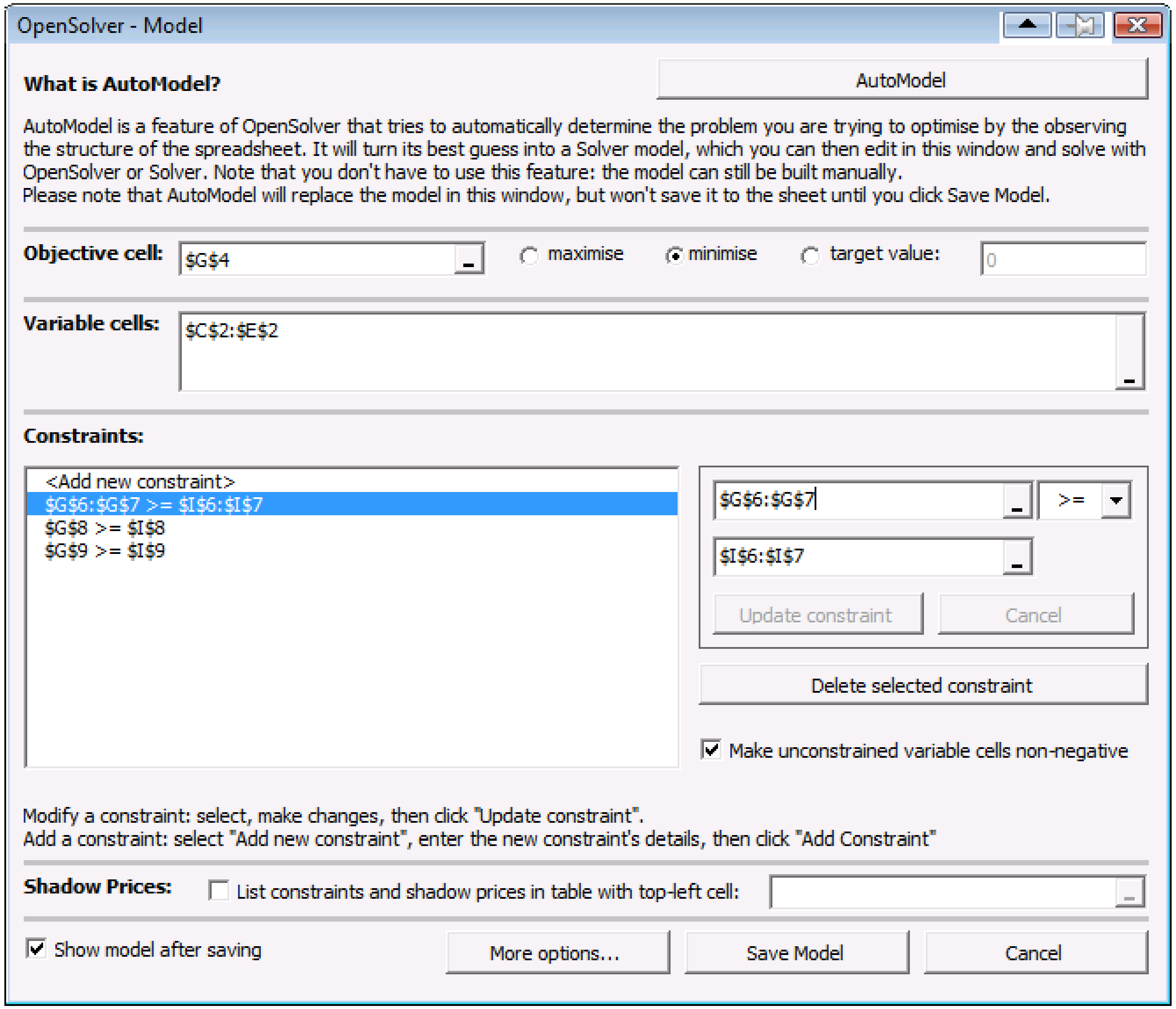
Using OpenSolver

You should start by downloading OpenSolver (see Download and Install), and extract the files from the .zip file to create your OpenSolver folder containing all the OpenSolver files. (Please do not place the OpenSolver folder on your desktop, as this seems to cause problems for some users in Windows 7. Instead, place it in your Documents or Program Files folder.) Then double click on the OpenSolver.xlam file. This will open Excel and load OpenSolver. After clicking on the security warning dialog, OpenSolver will appear in the Data tab. You are now ready to build a model and solve it.

