
Frontline Solvers®
The Leader in **Analytics** for **Spreadsheets** and **the Web**

Table of Contents

Getting the Most from the Excel Solver

Welcome to this eBook, written by Frontline Systems, creators of the **Excel Solver**. We developed, and licensed to Microsoft, every version of Solver in Excel for Windows and Macintosh; it has been included with Excel for over 30 years – since the 1990 launch of Excel 3.0 and Windows 3.0. And we've been offering powerful Solver upgrades for over 30 years.

We really do know Solver better than anyone else.

A Bit of History: How Solver Came to Be

A bit of history: Few people today remember this, but in 1989 Excel was the "upstart challenger" to Lotus 1-2-3, which was "king of the hill" with nearly 90% market share of spreadsheets on IBM PCs. At that time, Excel was popular on Macintosh, but on PCs, most people still used character-based MS-DOS; Windows 2.x was an optional "add-on". IBM was marketing its own new operating system OS/2 as a replacement for MS-DOS; Lotus released 1-2-3/G, its first graphical spreadsheet, only for OS/2. Besides offering a graphical display like Excel's, 1-2-3/G also featured a new "Solver" that Excel from Microsoft, and Quattro Pro from Borland, didn't have. Many people expected 1-2-3/G to crush its competitors.

Microsoft, and soon after Borland, went looking for a way to quickly match 1-2-3/G's Solver – which used mathematical optimization and genetic algorithm / AI methods they hadn't used before. But at Frontline Systems, we had previously created and marketed What-If Solver, an add-in for Lotus 1-2-3 Release 2.x (that "king of the hill" version), which matched 1-2-3/G's Solver and outperformed it on most common problems. (Indeed, recalling our meetings with Lotus, it's likely that 1-2-3/G's Solver was "inspired by" What-If Solver.) We negotiated an "OEM agreement" with Microsoft, went to work, and had the Excel Solver ready for release as part of Excel 3.0 – along with Windows 3.0 – in 1990. Soon thereafter, we completed development of Solver for Excel for Macintosh.

The rest is history: Windows – at 3.0 still an "add-on" to MS-DOS – grew rapidly, displaced OS/2, and ultimately became the operating system shipped with PCs; Excel 3.0 overtook Lotus 1-2-3/G, and gradually replaced 1-2-3 Release 2.x as well; Microsoft combined Excel with Word and other "productivity" apps to form Microsoft Office, which swamped its competitors. Solver, still part of Excel today, has been distributed to about 1.2 billion users worldwide.

So What is Solver – What Does it Do?

When you have multiple **decisions** or actions to take, and there many possible **combinations** of those decisions, Solver helps you find the **best feasible** combination of specific decisions.

Solver Compared to Machine Learning

Recently many companies have invested heavily in software for data mining and machine learning. These tools are valuable, but they only yield predictions (or classifications) – not decisions. To gain a business outcome or payoff, predictions aren't enough – decisions are needed. If you're interested in machine learning using Excel, see the end of this eBook for background on Frontline's Analytic Solver®.

Machine learning (ML) models are useful for uncovering hidden relationships in your data – for example, features of a transaction that may mean it is fraudulent, or sensor data that may mean a machine is about to fail. But they aren't models of a business situation – except in the sense that you (or a data scientist) chose to include certain input data, and exclude other data about the business. It's hard to get anything out of a ML model other than a prediction for a specific case.

Solver, in contrast, helps you make the best decisions. To do this, you will build an explicit model of the business situation. You'll use input data – numbers – from your business, but you'll also use formulas to describe key aspects of how your business works. For example, a SUM function might reflect the fact that you have parts arriving at a warehouse, and the new inventory amount is the sum of inventory you had before, plus the new arrivals. As simple as this sounds, it is missing from machine learning models.

Solver Compared to "What If"

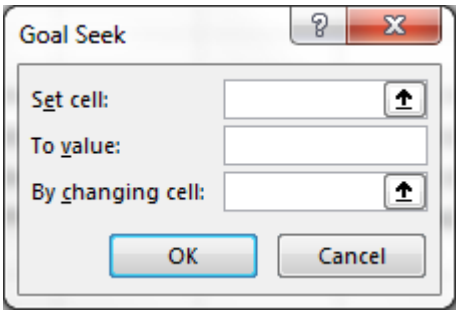
If you've used Excel for anything beyond simply tabulating data, you've probably done some "what-if analysis" – indeed this was the original purpose of an "electronic spreadsheet".

You begin by building a model of a business situation, with formulas (like the SUM function just mentioned), some input cells with numbers you can change, and some output cells of interest. For example, you might use a series of formulas to calculate total sales or net profit, with input cells such as product prices or marketing spending. You then "play what-if", changing the numbers in input cells, and observing what you get for the output cells of interest.

You can view Solver as "super-charged, automated what-if analysis". Solver works by *automatically* changing input cells, observing results, and re-adjusting the input cells. A Solver model *begins* with a what-if model – then you point out to Solver the roles played by certain key cells in the model.

Solver Compared to "Goal Seek"

You might be familiar with the "Goal Seek" feature of Excel – currently found at **What-If Analysis – Goal Seek** on the Data tab. The Goal Seek dialog accepts two single-cell selections, and a numeric value.



The "Set Cell" must be a *formula* cell; you want its calculated value to equal the "To Value" amount as closely as possible. The "By changing cell" is a single input cell – the Set Cell formula must depend, directly or indirectly, on this input cell (otherwise Goal Seek won't do anything). When you click OK, Excel will automatically adjust the input cell value, trying to make the Set Cell calculate to the

number you entered. This usually works, but sometimes Excel will have a hard time getting to the desired value.

> You can view Solver as "super-charged, multi-variable Goal Seek". Where Goal Seek works with just one input cell and one output cell, Solver can handle up to 200 input cells and 101 output cells concurrently. (Analytic Solver®, our Excel Solver extension, handles *millions* of input and output cells concurrently.)

In mathematical terms, where Goal Seek numerically finds a solution to a single equation, Solver finds an *optimal* solution to a set of simultaneous (linear or nonlinear) equations and inequalities – maximizing or minimizing your chosen criterion.

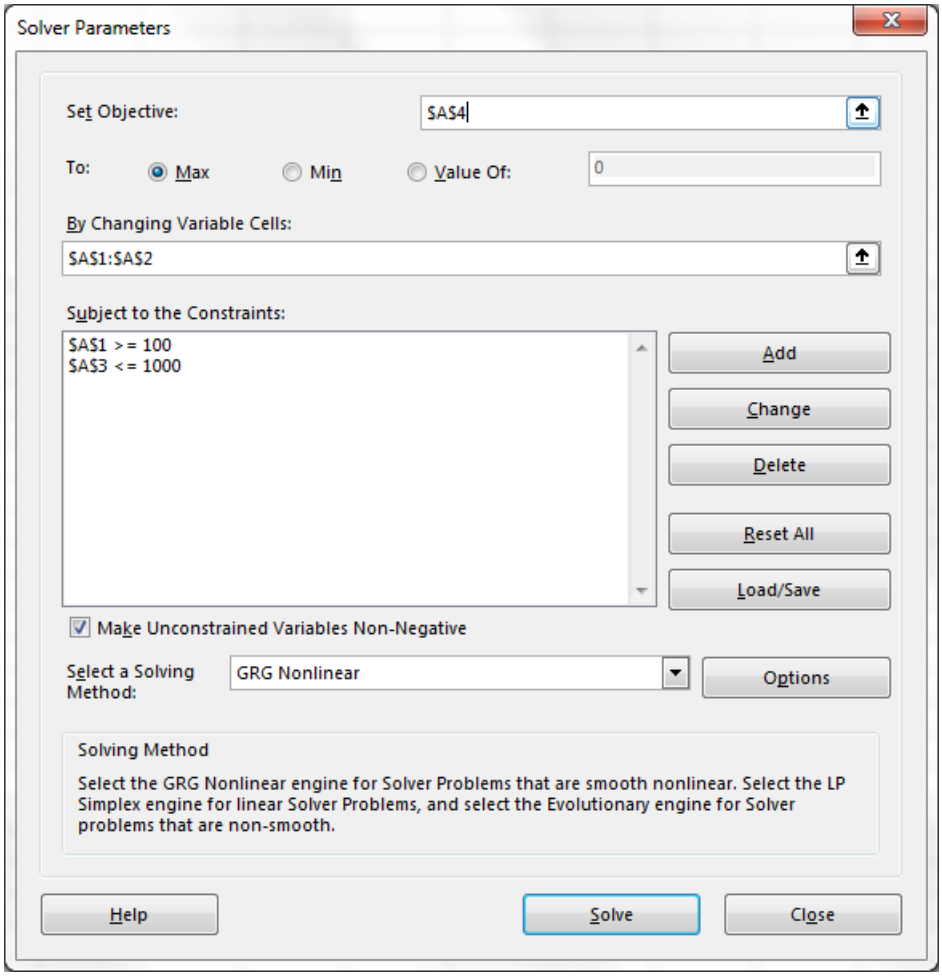
Solver Model and Your Business Situation

Earlier, we said that when you have **multiple** decisions or actions to take, and there many possible **combinations** of those decisions, Solver (unlike Goal Seek) helps you find the **best feasible** combination of specific decisions. So, what about "best" and "feasible"?

