Modeling Best Practices,

Especially for Excel/What's*Best*!

www.lindo.com

Keywords: Best practices, Documentation, Modeling.

Summary of Things to Do

Document your model. Give an overview or guide to what is being attempted.

Develop incrementally and start small.

KISS : Keep the Spread Sheet Simple. Avoid exotic features of Excel.

Establish a pattern or style, document it, and stick to it.

Isolate the optimization of the spreadsheet to one tab/sheet.

Solve the general case. If you can add generality at little cost, do it.

Separate data from the model equations. Do not hard code parameters such as cost of capital, etc.

Debugging: Don't. Do it correctly the first time.

Formulae Should be Computable for All Possible Values

Avoid numerical failures such as divide by zero, square root of negative number.

Use soft constraints, so always a feasible solution regardless of input.

Keep it linear. Nonlinear, non-convex, non-smooth functions increase solve time dramatically.

Details:

Document it. If you do not document your model, do not expect anyone to help you when the model is not giving you the results you expect. Give an overview or guide to what is being attempted.

What is the overall problem you are attempting to solve?
What are the inputs?
What are the decision variables?
What are the constraints?
What is the objective?
What are the outputs?
What is each section of formulae supposed to do?

Models, at least early in life, contain bugs/misrepresentations of what was intended. If there is no documentation of what the modeler intended, there is no way that a second party can deduce why the model code does not do what the model author intended.

Where should the documentation be stored?
It should be stored in the model or program itself, so it always accompanies the model. Documentation that is conveyed verbally is highly likely to be misunderstood or lost. Documentation stored in a separate file from the code is likely to get lost.

When should documentation be written?
It should be written first, before any code is written. First write the documentation to give structure to the model or program. Then fill in the code to achieve what has been outlined.

Develop incrementally and start small.

The reason for implementing incrementally is that if you get an error result, it is probably due to an error in the most recently made change to the model or program. If you first add a lot of features to a huge model before doing any testing, and then encounter an error when testing, you will be unsure of where to look for the cause of the error.

KISS : Keep the Spread Sheet Simple.
Avoid exotic features of Excel. There are many wonderful and exotic features in Excel. There is a temptation to use them. There are two reasons for avoiding exotic features of a language. a) Other people, who understand only a (different) subset of the exotic features of a language, will have a difficult time understanding your implementation and maintaining your model. b) different processors, e.g., What's*Best*!, may not support all the exotic features of a rich language such as Excel.

Style
Deciding upon a style, sticking to it, and documenting it, makes your model easier to understand. Following a style and documenting it is sometimes more important than which specific style is followed. Some examples of styles:

+ Use a pastel yellow background for input data cells in Excel.

+ In a multiple dimensional model, say with products, locations, and time periods, choose an ordering and stick to it, rather than group constraints, say, in order (product, location, period) for one type of constraint and in order (location, period, product) somewhere else for a different constraint.

+ In a spreadsheet, choose an orientation (horizontal vs. vertical) for each dimension (say for product, location, period, scenario) and stick to that orientation. Do not store products vertically one place and horizontally somewhere else. It is harder to add a new product if the orientation of products is mixed. In a spreadsheet, a good style has the feature that it is easy to add more of some object type (e.g., another product, location, period, or scenario) by doing a single insertion, rather than several insertions, horizontal in one place and vertical in another place.

+"Hungarian notation," originated by Charles Simonyi at Xerox Parc and Microsoft suggested using the first character(s) of a name to designate the type of object, e.g., aX for an array, pX, for a pointer, sX for a string, paX for a pointer to an array, etc.

Isolate the optimization of the spreadsheet to one tab/sheet.
A spreadsheet that uses optimization has three types of cells: a) "Fixed" cells that do not depend upon any Adjustable cells. b) Optimizable cells that depend upon an Adjustable cell and affect either the objective function or some constraints, and c) "Reporting cells" that depend upon Adjustable cells but do not affect any objective or constraint cell. You can use any exotic Excel function you wish for cells of

type (a) or (c). For cells of type (b) you want to restrict yourself to using only functions that What's*Best*! understands. For cells of type (c), you want to tell What's*Best*! that it can disregard them by declaring them as WBOMIT cells, (click on: What'sBest! | Advanced | Omit).
If one is debugging the optimization part of a spreadsheet it is convenient, although not necessary, if cells of type (b) are isolated to one tab of the spreadsheet. The most important and useful functions that What'sBest! understands are: SUM( ), SUMPRODUCT( ), SUMIF( ), and most of the simple math functions such as +, -, /, *, SIN( ), etc.

