# Building Mathematical Models in Excel

# Building Mathematical Models in Excel

## A Guide for Agriculturists

Christopher Teh Boon Sung

Universal-Publishers
Boca Raton

*Building Mathematical Models in Excel: A Guide for Agriculturists*



Universal-Publishers
Boca Raton, Florida • USA
2015

ISBN-10: 1-62734-038-6
ISBN-13: 978-1-62734-038-0

www.universal-publishers.com

Cover image © Lexmomot | Dreamstime.com

# Contents

Dedicated to my son, Zachary

# Preface

A model is a simplified representation of a real system. A model can be a picture, illustration, verbal or text description, or even a physical replica (mockup) of the real system.

However, one type of model that is of particular interest to scientists is the mathematical model because it represents a real system using mathematical principles. The properties and behavior of the real system are represented as one or, more typically, a series of equations which can be used to further understand the real system or to make predictions about the real system in certain scenarios or conditions.

Developing a mathematical model requires three steps. The first step is to uncover and understand the science behind the real system. The second is to translate all the essential properties and behavior of the real system into a meaningful and representative mathematical form. And the third step is to implement this mathematical form into a computer program that can be executed by computers.

So, unless we are dealing with very simple equations, computers are an essential tool in mathematical modeling. A mathematical model often consists of many equations, some of which too complex, tedious, or complicated to be reliably calculated by hand. Moreover, a mathematical model often involves iterative calculations where the same set of equations are repeatedly used in calculations.

This means mathematical modeling requires agriculturists to not only be knowledgeable in agriculture science but also to be proficient in mathematics and in computer programming.

Unfortunately, most agriculturists receive no to very little formal training in computer programming.

Consequently, they often struggle with this last step of mathematical modeling; that is, to translate their models into computer program that can be correctly understood and executed by a computer. For them, computer programming often becomes a tedious and time-consuming drudgery that distracts them from their main purpose of study or work.

Consequently, this book is targeted at agriculturists particularly those who are either novices or non-programmers and who wish to have an easier way to implement their models in computers. This book shows how model implementation can be easily carried out in a spreadsheet, with the spreadsheet of choice being Microsoft Excel.

But why spreadsheets, and why Excel? This is because spreadsheets provide a modeling platform that requires the least proficiency in computer programming. Unlike other modeling platforms that enforce a rigid programming structure, spreadsheets' unrestricted and open structure enable novices and non-programmers to easily implement their models in a spreadsheet and to have the spreadsheet run the model simulations. One spreadsheet program in particular is Excel. It is popular and the most widely used spreadsheet in the world – for several reasons. Excel is easy to use, versatile, and powerful, and it has a very gentle learning curve. Even first-time Excel users can quickly learn how to enter their data in Excel and have Excel perform calculations on them using formulas and functions.

Nonetheless, Excel does have several key limitations that prevent the implementation of large, complex models. To circumvent these limitations, Excel's programming language, VBA (Visual Basic for Applications), could of

course be used. However, using VBA requires programming skills in which most agriculturists lack. Moreover, requiring agriculturists to learn VBA would defeat the purpose of having an alternative and easier way for agriculturists to implement their models.

To overcome Excel's limitations, an Excel add-in, called *BuildIt*, was developed (Teh, 2011). Although BuildIt was developed using VBA, BuildIt shields users from VBA. With BuildIt, users are able to implement simple to complex models in Excel without requiring knowledge in VBA or a strong proficiency in computer programming.

BuildIt removes—or at least, greatly reduces—the distraction of computer programming and allows agriculturists to concentrate on the more important task of building their mathematical models and using them in their studies or work.

This book is a guide for agriculturists to learn how they can implement their mathematical models in Excel with the support from BuildIt. This book uses examples related to agriculture, but there is no reason why Excel or BuildIt cannot be used to implement models from other disciplines.

