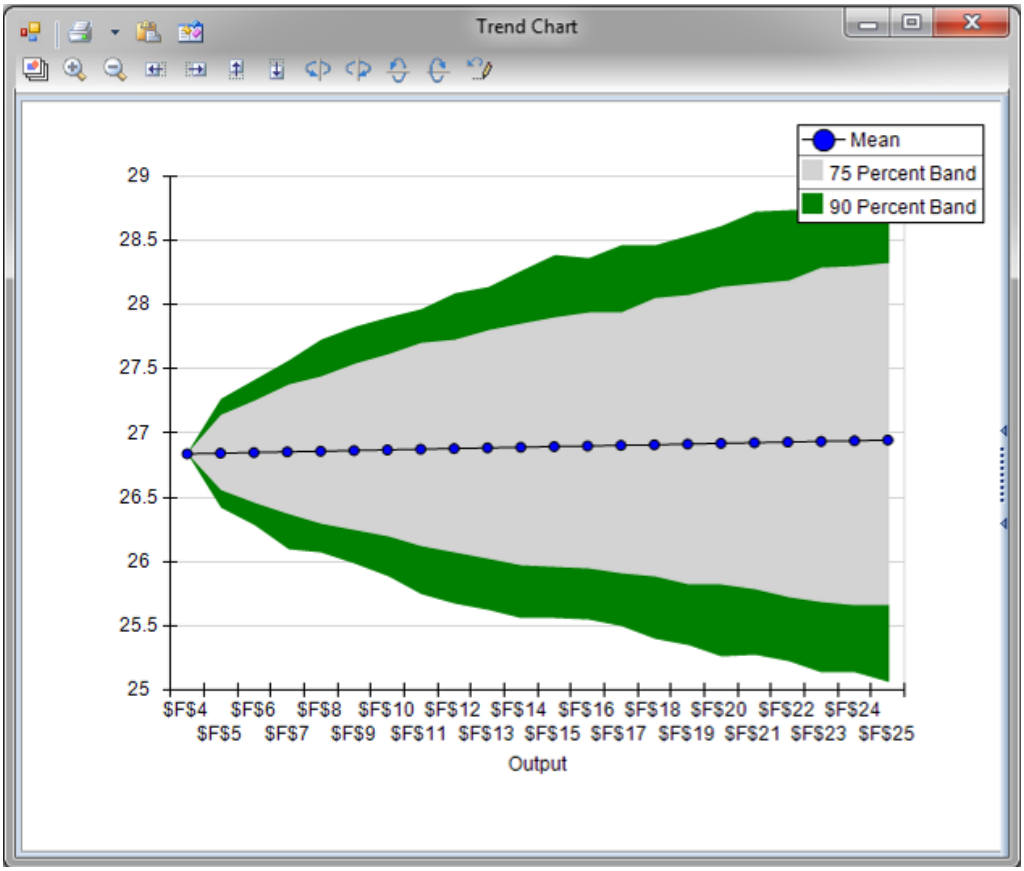
You can also fit an analytic distribution to data that you obtain from another source, such as historical observations of the process you want to model. If you have this sample data on your Excel worksheet, you can simply select the cell

range containing the data, and click the **Fit** icon on the Ribbon. Using this tool, you can easily create an **uncertain variable** whose distribution models this data.

Charting Multiple Uncertain Functions

In some simulation models, you will have a series of uncertain functions that represent the evolution of some process over time. The individual cells might represent sales or inventory levels, interest rates or exchange rates, or some other uncertain factor. You want to understand how this uncertain function changes over time, taking into account its *uncertainty*.

Risk Solver Platform has a general facility for creating charts of multiple simulation results, including Overlay, Trend, and Box-Whisker charts. Below, we've used this facility to visualize the evolution of a stock price over time, using the example model GBMSimpleModel.xls, which simulates stock prices using a Geometric Brownian Motion model. When we perform a simulation, select **Charts – Multiple Simulation Results – Trend**, and select the first 24 uncertain functions (out of 292 total) for inclusion in the chart, we get a Trend Chart like the one below:



Here we can see a slow upward drift of the mean value of the stock price, and also its volatility around the mean, depicted here with the 25th and 75th, and the 10th and 90th percentiles.

Multiple Parameterized Simulations

Once you have results from a single simulation, you normally want to explore "what if" scenarios, where you change some parameter that is under your control, and run simulations to observe the effects of the uncertain variables that are *not* under your control.

Using Interactive Simulation

Risk Solver Platform offers a powerful facility to do this through Interactive Simulation: You can simply click the Simulate button on the Ribbon to enter Interactive mode, then change a number on your worksheet and instantly see new simulation results. This is illustrated in "A First Simulation Example" in the chapter "Examples: Simulation and Risk Analysis." Risk Solver Platform is fast enough to make this sort of "what-if" analysis practical.

Using Multiple Simulations

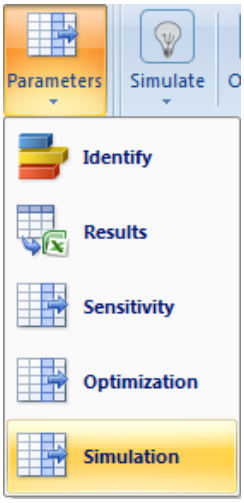
But after exploring the model and its results through Interactive Simulation, you'll likely want to automate the "what-if" process, and more systematically evaluate the effects of changing some parameter or parameters under your control on the simulation results. Risk Solver Platform makes this easy, with multiple parameterized simulations. This is illustrated in "An Airline Revenue Management Model" in the chapter "Examples: Simulation and Risk Analysis."

Setting Up Multiple Parameterized Simulations

Two simple steps are required to set up a multiple parameterized simulation:

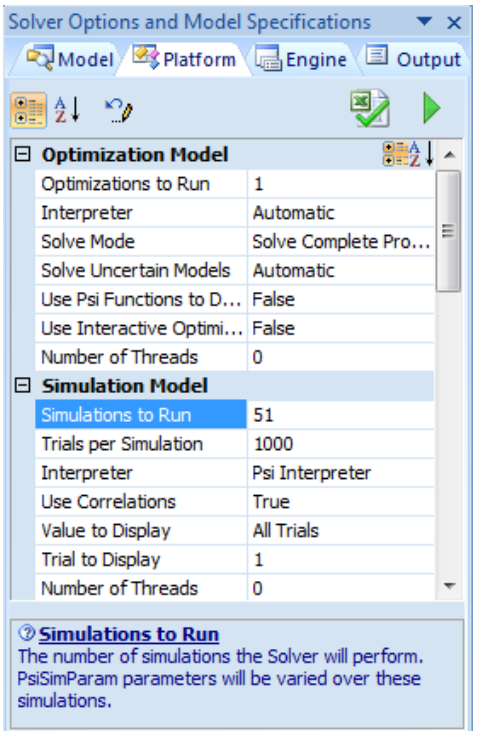
1. Define one or more **simulation parameters**, and use them in your model.
2. Set the **number of simulations** option to a value greater than 1, and run the simulation.

To define a simulation parameter, simply select **Parameters – Simulation** from the Ribbon:



Risk Solver Platform displays a dialog box, where you can enter a lower limit and upper limit for the parameter value, or a specific list of values that you would like the parameter to take on successive simulations. If you specify a lower and upper limit, then when you set the **number of simulations**, the parameter will take on equally-spaced values from the lower to the upper limit.

To set the number of simulations, simply edit the option in the Task Pane Platform tab Simulation group option **Simulations to Run**:



When you next click the **green arrow** in the Task Pane, or choose **Simulate – Run Once** from the Ribbon, all 51 (in this example) simulations will be run, automatically varying each simulation parameter over its range or list of values, and all the results will be collected and stored. (If you click the Simulate icon to activate *Interactive Simulation*, all 51 simulations will be run each time you make a change on the worksheet.)

Viewing Results of Multiple Simulations

After running multiple parameterized simulations, you can view the results on the worksheet, or in the Uncertain Function dialog, by selecting the simulation of interest from the **Sim #** dropdown. You can create reports and charts of results that span all or some of the simulations. We highly recommend that you read "An Airline Revenue Management Model" in the chapter "Examples: Simulation and Risk Analysis," which illustrates how this is done – creating a Trend Chart and a Box-Whisker Chart across all 51 simulations in this model.

Once you become familiar with multiple parameterized simulations, you'll likely want to use them in nearly every simulation analysis you do. They're a natural way to get further results from the effort you've put into a simulation model, and communicate those results to colleagues or clients.

Getting Results: Stochastic Optimization

Introduction

This chapter explains how to obtain and interpret results from stochastic optimization (optimization of models with uncertainty) in Risk Solver Platform. We'll discuss **what can go wrong**, and what to do about it, and also **how to get more** than a single optimal solution from your model.

Risk Solver Platform has powerful algorithms for automatically transforming and solving optimization models with uncertainty. But since a stochastic optimization model includes **decision variables**, **uncertain variables**, and **constraints** and an **objective** that may depend on both, there are "more things that can go wrong." Indeed, everything you've read in the previous two chapters, "Getting Results: Conventional Optimization" and "Getting Results: Simulation and Risk Analysis" will apply to stochastic optimization – plus there are a variety of special result messages and error messages that apply *only* to stochastic optimization models.

The good news is that everything you've learned so far about conventional optimization and Monte Carlo simulation *does* apply to stochastic optimization in Risk Solver Platform. Unlike most other software packages, Risk Solver Platform has one consistent user interface, set of terms, and model elements for all types of simulation and optimization models.

In the following sections, we'll focus on immediate actions you can take when you get an unexpected result – but if you read the chapter "Mastering Stochastic Optimization Concepts," you'll learn far more about stochastic optimization models and solution methods, and better understand *why* the unexpected result appeared, and how to design your model to get the solutions you want.

What Can Go Wrong, and What to Do About It

When you click the **Optimize** button on the Ribbon, or the **green arrow** on the Task Pane to solve, you'll normally get one of these outcomes:

1. A solution that makes sense to you. This is normally accompanied by a Solver Result message in **green** at the bottom of the Task Pane. You can proceed to "When Things Go Right: Getting Further Results."
2. A Solver Result error message that you understand and can correct, in **red** at the bottom of the Task Pane. You can take corrective action.
3. A Solver Result error message that you *don't* understand, in **red** at the bottom of the Task Pane. You should **read the solution log** in the Output tab, **click the error message** and read Help about the message.

4. A *solution* that you *don't* understand, or that seems wrong. Again before doing anything else, you should **read the solution log** in the Output tab, **click the error message** to display Help, **run available reports** as described below, and **read the section** below "When the Solution Seems Wrong."

5. Solving runs for a very long time, and you don't get a solution or a Solver Result message until you press ESC or click Pause/Stop. You should **read the section** below "When Solving Takes a Long Time."

In rare cases, you might find Excel shutting down or "locking up" (so nothing happens when you press and hold the ESC key for several seconds). In this case please contact Frontline Systems Technical Support at (775) 831-0300 x4 or support@solver.com. Some Solver Result error messages ask you to contact Technical Support. If you can send us your model, this will be very helpful.

But experience shows that 99% of all technical support cases involve "pilot error" by the user, and that 90% of all such cases could be easily resolved by reading online Help or the User Guide. So we hope you'll keep reading, and that you'll take these steps before calling technical support!

Review Messages in the Output Tab

Your first step should be to **review the messages** in the **solution log** in the Task Pane Output tab. This is **especially important** when you're solving a stochastic optimization model, to ensure that you understand the solution method that Risk Solver Platform used, and how your model was transformed and solved.

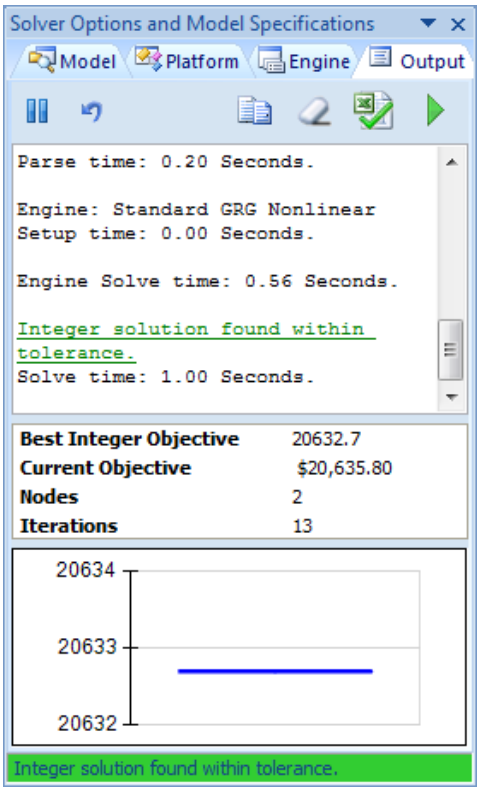
A Model Solved with Simulation Optimization

On the next page is an example of the Output tab at the solution of the example model **YieldManagementModel3.xls** (described more fully in the chapter "Examples: Simulation and Risk Analysis").

If you click the Copy  icon, the contents of the solution log will be copied to the Windows Clipboard, where you can paste it into Microsoft Word, NotePad, or an **email message to Frontline Systems Technical Support**. Also on the next page is the complete solution log from the Gas Company Chance model (before clicking the button to automatically improve the solution).

Note what happened during the solution process: Risk Solver Platform found that a Stochastic Transformation could not be applied to this model, because it has a *decision-dependent uncertainty* (the number of no-shows at cell C7 depends on the number of tickets sold, our decision variable, at C11). See the chapter "Mastering Stochastic Optimization Concepts" for more information on this important property of your model.

Risk Solver Platform diagnosed the model as "SIM NonCvx," and solved it using ***simulation optimization*** – the most general, but least scalable stochastic optimization method in Risk Solver Platform. It used the GRG Nonlinear Solver Engine (selected by the user) for optimization, and on every trial solution explored by the GRG Solver, it automatically performed a simulation.

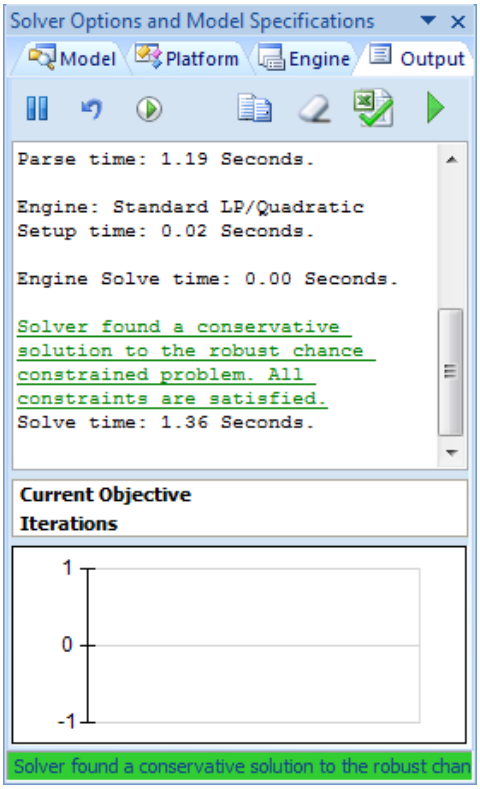



```
---- Start Solve ----
Using: Full Reparse.
Parsing started...
Uncertain input cells detected.
Using: Full Reparse.
Parsing started...
Diagnosis started...
Using: Full Reparse.
Parsing started...
Diagnosis started...
Convexity testing started...
Model transformation did not succeed.
Reverting to Simulation/Optimization.
Using: Full Reparse.
Parsing started...
Diagnosis started...
Model diagnosed as "SIM NonCvx".
User engine selection: Standard GRG Nonlinear
Model: [YieldManagementModel3.xls]Sheet1
Using: Psi Interpreter
Parse time: 0.20 Seconds.

Engine: Standard GRG Nonlinear
Setup time: 0.00 Seconds.

Engine Solve time: 0.56 Seconds.

Integer solution found within tolerance.
Solve time: 1.00 Seconds.
```

Let's contrast what happened above with another example, where there are no decision-dependent uncertainties, and the underlying optimization model is a linear programming problem, but there is a *chance constraint*.

A Model Solved with Robust Optimization

Below is an example of the Output tab at the solution of the Gas Company Chance model in **StochasticExamples.xls** (described more fully in the chapter "Examples: Stochastic Optimization").



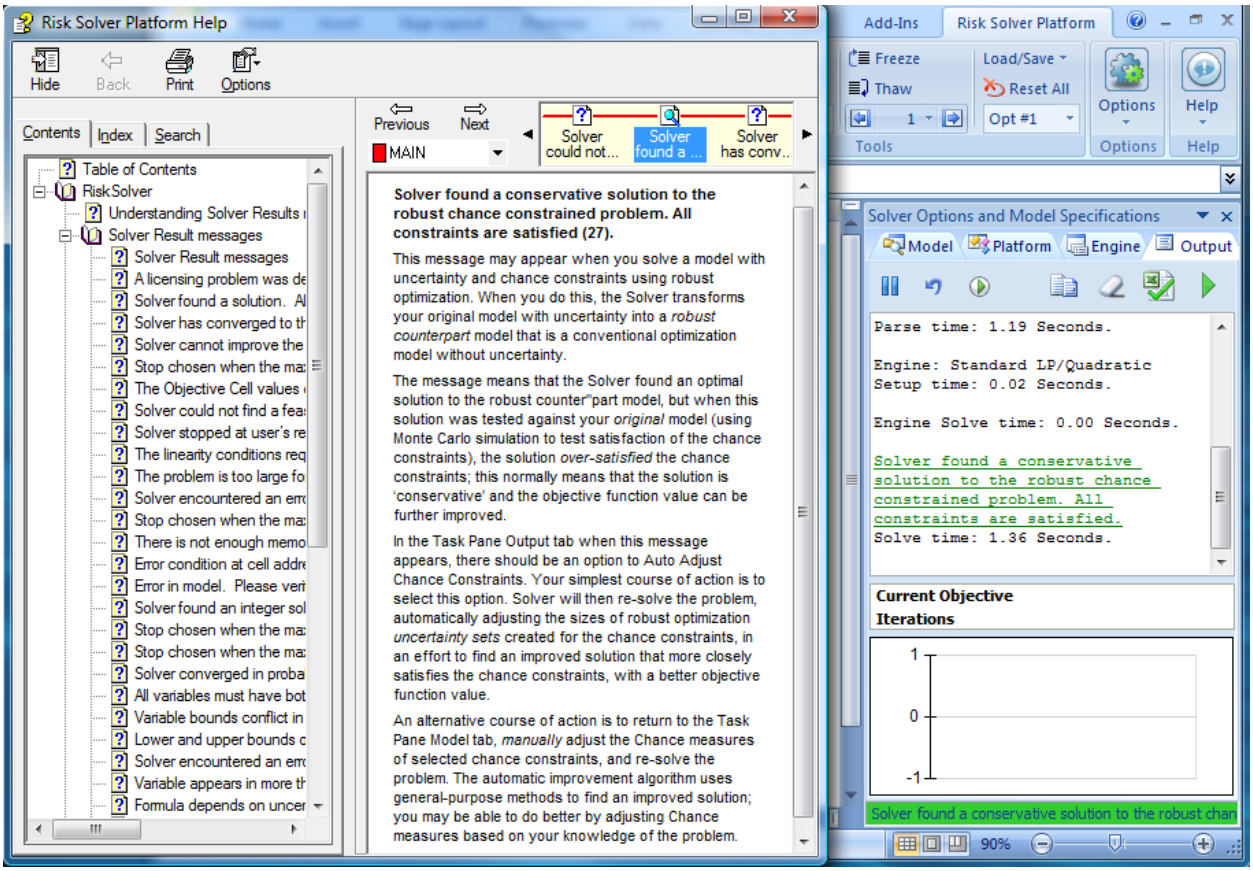
Again, if you click the Copy icon, the contents of the solution log will be copied to the Windows Clipboard, where you can paste it into Microsoft Word, NotePad, or an **email message to Frontline Systems Technical Support**. Below is the solution log from the Gas Company Chance model (before clicking the button to automatically improve the solution).

```
---- Start Solve ----
Using: Full Reparse.
Parsing started...
Uncertain input cells detected.
User requested Robust Counterpart.
Diagnosis started...
Convexity testing started...
Stochastic Transformation succeeded using Robust Counterpart
with D Norm.
Transformed model is "LP Convex".
Model: [StochasticExamples.xls]Gas Company Chance
Using: Psi Interpreter
Parse time: 1.19 Seconds.
Engine: Standard LP/Quadratic
Setup time: 0.02 Seconds.
Engine Solve time: 0.00 Seconds.

Solver found a conservative solution to the robust chance
constrained problem. All constraints are satisfied.
Solve time: 1.36 Seconds.
```

Click the Solver Result Message for Help

The Solver Result message is always **underlined** – it is a **hyperlink** to Help. If you aren't sure that you fully understand it, **click the link** to open online Help to a detailed discussion of the message.

Below is an example of Help that appears when you solve the Gas Company Chance model , and click on the Solver Result error message in green, "Solver found a conservative solution to the robust chance constrained problem. All constraints are satisfied."



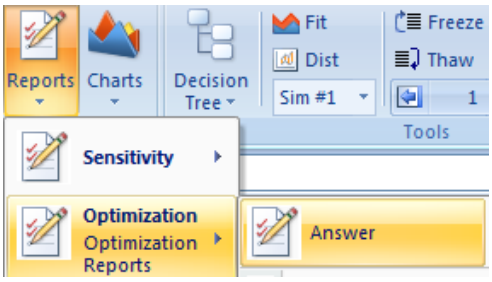
Notice that the Help topic points out the option to **Auto Adjust Chance Constraints**. This is one key to better solutions when robust optimization is used, as in this example. Clicking the  icon at the top of the Task Pane yields an improved solution, and the following solution log:

```
---- Start Solve ----
Model: [StochasticExamples.xls]Gas Company Chance
Using: Psi Interpreter
Using: Full Reparse.
Parsing started...
Diagnosis started...
Parse time: 0.19 Seconds.
Engine: Standard LP/Quadratic
Setup time: 0.00 Seconds.
Engine Solve time: 0.00 Seconds.
(above lines appear 7 times)

Solver has converged to the current solution of the robust
chance constrained problem. All constraints are satisfied.
Solve time: 0.39 Seconds.
```

This was a *successful* solution – but in cases where you have a Solver Result *error message* that you don't understand, or a *solution* that you don't understand, the solution log can be even more helpful.

Choose Available Optimization Reports

As for conventional optimization, the available reports can help you understand the properties of a solution found by Risk Solver Platform. Just select **Reports – Optimization Reports** on the Ribbon – the available reports in the gallery are updated each time you solve.



Below is an Answer Report obtained for the Gas Company Chance model when we first received the Solver Result message "Solver found a conservative solution to the robust chance constrained problem. All constraints are satisfied."

Microsoft Excel 14.0 Answer Report
Worksheet: [StochasticExamples.xls]Gas Company Chance
Report Created: 11/7/2011 3:54:52 PM
Result: Solver found a conservative solution to the robust chance constrained problem. All constraints are satisfied.
Engine: Gurobi Solver
Solution Time: 01 Seconds
Iterations: 0
Subproblems: 0
Incumbent Solutions: 0

Objective Cell (Min)

Cell	Name	Original Value	Final Value
$C$25	Minimize Total Cost	1739.894923	1508.896962

Decision Variable Cells

Cell	Name	Original Value	Final Value	Type
$D$14	Gas to Buy and Resell - year 1	100	100	Normal
$D$15	Gas to Buy and Store - year 1	100.0	0.0	Normal
$D$18	Gas to Buy and Resell - year 2	100	169.4943271	Normal

Constraints

Cell	Name	Cell Value	Formula	Status	Slack
$C$22	Meet Demand - year 1	0	$C$22=0	Binding	0
$C$23	Meet Demand - year 2	47.3	$Var_{0.95}$($C$23)>=0	99.5% satisfied	17.89514027
$D$14	Gas to Buy and Resell - year 1	100	$D$14>=0	Not Binding	100
$D$15	Gas to Buy and Store - year 1	0.0	$D$15>=0	Binding	0
$D$18	Gas to Buy and Resell - year 2	169.4943271	$D$18>=0	Not Binding	169.4943271

Note especially row 26, which reports the status of the chance constraint $VaR_{0.95}$($C$23) >=0: We asked for satisfaction of this constraint **95%** of the time, but at this solution the constraint is being satisfied **99.5%** of the time. This is a strong hint that the solution is conservative, and can be improved.

The Uncertainty Report for this model shows us the subset of the constraints and objective that depend on uncertainty. (*Note*: Since this model is automatically *transformed* to its robust counterpart, which has *no* uncertainty, you must set the Task Pane Platform tab Optimization Model group **Solve Uncertain Models** option to Simulation Optimization, then choose **Analyze without Solving** to produce this report.)

Microsoft Excel 12.0 Uncertainty Report
Worksheet: [StochasticExamples.xls]Gas Company Chance
Report Created: 6/13/2009 7:10:52 PM
Model Type: Stochastic LP Use Of Uncertainty: No Uncertainties

Statistics

	Variables	Functions	Dependents
All	3	3	6
Recourse	0	0	0
Uncertain	1	2	2

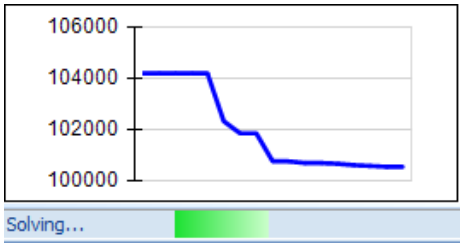
				Exception 1	
Cell	Name	Cell Value	Formula	Uncertainty	Formula
$C$23	Meet Demand - year 2	10.0947284	=D15+D18-C8	$D$5	Gas Company Chance'!$D$5
$C$25	Minimize Total Cost	1519.45697	=(D14*C4) + (C5*D18) + (D15*(C4+C6))	$D$5	Gas Company Chance'!$D$5

When Solving Takes a Long Time

When your model takes a long time to solve, the Task Pane Output tab can be helpful *during* the solution process – at a minimum, to reassure you that the Solver is still making progress, and has not "hung up." If it is not already visible when you first start solving, the Output tab will **appear automatically** after a few seconds of solution time, as long as the Task Pane itself is visible.

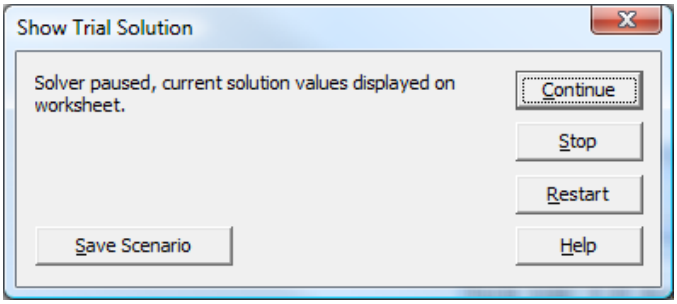
Running Chart of the Objective

The bottom part of the Output tab shows a running chart of the objective of the best solution found so far. This is most often useful for simulation optimization models, which may take some time to solve. Below is an example (staffing for a call center, not included among the installed example files):



Interrupting the Solution Process

You can interrupt the solution process at any time, by **pressing the ESC key**, or by **clicking the** **button** in the Task Pane Output tab. If your model is very large, you might have to hold down the ESC key for a second or two. Since the input focus may not be on the Task Pane when you click with the mouse, you might have to click the button twice. A dialog like the one on the next page will appear.



Click **Continue** if you want to continue solving; click **Stop** to cause the Solver to stop the solution process (this may take a few seconds) and display the message "Solver stopped at user's request." Clicking **Restart** may be useful with the Evolutionary Solver – see the Frontline Solvers Reference Guide.

Reasons Why Solving Takes a Long Time

There are several reasons why solving a stochastic optimization model might take a long time. To better understand why, read the chapter "Mastering Stochastic Optimization Concepts."

1. If your model doesn't meet the requirements for the stochastic programming or robust optimization solution methods, then the slowest method – **simulation optimization** – must be used. Since this requires an optimizer (such as the Evolutionary Solver) that will explore many Trial Solutions, and a *new simulation* must be performed for *every* Trial Solution, this method can take a lot of computing time.
2. If you have the Platform tab Simulation group Interpreter option set to **Excel Interpreter**, Excel is used to recalculate the worksheet on each Monte Carlo trial. This is usually an **order of magnitude slower** than the PSI Interpreter, though it requires significantly less memory.
3. If **model diagnosis** or **problem setup** is slow (see timing messages in the Task Pane Output tab), you might be using array formulas, LOOKUP functions with large ranges as arguments, or long "chains" of formulas where each formula depends on an earlier formula, which can slow down the **PSI Interpreter**. See the analysis in the chapter "Getting Results: Conventional Optimization."
4. Your model might be well formulated, but very large, or it might require non-smooth functions, or many integer variables. A faster processor, multi-core processor, or more memory may help.

When the Solution Seems Wrong

When your optimization results seem wrong, before doing anything else, you should **read any messages** in the Output tab, and **click the message** to display Help as described above. We emphasize this, because very often when we are contacted in technical support, the user has not taken these basic steps.

Bear in mind that the uncertain function cells in your optimization model normally display the calculated value for the **last** Monte Carlo trial of the **last** simulation run. (All of Risk Solver Platform's solution methods – stochastic programming, robust optimization, and simulation optimization – perform at least one simulation). It can be very helpful to **cycle through the Monte Carlo trials**, and examine the values of uncertain function cells, as well as intermediate formula cells on which they depend. Just use the **left and right arrows** on

the Ribbon to change the trial index. You may find that your model calculates a result you did not expect for some values of the uncertain variables.

Although software bugs are always possible, consider carefully the possibility that the simulation results are **correct for the model you've defined**, and that your expectation is wrong. We sometimes find in technical support that what your model actually says is different from what you intended.

When Things Go Right: Getting Further Results

When you *do* receive a solution that makes sense to you, and you have a Solver Result message in **green** at the bottom of the Task Pane for your stochastic optimization model, you have available **all the optimization and simulation reports and charts** that apply to your model and solution. Since a simulation is performed for every stochastic optimization model, you can **double-click** the uncertain function cells in your model to display the Uncertain Function dialog, and obtain many different views of the uncertainty in your model:

Dialog Tabs

- The **Frequency**, **Cumulative Frequency**, and **Reverse Cumulative Frequency** tabs provide three different views of the full range of outcomes for an uncertain function. You can click to display "crosshairs" with numerical values on these charts, or add Lower and Upper bounds to see the estimated probability of a profit or loss, for example.
- The Tornado chart on the **Sensitivity** tab quickly shows you which uncertain variables have the **greatest impact** on this uncertain function, across the full range of Monte Carlo trials. The **Scatter Plots** tab often reveals further insights about the behavior of this uncertain function versus each uncertain variable, or versus other uncertain functions.
- The **Statistics** and **Percentiles** tabs give you a full range of numerical results for this uncertain function, across the full range of Monte Carlo trials. Using PSI Statistics functions, you can place any of these values into a worksheet cell.

Panels and Toolbars

- Using the **right panel** of the dialog, you can change the chart type and color, add titles, legends and gridlines, or add chart markers. From the **Options dialog** (accessible from the Options button on the Ribbon), on the **Charts** and **Markers** tabs, you can set default chart and market settings for all of your charts.
- Using the **title toolbar icons**, you can save your settings, **print** the charts or numbers from any of the dialog tabs, or **copy** the charts or numbers to the Windows Clipboard, where they can be pasted into other applications. You can also **fit a distribution** to the simulation results, as described below.
- Click the rightmost icon on the title toolbar to display the **3-D toolbar** that lets you – with a single click – shift between 2-D and 3-D, **shrink** or **magnify** the chart, or **move** or **rotate** the chart to a different perspective.

Automating Optimization in VBA

Introduction

This chapter explains how to use the Object-Oriented API in Risk Solver Platform and its sub-set products to create, modify and solve optimization models under the control of your custom application written in VBA.

This API is compatible with the object-oriented API offered by Frontline's Solver SDK Platform and Solver SDK Pro, used to build custom applications of optimization and simulation using C++, C#, VB.NET, Java, MATLAB and other languages.

Risk Solver Platform also supports "traditional" VBA functions, which are upward compatible from the VBA functions supported by the standard Excel Solver. This API is described in the Frontline Solvers Reference Guide.

Why Use the Object-Oriented API?

The new object-oriented API is more powerful and much more convenient for programming than the "traditional" VBA functions.

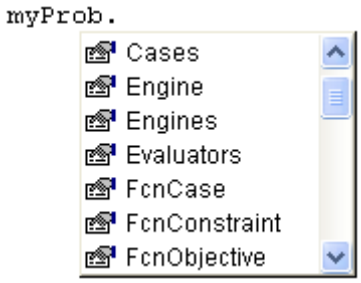
With the "traditional" VBA functions:

- You work with procedural functions that correspond to operations – such as **SolverOK** and **SolverSolve** – you can perform interactively in the Solver dialogs. To access the model and its variables and constraints, you must process the arrays of text and numbers returned by the **SolverGet** function.
- To obtain solution values, you must use the Excel object model (usually the **Range** object) to access the decision variable cells on the worksheet. You must take care to access the correct cells for specific decision variables.
- To obtain dual values and ranges, you must call the **SolverFinish** function to insert a report worksheet into the workbook, then use the Excel object model to access cells in the report. You must take *extra* care to access the correct report cells containing dual values and ranges.