You define what **best** means – by calculating an outcome such as profit, throughput, cost or delay time – then telling Solver *that's* the formula cell you want maximize (for profit or throughput) or minimize (for cost or delay time). Solver calls this the **objective** cell.

You also define what **feasible** means, by placing **constraints** or limits on other cells in your model. For example, recall the SUM function that calculates a new inventory amount as the sum of inventory you had before plus the new parts just arrived, in a cell. Suppose you have warehouse space for only 1000 of these parts. So you tell Solver *that* formula cell's value must not exceed 1000 at the solution.

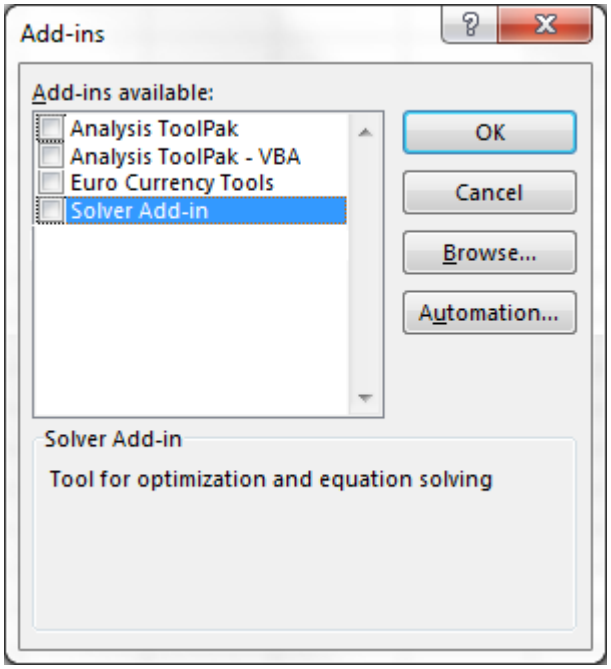
Here's an example of the basic Solver dialog. Cells A1 and A2 are our **decision** variables – cells that Solver can change automatically. Cell A4 calculates the **objective** – the quantity we want to maximize. And we've placed **constraints** on the calculated values of A1 (an input cell) and A3 (an output cell).



Where is the Solver in Excel?

The Solver main dialog pictured above is displayed when you select the **Data** tab, go to the **Analyze** group on the Ribbon, and click on **Solver**. If you *don't see* a Solver choice, this just means that Solver has not [yet] been enabled in your copy of Excel. You need to do this, one time:

In Excel for Macintosh, choose **Tools – Add-Ins**. In Excel for Windows, there are more steps: Choose the **File** tab, then **Options** to display the Excel Options dialog. In this dialog, click **Add-Ins** on the left at the bottom. Then click the button **Go...** next to Manage – Excel Add-Ins. On both Mac and Windows, you should see a dialog roughly like this one (Windows version shown):



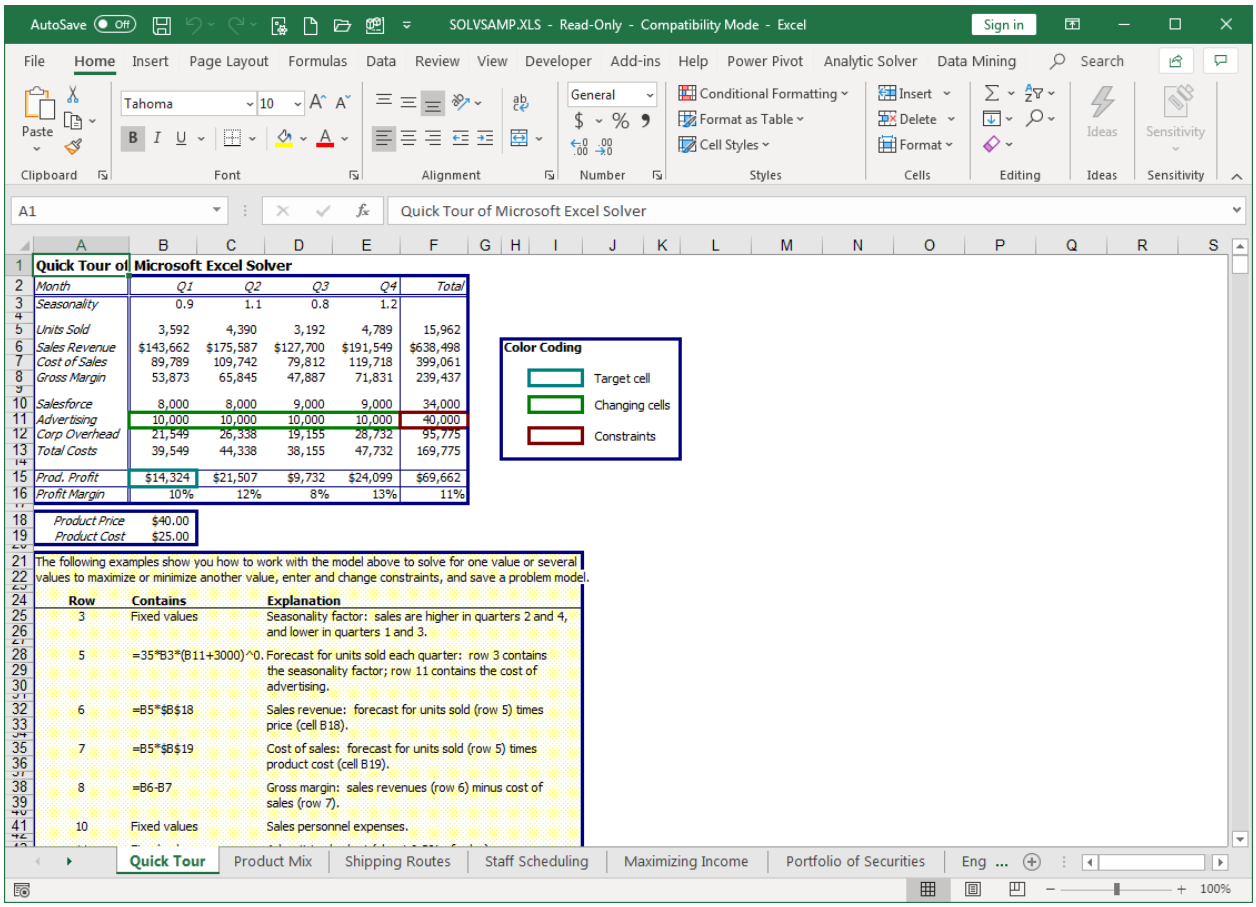
Just click the box next to "Solver Add-In", then click the OK button. When you return to the **Data** tab, and the **Analyze** group, you should now see the **Solver** option.

Where is the Solver Examples Workbook?

There's a useful set of example Solver models in a workbook that ships with Excel. But it's not easily accessible from Excel's menus. You'll have to navigate through the Microsoft Office folders on your hard disk to find it. In modern versions of Excel for Windows, you'll find this workbook at this path (use C:\Program Files (x86) if you have 32-bit Office, C:\Program Files for 64-bit Office):

C:\Program Files\Microsoft Office\root\Office16\SAMPLES\SOLVSAMP.XLS

This workbook is so old that it was saved in XLS format – after opening it, we recommend that you save it – in a more convenient location – as a modern Excel workbook in XLSX format! Here's how it looks:



There are **seven example** Solver models, each on its own worksheet, in this SolvSamp.xls workbook – which we created for Microsoft way back in 1989. All of them still work in modern versions of Excel! The first one, pictured above, is organized as a "Quick Tour" – so you start with an ordinary Excel "what-if" model for an **Advertising Mix** problem. The text below on the worksheet gives steps for "Solving for a Value to Maximize Another Value", "Resetting the Solver Options", "Solving for a Value by Changing Several Values", "Adding a Constraint", "Changing a Constraint", and "Saving a Problem Model".

The **other six** worksheets have Solver models already pre-defined and saved as part of the workbook – so you can just examine the worksheet, click **Solve** on the **Dat**a tab, *view* the filled-in Solver dialog, and click the **Solve** button in this dialog to find an optimal solution. These include (i) **Product Mix** – finding the optimal combination of products to build from limited inventories of parts; (ii) **Shipping Routes** – finding the lowest-cost way to ship products from three plants to five regional warehouses; (iii) **Staff Scheduling** – how to assign employees to work in a park to cover varying demand, at lowest payroll cost; (iv) **Maximizing Income** – deciding how to invest funds in CDs of different maturities, to maximize interest earned while meeting cash needs; (v) **Portfolio of Securities** – finding an "efficient portfolio" of stocks that maximizes portfolio return for a given level of risk; and (vi) **Engineering Design** – finding the right value for a resistor to dissipate charge in an electrical circuit.

Summary: Elements of a Solver Model

Your job is to create **model of the business situation** in Excel that describes, in quantitative terms, the decisions, constraints, and the objective.

Your **objective** is a business outcome – such as profit or throughput (to maximize), or cost or delay time (to minimize) -- represented in Excel by a cell that calculates this quantity. You select this cell in the "Set Objective" box in the main Solver dialog.