Examples in this book start by using simple models, but they soon include progressively larger, more complex, and more realistic agriculture models. By the end of the book, agriculturists would have learned how to implement a generic crop growth model that is made up of five components: meteorology, photosynthesis, energy balance, soil water, and crop growth development. Exercises included at the end of chapters are to test understanding and to enable the models to be run in other conditions.

Chapter 1 in this book starts by discussing about the different types of modeling platforms. This chapter focuses

primarily on why using spreadsheets like Excel can be a suitable modeling platform for novices or non-programmers. However, the weaknesses and limitations of spreadsheets as a modeling platform are also discussed. Lastly, good programming practices are listed at the end of this chapter

Part I of this book encompasses Chapters 2 to 5, and these chapters cover what BuildIt is and how it can be used to implement mathematical models in Excel. Several examples of models are used. Simple, rather than large, complex models, are used as examples so that certain key aspects in model implementation can be highlighted without the distraction of excessive details.

Chapters 3 and 4, in particular, discuss about BuildIt *actions*. These actions circumvent the limitations of Excel that would normally hinder the implementation of large, complex models. These actions, for instance, allow a spreadsheet cell to manipulate or change the content of another cell, or for a cell to update its own value. Both these operations are conventionally not allowed in Excel. In Excel, all external cell are strictly read-only. Cell A1, for instance, can read but cannot alter the content in cell B1. Likewise, cell A1 cannot update its own value, say, by having the following formula "=A1+1", where cell A1 refers to its current value, increments it by 1, then updates itself to have this new value. This self-referencing operation is not allowed in Excel because it would result in a never-ending loop of recalculations in cell A1. BuildIt actions remove such restrictions imposed by Excel and without causing the negative consequences. Typical mathematical operations like integrations and differentiations that are often needed in mathematical models are also supported by the BuildIt actions.

Chapter 5 discusses about scenarios. BuildIt setups a system or structure that allows a single model run to execute two or more scenarios in a sequential manner. BuildIt also provides three custom macros and two functions. Two of these macros deal with freezing and unfreezing the screen updates. By freezing the screen updates, the model run can be speeded up because Excel does not have to always refresh or update the screen whilst the model is running. The last macro deals with deleting the model output from the previous model run. BuildIt supplies two custom functions, `interpolate` and `solve`. The function `interpolate` is used for linear interpolation between two given values, and `solve` is to solve simultaneous linear and nonlinear equations.

Part II of the book encompasses Chapters 6 to 10. These chapters show how a non-trivial generic crop growth model is built sequentially from its five model components, starting from the meteorology component (Chapter 6), photosynthesis component (Chapter 7), energy balance component (Chapter 8), soil water component (Chapter 9), and finally, the crop growth development component (Chapter 10).

The final chapter, Chapter 11, shows how a tool of BuildIt, called *Trace*, can be used to draw a visual map of how spreadsheet cells are related to one another. By examining the cells network map, agriculturists can better understand how variables and formulas are related to one another. This can be particularly useful to detect errors in a model or to better understand how a model works.

At the end, the lessons learned from this book would allow agriculturists, as well as non-agriculturists, to implement their own mathematical models in Excel.

It is important to note that the intention of developing BuildIt was to overcome some of Excel's weaknesses so that simple and complex models can be implemented in Excel. The purpose of BuildIt is not to substitute or compete against other modeling platforms, but to be a useful tool that novices and non-programmers can rely on in their modeling work.

BuildIt was first published in the Journal of Natural Resources & Life Sciences Education (Teh, 2011).

*Christopher Teh Boon Sung*
*Serdang, Malaysia*
*March 3, 2015*

# Chapter 1. Spreadsheets for mathematical modeling

## 1.1 Types of modeling software (platform)

Mathematical modeling often incurs a steep learning curve to many agriculturists. This is partly because one of the stages in the modeling process involves computer programming, a skill in which many agriculturists lack. Consequently, many agriculturists struggle to translate their formulated models into a set of instructions that can be implemented by a computer system. For them, computer programming often degenerates into a tedious and distracting undertaking.