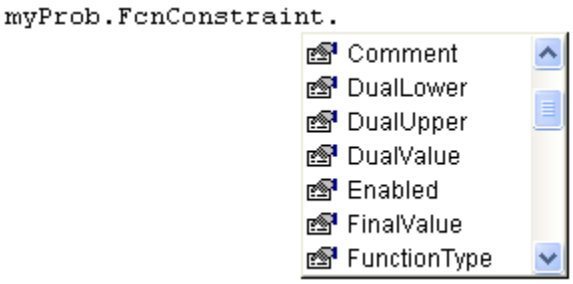
With the new Object-Oriented API:

- You work with objects that correspond to the **Problem**, **Model**, **Solver**, **Engine**, **Variables**, and **Functions**. You can access sets of variables and constraints in the current model directly with expressions such as myProb.VarDecision and myProb.FcnConstraint.
- You can obtain solution values directly, with expressions such as myProb.VarDecision.FinalValue(i). If you need the cell address for a set of decision variable cells, you can write myProb.VarDecision.Name.
- You can access dual values and ranges for variables and constraints directly, with expressions such as myProb.VarDecision.DualValue(i) or myProb.FcnConstraint.DualValue(i).

The result is VBA code that's easier to read, and easier to write in the first place. Since the VBA Editor recognizes the object model exposed by Risk Solver Platform – just as it recognizes the object model exposed by Excel – you'll receive **IntelliSense** prompts as you write code. For example, if you type a line **Dim myProb as New Problem**, then start a new line with **myProb.**, you'll be prompted with the properties and methods available for Problems:



If you select FcnConstraint and then type a period, you'll be prompted with the properties and methods available for Functions:



This makes it much easier to write correct code, without consulting the manual. What's more, you can use this object-oriented API when programming Excel and Risk Solver Platform from new languages such as **VB.NET** and **C#**, working in Visual Studio, and receive IntelliSense prompts in the syntax of these languages!

If you're using new functionality in Risk Solver Platform, the object-oriented API is your best bet. And if you're planning to move your application outside of Excel in the future – so it will run as a standalone program – you'll find that Frontline's **Solver Platform SDK** offers an object-oriented API that closely resembles the new APIs in Risk Solver Platform.

Running Predefined Solver Models

Controlling the Solver can be as simple as *adding two lines* to your VBA 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 Risk Solver Platform COM server (see below), activate the worksheet, and add these two lines of code:

```
Dim prob As New Problem
prob.Solver.Optimize
```

Using the Macro Recorder

If you want to set up a Solver model "from scratch" programmatically, one easy way to see how to use the object-oriented API is to turn on the Excel Macro Recorder (click Record Macro on the Developer tab in Excel 2007, or select Tools Macro Record New Macro in Excel 2000-2003), and then set up a Solver model interactively. Microsoft Excel will record a macro in VBA that calls the object-oriented API to mimic the actions you perform. You can then edit and customize this macro, and incorporate it into your application.

Note: By default, the Excel Macro Recorder will record calls to the object-oriented API.

Adding a Reference in the VBA Editor

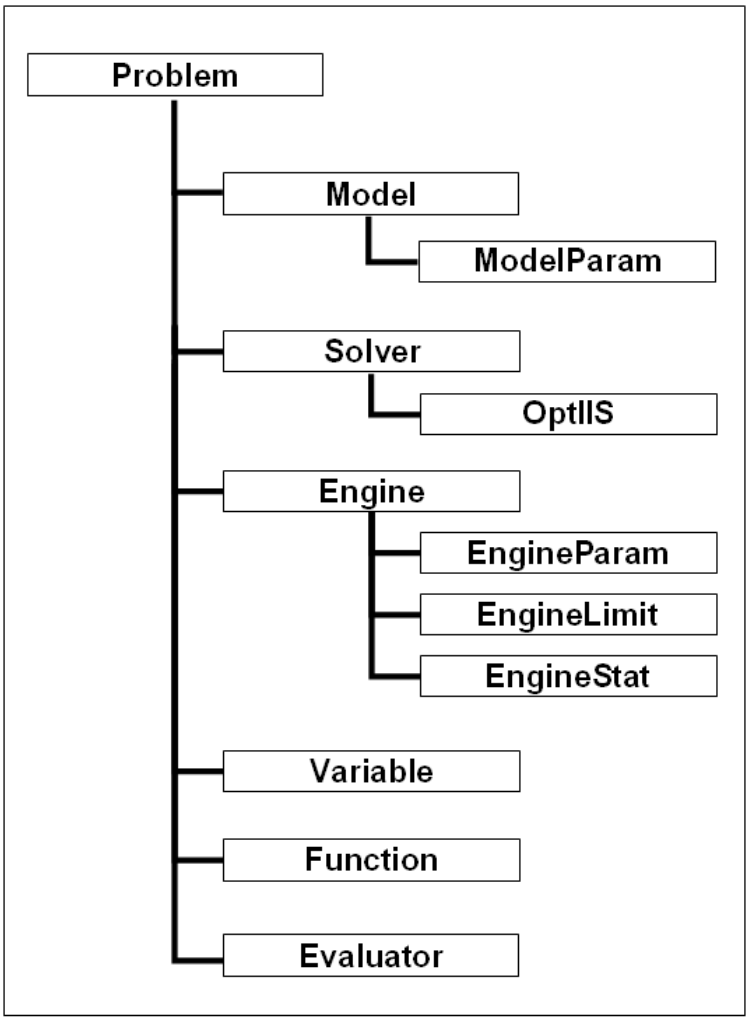
To use the new object-oriented API in VBA, you must first add a reference to the type library for the Risk Solver Platform COM server. To do this:

1. Press Alt-F11 to open the VBA Editor.
2. Select menu choice **Tools References**.
3. Scroll down until you find **Risk Solver Platform 11 Type Library**.
4. Check the box next to this entry, and click OK to close the dialog.
5. Use File Save to save your workbook.

Note that this is a **different** reference from **Solver**, which is the reference you add in order to use the "traditional" VBA functions.

Risk Solver Platform Object Model

Risk Solver Platform makes available a hierarchy of objects for describing optimization problems, as pictured on the next page. Note that the same objects are used for both optimization and simulation problems; see the next chapter, "Automating Simulation in VBA," for more information.

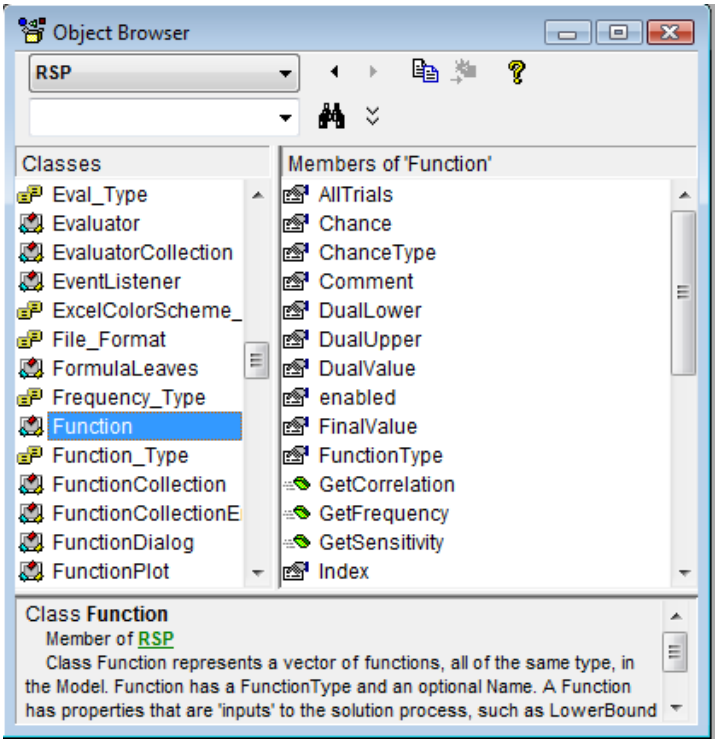


The **Problem** object represents the whole problem, and the **Model** object represents the internal structure of the model, which in Risk Solver Platform is defined by your formulas on the spreadsheet. The **Solver** object represents the optimization process – you call its Optimize method to find an optimal solution. The **Engine** object represents either a built-in or plug-in Solver engine. A **Variable** object represents a range of one or more contiguous cells that contains decision variables, while a **Function** object represents a range of cells that contains either constraint left hand sides or the objective. Each Problem has a collection of Variable objects, and a collection of Function objects. An **Evaluator** represents a function you write that the Solver will call on each iteration (Trial Solution), or on each subproblem in a larger problem.

The Model has a collection of **ModelParam** objects, each representing a single option or parameter of the PSI Interpreter (appearing in the Task Pane Platform tab). An Engine has a collection of **EngineParam** objects, each representing a single option or parameter of a Solver engine (appearing in the Task Pane Engine tab). It also has an **EngineLimit** object, holding problem size limits for this Solver engine, and an **EngineStat** object, holding performance statistics for the last optimization problem solved by this engine. An **OptIIS** object holds results of an infeasibility analysis of the problem.

Using the VBA Object Browser

You can examine Risk Solver Platform objects, properties and methods in the VBA Object Browser. To do this, press **Alt-F11** to open the VBA Editor, and select menu choice **View Object Browser**. This displays a child window like the one pictured below.



The dropdown list at the top left corner of the Object Browser initially displays <All Libraries> – change this to select **RSP**. In the object browser pictured, we've highlighted the properties of the Function object

Programming the Object Model

You use the Risk Solver Platform object-oriented API by first creating an instance of a **Problem**, and initializing it with the Solver model defined on a worksheet in an open workbook. When you do this, a collection of **Variable** objects and a collection of **Function** objects are created automatically. Each Variable object corresponds to a cell range of decision variables that appears in the outlined list of variables, and each Function object corresponds to a cell range of constraints that appears in the list of constraints in the Task Pane.

Once you have an initialized Problem object, you can do several things:

- Set Solver and Engine parameters such as the maximum time or number of iterations, the method used to compute gradients, and other options and tolerances.
- Perform an optimization, and check the final status of the solution process.
- Get results of the optimization, by accessing properties of the Variable and Function objects, and performance statistics, by accessing properties of the EngineStat object.

Example VBA Code Using the Object Model

Below is an example of VBA code that could be linked to a command button on the worksheet:

```
Private Sub CommandButton1_Click()
  Dim prob As New RSP.Problem

  prob.Engine = prob.Engines("Standard LP/Quadratic")
  prob.Engine.Params("MaxTime") = 600

  prob.Solver.Optimize
  MsgBox "Status = " & prob.Solver.OptimizeStatus
  MsgBox "Obj = " & prob.FcnObjective.FinalValue(0)

  For i = 0 To prob.Variables.Count - 1
    For j = 0 To prob.Variables(i).Size - 1
      MsgBox prob.Variables(i).FinalValue(j)
    Next j
  Next i

  Set prob = Nothing
End Sub
```

The first line creates an instance of a Problem, which by default is associated with the Solver model defined on the active worksheet. You could associate the Problem object with a different worksheet by calling the `prob.Init` method:

```
Private Sub CommandButton1_Click()
  ‘create new problem
  Dim prob As New RSP.Problem
‘initialize model on Invent2 worksheet
  prob.Init Worksheets("Invent2")
```

The second line selects the Standard LP/Quadratic Solver engine, and the third line sets the maximum solution time to 600 seconds. The **string names** of parameters such as "MaxTime" are documented in the Frontline Solvers Reference Guide.

The next set of three lines performs the optimization, displays the Solver Result status code (for example 0), and displays the final value of the objective.

The double for-loop in the next five lines steps through the Variable objects – each one representing a range of contiguous cells – and displays the final value for each variable cell in each range.

Evaluators Called During the Solution Process

You can write a VBA function that Risk Solver Platform will call at certain points during the solution process. In this “callback function,” you can access information about the problem and solution so far, to monitor or report progress and decide whether to stop or continue the solution process.

The object-oriented API defines an Evaluator object that is associated with your “callback function” and specifies when Risk Solver Platform should call it. You can define Evaluators to be called on each iteration or Trial Solution, or on each subproblem or each new solution (“incumbent”) when the solution process involves multiple subproblems (global optimization problems, and problems with integer variables).

The VBA function you write to serve as an Evaluator must be contained in a **class module** – not a 'regular' VBA module – and it must be declared to have the **With Events** property.

On the next page is an example of code for an Evaluator, in a class module named Class1.

```
Private WithEvents EvalIterator As RSP.Evaluator

Private Function EvalIterator_Evaluate _
  (ByVal Evaluator As RSP.IEvaluator) As _
  RSP.Engine_Action

  MsgBox "Iteration = " _
    & Evaluator.Problem.Engine.Stat.Iterations _
    & Chr(13) & Chr(10) & "Objective = " _
    & Evaluator.Problem.FcnObjective.Value(0) _
    & Chr(13) & Chr(10)

  EvalIterator_Evaluate = Engine_Action_Continue
End Function

Public Sub MySolve()
  Set EvalIterator = New RSP.Evaluator
  Dim prob As New RSP.Problem
  prob.Evaluators(Eval_Type_Iteration) = EvalIterator
  prob.Solver.Optimize
  Set EvalIterator = Nothing
End Sub
```

Having created the class module Class1, in a 'regular' VBA module you can create an instance of Class1, and then call the MySolve method in Class1:

```
Private Sub CommandButton1_Click()
  Dim c As New Class1
  c.MySolve
End Sub
```

Refinery.xls: Multiple Blocks of Variables and Functions

A further example of programming the object-oriented API is shown in the model **Refinery.xls**, which is installed in the Examples folder, normally at the path C:\Program Files\Frontline Systems\Risk Solver Platform\Examples.

The Refinery.xls model, which is based on Problem 12.6 in the 3rd edition of *Model Building in Mathematical Programming* by H.P. Williams (see the Recommended Books on www.solver.com for details), has ten blocks of decision variables and eleven blocks of constraints, plus bounds on certain variables.

The VBA code for this model illustrates some of the many ways you can use the object-oriented API. For example, the following line displays the time that was required to solve the problem:

```
  MsgBox prob.Engine.Stat.Milliseconds & " msec"
```

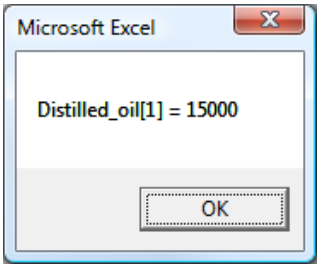
The following line inserts an Answer Report into the active workbook:

```
  prob.Solver.Report "", "Answer"
```

The VBA code on the next page illustrates one way to display final solution values for each of the ten blocks of decision variables in this problem:

```
For i = 0 To prob.Variables.Count - 1
  prob.Variables(i).MakeCurrent
  For j = 0 To prob.VarDecision.Size - 1
    MsgBox prob.VarDecision.Name _
      & "[" & j+1 & "] = " & _
      prob.VarDecision.FinalValue(j)
  Next j
Next i
```

When executed, this code displays MessageBoxes such as:



The line `prob.Variables(i).MakeCurrent` associates the Problem property VarDecision (a single block of decision variables) with each of the ten blocks of variables in turn, allowing you to refer to solution values and dual values of this block in more compact notation. A similar line of code can be used to make the Problem property FcnConstraint represent one of the eleven blocks of constraints. The .Name property of the block, which usually returns a string such as "$A$1:$A$10", returns "Distilled_oil" in this case, since the Excel model has a defined name for this block of cells.

Adding New Variables and Constraints to a Model

So far, we've seen how to access variables and constraints that were defined interactively through the Ribbon and Task Pane (or the Solver Parameters dialog), using the Problem object and its VarDecision and FcnConstraint properties, as well as the Variables and Functions collections. But you can also add new variables and constraints to a model through the Object-Oriented API, and have them appear in the Task Pane Model tab. To illustrate some additional properties used in full Risk Solver Platform, let's add a *recourse* decision variable and a *chance* constraint.

To add a new decision variable, we create a Variable object, associate it with a worksheet cell, set its properties, and add it to the Variables collection. To make this a recourse decision variable, we simply set its VariableType property:

```
Dim v As New RSP.Variable
v.Init "$D$18"
v.VariableType = Variable_Type_Recourse
v.NonNegative
prob.Variables.Add v
```

Similarly, to add a new constraint, we create a Function object, associate it with a formula cell on the worksheet, set its properties, and add it to the Functions collection. To make this a chance constraint, we simply set its FunctionType, ChanceType and Chance properties:

```
Dim f As New RSP.Function
f.Init "$C$23"
f.FunctionType = Function_Type_Chance
f.ChanceType = Chance_Type_VaR
f.Chance = 0.95
f.LowerBound.Array = 0 '>= 0
prob.Functions.Add f
```

CuttingStock.xls: Multiple Problems and Dynamically Generated Variables

A more ambitious example of programming the object-oriented API is shown in the model **CuttingStock.xls**, which is installed in the Examples folder, normally at the path C:\Program Files\Frontline Systems\Risk Solver Platform\Examples.

This application uses the object-oriented API to define and repeatedly solve two optimization problems, passing information back and forth between the two problems. One problem is instantiated from a worksheet with the prob.Init method as mentioned earlier; the other problem is created "from scratch," with new dynamically generated decision variables added each time the problem is solved.

Cutting Stock Problem

CuttingStock.xls solves a classical "cutting stock" problem, which arises for example in lumber and paper mills. Imagine that you have a number of sheets of wood or rolls of paper of a fixed width, waiting to be cut; you have customer orders for sheets or rolls of various different widths. Your task is to cut the larger, fixed-width sheets or rolls into different sizes in a way that minimizes the total stock used while meeting customer demand.

You might for example cut a 100 inch sheet into two sheets of 45 inches (leaving 10 inches wasted), three sheets of 31 inches (leaving 7 inches wasted), one 45-inch sheet and one 31-inch sheet (leaving 24 inches wasted), etc. Each of these is called a *pattern*, and the main problem will have a decision variable representing the number of copies of that pattern to cut. In a 'real-life" application, the number of possible patterns is exponentially large, yielding a model that's too large to solve.

Column Generation Method

We can instead use the technique of *column generation* ('columns' here refers to variables in the main problem). We choose a small initial set of patterns to include in the model, and solve the main problem (an LP). Since it is unlikely that we chose the perfect set of patterns initially, we use the dual variable information from the main problem to generate a new pattern. We generate this new pattern by solving a second optimization problem, called a 'knapsack' problem. A decision variable for the new pattern is dynamically created and added to the main problem, which is solved again. These two problems, the main problem and the knapsack problem, are solved in turn until no more patterns can be generated that will reduce the total stock used.

Worksheets and VBA Code

In CuttingStock.xls, sheet Input contains the 'knapsack' problem, which is solved to generate new patterns, and sheet Patterns contains the main problem. Our VBA code executes a loop, alternately solving the main problem and the

knapsack problem. When the solution to the main problem can no longer be improved, we solve a final problem where we add integer constraints on the variables, so the final solution yields an exact count of the patterns that should be cut.

Open CuttingStock.xls and press Alt-F11 to view its VBA code in the VBA Editor. The first block of code clears the Patterns sheet and sets up the initial patterns. This code simply sets values and formulas into cells, using the Excel object model.

```
  Dim nPat As Integer, i As Integer, _
    nNumDemands As Integer
  nPat = Range("Demands").Count
  nNumDemands = nPat
  Worksheets("Patterns").Activate
  Range("$A$1:$Z$100").Clear

  ' create initial patterns
  For i = 1 To nPat
    Cells(2 + i, i) = Int(Range("RollSize").Value2 _
      / Range("Widths").Cells(i).Value2)
    Cells(2 + i, nPat + 1).Formula = "=sumproduct(" _
      & Range(Cells(1, 1), Cells(1, nPat)).Address
      & "," & Range(Cells(2 + i, 1), _
        Cells(2 + i, nPat)).Address & ")"
    Cells(3 + nPat, i) = Range("RollSize").Value2 - _
        Cells(2 + i, i) *
Range("Widths").Cells(i).Value2
  Next i
```

The code then enters the main loop in which we add a new pattern to the main problem, and set up and solve that problem using Risk Solver Platform's object model:

```
  'generate patterns
  Cells(2, 1).Formula = "=sum(" _
    & Range(Cells(1, 1), Cells(1, nPat)).Address &
")"
  For i = 1 To nNumDemands
    Cells(2 + i, nPat + 1).Formula = "=sumproduct(" _
      & Range(Cells(1, 1), Cells(1, nPat)).Address _
      & "," & Range(Cells(2 + i, 1), Cells(2 + i, _
        nPat)).Address & ")"
  Next i

  Dim prob As New RSP.Problem

  ' variables
  prob.Variables.Clear
  prob.Functions.Clear
  Dim vars As New RSP.Variable
  vars.Init Range(Cells(1, 1), Cells(1, nPat))
  vars.NonNegative
  prob.Variables.Add vars
  Set vars = Nothing

  ' objective
  Dim objective As New RSP.Function
  objective.Init Range(Cells(2, 1), Cells(2, 1))
  objective.FunctionType = Function_Type_Objective
```

```
  prob.Functions.Add objective
  Set objective = Nothing

  ' constraints
  ReDim constraints(1 To nNumDemands) As _
    New RSP.Function
  For i = 1 To nNumDemands
    constraints(i).Init Range(Cells(2 + i, nPat + 1),
_
      Cells(2 + i, nPat + 1))
    constraints(i).LowerBound(0) = _
      Range("Demands").Cells(i).Value2
    prob.Functions.Add constraints(i)
  Next i

  prob.Solver.SolverType = Solver_Type_Minimize
  prob.Engine = prob.Engines("Standard LP/Quadratic")
  prob.Solver.Optimize
```

Next, the code obtains the dual values from the solution to the main problem, via the Risk Solver Platform object model, and stores these values as parameters of the knapsack problem on the Input worksheet, via the Excel object model:

```
  ' capture shadow prices
  For i = 1 To nNumDemands
    Worksheets("Input").Cells(2 + i, 5) = _
      prob.Functions(Range(Cells(2 + i, nPat + 1), _
      Cells(2 + i, nPat + 1))).DualValue(0)
  Next
  Worksheets("Input").Activate
  Dim j As Integer
  j = 1
  For i = 1 To prob.Functions.Count
    If prob.Functions(i - 1).FunctionType = _
       Function_Type_Constraint Then
         Cells(2 + j, 5) = _
           prob.Functions(i - 1).DualValue(0)
         j = j + 1
    End If
  Next i
```

The code then sets up and solves the knapsack problem, again using the Risk Solver Platform object model:

```
  Dim prob1 As New RSP.Problem
  prob1.Init Worksheets("Input")
  prob1.Engine = prob1.Engines("Standard
LP/Quadratic")
  prob1.Engine.Params("IntTolerance").Value = 0
  prob1.Solver.Optimize
```

If the objective value of the knapsack problem is less than 1 (allowing for rounding error) – meaning that there are no more patterns that will improve the solution – we can exit the loop.

```
  If 1 - prob1.FcnObjective.FinalValue(0) _
     >= -0.00001 Then
     Exit Do
  End If
```

Otherwise, the code writes the new pattern to the Patterns (main problem) worksheet, using the Excel object model:

```
  Worksheets("Patterns").Activate
  Cells(nNumDemands + 3, nPat + 1) = _
    Range("RollSize").Value2
  ' write out new pattern, and associated waste
  For i = 1 To nNumDemands
    Cells(2 + i, nPat + 1) = _
      prob1.VarDecision.FinalValue(i - 1)
    Cells(nNumDemands + 3, nPat + 1) = _
      Cells(nNumDemands + 3, nPat + 1).Value2 - _
      prob1.VarDecision.FinalValue(i - 1) * _
      Range("widths").Cells(i).Value
  Next i

  nPat = nPat + 1
  Set prob1 = Nothing
```

When the Do … Loop is exited, all patterns have been generated. Finally, the code solves one more problem with integer constraints on the variables, to ensure that we produce the exact count needed to meet customer demand:

```
  Worksheets("Patterns").Activate
  prob.Init Worksheets("Patterns")
  prob.Functions.Clear
  prob.Variables.Clear

  ' variables
  Dim finalvars As New RSP.Variable
  finalvars.Init Range(Cells(1, 1), Cells(1, nPat))
  For i = 1 To nPat
    finalvars.IntegerType(i - 1) =
Integer_Type_Integer
  Next i
  finalvars.NonNegative
  prob.Variables.Add finalvars

  ' objective
  Cells(3 + nNumDemands, nPat + 1).Formula = _
    "=sumproduct(" & Range(Cells(1, 1), _
    Cells(1, nPat)).Address & "," _
    & Range(Cells(3 + nNumDemands, 1), _
    Cells(3 + nNumDemands, nPat)).Address & ")"
  Dim TotalWaste As New RSP.Function
  TotalWaste.Init Range(Cells(3 + nNumDemands, _
    nPat + 1), Cells(3 + nNumDemands, nPat + 1))
  TotalWaste.FunctionType = Function_Type_Objective
  prob.Functions.Add TotalWaste

  ' constraints
  ReDim constraints(1 To nNumDemands) As _
    New RSP.Function
  For i = 1 To nNumDemands
    Cells(2 + i, nPat + 1).Formula = "=sumproduct(" _
      & Range(Cells(1, 1), Cells(1, nPat)).Address _
      & "," & Range(Cells(2 + i, 1), Cells(2 + i, _
      nPat)).Address & ")"
    constraints(i).Init Range(Cells(2 + i, nPat + 1),
_
```

```
      Cells(2 + i, nPat + 1))
    constraints(i).LowerBound(0) = _
      Range("Demands").Cells(i).Value2
    prob.Functions.Add constraints(i)
  Next i

  prob.Engine = prob.Engines("Standard LP/Quadratic")
  prob.Engine.Params("IntTolerance").Value = 0
  prob.Solver.Optimize
  Set prob = Nothing
```

This example illustrates some of the power of the object-oriented API. Although you could use the "traditional" VBA functions to obtain similar results, it would require quite a bit more code to do so, especially at the step of obtaining the dual values from the solution of the main problem and using them to solve the next knapsack problem.

If you wanted to move this application from Excel to a standalone program, you'd find that nearly all the code in CuttingStock.xls that references the Risk Solver Platform object model could be re-used, with few or no changes, in building an application for the Solver Platform SDK. You'd have to rewrite the code that references cells via the Excel object model to use arrays in a programming language instead, but this would not be difficult.

Automating Simulation in VBA

Introduction

This chapter explains how to use the Object-Oriented API in Risk Solver Platform or Risk Solver Pro to create, modify and solve simulation models under the control of your custom application written in VBA.

In the simplest case, you can use a few standard lines of VBA code to enable and disable Interactive Simulation, as described below. But you can do much more in VBA, to create custom risk analysis applications.

You can define a **Problem** and instantiate it from the spreadsheet with two lines of code, then access the uncertain elements of your model via **Variable** and **Function** objects. You can perform simulations, access trials and summary statistics, and present them the way you want to your end user. All the power of the Excel object model is available, including database access, charts and graphs, and custom dialogs and controls.

Risk Solver Platform's VBA object model closely resembles the object-oriented API of Frontline's **Solver SDK Platform** or **Solver SDK Pro** – which both include a complete toolkit for Monte Carlo simulation. This makes it easier to move an application from Excel to a custom program written in C/C++, Visual Basic, VB.NET, Java or MATLAB.

Adding a Reference in the VBA Editor

To use the new object-oriented API in VBA, you must first add a reference to the type library for the Risk Solver Platform COM server. To do this:

1. Press Alt-F11 to open the VBA Editor.
2. Select menu choice **Tools References**.
3. Scroll down until you find **Risk Solver Platform 11 Type Library**.
4. Check the box next to this entry, and click OK to close the dialog.
5. Use File Save to save your workbook.

Note that this is a **different** reference from **Solver**, which is the reference you add in order to use the "traditional" VBA functions.

You need this reference if your VBA code uses the Event Listener as described in the next section, or uses other elements of the Risk Solver Platform VBA object model, described in later sections of this chapter (or both).

Activating Interactive Simulation

If you simply want to activate or deactivate Interactive Simulation under program control, you need a few standard lines of VBA code. You can place

this code in any VBA procedure you write, and cause it to be run in any manner that is convenient for your application. For example, you could use:

```
Public Sub ISActivate
    Dim ev As New EventListener
    ev.AttachEvent Application
    Set ev = Nothing
End Sub

Public Sub ISDeactivate
    Dim ev As New EventListener
    ev.DetachEvent
    Set ev = Nothing
End Sub
```

You could use the Tools Macro Run menu choice in Excel to run these VBA procedures, or you could associate a Ctrl-Key combination with each procedure, so it is run when that key combination is pressed.

Another approach appears in the example workbook **BusinessPlanPsiChart.xls**: Two buttons named CommandButton1 and CommandButton2 are placed on the spreadsheet, along with two images named Picture Bulb On and Picture Bulb Off, that are stacked on top of each other. The VBA code is:

```
Private Sub CommandButton1_Click()
    Dim ev As New EventListener
    ev.AttachEvent Application
    Set ev = Nothing
    Me.Shapes("Picture Bulb On").Visible = True
    Me.Shapes("Picture Bulb Off").Visible = False
End Sub

Private Sub CommandButton2_Click()
    Dim ev As New EventListener
    ev.DetachEvent
    Set ev = Nothing
    Me.Shapes("Picture Bulb On").Visible = False
    Me.Shapes("Picture Bulb Off").Visible = True
End Sub
```

When the "On" button (CommandButton1) is clicked, Interactive Simulation is activated, and the light bulb is "turned on." When the "Off" button is clicked, Interactive Simulation is deactivated, and the light bulb is "turned off."

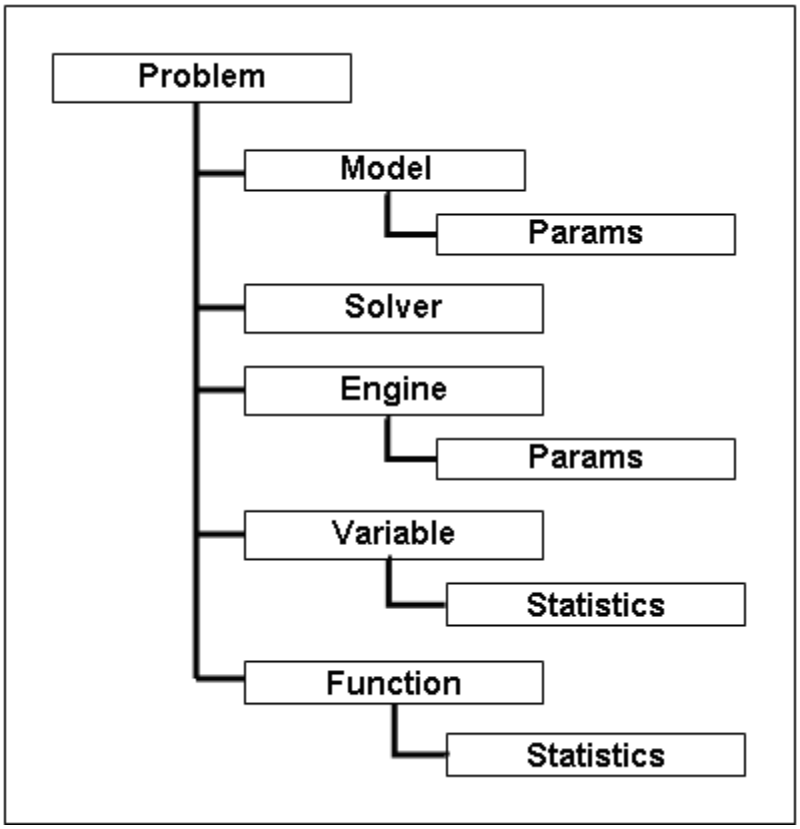
Using VBA to Control Risk Solver Platform

You can do much more with VBA: You can perform simulations under your control, rather than on every worksheet recalculation. You can obtain results of the simulation – statistics, percentiles, or even raw trial data – that you'd otherwise obtain through worksheet functions such as PsiMean(), PsiPercentile(), or PsiData(). You can even define certain worksheet cells as uncertain variables and supply trial data for them in a simulation, without using functions such as PsiNormal() or PsiUniform() in cell formulas.

You also have access in VBA to the Excel object model, which provides a very rich source of high-level functions for manipulating cells and ranges, creating charts and graphs, accessing external databases, and much more. This gives you a powerful set of tools for developing custom risk analysis applications.

Risk Solver Platform Object Model

Risk Solver Platform makes available a hierarchy of objects for describing Monte Carlo simulation problems, pictured below. This object model is a simplified subset of the object hierarchy offered by Frontline's Solver Platform SDK product, which is used to build custom applications in C/C++, Visual Basic, VB.NET, Java or MATLAB. Note that the same objects are used for both optimization and simulation problems.