An **decision** is something you can **choose** to do (or not do) – say, start up a machine on a production line – represented in Excel by a cell containing 1 (do it) or 0 (don't do it). A decision will often be **how many or how much** of something to do – represented in Excel, for example, by a cell specifying how many pounds of some material to load into the machine. You select all of your decision cells in the "By Changing Variable Cells" box in the main Solver dialog.

A **constraint** is a limit you place on any quantity (an input cell or a calculated result) in the model. Your constraints should reflect the reality of the business situation. You use the Add, Change and Delete buttons in the main Solver dialog to define constraints – always with a quantity of interest (the "left hand side"), a relationship (=, <=, >=), and number or cell containing the limit (the "right hand side").

Some constraints reflect *business limits* – for example, you have only so much space in the warehouse. Other constraints may reflect *business policies* – for example, your policy is to invest no more than 3% of funds in a single stock. Still other constraints may reflect *physical realities* – for example, you can't manufacture a negative number of products (you still need to tell Solver about this).

Solver finds a **solution** – values for the **decisions** – that satisfy the **constraints** or limits – a **feasible** solution – and maximizes or minimizes the **objective** – an **optimal** solution. Again, Solver is most useful when you have **multiple** decisions at once – say which of a dozen machines to start up, and how much of six different kinds of material to be loaded in each one.

Saving and Re-Using Your Solver Models

When you make selections in the Solver dialog, click **Solve** or **Close**, and then **save the workbook**, all your selections are preserved – so the next time you open the workbook, you'll be ready to solve immediately. And it's better than this: Solver models defined in any version of Excel, on Windows, Mac or the Web, from Excel 3.0 in 1990 all the way through 2022 and beyond, "just work" in the latest Excel version. And they work better and faster in Analytic Solver, our powerful Solver upgrade.

What Technical Terms Describe What Solver Does?

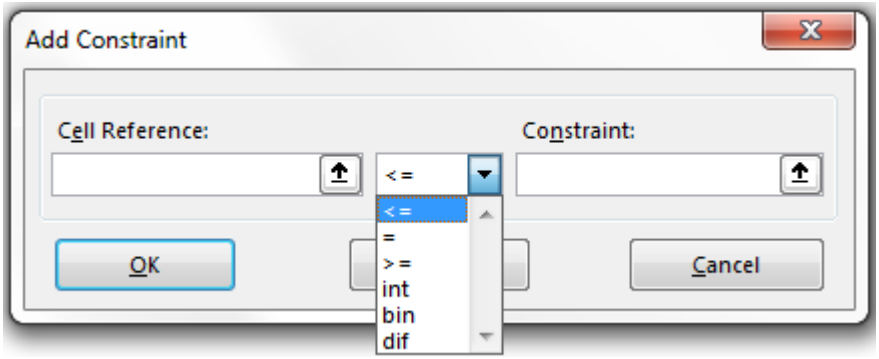
Excel's Add-In dialog, pictured earlier, describes Solver as a tool for "optimization and equation solving". We wrote earlier that Solver finds an *optimal* solution to a set of simultaneous (linear or nonlinear) equations and inequalities – maximizing or minimizing your chosen criterion.

If you define only an objective and no constraints, Solver will perform **unconstrained optimization** – seeking the largest (or smallest) possible value for the objective cell. It's also possible to use Solver *without optimization* – to simply find values for the decision variable cells that satisfy all the constraints. Most commonly though, Solver is used for **constrained optimization** – to maximize or minimize an objective, subject to constraints.

The mathematical **methods** needed to solve an optimization problem depend on the relationships, defined by your formulas, between the output (objective and constraint) cells and the input (decision variable) cells. We'll dig into this topic later in the eBook, but to summarize here, Solver includes three main "solving methods":

- Simplex LP – Solves **linear programming** or *constrained linear optimization problems* (where all formula relationships are linear), using the Simplex method
- GRG Nonlinear – Solves **nonlinear programming** or *constrained nonlinear optimization* problems (where formula relationships are *smooth* linear or nonlinear), using the Generalized Reduced Gradient method
- Evolutionary – Solves **arbitrary** constrained optimization problems (where there are no restrictions on the formula relationships), using Genetic Algorithm and Evolutionary methods

In addition, Solver supports three kinds of "integer restrictions" on decision variables – you add these as constraints, choosing **int**, **bin** or **dif** from the Relation dropdown in the Add Constraint dialog:



Solver uses further specialized mathematical methods from **mixed-integer programming** and **logic programming** to solve models when some or all of the decision variables have these restrictions. And such integer restrictions do make the model harder to solve, usually requiring more solution time.

What Applications and Industries are Best for Solver?

Solver is unusual for the sheer breadth of applications and industries where it can be used to realize **valuable outcomes** on business and elsewhere. At Frontline Systems, where we've sold advanced Solver products to over 10,000 companies and over 200,000 users, we've seen applications in virtually every industry, from manufacturing and transportation to healthcare and entertainment, as well as government, nonprofit, and even personal applications – from designing the hull of an America's Cup yacht, to timely delivering oxygen tanks to hospitals during the Covid-19 pandemic. It's easier to talk about common application scenarios that arise in multiple industries:

Product Mix and Process Decisions

The Product Mix model in the SolvSamp.xls workbook is an example of solving a problem that arises in many industries where products are assembled from parts. But real companies often have hundreds of products and thousands of parts, and they assemble products each day or week, with parts inventory flowing from one time period to the next. A similar problem is called Process Selection, where there are multiple possible sequences of manufacturing steps, with different processes used at each step, taking different amounts of time and material. A Solver model can be used to find the best choices – saving time and money, or increasing throughput.

Transportation and Routing Decisions

The Shipping Routes model in the SolvSamp.xls workbook is an example of solving a problem that arises in every industry where goods are shipped from multiple origins to multiple destinations. In that simple model, there are just 15 shipping routes (3 plants x 5 warehouses) – but for many companies, there are hundreds or even thousands of possible routes, with dozens to hundreds of products to ship, and decisions about what/how many to ship on each route. "What-if" analysis would almost never find the best of all these combinations, but Solver can find the best solution easily – saving lots of money.

Decisions Across Time Periods and Locations

In transshipment models – when for example we ship goods from manufacturing plants to warehouses, then on to distributors/dealers or end customers – our resources "flow" from one location to the next, and we need constraints to track the flows. Similarly, in cash management, inventory management, and other situations like the Maximizing Income model in the SolvSamp.xls workbook, our money or inventory "flows" from one time period to the next, and we need constraints to track the flows. Again it's usually beyond human ability to explore all possible "what-ifs" and track the details, but Solver handles these problems easily and reliably.

Scheduling: People, Vehicles, Materials

The Staff Scheduling model in the SolvSamp.xls workbook is an example of scheduling, a very common use of Solver across many industries, again with clear payoffs. At Frontline Systems, we've seen restaurants schedule servers, hospitals schedule doctors, nurses and operating rooms, airlines schedule their planes and crews, the army schedule troop movements, and a wide range of companies schedule trucks and deliveries. With so many combinations of possible assignments of people, vehicles, rooms and more in different time periods, it's rare that a manual

solution is the optimal one, and Solver can very often be used to identify decisions that save a good deal of time and money.

Blending: Chemicals, Grains, Investment Funds

The Portfolio of Securities model in the SolvSamp.xls workbook is an example of a blending problem: In that case we're allocating different amounts of money to create the least-risky blend of stocks, but the same kind of model can be used to create a mix of grains forming the lowest-cost blend of nutrients for livestock, or create a mix of chemicals forming a plastic with best characteristics. In these applications we often have both many possible combinations of decisions, and inputs that change rapidly from one time period to the next – so we want to solve the model each time the cost of grains changes, or each time our forecast of stock returns and volatility changes. Solver handles these kinds of problem easily.

A Common Denominator: Allocating Scarce Resources

Something all these examples have in common – and a good way to think about your problem, in your own application or industry – is the idea of **allocating scarce resources** to their **best feasible uses**. Your resources might be people, production machines, delivery vehicles, money, raw material or inventory, or the amount of time these things are used; the **decision variables** in your model will reflect all the ways those resources might be **used**. Your **objective** will calculate the effect of using those resources in specific ways, and your **constraints** will need to capture all the real-world limits on what can be done.

Often, thinking through and defining all of the relevant **constraints** is the most challenging work in creating a Solver model. In Frontline Systems' Solver upgrades, we supply literally hundreds of example models, Help and User Guides, and a Constraint Wizard to help in model building.

What Makes it Hard to Get the Solution I Want?

Solver is quite versatile, compared to other optimization tools that – for example – solve only linear programming or LP/MIP (linear/mixed-integer) problems. And it often finds solutions in seconds, where you might have needed hours or days to find that solution (if at all) via "what-if" or Goal Seek. But on some models, Solver seems to run forever (well, for many minutes, or even hours), and the solution it finds may be "good but not best" – not the true optimal solution.

Why is this? It's *all in* ***your formulas*** that relate the objective and constraints to the decision variables. To fully understand this, we're going to need some algebra, and even calculus. But it's possible to understand why, by thinking about Solver's ***search process***, and the "search space" that ***you create***.