This problem has been recognized even as early as four decades ago when Hillel (1977) commented that agriculturists, being novice programmers, are easily entangled in the complexities of translating their model into a computer source code. Rather than focusing on the important—and intended—task of mathematical modeling and simulation, they instead become distracted by the intricacies and mechanism of computer programming.

What is required then is some special software that allows non-programmers to build their own computer models. This will enable non-programmers, agriculturists included, to accomplish their modeling tasks directly and quickly and without the distraction of having to learn a new computer language (or to master a new simulation application).

There are generally four categories of software platforms to aid in model building and simulation. They are: 1) general purpose computer languages (*e.g.*, C, C++, FORTRAN, BASIC, Pascal and Java); 2) specialized

simulation applications (*e.g.*, FST, PowerSim, Stella and ExtendSim); 3) equation solver-based applications (*e.g.*, Maple, Mathematica, and Mathlab); and 4) spreadsheet-based applications (*e.g.*, Microsoft Excel and OpenOffice Calc)[1]. Of these four groups of software platforms, spreadsheet applications require the least level of proficiency in computer programming.

It is no surprise then that spreadsheets have been advocated by several workers (*e.g.*, Seila, 2005; Brown, 1999; Nardi and Miller, 1990) as a suitable platform, especially for non-programmers, to build and simulate their models.

Spreadsheets are immensely popular and widely used because they are easy to use and versatile. Additionally, they provide the following features (adapted from Seila, 2005): 1) a large number of numerical and non-numerical functions (*i.e.*, dedicated formulas) to do mathematical, statistical, database, date and time, financial, engineering, and other types of calculations; 2) database representations and access; 3) charting and graphing; 4) display and document formatting capabilities such as layout, fonts, and colors to improve presentation; and 5) scripting or

[1] FST (Fortran Simulation Translator) is available upon request from its developers (*see* van Kraalingen *et al.*, 2003);
Powersim® is a registered trademark of Powersim Software AS;
Stella® is a registered trademark of isee systems;
ExtendSim® is a registered trademark of Imagine That Inc.;
Maple® is a registered trademark of Waterloo Maple Inc.;
Mathematica® is a registered trademark of Wolfram Research Inc.;
Mathlab® is a registered trademark of The MathWorks Inc.;
Microsoft® Excel® are registered trademarks of Microsoft Corporation;
OpenOffice® is a registered trademark of The Apache Software Foundation.

programming language such as VBA (Visual Basic for Applications) in Excel and OpenOffice Basic in OpenOffice Calc.

Spreadsheets were initially conceived as electronic accounting books for financial analysis, but today, spreadsheets have progressed beyond their original intent. They have become powerful tools to manipulate, analyze, and present data, and to build models for simulating various phenomena in science and engineering (Khandan, 2001).

## 1.2 Advantages and disadvantages of spreadsheets

Spreadsheets are well known that they are easy to use, versatile, and powerful, but it is not so much these characteristics that qualify spreadsheets as a suitable modeling platform for non-programmers.

The first key benefit of using spreadsheets in modeling is that they do not require users to be proficient in computer programming. This is achieved by the spreadsheet system providing: 1) automatic control and maintenance of the program flow, 2) a simple, straightforward modeling framework, and 3) high-level and task-specific functions (Nardi and Miller, 1990).

A spreadsheet is a group of pages, or worksheets, where each worksheet is a two-dimensional table consisting of rows and columns of cells. Each cell can contain either a constant or a calculated value, derived from a function or formula. The open tabular format of the spreadsheet provides users a simple, straightforward framework in which to build their models.