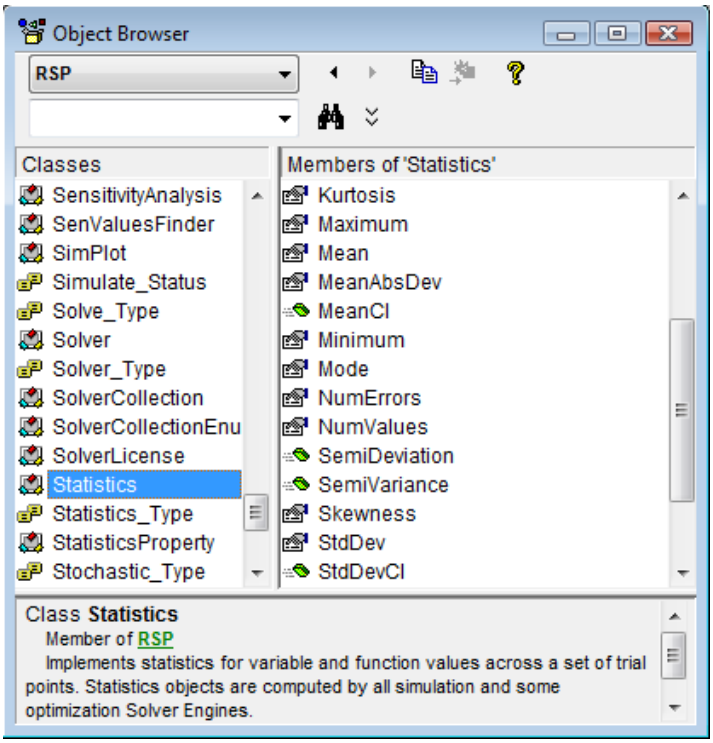


The **Problem** object represents the whole problem, and the **Model** object represents the internal structure of the model, which in Risk Solver Platform is defined by your formulas on the spreadsheet. The **Solver** object represents the Monte Carlo process – you call its Simulate method to perform a simulation. The **Engine** object represents the PSI Technology "engine" – its parameters include the sampling method, for example. A **Variable** object represents a range of one or more contiguous cells that contains uncertain variables, while a **Function** object represents a range of cells that contains uncertain functions. You may have a collection of Variable objects, and a collection of Function objects in one Problem.

Using the VBA Object Browser

You can examine the Risk Solver Platform objects, properties and methods in the VBA Object Browser. To do this, press Alt-F11 to open the VBA Editor, and select menu choice View Object Browser. This displays a child window like the one pictured below. The dropdown list at the top left corner of the Object Browser initially displays <All Libraries> – change this to select RSP.

Below, we've highlighted the properties of the Statistics object, which is a child of the Variable and Function objects.



Using Risk Solver Platform Objects

You use the Risk Solver Platform object model by first creating an instance of a **Problem**, and initializing it with the simulation model defined in your Excel workbook. Once you have an initialized Problem object, you can do several things:

- Add one or more new Variable objects to the problem. (This is an advanced step, covered near the end of this chapter.)
- Set Solver and Engine parameters such as the number of simulations, number of trials per simulation, the sampling method, and the random seed.
- Perform a simulation.
- Get results of the simulation, by accessing properties of the Variable and Function objects, and of their "child" Statistics objects.

The simplest action you might want to take is to create a Problem representing the workbook, set parameters, perform a simulation, and display results. Below is an example that could be linked to a command button on the worksheet:

```
Private Sub CommandButton3_Click()
   Dim prob As New RSP.Problem
   prob.Init ActiveWorkbook

   prob.Solver.NumTrials = 5000
   prob.Engine.Params("SamplingMethod") = 2
   prob.Solver.Simulate
```

```
    For i = 0 To prob.Functions.Count - 1
       MsgBox prob.Functions(i).Statistics.Mean(0)
    Next i
    Set prob = Nothing
End Sub
```

The first two lines create an instance of a Problem, and initialize it with the simulation model defined in your Excel workbook. The next two lines set the number of trials in the simulation to 5000, and the sampling method to Latin Hypercube. The fifth line performs a simulation.

The for-loop in the next three lines will step through the Function objects – assuming, for simplicity here, that each Function object represents just one cell – and display the Mean property of the child Statistics object (the mean or average value of the function across all trials) for each one.

Using Variable and Function Objects

When you create and initialize a **Problem**, a collection of **Variable** objects and a collection of **Function** objects are created automatically. Each Variable object corresponds to a range of one or more cells that contain PSI Distribution functions. Each Function object corresponds to a formula cell referenced as the first argument of a PSI Statistics function, or a range of one or more formula cells referenced by PsiOutput(). For more information on the grouping of cells into Function objects, please see the Frontline Solvers Reference Guide.

Indexing Variable and Function Objects

The VBA code example above assumes that any cells containing =PsiOutput() are separated from each other on the worksheet. When uncertain variable and uncertain function cells lie in a contiguous range – for example A1:A5 or A1:E1 – one Variable or Function object is created to represent all the cells in the range. The Size property of this object tells you the number of cells in the range, and its other properties may be indexed to access statistics – for example – of individual cells. The for-loop above could be written in more general form as:

```
    For i = 0 To prob.Functions.Count - 1
       For j = 0 To prob.Functions(i).Size - 1
          MsgBox prob.Functions(i).Statistics.Mean(j)
       Next j
    Next i
```

Percentiles and AllTrials Properties

In addition to the Statistics child object used above, Variable and Function objects have properties named Percentiles and AllTrials (the raw trial data).

The Percentiles property plays the same role as the PsiPercentile() function on the spreadsheet. It yields a DoubleMatrix object that takes two subscripts: the index (starting from 0) of the cell in the range represented by the Variable or Function object, and the percentile index, which runs from 0 to 98 for the 1st through 99th percentile.

The AllTrials property plays the same role as the PsiData() function on the spreadsheet. It yields a DoubleMatrix object that takes two subscripts: the index (starting from 0) of the cell in the range represented by the Variable or Function object, and the index of the Monte Carlo trial (starting from 0).

The example code below illustrates how to obtain and display the values of the uncertain functions for each simulation trial.

```
For i = 0 To prob.Functions.Count - 1
  If prob.Functions(i).FunctionType = Function_Type_Uncertain
        For j = 0 To prob.Functions(i).Size - 1
           MsgBox prob.Functions(i).Percentiles(j,94)
           For k = 0 To prob.solver.NumTrials -1
                MsgBox prob.Functions(i).AllTrials(j, k)
           Next k
       Next j
  End If
Next i
```

GetFrequency Method

Variable and Function objects also have a GetFrequency method, that plays the same role as the PsiFrequency() function on the spreadsheet. It takes an argument specifying the type of frequency distribution (density, cumulative, etc.) and an array argument specifying the upper limits of the "bins" for which you want to obtain frequency data. It yields a DoubleMatrix object that takes two subscripts: the index (starting from 0) of the cell in the range represented by the Variable or Function object, and the index (starting from 0) of the frequency bin.

The example code below illustrates the use of the Get Frequency method to create 14 different "bins" for categorizing the values of the uncertain functions for each simulation trial.

```
Dim binlimits(14) As Double
binlimits(0) = 0
For Count = 1 To 13
  binlimits(Count) = binlimits(Count - 1) + 25000
Next
For i = 0 To prob.Functions.Count - 1
  Dim mymat As New RSP.DoubleMatrix
  Set mymat =
 prob.Functions(i).GetFrequency(Frequency_Type_Density, binlimits)
  For Count = 0 To 13
  MsgBox "Upper Bin Limit" & ": " & binlimits(Count) & " = " &
mymat(0, Count)
  Next Count
 Next i
```

Controlling Simulation Parameters in VBA

Risk Solver Platform provides a number of parameters you can use to control the simulation process, such as the number of Monte Carlo trials to perform, or the random number seed. You can set these parameters interactively, or by setting certain Solver properties or Engine.Params properties in VBA.

For example, the first line below sets the number of Monte Carlo trials to 5000. The second line set the random number seed to a fixed value – so the same random number sequence is used on each run. The third line sets the sampling method to Latin Hypercube:

```
prob.Solver.NumTrials = 5000
prob.Engine.Params("RandomSeed") = 12345
prob.Engine.Params("SamplingMethod") = 2
```

For a list of simulation parameters that you can set in VBA, see the descriptions of the "EngineParam Object" and the "ModelParam Object" in the Frontline Solvers Reference Guide.

Evaluators Called During the Simulation Process

You can write a VBA function that Risk Solver Platform will call at certain points during the simulation process. In this "callback function," you can access information about the problem and simulation so far, to monitor or report progress and decide whether to stop or continue the simulation process.

The object-oriented API defines an Evaluator object that is associated with your "callback function" and specifies when Risk Solver Platform should call it. The VBA function you write to serve as an Evaluator must be contained in a **class module** – not a 'regular' VBA module – and it must be declared to have the **WithEvents** property.

Here is an example of code for an Evaluator, in a class module named Class1:

```
Private WithEvents EvalSim As RSP.Evaluator

Private Function EvalSim_Evaluate _
  (ByVal Evaluator As RSP.IEvaluator) As _
  RSP.Engine_Action

  MsgBox "Current Simulation = " _
    & Evaluator.Problem.Engine.Stat.Simulations

  EvalSim_Evaluate = Engine_Action_Continue
End Function

Public Sub MySim()
  Set EvalSim = New RSP.Evaluator
  Dim prob As New RSP.Problem
  prob.Init ActiveWorkbook
  prob.Evaluators(Eval_Type_Simulation) = EvalSim
  prob.Solver.Simulate
  Set EvalSim = Nothing
End Sub
```

Having created the class module Class1, in a 'regular' VBA module you can create an instance of Class1, and then call the MySim method in Class1:

```
Private Sub CommandButton1_Click()
  Dim c As New Class1
  c.MySim
End Sub
```

You can define Evaluators to be called when the random sample for all uncertain variables is generated (Eval_Type_Sample), after each simulation is completed (Eval_Type_Simulation), or after each trial is completed

(Eval_Type_Trial). However, because PSI Technology *evaluates all trials in parallel*, an Evaluator to be called on each trial is meaningful only if you use the Excel Interpreter rather than the PSI Interpreter to calculate the worksheet on each trial.

Working with Trials and Simulations in VBA

Risk Solver Platform can perform multiple simulations in one run, where each simulation consists of a number of Monte Carlo trials that you specify. Multiple simulations can be run whenever you choose **Simulate – Run Once**, whenever you change a number with Interactive Simulation, or whenever you call the Problem.Solver.Simulate method in VBA.

In VBA, you can set the property **Problem.Solver.NumSimulations** = *n* to perform *n* different simulations on a single call to Problem.Solver.Simulate.

After a simulation is run, Risk Solver Platform can display each Monte Carlo trial from the simulation on the Excel worksheet, if desired. In VBA, you can set the property **Problem.Solver.TrialIndex** = *n* to display the *n*th trial, or you can call the method **Problem.Solver.TrialStep** to cycle through the trials.

Displaying Normal or Error Trials

As explained in "Error Filtering and Conditional Distributions" in the chapter "Mastering Simulation and Risk Analysis Concepts," Risk Solver Platform **filters out** 'error trials' from your simulation results when computing statistics and displaying charts. An 'error trial' is a Monte Carlo trial where *any* uncertain function returns an Excel error value. However, all trials are saved in memory, and you can access the values of uncertain variables and uncertain functions for each trial, or display each trial on the Excel worksheet.

The Variable and Function objects contain an embedded Statistics object (described later in this chapter). The property **Statistics.NumValues** gives the number of 'normal' trials, over which statistics were calculated, and property **Statistics.NumErrors** gives the number of error trials that occurred.

You can cause a specific Monte Carlo trial to be displayed on the Excel worksheet by setting the property **Problem.Solver.TrialIndex** = *n* to the index (starting from 1) of the trial you want. This causes the PSI Distribution function for each uncertain variable to return the sample value it had on the *n*th trial. The worksheet is then recalculated, so that each uncertain function will have the value it had on the *n*th trial.

To "step through" all trials, normal trials only, or error trials only, and display them on the Excel worksheet, call the method **Problem.Solver.TrialStep** *stepsize*, *trialtype*. Trialtype is 0 for all trials, 1 for normal trials, and 2 for error trials. The *stepsize* may be positive (1, 2, etc.) or negative (-1, -2, etc.). After each call to the TrialStep method, the TrialIndex property is set to the index of the trial just displayed. If stepsize is so large that it would go beyond the last trial (if positive, or the first trial if negative), the TrialIndex will be set to the last (or first) trial, and that trial will be displayed.

Using Multiple Simulations

For multiple simulations to be useful, some parameter of the model – normally something you can control – must have a different value in each individual

simulation. Also, you must access statistics, percentiles, and trial data (if used) for each simulation, so you can compare the results.

If you set **Problem.Solver.NumSimulations** = *n* to perform several simulations on each run, you can use the function **PsiSimParam()**. This function takes either two arguments *lower* and *upper*, as in PsiSimParam(1,3), or one argument that's a cell range or array of numbers, such as PsiSimParam({6.0, 7.5, 9.0}). On the *n*th simulation, PsiSimParam() returns the *n*th value from its argument list, held constant for all the trials in that simulation. In calls to the **PSI Statistics** functions, you can select the simulation for which you want results. For example, if you have an uncertain function in cell F1, you can write =PsiMean(F1,1), =PsiMean(F1,2), and =PsiMean(F1,3) to access the mean value of F1 across all the trials in the first simulation, second simulation, and third simulation, respectively.

If you leave the NumSimulations property at its default value of 1, and call **Problem.Solver.Simulate** inside a for-loop, you can set parameter values that you compute on-the-fly in cells before each simulation using the Excel Range object. You can either get the values of PSI Statistics functions through the Excel object model, or (often better) you can access the Statistics, Percentiles and AllTrials properties, or call the GetFrequency method of your Variables and Functions, to retrieve the results of that simulation.

Creating Uncertain Variables and SLURPs in VBA

The chapter "Mastering Simulation and Risk Analysis Concepts" describes Stochastic Library Units, Relationships Preserved or SLURPs, and shows how to create a SLURP on the spreadsheet. You can also create SLURPs in VBA, and use the SLURP trial data for uncertain variables in your spreadsheet model.

To do this, you first create a Problem object, and initialize it with the simulation model defined in your Excel workbook. At this point, the Problem's collection of Variables will contain Variable objects for each contiguous range of cells containing PSI Distribution functions in the workbook (if any). You can then:

1. Create a new Variable object in VBA.
2. Set its Name property to an unused cell range (say "Sheet1:A1:A5").
3. Set its AllTrials property to the SLURP trial data.
4. Add the Variable object to the Problem's collection of Variables.

The simulation model then behaves just as if these cells contained =PsiSlurp() function calls that referred to the SLURP trial data you supplied via VBA. The SLURP data could be generated by your VBA program, read from a database, or otherwise obtained – it never appears on the Excel spreadsheet. For example:

```
Dim prob As New RSP.Problem
prob.Init ActiveWorkbook

Dim trials As RSP.DoubleMatrix
Set trials = New RSP.DoubleMatrix
trials.InitDense 1, prob.Solver.NumTrials

Randomize
For i = 0 To prob.Solver.NumTrials - 1
    trials(0, i) = Rnd()
Next i
```

```
Dim var As New RSP.Variable
var.VariableType = Variable_Type_Uncertain
var.Init Range("Sheet1$A$1")
var.AllTrials = trials

prob.Variables.Add var
prob.Solver.Simulate
```

The first two lines create an instance of a Problem, and initialize it with the simulation model defined in your Excel workbook. The next three lines create a DoubleMatrix object named *trials* to hold SLURP data, and set its dimensions – just 1 x NumTrials in this case, since we're adding just one uncertain variable.

The next four lines show how the trials matrix can be initialized with newly generated values. Of course, you'd want to use a better, application-specific method – not the Rnd() function – to generate this trial data, or perhaps read the trial data from a database or external file.

In the next three lines, we create a Variable object, set its Name property to the cell address Sheet1!A1 (this must be an empty cell on the spreadsheet), and set its AllTrials property to the trial data we just generated.

In the last two lines, we add the new Variable object to the Problem's collection of Variables, and then perform a simulation.

The newly added Variable object, and the SLURP data represented by its AllTrials property, participates in the simulation model only for so long as the VBA Problem object exists – it is "transient" and is not saved in the workbook. Of course, your VBA program code *is* saved with the workbook, and it can be run at a later time to re-create the Variable object and re-generate or retrieve the SLURP data.

Mastering Conventional Optimization Concepts

Introduction

This chapter explains basic and advanced concepts of optimization, such as 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 Risk Solver Platform (or its optimization sub-set products) and plug-in Solver engines.

This chapter focuses on conventional or *deterministic* optimization models, which do not include any uncertainty. The next chapter "Mastering Simulation and Risk Analysis Topics" explains the concepts of *stochastic* optimization models – those that do include uncertainty – and solution methods such as robust optimization, stochastic programming, and simulation optimization.

If you are using Risk Solver Platform for the first time, we recommend that you try out the examples described in the chapter "Examples: Conventional Optimization" before tackling this material. If you are relatively new to optimization, you may find it useful to read the first section below, "Elements of Solver Models," and then proceed to the Examples chapters. If you've 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 more advanced sections of this chapter, and the next chapter.

Elements of Solver Models

The basic purpose of the Solver is to find a *solution* – that is, values for the *decision variables* in your model – that satisfies all of the *constraints* and maximizes or minimizes the *objective function* 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 play "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 numbers that you use, but you cannot change on your own – for example, prevailing interest rates or supplier's prices. We'll call these values *parameters* of the model. You may have several cases, scenarios, or variations of the same problem to solve, and the parameter values will change in each problem variation; you can define these parameters via the Parameters button on the Ribbon. In this chapter, we'll assume that parameter values are *certain*, but the chapter "Mastering Stochastic Optimization Concepts" will cover situations where the parameter values are *uncertain*.

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 or *decision variables*; the Excel Solver refers to them as 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 Variables group in the Task Pane Model tab, or the Solver Parameters dialog.

The Objective Function

The quantity you want to maximize or minimize is called the *objective* or *objective function*; the Excel Solver often uses the term Set Cell for the objective. For example, this could be a calculated value for projected profits (to be maximized), or costs, risk, or error values (to be minimized). It appears under 'Objective' in the Task Pane Model tab, or the Solver Parameters dialog.

You may have a Solver model that has nothing to maximize or minimize, in which case no cell will be listed for the objective. In this situation the Solver will simply find a solution that satisfies the constraints. Typically this will be only one of many such solutions, located close to the starting values of the decision variables.

The Solver also permits you to enter a specific value that you want the objective function to achieve. This feature was originally included in the Excel Solver to match the Excel Goal Seek... command, which allows you to seek a specific value for a cell by adjusting the value of one other cell on which it depends. Using the 'Value Of' option for the objective cell has the same effect as adding an = constraint in the outlined list, with the objective cell on the left hand side and the constant value on the right hand side; again there is nothing to maximize or minimize.

There is rarely a good reason to use the Value of option. If your problem requires only a single objective cell and a single variable cell with no constraints, you can just use the Goal Seek... command. If you have nothing to maximize or minimize, we recommend that you omit the objective and enter all of your constraints in the outlined list under "Constraints."

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, A1:A5 = semicontinuous, and A1:A5 = alldifferent, are explored in the section "More About Constraints."

A new type of constraint supported by Risk Solver Platform 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 "More About Constraints."

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 Risk Solver Platform and Premium Solver Platform: the LP/Quadratic Solver, SOCP Barrier Solver, nonlinear GRG Solver, Interval Global Solver, and Evolutionary Solver. (Premium Solver Pro uses the Simplex LP Solver, the nonlinear GRG Solver, and the Evolutionary Solver and Risk Solver Pro uses the Evolutionary Solver.) Later sections of this chapter provide an overview of the optimization methods and algorithms employed by each of these Solver engines.

Linear Simplex and LP/Quadratic Solver

The Simplex LP Solver in Premium Solver Pro 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.

In Risk Solver Platform and Premium Solver Platform, the linear Simplex Solver is extended to the LP/Quadratic Solver. 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 Risk Solver Platform and Premium Solver Platform 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 and LSGRG Solvers

The nonlinear GRG Solver (in Premium Solver Pro) and the LSGRG Solver (in Risk Solver Platform and Premium Solver Platform) 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 and LSGRG Solvers 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 or LSGRG Solvers, 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."

Interval Global Solver

The Interval Global Solver in Risk Solver Platform and Premium Solver Platform finds *globally optimal solutions* to problems where the objective and constraints are all smooth (***convex*** or ***non-convex***) functions of the variables. Unlike the Evolutionary Solver or the GRG/LSGRG Solvers with multistart methods, the Interval Global Solver is normally able to determine *for certain* that the solution is globally optimal. The tradeoff is that the Interval Global Solver usually takes much more time to solve a given problem than the GRG Solver, and this time rises steeply as the number of variables and constraints in the problem increases. Hence, the Interval Global Solver is practically able to solve only smaller problems, compared to the GRG and LSGRG Solvers.

Evolutionary Solver

The Evolutionary Solver usually finds *good solutions* to problems where the objective and constraints include non-smooth or discontinuous 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 example, if your model uses IF, LOOKUP or similar functions of the variables, it's likely that the graphs of these functions will contain "jumps" or "breaks." 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 or LSGRG Solver alone; however, the Interval Global Solver or the combination of multistart methods and the GRG and LSGRG Solvers 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 or LSGRG Solvers on such problems. Similarly, it can be used on problems with all-linear functions, but there is little point in doing so when the Simplex, 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, most Solver Engines offer an option "Assume Non-Negative," which automatically places a lower bound of zero on every variable which has not been given an explicit lower bound in the model outline. If you need bounds other than zero, the Task Pane Platform tab Default Bounds section lets you set default *lower* and *upper* bounds on every decision variable.

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, the Interval Global Solver, and multistart methods for global optimization. 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.

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.

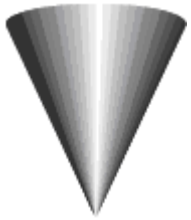
A linear equality constraint (like C1+C2 = 0 above) maintains the convexity of the overall problem, but a nonlinear equality constraint is ***non-convex***, and makes the overall problem non-convex. Interior point methods may have difficulty solving problems with nonlinear equality constraints, since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

A problem with *only* equality constraints (and no objective) is sometimes called a *system of equations*. The Solver can be used to find solutions to systems of both linear and nonlinear equations. If there are several different solutions (sets of values for the decision variables) that satisfy the equations, most Solver engines will find just one solution that is "close" to the starting values of the variables; but the Interval Global Solver in Risk Solver Platform and Premium Solver Platform can be used to find *all* real solutions to a system of smooth nonlinear equations – a capability that was once felt to be beyond the limits of any known algorithm.

Second Order Cone Constraints

Risk Solver Platform 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 and Binary 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 Add Constraint dialog 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.

Semi-Continuous Constraints

It is often useful to place a *semi-continuous* constraint on a decision variable. This specifies that, at the solution, the variable must be either 0, or else a continuous value within a range, determined by the bounds on the variable. For example, if a machine is either "off" or running at a speed between 5 and 50, you can model the machine's speed with A1 = semicontinuous, A1 >= 5 and A1 <= 50. In situations where you might need a binary integer variable *and* a regular (continuous) variable, you can sometimes use a single semi-continuous

variable instead – and such a variable can be handled very efficiently by most Solver engines.

Alldifferent Constraints

A special type of integer constraint supported by Risk Solver Platform and its subset products is called an "alldifferent" constraint. Such a constraint is of the form (for example) A1:A5 = alldifferent, where A1:A5 is a group of 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) can still 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

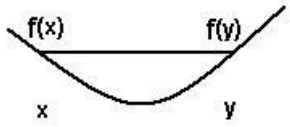
full domain, but smooth – in fact linear – if C1 is constrained to be 10 or more. =SIN(C1) is non-convex over its full domain, but is convex from –pi to 0, or from pi to 2*pi.

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.

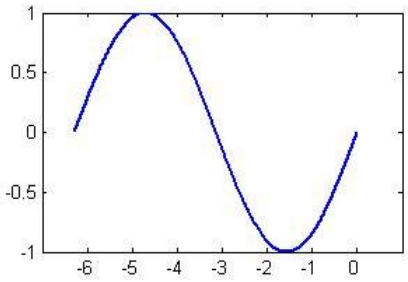
In contrast, if *any* of the constraints are non-convex, or if the objective is either non-convex, concave if minimizing, or convex if maximizing, then the problem is far more difficult: You cannot be certain of finding a feasible solution even if one exists; you must either "settle for" a locally optimal 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, …, x_n) and y =(y_1, y_2, …, 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) <= t*f*(x) + (1-t)*f*(y). A function *f* is *concave* if –*f* is convex, i.e. if *f*(tx + (1-t)y) >= t*f*(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) = t*f*(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 below.



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.

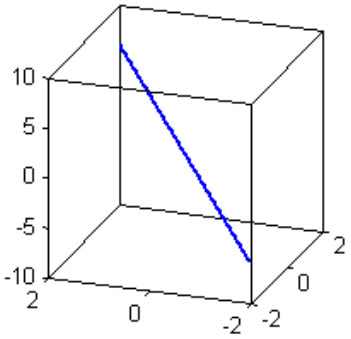
Linear Functions

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:

$a_1x_1 + a_2x_2 + ... + a_nx_n$

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. On the next page 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 and DOTPRODUCT functions compute 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." If the decision variable cells that should participate in the expression are not all contiguous on the spreadsheet, the DOTPRODUCT function can be used instead of SUMPRODUCT.

As explained below in the section "Derivatives, Gradients, Jacobians and Hessians," each coefficient a_i in the linear expression $a_1x_1 + a_2x_2 + ... + a_nx_n$ 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? If you have Risk Solver Platform or Premium Solver Platform, you can easily find out by pressing the Analyze button in the Task Pane. Moreover, you can easily obtain a report showing exactly which cells contain formulas that are nonlinear.

If you have Premium Solver Pro, you can try solving the model with the standard Simplex LP Solver. If the problem contains nonlinear functions of the variables, you will (in virtually all cases) receive the message "The linearity conditions required by this Solver engine are not satisfied." You can then ask the Solver to produce a Linearity Report, which shows whether the objective and each of the constraints is a linear or nonlinear function of the variables. This report also shows which variables occur linearly, and which occur nonlinearly in your model – another way of summarizing the same information. You should next look closely at the objective or constraint formulas that the Linearity Report indicates are nonlinear, and decide whether (or not) the formula can be written in linear form.

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. The QUADPRODUCT function computes values of exactly this form. If we put the constant 5 in A1, 0 in B1, 2 in A2, 4 in B2, 0 in A3 and 3 in B3, then we could write the above example as:

=QUADPRODUCT(C1:C2,A1:B1,A2:B3)

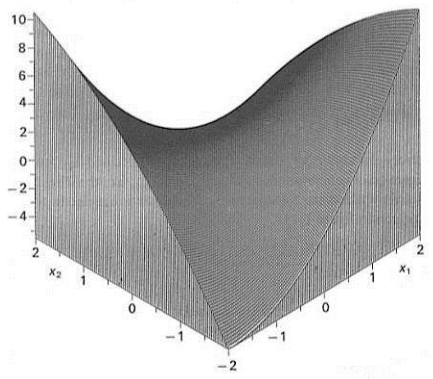
Common uses for quadratic functions are to compute the mean squared error in a curve-fitting application, or the variance or standard deviation of security returns in a portfolio optimization application.

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:

$a_1x_1 + a_2x_2 + ... + a_nx_n$

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 like =IF(C1>10,D1,2*D1). The Interval Global Solver does not accept discontinuous functions at all.

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). The Interval Global Solver does not accept any non-smooth functions.

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; such functions are not accepted by the Interval Global Solver. 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 Risk Solver Platform and its subset products, 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 constraints) 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 Risk Solver Platform or Premium Solver Platform, just create a model using the function, and use the Analyze button to automatically diagnose the model type.

Risk Solver Platform and Premium Solver Platform 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. An example is shown in the EXAMPLE5 worksheet of the **StandardExamples.xls** workbook, which you can easily open from Help – Optimization Examples choice on the Ribbon.

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 $f(x_1,x_2,\ldots,x_n)$ 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 Risk Solver Platform and Premium Solver Platform, versus Premium Solver Pro and Excel Solver, is the method used to compute derivatives. The Polymorphic Spreadsheet Interpreter in Risk Solver Platform and Premium Solver Platform 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 Risk Solver Platform, 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 linear Simplex or 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, when the Assume Linear Model box is checked in the Solver Options dialog. Premium Solver Pro use an improved implementation of the Simplex method with bounds on the variables, the dual Simplex method, and steepest-edge pricing, when the Simplex LP Solver is chosen from the dropdown list in the Task Pane Engine tab. Risk Solver Platform and Premium Solver Platform use 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, stable LU update, and dynamic Markowitz refactorization.

The Large-Scale LP/QP Solver and MOSEK Solver engines use even more powerful implementations of the methods mentioned above. They have been used to solve LP problems with millions of variables and constraints.

The Large-Scale SQP Solver engine includes a powerful linear programming Solver that uses "active set" methods (closely related to the Simplex method). It is practical for problems up to 100,000 variables and constraints. This same Solver engine also handles large-scale QP and NLP problems very efficiently.

The Gurobi Solver and XPRESS Solver engine are Frontline's fastest and most powerful Solvers for linear programming and especially *mixed-integer* linear programming problems. Their advanced 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. The LP/Quadratic Solver in Risk Solver Platform and Premium Solver Platform solves QP problems 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, Large-Scale SQP Solver, KNITRO Solver, Gurobi Solver, MOSEK Solver, and XPRESS 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 the LP/Quadratic Solver, the Large-Scale LP/QP Solver, and the XPRESS 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.

The SOCP Barrier Solver in Risk Solver Platform and Premium Solver Platform 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 Poly-morphic Spreadsheet Interpreter 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 $a_1 \geq \left\| a_2{:}a_5 \right\|_2$. SOCPs are always **convex**; the 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, Large-Scale SQP, or KNITRO Solver engines. Although these Solvers do not recognize SOC constraints directly, Risk Solver Platform and Premium Solver Platform 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 and Premium Solver Pro – 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 Task Pane Engine tab dropdown list box (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. Risk Solver Platform can automatically determine the type of problem, and select only the "good" or "best" Solver engine(s) for that problem.

The GRG Method

Risk Solver Platform and Premium Solver Platform include the enhanced LSGRG Solver while Premium Solver Pro include the 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 Risk Solver Platform and Premium Solver Platform 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.

The SQP Method

The Large-Scale SQP Solver engine for Risk Solver Platform and Premium Solver Platform uses a Sequential Quadratic Programming (SQP) method to solve nonlinear optimization problems. This method forms and solves a QP subproblem, with a quadratic merit function and linearized constraints, on each major iteration. Because it includes a highly efficient QP solver, a powerful linear programming solver using "active set" methods, and sparsity-exploiting matrix factorization, updating and refactorization methods, the Large-Scale SQP Solver engine is very fast in solving all types of LP, QP and NLP problems. However, the SQP method typically follows a path of *infeasible* trial points until it finds the solution that is both feasible and optimal. Hence, if you stop the Solver before it reports an optimal solution, the GRG method is far more likely than the SQP method to return a feasible solution as its "best point so far."

Interior Point and SLQP Methods

The KNITRO Solver engine for Risk Solver Platform and Premium Solver Platform 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 PSI Interpreter 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

Risk Solver Platform and its subset products include 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, the Large-Scale SQP Solver, and the KNITRO Solver; an Interval Global Solver in Risk Solver Platform and Premium Solver Platform that offers powerful interval methods for global optimization in a commercial software product; 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 Risk Solver Platform and its subset products.

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, 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. If you have many integer variables, or alldifferent constraints, try the Evolutionary Solver as an alternative to the multistart method.

The Interval Branch & Bound Method

In contrast to the multistart methods and the Evolutionary Solver's methods, which are *nondeterministic* methods for global optimization that offer no firm guarantees of finding the globally optimal solution, the Interval Global Solver in Risk Solver Platform and Premium Solver Platform uses a *deterministic* method: An *Interval Branch & Bound* algorithm that will find the globally optimal

solution – given enough time, and subject to some limitations related to roundoff error, as discussed in the Frontline Solvers Reference Guide.

The Interval Branch & Bound algorithm processes a list of "boxes" that consist of bounded intervals for each decision variable, starting with a single box determined by the bounds that you specify. On each iteration, it seeks lower and upper bounds for the objective and the constraints in a given box that will allow it to discard all or a portion of the box (narrowing the intervals for some of the variables), by proving that the box can contain no feasible solutions, or that it can contain no objective function values better than a known best bound on the globally optimal objective. Boxes that cannot be discarded are subdivided into smaller boxes, and the process is repeated. Eventually, the boxes that remain each enclose a locally optimal solution, and the best of these is chosen as the globally optimal solution.