What-If and Searching for Solutions

Think about playing "what if" by changing the input cells in your model, and observing the results. You change a number, and see if your objective (say, Profit calculated in a cell) increases or decreases – then you try a different number, based on what you've seen before***. Solver also must do something like this.***

Of course, "what if" gets harder if there are 3 different input cells to change, and *much* harder if there are 20, 50 or 100 input cells. You are *searching* for better choices in 3 dimensions instead of one – or in 20, 50 or 100 dimensions. ***Solver also must search in 20, 50 or 100 dimensions.***

Even though Solver can change input cell values much faster than you can type in changes, this speed advantage pales in comparison to the expanded range of values to explore in 20, 50 or 100 dimensions (mathematicians call this the ***"curse of dimensionality"***).

But aren't modern computers fast enough for this? Well, suppose your model covers 5 years by month (60 periods), and you have just one simple yes/no decision to make each month. The number of combinations of these yes/no decisions is 2^{60} – that's 1.15×10^{18} or 1.15E+18 in Excel.

OK, so how large is this? Astronomers estimate that the number of ***seconds since the Universe began*** is 4.57E+17 … so, if you started solving at the time of the Big Bang, you might be *one-tenth done* with this problem by now. There's *no way* to beat the curse of dimensionality with just faster computers.

Beating the Curse of Dimensionality

To find solutions quickly, Solver must do something besides just trying all possible combinations of values. And to do this, Solver must know (or assume) something about how your model behaves. What ***you*** can do is to use formulas in your model that satisfy those assumptions, enabling much faster search.

The three **Solving Methods** – Simplex LP, GRG Nonlinear, and Evolutionary – make different **assumptions** about how your model behaves. These assumptions allow Solver to be **selective** about which decision variables to change, and by how much – enabling it to find solutions **much faster**.

- The **Simplex LP method** assumes that the formulas that calculate your objective and constraints are all **linear functions** of the decision variables. When this is true, Solver can quickly find the ***globally optimal*** (very best feasible) solution.

- The **GRG Nonlinear method** assumes that the formulas that calculate your objective and constraints are all ***smooth* linear or nonlinear functions** of the decision variables. When this is true, Solver must do more work (taking more time), but it can normally find a ***locally optimal*** solution – the *best feasible* solution in the region around its *starting point*.

- The **Evolutionary method** makes **no assumptions** about the formulas that calculate your objective and constraints, as functions of the decision variables. When this method is used, Solver may take *much* more time – and it can only find a **"good"** solution, improved from its starting point – not an optimal solution.

Some quick comments:

- In the GRG Nonlinear method, **"smooth"** means mathematically *differentiable* – the first partial derivative or *rate-of-change* of each function, with respect each decision variable, is *defined*. As we'll see, functions like IF, CHOOSE or VLOOKUP are "non-smooth" or (worse) discontinuous.

- The GRG Nonlinear method is the ***default choice*** in the "Select a Solving Method" dropdown. This is because it performs pretty well on linear models, performs best on smooth nonlinear models, and still performs reasonably well on non-smooth or arbitrary models.

- When you use the Simplex LP method, the starting values of the decision variable cells **don't matter**: As long as the model is truly linear, this method will find the globally optimal solution. For the GRG Nonlinear and Evolutionary methods, the starting values of the decision variable cells **do matter** (the Multistart option for the GRG Nonlinear method can help with this).

- Since the basic Excel Solver must rely on Excel to calculate your formulas – and Excel doesn't "do algebra", it just calculates – the Solving Methods make ***assumptions*** that your formulas are linear, or smooth linear or nonlinear. These methods do use numerical tests to try to verify that the assumptions are valid, but Solver is really relying on you to use the right formulas.

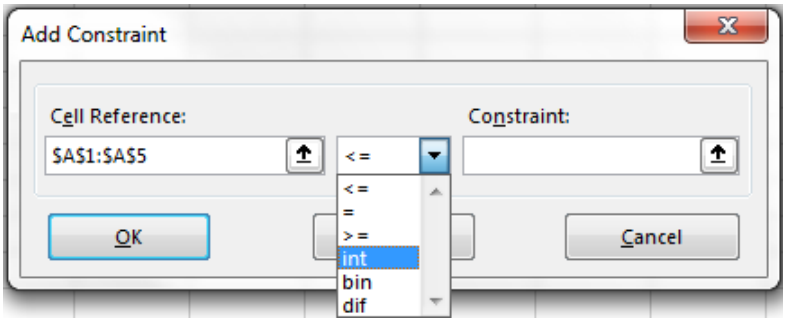
In Frontline's Analytic Solver upgrade, our software actually "does the algebra" and determines for sure whether your model meets the assumptions – so it can *automatically* select the best Solving Method or "Engine", from more than a dozen choices.

The idea of highly ***selective*** search, based on the behavior of your objective and constraints as functions of the decision variables, is ***the key*** to beating the "curse of dimensionality" in Solver's search. As we'll explain in the next section, having ***convex constraints*** is the key to finding **feasible** solutions, and having a ***convex objective*** is the key to finding a **globally optimal** solution. All ***linear*** functions are ***convex***.

Using Non-Negativity and Integer Variables Effectively

By default, decision variable cells can have any value – positive, negative, zero, integer (whole number) or fractional – and Solver may well try all these values in its search. But quite often, some of these values won't make sense for the decisions that you are representing. For example, it probably makes no sense to manufacture a negative number of products, or schedule a negative number of trucks. This is so common that Solver includes a check box "Make Unconstrained Variables Non-Negative" in its main dialog, which is ***checked by default*** – implying a constraint A1 >= 0 on every decision variable cell. But if you define a different constraint explicitly, such as A1 >= 5 or A1 >= -5, that constraint *overrides* the effect of the "Make Unconstrained Variables Non-Negative" option.

It also may make no sense to schedule a fractional number of trucks – you can't actually send 2.7 trucks on their way, it must be either 2 or 3. To specify that Solver should only allow an integer or whole number of trucks, you can add a constraint and select the "int" option from the dropdown list, as shown below. This constraint will appear as "$A$1:$A$5 = Integer" in the list of constraints in the main dialog.



Bear in mind that, to solve models with integer constraints, Solver must use specialized mathematical methods from **mixed-integer programming** and **logic programming** to find solutions, and this does make the model harder to solve, usually requiring much more solution time. If you're scheduling 300 trucks instead of 3, the difference between 300 and 300.7 is quite small, and could be settled by rounding – so you might skip the integer constraint.

Binary Integer Variables vs. IF Functions

You can also specify that a decision variable must not only be integer, it must be either 0 or 1 at the solution. This is called a **binary integer variable** – you simply choose "bin" rather than "int" from the dropdown list above. Such variables are the ***preferred way*** to represent **yes/no decisions** – by convention 1 = Yes and 0 = No.

Even though the use of binary integer variables makes the problem harder to solve, requiring more solution time, this is ***much better*** then trying to model yes/no decisions – or the consequences of these decisions – with IF functions in Excel. With IF functions "buried" in the middle of your formulas, Solver cannot perform its "what-if" *search* effectively: As we'll explain below, the IF functions make your model ***non-***

smooth, ***preventing*** use of the much faster Simplex LP and GRG Nonlinear Solving Methods, and making things difficult even for the Evolutionary Solving Method.

With ***explicitly defined*** binary integer variables, Solver "knows" that it has to explore the yes/no alternatives, by testing 1 and 0 values in those variable cells. And the methods from **mixed-integer programming** that Solver uses are a big improvement over a search through all possible combinations of variable values – minimizing the effect of the "curse of dimensionality".

Frontline's Analytic Solver upgrade goes much, much further to solve problems with binary integer and general integer variables, in all of its Solver "Engines". Analytic Solver can be hundreds or thousands of times faster on these problems than the basic Solver in Excel – and can fully match the performance of non-Excel-based optimizers that might require you to define your model in code. The world's best Solvers for these problems, such as the Gurobi and Xpress Solvers, work seamlessly in Analytic Solver.

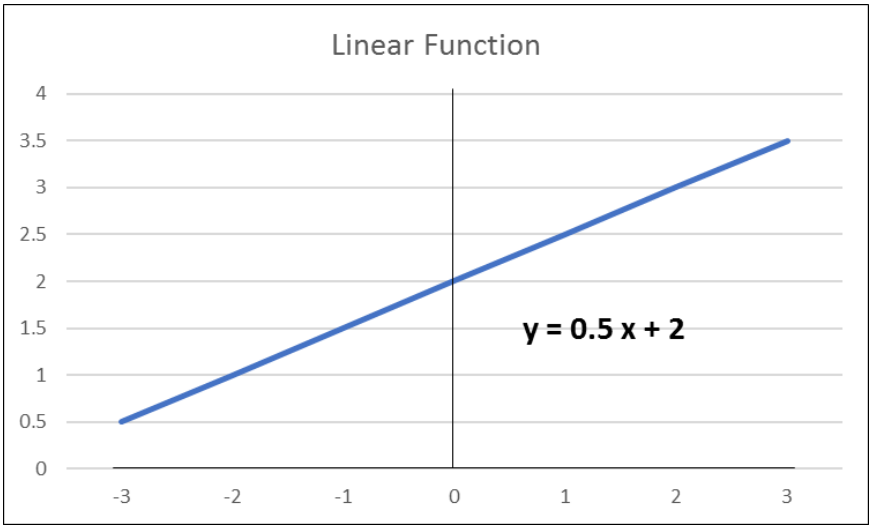
(Advanced) How Do the "Solving Methods" Really Work?