As such, spreadsheet users only need to understand two basic concepts: to treat the cells as variables and the functions (or formulas) as the relationship between these

variables. Users specify the way variables depend on each other via formulas or functions, and the spreadsheet system maintains these variable dependencies. So, if one part of the spreadsheet changes, it triggers an update to the whole spreadsheet so that all the dependent variables are automatically recalculated to reflect their new values—giving users immediate feedback. The order in which the variables are calculated is worked out by the spreadsheet system based on the variable dependencies. Users cannot contravene this calculation sequence by giving the spreadsheet system a different calculation or action sequence to be performed. Though this might seem restrictive, it is actually this feature, among others, that makes spreadsheets appealing to non-programmers.

Spreadsheets not only relieve users from having to maintain the program flow themselves, but also from having to design their own modeling framework. Flow control and framework design are both difficult programming concepts for non-programmers to master (Nardi and Miller, 1990; Hoc, 1989; Lewis and Olson, 1987; Soloway et al., 1983). Without automatic flow control, users will have to write their own programming code to track variable dependencies and to update all effected variables iteratively when required. And users need not design their own modeling framework because the spreadsheet already provides users with one.

Spreadsheets further cater to non-programmers by providing high-level and task-specific functions, meaning that these functions can be used without users having to understand how they work or to require computer programming expertise. One commonly used function in Excel, for example, is the `SUM` function that, as its names implies, calculates the total of a set of given values.

Spreadsheet users merely write the `SUM` function and specify all the cells that contain the values to be added. Without such a high-level function, users will require to write a loop to iterate through the elements in an array, summing each element and accumulating the total. Other routinely used functions are such as `AVERAGE` (for calculating the mean of values), `MIN` and `MAX` (for determining the smallest and largest value, respectively, from a given set of values), `VLOOKUP` and `HLOOKUP` (for looking up a value from a vertically and horizontally tabulated data, respectively), `IF` (for conditional computations), and trigonometric functions like `SIN` and `COS`. These are only a few of the functions listed; hundreds more functions are available in Excel (as well as in other spreadsheets) to support users' requirements.

The second key benefit of using spreadsheets is its table-oriented interface. It provides a strong visual format to manipulate, organize and present data as well as a problem-solving tool (Nardi and Miller, 1990). The spreadsheet's open tabular format allows direct access to the variables' values. This is in contrast to traditional programming efforts which require some special and intentionally constructed mechanisms for the input and output of variables. Brown (1999) additionally noted that because the values of all variables are always updated and displayed in the table, this could help to detect the location of errors during model construction by the presence of nonsensical or questionable values.

The spreadsheet's tabular format additionally allows users to organize visually their model into subparts or modules, where each module performs a specific task to solve an overall, larger problem. This organization can be done by segmenting the large model into smaller modules

by, for instance, leaving an empty row between each module or by placing each module in a separate worksheet. Formatting the module sections such as by applying different fonts, shading, style layout or colors further help in code organization. Consequently, the model's code is distributed visually over one or more grids in such a way that makes the model more comprehensible and maintainable. Visual code organization is not merely for aesthetic reasons. The very act of laying out data, organizing and formatting the code provide an important visual feedback mechanism that helps to organize users' thoughts and shape their problem solving process (Nardi and Miller, 1990).

Spreadsheets, however, have several important limitations. Spreadsheet models can be difficult to understand when studied by others. As mentioned previously, the modeling framework provided by spreadsheets is a simple, straightforward structure, essentially consisting of variables (cells) and their linkages with each other (formulas). But as models increase in complexity, these variable linkages can grow into an intricate network of relationships. Since the spreadsheet system maintains the program flow and shields users from viewing this network of relationships, it can be difficult for other users (even for the model developers themselves) to grasp the computations as a whole. This causes difficulty not only in understanding the model but also in debugging it (that is, to locate and correct the errors in the model).

Spreadsheets offer to non-programmers a modeling platform that is almost unrestrictive and freeform and with little validation checks. While this feature can be convenient for those who wish to develop their models easily and quickly, it does, however, make it easy for users

to introduce errors unwittingly into their models. Errors in large spreadsheet models, in particular, can be difficult to locate and, at times, be undetected.