Several methods are used to obtain good bounds on the values of the objective and constraints within a box or region. *Classic interval methods* rely on the ability of the PSI Interpreter to evaluate Excel functions over intervals and interval gradients. Local constraint propagation methods (also known as *hull consistency* methods) are used to narrow intervals at each stage of evaluation of the problem functions. *Second-order methods* rely on the Interpreter's ability to compute interval Hessians of Excel functions, and use a variant of the *Interval Newton* method to rapidly minimize function values within a region. Innovative *linear enclosure* methods – implemented for the first time in the Interval Global Solver – bound each problem function with a linear approximation that can be used in a Simplex method-based test for feasibility and local optimality.

The Interval Global Solver also has a unique ability to find *all real solutions* for a system of nonlinear equations – which can be listed in the Solutions Report. It can also find an "inner solution" for a system of nonlinear inequalities – a region or "box" (bounds on the variables) within which *all* points satisfy the inequalities. These capabilities are summarized in the chapter "Getting Results: Conventional Optimization."

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. The Evolutionary Solver, based on genetic algorithms, relies fairly heavily on controlled random search, whereas the OptQuest Solver Engine, based on tabu search and scatter search, relies more heavily on deterministic search methods.

A drawback of these methods is that a solution is "better" only in comparison to other, presently known solutions; both the Evolutionary and OptQuest Solvers normally *have 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.)

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. 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 Risk Solver Platform and Premium Solver Platform, the Evolutionary Solver takes advantage of the diagnostic information available from the PSI 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 has been able to find good solutions to non-smooth problems with many – even hundreds – of constraints.

Tabu Search and Scatter Search

The OptQuest Solver Engine for Risk Solver Platform and Premium Solver Platform is based on the principles of tabu search and scatter search. These methods have strong analogies with – and actually predate – genetic algorithm methods, but they rely less heavily on random choice. They work with a population of solutions, which are modified and combined in different ways, then subjected to a selection process. Scatter search methods can sample the space of possible solutions, avoid becoming "trapped" in regions close to local optima, and adaptively diversify or intensify the search. Tabu search uses memory of past search steps to avoid repeated steps and improve future searches. Use of the OptQuest Solver is described in more depth in the Frontline Solvers Reference Guide.

Integer Programming

When a Solver model includes integer constraints (for example A1:A10 = integer, A1:A10 = binary, A1:A10 = semicontinuous, or A1:A10 = alldifferent), it is called an *integer programming* problem. Integer constraints 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 Pro 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. In Risk Solver Platform and Premium Solver Platform, the 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

The LP/Quadratic Solver, the Large-Scale LP/QP Solver, Large-Scale SQP Solver, MOSEK Solver, Gurobi Solver, and XPRESS Solver 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 Risk Solver Platform and its subset products, 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, Risk Solver Platform employs 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, Large-Scale GRG Solver, Large-Scale SQP Solver, and KNITRO Solver engines use this extended Branch & Bound method to solve problems with integer and alldifferent constraints.

The Large-Scale LP/QP Solver, MOSEK Solver, Gurobi Solver, and XPRESS 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.

Looking Ahead to Models with Uncertainty

If you've read through this chapter, **congratulations** – you've learned a *great deal* about the nature of optimization problems, and how they are solved! But as we said in "Elements of Solver Models" at this chapter's beginning, we've assumed that all parameters of the model are *certain*. The chapter "Mastering

Stochastic Optimization Concepts" will cover situations where the parameter values are *uncertain* – and the problems are called *stochastic optimization* problems. In these problems, it's ***more important than ever*** to know how the objective and constraints depend on the decision variables.

We'll see that **stochastic linear programming** problems can be transformed and solved to optimality far more easily and quickly than **stochastic nonlinear, non-convex** problems. We'll also see that certain ways of summarizing the **uncertainty** involved in the model, in chance constraints and expected-value and risk-measure objectives, allow us to form a **convex** optimization model, that can be solved quickly and 'scaled up' to large size. We'll see that even one **decision-dependent uncertainty** usually makes a model non-convex and non-smooth, and far more difficult to solve – but we can still find *good* solutions for such a model, using high-speed simulation optimization in Risk Solver Platform.

We explained in this chapter that it's usually a mistake to apply a *general-purpose* Solver to a problem that has a simpler structure – for example, applying the GRG Solver or Evolutionary Solver to a linear programming problem: Solution times are much longer than necessary, solutions are less reliable, and the problem typically cannot be scaled up to large size and still solved by these methods.

In a similar way, we'll see that it's usually a mistake to apply a general-purpose *stochastic optimization method* to a problem that has a simpler structure – for example, applying **simulation optimization** (which requires a general-purpose Solver) to a stochastic linear programming problem. Again solution times will be *much* longer than necessary, solutions are *much* less reliable, and the problem typically cannot be scaled up to large size and still solved by these methods.

Mastering Simulation and Risk Analysis Concepts

Introduction

To build a Monte Carlo simulation model in Excel, you begin with a conventional spreadsheet model, designed for 'what-if' analysis. Next, you identify the *inputs* to your model that are uncertain, and use PSI Distribution functions to describe the uncertainty. Then, you identify the *outputs* of special interest (such as Net Profit), and use PSI Statistics functions to examine or summarize how they behave in light of the uncertainty.

This User Guide assumes you have some experience building conventional spreadsheet models, and you almost certainly have experience using such models to ask 'what if' questions, by manually changing input values on the spreadsheet. **The key step in risk analysis is to think in terms, not of just one 'what if' scenario at a time, but of a *range of scenarios, considered at once*.** Think of your spreadsheet cells as representing not just single numbers, that you change once in a while to explore alternatives, but *arrays* of numbers that cover the range of possibilities.

Although you can usually visualize the range of values for one *input* value at a time, it is very difficult to foresee – without computer assistance – the range of outcomes for an *output* value that depends on *several* interacting inputs, each one subject to uncertainty. But Risk Solver Platform and Risk Solver Pro *automatically* compute the full range of outcomes for every cell in your spreadsheet model, and enables you to quickly see statistics such as the mean, standard deviation, or 10^{th} and 90^{th} percentiles of the range of outcomes.

What Happens During Monte Carlo Simulation

You'll find it easier to understand simulation results if you have a good grasp of the Monte Carlo simulation process. At its heart, this process is very simple. It consists of the following steps:

1. Generate a **random sample** for the **uncertain variables** in your model. If you specify (say) 1,000 Monte Carlo trials per simulation, then 1,000 randomly chosen values will be generated for *each* uncertain variable.
2. For each of 1,000 Monte Carlo trials, **recalculate** your model, with the right sample values in each uncertain variable cell. When Excel is used to recalculate, the PSI Distribution function in each uncertain variable cell returns the correct sample value for that trial. When the PSI Interpreter is used, this is done internally (much faster) by Risk Solver Platform.
3. On each Monte Carlo trial, monitor and save the calculated value of each **uncertain function** in your model. (Recall that any formula cell containing a call to PsiOutput(), or referenced in the first argument of PsiOutput() or a

PSI Statistics function, is monitored as an uncertain function cell.) For 1,000 trials, there will be 1,000 saved values for *each* uncertain function.

When the simulation process is complete, Risk Solver Platform uses the 1,000 saved values of each uncertain function to calculate statistics and percentiles, draw frequency distributions, scatter plots and other charts, and compute values for each PSI Statistic function call in your model.

Random Number Generation and Sampling

On each Monte Carlo trial, *sample values* are drawn from the probability distributions represented by the PSI Distribution functions in your model. Sample values are computed by first *drawing* a "random number" between 0 and 1, then *transforming* this uniform random sample value into a sample value that:

- Constrains the samples drawn to obtain better coverage of the sample space, where each PSI Distribution function is a 'dimension' of that space
- Ensures that the frequency distribution of samples drawn properly reflects the shape and parameters of the PSI Distribution function
- Ensures that the samples drawn for multiple PSI Distribution functions properly reflect the correlation of distributions with each other

Random Number Generator

Risk Solver Platform includes an advanced set of random number generation capabilities. In common applications, any good random number generator is sufficient – but for challenging applications (for example in financial engineering) that involve many uncertain variables and many thousands of trials, the advanced features of Risk Solver Platform can make a real difference.

Computer-generated numbers are never truly "random," since they are always computed by an algorithm – they are called *pseudorandom* numbers. A random number generator is designed to quickly generate sequences of numbers that are as close to statistically independent as possible. Eventually, an algorithm will generate the same number seen sometime earlier in the sequence, and at this point the sequence will begin to repeat. The *period* of the random number generator is the number of values it can generate before repeating.

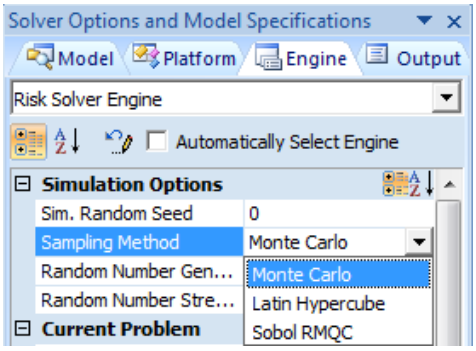
A long period is desirable, but there is a tradeoff between the length of the period and the degree of statistical independence achieved within the period. Hence Risk Solver Platform offers a choice of four random number generators:

- **Park-Miller** "Minimal" Generator with Bayes-Durham shuffle and safeguards. This generator has a period of 2^{31}-2. Its properties are good, but the following choices are usually better.
- Combined Multiple Recursive Generator (**CMRG**) of L'Ecuyer. This generator has a period of 2^{191}, and excellent statistical independence of samples within the period.
- Well Equidistributed Long-period Linear (**WELL1024**) generator of Panneton, L'Ecuyer and Matsumoto. This very new generator combines a long period of 2^{1024} with very good statistical independence.
- **Mersenne Twister** generator of Matsumoto and Nishimura. This generator has the longest period of 2^{19937}-1, but the samples are not as "equidistributed" as for the WELL1024 and CMRG generators.

Random Number Seeds

As explained in the chapter "Getting Results: Simulation," the **seed**, or initial value, of the random number generator determines whether your results are *exactly reproducible* when you re-run a simulation, or whether your results are *similar but not identical* because a different random sample was drawn.

You can set a seed for the entire simulation run using the **Sim. Random Seed** option – the first option on the Task Pane Engine tab when you select **Risk Solver Engine** (the simulation engine) from the dropdown list. Any positive integer sets a specific seed; 0 means the seed will be *different* on every run.



You can also set a seed for any specific uncertain variable, using the Seed option in the Uncertain Variable dialog, or by supplying a **PsiSeed()** property function as an argument to the PSI Distribution function call. This means the uncertain variable will have its own independent stream of random numbers starting from the given seed, whether or not you've set a seed for the whole simulation run.

Sampling Method

In standard Monte Carlo sampling, numbers generated by the chosen random number generator are used directly to obtain sample values for the uncertain variables (PSI Distribution functions) in the model. With this method, the variance or *estimation error* in computed samples for uncertain functions is inversely proportional to the square root of the number of trials; hence to cut the error in half, four times as many trials are required.

Risk Solver Platform provides two other sampling methods than can significantly improve the 'coverage' of the sample space, and thus reduce the variance in computed samples for output functions. This means that you can achieve a given level of accuracy (low variance or error) with fewer trials. You choose this via the **Sampling Method** option on the **Engine tab**, shown above.

Latin Hypercube Sampling. Latin Hypercube sampling begins with a stratified sample in each dimension (one for each uncertain variable), which constrains the random numbers drawn to lie in a set of subintervals from 0 to 1. Then these one-dimensional samples are combined and randomly permuted so that they 'cover' a unit hypercube in a stratified manner. This often reduces the variance of uncertain functions.

Sobol numbers (Randomized QMC). Sobol numbers are an example of so-called "Quasi Monte Carlo" or "low-discrepancy numbers," which are generated with a goal of coverage of the sample space rather than "randomness" and statistical independence. Risk Solver Platform adds a "random shift" to Sobol numbers, which improves their statistical independence. Sobol numbers are frequently used in quantitative finance applications, where they are often effective at reducing variance.

Random Number Streams

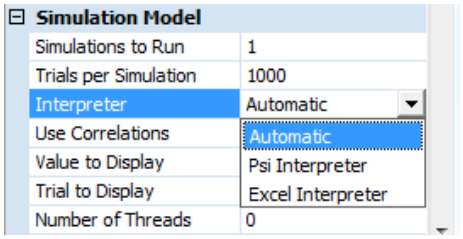
Most Monte Carlo simulation tools generate a *single* sequence of random numbers, taking values consecutively from this sequence to obtain samples for each of the distributions in a model. This introduces a subtle dependence between the samples for all distributions in one trial. Risk Solver Platform allows you to specify that an independent random number sequence (stream) should be used for each distribution in the model – using the **Random Number Streams** option on the **Engine tab**, as shown on the previous page. This capability works for Monte Carlo sampling and Latin Hypercube sampling; it does not apply to Sobol numbers.

In many applications, the difference between a single stream and multiple strams is very small – but in some cases, found in financial engineering and other demanding applications, better results are obtained if independent random number sequences (streams) are used.

The PSI Interpreter and Simulation

Risk Solver Platform and Risk Solver Pro can use either Excel, or its own *Polymorphic Spreadsheet Interpreter* (PSI Technology) to perform Monte Carlo simulation trials at high speed – often 100 times faster or more than performing the trials by allowing Microsoft Excel to recalculate the spreadsheet. Normally, you'll want to use the PSI Interpreter for simulation trials, since it is designed to compute the same values as Excel does, but much faster than the Excel Interpreter.

However, there are a few features of Excel formulas and functions that the PSI Interpreter does not handle; if you use these features in your model, you'll see an error message when you try to run a simulation. If your model requires the use of Excel features that are not supported by the PSI Interpreter, you may have to use the Excel Interpreter instead. To do this, change the Task Pane Platform tab Simulation Model group **Interpreter** option, as shown below.



More About the PSI Interpreter

So, how does the PSI Interpreter work, and how can it be so much faster than Excel alone for simulation? To answer this, we must look at how Excel itself computes values for your formulas.

Microsoft Excel is an *interpreter* for spreadsheet formulas. When you enter a formula such as =A1+A2*(A3-4) in cell B1, Excel first scans and *parses* the formula, storing the results in a coded internal form. It also maintains storage for A1, A2, A3 and thousands of other cells. When you change a number and recalculate, Excel refers to the coded internal form, looks up the values of A1, A2 and A3 and fetches the constant 4, performs the arithmetic, and saves the result in storage reserved for B1. *Only a small part of the time is spent on the actual arithmetic* – most of the time goes into processing the encoded formula, and looking up and saving cell values.

PSI stands for ***Polymorphic Spreadsheet Interpreter***. PSI is also an *interpreter* for spreadsheet formulas, that accepts the same formula syntax and built-in functions as Microsoft Excel. But where Excel evaluates formulas only for one datatype – single numbers, PSI can evaluate formulas for many different datatypes. (The word "polymorphic" comes from object-oriented programming, where it has essentially the same meaning.)

One of the special PSI datatypes is Monte Carlo trials, where each number is replaced by a vector of 1,000 or more trial values. The heart of PSI speed comes from the fact that it processes the encoded formula *once*, fetches 1,000 data values *at once*, and performs the arithmetic for all 1,000 values *at once*. Hence, the "overhead" of the interpreter is incurred *once* rather than 1,000 times; *most of the time is spent doing the actual arithmetic*. PSI is so fast because it is *specialized* for simulation (and optimization).

Uncertain Functions, Statistics, and Risk Measures

Once a Monte Carlo simulation is complete, what statistics should you use to evaluate the outputs of special interest in your model? The answer depends on your application, but some general guidelines can be given.

Descriptive statistics are usually classified into measures of central tendency, and measures of variation or dispersion. You should look at both kinds of measures, and at some kind of *quantile* measure, to assess almost any output in your model.

Measures of Central Tendency

Risk Solver Platform provides several measures of central tendency:

- **PsiMean**, the average of all the values
- **PsiPercentile** for the median or 50^{th} percentile
- **PsiMode**, the most frequently occurring single value

Measures of Variation

Risk Solver Platform provides several standard measures of variation:

- **PsiVariance**, which describes the *spread* of the distribution of values
- **PsiStdDev** for standard deviation, the square root of variance
- **PsiSkewness**, which describes the *asymmetry* of the distribution of values
- **PsiKurtosis**, which describes the *peakedness* of the distribution of values
- **PsiMin**, **PsiMax**, and **PsiRange** for the minimum and maximum values, and the difference between them

Risk Measures

Risk Solver Platform also provides several risk measure functions that are most often used in quantitative finance applications, but may be used in any model.

- **PsiAbsDev** for 'MAD', which measures *absolute* deviations from the mean

- **PsiSemiVar** for semivariance or lower partial moment, which measures and weights *negative* deviations from the mean
- **PsiSemiDev** for semideviation, the square root of semivariance (*q*th root for the lower partial moment)

PsiSemiVar and PsiSemiDev are useful in situations where 'upward' variation – for example, higher stock prices or increased profits – is desirable, but 'downward' variation – lower prices or losses – is undesirable.

Quantile Measures

To get a complete grasp of the range of outcomes, it's essential to look at *quantile* measures, such as percentiles and Value at Risk, in addition to measures of central tendency and variation. Quantile measures allow you to answer questions such as 'How much money might we lose, with 5% or 10% probability?" or "What is the probability that we'll make at least $100,000?" based on your simulation model. Risk Solver Platform provides:

- **PsiPercentile**, which provides percentile values from 1% to 99%
- **PsiTarget**, which returns the proportion of values less than or equal to a target value
- **PsiBVaR**, which measures standard ('Basel') Value at Risk
- **PsiCVaR**, which measures Conditional Value at Risk

Confidence Intervals

Every Monte Carlo simulation uses a *sample* of the possible values of your uncertain variables; hence any statistic resulting from the simulation involves some degree of sampling error. For the mean and standard deviation of an output value, Risk Solver Platform provides functions that help you assess this error, and estimate the range or interval in which you can be *confident* that the true mean or standard deviation lies, at a confidence level that you specify:

- **PsiMeanCI**, which returns a confidence interval for the mean
- **PsiStdDevCI**, which returns a confidence interval for the standard deviation
- **PsiCITrials**, which returns the number of simulation trials needed to obtain a confidence interval of a given size, at a given confidence level

Uncertain Variables and Probability Distributions

For experienced spreadsheet modelers, the most challenging task in creating a Monte Carlo simulation model is usually not identifying the key inputs and outputs, but selecting an appropriate probability distribution and parameters to model the uncertainty of each input variable.

Risk Solver Platform provides over 40 analytic probability distributions – which one should you use? Again the answer depends on your application, but some general guidelines can be given.

If a **Certified Distribution** (see below under "Probability Management Concepts") is available for an uncertain variable – for example, your company's estimate of the range and probabilities for prices of certain chemical feedstocks,

or a service provider's estimate of the range and probabilities for stock or bond prices – you can simply use the Certified Distribution.

Discrete Vs. Continuous Distributions

If you must choose or create your own distribution, the first step is to determine whether to use a **discrete** or **continuous** form. If there are a small number of possible values for the uncertain variable, you may be able to use a discrete analytic distribution, or construct a discrete custom distribution. If the underlying physical process involves discrete, countable entities – such as the number of customers arriving at a service window – you can use a discrete distribution. If the possible values are highly divisible – such as most prices, volumes, interest rates, exchange rates, weights, distances, etc. – you will likely use a continuous distribution. In some cases, you may use a continuous distribution to *approximate* a discrete distribution.

Bounded Vs. Unbounded Distributions

Another characteristic that distinguishes input distributions is the range of sample values they can generate. Some distributions are intrinsically **bounded** – samples are guaranteed to lie between a known minimum and maximum value. Examples are the Uniform, Triangular, Beta, and Binomial distributions. Other analytic distributions are **unbounded** – sample values may cluster around the distribution's mean, but may sometimes have extreme negative or positive values. Examples are the Normal, Logistic, and Extreme Value distributions. Still other distributions are **partially bounded**, with a known minimum such as zero, but no maximum value. Examples are the Exponential, Poisson, and Weibull distributions.

At times, you may find that the most appropriate distribution (say the Normal) is unbounded, but you know that the realistic values of the physical process are bounded, or your model is designed to handle values only up to some realistic limit. You can impose bounds on any distribution using Risk Solver Platform's Uncertain Variable dialog. For information about "cutoff bounds" and "censor bounds" (both may be used), consult the Frontline Solvers Reference Guide.

Analytic Vs. Custom Distributions

A third characteristic of input distributions is whether they are analytic (also called parametric) or custom (sometimes called non-parametric) distributions. An **analytic** distribution has a form derived from certain theoretical assumptions about the problem. For example, a Poisson distribution is derived from an assumption that events are *independent* and occur at a known *average rate*, and an Exponential distribution is derived from an assumption of a *constant rate* of decay in some process. A **custom** distribution has a form dictated by either past data or expert opinion about the range and frequency of sample values. Risk Solver Platform offers five general-purpose functions – PsiCumul, PsiDiscrete, PsiDisUniform, PsiGeneral and PsiHistogram – to help you model custom distributions. Generally speaking, you should choose an analytic distribution if – and *only* if – the theoretical assumptions truly apply in your situation.

Creating your Own Distributions When Past Data is Available

If you have, or you can collect **data on the past performance** of the uncertain variable – and if you believe that 'past performance' is likely to be representative of future performance – you have three options:

- If you can **fit the data** (past observations) to a specific type of analytic distribution and its parameters, and if there is reason to believe that the underlying process that the uncertain variable is measuring is *consistent* with the *assumptions* from which the analytic distribution is derived, you can use this distribution (for example PsiNormal, PsiWeibull, etc.) for the variable. See "Using the Fit Feature" section below.
- If you have a reasonably **large number** of observations of past performance of the variable, compared to the number of simulation trials you want to run, you can use the *past data itself* for simulation trials, in the form of a SIP (Stochastic Information Packet) or a DIST (Distribution String), and use the PsiSip() or PsiSlurp() distribution function for the uncertain variable. See "Probability Management Concepts" and, for DISTs, "Stochastic Libraries: SIPs and SLURPs" below.
- If – as is often the case – you have a relatively **small number** of observations of past performance compared to the number of trials you want to run, you may be better off *resampling* the past performance data. To do this, store the past data in a cell range or SIP, and use the PsiDisUniform() function (single values) or the PsiResample() function (multiple values). The difference is that, instead of using all of the past observations (one per simulation trial), you *randomly sample* the past observations on each trial.

You can use the Risk Solver Platform GUI to find the best-fitting analytic distribution. Both discrete and continuous distributions can be fitted to data; 29 common distributions (out of the 40+ available) are used for fitting. You can also call the Distribution object Fit and AutoFit methods in your VAB code, as described in the chapter "Using the VBA Object Model."

When Past Data is Not Available

If you *don't* have, and you cannot easily collect data on the past performance of the uncertain variable – or if past performance is not likely to be representative of future performance – you must tackle the problem in a different way:

- Consult the **literature for your industry**, if available, to find examples of applications like yours where simulation models were built. Find out – by contacting the authors if necessary – what kinds of distributions were used for the uncertain variables, and the rationale for choosing them.
- If you cannot find reports on industry-specific applications like yours, consult the **publications of professional societies** like INFORMS, where simulation applications are reported. One rich source is the past proceedings of the Winter Simulation Conference (www.wintersim.org).
- In the chapter **PSI Function Reference**, read the descriptions of the different PSI Distribution functions, which include brief comments on the types of applications where each distribution has been used in the past. See the books listed at the beginning of that chapter for further information.

You are well-advised to *keep it simple!* Many physical, social and biological phenomena are well described by the Normal distribution, or – *if* the possible

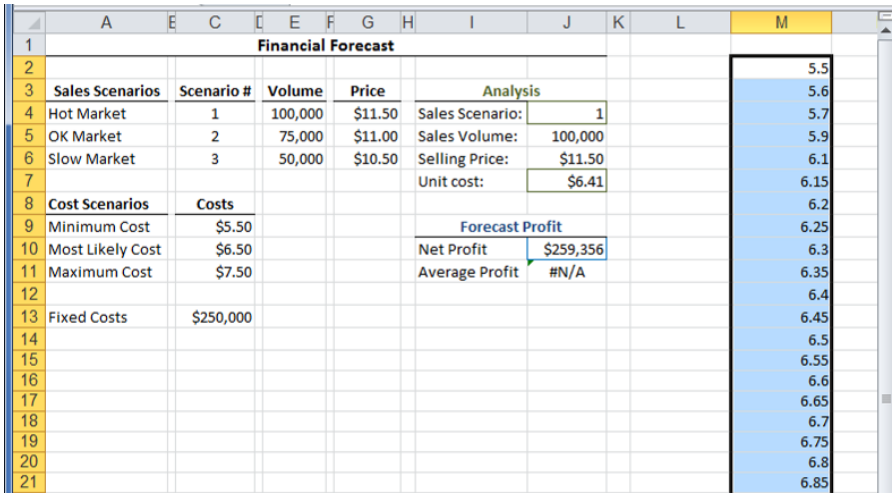
values are *equally likely* to occur, as in a coin flip or single die – the Uniform distribution. Bear in mind that when any set of distributions are summed, the result (quickly) tends towards the Normal distribution.

Applications that involve *queuing* – customers arriving or departing, parts awaiting assembly, etc. – have been well studied, so you can often find appropriate distributions in the literature. Applications that use the Project Evaluation and Review Technique (PERT) can often use the PsiPert() function to model uncertainty.

Using the Fit feature

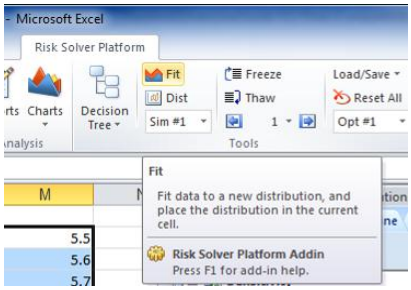
To fit a series of data using the Fit tool in Risk Solver Platform or Risk Solver Pro follow the following steps:

1. Select the range of data you want to fit. In we've selected data in Column M.

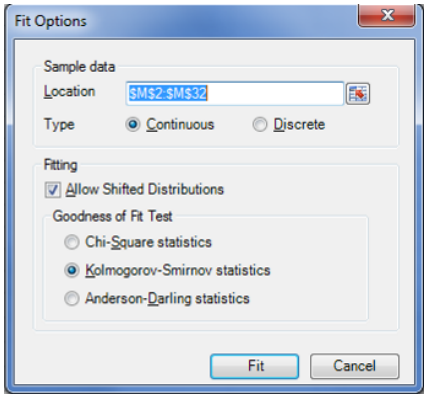


	A	C	E	G	I	J	M
1	Financial Forecast						
2							5.5
3	Sales Scenarios	Scenario #	Volume	Price	Analysis		5.6
4	Hot Market	1	100,000	$11.50	Sales Scenario:	1	5.7
5	OK Market	2	75,000	$11.00	Sales Volume:	100,000	5.9
6	Slow Market	3	50,000	$10.50	Selling Price:	$11.50	6.1
7					Unit cost:	$6.41	6.15
8	Cost Scenarios	Costs					6.2
9	Minimum Cost	$5.50			Forecast Profit		6.25
10	Most Likely Cost	$6.50			Net Profit	$259,356	6.3
11	Maximum Cost	$7.50			Average Profit	#N/A	6.35
12							6.4
13	Fixed Costs	$250,000					6.45
14							6.5
15							6.55
16							6.6
17							6.65
18							6.7
19							6.75
20							6.8
21							6.85

2. Select the "Fit" icon on the Ribbon

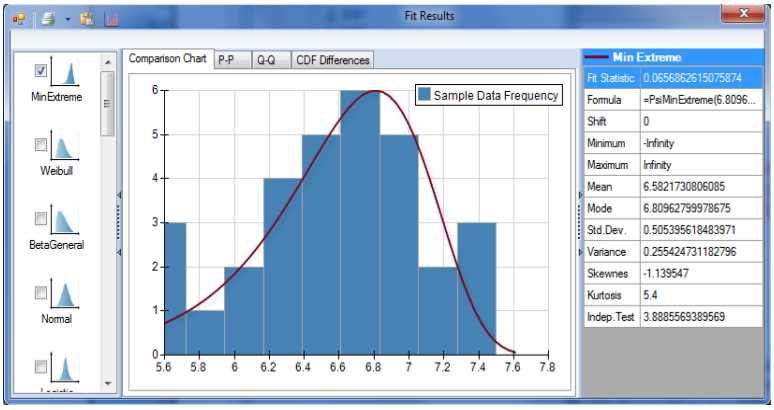


3. Make any changes to the default settings on the Fit Options dialog

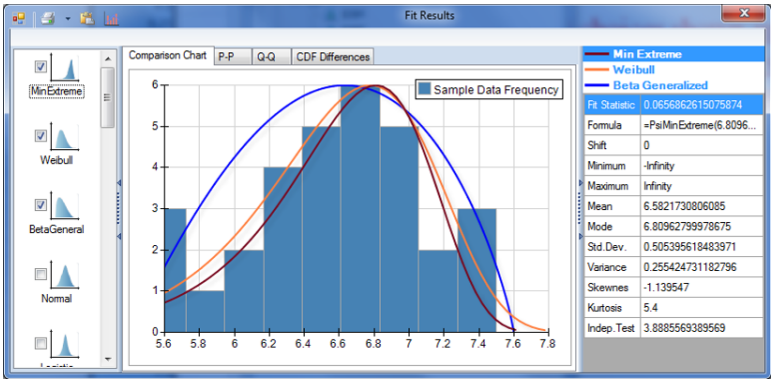


Tip: Try fitting with "Allow Shifted Distributions" selected and again unselected to see which result works better for your data.

4. Review and accept or modify the results



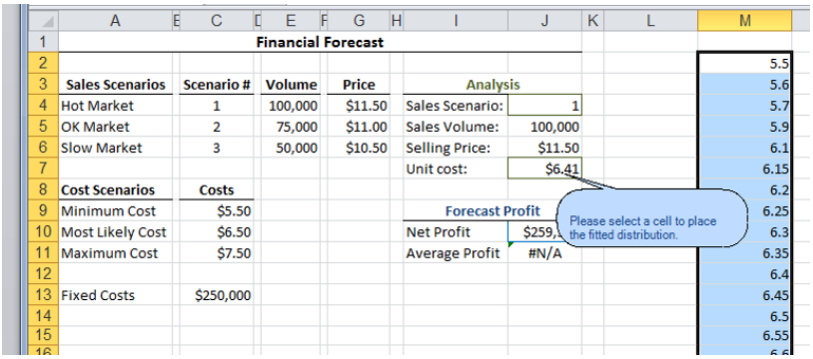
On the left of the dialog is a list of fit choices shown in order of decreasing Goodness of Fit based on the test you chose. You can select multiple choices to see how they compare and choose the one you are most happy with. See the example shown on the next page.



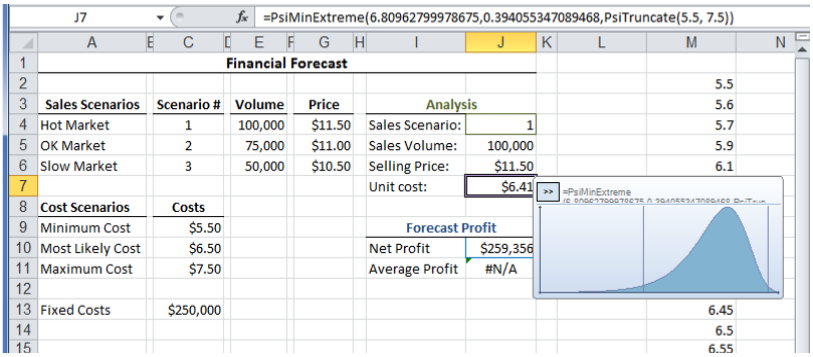
You can also click on the P-P, Q-Q, CDF Differences tabs to get additional perspective on which distribution best fits.

Tip: Once you have accepted the distribution you can open it back up and modify the lower and upper bounds, shift, etc.

5. Place the resulting distribution by clicking "X" at the top right of the menu, clicking on "Yes" in the resulting dialog to accept, and choosing where you want to place it in Excel.



You can see the new distribution where you placed it by clicking the cell. It will show up in the formula bar and if you leave the mouse over that cell a pop-up will show your distribution.



More Hints and Warnings

Using a Triangular Distribution

If you have only estimates of the minimum, maximum, and most likely values of an uncertain variable – and no other past data or literature references – you can create a **PsiTriangular** distribution from these three numbers. This is *unlikely* to be a highly accurate representation of the uncertainty, but it will allow you to get started, and it is *far* better than a single average value. If your 'minimum' and 'maximum' values are really low- and high-percentile *estimates* rather than the absolute lowest and highest values that can occur, consider using the **PsiTriangGen** distribution instead.

Define Each Uncertain Variable Only Once

Often, you'll need to use the same uncertain variable in several different formulas in your model. A very common error is to enter the same distribution

function, with the same parameters (say PsiNormal(100, 10)), *several times* in a model – in a belief that these instances will yield the same results on each trial. This is *incorrect* – by doing this, you've actually defined *several independent* uncertain variables that may well sample *different values* on each trial. You should instead enter =PsiNormal(100, 10) in a cell such as A1, and use A1 in each cell formula where the variable is needed.