Even if it's been a long time since you studied algebra and calculus (if you did), you can learn to design your Excel model to get the best solutions. You just have to pay attention to the ***formulas*** you use, when those formulas ***depend on the decision variables*** in your model. (You can use any formulas you want *without* affecting Solver, if those formulas *don't depend* on the decision variables.)

Linear Functions and the Simplex LP Method

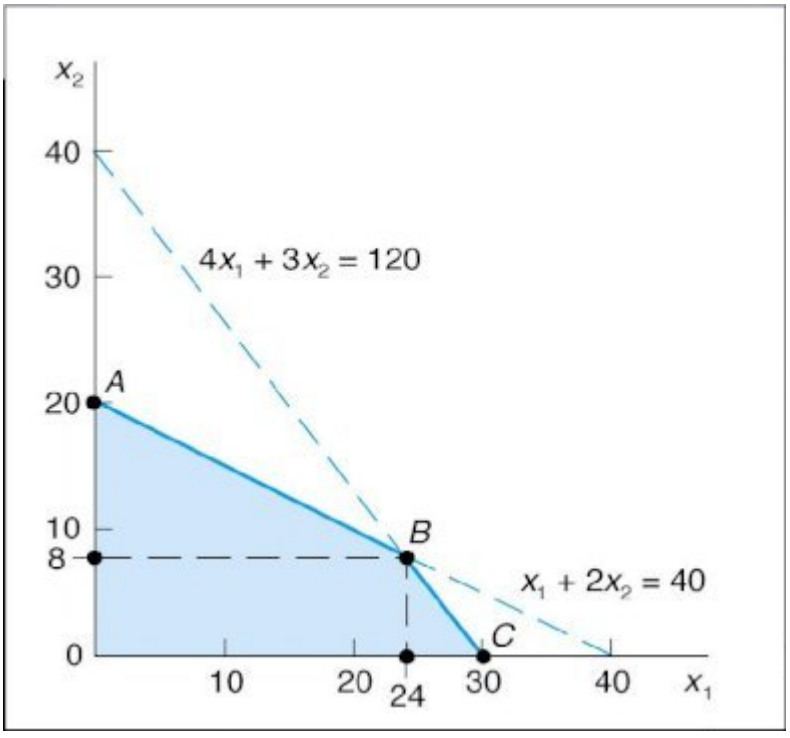
Let's start with the simplest (and best) case, where your objective and constraints are ***linear*** functions of the decision variables. It turns out that most common business problems **can** be expressed this way – possibly using some integer variables (covered later) instead of functions like IF and CHOOSE.

As the name implies, the graph of a linear function such as y = f(x) is always a ***straight line***. And the formula for f(x) always has the form **y = m*x + b**: m is called the ***slope*** – the rate-of-change of y versus x – and b is called the **y-intercept** – the value of y when x is 0. In the example below, y = 0.5 * x + 2.



For Solver, we care about **y = f(x) = m*x + b** when y is an **objective** or **constraint**, and x is (or depends on) a **decision variable**. The key property of a linear function is that m and b are *constant in the problem* – they *don't* depend on any decision variable. We can generalize this to linear functions of two or more decision variables, with different m-values (called *coefficients*) for each variable: $y = f(x_1, x_2) = m_1*x_1 + m_2*x_2 + b$. Again it's **key** that m_1, m_2 and b are *constant in the problem*: they could be calculated by other formulas, as long as those formulas don't depend on any decision variable.

Why is this important? When the objective and all constraints are linear, Solver can be ***very selective*** about the "what-if" values it tests for decision variables. Below is a simple example.



This model has a linear objective such as $40*x_1 + 50*x_2$ to maximize, and two linear constraints $4*x_1 + 3*x_2 \le 120$ and $1*x_1 + 2*x_2 \le 40$, plus it assumes $x_1 \ge 0$ and $x_2 \ge 0$ (the variables are non-negative). The **feasible region**, where the constraints and non-negative bounds are satisfied, is the light blue shaded area: This is called a *polyhedron*, and in higher dimensions (i.e. more variables) it's called a *polytope*.

Instead of testing **all possible** values of x_1 and x_2 between 0 and 40, Solver's "what-if" search will test only the **three points** labeled A, B and C! How does Solver know it can do this?

George Dantzig discovered why, and invented the "Simplex method of linear programming" in 1947. (The name "linear programming" derives from the practice at the time of "planning and programming" – it's unrelated to "programming" in the sense of coding that we have today.) Dantzig realized that, when all the constraints are linear and all are satisfied, they will form a closed ***convex region*** (for "convex" think "no nooks and crannies that must be searched"). And a linear objective will always find its feasible maximum at region's boundary, in fact at a "corner point" where constraints intersect. Remember: ***Linear functions are always convex***.

Note that a linear function $y = f(x_1, x_2)$ can *always* be written in the form $y = m_1*x_1 + m_2*x_2 + b$. When maximizing or minimizing the objective, b doesn't matter, and for constraints we can always write for example $m_1*x_1 + m_2*x_2 \leq$ *constant - b*. Hence in Excel, you can use SUMPRODUCT (*m-range, x-range*) to calculate *any* linear function, as long as the *m-range* cells are constant in the problem. You can also use formulas like =*constant*variable* in cells, and sum these with =SUM (cell-range) for a linear function.

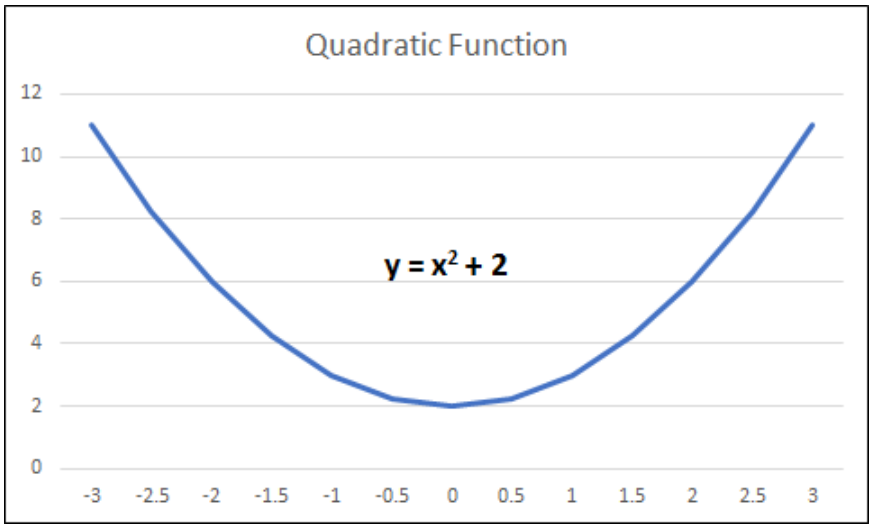
Again, this idea of highly selective search, based on the behavior of your objective and constraints as functions of the decision variables, is ***the key*** to beating the "curse of dimensionality" in Solver's search. Having ***convex constraints*** – "no nooks and crannies" – is the key to finding **feasible** solutions, and having a ***convex objective*** – "no bends up and down" – is the key to finding a **globally optimal** solution.

Smooth Functions and the GRG Nonlinear Method

So, what if you *cannot* define your objective or constraints using simple linear functions? The next level of generality is to use ***smooth*** functions. Recall that a linear function's graph is always a straight line, and it can be written as **y = mx + b** where m is called the ***slope*** or rate-of-change of y versus x. In calculus, m is called the ***derivative*** of y with respect to x. In a formula like $y = m_1*x_1 + m_2*x_2 + b$ with two variables, m_1 is the ***partial derivative*** of y with respect to x_1, and m_2 is the partial derivative of y with respect to x_2.

Note that "the graph is a straight line" and "the function has a constant slope" (a constant derivative) mean the same thing! To generalize, we're now going to allow functions whose graph is ***curved*** – which means the ***slope can change*** as x changes. A function where the slope changes is called smooth as long as the slope is always ***defined*** – you can calculate it, without, say, dividing by zero. (If you think ahead, you might realize why using IF, CHOOSE, etc. introduces a divide-by-zero problem!)

Here's an example: **$y = x^2 + 2$** or **y = x*x + 2**. This is a ***quadratic*** function, perhaps the simplest example of a ***nonlinear*** function. Note that the line being graphed has **no breaks** – in calculus, this is called a *continuous* function. When the graph of the ***derivative*** of this function has no breaks, it's called a ***smooth*** function. In the Excel Solver, the GRG Nonlinear solving method is designed to handle models where the objective and constraints are all smooth functions of the decision variables.



Why is this important? When the objective and all constraints are smooth functions, Solver (using the GRG Nonlinear method) can be ***fairly selective*** about the "what-if" values it tests for decision variables.

How? In the above example, Solver can set x to 0 and calculate y, set x to 0.5 (or even closer, say 0.1 or 0.01) and calculate y, then compute a ***rate-of-change*** or "rise over run" (***derivative***) of y with respect to x. If that derivative is ***positive***, the line is sloping up – so if we're maximizing y, Solver should go further in that direction! But if the rate-of-change is ***negative***, then the line is sloping down, so Solver is better off searching in the opposite direction, or some other direction.