OpenSolver works with your existing Solver models, so you can still use Solver to build your models. If you prefer, you can use OpenSolver's own model editor, accessed using OpenSolver's Model button, to build your model. We hope you find it to be more intuitive, and that you enjoy the automatic highlighting of constraints directly on the sheet.



OpenSolver - Model

What is AutoModel?

AutoModel

AutoModel is a feature of OpenSolver that tries to automatically determine the problem you are trying to optimise by the observing the structure of the spreadsheet. It will turn its best guess into a Solver model, which you can then edit in this window and solve with OpenSolver or Solver. Note that you don't have to use this feature: the model can still be built manually.
Please note that AutoModel will replace the model in this window, but won't save it to the sheet until you click Save Model.

Objective cell: $G$4 ○ maximise [✓ minimise] ○ target value: 0

Variable cells: $C$2:$E$2

Constraints:

<Add new constraint>
$G$6:$G$7 >= $I$6:$I$7
$G$8 >= $I$8
$G$9 >= $I$9

$G$6:$G$7 >= $I$6:$I$7

Update constraint | Cancel

Delete selected constraint

[✓] Make unconstrained variable cells non-negative

Modify a constraint: select, make changes, then click "Update constraint".
Add a constraint: select "Add new constraint", enter the new constraint's details, then click "Add Constraint"

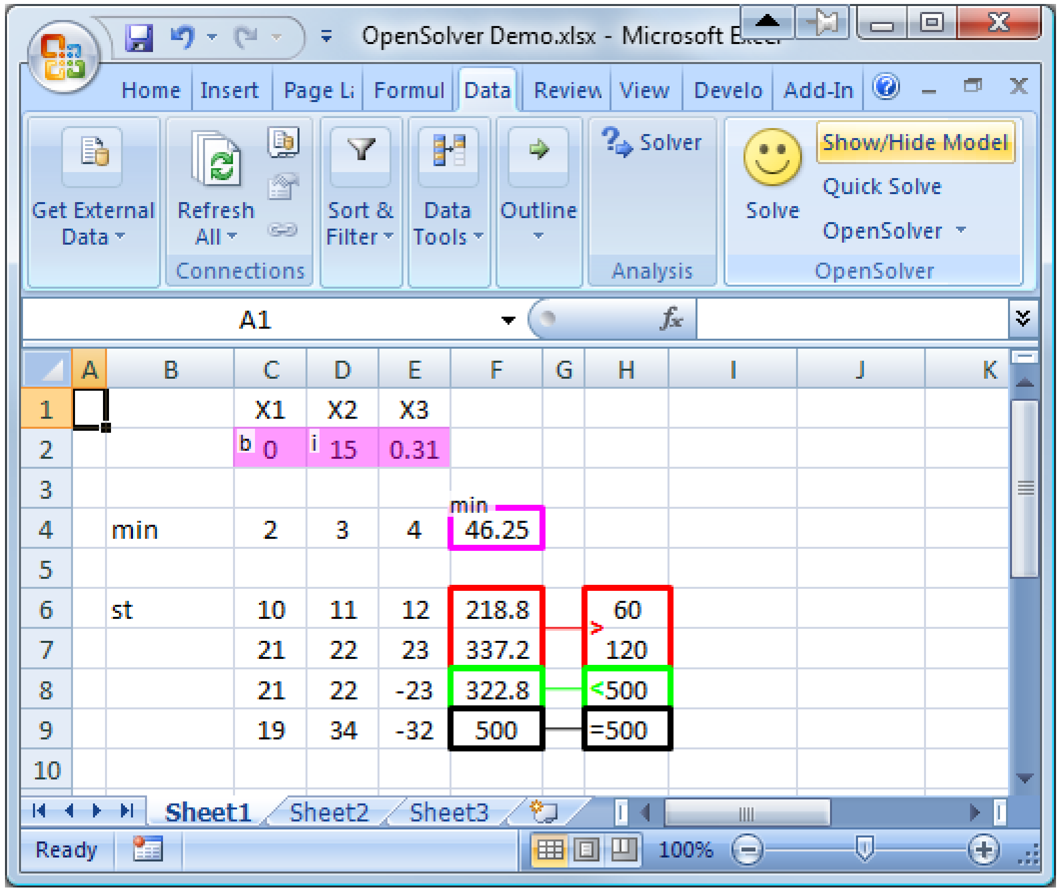
Shadow Prices: ☐ List constraints and shadow prices in table with top-left cell:

[✓] Show model after saving

More options... | Save Model | Cancel

The OpenSolver Model dialog highlights your constraints on the sheet, making them easier to check.

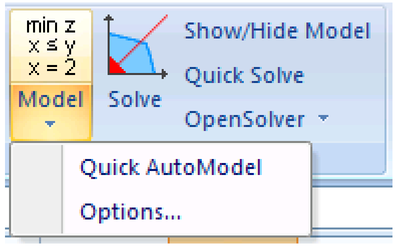
Once you have built your model, you can check it using OpenSolver's "Show/Hide Model" button. This produces a display such as that shown above where the left hand side and right hand side of each constraint are boxed and joined, and the constraint sense indicated (being either '<' for '<=', '>' for '>=', or '=', where each constraint is read left-to-right and/or down the page). The objective (goal) cell is highlighted and tagged as either min (minimise) or max (maximise). The adjustable cells (decision variables) are shaded, with any integer variables being tagged with an ‘i’ and and binary variables with a ‘b’.



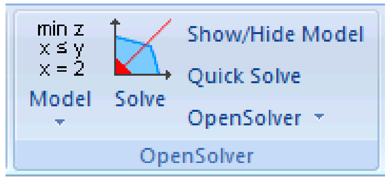
	A	B	C	D	E	F	G	H	I	J	K
1			X1	X2	X3						
2			b 0	i 15	0.31						
3											
4		min	2	3	4	min 46.25					
5											
6		st	10	11	12	218.8	>	60			
7			21	22	23	337.2		120			
8			21	22	-23	322.8		<500			
9			19	34	-32	500		=500			
10											

OpenSolver allows you to eaily view your model’s adjustable cells, objective cell, and the constraints. Integer and binary decision cells are indicated.

OpenSolver comes with a range of “solvers” designed for different types of optimisation problems; these are listed on our guide to solvers. To solve this linear model, choose a linear Solver such as CBC (which is the default solver), or Gurobi (if you have this commercial software installed), using OpenSovler's Options button accessible from the OpenSolver Model menu. (You can also access this from the Model dialog.)



If you turn on "Show optimisation progress while solving" (which somewhat mimics the "Show Iteration Results" found in other solvers), OpenSolver will display the CBC output during the solution process (which is typically very fast).



Click OpenSolver's Solve button to solve the problem. OpenSolver then analyses your spreadsheet to extract the optimization model, which is then written to a file and passed to the CBC optimization engine to solve. The result is then read in, and automatically loaded back into your spreadsheet. A dialog is shown only if errors occur.

Note that OpenSolver does some checks that your model is a linear one, but it is up to you to ensure this is the case by avoiding functions such as sqrt(), if(), abs(), max(), min(), etc.

YouTube Videos of OpenSolver

You may wish to view this YouTube OpenSolver introduction by Kevin Jia (Engineering Science department, University of Auckland)

or this YouTube movie showing the installation and use of OpenSolver:

Learn about Linear and Integer Programming

OpenSolver lets you solve linear and integer programming models. To find out more about these optimization models, you may wish to look at the online notes at Linear Programming – Foundations and Extensions. (This page did include a textbook by Robert Vanderbei, but this is no longer available. However, the lecture notes are still there.)