Compute Statistics Only on Final Outputs

Another common error is to use formulas in your model to compute a statistic across two or more uncertain variables, and then *use this statistic* in further calculations that are part of the same simulation model.

For example, suppose you have two uncertain variables, =PsiUniform(0,10) in cell A1 and =PsiUniform(0,20) in B1. You want to know the mean or average value of these two variables, across the trials of the simulation. The *wrong* way to do this is to put = (A1+B1)/2 or =AVERAGE(A1,B1) in cell C1. This takes the average of A1 and B1 *on each trial* – the result will be a distribution that tends to the Normal, with a different (larger) mean than you probably intended. The right way to do this is to put =PsiDisUniform(A1,B1) in cell C1. Now the distribution of cell C1 is "flat:" On each trial only one of A1 or B1 is selected.

David Vose, principal of Vose Consulting, offers his ***cardinal rule of risk analysis modeling***: "Every trial of a risk analysis model must be a scenario that could actually occur." To ensure that this is true, you may need to specify *correlations* between your uncertain variables, as described in the next section.

Dependence and Correlation

Unless you specify otherwise, Monte Carlo simulation assumes that each of your uncertain variables is *independent* of all the other uncertain variables. When two variables are independent, the value of one variable on a given trial conveys no information about the value of the other variable. When two variables are *dependent*, there is a statistical relationship between them: On average, the value of one variable on a given trial *does* convey information about the value of the other variable on that trial.

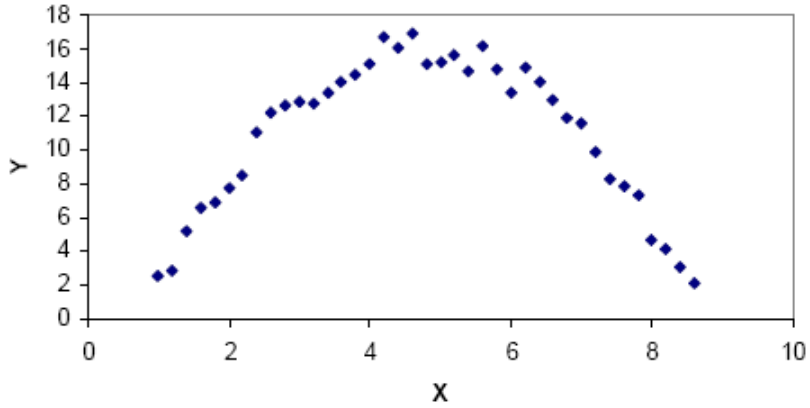
Measuring Observed Correlation

Correlation is a statistical measure of the degree to which one variable is related to another. When we observe that two variables are correlated, it may be that one variable is dependent on the other, or that both variables are dependent on a third variable, *or* that the correlation appeared by chance and there is no real dependence of one variable on the other.

The most common parametric measure of correlation is the **Pearson product moment correlation coefficient**. You can use the **PsiCorrelation** function to compute this correlation coefficient, across the trials of a simulation, for any pair of uncertain functions or variables in your model. The value of a product moment correlation coefficient can range from -1 to +1.

A correlation coefficient captures a simple *linear* relationship between two variables; it cannot capture all of the ways one variable may depend on another. A low or zero correlation coefficient between two uncertain variables or functions does *not* necessarily mean that the variables or functions are *independent* – the two variables or functions might have a strong relationship, but one that cannot be captured by a single correlation coefficient.

The figure below is a scatter plot of two variables X and Y. There is an obvious relationship between these two variables – when X is 1, Y is about 2, when X is 5, Y is about 15, when X is 9, Y is about 2, and so on – yet the Pearson product moment correlation coefficient between X and Y is zero.



More complex relationships among variables can be expressed in several ways; one very general way is by creating SLURPs (Stochastic Library Units, Relationships Preserved), as described below under "Probability Management Concepts."

Inducing Correlation Among Uncertain Variables

If you can compute a value for A1 with a formula based on an uncertain variable B1, then certainly A1 is dependent on B1. But in many modeling situations, both A1 and B1 are *uncertain* variables, you have no way to directly compute one from the other, yet A1 may *statistically depend* on B1. For example, mortgage interest rates depend on bond market interest rates (since mortgages are pooled and sold as securities in the bond market), and both rates depend on inflation expectations, but it is quite difficult to specify a formula that relates these variables.

In cases like these, you'll need to specify the statistical dependence between uncertain variables using PSI Property functions, supplied as arguments to PSI Distribution functions. This is called *inducing correlation* between uncertain variables, that would otherwise be considered independent. For example, you can write =PsiNormal (100, 10, **PsiCorrIndep("MyCorr")**) in cell B1 and write =PsiUniform(0, 100, **PsiCorrDepen("MyCorr", 0.9)**) in cell A1 to specify that A1 depends heavily on B1. "MyCorr" is an arbitrary string name.

The number 0.9 in the example above is a **Spearman rank order correlation coefficient**. This is a *nonparametric* measure of correlation that is computed from a rank ordering of the trial values drawn for both variables. It can be used to induce correlations between *any* two uncertain variables, whether they have the same or different analytic distributions, or even custom distributions. Like the product moment correlation coefficient, its value can range from -1 to +1.

Meaning of Rank Correlation Coefficients

- A correlation coefficient of +1 forces the sampled values of the uncertain variables to be exactly positively correlated, meaning that the p^{th} percentile value from one distribution will be sampled on the same trial as the p^{th}

percentile value from the other distribution. Coefficients from 0 to +1 will produce varying degrees of positive correlation.

- A correlation coefficient of -1 forces the sampled values of the uncertain variables to be exactly negatively correlated, meaning that the p^{th} percentile value from one distribution will be sampled on the same trial as the (100-p^{th}) percentile value from the other distribution. Coefficients from 0 to -1 will produce varying degrees of negative correlation.
- A correlation coefficient of 0 means there is no induced relationship between the variables. In practice, one usually uses coefficients less than +1 or -1, and uses 0 only in a correlation matrix that defines relationships among several variables (see below).

Computing Rank Correlations from Sample Data

If you have sufficient data on the past performance of two uncertain variables where the observations occurred at the same time for each variable, you can compute the Spearman rank correlation coefficient *r* as follows, where ΔR is the difference between the ranks of corresponding observations of the two variables:

$$r = 1 - \frac{6\sum_{i=1}^{n} \Delta R_i^2}{n\left(n^2 - 1\right)}$$

$$\Delta R_i = \Delta R_{i1} - \Delta R_{i2}$$

ΔR_i is the difference between the ranks of observation i for the two variables

ΔR_{ij} is the rank of observation i for variable j, $j = 1, 2$

You can easily compute this value on a spreadsheet, using Excel's RANK function, which returns the rank of a given cell value within a range of values. Suppose that A1:A100 contains trial values for uncertain variable A, and B1:B100 contains trial values for uncertain variable B. In C1 enter the formula =RANK(A1, $A$1:$A$100,1), in D1 enter =RANK(B1,$B$1:$B$100,1), and in E1 enter the formula =(C1-D1)^2. Then copy C1:E1 down to row 100. Then the rank order correlation coefficient is =1-(6*SUM(E1:E100)/(100*(100^2-1)).

If you don't have data on past performance of the uncertain variables, you will have to use judgment to estimate rank correlation coefficients.

Using a Rank Correlation Matrix for Several Variables

What if you have three, four, or more uncertain variables that should all be correlated with each other? You can create a small table or matrix of rank correlation coefficients in a cell range on the worksheet, and use this cell range in the PSI Property function **PsiCorrMatrix**. Below is an example of a 3x3 correlation matrix:

	A	B	C
1	1	0.8	0.5
2	0.8	1	0.2
3	0.5	0.2	1

You pass PsiCorrMatrix (*matrix cell range*, *position*) as an argument to the PSI Distribution function, for example =PsiNormal (10,5,PsiCorrMatrix(A1:C3,1)) for the first uncertain variable covered by the correlation matrix. You'd pass PsiCorrMatrix(A1:C3,2) to the PSI Distribution function for the second variable, and PsiCorrMatrix(A1:C3,3) for the third. This specifies that the first variable has a rank correlation coefficient of 0.8 with the second variable, and 0.5 with the third variable. The second and third variables are correlated with each other, with a rank correlation coefficient of 0.2.

Note that a correlation matrix must always have 1's on the diagonal, because an uncertain variable is always perfectly correlated with itself. Also, the matrix must be symmetric: If row 2, column 1 contains 0.8, then row 1, column 2 must also contain 0.8. Finally, the correlation coefficients must be consistent with each other: For example, if uncertain variable 1 is strongly positively correlated with variable 2, and variable 2 is strongly positively correlated with variable 3, then variable 1 cannot be negatively correlated with variable 3. Formally, the matrix must be *positive semidefinite* – it cannot have any negative eigenvalues. Risk Solver Platform tests for this condition, and displays #N/A in the PSI Distribution cells for the uncertain variables that refer to an inconsistent correlation matrix.

To learn more about rank correlation and its uses in Monte Carlo simulation, consult the book *Risk Analysis: A Quantitative Guide* by David Vose, mentioned in the Introduction.

Using the Correlations Dialog

The Correlation dialog offers an easy way to create, edit, and remove correlation matrices in your model. It appears when you click the Correlations button on the Ribbon.

You use a correlation matrix to *induce* a statistical correlation among two or more uncertain variables. When you do this, the trial values of these variables in a Monte Carlo simulation will tend to be drawn from related percentiles of their distributions on each trial. The numbers in a correlation matrix are **Spearman rank order correlation coefficients.** For more information on correlation matrices, see "Inducing Correlation Among Uncertain Variables" in the Dependence and Correlation section above.

A correlation matrix is stored in a contiguous cell range, with an equal number of columns and rows. To correlate *N* variables together, you need a matrix of *N* columns and *N* rows. Here's an example of a 3x3 correlation matrix:

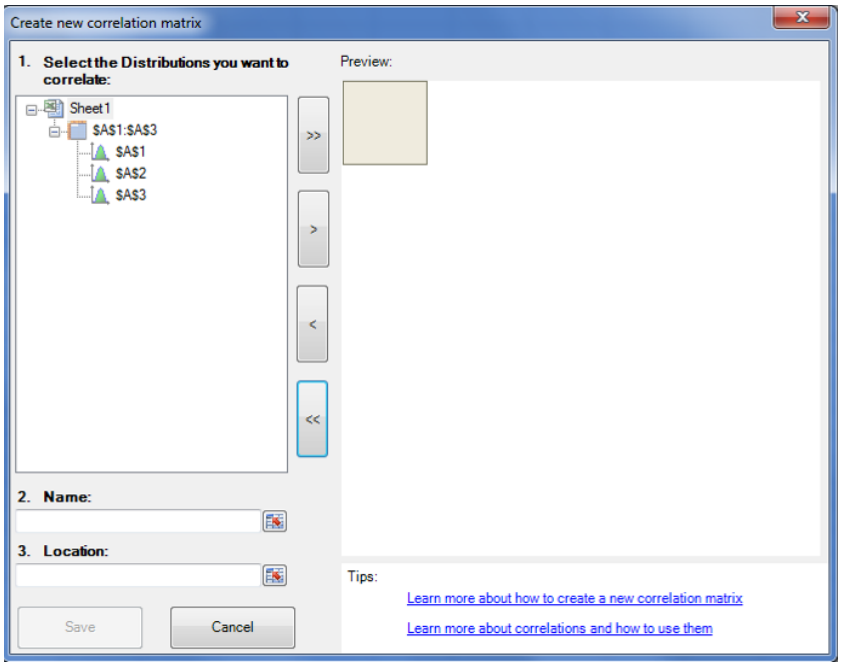
1	0.8	0.5
0.8	1	0.2
0.5	0.2	1

You can type in values for a correlation matrix directly on the Excel worksheet, and then reference this matrix in the PsiCorrMatrix() property function, passed as an argument to PSI Distribution functions for each of the correlated variables. But it is easier to use the Correlation Dialog to create the matrix for you.

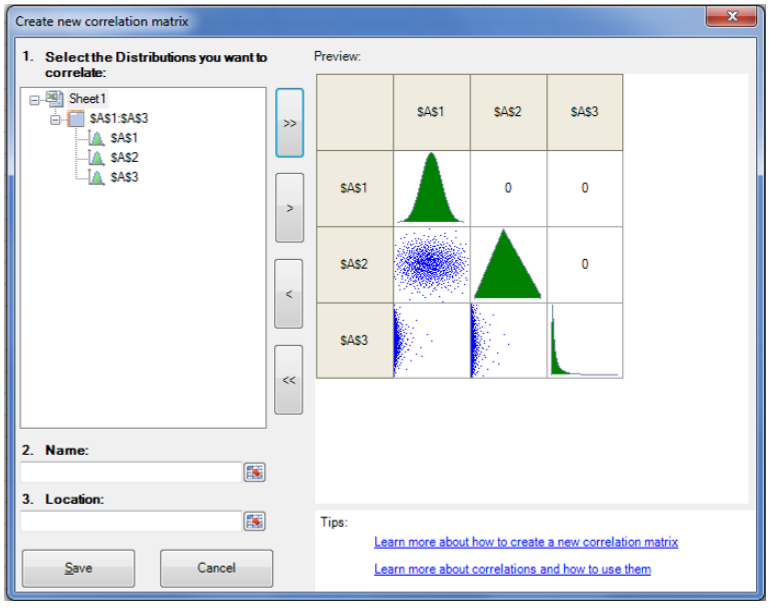
Creating a Correlation Matrix

When you first click the Correlations button on the Ribbon, the Correlation dialog appears with an empty matrix area, and a left pane (similar to the Task

Pane displayed by the Model button) that lists the uncertain variables in the model:



You can either choose the specific cells for the uncertain variables you want to correlate (press Ctrl or Shift and click for more than one) and then click the ">" button to include them in the matrix, or your can click the ">>" button to include all of them. The initial matrix with correlation values set to zero will display similar to the example below:



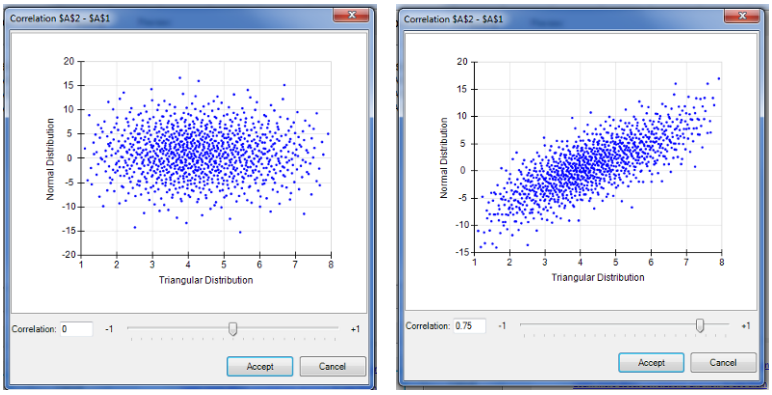
The preview for the correlation Matrix has three parts:

1. Black numerical values for each correlation in the top right of the matrix. In this case we have three values initially set with a zero correlation which you can edit. Since correlation matrices have to be symmetrical (the correlation

between A1 and A2 must be the same as the correlation between A2 and A1) we only show one value.

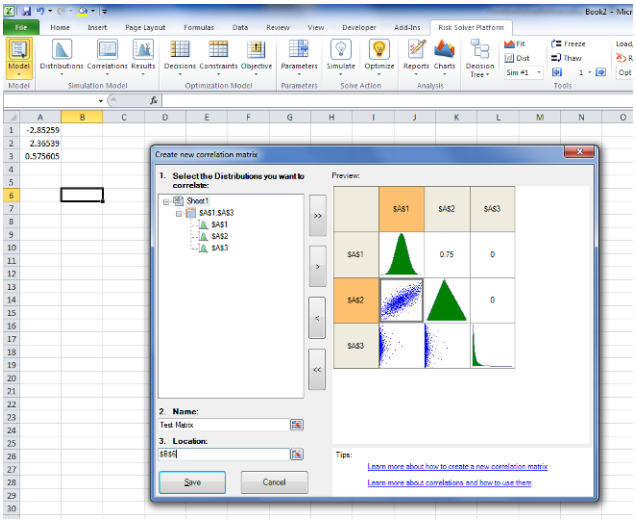
2. Green graphical representations of each distribution diagonally from top left to bottom right. In this example, we used a Normal distribution in our worksheet in cell A1 (show here in the top left part of the matrix), a Triangular distribution in cell A2, and a LogNormal distribution in cell A3.

3. Blue scatterplots reflecting the current correlations between the distributions. When you edit the values of each correlation, these will automatically update giving you a visual representation of the updated correlation. Note, you can click on any scatterplot here and a dialog will come up where you can more clearly see the scatterplot and how the scatterplot changes as you adjust the correlation (see below). In the example on the left the correlation is set to zero while on the right we have adjusted it (either directly typing in 0.75 into the Correlation field or adjusting the slider:

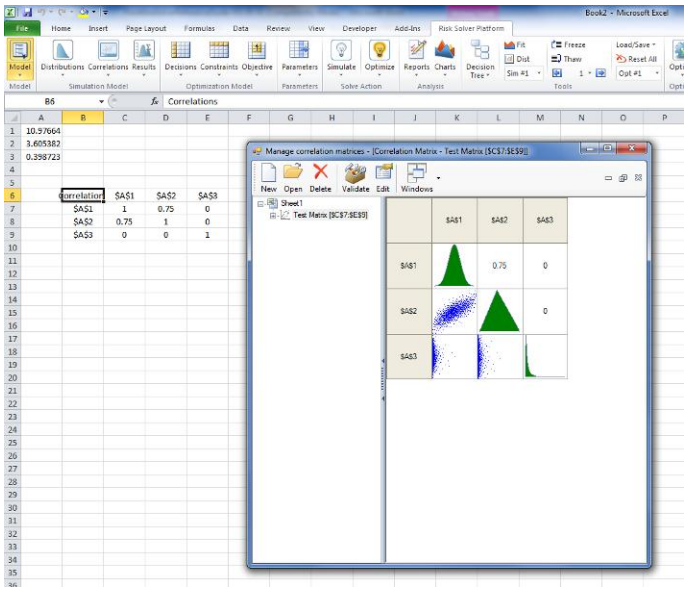


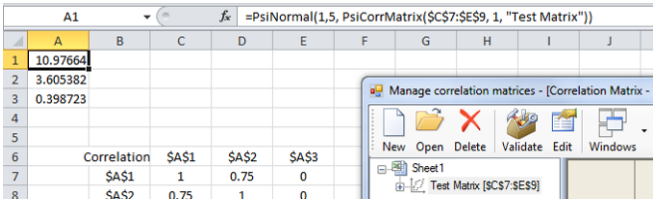
Once you click "Accept" you are taken back to the initial dialog and it shows the updated correlation matrix.

At this point you can name your correlation matrix and then choose the location you want to place it in your worksheet. While optional, naming your matrix is recommended, and is important if you want to use the same matrix several times. To place your matrix, simply click your cursor in the location field and click where you want to place the matrix on your worksheet.



Lastly, click "Save" to complete the process. Your new correlation matrix will then be saved and placed on your worksheet (in this case in cell B6) and the Manage Correlation Matrices Dialog will show in place of the Create New Correlation Matrix dialog. This dialog lists all the matrices you have set up on the left side of the dialog, and when you double-click on a particular matrix it will show on the right side.





As the matrix is created, Risk Solver Pro or Risk Solver Platform also edits the PSI Distribution function formulas for the uncertain variables (at A1, A2, & A3 in the example above) to include the property function **PsiCorrMatrix(C7:E9, *n*)**, where *n* is 1 for A1, 2 for A2, and 3 for A3. During a Monte Carlo simulation, when trial values are drawn for the uncertain variables A1, A2 and A3, Risk Solver Platform (or Pro) will use the correlation matrix at C7:E9 to adjust the 'draws' so that the values are properly correlated across all the simulation trials.

The matrix C7:E9 now appears in the left pane of the Manage Correlation Matrices dialog. If you close and later reopen the Correlation dialog, this matrix will reappear in the lower left pane. If you create a correlation matrix 'by hand' on the Excel worksheet, and insert the PsiCorrMatrix() calls yourself into the PSI Distribution function calls for the variables, the matrix you create will also appear in this left pane the next time you open the Correlation dialog. You can double-click on each matrix to display and edit that matrix.

Removing a Correlation Matrix

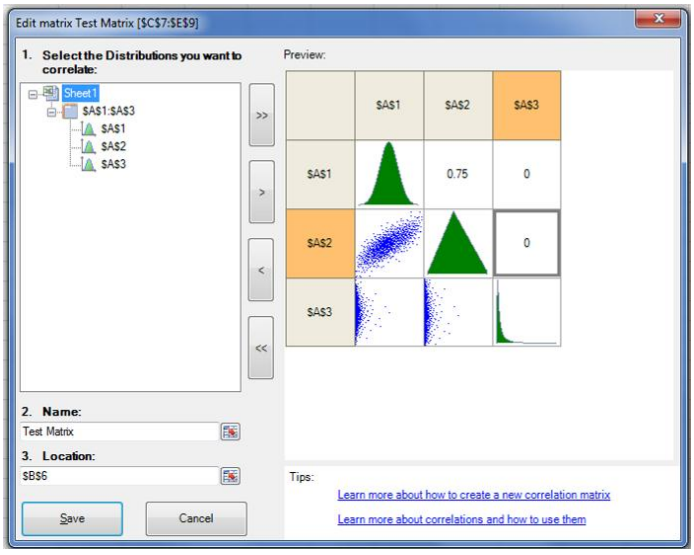
To remove a correlation matrix from your model, in the Manage Correlation Matrices dialog click on the matrix you want to remove and then click on the "Delete" icon in the dialog's ribbon menu. The matrix will disappear from the Correlations dialog, and the PsiCorrMatrix() property function formula referencing this matrix will be removed from the PSI Distribution function calls for the corresponding uncertain variables. However, the cell range on the worksheet where the matrix elements were written is left undisturbed – you can either delete them in Excel worksheet mode, or re-use them later.

Editing a Correlation Matrix

As shown in the both the Create Correlations and Manage Correlations dialogs, the value of each correlation is numerically shown in the top right portion of the matrix. To edit the value of a correlation you can simply click on the correlation you wish to edit and enter a new value from -1 (exactly negatively correlated) to +1 (exactly positively correlated). The default value of 0 means the two variables have zero correlation.

When you are finished, click the "X" button in the top right of the dialog to close it and your change(s) will be saved. Importantly, we will automatically check to make sure your matrices are valid (Positive Semi-Definite; see "Making a Matrix Consistent" below) and if they are not will offer help to correct them.

Alternatively, you can click on the Edit button on the Manage Correlations Ribbon to go to the Edit Matrix Dialog. The Edit Matrix dialog (see below) will appear, which looks very similar to the Create Matrix dialog. At this point you can change the name and location of the matrix, add or remove uncertain variables, and change correlation values. When done, simply hit **Save**.

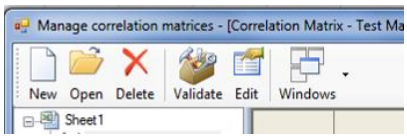


Making a Matrix Consistent

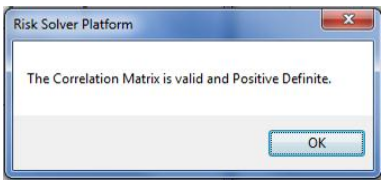
A correlation matrix must be not just symmetrical, but ***consistent***. This means for example that if variable A has a high positive correlation with variable B,

and B has a positive correlation with C, then variable A cannot have a high negative correlation with C. In mathematical terms, the correlation matrix must be *positive semidefinite*. This property depends on all of the elements of the matrix. If the matrix you create by hand does not satisfy this property (the example matrix above does not), Risk Solver Platform can automatically adjust the matrix elements so that the matrix is positive semidefinite.

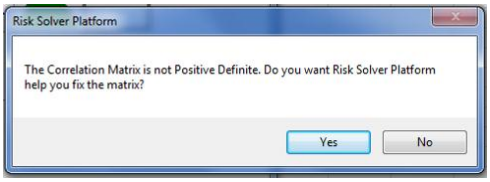
To check whether the matrix you've entered is positive semidefinite, in the Manage Correlations dialog simply click the Validate icon in the dialog's ribbon menu:



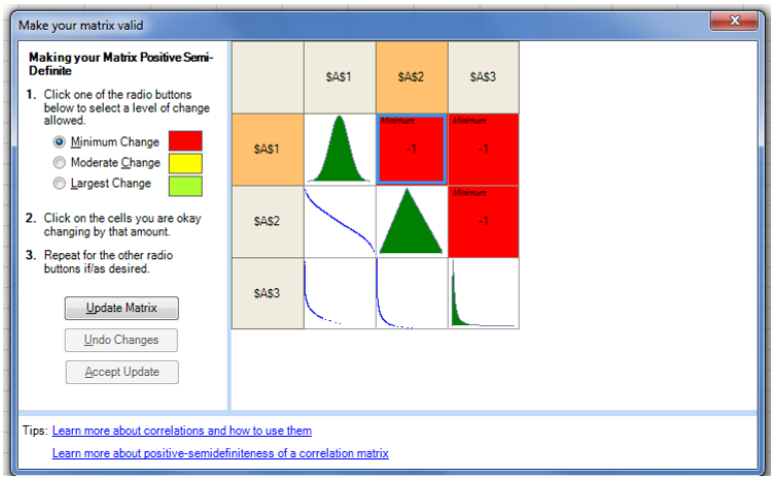
In our example, the matrix is valid so the following message is displayed and no action is needed:



If, for example, we set all our correlations to be -1, we would have a matrix which could not be valid and you will see the following message instead:



If you click No, the matrix remains as-is. If you click Yes, a new dialog appears:



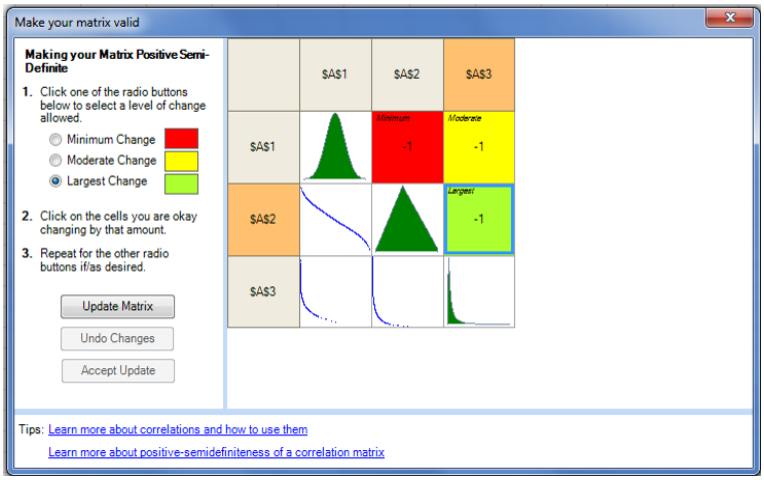
In this dialog, we can simply click the Update Matrix button to allow Risk Solver Pro or Platform to adjust the matrix to be consistent (positive

semidefinite). This may involve changing all of the elements of the matrix, by various amounts. But Risk Solver Pro and Platform can do more than this.

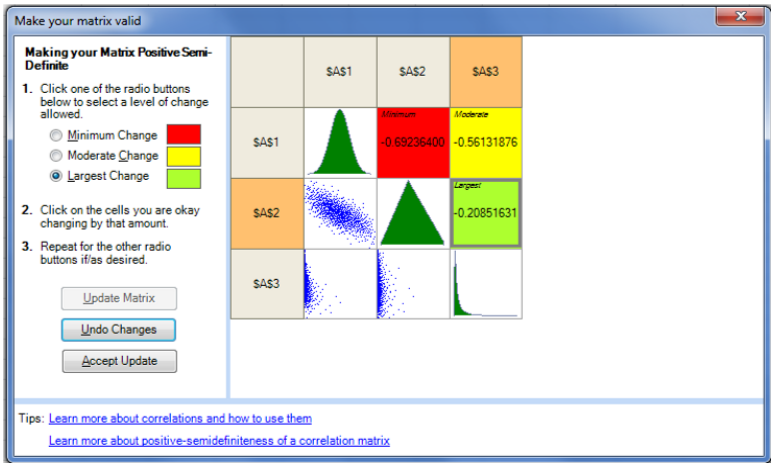
In many applications, you can determine 'good' values for correlations between certain pairs of variables, but for other pairs you have no special information. Risk Solver can adjust the matrix to make it positive semidefinite while minimizing changes to the matrix elements you care about, and making greater changes in the elements you don't care about.

In the example above, let's say we feel very confident there is a strong negative correlation between A1 and A2 but are less sure about A1 and A3 and very uncertain about any correlation between A2 and A3. We can click first on the radio button on the left relating to the amount of change we are comfortable with and then click on the cell(s) we want changed by that general amount.

Using the information above we would click on the "Moderate Change" radio button and then click on the correlation for A1 and A3 and then click on the "Largest Change" radio button followed by clicking on the correlation for A2 and A3. Our Matrix will now look like the following:



At this point we just click the "Update Matrix" button and our matrix is updated to be consistent (Positive Semi-Definite) with new correlation values as shown below:



We can now accept the update or undo the changes. We can of course further refine our correlation values if/as we wish.

Probability Management Concepts

In the February and April 2006 issues of the publication *OR/MS Today*, Dr. Sam Savage and coauthors Stefan Scholtes (University of Cambridge) and Daniel Zweidler (Shell Global Exploration) advance a series of ideas called "Probability Management." They argue for much broader use of risk analysis methods in planning and management, especially in large enterprises, based on three ideas:

- Interactive Simulation
- Stochastic Libraries
- Certification Authority

Probability Management calls for an up-front investment in the creation of standardized Certified Distributions, created or reviewed by an expert authority, and distributed to simulation modelers, often in the form of Stochastic Libraries. The payoffs of Probability Management include:

- Much easier creation of new simulation models
- Ability to capture complex dependencies – beyond what's possible with correlation coefficients
- 'Apples-to-apples' comparison of simulation model results
- Valid 'roll-ups' of simulation models from different groups

Risk Solver Platform is the first product designed to support the concepts of Probability Management. In addition to best-in-class support for *Interactive Simulation*, it provides direct support for creating, 'publishing' and using *Certified Distributions* and *Stochastic Libraries*, as discussed in this chapter.

Analytic Distributions

As described above, and documented in the Reference Guide, Risk Solver Platform provides a wide range of analytic probability distributions – more than 40 distributions are available, from **PsiBernoulli()** to **PsiWeibull()**. Given knowledge and experience, you can choose and use these distributions in your simulation models. But as discussed above, this is often the most challenging part of creating a Monte Carlo simulation model, especially for users with spreadsheet modeling expertise, but limited math and statistics background.

Moreover, when different end user modelers choose their own analytic distributions and parameters, they will often create models that cannot be compared or combined ('rolled up'), because the relationships between uncertain variables and functions would not be preserved. In large firms especially, where different business units must take into account common uncertainties, a better approach is needed.

Certified Distributions and Stochastic Libraries

A Certified Distribution is a custom probability distribution, created and/or reviewed by an expert, that is made available to end user modelers as a 'prepackaged unit.' These modelers need only the *name* of the Certified Distribution; they need not choose, or even be aware of, its analytic form, parameters or correlations.

An analytic Certified Distribution can be given its own random number seed, which overrides a user-specified seed for the model. This ensures that end user

modelers employing the Certified Distribution will use the same *samples* (sequence of Monte Carlo trials) each time they run a simulation – as long as they use the same simulation software package and version.

Given the capacity of modern computers and networks, an even better idea is to create and distribute Certified Distributions in the form of **Stochastic Libraries**. Such libraries contain *pre-generated trial data* for a group of (potentially statistically dependent) distributions. The sequence of trials is predetermined, so that if two or more end users develop and run simulation models using the same Stochastic Library, their model results can be compared and combined, on a trial-by-trial basis if necessary.

Risk Solver Platform supports the use of both analytic distributions and Stochastic Libraries as Certified Distributions. But since they contain pre-generated trial data, the use of Stochastic Libraries can *guarantee* that models will be run with exactly the trial data intended, regardless of the software (or version thereof) installed on end user modelers' PCs – as long as it accepts Stochastic Libraries.

Publishing and Using Certified Distributions

Certified Distributions can be prepared and tested using a variety of tools. One of these tools is Risk Solver Platform itself, as illustrated below. When creating a Stochastic Library, any good Monte Carlo simulation software package can be used, inside or outside of Excel, as long as it offers a way to save all the Monte Carlo trials. Once prepared, Certified Distributions should be approved and 'published' in a form where they can be made available to end user modelers.

Risk Solver Platform provides a property function **PsiCertify()** that you can use to name and "certify" a distribution as ready for publication. It also provides a distribution function **PsiCertified()** that end user modelers can use to access the Certified Distribution using only its name.