Splitting Complex Formulae If What's*Best*! encounters a cell containing an Excel function that What's*Best*! does not understand, then What's*Best*! treats the cell as a constant equal to its current value, and displays a warning message in the WB Status tab. Suppose a cell contains the formula:

R1=ACCRINTM(IssueDate,SettleDate,Yrate,ParVal)*X1.

Because ACCRINTM is not a function that What's*Best*! supports, What's*Best*! will treat this cell as a constant. If X1 is an optimizable (type b) cell, this is not a good thing to do. If, however, IssueDate, SettleDate, Yrate, and ParVal are fixed input values (type a), then there is an easy work-around. Introduce an additional cell R2 and write:

R2=ACCRINTM(IssueDate,SettleDate,Yrate,ParVal)
R1=R2*X1

R2 will be treated as a fixed cell, but that is OK. The formula for R1 is now recognized by What's*Best*!, so R1 will now be treated as an optimizable cell, as we want it to be. A side benefit of this approach is that a long complex expression may be more understandable if split into components that each have a meaning.

Some users like to use INDEX() and MATCH() to do lookups. What's*Best*! does not support INDEX() if it appears within a nontrivial formula. INDEX() is supported if it appears alone in a cell, e.g. =INDEX(C4:C9,H2).

Keep it linear.

Nonlinear, non-convex, non-smooth functions increase solve time dramatically. What'sBest! will report on the What's*Best*! Status tab which cells contain nonlinear functions. For example, the nonlinear constraint, x1/x2 >= 10 is equivalent to the linear constraint x1 >= 10 * x2 (if x2 is > 0). If you are not familiar with methods for avoiding nonlinearities, contact LINDO Systems.

To illustrate the usefulness of documentation, as well as linearization, we recently encountered a spreadsheet containing:

```
X2 = X3*K1;              ! K1 is a nonzero constant
X1 = IF( X2 = 0, 0, K2); ! Where K2 is a constant. This is a nonlinear expression;
X3 = X1;                 ! This is a constraint;
```

Is it obvious what is the effect of the above 3 equations? They would be much easier to understand if they were documented with the simple statement: "Force X3 to be either 0 or K2." Once this is understood, the obvious (linear) replacement of the above three constraints is:

```
X2 = X3*K1;  ! K1 is a nonzero constant
X3 = K2*B1;  ! Where K2 is a constant and B1 is a binary (0 or 1) variable;
```

Full truck/Container constraints. Suppose a truck or a container can carry 20,000 kg. If you want to send 25,000 kg to a customer, you must send two trucks. You cannot send fractional trucks. If X1 is the number of kg sent and T1 is the number of trucks needed, you might be tempted to write

T1=ROUNDUP(X1/20000,0).

This is a nonlinear expression. If you are using optimization It is better to write the linear constraints:

K1=20000 (Define truck capacity once and for all).

T1 >= X1/K1 (and declare T1 to be a general integer variable).

This change may also make more sense if this little code fragment is part of larger routing model. It may be that even though two trucks will be sufficient, we may want to move more than two trucks because there are plenty of loads to be picked up at the destination. The formula T1=ROUNDUP(X1/20000,0) does not allow this. The constraint T1 >= X1/K1 does.

Separate data from the model equations.

Do not hard code parameters such as cost of capital, etc. Eventually, someone will ask, "What happens if we change our cost of capital?". You do not want to face the task of searching the code for all the places where 0.12 has been hard coded into formulae. You want to change only a single defined parameter or cell in a spreadsheet.

Formulae Should be Computable for All Possible Values in Optimization Models

1) IF statements are very popular with Excel users. IF statements are a source of a lot of trouble in optimization models. Consider the little model:

```
FreeGift=IF(AmtBuy>0,1,0)
Profit=FreeGift-AmtBuy
```

This is a perfectly acceptable pair of expressions for a simple what-if spreadsheet. Now suppose you want to optimize, with AmtBuy being an Adjustable cell and maximizing Profit. What should be the value for AmtBuy?

The advice here is to avoid using strict inequalities in optimization models. A well-defined and computable model (and probably makes more sense to the vendor who is offering free gifts) is

```
FreeGift=IF(AmtBuy>=5,1,0)
Profit=FreeGift-AmtBuy
```

i.e., you have purchase of 5 or more to qualify for a free gift.

2) Ratios are very popular with Excel users. Ratios such as FractionStock=Stocks/(Stocks+Bonds) are fine as long as the portfolio contains a strictly positive amount of either Stocks or Bonds. What if the investor decides the optimal strategy is to put all her money in Gold? You will get a divide by zero when trying to compute FractionStock. Many users would be happy if FractionStock is defined to be 0 in this case. A simple device for achieving this for most practical purposes is to write something like:

```
TolDivZ = 0.0000001
FractionStock=Stocks/(Stocks+Bonds+TolDivZ)
```