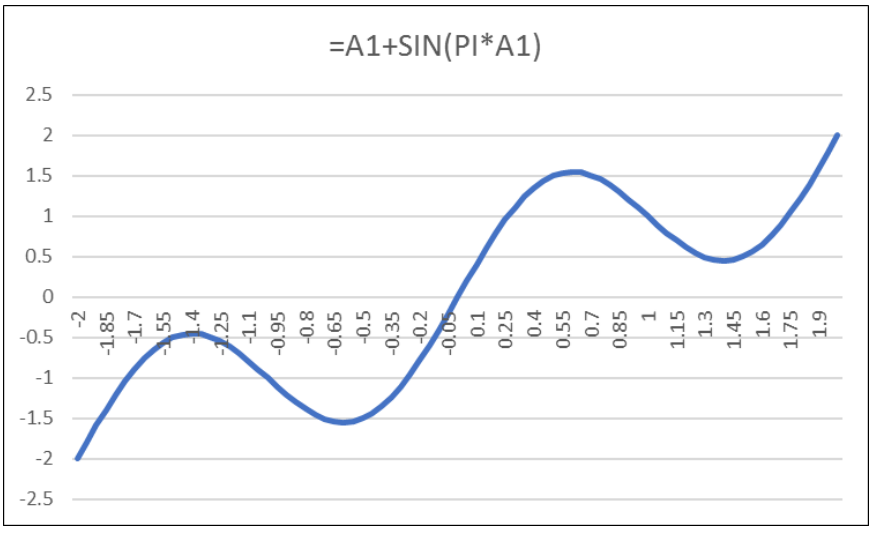
When the function depends on many decision variables, Solver can slightly adjust each of those variable values, and compute a rate-of-change with respect to each variable. Then it can choose – for example – to focus on the variable that has the ***greatest positive*** rate of change.

While "following the rate-of-change" – the ***partial derivatives*** – is not nearly as quick as "jumping to the corner points" when we have all linear functions, it's a *whole lot better* than trying all possible combinations of values for different variables – and it is a way to beat the "curse of dimensionality".

Convex Objectives and Globally vs. Locally Optimal Solutions

The bowl-shaped function **y = x² + 2** pictured earlier is smooth nonlinear – it's not linear, but it is ***convex***. Remember that earlier we said "***convex – no bends up and down***" – that's the key idea. You can see right away that we can "slide down either side" to the minimum at x=0 (where y=2). If this is the objective and we've chosen to minimize it, this is the ***globally optimal*** solution.

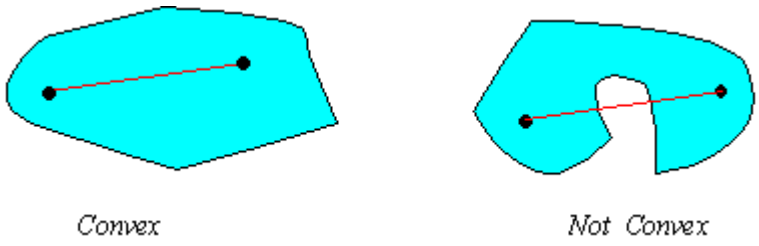
Below is an example of a smooth nonlinear, ***non-convex*** function – it *does* have "bends up and down". (Bear in mind that Solver doesn't have the whole chart to examine, unless it tests every possible value for A1 and saves the results.) We can compute a rate-of-change, and move in the direction of a positive rate-of-change – but what if we start at x = -1.1 or x = +0.7, and we see that the rate-of-change is negative? In this situation, the GRG Nonlinear method (without using the Multistart option) will stop at about x = -1.4 or x = +0.55, finding only a ***locally optimal*** solution – one where there's no better solution "nearby", but there *could* be a better solution some distance away.



Convex Constraints versus "Nooks and Crannies"

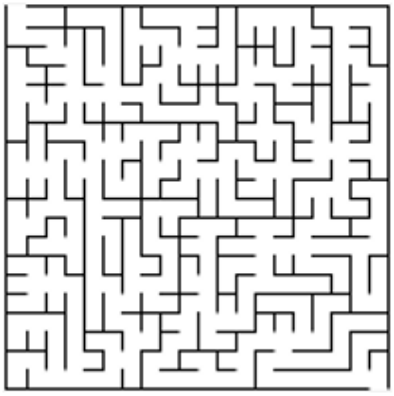
When talking about the Simplex LP method, we said that "when all the constraints are linear and all are satisfied, they will form a closed ***convex region***". And we said for "convex" think "no nooks and crannies that must be searched". This is a ***very*** important idea in optimization: Convexity also generalizes from linear functions to smooth nonlinear functions for the ***constraints*** – where it is even ***more important***.

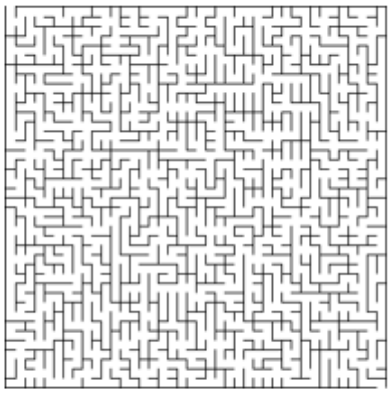
A region is ***convex*** if, when you draw a straight line from any point in the region, to any other point in the region (called a chord), that line lies ***entirely inside*** the region. A picture should clarify things:



This picture is just in ***two*** dimensions, but let's imagine you are standing inside a building, shaped as above (overhead view). Suddenly there's a fire, the building is filling with smoke, and you have to dash to the ***lowest*** point in the building (assume you can 'feel" downward slopes). In the building on the left, this is pretty easy: you just run in a downhill direction. In the building on the right, it ***not*** so easy – the low point might be on one side or the other, so you have to run back and forth between both.

If you followed this exercise, you have an idea what it's like to be Solver, searching inside a good (convex) versus not-so-good (non convex) region defined by the constraints. And if the model includes ***non-smooth*** constraints, for example with IF functions (see below), things get more like a maze.



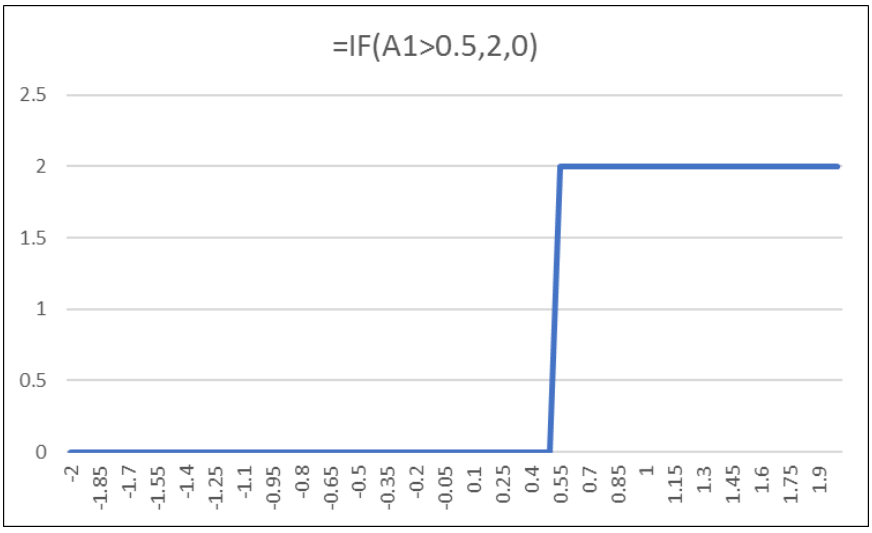


If you now increase the ***number*** of constraints – from 20 x 20 on the left to 40 x 40 on the right – you might be able to envision why even a fast Solver can have a hard time searching for solutions.

Non-Smooth Functions and the Evolutionary Method

We said that a function where the slope changes is called ***smooth*** of the slope is always ***defined*** – you can calculate it, without, say, dividing by zero. The GRG Nonlinear Solving Method ***follows the slope***, in order to explore possible solutions ***selectively***, instead of trying all combinations of variable values.

But what happens if a function is not smooth? Let's look at a common example of a non-smooth (in fact discontinuous) function – the ever-popular IF function:



Excel draws this chart with an almost-vertical "jump" at 0.5 – in the actual math, it *is* a vertical jump: a rise of 0.5 for a run of 0, so the rate-of-change calculation is 0.5 / 0 – ***undefined*** or #DIV/0! Perhaps you can see how this causes problems for the GRG Nonlinear method: When it's exactly at 0.5, how does Solver decide whether to adjust A1 up or down? If we're maximizing, in *this case* up is the right move, but in many other cases down would be best.

For models with functions like IF, CHOOSE and VLOOKUP, we can use the **Evolutionary** Solving Method – but there are trade-offs: Solver will take more time to find a solution, and it won't be able to verify that the solution is even locally optimal – only that it is better than the starting point and other possible solutions found during its search. This method *will* struggle with the "curse of dimensionality", but it will do significantly better than just trying all possible combinations of decision variable values.