Certified Distributions can be physically distributed or made accessible in a variety of ways: On CDs or DVDs, via a network file share, or via email or Web download. The data for Monte Carlo trials may be easily stored in a relational database or a multidimensional data warehouse.

Risk Solver Platform, with its *Interactive Simulation* facilities, is the ideal 'client' tool for end user modelers using Certified Distributions. It provides an easy way to access Certified Distributions by *name*, using Microsoft Excel as a "universal data access client" to access distributions and Stochastic Libraries.

Risk Solver Platform makes it easy for an expert or CPO to create an Excel workbook, separate from any end user's simulation model workbook, where Certified Distributions are defined. This may be a **standard workbook** or an Excel **add-in workbook**; the latter provides certain advantages, since it is normally hidden from display and loaded automatically when Excel starts.

For example, it is straightforward to create an Excel add-in workbook that defines a Certified Distribution that draws its trial data from a Stochastic Library (see below) that is actually stored in a relational database or data warehouse. When the add-in workbook is (automatically) opened, either Excel data access functions or VBA code can be used to query the database for the trial data.

Stochastic Libraries: SIPs and SLURPs

As described by Dr. Savage in *OR/MS Today*, the simplest element of a coherent Stochastic Library is a *Stochastic Information Packet* or SIP, which is just a list of trial values for a single uncertain variable. An example might be 1,000 sample values, in a specific sequence, for the future price of oil at some point or interval of time. In a Monte Carlo simulation, trials will be drawn from the SIP ***in the order in which they were generated***.

If the SIPs for several different uncertain variables are generated as a group, in a manner that preserves the statistical dependence between them, they may be combined into a Stochastic Library Unit, Relationships Preserved, or SLURP. A SLURP is a table of trial values, where each column represents a specific uncertain variable, and each row represents a distinct trial. The SLURP's uncertain variables may be dependent in ways not measured by traditional correlation. But if a Monte Carlo simulation draws trials in the specific order given by the SLURP, this dependence will be reflected in the simulation model results.

On an Excel spreadsheet, a SIP is naturally represented by a column of cell values, and a SLURP is most easily represented by a two-dimensional table of cell values. As noted above, the data for this table of cell values may be easily drawn from a relational database or multidimensional data warehouse, or it may be created and maintained directly in Excel.

Risk Solver Platform provides two PSI Distribution functions, **PsiSip()** and **PsiSlurp()**, that make it easy to define distributions using SIPs and SLURPs. PsiSip() takes one argument: a cell range (normally a column) containing the trial values for one uncertain variable. PsiSlurp() takes two arguments: a two-dimensional cell range containing the SLURP data, and a column index (starting from 1) for the uncertain variable whose trials should be returned by the PsiSlurp() function. You'll want to ensure that the number of rows in the first argument range is at least as large as the number of simulation trials to be run.

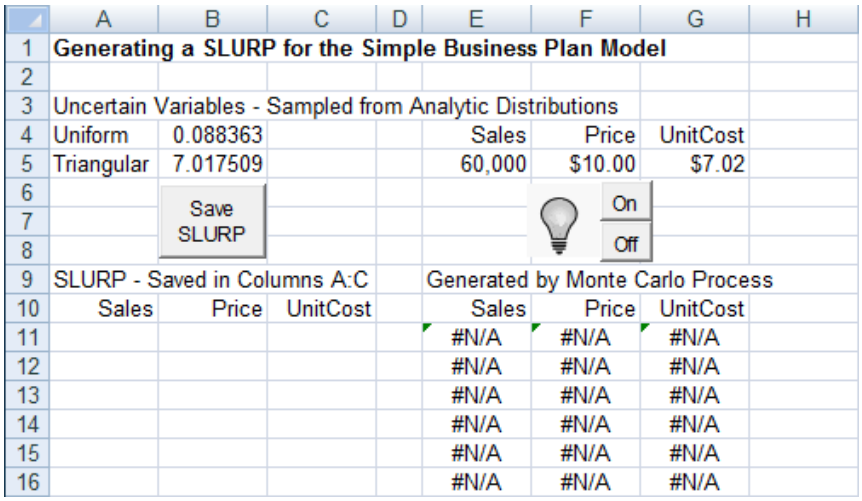
PsiSip() and PsiSlurp(), like other PSI Distribution functions, can be used directly in simulation models to return sample values from a SIP or SLURP. But they are especially powerful when you name and publish a Certified Distribution based on them. Doing this is as simple as including a **PsiCertify()** function call as an argument to PsiSip() or PsiSlurp() – as described below in the section "Creating and Using Certified Distributions."

Creating Stochastic Libraries

Of course, a Stochastic Library must first be created before it can be used. In some cases, you may have a data source from which you can draw SLURP data directly. For example, if you are working with demographic data such as age, family size, income, and taxes paid, you might be able to use a representative sample directly as a SLURP. But in many cases, an expert (perhaps like you) will have to select appropriate analytic probability distributions and their parameters, determine whether and how they should be correlated, and then generate the trial data through a Monte Carlo process.

Risk Solver Platform is a great tool for *creating* Stochastic Libraries, as well as using them. It supports a wide range of analytic distributions, shifting and truncation of distributions, and rank order correlation of different distributions. When you run a simulation, the trial data is generated, and with the PsiData() function, you can easily save the trial data in a column on the spreadsheet – this will create a SIP. Several PsiData() functions in adjacent spreadsheet columns

will create a SLURP. To see an example, open the simulation example workbook **BusinessPlanSLURP.xls**, pictured below.

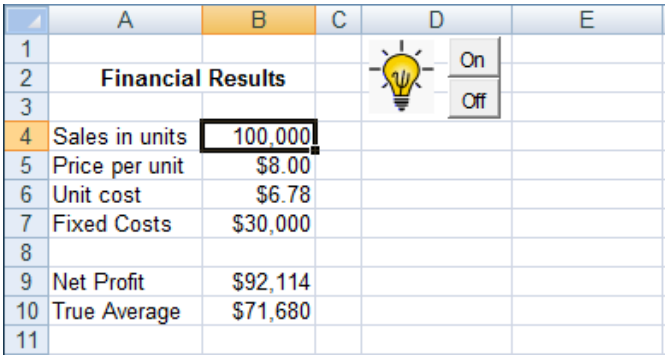


In this workbook, we generate SLURP data for the BusinessPlanPsi.xls model (also included as an example workbook). This model has three uncertain variables: Sales in units, Price per unit, and Unit cost. In cells B4 and B5 we have **PsiUniform()** and **PsiTriangular()** functions, and in cells E5, F5 and G5 we generate trial data as we did in the BusinessPlanPsi.xls model. For example, E5 contains the formula **=IF(B4>0.5,100000,60000)**.

In the cell range E11:E1010, we've array-entered the formula {=PsiData(E5)}. This will cause 1,000 trial values for E5 – a SIP for Sales in units – to be inserted in this cell range, each time a simulation is performed. Similarly, F11:F1010 contains {=PsiData(F5)}, and G11:G1010 contains {=PsiData(G5)}. A macro attached to the "Save SLURP" button performs an Edit Copy, Paste Values operation, copying the current data (numbers only) from columns E-F-G to columns A-B-C.

To create and save the SLURP, simply press the **On** button, then the **Off** button, then the **Save SLURP** button. The SLURP data will appear starting at cells A11, B11 and C11. Also note that cell range A11:A1010 has the defined name 'Sales', B11:B1010 is named 'Price', and C11:C1010 is named 'UnitCost'. BusinessPlanSLURP.xls should then be saved. We've generated 1,000 trials for each uncertain variable, enough to match the number of trials in our simulation.

We can test the use of this SLURP data in the **BusinessPlanSIP.xls** workbook (shown on the next page) which has **=PsiSip(BusinessPlanSLURP.xls!Sales)** at cell B4, =PsiSip(BusinessPlanSLURP.xls!Price) at cell B5, and **=PsiSip(BusinessPlanSLURP.xls!UnitCost)** at cell B6. Note that BusinessPlanSIP.xls is much simpler than BusinessPlanPsi.xls – it must deal only with the profit and loss calculations, not the probability distributions or their parameters.



	A	B	C	D	E
1					
2	**Financial Results**				
3					
4	Sales in units	100,000			
5	Price per unit	$8.00			
6	Unit cost	$6.78			
7	Fixed Costs	$30,000			
8					
9	Net Profit	$92,114			
10	True Average	$71,680			
11					

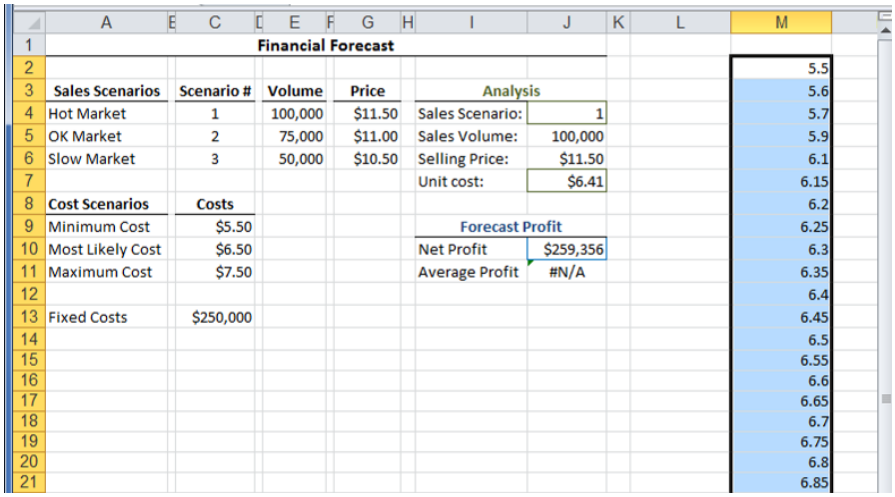
However, BusinessPlanSIP.xls does contain an Excel 'external reference' to BusinessPlanSLURP.xls, with a folder path, workbook filename, and cell range or defined name. This reference may 'break' if BusinessPlanSLURP.xls is moved to a different folder, or if the SLURP data must be reorganized in this workbook. Risk Solver Platform's support for publishing and using Certified Distributions provide a more flexible solution – as described in the next section.

Using the DIST Feature

Another way to use historical data is to use the DIST feature in Risk Solver Platform and Risk Solver Pro.

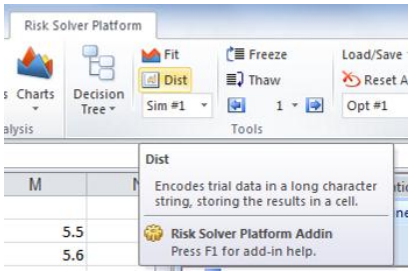
This tool allows you to encode your specific historical data as a distribution and then use the PsiSip function to sample that data during the simulation trials.

1. Select your data just as you did for the Fit process above.



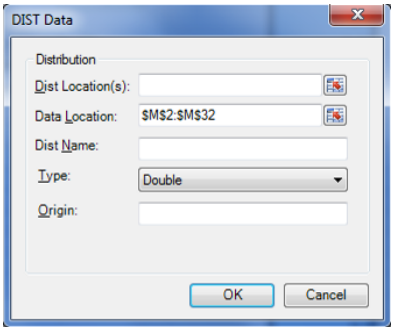
	A	C	E	G	I	J	M
1	**Financial Forecast**						
2							5.5
3	**Sales Scenarios**	**Scenario #**	**Volume**	**Price**	Analysis		5.6
4	Hot Market	1	100,000	$11.50	Sales Scenario:	1	5.7
5	OK Market	2	75,000	$11.00	Sales Volume:	100,000	5.9
6	Slow Market	3	50,000	$10.50	Selling Price:	$11.50	6.1
7					Unit cost:	$6.41	6.15
8	**Cost Scenarios**	**Costs**					6.2
9	Minimum Cost	$5.50			Forecast Profit		6.25
10	Most Likely Cost	$6.50			Net Profit	$259,356	6.3
11	Maximum Cost	$7.50			Average Profit	#N/A	6.35
12							6.4
13	Fixed Costs	$250,000					6.45
14							6.5
15							6.55
16							6.6
17							6.65
18							6.7
19							6.75
20							6.8
21							6.85

2. Click on the Dist button on the ribbon.



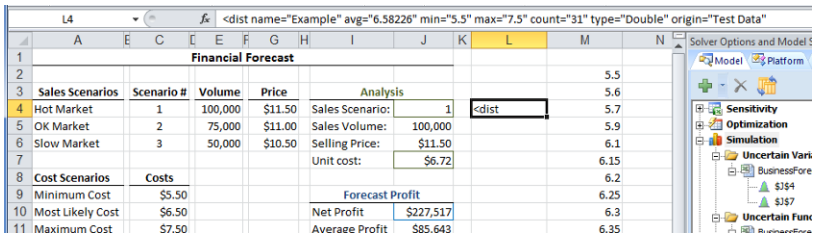
3. You will see a "Generate DIST from Data" Dialog. Click "OK".

4. The next dialog will allow you to choose where to place the DIST after it is generated as well as add information to it such as a name, comments on where the data came from, as well as the type of encoding.



Tip" The default encoding choice is "Double". If you have 0-10,000 data points you can use this setting. If you have 10,001 to 20,000 data points you should use "Single". If you have more than 20,000 data points you should use the Fit feature.

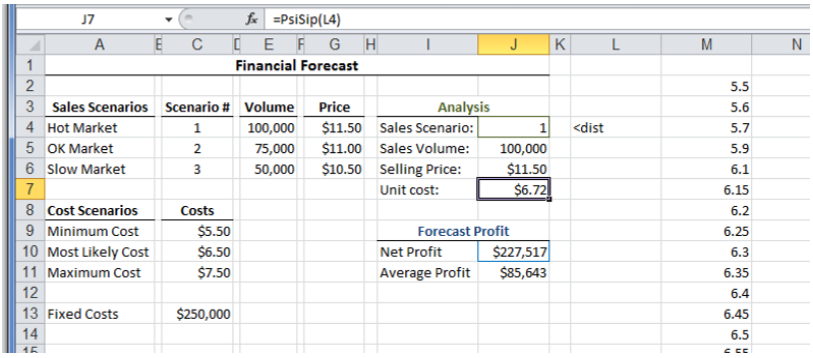
5. You can see the results after you hit "OK" below.



Note: In this case we chose cell L4 to place the data. You will see the DIST in the formula bar (only part of the DIST is visible above). The more data points you have the longer the DIST string will be.

Tip: The DIST you've created can now be copied and used in any spreadsheet. The data that created it no longer needs to be kept for it to work.

6. Use the PsiSip function to add the DIST to your model.



To use this DIST we enter "=PsiSip(L4)" in cell J7. Now our simulation model will use the DIST we just created.

Tip: When you run a simulation using a DIST you won't see the value in the cell containing it change as it will always show the first data point in your distribution and is working correctly.

Creating and Using Certified Distributions

As described above, Risk Solver Platform supports the use of both analytic distributions and Stochastic Libraries as Certified Distributions. It provides a property function **PsiCertify()** that you can use to name and certify a distribution as ready for publication. It also provides a distribution function **PsiCertified()** that end user modelers can use to access the Certified Distribution via its name.

Certified Distributions may be defined in an Excel workbook different from the end user modeler's workbook. This may be a **standard workbook** or an Excel **add-in workbook**; the latter provides certain advantages, since it is normally hidden from display and may be loaded automatically when Excel starts up.

***Note*:** The workbook that defines Certified Distributions must be a 'complete model' on its own: It must **define at least one output** by using a PSI Statistics function (PsiMean(), PsiOutput(), etc.) in a formula.

Publishing Distributions with PsiCertify

To name and certify an analytic distribution, you simply include the PsiCertify() property function in the formula that defines the distribution. For example:

=PsiNormal(0.1, 1, PsiTruncate(-2,2), PsiSeed(3), PsiCertify("MyDist",0.1))

will define a Certified Distribution named "MyDist" based on a Normal distribution with mean 0.1 and standard deviation 1, truncated so that its samples lie in the interval -2 to 2, and generated using a random number seed of 3 (this overrides any random seed specified in the end user's model). To use this distribution, the end user modeler enters **=PsiCertified("MyDist")** in his or her own workbook. When no simulation has been performed, the cell containing the PsiCertified() call will display the default value 0.1.

To name and certify a distribution based on trial data in a Stochastic Library, you also use the PsiCertify() property function in the formula defining the distribution. For example:

=PsiSip(A1:A1000, PsiCertify("MyDist",100))

will define a Certified Distribution named "MyDist" whose trials are drawn sequentially from the range A1:A1000, with default value 100 to be displayed when no simulation has been performed.

When (as is often the case) the SIPs for several different uncertain variables are generated as a group, in a manner that preserves the statistical dependence between them, they should be combined into a SLURP, i.e. a table of trial values where each column represents an uncertain variable, and each row represents a Monte Carlo trial. Risk Solver Platform allows you to place a character string name for each column (uncertain variable) in the first row of the SLURP data. You can then name and certify the entire SLURP, for example:

=PsiSlurp(A10:C1010, PsiCertify("MyPlan"))

where range A10:C10 contains labels such as "Sales", "Price" and "UnitCost", and A11:C1010 contains the SLURP trial data.

Now the end user modeler can write **=PsiCertified("MyPlan", "Sales")** to define an uncertain variable in a model whose samples will be sales figures.

The PsiCertify() function allows you to define more properties of the Certified Distribution than just its name. It is good practice to document the name, description, author, history of creation, and other relevant information about a Certified Distribution using PsiCertify(). The full set of arguments is:

=PsiCertify(***name, default value, short description, full description, version, author, copyright, trademark, history*****)**

Using Distributions with PsiCertified

To use a Certified Distribution, the end user modeler simply enters:

=PsiCertified(*name*) or **=PsiCertified(*slurp*,*sip*)**

in his or her own workbook, where *name* or *slurp* and *sip* are quoted strings. The user needs no knowledge of the form or parameters of the distribution, or of the name or folder path of the workbook that defines the distribution.

Risk Solver Platform scans the open workbooks (including regular workbooks and add-in workbooks) for distributions that include the **PsiCertify()** function, and adds the names found in PsiCertify() calls to the Certified dropdown gallery.

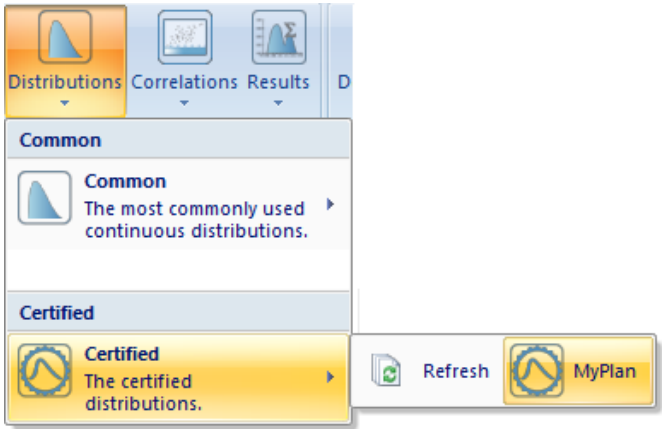
Certified Distributions may also be used with Risk Solver Engine, when the Risk Solver Platform GUI is not available. The PSI Intepreter matches calls to **=PsiCertified()** to the known Certified Distributions. If a named distribution is not found, the PsiCertified() call will return the error value #NAME?

To see an example of publishing and using Certified Distributions, open the simulation example workbook **BusinessPlanCertify.xls**, shown on the next page. This example uses a regular workbook, rather than an add-in.

	A	B	C	D	E	F	G	H
1	**Publishing Certified Distributions Based a SLURP Data for Business Plan**							
2	#VALUE!	A2 Contains =PsiSlurp(A10:C1010,PsiCertify("MyPlan"))						
3								
4	Uniform	0.188242			Sales	Price	UnitCost	
5	Triangular	7.849632			60,000	$10.00	$7.85	
6								
7		Save SLURP				On		
8						Off		
9	SLURP - Saved in Columns A:C				Generated by Monte Carlo Process			
10	Sales	Price	UnitCost		Sales	Price	UnitCost	
11	100,000	$8.00	$7.27		#N/A	#N/A	#N/A	
12	60,000	$10.00	$7.64		#N/A	#N/A	#N/A	
13	60,000	$10.00	$7.94		#N/A	#N/A	#N/A	
14	100,000	$8.00	$7.24		#N/A	#N/A	#N/A	
15	60,000	$10.00	$7.33		#N/A	#N/A	#N/A	

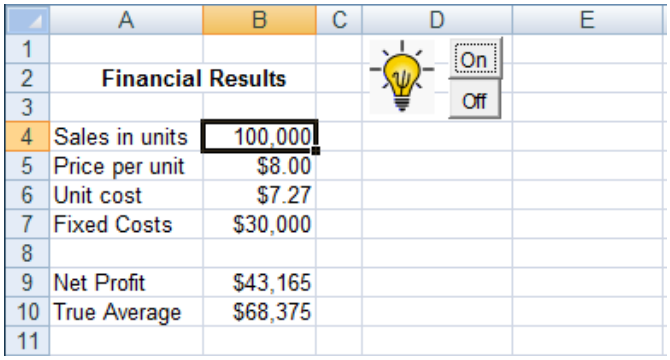
This workbook is identical to BusinessPlanSLURP.xls, except that the function call =PsiSlurp(A10:C1010,PsiCertify("MyPlan")) has been added in cell A2. A2 displays #VALUE! since there are labels rather than numbers in the first row of the SLURP range, but this doesn't affect use of the Certified Distribution.

With BusinessPlanCertify.xls open, if you click the **Certified** button in the Risk Solver Platform Ribbon, the Certified Distribution "MyPlan" will appear:



Simply clicking "MyPlan" will insert a function call =PsiCertified("MyPlan") into the currently selected cell. That's all there is to it.

To see how this set of Certified Distributions can be used, open the simulation example workbook **BusinessPlanCertified.xls**, shown on the next page.



This model is identical to BusinessPlanSIP.xls, except that it uses **=PsiCertified ("MyPlan", "Sales")** in cell B4, **=PsiCertified ("MyPlan", "Price")** in B5, and **=PsiCertified ("MyPlan", "UnitCost")** in B6. It no longer requires an Excel 'external reference' to a specific folder path, workbook filename, or cell range. In fact, there might not *exist* a single cell range containing fixed trial data – the trials might be loaded from an external database, or generated dynamically from one or more analytic distributions – as illustrated below.

Packaging Analytic Distributions

As described above, you can use PsiCertify() to name and publish an analytic distribution, with a specific form and parameters and a pre-specified random number seed, as a Certified Distribution. Further, you can "package" several analytic distributions that are correlated through a rank correlation matrix in the form of Certified Distributions. You can place these definitions in a standard workbook, or in an add-in workbook that is automatically loaded when Excel starts, hiding a good deal of complexity from end users who simply need to use these Certified Distributions.

Mastering Stochastic Optimization Concepts

Introduction

Risk Solver Platform offers you one unified framework for modeling and solving optimization problems, with or without uncertainty. Models may include 'normal' decision variables, constraints and an objective, plus uncertainties, recourse decision variables, chance constraints, and recourse constraints. This chapter seeks to explain this framework, starting from the basics of Solver models. It assumes only a limited knowledge of optimization – but if you've first read the chapter "Mastering Conventional Optimization Concepts," you'll gain a much deeper understanding from reading this chapter.

Our goal is to find a ***solution*** – values for the ***decision variables*** in our model – that satisfies the ***constraints*** and that maximizes or minimizes the ***objective***. In computing the objective and constraints, we use ***parameters*** of the model – for example quantities, distances, shipping costs, interest rates, etc.

Certain and Uncertain Parameters

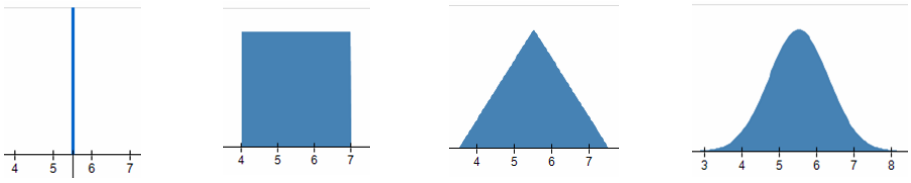
In conventional optimization, all the parameters are *assumed* to be numbers known with certainty. If estimation of a parameter involves uncertainty or noise, an average or other 'nominal' value is used. This is a simplification and *idealization* of the real world situation: In almost all business and engineering problems, there is at least some uncertainty in the values of the parameters.

In some models, the uncertainty or variation in the parameters may be small enough to be ignored. But in many models, it is large enough to be of concern. And in some models, the uncertainty is so large that it is a major feature of the problem, and much effort is (or should be) invested in modeling the uncertainty.

In the discussion to follow, by **uncertain parameter** we mean a parameter of your optimization model, such as an objective coefficient or constraint right hand side, that is subject to uncertainty. Such parameters might be *computed* from a "primitive uncertainty," such as a future interest rate or rainfall amount; we will call this an **uncertain variable**, and in Risk Solver Platform, we'll use the Uncertain Variable dialog to define and quantify this primitive uncertainty.

Realizations of Uncertain Parameters

Where a *certain* parameter has a *single*, known value, we can think of an *uncertain* parameter as having *many possible values*. We call each such value a *realization* of the uncertainty. Below, we've pictured one certain and three uncertain parameters.



An uncertain parameter might have an infinite number of possible realizations, but – as discussed near the end of this chapter – normally we can work with a finite ***sample*** of possible realizations. We just don't know *which one* of the sample values is the 'true' value, or closest to this true value.

Decision-Dependent Uncertainties

In many real-world problems, the uncertainties being modeled are ***independent of the decision variables***. For example, in a crop planning model where the acres to plant of different crops are decision variables, and the rainfall in a future period is an uncertainty, the uncertainty is almost surely independent of the decisions.

In other real-world problems, the uncertain parameters being modeled are ***dependent* on the decision variables** – they change if the decisions change. For example, in a market response model that includes competitor actions in a future period, where your product prices are decision variables, and your competitors' product prices are uncertainties, it is quite likely that the uncertainties will depend on the decisions.

In portfolio models where one is investing in publicly traded, large capitalization stocks, a decision to purchase some number of shares is unlikely to affect the uncertain future returns on those stocks. On the other hand, a large institution investing in thinly traded small-capitalization stocks may find that decisions to purchase a large number of shares may move the market price and impact the uncertain future returns. So both types of problems arise in practice. As we'll see later, the presence of ***decision-dependent uncertainties*** has a **huge impact** on how, and even whether, our model can be solved.

Resolving Uncertainty and Recourse Decisions

In some real-world problems, we are uncertain about the value of a parameter today, and **we will remain uncertain about its value in the future**; we have to make decisions in the presence of this uncertainty. This often arises in engineering design models, where there is uncertainty about the exact strength of a steel girder or the exact resistance or capacitance of an electronic component. The uncertainty may be *irreducible* – perhaps inherent in nature – or it may be *reducible* through better data collection, measurement or calibration – but these steps may be impractical for us.

In other real-world problems, we are uncertain about the value of a parameter today, but **we will know its value with certainty at some point in the future**. This often arises in marketing and finance models, where a competitor's price or next year's commodity prices or interest rates are uncertain today, but will become known at some future date. We say that the uncertainty will be ***resolved*** at the future date; its *many possible values* will be reduced to a *single* value, and the uncertain parameter will become certain.

If we are dealing with uncertainty that will not be resolved, we must find values of the decision variables that yield feasible (or *nearly* feasible) solutions for

many possible realizations of the uncertainty. On the other hand, if we are dealing with uncertainty that *will* be resolved at a future date, we can ask what decisions we must make 'here and now,' *before* the uncertainty becomes known, and what decisions we can make on a 'wait and see' basis, *after* the uncertainty is known. This leads to the concept of ***recourse decisions***, described below – and it has a **huge impact** on how our model should be formulated and solved.

Uncertainty and Conventional Optimization

The impact of uncertainty is *magnified* by conventional optimization. By its nature, optimization drives the decision variables to values that *just satisfy* certain constraints, while maximizing or minimizing the objective function. In doing so, optimization exploits any 'noise' or error, as well as the 'true value,' captured in the fixed numbers used in the model for uncertain parameters.

This effect is very evident in investment portfolio optimization, where the parameters of the model are estimated returns, variances, and covariances between securities. Practitioners often describe portfolio allocations from an optimizer as "unrealistic," and such allocations often do not perform well in practice. When the actual security returns, variances and covariances in future periods vary from the parameters used in the optimization, even by small amounts, the 'optimized' portfolio return often suffers by large amounts.

As shown by Ben-Tal and Nemirovski in a 2000 study of the NETLIB linear programming problems, "In real-world applications of linear programming, one cannot ignore the possibility that a small uncertainty in the data can make the usual optimal solution completely meaningless from practical viewpoint." Summarizing the effects of conventional investment portfolio optimization, Michaud wrote in a 1998 book "The result is that optimized portfolios are 'error maximized' and often have little, if any, investment value." ***Conventional optimization is not enough!***

Elements of Solver Models

We'll now describe all the elements of a Solver model with uncertainty. Again our goal is to find a ***solution*** – values for the ***decision variables*** in our model – that satisfies the ***constraints*** and that maximizes or minimizes the ***objective***. In computing the objective and constraints, we may use many ***parameters*** – some of which are computed from primitive ***uncertain variables***. In Excel, the decision variables and parameters are represented by worksheet cells; the objective and constraints are computed by formulas in other worksheet cells.

Uncertain Variables

A *certain* parameter is easy to model, in a worksheet cell containing a number. A primitive uncertainty requires a way to specify the range and 'shape' of the values it can assume. In Risk Solver Platform, we model primitive uncertainties as **uncertain variables** in worksheet cells that contain special add-in functions. For example:

=PsiUniform(0,1) – a uniform distribution ranging from 0 to 1

=PsiNormal(2,1.5) – a Normal distribution with mean 2, standard deviation 1.5

=PsiBeta(A1,B1) – a Beta distribution with shape parameters A1 and B1

=PsiSip(A1:A1000) – a 'Stochastic Information Packet' containing a range of user-supplied sample values, or realizations, for this uncertainty

Any of the 40-odd PSI Distribution functions that you can use for Monte Carlo simulation may also be used as uncertain variables for stochastic optimization.

You can think of the cell for an uncertain variable, containing a formula such as =PsiUniform(0,1), as holding an *array* of sample values, each one selected at random from the range 0 to 1 inclusive. Each value is one possible *realization* of the uncertainty. Risk Solver Platform can generate such an array of sample values automatically, using Monte Carlo simulation 'behind the scenes'.

Looking ahead to the section "Functions of the Variables and Uncertainties," note that any formula cell that *depends* on an uncertain variable cell must also compute an *array* of sample values. Each value corresponds to one possible *realization* of all the uncertain variables on which the formula depends.

In the example **PsiBeta(A1,B1)**, it is important to know whether A1 or B1 is dependent on the decision variables. A model that includes such a ***decision-dependent uncertainty*** is *much* harder to solve, and will require the methods of simulation optimization. Risk Solver Platform can automatically diagnose your model and determine whether any uncertain variables depend – possibly through a chain of formula cells – on the decision variables.

Decision Variables

Conventional optimization deals with only one type of decision variable, which represents a decision that must be made 'here and now,' irrespective of any uncertainty in the model. We call this a normal or first-stage variable.

If we are dealing with uncertainty that will be resolved in the future, then at some point the *array* of sample values for the uncertainty is effectively replaced by a *single* value, the *realization* of the uncertainty as it actually occurs.

If the situation we are modeling allows us to make certain decisions *after* the uncertainty becomes known, on a 'wait and see' basis, we can model these decisions with recourse variables, also called second-stage variables. (At the 'second stage,' the uncertainty has become known.)

Normal or First-Stage Variables

A **normal variable** for a 'here and now' decision is represented by a worksheet cell – say A1 – containing a number; the Solver will replace this number with an optimal value, when the problem is solved. (The term 'normal' here is unrelated to a Normal distribution, which is one way to define an uncertain variable.)

Recourse or Second-Stage Variables

A **recourse variable** for a 'wait and see' decision is also represented by a worksheet cell – say A2 – containing a number. But at the time the optimization is performed, a single recourse variable effectively represents an *array* of solution values, one for each *realization* of the uncertainties in the model.

Functions of the Variables and Uncertainties

Any formula that participates (in the objective and constraints, see below) in your model will normally depend on the decision variables and the parameters.

As a simple example, the objective of the Product Mix model EXAMPLE1in the **Examples.xls** workbook, is SUMPRODUCT(D17:F17,D9:F9) which can also be written as D17*D9 + E17*E9 + F17*F9. Here, D17:F17 are parameters (the selling prices of the three products), and D9:F9 are decision variables (the amounts of each product to build). In this model, the parameters are assumed to be *certain*.