Quick Solves

An advanced feature of OpenSolver is quick solving using 'parameters' that the user can change between solves. Building a large model is often much slower than solving the model. This can be very frustrating as users often need to solve the same model repeatedly after making changes to the constraint right hand sides. To handle this, OpenSolver allows you to define 'parameter' cells (using the "Set Quick Solve Parameters" menu) that are the cells you will be changing between successive solves. (These may be the actual right hand side cells, or other cells that determine the right hand side values.) You then choose the "Initialise Quick Solve", at which point OpenSolver analyses your spreadsheet (which is has to do just once) to build the model and determine how your constraint right hand sides change as each parameter cell changes. OpenSolver assumes that this change is a linear one; it is up to you to make sure this is the case. You can then change your parameters, and click the Quick Solve button to very quickly solve the modified model. (See below for using Quick Solve from VBA.)

Solving the relaxation of an IP

OpenSolver provides a menu item to solve the relaxation of an integer program. This is often useful to understand how hard the problem will be to solve to optimality.

Extra Solver Parameters

OpenSolver allows the user to pass extra parameters to the solver by creating a named table on their spreadsheet. The table should have two columns, with a row for each solver option the user wants to set. The left column of each row should contain the name of the option (without any '-') and the associated right column should specify its value. You can then tell the solver to use these options by specifying this table as the `Extra Solver Parameters` range in the Options dialog.

See the reference guide for each solver to learn which parameters are supported:

- CBC
- Gurobi
- Bonmin
- Couenne (section 6)
- NOMAD (sections 4.2 and 6)

Advanced CBC Control

CBC LOGGING

If you are solving a large problem, then a useful extra parameters range for the CBC solver might look like the following. This turns on simplex and branch-and-bound logging; it will give you lots of output. Make sure you have used *Model... Options... Show optimisation progress while solving* to turn on display of CBC output.

slogLevel	2

logLevel	2

USING CBC INTERACTIVELY

If you want to explore their problem directly with CBC, you can open a CBC command line window with the model loaded ready to be solved. Note that this does not allow solutions to be loaded back into Excel, but does allow different CBC options to be explored for use in subsequent solves by OpenSolver. This is useful if the problem takes a long time to build. To do this, select the *OpenSolver… Open Last Model in CBC* menu item.

If you wish to take control of CBC during an Excel run, then use the following as part of the extra parameter range:

slogLevel	2
logLevel	2
–	

The last parameter `-` with no value will cause CBC to enter interactive mode. Make sure you have used *Model… Options… Show optimisation progress while solving* to turn on display of CBC output. Then, after clicking OpenSolver's Solver button, you will be at the CBC command prompt. Type: 'stat' to get a summary of your problem details, 'solve' to solve the problem, then 'solution modelsolution.txt' to write a solution, and 'quit' to return to Excel. OpenSolver will then load in the solution. This can be useful for debugging.

OpenSolver and Visual Basic for Applications (VBA)

OpenSolver can be called from your own VBA routines. To do this, in your VBA project you either need to add a reference to "OpenSolver" (or, as detailed below, use "Application.Run.) This site shows how to use the Tools menu to add OpenSolver as a Reference. Make sure you have opened OpenSolver before trying this. Then, you can take full control of OpenSolver using VBA with the functions listed here. OpenSolver is written in VBA, and so you can have a look at the code to see how it all works.

For example, you can run OpenSolver as follows:

```
Dim Result as OpenSolverResult
Result = RunOpenSolver(False, True) ' do not relax IP, do hide dialogs
```

Result will be one of the following: OpenSolverResult.ErrorOccurred, OpenSolverResult.Optimal, OpenSolverResult.Unbounded, OpenSolverResult.Infeasible, OpenSolverResult.TimeLimitedSubOptimal

Model Setup with VBA: We recommend that you set up the model on the spreadsheet in the normal way, and then in your VBA make any (presumably small) changes you want to the resulting spreadsheet (such as changing values on the spreadsheet that specify a constraint's right hand side), and then call RunOpenSolver. If you really want to set up the whole model in VBA, then you should use the OpenSolver API functions to modify the model.