One key difference in the Evolutionary method is that it always maintains a ***population*** of candidate solutions, rather than a "single best solution found so far". Its next search move may start from any of the points in the population; hence it is less likely to become "trapped in a region with a local optimum". (The Multistart option does something similar for the GRG Nonlinear method.)

The Evolutionary method is based on Genetic and Evolutionary Algorithms. As its next search move, it will sometimes use a random ***mutation*** (change of value) in one decision variable – by analogy with gene mutations in natural evolution. At other times it will use a ***crossover***: combining a subset of the decision variable values from one member of the population with other decision variable values from *another* member of the population – by analogy with sexual reproduction in natural evolution. And it will repeatedly perform a ***selection*** step, where improved solutions (in constraint feasibility or objective value) are kept, and one or more "worse" members of the population are eliminated to make room.

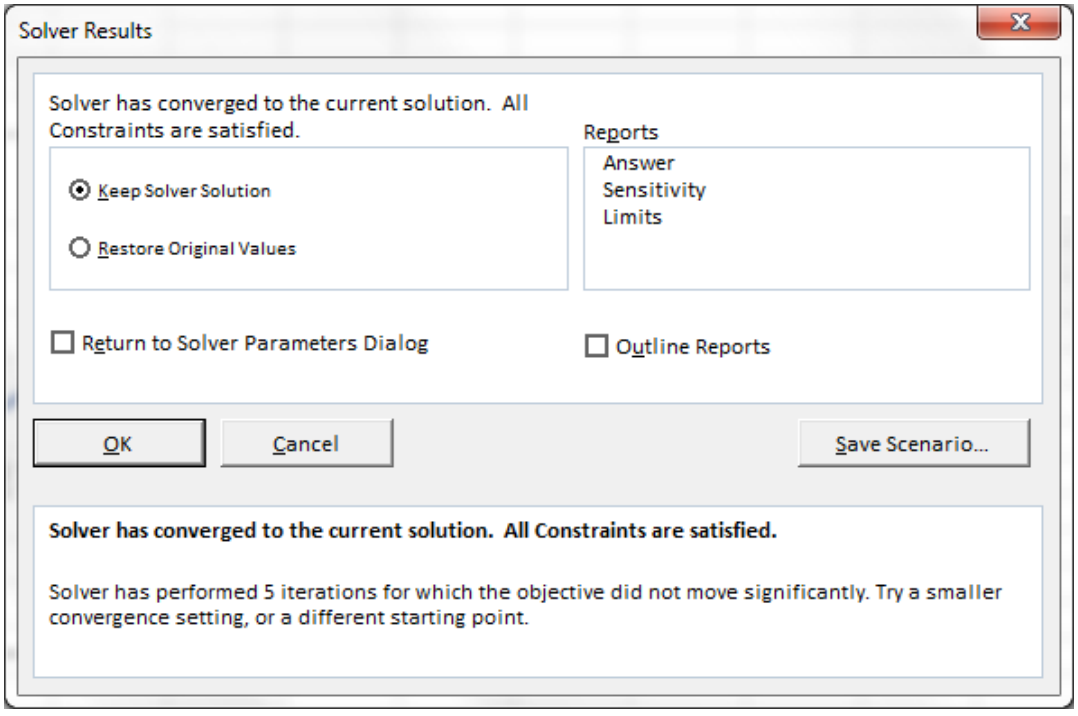
The Evolutionary method uses many enhancements of these basic methods – even borrowing at times from the Simplex LP and GRG Nonlinear methods to handle linear or smooth functions. The overall result is a method that is "robust but slow": If you choose the Evolutionary method for a model with all linear functions, it will take much more time than the Simplex LP method, and it may or may not find the globally optimal solution, that the Simplex method would quickly find. So it pays to chose the right Solving Method for the formulas in your model – and where possible, simplify the formulas!

How Can I Avoid Some Common Mistakes?

At Frontline Systems, we've helped hundreds of thousands of users build models, over more than 30 years – we've learned a lot about common mistakes that users make, and we've advised countless users on how to make their models solve at all, or solve faster. Let's first deal with some common mistakes:

Not Paying Attention to the Solver Results Message

We've often dealt with cases where a user simply reports that "Solver did not work". On probing, we learn that they mean "Solver did not find the solution I was expecting". When we ask what message appeared in the Solver Results dialog … they didn't notice. Belong is an example dialog that appears when the Excel Solver is applied to the simple function =A1+SIN(PI*A1) shown earlier:



Solver is telling you that it "converged to the current solution", and that constraints (if any) are satisfied – but it *doesn't* say that the ***optimality conditions*** are satisfied.

The message even includes a pretty good suggestion to "Try a smaller Convergence option setting, or a different staring point". Do this first!

Problems with Poorly Scaled Models

Many unexpected Solver Result messages are due to a ***poorly scaled*** model. This is a model that computes values of the objective, constraints, or intermediate results that differ by many factors of 10 or ***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."

Poor scaling in a large, complex model can cause Solver to return messages such as "Solver could not find a feasible solution," "Solver could not improve the current solution," or even "The linearity conditions are not satisfied," with results that are suboptimal or otherwise very different from your expectations. Everything may look OK at the initial values of the variables, but when the Solver explores solutions with very large or small values for the variables, numerical errors will be greatly magnified.

Solver's option **Use Automatic Scaling** is checked by default, and this often helps minimize the ill effects of poor scaling. But this can only help with 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.

Effect of the Integer Optimality Option

For models with integer constraints on variables, users 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 Solver stops with the message "Solver found a solution" because it found a solution whose objective is within **x%** of the true integer optimal solution – where **x** is the **Integer Optimality** option in the Solver Options dialog. So, you may know of or be able to discover an integer solution that is even closer to, or is equal to the true integer optimal solution.

(In similar cases, our Analytic Solver upgrade displays a message "Solver found an integer solution within tolerance," to avoid confusion.)

The "flip side" of this issue is that 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. For this reason, the default setting for the Integer Optimality option is 1% -- striking a balance between solution time versus stopping short of the true optimal solution. But you may get better – or faster – results for your model by adjusting this option value.

(Advanced) How Can I Design my Model to Get the Best Answers?

As you probably realize by now, if you can define your model so its objective and constraints are all linear functions of the decision variables, you're "golden" – you can use the Simplex LP method to quickly find the globally optimal solution to your model, a linear programming problem. You can do this in many cases we described earlier: Product mix and process decisions, transportation and routing decisions, models involving time periods and locations, and many scheduling and blending problems.

Some classic problems, like portfolio optimization using the Markowitz or Sharpe models, some chemical process problems, hydroelectric power and others, intrinsically involve nonlinear – but often smooth nonlinear – functions, like the electricity generated by water turbine. If you model these problems carefully, you can usually get very good results from the GRG Nonlinear method.

"Linearizing" Some Common Constraints

Below are two 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*** 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 Nonlinear method 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 LP method 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 MAX (and MIN) are ***non-smooth*** functions, so you'd need to use at least the GRG Nonlinear method, and perhaps the Evolutionary method 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 LP method to find a solution.

"Linearizing" Non-Smooth Functions

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.

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 <= M*y, 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 at all 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 LP method. This is called a fixed-charge constraint.

Since this situation is quite common, our Analytic Solver upgrade supports a dropdown option **semi** that defines a decision variable as a "semi-continuous" variable. At the optimal solution, a semi-continuous variable must either be zero, or must lie within a specified continuous range. This avoids the need for the "Big M constraint" and is usually even more efficient than the approach 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 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 <= M*y 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, suppose 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.

Replacing IF Functions

Consider the formula =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 have use the Evolutionary Solving Method. Instead, you can avoid the IF function and solve the problem with the GRG Nonlinear method – or even the Simplex LP method – 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):

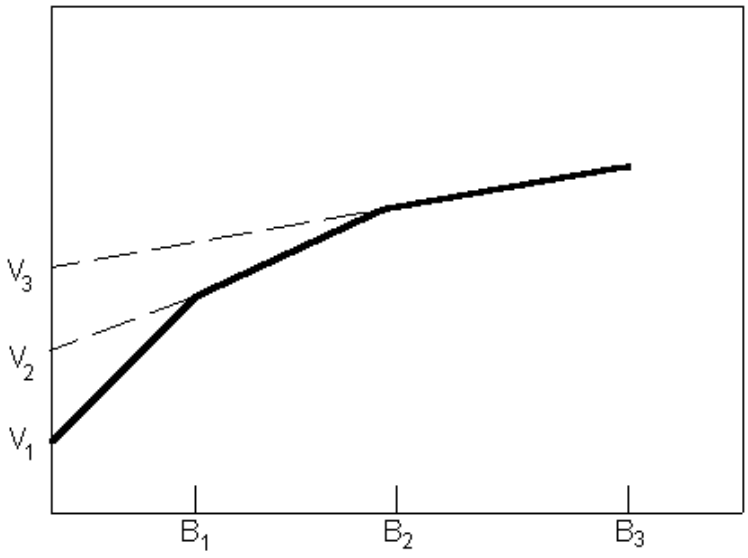
f(x) – F <= M*y
F – f(x) <= M*(1-y)
g(x) – G <= M*(1-y)

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 the Simplex LP method.