An objective or constraint might be computed, from the parameters and the decision variables, through a chain of several formula cells. For example, in the Gas Company Chance and Recourse models in **StochasticExamples.xls**, illustrated in the chapter "Examples: Stochastic Optimization," the constraint at cell C23 is D15+D18-C8 >= 0, where D15 and D18 are decision variables (amounts of gas to purchase), C8 = 100+80*D5 (estimated demand next year), and D5 is an uncertain variable in the model.

Model Uncertainty and Optimization Methods

Recall that an uncertain variable effectively holds an *array* of sample values. For example, D5 =PsiUniform(0,1) represents an array of *realizations* of the uncertainty, each one selected at random from the range 0 to 1. If your objective or constraint (say C8 = 100+80*D5) depends on D5, it must also compute an *array* of sample values. Each value corresponds to one possible *realization* of all the uncertainties on which the formula depends.

Traditional Solver Engines cannot handle an objective or constraints that depend on uncertainty, and hence represent an array of sample values. To solve a problem with such an objective and constraints, we must choose one of three possible approaches:

1. Use a 'stochastic Solver Engine' that *can* handle an objective or constraints that depend on uncertainty. (Frontline is planning for such Solvers, but they aren't yet available in Version 11.)
2. *Transform* a model whose objective and/or constraints depend on uncertainty into a 'deterministic equivalent' or 'robust counterpart' model that may be much larger, but whose objective and constraints no longer depend on uncertainty. Then we can solve the transformed model using a traditional Solver Engine, and translate the results into a solution for the original model. To be transformed successfully, the original model must have only uncertainties that are *independent* of the decisions, and have an objective and constraints that are *linear* (or certain other) functions of the decision variables.
3. Ensure that all 'top-level' formula cells computing the objective and constraints represent only *single* values – statistical summaries of the arrays of sample values. We can do this manually, with formulas in top-level cells that call PSI Statistics functions such as PsiMean(), PsiVariance() or PsiPercentile() – or Risk Solver Platform can do this automatically and implicitly, when we define our objective and constraints as described in the next two sections. This choice is flexible, but it requires that we use simulation optimization – the most general, but also the slowest and least scalable optimization method.

In all these approaches, we must specify what it *means* to maximize or minimize an objective, or to satisfy a constraint that depends on uncertainty – implicitly or explicitly reducing an array of sample values to a single value. The next two sections explain how we can do this.

The Objective Function

The quantity you want to maximize or minimize is called the *objective function*. In Excel, the objective is calculated by a formula cell – say A1 – listed in the Task Pane Model tab or the Solver Parameters dialog, in the outline list box under **Objective**. This could be a calculated value for projected profits (to be maximized), or costs, risk, or error values (to be minimized).

If the objective function depends on uncertainties, we must specify how we want to 'optimize' this function. The most common practice is to maximize or minimize the *expected value* (informally, the mean value) of the objective, over all realizations of the uncertainties. In Risk Solver Platform, you do this by selecting the objective in the Task Pane Model tab, and in the lower Properties area, selecting **Expected** from the Type dropdown list in the Stochastic group.

Instead of maximizing or minimizing the *expected value* of a function of the decision variables and uncertainties, you can maximize or minimize a *measure of the uncertainty* in a function. Risk Solver Platform converts the objective to the form max *t* or min *t*, where *t* is a new variable, and the model includes a *chance constraint* A1 >= *t* or A1 <= *t* with the measure of uncertainty that you specify.

Implicit and Explicit Forms for the Objective

When you select Expected from the Type dropdown list in the Stochastic group, Risk Solver Platform treats the objective cell as implicitly containing **E**[*objective*], or using sample realizations of the uncertainty PsiMean(*objective*). To make this *explicit* on the worksheet, you can direct Risk Solver Platform to automatically create a **PsiObj()** function call with an "Expected" argument – using either the "Use Psi Functions to Define Model" option, or the Save Model button. You can also manually enter a **PsiMean()** function call in the objective cell. Similar options apply to the constraints, as discussed below.

Constraints: Normal, Chance, Recourse

Constraints are relations such as $f(x_1, x_2 \ldots, x_n) <= b$, where $x_1, x_2 \ldots, x_n$ are decision variables. In Excel, each x_i corresponds to a cell, $f(x_1, x_2 \ldots, x_n)$ is computed by a formula in another cell (say A1), the constant *b* is in another cell (say B1), and we enter **A1 <= B1** via the Ribbon or Task Pane. A constraint is *satisfied* when the relation it specifies is true within a small tolerance.

When your model includes uncertainty, we must examine how each constraint depends on the uncertainties and the decision variables:

- If a constraint depends only on *certain* parameters and *normal* decision variables, it is 'deterministic' and is handled in the usual way by the Solver. We call this a ***normal* constraint**.
- If a constraint depends on *uncertain variable*s and *normal* decision variables, we must specify what it *means* for the constraint to be satisfied. There are many possible *realizations* for the uncertain variables, but only single values for the decision variables. The Solver must find values for the decision variables that cause the constraint to be satisfied for *all*, or perhaps *most* but not all, realizations of the uncertainties. We call this a ***chance* constraint**. For example, we might specify that the constraint must be satisfied 95% or 99% of the time; it can be violated 5% or 1% of the time. For 95%, we denote such a constraint as **VaR $_{0.95}$ A1 <= B1**. But this form

may not be your best choice – alternatives called CVaR and USet are discussed in the section "More on Chance Constraints."

- If a constraint depends on *uncertain variable*s and *recourse* decision variables, then the Solver will find an *array* of values for each of the recourse variables, corresponding to the *realizations* of the uncertain variables. Recourse decisions give the Solver flexibility to satisfy constraints that involve uncertainty; but in effect, each such constraint has many *realizations* – one for each realization of the uncertainties. We call this a ***recourse* constraint**.
- A constraint may also depend on *recourse* decision variables, and possibly normal decision variables, but not depend on any uncertain variables. This is also a ***recourse* constraint**, with many *realizations*. The Solver must find values for the recourse variables that satisfy all the constraints where they appear – some with uncertainties, and some without.

Implicit and Explicit Forms for Constraints

When you select VaR from the dropdown list in the Constraint dialog and enter a Chance of 0.95, Risk Solver Platform treats the constraint cell as implicitly containing $\mathbf{VaR}_{0.95}$[*constraint*], or using sample realizations of the uncertainty PsiPercentile(*constraint*, 0.95) where *constraint* is the left hand side. To make this *explicit* on the worksheet, you can direct Risk Solver Platform to automatically create a **PsiCon()** function call with a "VaR" argument – using either the "Use Psi Functions to Define Model" option, or the Save Model button. You can also manually enter a **PsiPercentile()** function in the constraint left hand side cell. For background on the PsiObj() and PsiCon() functions, see "Defining Your Model with PSI Functions" in the chapter "Building Solver Models."

Multiple Uncertainties May Offset Each Other

What happens when a constraint depends on *several* different uncertainties? Is such a constraint harder or easier to satisfy than a constraint that depends on just one uncertainty?

In the simplest case, suppose we have a *linear* constraint, with coefficients a_i and decision variables x_i:

$$a_1x_1 + a_2x_2 + ... + a_nx_n \leq b$$

Suppose that each coefficient a_i is uncertain (and independent of all the others), with sample values drawn randomly from PsiUniform ($a_i - 0.5$, $a_i + 0.5$).

The average or *nominal* value of each coefficient is a_i. The 'worst' that can happen is that a sample is drawn where *every* coefficient is $a_i + 0.5$ – this makes the left hand side (LHS) as large as possible, so it is very likely to violate the condition LHS $\leq b$. *But this case is very unlikely to occur*.

In *most* realizations of the uncertainties, some coefficients (randomly drawn from the range $a_i - 0.5$ to $a_i + 0.5$) will be less than a_i, and some will be greater than a_i. The more uncertainties are involved, the greater the chance that some of them will draw samples less than a_i.

If we use a *chance constraint* to specify that the relation must be satisfied (say) 95% or 99% of the time, we actually have a *better* chance of satisfying this constraint when it depends on *many* uncertainties than when it depends on just one – as long as the uncertainties are independent, or at least not highly correlated with each other.

Solutions: Feasible, Optimal, Well-Hedged

A solution (set of values for the normal and recourse variables) for which all of the constraints in the Solver model are satisfied is called a *feasible solution.* For constraints that depend on uncertainties, this means that chance and recourse constraints meet their criteria for satisfaction over all the *realizations* of the uncertainties considered in the optimization.

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. When the objective function depends on uncertainties, this normally means that the *expected value* of the objective reaches a maximum (minimum) – but you can also use max *t* or min *t* with a chance constraint A1 >= *t* or A1 <= *t*, as described earlier.

The optimal solution to a problem with uncertainty (the 'stochastic solution') will never have an objective value that is better than one could obtain 'with 20/20 hindsight,' by solving a conventional optimization problem where the uncertainties are known. The difference between the objective value obtained 'in hindsight' and the optimal value obtained 'in foresight' is called the *expected value of perfect information* (EVPI).

The stochastic solution will often be (quite) different from the optimal solution to a problem where all uncertain variables are replaced with their nominal or average values. The latter optimization will exploit the 'point values' of the parameters, with no consideration for their variability; this will often yield a better objective value for the nominal problem, but the solution may not be feasible, let alone optimal, for many different realizations of the uncertain variables. Compared to the 'nominal' solution, the stochastic solution is more robust or 'well-hedged' against possible variations in the uncertain variables.

Solving for Recourse Variables

If we are dealing with uncertainty that *will* be resolved in the future, then at that future date, each uncertain variable will be replaced by a single value, the *realization* of the uncertainty that actually occurs. In Risk Solver Platform, you can enter these single values for uncertain variables in the Lock value field in the Uncertain Variable dialog, then choose to *solve for* single values of the *recourse* variables, given known single values for both the normal variables (determined by an earlier optimization) and for the uncertainties.

When Risk Solver Platform solves a problem by transforming the model, using the Deterministic Equivalent or the Robust Counterpart transformation, it computes an *array* of values for the recourse variables, one for each *realization* of the uncertainties considered during the optimization. Using the left and right Trial # arrows on the Ribbon, you can display on the worksheet the different sample values of the uncertain variable cells, and the corresponding values of the recourse variable cells – the 'wait and see' decisions in many possible scenarios. Now, the *actual* values of the uncertainties, once they become known, might be different from any one of these scenarios – so it is still useful to solve for single values of the recourse variables as described above.

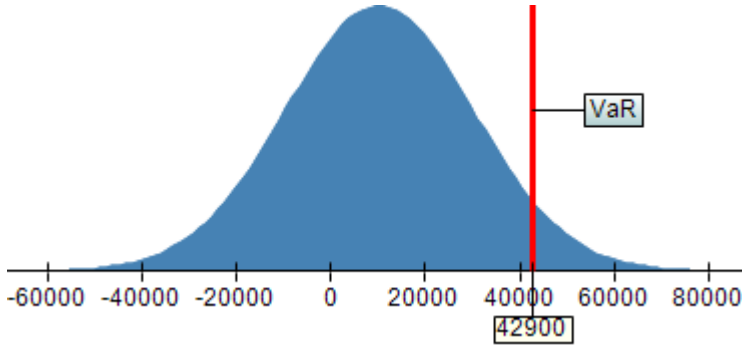
More on Chance Constraints

As explained above, if a constraint depends on *uncertain variable*s and *normal* decision variables, we can seek solution values for the variables that cause the constraints to be satisfied for *all*, or perhaps *most* but not all, realizations of the uncertainties. If we insist that the constraints are satisfied for *all* realizations,

we may not be able to *find* values for the decision variables that meet this requirement – and if we do, we will very likely 'pay for this' via worse values for the objective function.

Instead, we can seek solution values for the variables that cause the constraints to be satisfied for *most*, but not necessarily all, realizations of the uncertainties. We might specify that the constraint must be satisfied (it must not exceed a given limit) 95% or 99% of the time; it can be violated 5% or 1% of the time.

This is depicted in the chart below, where 95% of the area under the curve is to the left of the bar (i.e. the constraint right hand side value), and 5% is to its right.



This is one form of a *chance constraint*; the criterion that it must be satisfied for all realizations of the uncertainties up to a given percentile (say 95%) makes it a *VaR* (Value at Risk) constraint. We write this constraint as **VaR $_{0.95}$ A1 <= B1**. Risk Solver Platform supports two other criteria besides *VaR* that may be better choices for many models.

A chance constraint includes:

- A *left hand side* that depends on decision variables and uncertainties.
- A *relation* that must be either <= or >=. (A *chance* constraint can't be an equality. Note however that a *recourse* constraint *can* be an equality.)
- A *right hand side* that should be constant in the problem (if the RHS depends on decision variables or uncertainties, Risk Solver Platform converts LHS <= RHS into the form LHS – RHS <= 0).
- A *criterion* that may be VaR (Value at Risk), CVaR (Conditional Value at Risk), or USet (Uncertainty Set). These criteria are discussed below.
- A *measure* that may be a percentile 0.01 – 0.99 for VaR or CVaR, or a 'budget of uncertainty' (any positive value) for USet.

Value at Risk Measure

Chance constraints defined by a percentile or VaR (Value at Risk) measure have been used since the early 1960s. Such constraints offer a good deal of modeling flexibility, and they are easy to understand in terms of the probability that the constraint will be satisfied. Value at Risk is used in the banking and securities industries, and its use is mandated by the international Basel II accords. But chance constraints in this form have several drawbacks:

- A VaR constraint with probability 95% requires only that the constraint be satisfied – not violated – 95% of the time; it **says nothing about the magnitude of the violation** that may occur the other 5% of the time. For

example, a portfolio of securities that is VaR-constrained to lose no more than $100,000 95% of the time could still lose $1 million+ at other times.

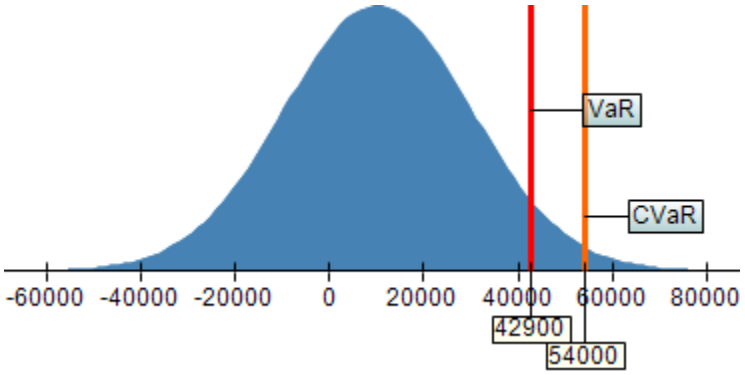
- As a measure of risk, the VaR criterion is **not subadditive**, a property expected of any 'coherent risk measure.' For example, if two portfolios A and B are VaR-constrained to not lose money 95% of the time, it is reasonable to expect that a combined portfolio A+B should have a *95% or better* chance of not losing money – but this is *not* guaranteed by the two portfolio VaR constraints.
- A VaR constraint is **not necessarily convex**; hence, using such a constraint in an otherwise convex model (for example, any linear programming or convex quadratic model) will radically affect its 'solvability' – it means that a globally optimal solution cannot be guaranteed, and solution time may rise exponentially with model size.

Further, when Risk Solver Platform uses robust optimization methods to automatically transform a model with VaR constraints into a larger but deterministic 'robust counterpart' model, it first approximates the non-convex VaR constraint with a convex CVaR constraint, and then transforms the CVaR constraint. Since CVaR is always **more conservative** than VaR as a risk measure, the robust counterpart solution will 'pay a price' in conservativeness, with a worse objective value. Users are often better off using CVaR directly.

Conditional Value at Risk Measure

To deal with the problems of Value at Risk cited above, an alternative risk measure called *Conditional Value at Risk* or CVaR (also called *Expected Tail Loss* or ETL) was developed in the late 1990s. **VaR $_{0.95}$ A1 <= B1** specifies that the 95th percentile of the realizations of A1 must be less than or equal to B1; realizations beyond the 95th percentile may be greater than B1 by any amount. In contrast, **CVaR $_{0.95}$ A1 <= B1** specifies that the *expected value* of ***all*** the realizations of A1 beyond the 95th percentile must be less than or equal to B1.

Below is a chart that compares VaR and CVaR. VaR is the value (42,900) that lies at the 95th percentile of the realizations of the constraint left hand side; 5% of the realizations are greater than 42,900 and lie in the 'tail' to the right of this point. CVaR (54,000) is the *expected* value (i.e. the mean or average value) of ***all*** the realizations that lie in the 'tail.' Note that, if **CVaR $_{0.95}$ A1 <= B1** is satisfied for some B1, then **VaR $_{0.95}$ A1 <= B1** is also (more than) satisfied.



As a risk measure, Conditional Value at Risk has several advantages over VaR:

- Unlike VaR, a CVaR constraint at 95% places a **bound on the average magnitude of the violations** that may occur 5% of the time.

- CVaR is a 'coherent risk measure.' It is **subadditive**, so if two portfolios A and B are CVaR-constrained to not lose money 95% of the time, then a combined portfolio A+B has the *same or better* chance of not losing money.
- A CVaR constraint is **always convex**. Models consisting of all convex functions can be solved to global optimality, and solved to very large size using modern interior point optimization methods.

Uncertainty Set Measure

Risk Solver Platform supports a third criterion for uncertainty in a chance constraint, which reflects the approach taken in most of the literature on *robust optimization* methods. This criterion, called *USet* (for uncertainty set), is applicable only to *linear* constraints, with coefficients a_i and variables x_i:

$a_1x_1 + a_2x_2 + ... + a_nx_n \leq b$

where some or all of the coefficients a_i may depend on the uncertainties. It is useful to think of the vector $[a_1\ a_2\ ...\ a_n]$ as having a *nominal* or expected value, and a *variation* from this value for each realization of the uncertainties.

A constraint of the form **USet$_\Omega$ A1 <= B1**, where $a_1x_1 + a_2x_2 + ... + a_nx_n$ is in A1, and b is in B1, specifies that A1 <= B1 must be satisfied for ***all*** *variations* from the *nominal* value of $[a_1\ a_2\ ...\ a_n]$ that do not exceed a bound Ω, measured by a norm. The bound Ω is often called the *budget of uncertainty* for the constraint. A very large Ω says that the constraint must be satisfied for practically all variations of the coefficients from nominal; a Ω of 0 effectively ignores uncertainty, requiring only that A1 <= B1 for the nominal value of $[a_1\ a_2\ ...\ a_n]$, and saying nothing about departures from this value.

Risk Solver Platform allows you to choose among four different norms to measure variation from the nominal value: The L1, L2, L-Infinity and D norms – as described in "Uncertainty Sets and Norms" later in this chapter.

Diagnosing Your Model's Use of Uncertainty

When you create a large model, or modify a large conventional optimization model to include many uncertainties and many constraints, you may or may not realize exactly how the model depends on the uncertainties. Just as you might accidentally use a formula that creates a nonlinear dependence on a decision variable, when you intended to create a linear programming model, you might introduce a dependence on uncertainty in a constraint that you intended to be deterministic. Risk Solver Platform can *diagnose* the use of uncertainty in your model, comparing it to an 'intended' use of uncertainty that you specify in the Task Pane Platform tab Diagnosis group **Intended Use of Uncertainty** option. After clicking the Analyze button, you can select **Reports – Optimization – Uncertainty** for a report of *exceptions* to your intended use of uncertainty.

Problems and Solution Methods

Previous sections of this chapter have described how you can **create models that involve uncertainty** in Risk Solver Platform, using certain and uncertain parameters, normal and recourse decision variables, normal and chance constraints, and your objective function. You create your models in the same way, regardless of the model transformation or solution method that will be used to solve the problem.

As outlined in the Introduction chapter, Risk Solver Platform is designed to find *good* solutions – given enough time – to models involving uncertainty in their **most general form**. But Risk Solver Platform is also designed to *automatically* recognize common special cases such as stochastic linear programming models, and find proven *optimal* solutions at much *faster* speeds. As with conventional LPs, the modeling assumptions of stochastic LPs are somewhat restrictive, but in a wide range of real-world applications, these assumptions can be satisfied. The great advantage of a stochastic LP is its potential *scalability* to tens of thousands of variables and constraints or more.

Before we can discuss classes of problems and how they are solved using Risk Solver Platform, we must explain how two key *concepts* – decision-dependent uncertainties and recourse decisions – affect the use of two key *solution methods* – stochastic programming and simulation optimization.

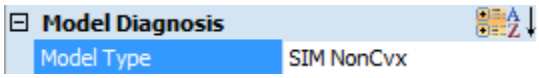
Decision-Dependent Uncertainties

As described in the Introduction section, in many real-world problems, the uncertain variables being modeled are *independent* of the decision variables. In other real-world problems, the uncertainties are *dependent* on the decision variables – they change when the decision variables change.

The methods of ***stochastic programming*** (SP) were created, more than fifty years ago, to deal with problems where the uncertainties are *independent* of the decisions. Much more recently, methods of *robust optimization* (RO) were created, also to deal with problems where uncertainties are *independent* of the decisions. Most methods in the SP and RO literature **cannot be applied** to problems where the uncertainties are dependent on decision variables. But where they can be applied, these methods are scalable to large models, and they generally yield optimal solutions.

The methods of ***simulation optimization*** were created, also more than fifty years ago, to deal with problems where the uncertainties may be *dependent* on the decisions. *This dependence makes the problem much harder to solve*. Where SP and RO methods often are based on linear programming and conic optimization, simulation optimization is usually based on methods such as genetic and evolutionary algorithms, and tabu and scatter search. These methods are more flexible and general – but they require far more solution time, are generally not scalable to large models, and yield only 'improved' solutions, not optimal solutions.

In Microsoft Excel, you can create models where the uncertainties are either independent of, or dependent on the (normal or 'first stage') decision variables. Risk Solver Platform can detect decision-dependent uncertainties and report this when you choose **Optimize – Analyze without Solving** or click the Analyze button in the Task Pane:



To solve *SIM NonCvx* (non-convex) problems using simulation optimization, you can use the built-in Evolutionary Solver (based on genetic algorithms) or the plug-in OptQuest Solver (which uses tabu and scatter search). You can also use built-in or plug-in nonlinear Solver Engines, in conjunction with Risk Solver Platform's *multistart* methods, to solve many such problems.

Resolving Uncertainty and Recourse Decisions

The methods of ***simulation optimization*** were developed to solve problems where the uncertainty will not be resolved over the horizon of interest. Hence, these solution methods find optimal values only for normal or 'here and now' decision variables; virtually all simulation optimization methods in the literature **have no concept of recourse decisions**.

Unfortunately, simulation optimization has often been misapplied to problems where the uncertainty *will* be resolved over the horizon of interest. Users may believe that their simulation optimization model will find the best possible decision in the presence of uncertainty. But this is very unlikely to be true, if it's possible to make certain decisions on a 'wait and see' basis. Recourse decisions create *flexibility* for the optimizer, making it easier to satisfy the constraints and attain a better objective.

The methods of ***stochastic programming*** were developed to solve problems where the uncertainty will be resolved over the horizon of interest. Recourse decisions – the ability to make certain decisions on a 'wait and see' basis – have been extensively studied in the SP literature, and many high-performance solution methods for two-stage SP problems with recourse have been developed.

Classes of Problems Involving Uncertainty

An optimization problem that includes uncertain variables is usually called a *stochastic optimization* or *stochastic programming* problem.

Below, we'll classify such problems based on the following factors:

- How the objective and constraints depend on the decision variables
- Whether the uncertain variables depend on the decision variables
- Whether the problem includes recourse decisions (i.e. whether it is one-stage with only normal variables, or two-stage with recourse variables)
- Whether the problem includes chance constraints

We'll also briefly describe how the three solution methods currently supported by Risk Solver Platform – robust optimization, stochastic programming, and simulation optimization – can be applied to these problems.

One-Stage Problems

One-Stage Linear Programming Problems

The simplest case is a one-stage stochastic linear programming problem:

- The objective and constraints **depend linearly** on the decision variables.
- The uncertain variables are **independent** of the decision variables.
- The problem includes **no recourse decisions**.
- The problem includes **chance constraints** to deal with the uncertainty.

This kind of problem arises if you start with a conventional, deterministic linear programming model (where all coefficients are certain), you add uncertain variables using functions such as **=PsiUniform(0,1)** or **=PsiNormal(2,1.5)** in

cells, and you add formulas to compute the objective or certain constraints based on these cells, so that the LP coefficients are now uncertain.

Since no recourse decisions are available, the Solver must find values for the (normal or first-stage) decision variables that satisfy the constraints (normal and chance constraints) and maximize or minimize the objective.

We could solve this problem with **simulation optimization** methods. But since it is linear, this problem is especially well suited for **robust optimization** methods. With RO methods we can solve *much larger* models, hundreds of times faster than with simulation optimization. In the Task Pane Platform tab, we set the Optimization Model group option **Solve Uncertain Models** to Stochastic Transformation, and we set the Transformation group option **Stochastic Transformation** to Robust Counterpart. When we click Optimize, the transformed model is created and solved.

One-Stage Quadratic Programming Problems

A slightly more complex case is a one-stage stochastic problem with a quadratic objective and all linear constraints:

- The objective is a **deterministic, convex quadratic** function of the decision variables – it doesn't depend on the uncertainties.
- The uncertain variables are **independent** of the decision variables.
- The problem includes **no recourse decisions**.
- The problem includes **chance constraints** to deal with the uncertainty.

The situation here is basically the same as for one-stage stochastic linear programming problems. We can apply **simulation optimization** methods, but we can also apply more scalable **robust optimization** methods, as long as the objective doesn't depend on the uncertainties.

One-Stage Nonlinear Optimization Problems

The situation changes if we have a one-stage stochastic *nonlinear* problem:

- The objective or the constraints are **general nonlinear**, convex or non-convex functions of the decision variables.
- The uncertain variables may be **dependent** on the decision variables.
- The problem includes **no recourse decisions**.
- The problem includes **chance constraints** to deal with the uncertainty.

We cannot use Risk Solver Platform's robust optimization methods or stochastic programming methods on problems with general nonlinear, possibly non-convex functions. Our only option is to use **simulation optimization**; this is feasible since the problem does not include any recourse decisions. Given that we are using simulation optimization, we can allow the uncertainties to be dependent on the decisions, although some speed gains are possible if they are not.

Two-Stage Problems

Two-Stage Linear Programming Problems

A very common case is a two-stage stochastic linear programming problem:

- The objective and constraints **depend linearly** on the decision variables.

- The uncertain variables are **independent** of the decision variables.
- The problem includes both normal decisions and **recourse decisions**.
- The problem may or may not include **chance constraints**.

The presence of recourse decisions in the problem implies that at least some (and usually all) of the uncertainty *will* be resolved over the horizon of interest. Hence the problem is called 'two-stage,' where the normal decisions are made at the first stage – before the uncertainties are resolved – and the recourse decisions are made at the second stage – after they are resolved. The Solver must find single values for the normal variables, and multiple values for the recourse variables (for each realization of the uncertainties) that satisfy the constraints and optimize the objective.

We cannot solve this problem with conventional **simulation optimization** methods, because these methods have no concept of recourse decisions. But since the problem is linear, we can use either **stochastic programming** or **robust optimization** methods – and with these methods we can solve *much larger* models, hundreds of times faster than with simulation optimization. In the Task Pane Platform tab, we set the Optimization Model group option **Solve Uncertain Models** to Stochastic Transformation, and we set the Transformation group option **Stochastic Transformation** to either Deterministic Equivalent or Robust Counterpart. When we click Optimize, the transformed model is created and solved.

If we choose the 'Determ Equivalent,' we can place integer constraints on the normal and recourse variables (though this may make the model more difficult to solve). But the model cannot include chance constraints, and the transformed model may become quite large (its size depends on the *product* of the number of recourse variables and constraints, and the number of simulation trials).

If we choose the 'Robust Counterpart', the model can include chance constraints, and the transformed model will be significantly smaller and solve faster than if we choose 'Determ Equivalent.' But we cannot place integer constraints on the recourse variables, and the solution may be more conservative than with 'Determ Equivalent.'

Two-Stage Quadratic Programming Problems

A slightly more complex case than the one above is a two-stage stochastic problem with a quadratic objective and all linear constraints:

- The objective is a **deterministic, convex quadratic** function of the decision variables – it doesn't depend on the uncertainties or the recourse variables.
- The uncertain variables are **independent** of the decision variables.
- The problem includes both normal decisions and **recourse decisions**.
- The problem includes **no chance constraints**.

Again the presence of recourse decisions in the problem implies that at least some (and usually all) of the uncertainty *will* be resolved over the horizon of interest. We cannot solve this problem with conventional **simulation optimization** methods, because these methods have no concept of recourse decisions. But we can use **stochastic programming** methods – and these methods are far more scalable than simulation optimization methods would be. In the Task Pane Platform tab, we set the Optimization Model group option **Solve Uncertain Models** to Stochastic Transformation, and we set the Transformation group option **Stochastic Transformation** to either

Deterministic Equivalent. When we click Optimize, the transformed model is created and solved.

Two-Stage Nonlinear Optimization Problems

If we have a two-stage stochastic nonlinear problem:

- The objective or the constraints are **general nonlinear**, convex or non-convex functions of the decision variables.
- The uncertain variables may be **dependent** on the decision variables.
- The problem includes **recourse decisions**.
- The problem may or may not include **chance constraints**.

We cannot use Risk Solver Platform's **robust optimization** methods or **stochastic programming** methods on problems with general nonlinear functions, and we cannot use conventional **simulation optimization** methods either, because these methods have no concept of recourse decisions. But future Frontline Systems products may be able to solve such problems; contact us for more information.

Advanced Topics

Bounds, Discretization, and Correlation

An uncertainty specified via a probability distribution such as PsiUniform(0,1) has finite bounds of 0 and 1, but it has a practically infinite set of possible values between 0 and 1; we can only consider a *sample* of the possible values during an optimization. An uncertainty specified via PsiNormal(0,1) has infinite bounds, as well as an infinite set of possible values. In practice, however, most of the probability mass of the Normal distribution lies within three standard deviations on either side of the mean, and a reasonable *sample* of values from the Uniform and Normal distributions can be used for computations.

Risk Solver Platform uses Monte Carlo simulation to generate samples of uncertainties, based on the probability distributions specified by the user. It uses the minimum and maximum values for each distribution obtained during the simulation as 'effective' lower and upper bounds, which are used in constructing the robust counterpart to an uncertain model.

When Risk Solver Platform solves a stochastic LP (as explained above) by forming the 'deterministic equivalent' problem, the Monte Carlo trials are used as 'scenarios' in constructing this problem; hence they serve as a discretization of any continuous probability distributions specified by the user.

The impacts of uncertainties on each coefficient of the objective and the constraints may be *correlated* – either because multiple coefficients depend on the same primitive uncertainty, or because correlation was deliberately induced among several uncertain variables. When Risk Solver Platform solves a stochastic LP by forming the 'deterministic equivalent' problem, the correlations are reflected in the Monte Carlo trials used to construct the problem. When it solves a stochastic LP using robust optimization methods, the robust transformation takes into account the observed correlations in the sample.

Uncertainty Sets and Norms

As described in the earlier section "More on Chance Constraints," if a chance constraint is linear in the decision variables, you can use the *USet* (uncertainty set) criterion, in lieu of the VaR or CVaR criterion. The advantage of using this criterion is that the robust counterpart model more accurately reflects the degree to which you want the chance constraint to be satisfied, which can lead to less conservative solutions, with better objective values. Consider a constraint:

$a_1x_1 + a_2x_2 + ... + a_nx_n \leq b$

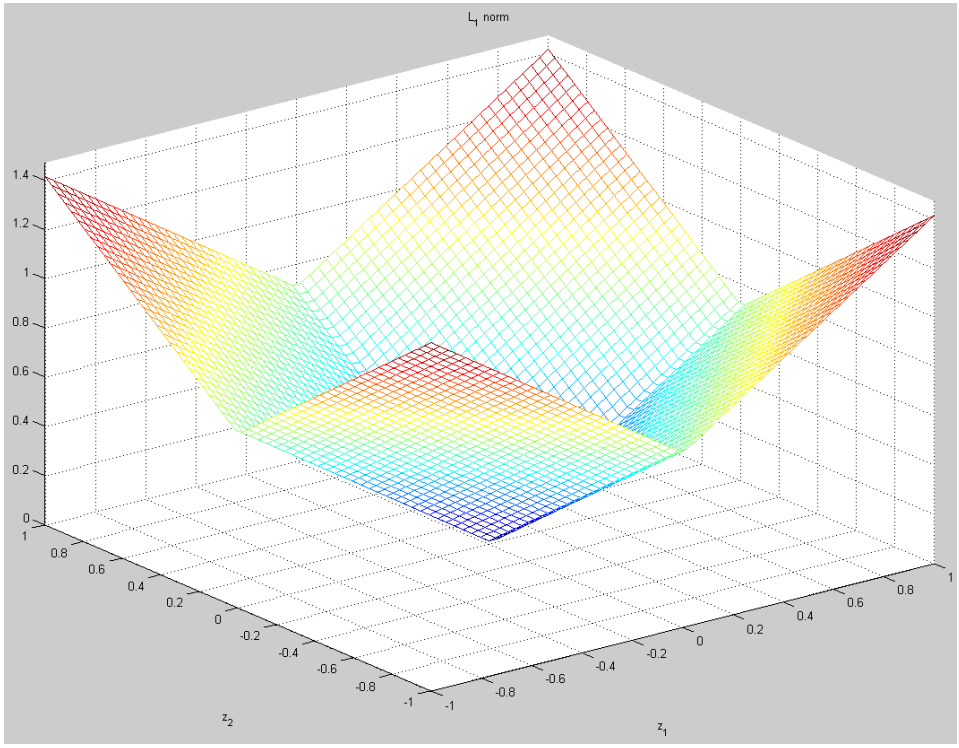
where $=a_1x_1 + a_2x_2 + ... + a_nx_n$ is in A1, b is in B, and some or all of the coefficients a_i may depend on uncertain variables z_1, z_2, ... It is useful to think of the vector $[a_1\ a_2\ ...\ a_n]$ as having a *nominal* or expected value, and a *variation* from this value for each realization of the uncertain variables.