Note: As of OpenSolver 2.7.0 we **strongly** recommend you use the OpenSolver API instead of the Solver API detailed below. If you want to use the Solver API, after adding a reference to Solver, you can also use of the standard Solver VBA commands which are documented here. You can set up the model using these Solver commands, and then solve the resulting model using RunOpenSolver. If using the Solver API, we strongly suggest all references in your constraints are absolute (not relative) to avoid unpredictable behaviour (Thanks to Lukas for this feedback). This is not a problem with the OpenSolver API.

QuickSolve with VBA: To use Quick Solve from VBA, your code should look something like the following

```
 SetQuickSolveParameters QuickSolveParameters ' This is the range you want to change
InitializeQuickSolve
' Set the value of a Parameter cell, which changes the RHS value of some constraint
RunQuickSolve
' You now need to change the value of a Parameter cell, which changes the RHS value of some constraint
RunQuickSolve
```

See also the definitions of these function (in the API reference) for more fine-grained control of Quick Solve.

If there is a lot of demand for this feature, then we will look at making this VBA interface more powerful.

Running OpenSolver using Application.Run: If you include as reference to OpenSolver (as detailed above), then when you spreadsheet runs on another machine, OpenSolver has to be either in the same place on disk or already opened by the user. An alternative is not to add a reference, but run OpenSolver using Application.Run, as shown in the following code. This can be a good approach for sharing OpenSolver-based VBA solutions.

```
Sub DoRunOpenSolver()
    ' Run OpenSolver assuming it has been opened (but without requiring a reference to OpenSolver). Ask the
user to open OpenSolver if it is not currently open.
    On Error GoTo errHandler_NoOpenSolver
    Application.Run "OpenSolver.xlam!RunOpenSolver"
    On Error GoTo errHandler
    Exit Sub
errHandler:
    Err.Raise Err.Number, Err.Source, Err.Description, Err.HelpFile, Err.HelpContext
errHandler_NoOpenSolver:
    Err.Clear
    MsgBox "This workbook requires OpenSolver, a free Excel addin available at http://opensolver.org" + vbCrLf
+ vbCrLf + "Please install & then open OpenSolver, and then try again.", vbOKOnly, "OpenSolver"
End Sub
```

If you need the result code (one of these OpenSolverResult values), then use:

```
Sub DoRunOpenSolver()
    ' Run OpenSolver assuming it has been opened (but without requiring a reference to OpenSolver). Ask the
user to open OpenSolver if it is not currently open.
    Dim result as Long
    On Error GoTo errHandler_NoOpenSolver
    result = Application.Run("OpenSolver.xlam!RunOpenSolver")
    On Error GoTo errHandler
    ' if result <> 0 then ... ' Something went wrong; not optimal
    ' if result = 5 then ... ' infeasible
    Exit Sub
errHandler:
    Err.Raise Err.Number, Err.Source, Err.Description, Err.HelpFile, Err.HelpContext
errHandler_NoOpenSolver:
    Err.Clear
    MsgBox "This workbook requires OpenSolver, a free Excel addin available at http://opensolver.org" + vbCrLf
+ vbCrLf + "Please install & then open OpenSolver, and then try again.", vbOKOnly, "OpenSolver"
End Sub
```

The Excel Solver is a product developed by Frontline Systems for Microsoft. OpenSolver has no affiliation with, nor is recommend by, Microsoft or Frontline Systems. All trademark terms are the property of their respective owners.

1 thought on “Using OpenSolver”

Pingback: OpenSolver 2.6.1 (15 Feb 2015) « OpenSolver for Excel

Comments are closed.