CHOOSE, VLOOKUP and Piecewise-Linear Functions

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

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



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

$V_2 = V_1 + c_1*B_1 - c_2*B_1$

$V_3 = V_2 + c_2*B_2 - c_3*B_2$

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

Minimize $V_1*y_1 + V_2*y_2 + V_3*y_3 + c_1*x_1 + c_2*x_2 + c_3*x_3$

Subject to $x_1 \le B_1*y_1$, $x_2 \le B_2*y_2$, $x_3 \le B_3*y_3$

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

$y_1 + y_2 + y_3 \le 1$

$x_1 \le B_1*y_2$

$x_2 \le B_2*y_3$

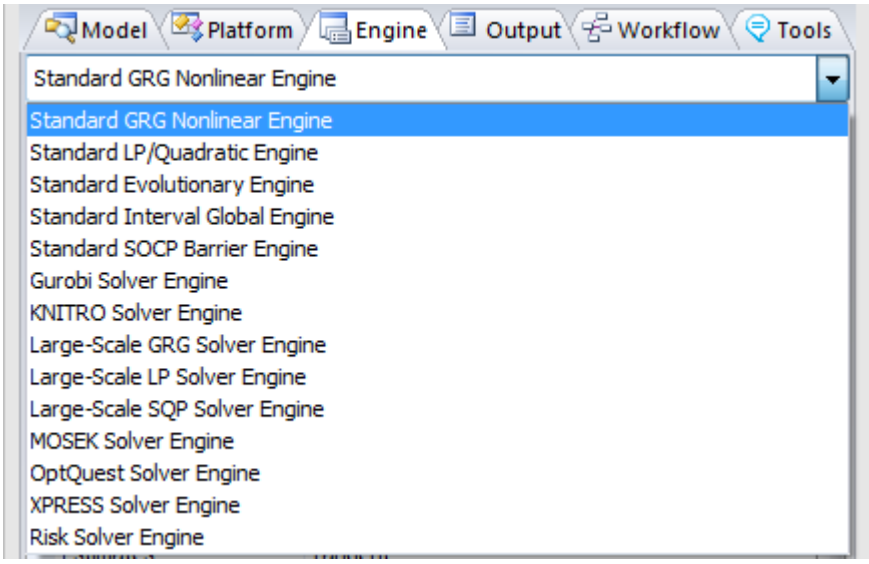
This approach is called a "piecewise-linear" function. It can be used in place of a CHOOSE or LOOKUP function, and it results in a linear integer model instead of a difficult-to-solve non-smooth model. Piecewise-linear functions can also be used to approximate a smooth nonlinear function, by using line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

How Can I Solve More Ambitious Models?

While this eBook has focused on the Excel Solver, we've referred a few times to our **Analytic Solver** upgrade – the culmination of over 30 years of continuous software development, millions of lines of code, and well over $20 million invested. We really believe it's an "undiscovered gem" for many users, and for serious commercial and industrial modeling projects, it's "worth its weight in gold".

"Scaling Up" Your Solver Model

As we noted earlier, Solver models defined in any version of Excel, on Windows, Mac or the Web, from Excel 3.0 in 1990 all the way through 2022 and beyond, "just work" – better and faster – in Analytic Solver. As you probably know, Solver in Excel has model size limits of 200 decision variables and 100 general constraints (in addition to bounds and integer constraints on variables); with Analytic Solver, you can easily "scale up" your model's size, to thousands, tens of thousands or even millions of decision variables and constraints. Analytic Solver integrates the world's best optimizers from other vendors, giving you a choice of over a dozen Solver "Engines" – see the dropdown list pictured below.



You can also solve new ***kinds*** of optimization problems using Analytic Solver. A specialized example is SOCP or "second order cone programming" problems. But a wide class of problems that are of great interest to almost everyone, are optimization problems with uncertainty.

Optimization Problems with Uncertainty

In conventional optimization models, the type that Solver – and most other optimization software – handle, there is ***no uncertainty*** – it's assumed that all the data in the model, such as constraint limits and cost coefficients, are quite ***accurate*** and won't ***change*** in the time frame relevant for the model.

When you think about it, this is almost ***never*** true – it's just a matter of *how* inaccurate or *how* subject to change the numbers really are. When uncertainty is a significant factor, you need to represent that uncertainty ***explicitly*** in your model. And ***new and different*** optimization methods are needed to solve such models. Indeed, even the meaning of an "optimal solution" is a little different under uncertainty. Analytic Solver has ***extremely powerful capabilities*** to deal with all these issues.

First, Analytic Solver has comprehensive support for designing and running **Monte Carlo simulation** models, with more than 90 probability distribution functions, 90 statistics and risk measures, correlation matrices and copulas to relate multiple uncertain variables, automatic fitting of distributions to your historical data, multiple random number generators and stratified sampling methods, multiple parameterized simulations, and a rich set of charts and graphs. Uniquely at present, Analytic Solver has comprehensive support (including automatic fitting) for the new family of Metalog distributions, which are capable of replacing most "classical" probability distributions.

Second, Analytic Solver supports ***multiple methods for solving*** optimization problems where the objective and/or constraints depend on uncertainty. Other software products use only so-called ***simulation optimization***; Analytic Solver has by far the *fastest* simulation optimization, but also supports ***robust optimization*** and ***stochastic linear programming*** methods – which are capable of solving much larger models than simulation optimization. Further, Analytic Solver enables you to define decisions that are conditional on uncertain outcomes – such ***recourse decision variables*** are simply missing from simulation optimization, but they're essential for real-world modeling of important decisions that must be made under uncertain conditions. You can see this for yourself in a free trial of Analytic Solver.

Forecasting, Data Mining and Machine Learning

At the very beginning of this eBook, we compared what Solver does to machine learning – the "hot method" of the last ten years – and we mentioned "if you're interested in machine learning using Excel, see the end of this eBook for background on Frontline's Analytic Solver". So here we are!

In addition to its rich support for conventional optimization, Monte Carlo simulation, and optimization with uncertainty, Analytic Solver includes deep and comprehensive support for **forecasting, data mining (and text mining) and machine learnin**g. This ranges from exponential smoothing and ARIMA models for time series forecasting, to supervised machine learning: linear and logistic regression, classification and regression trees (CART), multi-layer neural networks, k-nearest neighbors, naïve Bayes, ensembles of multiple models, and association rules – plus unsupervised methods such as k-means and hierarchical clustering, principal components, and automated feature selection.

You can build and run multi-stage "data science workflows" that transform data, train a machine learning model, and apply it to new data; you can even use a ***Find Best Model*** option to automate the entire process of testing and comparing

machine learning methods against your data, then choosing the best-performing model. You can export your model in industry-standard **PMML** format – all in Excel!

Business Rules, Decision Tables and Decision Trees

There's more – Analytic Solver also includes comprehensive support for rule-based and sequential decisions, using decision tables and multi-stage decision trees with expected values and utility functions. The rules that make up decision tables are expressed in industry-standard DMN (Decision Model and Notation) – in fact Analytic Solver's conformance level to the standard is higher than some other "business rule only" products! And yes – you can use decision trees and decision tables in your simulation and even optimization models. Analytic Solver is truly an "all in one" solution for predictive, prescriptive, and decision analytics.

RASON Cloud Service: Deploying Your Model for Use by Others

There's one more important piece to the Analytic Solver story – which goes well beyond desktop Excel (and Excel for the Web and Microsoft Teams, where Analytic Solver also runs): Frontline Systems offers a cloud platform called **RASON**® – an acronym for RESTful Analytic Solver Object Notation. With a few mouse clicks, you can deploy your model for use "on-demand" by any web or mobile application. With RASON you can run your model "live" in Tableau and Power BI, connected to data on your dashboard; with just a little work, you can run your model on-demand from Power Apps and Power Automate. You can even see how your model(s) have been used by other apps and users, all without leaving Excel.

There's much more to RASON – for example, you can use it to connect your model to online databases and cloud data sources, handle sharing under high-security conditions, and even automate high-level, multi-stage "decision flows". To learn more about this, just visit solver.com/rason or pay a visit directly to rason.com. You can even try out model deployment to RASON during your Analytic Solver free trial!

Conclusion: Where Do I Go from Here?

We hope this eBook has answered some of your questions, and cleared up some of the "mystery" (or common misunderstandings) about Solver in Excel, and how it works. We know the "Advanced" sections were a little challenging, but if you studied them, you've come away with some new skills for building Solver models – indeed, optimization models that work with any software – that solve faster and more reliably. And we hope we've opened your eyes to some of the many possibilities for making ***better business decisions*** with Solver, and how to "think about" problems that are candidates for Solver and optimization.

If your problem exceeds the size limits or capabilities of the Excel Solver (or it soon will), an obvious "next step" is to start a free trial of Frontline Systems' **Analytic Solver** upgrade. To do this, just visit Solver.com – there's a **Free Trial** button at the top right of every page.

But we really suggest that you ***talk with us*** – have a no-obligation conversation or Teams / Zoom meeting with one of our professionals. (Indeed, that's the first step when you request a Free Trial.) We'll seek to understand your objectives, and we'll be candid about whether our software is a good fit for you. We've been doing this for 30 years, and the chances are very good that we can save you time and help you reach your objectives faster, with fewer missteps. We look forward to working with you.