A constraint of the form **USet$_\Omega$ A1 <= B1** specifies that A1 <= B1 must be satisfied for ***all*** *variations* from the *nominal* value of $[a_1\ a_2\ ...\ a_n]$ that do not exceed a bound Ω, measured by a norm. The *uncertainty set* includes all the points formed by adding a vector of allowed variations to the vector of nominal values; the bound Ω is often called the *budget of uncertainty* for the constraint.

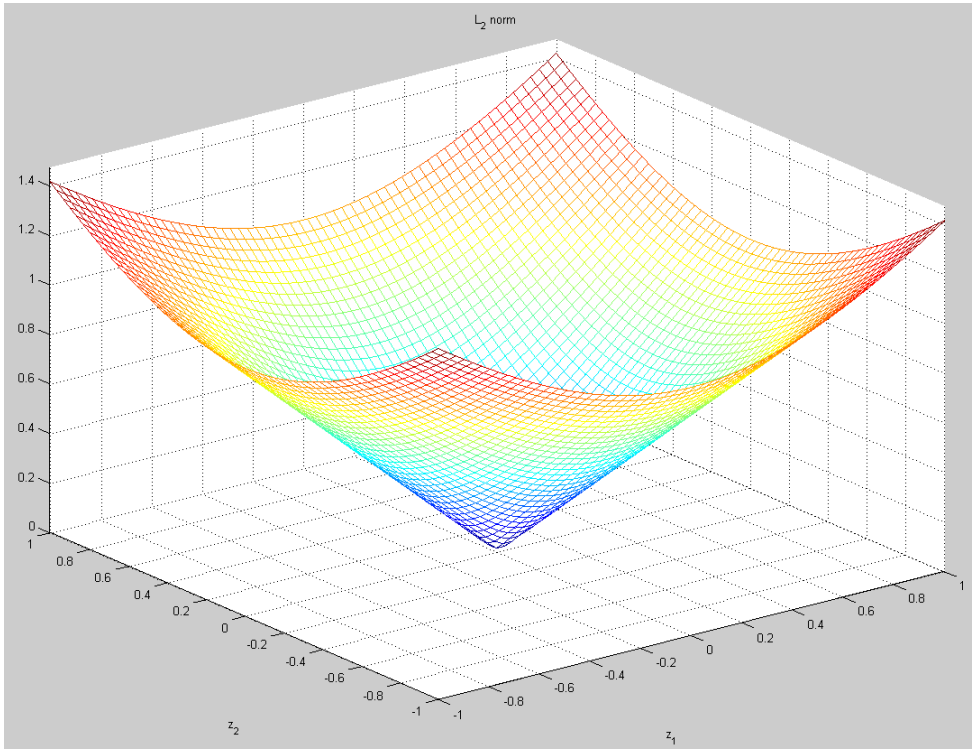
Risk Solver Platform allows you to choose among four different norms to measure variation from the nominal value: The L1, L2, L Inf (Infinity) and D norms. (One choice of norm applies to all chance constraints.) The graphs shown on the following pages may help you visualize the shape of the uncertainty set (based on two uncertain variables z_1, z_2) for each of the norms.

The D norm represents the intersection of the L1 norm and the L-Inf norm; thus it can define a 'tighter' uncertainty set than either of these norms alone. For the D norm, Ω can be interpreted as a bound on the *number* of coefficients $[a_1\ a_2\ ...\ a_n]$ that depart from nominal values. When the D norm is used, the robust counterpart of a stochastic LP problem is a (larger, conventional) LP problem; when the L2 norm is used, the robust counterpart is an SOCP problem.

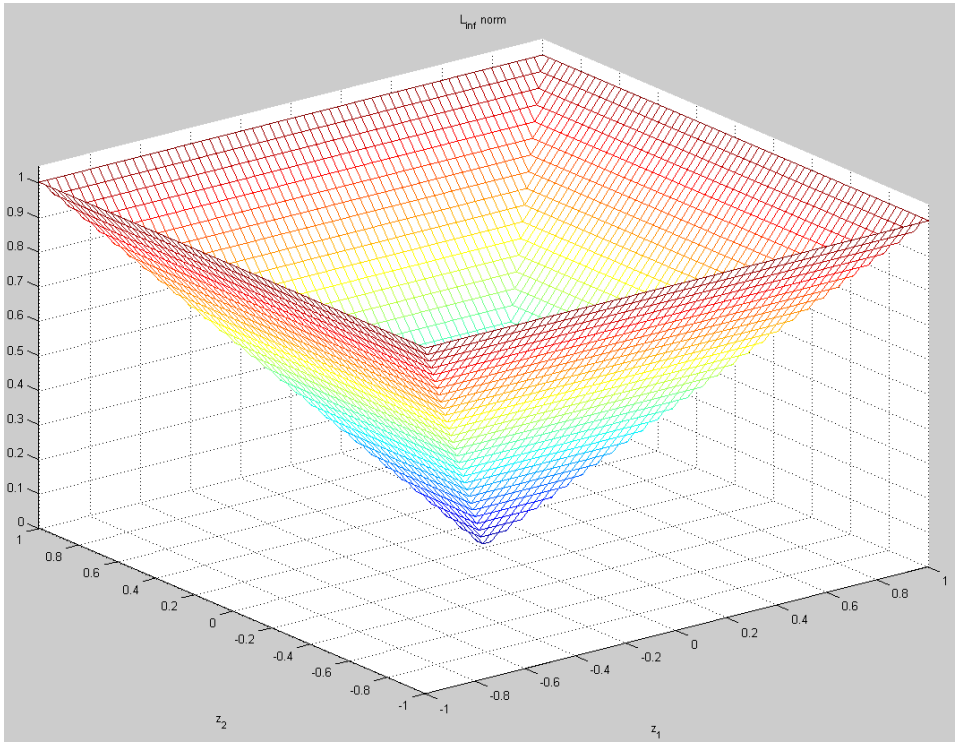
L1 Norm



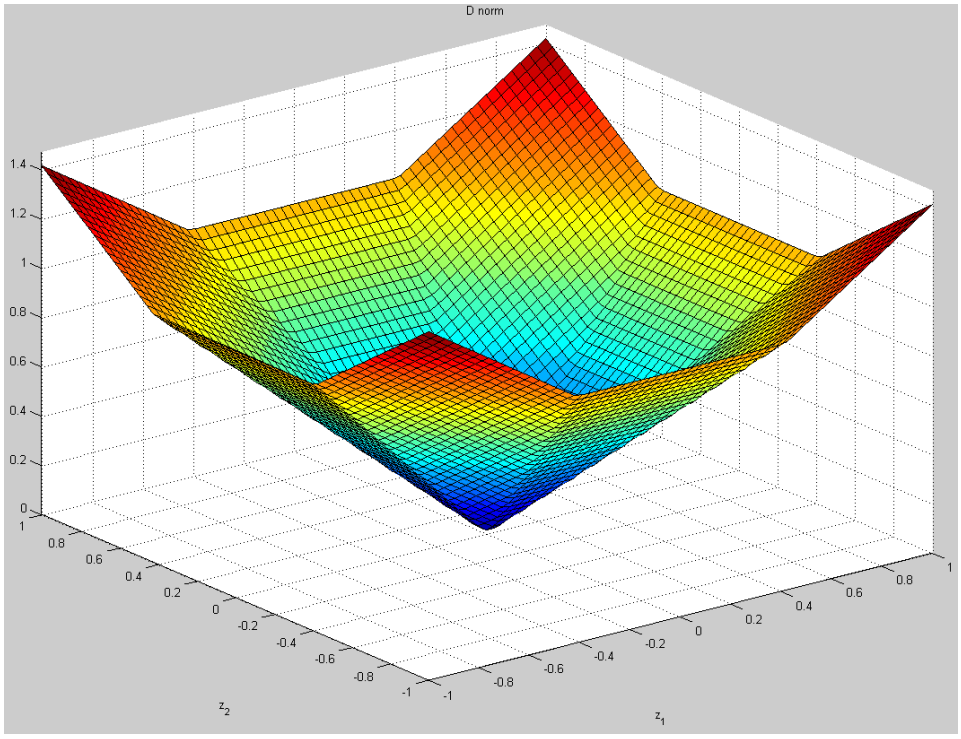
L2 Norm



L Inf Norm



D Norm



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 "Examples: Conventional Optimization" – 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 using the Premium Solver Pro and Premium Solver Platform, 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 Pro (and in some cases, Premium Solver Platform and Risk Solver Platform) 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 *Management Science: The Art of Modeling with Spreadsheets* 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 slider to zoom in Excel 2007/2010, or the View Zoom menu option in Excel 2003 to get a high-level view of your spreadsheet's structure. Select 'Show Formulas' on the Formulas tab in Excel 2007/2010, or check the 'Formulas' box on the Tools Options View tab in Excel 2003 to display formulas instead of values, and scan them for consistency. Learn to use Excel's auditing functions 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 Pro requires that cells containing decision variables and constraint left hand sides are on the active worksheet. But Risk Solver Platform and Premium Solver Platform allow 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 one of these products 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 the Frontline Solvers Reference Guide.

Several commentators on good spreadsheet modeling practice feel that models defined on a single worksheet are easier to understand and maintain. In Excel 2007 and beyond, 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 Risk Solver Platform. This section describes steps that are easy to apply in most situations.

For users of Premium Solver Pro, the best recommendation we can make to improve performance is to *upgrade to Premium Solver Platform or Risk Solver Platform*. This is more than just a "sales pitch" – every step you take costs something, either in terms of money or your effort. For most professionals, the cost of upgrading will be repaid if it saves just a few hours of time. And you can find out *at no cost* whether the upgrade will be worthwhile – just download the Risk Solver Platform Setup program, request a free 15-day evaluation license, and try solving your model with the actual software.

For users of Risk Solver Platform and Premium Solver Platform, we highly recommend that you *try solving your model with our field-installable Solver Engines* – especially the Large-Scale SQP Solver, KNITRO Solver, MOSEK Solver Engine, and XPRESS 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* 1 GB RAM if you are working with large Solver models – 2 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 "Getting Results: Conventional Optimization."

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 via the Ribbon Constraints choice, 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 Engine tab options for these Solver engines include an option **Require Bounds on Variables**, which is *True* 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. If you are using the Interval Global Solver or the OptQuest Solver, bounds on all variables are *required* – the Solver will always stop with an error message if bounds on the variables are missing.

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 "Mastering Conventional Optimization Concepts," 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, MOSEK Solver or XPRESS Solver, 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 XPRESS Solver 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 "Mastering Conventional Optimization Concepts," 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 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. If you're using the GRG Solver, you can set its Engine tab option **Recognize Linear Variables** to True, to save time during the solution process. And the Large-Scale GRG Solver and Large-Scale SQP Solver engines also recognize both *linearly occurring variables* and *linear constraints* automatically, for still faster solutions. The Large-Scale SQP Solver is particularly effective when used with Premium Solver Platform or Risk Solver Platform, because it uses the Interpreter's Structure analysis to *break down each function into linear and nonlinear terms*, which it handles as efficiently as possible.

You can use the Ribbon choice **Optimize – Analyze without Solving** or the Task Pane Analyze button to easily determine the number of linear variables, functions, and occurrences of variables in functions, as illustrated in the chapter "Examples: Conventional Optimization."

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 Risk Solver Platform *automatically transforms* your model, 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.

You can sometimes use a **semi-continuous** variable to model this kind of situation, instead of a binary variable, continuous variable, and "Big M" constraint. At the optimal solution, a semi-continuous variable must either be zero, or must lie within a specified continuous range. This is usually even more efficient than using a fixed-charge constraint as outlined above.

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 "Mastering Conventional Optimization Concepts," 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*; 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 "Mastering Conventional Optimization Concepts," 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 Task Pane Platform tab **Nonsmooth Model Transformation** option, or 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 smooth nonlinear. 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 $g(x)$:

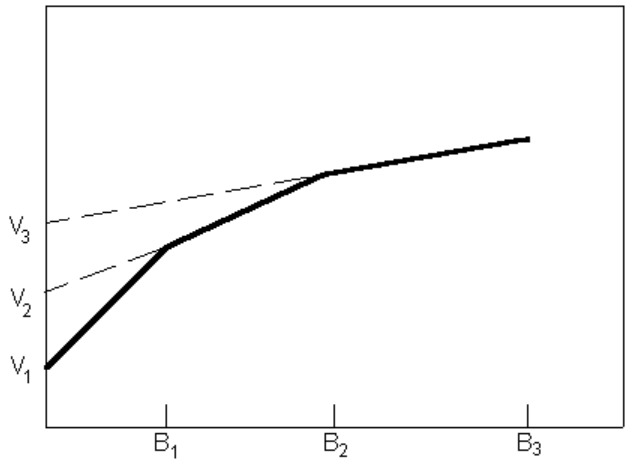
$f(x) - F <= My$
$F - f(x) <= M(1-y)$
$g(x) - G <= M(1-y)$

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 on the next page 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:

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

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.

Organizing Your Model for Fast Solution

This section describes ways you can organize your model so that Premium Solver Pro and Premium Solver Platform can analyze it more efficiently. Most of this section is devoted to an in-depth discussion of "fast problem setup" for *linear* and *quadratic* models (possibly with integer variables); it is not applicable to *nonlinear* and *non-smooth* models. Because the Polymorphic Spreadsheet Interpreter in Premium Solver Platform and Risk Solver Platform

has largely superseded this form of fast problem setup, this section is relevant primarily for (i) Premium Solver Pro, where the Interpreter is not available and (ii) very large LP models (100,000 variables or more), where memory required for the Interpreter may be greater than available RAM and cause swapping to disk.

Fast Problem Setup

In Premium Solver Pro, if your model is linear or quadratic, you may find that the Solver spends most of its time with "Setting Up Problem..." on the Excel status bar, then speeds through the "Trial Solutions" very quickly. The setup time is required to extract the LP coefficients of the problem functions by recalculating the worksheet. (The LP coefficients are the first partial derivatives of the problem functions with respect to the variables, and they are obtained by the method of *finite differencing*, which requires $n + 1$ recalculations if there are n decision variables.)

Much of this setup time can be avoided if you write the formulas for the objective function and *all* of the constraint left hand sides using the functions recognized for *fast problem setup*: SUM, SUMPRODUCT, DOTPRODUCT, QUADPRODUCT and MMULT. This may require some work on your part to revise a model you have already constructed, but you'll be rewarded with a *5- to 100-fold speed improvement* in setup time, compared to the time taken by the Premium Solver Pro when finite differencing is used.

You can always express a linear or quadratic programming problem using these functions for the objective and all of the constraints, although you may need to introduce new sets of cells to hold the calculated coefficients so that these cells can be referenced by one of the fast problem setup functions.

The PSI Interpreter in Premium Solver Platform and Risk Solver Platform uses the techniques of *automatic differentiation* to obtain the LP coefficients faster and more accurately than they can be obtained via finite differencing. Because the Interpreter handles almost every kind of Excel formula and built-in function, you don't have to do the work of designing your model – or revising an existing model – to use only the small set of functions recognized for fast problem setup.

Fast problem setup is still available in Premium Solver Platform and Risk Solver Platform as a specialized method of extracting the constant Jacobian matrix (the LP coefficients) of a linear or quadratic problem, and the constant Hessian matrix (the QP coefficients) of a quadratic objective function – but it is used only if you set the Task Pane Platform tab Optimization group Interpreter option to **Excel Interpreter**. If your LP or QP model is very large, defining it in fast problem setup format may still save time compared to use of the PSI Interpreter – but the advantage is not nearly as great as the 5- to 100-fold speed improvement mentioned above.

The following subsections describe the functions supported for fast problem setup, and the form of the function arguments that you must use to ensure that they are recognized for fast setup purposes.

The SUM Function

The simplest case of a function recognized for fast problem setup is a formula such as =SUM(C1:C10) where C1 through C10 are decision variables. An example of the use of SUM to define constraints can be found in the "Shipping Routes" sheet in the SOLVSAMP.XLS workbook included with Microsoft Excel. Note that a SUM of decision variables is a linear function where all of

the coefficients are 1. To be recognized in fast problem setup, your formula must consist *only* of =SUM(*cells*) (with no constants) where every cell referenced is a decision variable. You may use absolute or relative references or defined names in the arguments to SUM.

The SUMPRODUCT Function

The SUMPRODUCT function is documented in Microsoft Excel online Help. It returns the sum of the products of corresponding elements in its two arguments, which is exactly the form of a linear function:

$a_1x_1 + a_2x_2 + ... + a_nx_n$

SUMPRODUCT requires that its two arguments refer to the same *number* of cells in the same *orientation* (either a row, a column or a rectangular area of cells). Only single selections, not multiple selections, are permitted in the arguments. If SUMPRODUCT is used in an array formula (see below), it will return the same value in every resulting array element, which is usually not the result you want. (The DOTPRODUCT function, described below, has more flexible arguments and is far more useful in array formulas.) To be recognized in fast problem setup, your formula must consist *only* of =SUMPRODUCT(*cell range*, *cell range*) where one of the cell ranges consists *entirely* of decision variables, and the other cell range consists *entirely* of cells that are constant in the Solver problem. You may list the arguments in either order, using absolute or relative references or defined names.

Other Functions for Fast Problem Setup

Use of the MMULT function is illustrated below under "Using Array Formulas." To be recognized in fast problem setup, your formula must follow the same rules as for SUMPRODUCT: It must consist *only* of =MMULT(*cell range*, *cell range*) where one cell range specifies the decision variables, and the other cell range specifies the corresponding coefficients.

The DOTPRODUCT and QUADPRODUCT functions are described in their own sections below. To be recognized in fast problem setup, your usage of these two functions must follow the same rules as for SUMPRODUCT.

To qualify as a quadratic programming (QP) problem – which can be solved efficiently by the LP/Quadratic Solver engine – only the *objective function* (not any of the constraints) may use QUADPRODUCT, or any other quadratic form.

Using Array Formulas

Optimization models in algebraic form can often be expressed more compactly using indexing and summation notation. For example, the five constraints in EXAMPLE1 could be written as:

$$\sum_{j=1}^{3} a_{ij} x_j,\ i=1,...,5$$

The SUMPRODUCT function corresponds to the summation expression above for one constraint. The *entire set* of five constraint formulas could be defined with the array form of the DOTPRODUCT function (described in detail later in this section). In EXAMPLE1, you would select cells C11 to C15 and "array-enter" the following formula:

{=DOTPRODUCT(D9:F9,D11:F15)}

The braces above are not typed, but they appear when you "array-enter" the formula by pressing CTRL-SHIFT-ENTER instead of just ENTER. If you aren't familiar with array formulas in Microsoft Excel, you can read about them in Excel's online Help. They are one of the most useful features of Microsoft Excel.

If your LP model is "dense" and regular in form rather than "sparse," you may wish to consider use of Microsoft Excel's matrix built-in functions, such as MMULT which (when array-entered into a group of cells) yields the matrix product of its two operands. For example, the five constraints in EXAMPLE1 could be written in vector-matrix form as:

Ax <= **b**

where **A** is the matrix of coefficients, **x** is the vector of decision variables and **b** is the vector of constraint right hand sides. In the Microsoft Excel formula language, the left hand side of this expression could be written as:

{=MMULT(_A,TRANSPOSE(_X))}

(The TRANSPOSE function is needed only to "match up" the orientation of the matrix _A with the row vector _X.) In worksheet EXAMPLE1, if you insert defined names _A for the coefficients D11:F15 and _X for the variables D9:F9, then select cells C11:C15 and array-enter the above formula, it will compute the values of all five constraints.

The PSI Interpreter in Premium Solver Platform and Risk Solver Platform recognizes most kinds of array formulas supported by Microsoft Excel. But (for rather technical reasons) the use of array formulas actually involves a speed *disadvantage* in the PSI Interpreter when the coefficients are extracted via automatic differentiation.

If you're using Premium Solver Platform and Risk Solver Platform, we recommend that you use array formulas where they make sense, and focus on making your model simple and easy to maintain. In a large model, you'll probably find that you want or need to use multiple tabular areas for the formulas that define your constraints, and it may be inconvenient or impractical to define entire constraint left hand sides with functions like MMULT and TRANSPOSE.

Using the Add-in Functions

Risk Solver Platform and its subset products define two special Excel functions: DOTPRODUCT and QUADPRODUCT. These functions behave just like Excel built-in functions: You can use them in formulas in any spreadsheet (not only in Solver models). When you use the Insert Function… menu option, these functions will appear in the "Select a Function" or "Paste Function" list (classified as Math & Trig functions), and you'll be prompted with named edit fields for their arguments.

In addition, DOTPRODUCT and QUADPRODUCT are recognized for purposes of fast problem setup as described earlier in this chapter. They are also recognized by the PSI Interpreter.

Using DOTPRODUCT

DOTPRODUCT is a generalized version of the Excel function SUMPRODUCT, and it is very useful for defining the objective function and constraints of linear programming problems. DOTPRODUCT is also recognized for fast problem setup as described above, provided that you follow the rules outlined earlier:

Your formula must consist *only* of =DOTPRODUCT(*cell reference*, *cell reference*) where all of the cells in one of the cell references are decision variables, and all of the cells in the other cell reference are constant in the Solver problem. Each *cell reference* must be either an *individual selection* or a *defined name*, but the cell ranges specified by the two arguments need not have the same "shape" (row, column, or rectangular area).

For use in Excel and for purposes of fast problem setup, DOTPRODUCT will accept defined names that specify *multiple selections* for either of its arguments. For example, if you had designed a model where the decision variables consisted of *several* rectangular cell selections, you could still calculate the objective function for your model with *one* call to DOTPRODUCT.

DOTPRODUCT always processes its arguments in *column, row, area* order – in an individual selection it reads cells across columns, wrapping around to subsequent rows, and in a multiple selection it reads the individual cell selections in the order in which they are listed. For example, the formula:

=DOTPRODUCT(A1:C2,D1:D6)

will calculate as =A1*D1+B1*D2+C1*D3+A2*D4+B2*D5+C2*D6.

The Array Form of DOTPRODUCT

If SUMPRODUCT is used in an array formula, it returns a scalar (single number) result, which is returned in every cell of the array. However, if DOTPRODUCT is used (with the proper arguments) in an array formula, it returns an *array result*. You can use this capability to calculate the left hand sides of several constraints with a single array formula. In a sparse model where you'd like to use the built-in function MMULT to compute the constraint values, but the variables and constraints aren't laid out in a single matrix, you can use the array form of DOTPRODUCT instead.

Further, when you use the array form of DOTPRODUCT, the Premium Solver products will recognize this form and use it to process many constraints at once in problem setup. (The array form is recognized for fast problem setup, and it's also recognized by the PSI Interpreter in Premium Solver Platform and Risk Solver Platform.) If you can't use the array form, even the simple form of DOTPRODUCT will save time in problem setup.

DOTPRODUCT will return an array value when the number of cells in one of its arguments is an *even multiple* of the number of cells in its other argument. As an example, consider the calculation of parts used in the LP model EXAMPLE1. The decision variables are in cells D9 to F9 (3 cells), and the coefficients of the constraint left hand sides – the number of parts used for each product – are in cells D11 to F15 (15= 3*5 cells). We want to calculate the left hand sides of the constraints in cells C11 to C15. To do this, we would first select the group of five cells C11:C15 with the mouse. Then we would type:

=DOTPRODUCT(D9:F9,D11:F15)

completing the entry with CTRL+SHIFT+ENTER instead of just ENTER. The formula will display as {=DOTPRODUCT(D9:F9,D11:F15)} – the braces are added by Microsoft Excel when the formula is array-entered. With the cell values shown in EXAMPLE1 prior to solution (e.g. 100 for each of the decision variables), this array formula will calculate 200 in C11, 100 in C12, 500 in C13, 200 in C14 and 400 in C15. Hence, it will compute the same set of values as the array expression shown earlier: {=MMULT(_A, TRANSPOSE(_X))}.

Whether it is used in the simple form or the array form, DOTPRODUCT always processes its arguments in *column, row, area* order. In the array form, when the

cells in the "shorter" argument have all been processed and cells remain to be processed in the "longer" argument, DOTPRODUCT "wraps around" to the beginning of the "shorter" argument. In the example above, cell C11 calculates the value =D9*D11+E9*E11+F9*F11; cell C12 computes =D9*D12+E9*E12+F9*F12; and so on. Keep this rule in mind when you use the array form of DOTPRODUCT, and keep your spreadsheet layouts as simple as possible!

Using QUADPRODUCT

The QUADPRODUCT function can be used to define the objective for quadratic programming problems in a single function call, as required for fast problem setup.

As explained in "Quadratic Functions" in the chapter "Mastering Conventional Optimization Concepts," a quadratic function is a sum of terms, where each term is a (positive or negative) constant (called a *coefficient*) multiplied by a *single variable* or the *product of two variables*. This means that in order to represent the most general quadratic function, we might have a coefficient for each instance of a single variable, and a coefficient for each possible *pair* of two variables. The QUADPRODUCT function is designed to supply coefficients for each single variable and each pair of variables, in a manner similar to SUMPRODUCT and DOTPRODUCT.

You supply the arguments of QUADPRODUCT as shown below:

=QUADPRODUCT(*variable cells, single coefficients, pair coefficients*)

The first argument must consist entirely of decision variable cells. The second and third arguments must consist entirely of cells whose values are *constant* in the optimization problem; if these cells contain formulas, those formulas must not refer to any of the decision variables. The second argument supplies the coefficients to be multiplied by each single variable in the first argument, using an element-by-element correspondence. The third argument supplies the coefficients to be multiplied by each *pair* of variables drawn from the first argument. Hence, if there are n cells in the first argument, there must be n^2 cells in the third argument. If the variables are represented by $x_1,x_2,...,x_n$, the single coefficients by $a_1,a_2,...,a_n$, and the pair coefficients by $c_1,c_2,...,c_N$ where $N = n^2$, QUADPRODUCT computes the function:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{n(i-1)+j}\, x_i\, x_j + \sum_{j=1}^{n} a_j\, x_j$$

The pairs are enumerated starting with the first cell paired with itself, then the first cell paired with the second cell, and so on. For example, if the first argument consisted of the cells A1:A3, there should be nine cells in the third argument, and the values in those cells will be multiplied by the following pairs in order: A1*A1, A1*A2, A1*A3, A2*A1, A2*A2, A2*A3, A3*A1, A3*A2, and A3*A3. The value returned by QUADPRODUCT is the sum of all of the coefficients multiplied by their corresponding single variables or pairs of variables.

Multiple selections can be used for each argument of QUADPRODUCT, subject to the same considerations outlined above for DOTPRODUCT: You can use the general syntax for multiple selections in Microsoft Excel, but defined names are needed for purposes of fast problem setup, and multiple selections are not accepted by the PSI Interpreter.

A common application of quadratic programming is to find an "efficient portfolio" of securities – often called *portfolio optimization*. Worksheet EXAMPLE4 in the **StandardExamples.xls** workbook, used in the chapter "Examples: Conventional Optimization," illustrates portfolio optimization using the Markowitz method. This model uses the QUADPRODUCT function to compute the variance of a portfolio of five securities, and uses this quantity as the objective to be minimized, subject to a constraint that gives a lower limit on the portfolio return. Because the *objective* is a quadratic function, and the *constraints* (including the bound on return) are all linear functions, this Solver model is a quadratic programming (QP) problem which can be handled by the LP/Quadratic Solver in Premium Solver Platform and Risk Solver Platform.

References and Further Reading

Introduction

Although this Guide provides many valuable hints for making effective use of Risk Solver Platform, it does not attempt to teach you everything about optimization, Monte Carlo simulation, and stochastic optimization. We strongly recommend that you consult one of the books cited below, or discuss your problem with a **knowledgeable consultant** at Frontline Systems, at a nearby university, or at a firm specializing in optimization and simulation methods. There's 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 **training seminar** on risk analysis and simulation, optimization, and other advanced techniques in Excel, presented by Frontline Systems instructors or consultants who work closely with us. For current information on training seminars, visit www.solver.com or contact us at info@solver.com.

Textbooks Using Solver and Risk Solver Platform

***Management Science: The Art of Modeling with Spreadsheets*, 3rd** Edition by Stephen G. Powell and Kenneth R. Baker, published by John Wiley & Sons, ISBN 978-0470530672.

Now in its third edition, Management Science helps business professionals gain the essential skills needed to develop real expertise in business modeling. The biggest change in the text is the conversion of software from Crystal Ball to Risk Solver Platform to reflect changes in the field. More coverage of management science topics has been added. Overall, 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. Additional open-ended case studies that are less structured have also been included along with new exercises. These changes will help business professionals learn how to apply the information in the field. A limited term license for the education version of Risk Solver Platform is available for free download with the textbook. However, this education version has much lower problem size limits than the commercial version.

***VBA for Modelers - Developing Decision Support Systems Using Microsoft Excel*, 2nd** Edition by S. Christian Albright, published by South-Western College Publishing. This unique book will prove invaluable to anyone seeking to use VBA (Visual Basic Application Edition) to programmatically control Microsoft Excel and build custom applications. It includes a basic introduction to VBA

and the Excel object model, and 16 example applications developed in depth with VBA source code, many of them calling the Solver's VBA functions. The applications include blending, product mix, production scheduling and similar models, plus capital budgeting, stock trading, option pricing and portfolio optimization. The 3rd Edition of this book is due in the Fall of 2009.

Spreadsheet Modeling and Decision Analysis: A Practical Introduction to Management Science, 6th Edition by Cliff T. Ragsdale, published by South-Western College Publishing. ISBN 978-0538746311.

Cliff Ragsdale's book was first to base all of its optimization examples on the Microsoft Excel Solver, and it has become the best-selling MBA textbook for management science. You'll find a discussion of linear, nonlinear and integer programming; an explanation of sensitivity analysis and how to use the Solver's reports; topics like goal programming and multiobjective optimization; and additional coverage of regression, time series analysis, queuing, project management, decision analysis, and other topics. The 6th Edition uses Risk Solver Platform for its examples. A limited term license for the education version of Risk Solver platform is available for free download with the textbook. However, this education version has much lower problem size limits than the commercial version.

Practical Management Science, 3rd Edition by Wayne Winston and S. Christian Albright, published by South-Western College Publishing. This textbook, also widely used in new MBA courses on management science, provides an extensive introduction covering linear and integer programming, nonlinear optimization, and genetic and evolutionary algorithms using Frontline's Evolutionary Solver, as well as other management science topics. This book is slightly more challenging than Cliff Ragsdale's book, but includes an extensive set of spreadsheet models and a whole chapter on the Evolutionary Solver. It also includes Premium Solver for Education on CD-ROM.

Introduction to Mathematical Programming, 4th Edition by Wayne Winston and Munirpallam Venkataramanan, published by Duxbury Press. This book focuses entirely on optimization, at a more technical level than the textbooks described above – including topics in linear algebra, the Simplex method, goal programming, integer programming and the Branch & Bound method, and the differential calculus topics underlying nonlinear optimization. It also includes Premium Solver for Education on CD-ROM.

Managerial Spreadsheet Modeling and Analysis by Richard Hesse, published by Richard D. Irwin. This "good, but hard to find" book teaches you how to formulate a model from a complex business situation, using a four-step process: Picture and paraphrase, verbal model, algebraic model and spreadsheet model. It covers types of models ranging from simple goalseeking and unconstrained problems to linear, nonlinear and integer programming problems. And it includes over 100 Microsoft Excel spreadsheets, covering a wide range of deterministic and stochastic models.

Model Building in Mathematical Programming, 4th Edition by H.P. Williams, published by John Wiley. Though it doesn't cover spreadsheet optimization, this book is still valuable for its explanation of model-building approaches, especially if you are building large-scale optimization models. It provides an in-depth treatment of modeling for linear and integer programming problems. It mentions nonlinear models only briefly, but it offers a unique treatment of large-

scale model structure and decomposition methods. It also includes a complete discussion of 24 models drawn from various industries.

Academic References for Risk Solver Platform

The following academic journal articles, written by the developers of the Excel Solver, Premium Solver Pro and Risk 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 the 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.