In the business sector, the frequency of errors in their spreadsheets are reported to be alarmingly high. Rajalingham et al. (2001) reported that as much as 90% of real-world spreadsheets contained errors. Furthermore, the European Spreadsheet Risks Interest Group (www.eusprig.org) lists nearly a hundred examples of newsworthy incidences where spreadsheets errors caused companies either financial losses, incorrect forecasting, or simply, embarrassment. Although spreadsheets from the business sector are often cited as examples of high occurrence of errors, we cannot disregard the possibility that spreadsheets from other disciplines (such as agriculture) are plagued too by high number of errors.

The simple modeling framework provided by the spreadsheet also makes spreadsheet models difficult to *reuse* and *extend*. Reusability and extendibility are two software engineering concepts related to ways to increase the usefulness, applicability and lifespan of software. A modeling framework that adheres to these two concepts means that, ideally, the framework is flexible and adaptable enough for algorithm modification, substitution and addition. In some ways, reusability and extendibility are analogous to the "plug-and-play" feature in modern computer systems. For instance, adding, replacing or removing a computer peripheral (such as a printer, scanner or mouse) should be a seamless operation that does not disrupt a computer system or cause it to malfunction. In a similar way, a reusable and extendible model means that one or more parts in the model can be modified easily or even substituted with other parts from another model. It

also means that additional features or functions can be added to the model. All these operations can occur without having to break or redesign the existing modeling framework.

There are several examples in agriculture of modeling frameworks developed with concerns of achieving reusability and extendibility (*e.g.*, Papajorgji and Pardalos, 2006; Papajorgji et al., 2004; Hillyer et al., 2003; Caldwell and Fernandez, 1998; Acock and Reddy, 1997).

Nevertheless, until today, none of them has yet to be adopted widely by agriculturists. One could speculate that these framework designs are still not sufficiently adaptable and flexible enough to cope with the unpredictable and transient requirements of agricultural models. Or it may simply be that agriculturists do not have the interest or time to learn about reusability and extendibility, both concepts of software engineering, not agriculture. Consequently, for many agriculturists, the problem that spreadsheet models are difficult to reuse and extend may actually be an unimportant concern in their work.

Spreadsheets have other limitations that are particularly relevant for models with complex data structures and algorithms (Seila, 2005).

A spreadsheet is a two-dimensional table of cells, and while this format is adequate in most cases, it is a limitation when more elaborate data structures are required such as lists and trees. Although spreadsheets can be coerced to handle such complex data structures, the way they are achieved in spreadsheets can be inefficient and convoluted. Complex algorithms can be particularly difficult to implement in a spreadsheet.

Spreadsheets lack explicit loop and conditional controls commonly found in general purpose languages.

Spreadsheets actually have a conditional construct IF-THEN-ELSE, but its effect is only local to the cell that contains the construct. The construct cannot be used to change the values in other cells or to transfer control to another cell or to another part of the spreadsheet. As stated earlier, spreadsheets relieve users from having to maintain the control flow themselves. While this is a convenient feature for non-programmers, it becomes a severe limitation when an algorithm requires users to specify explicitly the sequence of actions to be performed.

Spreadsheets like Excel and OpenOffice.org Calc have their own scripting language that can be used to circumvent this problem, but they require proficiency in computer programming. But non-programmers may find this solution unappealing because their use of spreadsheets is precisely to avoid computer programming in the first place.

Lastly, spreadsheet models often run slower than models developed in other software platforms. This is partly because models developed in other platforms, such as in C, C++, or FORTRAN, are compiled (translated) into fast machine code that is in a format directly executable by the computer.

In contrast, no such compilation occurs for spreadsheet models. The formulas in spreadsheets are instead interpreted first before execution. This slows down program execution. Dalton (2005) roughly estimated that models written in Excel could be slower by as much as five to twenty times than if they were written in C and C++ computer languages. He further reported that Excel's text-manipulation routines are very much slower than the same routines in C and C++ by as much as over 300 times. Slow execution speeds may not be noticeable for simple